Hierarchical Caching and Prefetching for Improving the Performance of Continuous Media Servers

by

Stavros Harizopoulos

Submitted to the Department of Electronic and Computer Engineering

in Partial Fulfillment of the Requirements for the Diploma of Engineering in Electronic and Computer Engineering at the Technical University of Crete

August 11, 1998

Author..
Department of Electronic and Computer Engineering
August 11, 1998

Certified by..
Peter Triantafillou
Thesis Supervisor
Assistant Professor, Department of Electronic and Computer Engineering

....
Stavros Christodoulakis
Professor, Department of Electronic and Computer Engineering

....
Michael Paterakis
Associate Professor, Department of Electronic and Computer Engineering

Hierarchical Caching and Prefetching for Improving the Performance of Continuous Media Servers

by

Stavros Harizopoulos

Submitted to the Department of Electronic and Computer Engineering

August 11, 1998

in Partial Fulfillment of the Requirements for the Diploma of Engineering in Electronic and Computer Engineering at the Technical University of Crete

Abstract

The number of concurrent video or audio streams a disk-based Continuous Media Server can support, is limited by the time needed for the associated data blocks to be retrieved from secondary storage. Along with the cost of RAM, these two strongly interrelated parameters determine the cost/performance metric of the Media Server. Research efforts have focused either on maximizing disk performance but typically at high expenses of main memory, or on using efficiently memory to achieve high performance. In this study both factors are considered and towards this, pre-fetching strategies into cache hierarchies are introduced. These techniques exploit the large predictability that continuous media requests show, to increase the number of supported streams while at the same time keep low the memory requirements and the start-up latency. The analysis and experimentation with drive-accurate simulation models of modern disk drives show significant gains that become more encouraging when projected with the trends in storage technology.

Thesis Supervisor: Peter Triantafillou

Acknowledgments

It is with certainty that I acknowledge that I would not have been able to undertake and complete this thesis without the help of many people. Firstly, I would like to thank Makis Liousas and Sotiris Karakitsos for believing in me and providing the strongest foundation for my studies.

I would like to thank Professor Michael Paterakis and Professor Stavros Christodoulakis for examining my thesis as well as Professor Apostolos Dollas for his helpful advising. The completion and presentation of this work would not be possible without the technical support found at the Telecom Laboratory and the assistance of Vlasis Tsiatsis and Lina Papadaki. Part of this thesis is based on a disk system simulator developed at the University of Michigan; I would like to acknowledge the authors of this simulator – the DiskSim Simulation Environment – professors Gregory R. Ganger, Bruce L. Worthington, and Yale N. Patt.

In addition, I wish to acknowledge friends, co-students, stuff and professors of the department of Electronic and Computer Engineering for their encouragement and support throughout my five-year career at T.U.C. I could not thank enough the whole Harizopoulos family – Ioannis, Zaharenia, Nikolaos, Evanthia and Ioannis – for keep standing beside me all these years.

Above all, I am deeply indebted to Professor Peter Triantafillou for guiding me and supervising this thesis. Our working together is something to remember…

Contents

Chapter 1

Introduction

1.1 Applications

The technological evolution of our century has been uniquely expressed in terms of transmission of information. This is exemplified best by the ever-growing, beyond any predictions, use of the Internet (over 700 millions of users by the end of the decade [26]) or even by the continuously increasing market of mobile phones (it is predicted that the GSM standard will have a combined subscriber base of 300 million by the year 2001 [27]). The underlying infrastructure of these technological advances involves two entities; (a) a centralized node, which handles the collection and distribution of information (a Continuous Media Server in this study) and (b) a high-bandwidth network, over which, the transportation of the information, to a multitude of end users, takes place.

Modern applications that base their existence on the advances in computer and communication technologies are characterized as broadband applications [31]. Those that involve the use of a Continuous Media server, are classified into three categories: (a) Retrieval, (b) Messaging and (c) Distribution services [32]. Most ongoing research is focused on video-on-demand services, a retrieval service, that allow geographically distributed viewers to request video objects from the central server complex [33, 34, 35, 36]. In addition to entertainment purposes -- as a replacement for cable television broadcasts and video rental -- video-on-demand can be used for remote education and training purposes, as well. Other broadband applications include video mail, a Messaging service, while Distribution services employ in general video and audio transmission services, such as distribution of TV programs to local stations, video information and high-quality audio. A concept that underlies over these services, is the Digital Library, which refers to the collection and storing into tapes and disks, of large portions of multimedia: video, audio, still images, and text.

1.2 Technologies and Trends

Networks

Research into Network technologies aims to integrate all services in a digital high-bandwidth network, in a cost-effective manner. The Internet, its channels and its underlying transport layer protocols -- TCP/IP -- are characterized as insufficient to support broadband applications. Instead, the network community focuses on Broadband Integrated Services Digital Networks (B-ISDNs), where the use of high-speed optical fiber channels and ATM -- as the transfer mode -- make possible the modern, bandwidth-consuming services. ATM networks offer unique scalability in speed and network size supporting link speeds of T-1/E-1 to OC-12 (622 Mbps) today and into the multi-Gbps range before the end of the decade [30].

Video, audio

The efficient exploitation of network bandwidth imposes the use of compression techniques onto deliverable information. Multimedia services are currently evolving around the Joint Photographic Experts Group (JPEG) standard for still-image delivery and Motion Picture Experts Group (MPEG) video technology for motion video. The underlying compression technique, which is used by both standards, is Discrete Cosine Transform (DCT), along with Motion Compensation (MC) algorithms that apply to MPEG. The quality of video compressed with an MPEG scheme at rates of 1.2 Mbps has often been observed to be similar to that of VHS recording [28]. Two widely known members of the MPEG family are MPEG-1, with an average data rate of 1.5 Mbps and MPEG-2, which, at 4 Mbps, can provide HDTV program quality. After November 1998, MPEG-4 will be the newest member. It will provide digital video at rates lower than 1 Mbps, while it promises wide compatibility;

part of its technology is based on QuickTime. At the same time, other compression techniques are being studied, such as Fractal compression, Region Object Based (ROB) coding and Model coding.

MPEG family algorithms have been used in audio compression too, to provide CD-like sound quality. MPEG audio layer III, for example, achieves a compression ratio of 1:10 - 1: 12 and a data rate of 128 - 112 kbps. The new MPEG-2 Advanced Audio Coding (AAC) scheme, based on Modified Discrete Cosine Transform (MDCT), achieves even lower data rates, at the same sound quality, but it is not yet widely spread, due to its late introduction (it did not get into the DVD specifications). Another well known compression scheme is Dolby AC-3 (at 192 kbps for stereo sound), which owes its popularity to its use by systems for cinema (theaters and DVD).

Computer systems

Moore’s law, which states that semiconductor power doubles every 18 months, has been an accurate predictor of chip advancement for decades, but recent technological evolution seems to defy it. As for today, there are CISC CPU’s running at 400 MHz (Intel’s Pentium II) and RISC CPU’s at 600 MHz (Digital’s Alpha). More manufactures use 0.25-micron technology and the projection for the next three years is that the increase in CPU’s performance will continue. (Intel has already announced a 0.18-micron technology CPU by the end of 1998). A typical workstation nowadays includes 128-512 MB of RAM while the system buses can be as fast as 80 MB/sec (Ultra-2 SCSI). Disk technologies and trends are described analytically in section 2.1. What can be safely stated after the examination of all related technologies, is that the bottleneck in a complete multimedia delivery system will be the I/O subsystem.

1.3 Architecture of Continuous Media Servers

A media server offers access to a collection of multimedia: video, audio, images, and text. These media data must be retrieved from the secondary and tertiary memory at a specific rate (or else we experience "hiccups" or "glitches"). The usual model of a Media server involves a tape library from which large portions of media streams are extracted and written onto hard disks periodically. Then, the server services the requests from the disk. To ensure continuous playback for a data stream (video or audio), a double buffer in main memory is typically needed. The data read from disk fill one half of this buffer while at the same time the data previously stored from disk to 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.

This hierarchical model of the media server -- RAM on the top of the hierarchy, in the middle a disk array, and at the bottom of the hierarchy a large tape library -- has been evolved around cost / performance tradeoffs. As it is too costly to store all information on disks (and typically impossible to store it in main memory) and access to tertiary storage is too slow (up to a few minutes), this hierarchical model emerged.

As multiple streams must be supported concurrently, the media server typically services these streams in *rounds*. During a round of a given time length the system reads one segment from each stream that is adequate to sustain playback for the duration of the round.

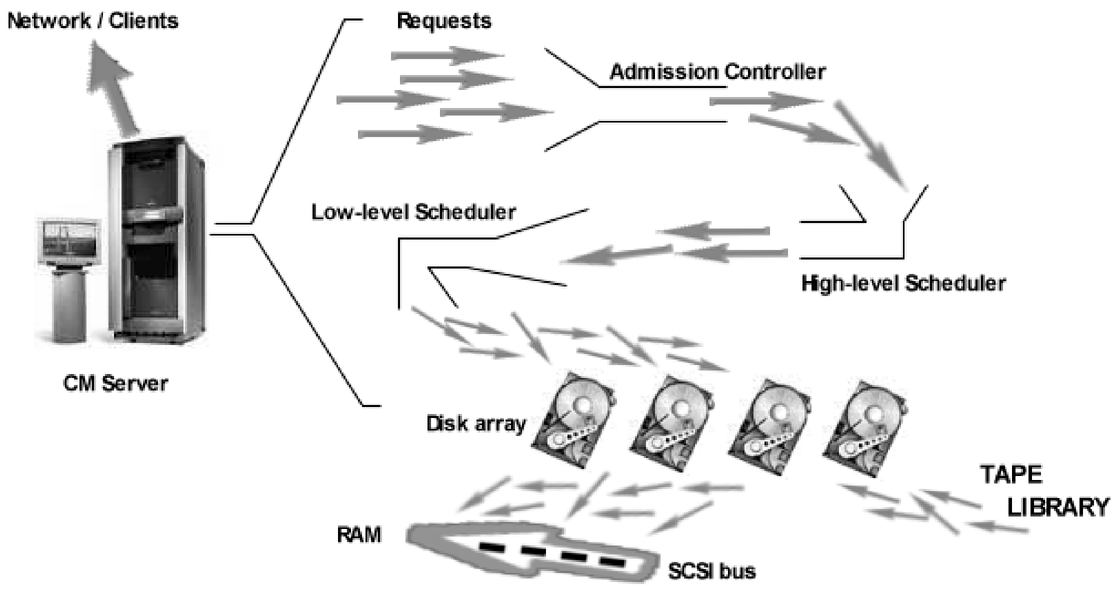

Figure 1: Architecture of a CM Server

Figure 1 gives a diagram of the basic parts found in a Continuous Media server. As soon as the clients' requests arrive, a call admission policy decides which requests will be satisfied immediately, based on the available resources. The remaining requests -- if any -- will be rejected or placed in a queue. A high-level scheduler processes the admitted requests: it looks up where the requested data lie (in which disk or tape), fetches the data from tapes to disks (if needed), and sends the resulting batch of requests to disk drives. Then, the low-level scheduler schedules separately for each drive the corresponding batch of requests, reducing as possible the disk head positioning overheads. The requested data are transferred from the disk media to the disk buffer and then through the SCSI bus to server's (host) RAM. Finally, the application running on the CM server is responsible to transport the data through the network to clients.

1.4 The Problem

The techniques presented here aim to increase the disk drive's performance for video and audio applications. Although a client may be able to perform operations that result into not continuous data requests (i.e. VCR operations on a video), this study considers the continuous requests that occur from simple playback of video and audio. The primary performance metric is the *maximum throughput* achievable by a drive, which is defined to be the maximum number of continuous streams (e.g., videos or audio) that can be supported by the drive. To complete the performance goals, it is added that all supported streams should not suffer from any interruptions in the timely delivery of stream data (termed as hiccups or glitches).

