Minimizing Latency and Jitter for Large-Scale Multimedia Repositories through Prefix Caching

SUNIL PRABHAKAR

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
U.S.A.

sunil@cs.purdue.edu

RAHUL CHARI

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
U.S.A.

rahul@cs.purdue.edu

Multimedia data poses challenges for efficient storage and retrieval due to its large size and playback timing requirements. For applications that store very large volumes of multimedia data, hierarchical storage offers a scalable and economical alternative to store data on magnetic disks. In a hierarchical storage architecture data is stored on a tape or optical disk based tertiary storage layer with the secondary storage disks serving as a cache or buffer. Due to the need for swapping media on drives, retrieving multimedia data from tertiary storage can potentially result in large delays before playback (startup latency) begins as well as during playback (jitter). In this paper we address the important problem of reducing startup latency and jitter for very large multimedia repositories. We propose that secondary storage should not be used as a cache in the traditional manner – instead, most of the secondary storage should be used to permanently store partial objects. Furthermore, replication is employed at the tertiary storage level to avoid expensive media switching. In particular, we show that by saving the initial segments of documents permanently on secondary storage, and replicating on tertiary storage, startup latency can be significantly reduced. Since we are effectively reducing the amount of secondary storage available for buffering the data from tertiary storage, an increase in jitter may be expected. However, our results show that the technique also reduces jitter, in contrast to the expected behavior. Our technique exploits the pattern of data access. Advance knowledge of the access pattern is helpful, but not essential. Lack of this information or changes in access patterns are handled through adaptive techniques. Our study addresses both single- and multiple-user scenarios. Our results show that startup latency can be reduced by as much as 75% and jitter practically eliminated through the use of these techniques.

Keywords: Prefix Caching; Tertiary Storage; Multimedia Placement.

1. Introduction

Multimedia data poses challenges for efficient storage and retrieval due to its large size and playback timing constraints. Consequently the problem of multimedia storage has received significant attention from the research community. Due to the need for efficient retrieval, the research has focussed chiefly on magnetic disk technology. The falling cost per megabyte for disk storage has made it possible to store data for many applications on disk. However, for applications that need to store very large amounts of data, storing only on magnetic disks is still too expensive. Examples of such applications include telemedicine, online multimedia manuals, and television broadcast for which the storage requirements can easily exceed several tens of terabytes. For example, a small sample of video and image data from the MedInstitute in Indianapolis constitutes 200GB of data