Two important factors that limit the cost / performance of the media server are the time needed for the data to be transferred from the disk to main memory and the cost (or implementation) of the RAM. Recent research has shown that these two parameters are strongly interrelated and achieving high throughput often comes at a huge cost in memory [1]. This work, instead, tries to increase the maximum throughput while at the same time keep the duration of a round constant with relatively lower memory requirements and shorter start-up latency.

What it is proposed here is a set of algorithms that exploit:

(i) the large predictability that continuous media requests show,

(ii) the existence of drive-level and other higher-level caches,

(iii) the existence of powerful controllers which can accept "hints" as to the sequential nature of future requests by an application, and

(iv) the faster pace of improvement of the transfer time versus that of the disk head positioning time.

The proposed algorithms introduce caching and prefetching strategies so to improve the drive's maximum stream throughput.

Specifically, disk and higher-level caches will be used to prefetch continuous data segments of one or more of the predicted requests that logically follow the demand of a single stream of continuous media. While significant portion of time seems being lost this way during prefetching, it is showed

(i) how to utilize higher-level caches in order to avoid experiencing stream glitches because of the prefetching time overhead, and

(ii) that the overall stream throughput is actually increased as the prefetched requests will be served from the disk's cache, without the overhead time needed for positioning onto disk.

Our analyses and experimentation show that achieving high throughput gains while keeping low the memory requirements and reducing the start-up latencies at the same time, is feasible.

The study will concentrate on the performance of a single disk drive. It also assumes that each disk drive is logically divided in partitions. Each partition is a group of contiguous tracks. It stores a number of consecutive blocks of a continuous data object. For example, if 25 videos can be supported by a disk's bandwidth, given the size of each video (over 2GB for an MPEG-2 90-minute video) it is clear that not all 25 videos can fit in a single disk. So it is assumed a segmentation of 25 logical partitions, each storing 1/25th of a video. This is a realistic scenario. For example, in a hierarchical multimedia storage [24,25] the next (1/25th) video segment can be brought in time from tertiary storage. In addition, in disk array environments, using the coarse-grained striping technique like the one suggested in [16,17] can result in a placement like the one it is assumed.

1.5 Organization of the Thesis

The rest of the thesis is organized into four chapters. Chapter 2 describes several features of modern disk drives and discusses the trends in storage technologies; it also summarizes the related research in disk scheduling algorithms and points out the research directions that developments in drives have deployed. In chapter 3, the proposed algorithms along with their analyses, are introduced. Graphs that show the expected gains are presented. Next (chapter 4) the experimentation test bed is described and the simulation results are presented; a discussion follows. Finally, the conclusions drawn from this study are included in chapter 5 along with references to future work.

Chapter 2

Modern Disk Drives

2.1 Technological Characteristics

Disk drives base their function in both mechanical and electronic parts. The mechanical parts are the recording components (the rotating disks and the heads that access them) and the positioning components (an arm assembly that moves the heads into the correct position together with a track-following system that keeps it in place). The electronic part of the drive consists of the disk controller; it contains a microprocessor, a memory buffer, and an interface to the host bus. The controller is responsible for the storage and retrieval of data via the mechanical components and performs mappings between incoming logical addresses and the physical disk sectors that store the information [37]. More complex controllers include logic that interferes in the order that data requests are serviced in order to improve performance.

Geometry and Components

A disk drive stores its information on a set of rapidly rotating platters (on a common axis) coated on both sides with magnetic media (see figure 2). Data blocks are written in a series of concentric circles, or tracks on each surface. A track is divided into sectors, the minimum unit of data storage; a sector typically holds 512 bytes of data plus some header/trailer information such as error correction data or positioning information. A set of tracks, each from a different surface and equidistant from the center of the spindle, form a cylinder. Each platter surface has an associated disk head responsible for recording (writing) and later sensing (reading) the magnetic flux variations on the platter's surface. The disk drive has a single read-write data channel that can be switched between the heads. Each disk head is attached to a disk arm - a lever that is pivoted near one end on a rotation bearing. All the disk arms are attached to the same rotation pivot, so that moving one head causes the others to move as well. The heads are grouped together such that a given disk arm position corresponds to a specific cylinder of data. Therefore, a physical location is defined by its cylinder, head, and sector [37, 38].

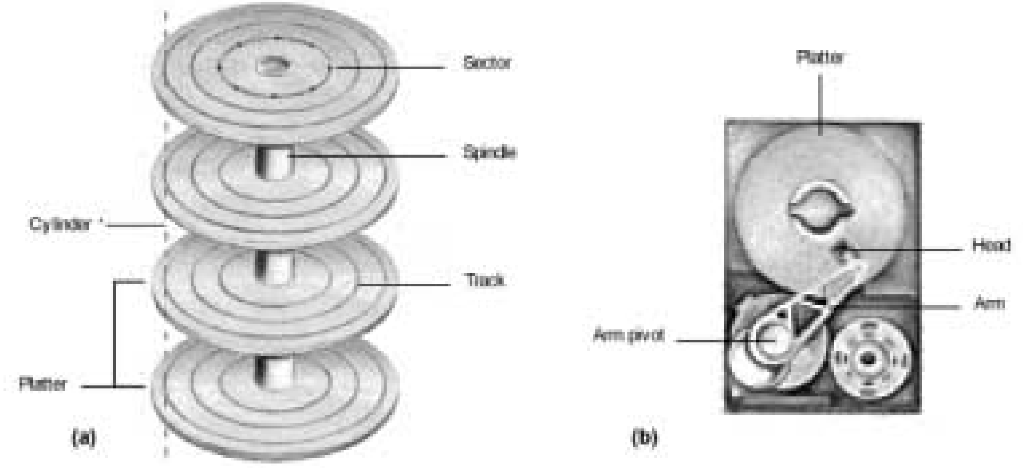

Figure 2: Mechanical components of a disk drive; (a) side view, (b) top view

Disk Access Cost

A data request serviced by the disk drive may suffer one or more mechanical delays. First, a head movement, or seek, towards the target cylinder occurs. Once the disk arm is located at the correct track, a rotational delay is incurred while waiting for the requested sectors to rotate into position under the read/write head. Then, the data transfer begins. Further mechanical delays result if multiple cylinders or tracks must be accessed. The latter can happen when more than one sectors are requested. The way the stored data are organized in a set of platters is in ascending physical address order from the lowermost track of a cylinder to the uppermost of the same cylinder and from the innermost cylinder to the outermost. Therefore, it is possible that a large request corresponds to sectors which lie in different surfaces within the same or different cylinders.

A seek is composed of

(i) a *speedup*, where the arm is accelerated until it reaches half of the seek distance or a fixed maximum velocity,

(ii) a *coast* for long seeks, where the arm moves at its maximum velocity,

(iii) a *slowdown*, where the arm is brought to rest close to the desired track, and

(iv) a *settle*, where the disk controller adjusts the head to access the desired location [37].

Seeking over tiny distances (less than 10 cylinders) is dominated by the settle time (1-3 msec). Short seeks (less than 20% of the total number of cylinders) are affected mostly by the constant-acceleration phase, and so their time is proportional to the square root of the seek distance plus the settle time. The time that long seeks take, is determined by the coast time plus a constant overhead. The function of seek time vs seek distance is usually modeled by an equation like the following:

Seek *DISTANCE*	Seek time	
$< C$ cylinders	$a + b * \sqrt{DISTANCE}$	
$\geq C$ cylinders	$A + B * DISTANCE$	(1)

where the parameters a, b, A, B and C are specified by the vendor or can be estimated.

At the end of a seek the head position must be fine-tuned and kept on the desired track. This is the function of the *track-following* system that uses positioning information recorded on the disk at manufacturing time to determine whether the disk head is correctly aligned. It also performs the *head switches* and *cylinder switches.* When the controller switches its data channel between two adjacent surfaces in the same cylinder, the new active head may need repositioning as the alignment of the tracks within a cylinder is not perfectly vertical (head switch). Similarly, repositioning of the disk head is needed when the disk arm has to be moved from the last track of a cylinder to the first track of the next (cylinder switch). A head switch takes typically one third to one half of the time taken to perform a settle at the end of a seek, while a cylinder switch needs about the same time as the settle [37].

After the accurate settling of the disk head on the target track, the requested sectors must rotate into position under the head. The elapsed time between the end of settling and the beginning of the read/write operation, is known as *rotational latency*. This time depends directly on the spindle rotation speed which has increased recently to as much as 10,000 rpm [29]. The transfer time of a sector depends on the rotation speed too, as well as on the linear density with which bits can be recorded on tracks, and on the encoding method that is used on the written/read data.

Data Layout

A disk appears to its host as a linear vector of addressable blocks. However, the mapping of these blocks to physical sectors on the disk is not highly sequential. The latter occurs as techniques that improve overall characteristics of the disk interfere in the way the mapping is done. Such schemes are *zoned recording* or *zoning*, *track and cylinder skewing*, and *sparing*.

Under zoning schemes the set of cylinders is partitioned into multiple zones. Zones near the outer edge have more sectors per track than zones in the inside. This way, the storage capacity is maximized as linear density remains near the maximum that the drive can support. At the other hand, since the data transfer rate is proportional to the rate at which the media passes under the head, the outer zones have higher transfer rates.

Track and cylinder skewing succeed faster sequential access across track and cylinder boundaries. Given that a positioning delay is incurred when switching heads or seeking between adjacent cylinders, skewing schemes assure that the first logical block of each track or cylinder is generally offset (skewed) from the first logical block of the previous track or cylinder. Thus a request that crosses a track or cylinder boundary is satisfied at nearly full media speed as the next logical block lies under the disk head immediately after the settling of the head has finished. Otherwise, a delay of an almost full rotation would occur. Each zone has its own track and cylinder skew factors.

Sparing is a process during which, references to defective sectors are remapped to other portions of the disk. Defects are identified at the factory and a list of them is constructed. As more defects occur during the course of the disk's lifetime, they are added to this list. *Slip sparing* or *slipping* takes place when the logical block that would map to the bad sector and the ones after it are "slipped" by one sector or by a whole track. *Reallocation* occurs when defective sectors are discovered during normal disk usage and so dynamically remapping to spare regions (that are reserved in disk for this purpose) is done.

On-Board Cache

A critical development has been the incorporation of drive-level caches into disk drives (figure 3). These caches have been evolved around the extension of the primitive speed-matching buffers. A speed-matching buffer built into the drive electronics is essential, as it provides a temporary storage space that eliminates or reduces any bottleneck in the flow of data between the disk and host CPU. This results from either the speedy CPU having to wait for data from the much slower electromechanical disk, or to the disk drive controller transferring information faster than the computer operating system can respond. The buffer is also used for temporarily storing requested data when the host bus is not available; otherwise an extra rotational delay would occur.

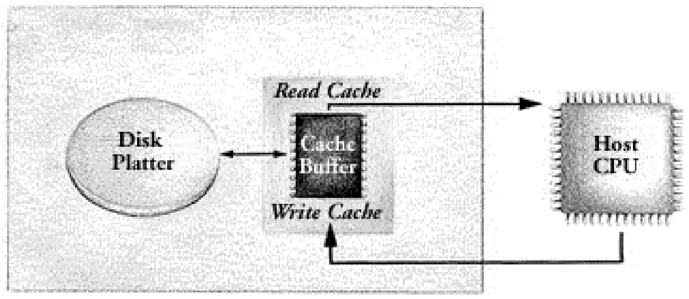

Figure 3: On-board cache buffer

Nowadays, more space and logic on the disk is dedicated to drive-level caches. For example, Quantum drives [10] can be bought with up to 4MB of cache. When accompanied with caching and prefetching techniques these embedded disk caches can significantly increase the drive's performance. Caching is performed in both reads and writes. A read that hits in the cache can be satisfied much quicker than seeking to the data and reading it off the disk. Read-ahead becomes more beneficial in environments where highly sequential requests occur; an aggressive read caching policy is suited best for this kind of workloads. Write caching is used to reduce latency of write requests through the immediate acknowledge of them and to increase throughput through reducing overwrites on the disk media and allowing the controller to reschedule a batch of stored in cache writes in near-optimal fashion. In order for the cache logic to keep up with these functions, buffer segmentation methods have been developed, that use the available buffer space more efficiently. Instead of using fixed length segments dedicated to use with only read or write operations, an algorithm determines the size of the data transfer, regardless of whether the operation is a read or write, and calculates the optimum amount of the buffer to allocate. An example of this *Adaptive Segmentation* is given below [10].

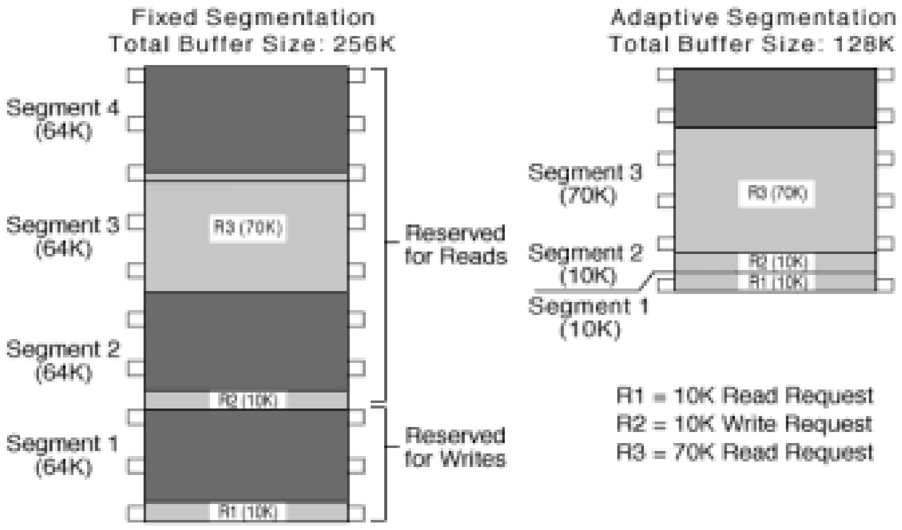

Figure 4: Example of adaptive vs. fixed segmentation

A 128K buffer with Adaptive Segmentation can provide as much or more usable buffer space as a 256K buffer with fixed segmentation. After the three data requests, R1, R2, and R3, the buffer with fixed segmentation is full. At the next request, if the data is not already in the buffer, at least one 64K segment will be overwritten with new data. With the same three requests, the buffer with Adaptive Segmentation still has space available. In addition, when this buffer is filled, the drive will overwrite only the amount of buffer space required.

I/O Channels and Buses

Since new generations of higher-speed microprocessors are emerging and disk recording densities are increasing, hard disk drive manufacturers are presenting drives with faster read channels and higher sustained data transfer rates. Improving internal (disk surface-to-cache buffer) data rate, however, creates a need for faster external (disk buffer-to-host) data transfer rates. Two widely used drive interfaces are Intelligent Disk Electronics (IDE) and Small Computer System Interface (SCSI). The first of them is a low-cost interface that is mainly used by home computers. It is usually connected to a bridge to a faster bus such as PCI (Peripheral Components Interface, it supports up to 132 MB/sec transfer rate) and its latest variation, Enhanced IDE with Ultra DMA [10, 39], can deliver data transfer rates up to 33 MB/sec.

SCSI has been traditionally the choice of high-performance servers and workstations as well as Apple’s Macintosh. Because SCSI is a bus, more than one device can be attached to it. Current implementations support up to 16 addresses (wide SCSI) using only one interrupt request. Its underlying technology is continually evolving, providing more bus bandwidth, increases in packaging and density, and increases in configuration flexibility. One of the more recent developments is Ultra2 SCSI (LVD), which has a low-voltage differential (LVD) interface and transfer rates of up to 80 MB/sec.

Serial channels offer unique characteristics that outperform SCSI, but they are not widely used yet. Serial interfaces offer higher speeds, fewer wires and require a smaller connector. The first generation of the Fibre Channel Arbitrated Loop (FC-AL) provides a path to much higher performance than other serial storage interfaces. Currently, with dual porting, FC-AL drives are capable of delivering buffer-to-host data transfer rates of 200 MB/second while enabling up to 126 devices per interconnect. Furthermore, the FC-AL interface allows data to travel over longer distances at higher speeds than parallel (single-ended) SCSI.

Trends

The general technology trends currently call for even higher-performance disk controllers (with performance similar to a >200 MHz Pentium) before year 2000. These "embedded CPUs" will have available an "embedded cache" with much greater capacity than what is available today. 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 (which has doubled in the last four years). It is expected that the 10MB/s transfer rates will near 40MB/s by year 2,000. Further upgrades to I/O buses are now in development. Ultra3 SCSI will provide up to 160 MB/sec transfer rate; Fibre Channel future products will push that level of performance to 400 MB/sec, while increasing the level of chip integration to improve costs. At the same time the expected speed-ups in the disk head’s positioning times are not expected to keep pace (the observed compound annual improvement for the last few years has been around 10%). The latter fact 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. NASDs are storage devices with powerful controllers, operating on large on-board caches, able to run streamlined versions of storage and file related, as well as network-related, software protocol stacks.

2.2 Related Research

Disk Scheduling Algorithms

Early scheduling algorithms focused on reducing seek times. The Shortest Seek Time First (SSTF) algorithm [4] achieved this; however, it incurs a high response time variance and is starvation-bound. SCAN scheduling was also proposed in [4]. It serves requests in an elevator-like manner, (as it sweeps the disk in either one or both directions) allowing seek-time optimizations, while reducing the response time variance. In [5] the authors proposed a parameterized generalization of SSTF and SCAN. The V(R) algorithm operates as SSTF except that, every time it changes direction it adds a penalty, dependent on the parameter R and the seek distance. When R=1 (R=0) V(R) reduces to SCAN (SSTF). The performance of these algorithms, as well as other variations of SCAN, such as the LOOK algorithm (which changes scanning direction when no more requests are pending in the current direction) has been studied experimentally in [7] and analytically in [6].

A significant development in modern-disk scheduling was to also target the high costs owing to rotational delays. These algorithms [9, 8] attempted to minimize the sum of seek and rotational delays by favoring, for instance, the request with the Smallest Positioning Time First (SPTF). Recently, disk controllers have been developed which can rearrange the order of submitted requests exploiting their detailed knowledge of the requested blocks' positions and minimize seek and rotational delays [10].

Disk Scheduling Algorithms for Continuous Data Requests

The SCAN algorithm is inapplicable for continuous data requests since it is oblivious of deadlines. The Earliest Deadline First (EDF) algorithm [11] is a natural choice; however, it has poor performance since it does not attempt to reduce the overhead. SCAN-EDF [12] is a hybrid that serves requests in EDF order, but when several requests have the same deadline, they are served using SCAN. Most related recent research has adopted the notion of scheduling with rounds [13, 14, 15, 16, 17]. Each continuous data object (stream) is divided into blocks (also called fragments) such that the playback duration of each fragment is some constant time (typically, from one to a few seconds).

The round length represents an upper bound on the time in which the storage server must retrieve from disk the next fragments for all active continuous displays, or some displays will suffer a glitch. Within a round, it is possible to employ either a round-robin or a SCAN algorithm. The latter performs seek optimizations, resulting in better disk throughput. However, this is achieved at the expense of higher start-up latencies; the display cannot be started immediately after the retrieval of its first block but only after the end of the round. This is done to avoid glitches, since the service order differs from round to round. This limitation is not present when using round-robin scheduling, which also has lower RAM buffer requirements since it does not require the double-buffering scheme required by SCAN between successive rounds. A compromise was achieved with the Group Sweeping Scheduling (GSS) algorithm [7]. GSS groups streams and employs round-robin scheduling for the different groups and SCAN scheduling for the streams' blocks in a group. Thus, when there is only one group GSS reduces to SCAN and when each stream is in its own group GSS reduces to round-robin.

Disk Scheduling Algorithms for Mixed Media Workloads

Works with some relevance to mixed-media disk scheduling are [12, 18, 19, 20, 21]. The work in [12] overviewed the performance goals for mixed workload scheduling, which are similar to [22], and studied only how some known algorithms for continuous-request scheduling, such as EDF and SCAN/EDF, affect the response time of discrete (or aperiodic, as they call them) requests. A simple scheduling scheme, called "the immediate server approach" [18] was employed for discrete requests, according to which discrete requests were served in between two successive EDF-selected continuous requests.

Since multimedia disk scheduling research has moved away from EDF-based algorithms, the works in [19, 20] attempted to analytically model and predict the performance of multimedia servers with mixed workloads, when scheduling is based on the notion of rounds. The work in [21] is a preliminary attempt to define the issues and present initial algorithms towards efficient mixed-media disk scheduling. The work in [22] takes further the work in [20, 21] and presents novel scheduling algorithms, which introduce drastic performance improvements; it also defines a taxonomy of related algorithms. The work reported here, proposes further performance improvement by exploiting cache hierarchies, including the large on-board buffer of modern disk drives, for prefetching purposes.

Caching and Prefetching Techniques

The on-board memory buffers of modern disk drives have been widely used for increasing performance via prefetching in traditional applications. Read-ahead -- retrieving and caching data that are expected to be requested by the host in the near future -- has proved to be very beneficial especially for highly-sequential workloads. The disk controller decides on the aggressiveness of the prefetching strategy -- i.e. should a read-ahead fill a cache segment or read a whole track and stop or it should cross track and cylinder boundaries and fill a larger cache line -- based on various assumptions as for the nature of the workload and monitoring the system.

The disk cache has been also used in traditional applications for "zero-latency reads" (requests for whole tracks that are satisfied as soon as the disk head ends seeking; the track is read into cache during one rotation and it is delivered logically reordered). However, the increase in track densities minimized the benefits of this technique. Caching strategies are used during writes too. Immediate reporting of writes that arrived in cache and are being scheduled for actual writing on media can result in reduced overhead, while overwrites that meet their previous copies in cache (as these writes are artificially delayed in cache before written on disk media) introduce significant savings. Once the controller has several pending writes in cache, it can schedule them in a near optimal manner.

To the best of our knowledge, there are no caching strategies in the literature to improve CM server retrieval performance except for those aggressive policies designed for large sequential accesses.

Hierarchical Caching

Storage technology for large storage servers allows the introduction of caches at multiple levels of the storage hierarchy, [23] such as at the host's RAM, at a multi-device controller's cache, or at a disk array's controller cache, and at the drive-level cache [23]. A significant thread of research deals with allowing the application to pass hints to the underlying operating system (or even the controller) which would describe its future access references and help thus in forming an efficient plan for prefetching and caching. To our knowledge the exploitation of these hierarchical caches at CM servers has not been addressed by related work. This is a study towards this goal.

Memory Management in Scheduling Algorithms for CM Servers

Research has also been conducted on employing efficient memory management strategies. The scheduling of continuous media requests is interfered so to reduce memory requirements via buffer sharing [2, 3]. Towards this, a shared memory pool can be used; when the consumption of the data of a stream starts, the continuously increasing portion of data "played" is no longer needed and so its space can be occupied by data from other requests**.** A continuous media retrieval scheme studied in [1], Fixed-Stretch, tries to reduce the RAM requirements to the minimum possible value by equally spacing the requests within a round. That is, artificial delays are introduced so that the order in which streams are serviced in each round can be predefined. The latter permits maximum memory utilization in that the double buffers used to sustain playback are no longer needed as variations in retrieval times do not occur.

Chapter 3

Proposed Algorithms

In this chapter, four new techniques for retrieving streams of continuous media from disk-based secondary storage are presented. Prefetching of data segments in the disk cache as well as in higher-level caches is used in various ways according to possible system configurations. These algorithms reduce the disk access overhead more than it is feasible with conventional policies, while preserving the Quality of Service that the avoidance of glitches and reduced start-up latencies imply. Prefetching strategies surpass other continuous media retrieval schemes in that they can be adjusted to respond to increased throughput demands in a low memory consumption way. They can also be applied on top of other algorithms that involve the use of data placement policies, introducing additional gains. The existence of a large drive-level cache is not a prerequisite for prefetching to be applied; in one of the techniques described below, a prefetching algorithm is proposed showing a new way of improving the performance of a CM server.

3.1 The Sweep & Prefetch (S&P) Technique

Scheme Sweep is the well-known continuous media retrieval scheme that reduces disk seek overhead to improve throughput. During a round, Sweep reads each stream's next segment with a SCAN like policy. Double buffers in main memory are needed to ensure continuous playback. The scheme that is proposed here and described below, is called Sweep & Prefetch (S&P).

For a given duration of a round there is a specific upper bound in the number of streams *N* that can be supported by a disk in a feasible media server. This number is mainly dependent on two parameters: the total positioning overhead (the time needed for the disk head to be positioned on the beginning of a segment) and the total transfer time (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 from now on 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, by prefetching a segment -- that is the disk head continues reading the adjacent segment of the previously read segment and stores it into the disk cache -- the prefetched segment will be read at no positioning overhead; the number of prefetched segments in a round is denoted by the letter ***p***.

By forcing the disk head, after a segment read, to continue reading the next segment of the same stream into the disk cache and by doing this for ***p*** streams, then, at the end of the round, ***v*** streams will have been served (as random retrievals), while ***p*** segments will be in cache (prefetched segments). In the next round, those ***p*** segments will serve ***p*** streams at no positioning overhead, so that there will still be enough time to serve ***v*** additional streams and prefetch the next segments from ***p*** of the ***v*** streams served. Because the ***p*** segments are read at no positioning overhead, the total number of supported streams $(p+v)$ is now greater. That is, the maximum throughput has been increased at the expense of cache memory, while the duration of the round and the size of the segments remained constant. (The latter is a very desirable effect as it is shown in a subsequent section.)

Figure 5 demonstrates in a pictorial way the technique described. A box under another box stands for the next segment -- contiguously placed on disk -- of the same stream, while two adjacent boxes in the same line stand for the segments of two streams -- that will be scheduled within a round in the same order as they appear. Figure 5(a) introduces the maximum number of streams (25 in this example) that can be supported with Sweep in a given configuration. Then, in Figure 5(b), prefetching of 8 segments -- one for each of 8 streams -- is used to increase the supported 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 higher-level cache buffer. These streams will not suffer a glitch. This can be ensured as follows. 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 a higher-level cache (e.g., the host's RAM). The display of these 5 streams is delayed until all three blocks have been retrieved from the disk's surface. In this way, the higher-level cache(s) permit not to issue disk retrieval requests for some or all of these 5 streams in some round, without these streams experiencing any glitches. The higher-level cache buffers needing to implement this "triple buffering" are no longer needed after round 2 in Figure 5(b) and can be used by the new streams supported.

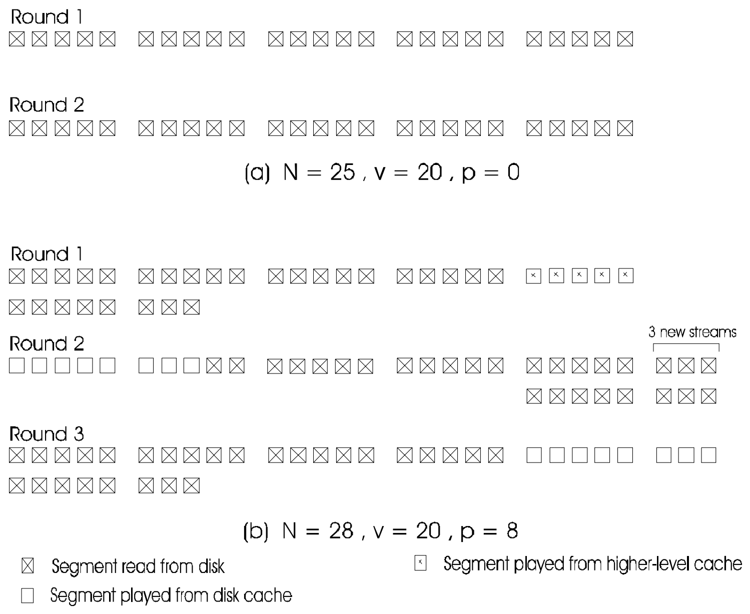
Round 1
Round 2
(a) N = 25 , v = 20 , p = 0
Round 1
Round 2
3 new streams
Round 3
(b) N = 28 , v = 20 , p = 8
Segment read from disk
Segment played from higher-level cache
Segment played from disk cache

Figure 5: Throughput increment via prefetching

For demonstration purposes of this series of schematic figures, we consider a disk drive with 10 MB/sec transfer time and a combined seek and rotational fixed overhead of 15 ms for every request. It is also assumed a constant display rate of 2 Mbps and round lengths of 1 sec. The latter implies that the size of any segment will be 250 KB. Given these values, the maximum number of segments that can be read within a round is 25, while the ratio of random retrieved segments that can be "exchanged" (in terms of time constraint) with prefetched segments, is 5 / 8 (40 ms and 25 ms retrieval times correspondingly).

Scheme S&P can be fully parameterized as far as ***v*** and ***p*** are concerned, always at the expense of disk cache memory size. Figure 6 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 3.3 another algorithm that is an extension of S&P is proposed, which can overcome this upper bound and further increase the maximum throughput.

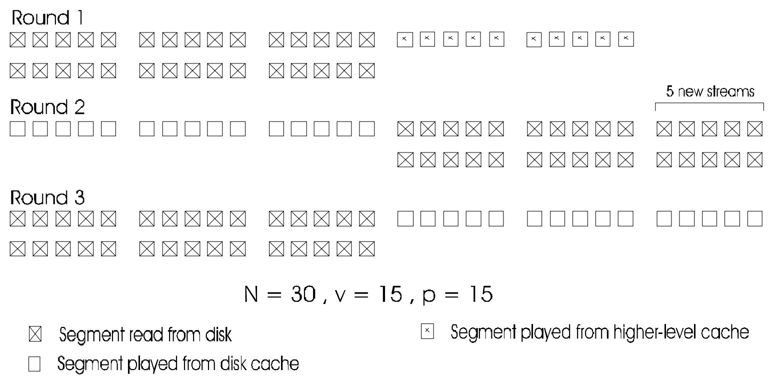

Figure 6: Further throughput increment

3.2 Gradual Prefetching

Scheme S&P, as presented in section 3.1, starts servicing the streams without prefetching until the maximum number of streams are served. Then it switches into the "prefetch" mode, and exploiting the higher-level caches, increases the maximum number of streams without any of them suffering from glitches. However, it has the drawbacks of requiring three cache buffers for each stream, which will be "skipped" in some round and the associated extra start-up latency. In this section it is shown how these drawbacks can be avoided.

While the extra space that the third buffer occupies is not a major drawback -- since it can be stored in any level of the cache hierarchy and it will be replaced by the new streams, that the "prefetch" mode will support -- the extra start-up latency could cause a degradation in the Quality of Service. In order to avoid the triple buffering of some streams and the associated start-up latency, the media server can be forced to function under the scheme S&P all the time, independently of the number of concurrent streams serviced. The number of streams that are prefetched at any time will be half of the overall streams supported. This implies that for every two newly admitted streams, during their first round, for one of them there will 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 3.1.

Thus, Gradual Prefetching is employed until the number of supported streams becomes the maximum possible. Therefore, under Gradual Prefetching there will always be enough time within a round to prefetch an additional segment for one of every two newly admitted streams. Thus, no stream will experience extra start-up latency.

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

Further increase of ***p*** is possible by prefetching more than one segment for some streams. Figure 7 exemplifies this by demonstrating 2-level and 3-level prefetches. The throughput is increased from 25 to 34 but the required cache size is no longer equal to the number of prefetched segments (analysis follows in section 3.5). Gradual prefetching can be applied to this scheme as well, once the depth of prefetching is known (this depends directly on the value of ***p***).

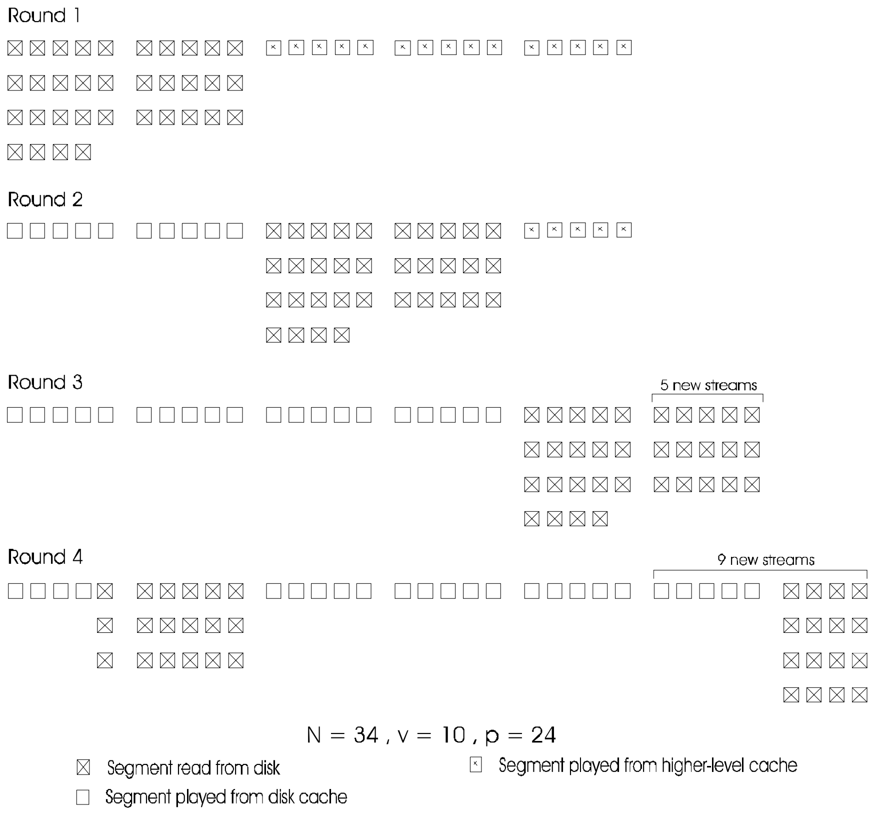

Figure 7: Throughput increment via multilevel prefetching

In MS&P, since multiple segments are prefetched for a given stream, the next stream S_i to be read from disk will be separated -- in terms of data placement onto disk -- by the previously read stream $S_j, j < i$ by as many streams as the number of prefetched segments for S_j. This assures 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 (the appropriate segments have been prefetched during earlier rounds) in time that a place in cache will be free for the segments of the stream S_i to be prefetched. It is reminded that the scheduler performs a SCAN-like policy.

When different levels (depths) of prefetches occur in MS&P -- at most two different levels -- a similar scheduling and caching policy like the one just described must be followed, but with two different values of the number of skipped streams, each one for each of the two groups of streams that have the same number of prefetched segments. In figure 8 the diagram of MS&P in figure 7 is redrawn to exemplify the policy described in this paragraph.

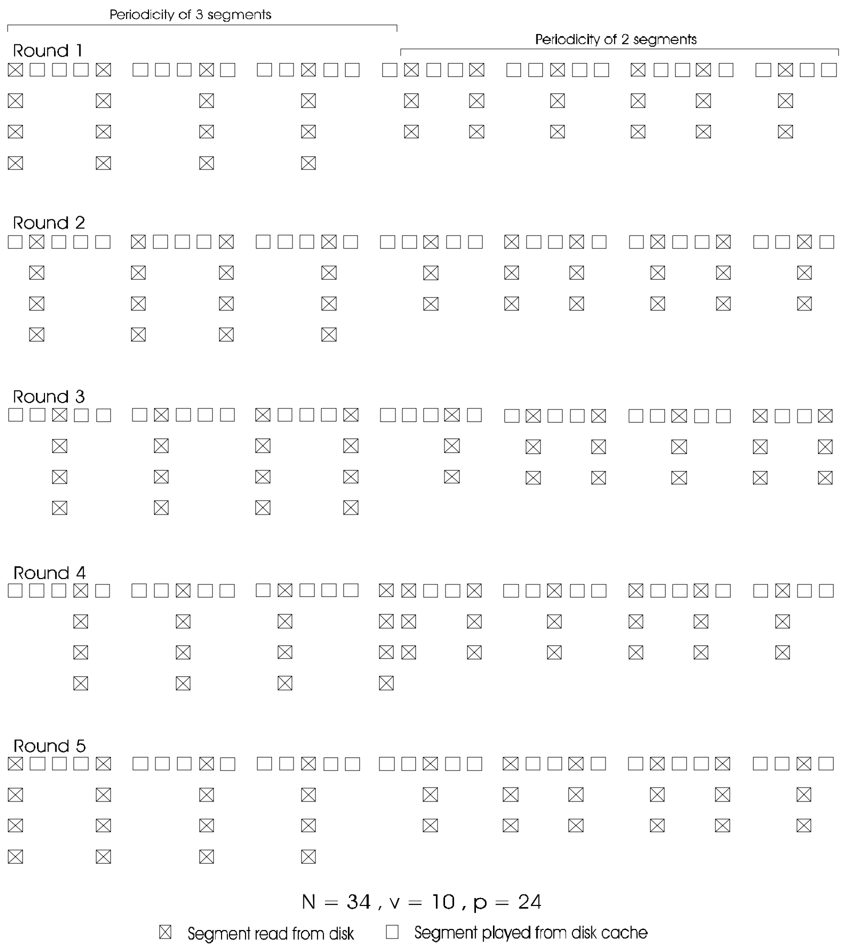

Figure 8: 2-way periodical MS&P

3.4 Grouped Periodic Multi-Round Prefetching (GPMP)

The algorithm described here is based on multiple prefetches but uses the host's RAM to temporarily store the prefetched segments. Caches in other levels as well as disk caches can be used to store a portion of these segments, but the technique will be analyzed using only the RAM, in order to facilitate a direct comparison to other multimedia retrieval schemes with memory management that do not exploit prefetching. This algorithm will be called Grouped 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.

During a round of GPMP all streams will be served -- that is, all segments that sustain the playback of the currently supported streams will be delivered to the host application that manages the flow of data. The key idea is that these segments don't have to be retrieved during each round from the disk; some of them will be found directly in the RAM. Thus, during a round only a fraction of the total streams supported will be randomly retrieved (denoted again with the letter ***v***) 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 -- the letter ***p*** that has been used in the previous algorithms was referring to the total number of prefetched segments.

In the next round, the segments that sustain the playback of the next ***v*** streams will be read along with the ***u*** prefetched segments for each one of the ***v*** streams. (The streams within a round as well as the different streams in subsequent rounds, are being served as before in a SCAN-like manner). It is true that 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 RAM as they had been prefetched in the previous rounds. (***N*** is the number of the total streams supported). The ***u*** prefetched segments of a stream read during a given round will sustain its playback from RAM for the next ***u*** rounds. That is, after ***u*** rounds the same streams read from disk during that given round have to be read from disk again.

The *epoch* consists of $(u+1)$ rounds. During an epoch a complete sweep of the disk is performed. From the disk behavior point of view, GPMP with the perspective of 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 streams supported under GPMP is $N = v \times (u+1)$. Figure 9 demonstrates GPMP in the same framework of the previous diagrams. The maximum number of streams supported at $u=5$ is 36. ($v=6$). Gradual Prefetching strategies assure that during Round 1 in figure 9, the 30 streams not served from disk have their segments already stored in RAM 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 RAM, each of the third group of 6 streams have 2 remaining segments in RAM and so on. Recall that "played" segments are discarded immediately.

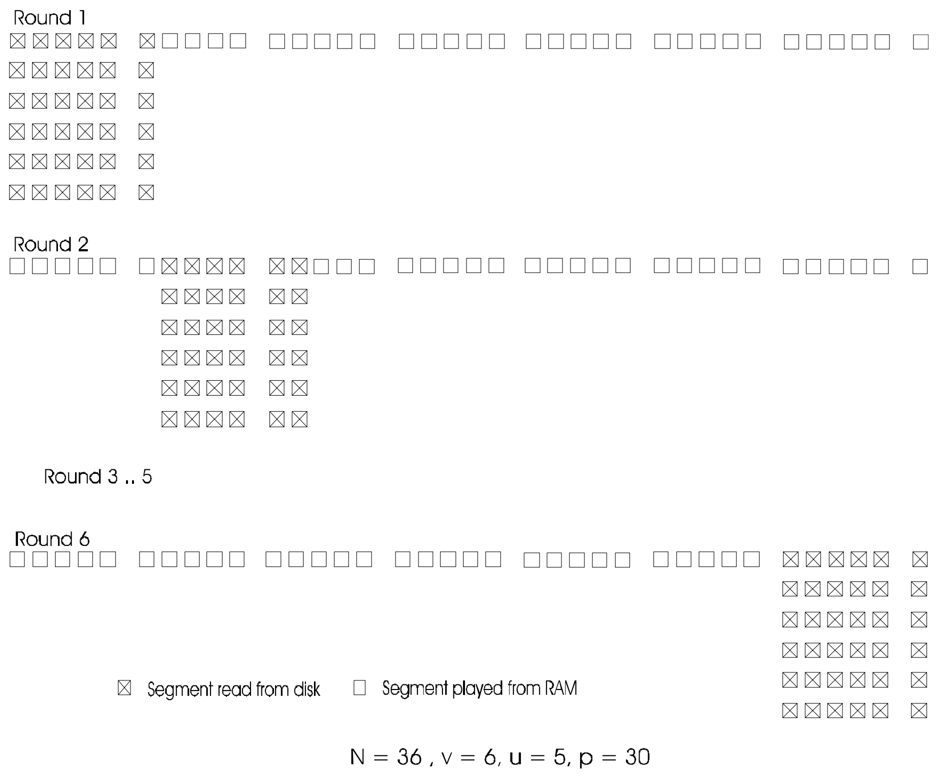

Figure 9: Grouped Periodic Multi-Round Prefetching (GPMP)

The scheduling algorithm used in GPMP is the same as in Sweep. In GPMP -- as a contrast to S&P and MS&P -- there is no need for a special schedule as the prefetched segments will lie already in RAM; only in the case that a cache would be used a schedule/caching policy like the previous one would be necessary.

3.5 Analysis

To analyze and compare the performance of Sweep versus the algorithms proposed, a single disk will be assumed as well as no special data placement policy (segments of media files are stored contiguously on disk). Each stream is assigned a private buffer in main memory to cope with variations in times between segment reads and no buffer sharing exists among the requests. Buffer sharing does introduce significant savings, but this is discussed later. The parameters used for the analysis, are:

- R: the duration of a round; it is constant for a given configuration.

- N: the maximum number of streams that can be displayed with each algorithm; it is computed by taking into account the worst case scenario, i.e., maximum disk head positioning overhead.

- B: the size of a continuous object segment (e.g., 250KB MPEG segments); this is taken as constant, for simplicity.

- p: the total number of segments prefetched, to be read from disk cache in later rounds.

- v: the number of segments that are randomly retrieved (number of streams read from disk in a round).

- u: the number of prefetched segments that belong to the same stream (it is used only in GPMP).

- Mem: the total size of system's main memory that is needed during the operation of the media server.

- dCache: the size of the disk cache memory that is needed for prefetching purposes; it is assumed a full utilization of disk cache while further cache requirements are assumed to be satisfied from higher-level caches.

- Transfer: the data transfer time of disk (disk media surface to main memory).

- Display: the display rate of a stream; the same display rates for all streams will be assumed (e.g. MPEG-1).

- pos(d): the function that computes the positioning overhead of the disk head, given a seek distance d (rotational delays are included).

3.5.1 Throughput

The round R 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 overheads for Sweep, while in the S&P family of algorithms only ***v*** segments meet these delays; the rest ***p*** segments are played directly from the cache at no time overhead and the time that remains until the completion of the round is used for prefetching ***p*** additional segments at disk transfer time overhead 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 $pos\left(TotalCyl / N\right)$. 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***:

Sweep: $$R = N \times (POS + B / Transfer) \qquad (2)$$

S&P family: $$^{1}R = v \times (POS + B / Transfer) + p \times B / Transfer \qquad (3)$$

In the above equation, we assume that 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 $N_{M3S\&P} = v \times (u+1)$, but by setting $p = v \times u$, the previous one relation can be used. It should be reminded here that ***N*** is the maximum possible number of streams and is different for different schemes. Equation (3) can be written as:

S&P family: $$R = N \times B / Transfer + v \times POS \qquad (4)$$

[1] 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.

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

Sweep:
$$B = \frac{N \times POS \times Display \times Transfer}{Transfer - N \times Display} \qquad (5)$$

S&P family:
$$B = \frac{v \times POS \times Display \times Transfer}{Transfer - N \times Display} \qquad (6)$$

From equations (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:

Sweep:
$$N_{Sweep} = \frac{1}{POS/R + Display/Transfer} \qquad (7)$$

S&P family:
$$N_{S\&P} = \frac{R + p \times POS}{R \times Display/Transfer + POS} \qquad (8)$$

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

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

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 consequently extend the duration of the round. This would mean also higher latency and higher memory requirements.

We can also see from equation (9) that the throughput gain of S&P algorithms over Sweep is $\frac{p \times POS}{R}\%$. The throughput gains over scheme Sweep with the same round length that introduced in the scenarios of figures 5-8 can be now computed accurately according to the last relation:

p = 8	(Figure 5)	throughput gain = 10.4 %
p = 15	(Figure 6)	throughput gain = 19.5 %
p = 24	(Figure 7)	throughput gain = 31.2 %
p = 30 (u = 5)	(Figure 8)	throughput gain = 45 %

3.5.2 Memory

For the time it has been assumed that no buffer sharing exists. Each stream requires twice the size of a segment in main memory so that the playback rate is sustained, independently of the variations in segment reads within two consecutive rounds. That is, there is a memory requirement for both Sweep and S&P algorithms of:

$$Mem = 2 \times N \times B \qquad (10)$$

S&P family of algorithms have an additional cache memory requirement that depends on the value of ***p*** for S&P and on the values of ***p*** and ***v*** for MS&P and GPMP (or on ***v*** and ***u***, for GPMP). The formulas computed in this section assume a provided scheduling and caching policy that delivers at least one segment from cache before it prefetches another one, so that an unread disk cache resident segment is not overwritten. This scheduling policy is the one presented in section 3.3.

When $p \leq v$ the cache required for S&P is :

S&P ($p \leq v$) $$dCache = B \times p \qquad (11)$$

The next equations compute the cache requirements of MS&P $(p > v)$:

MS&P $$dCache = B \times \sum_{i=0}^{\lfloor p/v \rfloor} [p - i \times v] \qquad (12)$$

Or (by substituting $p = N - v$ in (12)) *we set:*

MS&P $$dCache = B \times \sum_{i=0}^{\lfloor N/v - 1 \rfloor} \left[N - v \times (1+i) \right] \qquad (13)$$

To understand (12) please refer to figures 7 & 8.

The above equations hold for GPMP. In GPMP, since $N = v \times u$ the following holds:

GPMP $$dCache = B \times \sum_{i=0}^{u-1} \left[N - v \times (1+i) \right] \qquad (14)$$

The total memory that GPMP needs is computed from equations (10) and (14) as:

GPMP $$Mem = B \times \left(2 \times N + \sum_{i=0}^{u-1} \left[N - v \times (1+i) \right] \right) \qquad (15)$$

For a given N, we can use equations (5), (6) and (10) to derive the memory savings that S&P family algorithms introduce over Sweep. From equations (5), (6) we get:

$$B_{S\&P} = \frac{v}{N} \times B_{Sweep} \qquad (16)$$

Then, using equation (10) we derive for S&P and MS&P:

$$Mem_{S\&P} = \frac{v}{N} \times Mem_{Sweep} \qquad (17)$$

The above equation does not include the cache that S&P and MS&P use; the associated cache size -- as computed in equations (11) and (12) -- can be added to the above equation. However, as it is pointed out in the first two chapters, modern disk drives come with large on-board caches that are not used by Sweep but can be exploited by S&P. Other cache hierarchies can be used too. Even with that addition, S&P and MS&P outperform Sweep; this can be extracted from the graphs in section 3.5.3 and is verified in the next chapter.

GPMP can be compared directly to Sweep as it can be configured to not use any disk cache. Thus, equations (10), (15) and (16) result in the memory savings of GPMP over Sweep (the rather complicated fraction can never be greater or equal to one -- this can also be seen in the graphs of section 3.5.3):

$$Mem_{GPMP} = \frac{v \times \left(2 \times N + \sum_{i=0}^{u-1} \left[N - v \times (1+i) \right] \right)}{2 \times N^2} \times Mem_{Sweep} \qquad (18)$$

Memory Sharing

So far, it has been assumed that each stream needs a fixed buffer in host's RAM of size $2 \times B$. Memory requirements can be further reduced if the requests share their buffer space [2, 3]. Towards this, a shared memory pool can be used; when it starts the consumption of the data of a segment, the continuously increasing portion of data "played" is not longer needed and so its space can be occupied by data from other requests. According to [1] the minimum memory space required to support N streams under scheme Sweep -- with memory sharing -- is:

$$Mem = (N-1) \times B + N \times Display \times (R - \frac{(N-2) \times S}{Transfer}) \qquad (19)$$

The same relation can be used for S&P and MS&P. GPMP though, cannot use a memory sharing technique on the prefetched segments. Nevertheless, memory sharing can be applied to the segments that correspond to the playback in the current round. Thus, the memory needed under GPMP with memory sharing is derived from equations (14) and (19):

$$Mem_{GPMP} = (N-1) \times B + N \times Display \times (R - \frac{(N-2) \times S}{Transfer}) + B \times \sum_{i=0}^{u-1} \left[N - v \times (1+i) \right] \qquad (20)$$

or

$$Mem_{GPMP} = B \times \left((N-1) + N \times (1 - \frac{(N-2) \times S}{Transfer \times R}) + \sum_{i=0}^{u-1} \left[N - v \times (1+i) \right] \right) \qquad (21)$$

Even with this observation the overall memory needs of GPMP are far less than those of Sweep.

A scheme studied in [1] that tries to reduce the variability between the I/Os of a request by performing them in a fixed order from round to round, is the Fixed-Stretch. Because I/Os in a round are spaced out (the worst possible seek distance *TotalCyl* is used to ensure continuous playback) memory sharing is done at its maximum; the segment size, though, is larger as the round duration is artificially enlarged. According to [1] the minimum memory requirements of Fixed-Stretch with memory sharing and the size of the associated segment size are:

$$Mem_{F-S} = \frac{B \times (N+1)}{2} \tag{22}$$

$$B_{F-S} = \frac{N \times pos(TotalCyl) \times Transfer \times Display}{Transfer - (N \times Display)} \tag{23}$$

The segment sizes of Fixed-Stretch and Sweep are compared in the following equation:

$$B_{F-S} = \frac{pos(TotalCyl)}{POS} \times B_{Sweep} \tag{24}$$

3.5.3 Analytical Results

In order to compare better the performance of the new algorithms at various cache size configurations, different values of ***p*** and different round lengths, the graphs in this section based on the above equations have been constructed. The values of *Transfer* and *POS* are different from those used to demonstrate the techniques in figures 5-8. A real-world, projected to be up-to-date for the end of 1998, disk drive is assumed with average transfer rate of 14 MB/sec and average positioning delay of 11 msec. This disk drive is discussed analytically in chapter 4. The graphs in this section use these values for *Transfer* and *POS* so that a direct comparison with the experimental results (of the next chapter) can be done.

Other multimedia retrieval schemes have also been proposed in the academia. Nevertheless, in this evaluation only Sweep and Fixed-Stretch are considered as a comparison to the techniques presented here. This is done because these two schemes 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. Other schemes lie between these two schemes, such as GSS proposed in [7]. The algorithms that this study proposes -- especially GPMP -- exploit only the advantages of each of the two extreme schemes -- Sweep and Fixed-Stretch (as it can been seen from the graphs). Therefore, a comparison is not necessary to show that GPMP outperforms all other hybrid schemes.

In figure 10, 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 – the throughput achieved -- 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.

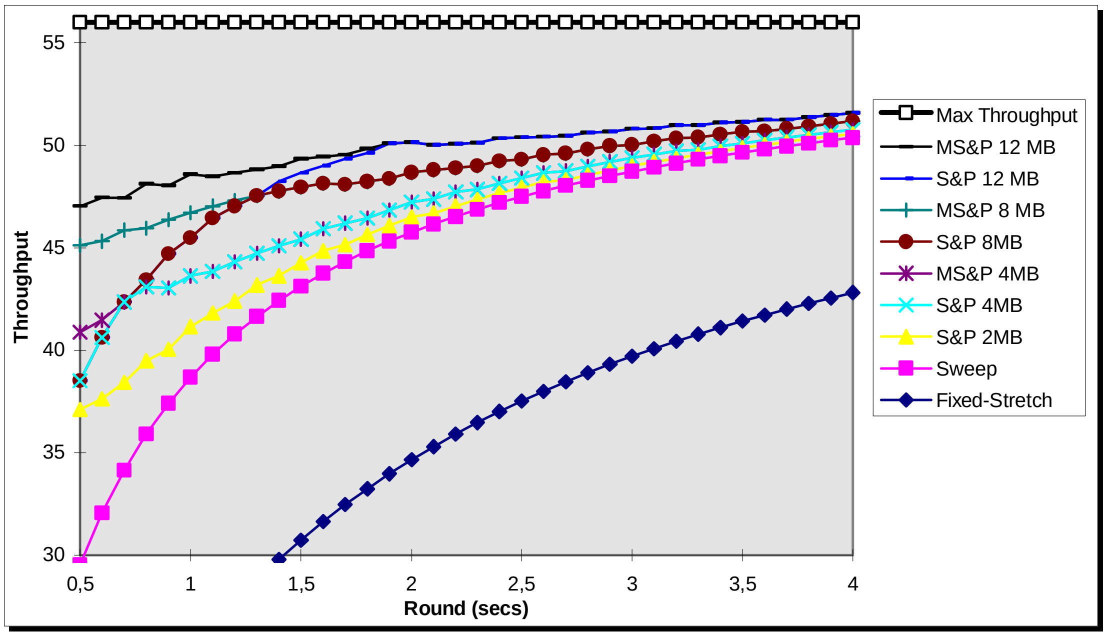

Figure 10: Throughput versus round duration

It can be seen from figure 10 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.

It is reminded that GPMP achieves high throughput by keeping the round duration short, so it is not included in figure 10. In addition, as the round increases, MS&P curves meet those of S&P for the same cache configurations. This happens as the segment sizes become large and there is not enough cache to support multiple round prefetches. Correspondingly, as the round duration decreases, more segments can be prefetched and so, after a bound, S&P does not fully exploit the disk cache.

In the next figures, graphs of throughput versus memory are constructed for all schemes without memory sharing (figure 11, Fixed-Stretch is not included) and with memory sharing techniques as discussed in the previous section (figure 12). GPMP is plotted on these graphs assuming a **constant** round duration of 200 msec. GPMP is the only algorithm -- among those plotted -- that keeps the round length constant and increases its throughput through further memory requirements. The other algorithms are based on lengthening the round duration to increase their throughput, as it can be seen from the previous graph. We can see that the S&P family algorithms outperform Sweep and Fixed-Stretch. Especially for GPMP, the memory savings at high throughputs over Sweep and Fixed-Stretch are very significant (over 60% for specific values of throughput).

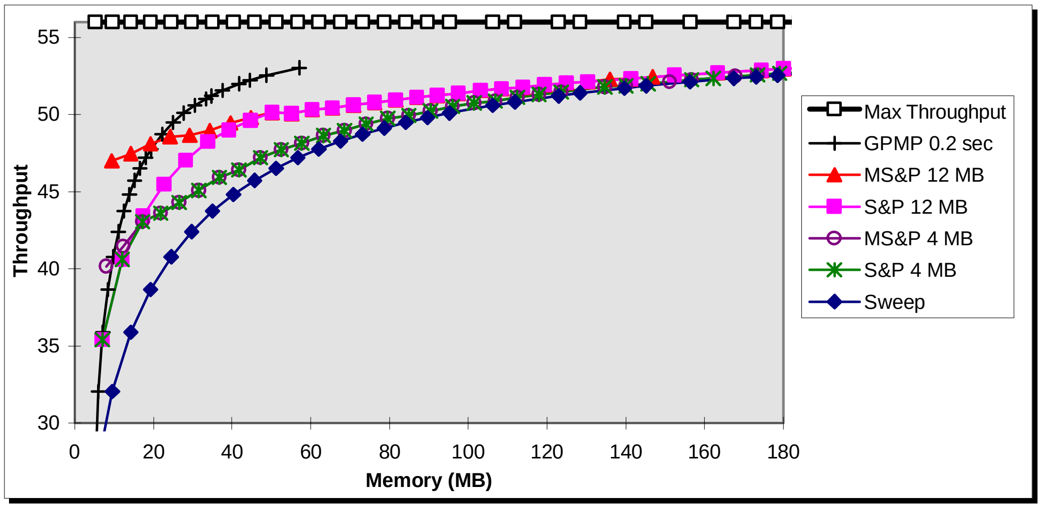

Figure11: Throughput versus Memory

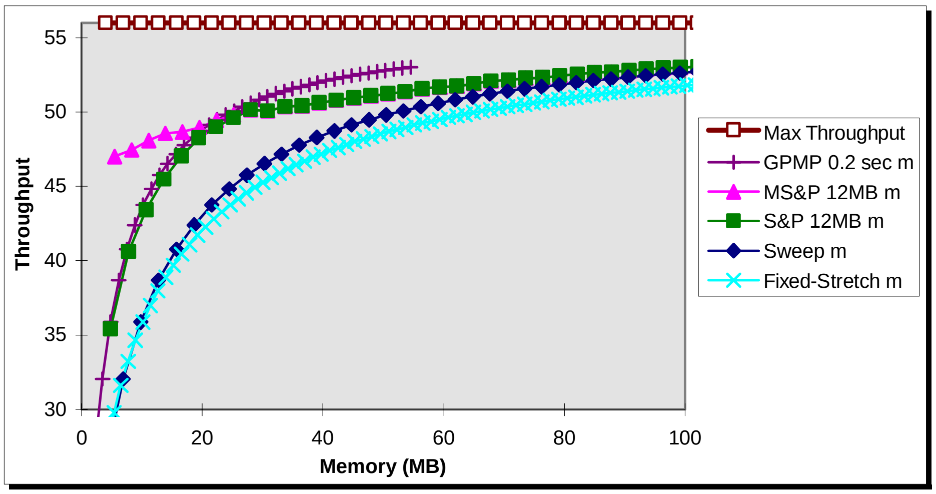

Figure 12: Throughput versus Memory under Memory Sharing

3.6 Interlaced Placement and Prefetching

The cache size that S&P and MS&P need can be further reduced if we interfere in the way the segments are placed onto disk. The data placement policy that is proposed, interlaces the segments of different streams so that different requests can be served at no positioning overhead. The aim of the scheme is to combine prefetching and seek overhead elimination for a certain number of streams and produce a lower disk cache requirement. With an interlaced placement on disk it may be possible during the prefetching round to read segments from different streams, in addition to the next segments of the initial streams. Those segments belong to other streams can be played directly as they were random retrievals, while the rest of the segments originate from traditional prefetching and so they can be kept in cache for the next round. It is clear that if this placement is achievable, it can multiply the benefits gained from both S&P and MS&P. Figure 13 demonstrates this interlaced placement. The same throughput as in Figure 7 is achieved, but this time the disk cache needed is 24 segments instead of 42.

Although it is difficult to predict the demand of requests and their relative timing, it is possible to transfer the problem of an optimal placement to the writing phase, through delaying some requests, leaving empty spaces and monitoring the use of the media server. Even if an optimal placement is not achieved this way, we can interfere in tertiary memory and have the most popular streams already interlaced in tapes. In the future we plan to study interlaced placement in more detail.

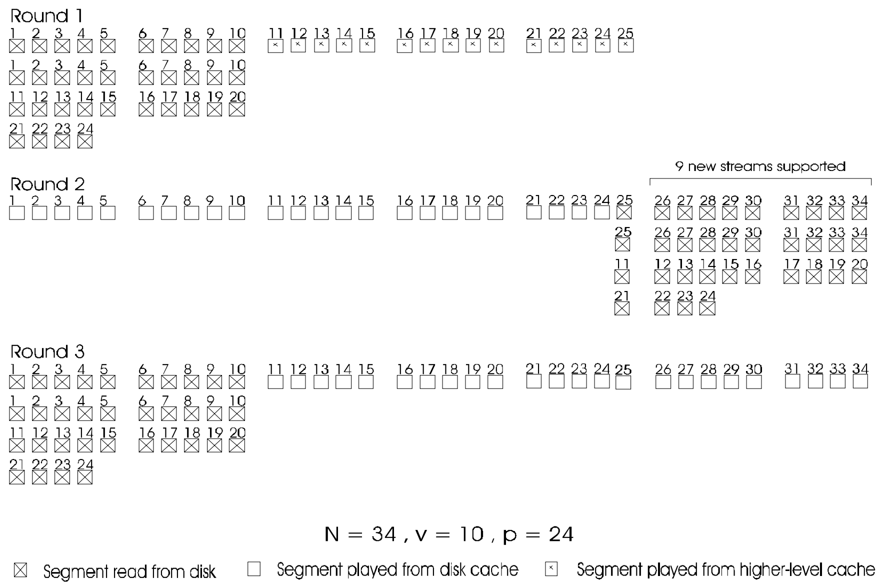

Figure 13: Interlaced Placement and Prefetching

Chapter 4

Experimentation

Further evaluation and verification of the algorithms analyzed in the previous chapter came through simulation of part of a hypothetical Continuous Media server. This simulated part includes the retrieval of continuous media streams from secondary storage into RAM through a system bus as well as an already generated workload that represents a typical use of the Media server. The latter is taken as a substitute of the function of the CM server parts' – not simulated here -- that lie above and below of the disks-to-RAM data flow level in the server's hierarchy (please refer to figure 1 in chapter 1). Towards the purposes of an accurate simulation the DiskSim Simulation Environment [40] has been used; minor changes have been made in the source code so that the algorithms proposed in this study could be supported.

4.1 The DiskSim Simulation Environment

DiskSim is an efficient, accurate and highly-configurable disk system simulator developed at the University of Michigan [40, 41]. It includes modules that simulate disks, intermediate controllers, buses, device drivers, request schedulers, disk block caches, and disk array data organizations. In particular, the disk drive module simulates modern disk drives in great detail while it has been carefully validated against several production disks at high accuracy. Some of the major functions (e.g., request queuing/scheduling, disk block caching) that can be present in several different components (e.g., operating system software, intermediate controllers, disk drives) are implemented as separate modules that are linked into components as desired. DiskSim can be driven by externally-provided I/O request traces or internally-generated synthetic workloads.

However, DiskSim does not address some of the aspects of latest-technology disk drives that concern the management of the on-board cache and are essential for the implementation of the proposed algorithms. These are:

(i) The capability of the disk controller to dynamically interfere in the way caching is done as to allow extremely aggressive read-ahead and no read-ahead to co-exist in a given schedule. Also, the capability of the controller to enable/disable caching of requested data so that replacements in cache can occur in various ways.

(ii) The adaptive segmentation of the disk cache as described in chapter 2, section 2.1.

(iii) The capability of the system to pass hints to the disk controller as for the nature of the workload or to dynamically impose caching policies.

The above matters have been addressed in this study by slightly modifying the source code of DiskSim as well as by interfering in the parameters of the parameter file – the input of DiskSim that specifies the simulated system. The latter is done in such way that certain computed values along with some assumptions from a macroscopic point of view can result in the same effects the actually implemented modules would have.

More specifically, a new trace format created as to include hints provided from a higher level of the CM server, that concern the prefetching strategy. Towards the implementation of S&P algorithms, an integer value has been added in the new format for every request; this number will specify the number of segments to be prefetched each time (can be zero), according to each algorithm. When it comes to the controller to service a request, the code has been changed as to permit dynamically defining the size of prefetching according to the input. In addition, many large cache lines are defined and combined with a FCFS replacement policy, so that the ordering and the values of the requests within the trace file can result in the same effects of an adaptively segmented cache which can be dynamically tuned as for the replacement policy.

4.2 Experimental Testbed and Simulated Disk

The same assumptions as in the theoretical analyses hold: a single disk, constant display rate of 2 Mbps, contiguous placement of streams onto disk, double buffering scheme in main memory. Also the same restrictions hold too: the service of the requests is done in rounds and uninterrupted playback of streams is ensured – that is all requests belonging in a round should be satisfied within the time limits of the round.

A video library is considered that holds videos of 90-120 min of length, ordered according their popularity. Our trace generator considers newly arrived videos at a Poisson distribution (or, alternatively, in a constantly increasing rate) and picks their time lengths from the video library according to Zipv's law. Partitioning in disk is assumed so that large portions of the video are placed onto disk successively. The S&P algorithms are implemented in the trace generator as previously discussed. Furthermore, scheme Sweep is also evaluated.

Capacity (MB)	10,886
Heads/Recording Surfaces	18
Cylinders	6720
Sectors per Track	140 to 220
Sector Size (bytes)	512
RPM	10000
Seek Function (msec)	$seek(d) = \begin{cases} 1.867\times10^{-3} + 1.315\times10^{-4}\sqrt{d}\,; d < 1344 \\ 3.8635\times10^{-3} + 2.1*10^{-6}\,d\,; d \geq 1344 \end{cases}$
Average Seek Time (msec)	8
Average Transfer Rate (MB/sec)	14
System Bus Transfer Rate (MB/sec)	40 (Ultra SCSI)

Table 1 : Simulated disk parameters

As the disk models delivered along with the software release of DiskSim [40] are out-of-date, another disk model was assumed. The values that determine the seek function are the same as in [52]. The values of various delays and overheads have been set to be representative of modern disk drives. While our model is not a real word product, its characteristics are very close to those of existing ones (please see table 1). Various cache sizes are considered during the simulation.

4.3 Experimental Results

Scheme Sweep, S&P, and MS&P were tested for their maximum achievable throughput –- at the constraint of uninterrupted playback -- for disk cache sizes of 2, 4, 8, and 12 MB and for round lengths of 0.5, 1, 1.5, 2, 3, and 4 sec. The results are presented in the following graphs. In figure 14 a curve of throughput versus round length is constructed while in figure 15 a graph of throughput versus memory requirements is shown. For the latter case, a double memory buffer scheme is assumed (as discussed in chapter 3). In the second graph, scheme GPMP is introduced. It is reminded that GPMP behaves identically with Sweep with a round duration equal to an epoch. Thus, a fixed round of 200 msec is assumed -- same as in theoretical analysis -- and no further simulation results are needed (only memory computations). Finally, Fixed-Stretch has been left out for a future implementation, since the theoretical results were not very encouraging.

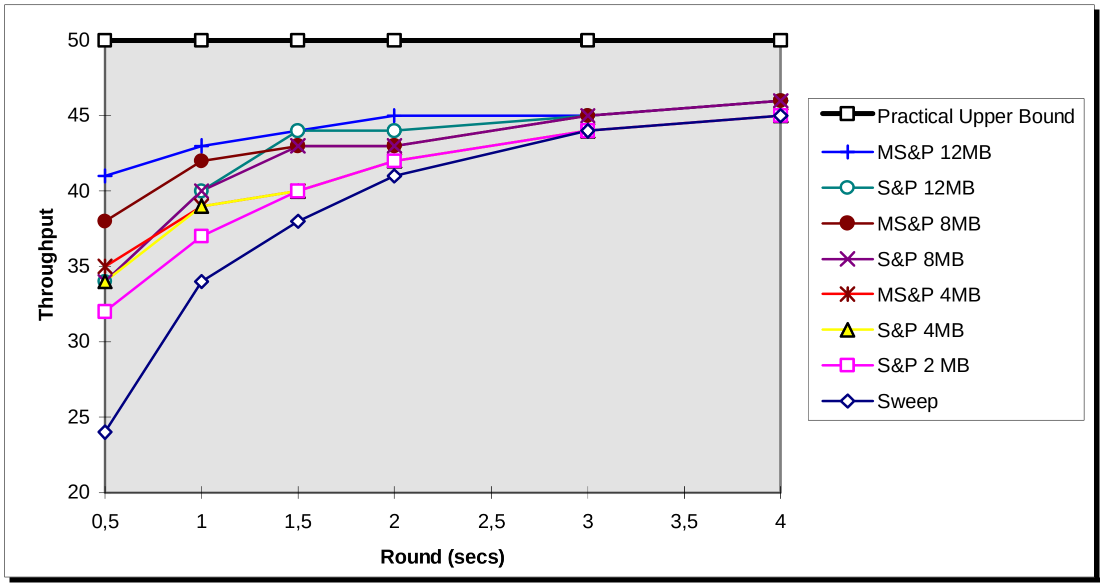

Figure 14: Experimental Throughput versus Round

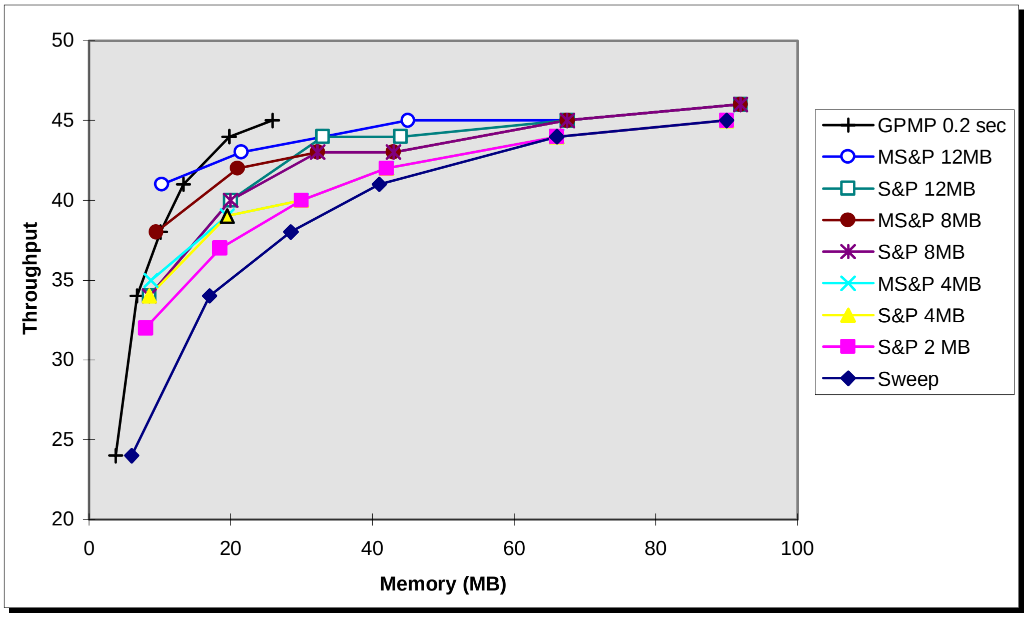

Figure 15: Experimental Throughput versus Memory

4.4 Discussion

By comparing the graphs of the previous section to those derived theoretically in section 3.5.3, we can verify the expected performance improvements of S&P algorithms over Sweep. The only difference between the experimental and the theoretical results is a down shift of all experimental curves by approximately 5 streams (units of Throughput). However, the correlation of the algorithms remains as theoretically defined. The shift occurs as the actual disk transfer rate for the kind of the workload studied is not equal to the expected one. The latter is justified as a large read request has to suffer additional delays due to head switches when crossing track and cylinder boundaries; that is, the transfer rate of a large segment is no longer equal to the rate with which a single sector is read. In our disk model, the expected transfer rate is 14 MB/sec (as the one used in theoretical evaluation) while the measured one is 12.6 MB/sec. If we set the latter value in the evaluation of chapter 3, then, approximately, we derive the same graphs (not shown here) as the experimental ones.

Apart from the verification of the new algorithms, the simulator was used to study the functionality and efficiency of the disk cache. Observations on steps followed by the simulated system show that a large on-board buffer is not exploited by Sweep. (To our knowledge, disk manufacturers have not published any prefetching technique designed for continuous media retrievals that takes also into account the concept of a *Round*). Furthermore, interfering in the prefetching strategy actually leads to the expected benefits. More specifically, it has been observed that requests found in cache (read hits) can be transferred to the host in parallel to disk-to-buffer transfers of other requests retrieved from the disk surface. For example, during the service of 3 requests found in the cache as a result of a 3-depth MS&P prefetching, the disk-to-buffer transfer of the next 4 segments (1 random retrieval and 3 prefetches) goes in parallel to the buffer-to-host transfer of the initial requests' data.

The central conclusions are that:

- Our hierarchical prefetching strategies introduce significant higher disk maximum throughput compared to the traditional strategy (Sweep) for retrieving CM blocks. For example, for round length equal to 1 sec and cache size equal to 8 MB the number of streams that can be supported by MS&P is almost 30% greater than Sweep (see Figure 14).

- In addition, figure 15 clearly shows that despite the additional disk cache size required by our prefetching strategies, the same investment in total RAM 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. For example, 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.

Chapter 5

Conclusions

5.1 Summary of Contributions

Current and future technologies to be used in Continuous Media servers architectures, especially in modern disk drives, have been carefully studied and reported.

Towards the improvement of cost/performance of a CM server, this study contributes a number of techniques based on the idea of prefetching amounts of data equal to the size of the individual requests.

- Today’s state of the art as it is expressed through the introduction of cache hierarchies into Media servers and employment of large on-board cache buffers onto modern disks,
- Promising innovations such as NASDs,
- Advances in communications and data channels, and
- Evolution, beyond any predictions in microprocessor technologies

create a strong foundation on which new, aggressive prefetching strategies can be studied, proposed and long-term evaluated.

In this work,

- Prefetching into the disk cache the next segment of several streams resulted into the S&P scheme; however there exists an upper bound on the number of streams that prefetching can be performed..
- The Gradual Prefetching technique assured short start-up latencies for the S&P algorithm.
- The maximum throughput upper bound of S&P has been overcome by aggressively prefetching more CM data blocks which will be needed in future rounds; the MS&P algorithm addressed the issues around multi-round prefetching.
- Combining extremely aggressive multi-round prefetching and traditional ways of increasing throughput -- round duration lengthening -- the GPMP algorithm introduced a round breakdown scheme and an efficient way of organizing the required memory which lead to even higher performance.

The performance of above schemes was studied first through analysis. Detailed graphs outlined the expected gains. The validation of the theoretic results came through the implementation of the proposed algorithms into a drive accurate simulation model. Through simulation, the underlying technological premises have proved to be feasible.

The techniques proposed in this research:

- Introduced significant throughput improvements (up to 60-70% when compared to Sweep at a 500 msec round).
- Showed how to improve the cost/performance of a CM server by exploiting existing hierarchical caches.
    - Achieved low start-up latency and eliminated any associated tradeoffs.
    - Pointed out new research directions.
- Proved that the improvement in the maximum throughput requires a smaller investment in RAM cache memories, despite our schemes' heavy reliance on host and disk caches.

5.2 Future Work

The proposed schemes should be studied under coarsed-grained striping techniques, since the existing striping techniques are completely oblivious to the existence of the multi-level caches and their exploitation with our prefetching techniques, which can significantly improve a disk array's performance.

The GPMP algorithm should be further studied and analyzed. An optimal round breakdown should be computed. A common case in GPMP is a value of ***u*** that results into a non integer value for the number of groups. While the gains are not affected, this case has to be studied as for the delivery scheme within a round. Analytical formulas should be derived.

Finally, the scheduling policy within a round has to be studied as to service new requests at minimal start-up latencies.

The data placement policy discussed in Interlaced Placement should be further inspected. Algorithms that take into account the workload of the media server and affect the writing of the streams from tertiary memory onto disk will be implemented. Ways of alternatively storing continuous media data in tape libraries may be proposed. A thorough evaluation should indicate the further benefits.

The final step should be a study over a mixed workload (both continuous and discrete requests) based on real world parameters. The management of the cache should be well defined before mixed requests can be efficiently serviced.

Bibliography

[1] E.Chang and H. Garcia-Molina. Effective memory use in a media server (extended version). Stanford Technical Report SIDL-WP-1996-0050 URL: http://www-diglib.stanford.edu

[2] D. Makaroff and R. Ng. Schemes for implementing buffer sharing in continuous-media systems. Information Systems, 20(6):445-464, 1995.

[3] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming raid - a disk array management system for video files. First ACM Conference on Multimedia, August 1993.

[4] P.J. Denning. Effects of scheduling on file memory operations. Proc. AFIPS Conf., April 1967, pp. 9-21.

[5] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. ACM Transactions on Computer Systems, Feb. 1987, pp.77-92.

[6] Coffman, Jr and M. Hofri. Queueing Models of Secondary Storage Devices. Stochastic Analysis of Computer and Communication Systems. Ed. Hideaki Takagi, North-Holland, 1990.

[7] 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.

[8] D. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Tech. Report, HPL-CSP-91-7, Feb 1991.

[9] M. Seltzer, P. Chen, and J. Ousterhout. Disk Scheduling revisited. Proc. 1990 USENIX Technical Conf, pp.313-323.

[10] Quantum Corp., Storage Basics, ORCA. www.quantum.com/src/storage/_basis.

[11] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. J. ACM, Jan 1973, vol. 20, no. 1, pp. 46-61.

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

[13] S. Berson, S. Ghandeharizadeh, R.R. Muntz and X. Ju. Staggered Striping in Multimedia Information Systems. Proc. of the Intern. Conf. on Management of Data (SIGMOD), Minneapolis, Minnesota, pp. 79-90, 1994.

[14] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. Multimedia Storage Servers: A Tutorial. IEEE Computer, Vol.28, no.5, May 1995, pp.40-49.

[15] S. Ghandeharizadeh, S.H. Kim and C. Shahabi. On Disk Scheduling and Data Placement for Video Servers. ACM Multimedia Systems, 1996.

[16] 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.

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

[18] T.H. Lin and W. Tarng. Scheduling periodic and aperiodic tasks in hard real time computing systems. Proc. Intern. ACM SIGMETRICS, Conf., 1991.

[19] G. Nerjes, P. Muth, G. Weikum. Stochastic Performance Guarantees for Mixed Workloads in a Multimedia Information System. Proc. IEEE Intern. RIDE Workshop, April 1997.

[20] G. Nerjes, Y. Robogianakis, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Scheduling strategies for mixed workloads in multimedia information servers. Proc. IEEE Intern. RIDE Workshop, Feb. 1998

[21] G. Nerjes, Y. Robogianakis, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. On mixed-Workload Multimedia Storage Servers with Guaranteed Performance and Service Quality. 3rd Intern. Workshop on Multimedia Information Systems, Sept. 1997.

[22] Y. Robogianakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Disk Scheduling for Mixed-Media Workloads in Multimedia Servers. 6th ACM International Multimedia Conference, ACM Multimedia 98, September 1998, (to appear).

[23] R. Karedla, J.S. Love, and B.G. Wheery, Cache Strategies to Improve Disk System Performance. IEEE Computer Magazine, March 1994, 38-46.

[24] S. Ghandezarizadeh and C. Shahabi, On Multimedia Repositories, Personal Computer and Hierarchical Storage Systems, Proceedings ACM Multimedia Conference, 1994.

[25] P. Triantafillou and T. Papadakis, On-demand Data Elevation in Hierarchical Multimedia Storage Servers, Proceedings 23rd Int. Conf. on Very Large Databases (VLDB), August 97, 226-235.

[26] MIDS, Austin, Texas. http://www.mids.org

[27] Ericsson. http://www.ericsson.com/systems/gsm/future.html

[28] Le Gall, D. J. «MPEG: A Video Compression Standard for Multimedia Applications.» Comm. ACM, Vol. 34-4, 1991, pp. 46-58.

[29] Seagate http://www.seagate.com/disc/cheetah/cheetah.shtml

[30] http://www.webproforum.com/nortel2/topic02.html

[31] CCITT Recommendation 1.113. «Vocabulary of Terms for Broadband Aspects of ISDN.» 1991.

[32] CCITT Recommendation 1.211. «B-ISDN Service Aspects.» 1990.

[33] E. A. Fox. The Coming Revolution in Interactive Digital Video. Communications of the ACM, 7:794 - 801, July 1989.

[34] W. D. Sincoskie. System Architecture for a Large Scale Video On Demand Service. Computer Networks and ISDN System, 22:155 - 162, 1992.

[35] P. V. Rangan, H. M. Vin, and S. Ramanathan. Designing an On-Demand Multimedia Service. IEEE Communication Magazine, 30:56 - 65, July 1992.

[36] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In Second Annual ACM Multimedia Conference and Exposition, San Francisco, CA, 1994.

[37] Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling, IEEE Computer March 1994, 17-28.

[38] Gregory R. Ganger, Bruce L. Worthinghton, and Yale N. Patt. Scheduling for Modern Disk Drives and Non-Random Workloads, March 1, 1994, The University of Michigan.

[39] Intel Corp. http://www.intel.com

[40] DiskSim1.0 http://www.ece.cmu.edu/~ganger/

[41] Gregory R. Ganger, Bruce L. Worthinghton, and Yale N. Patt. The DiskSim Simulation Environment Version 1.0 Reference Manual, CSE-TR-358-98, The University of Michigan.

[42] G. Nerjes, Y. Robogianakis, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Incremental Scheduling of Mixed Workloads in Multimedia Information Servers.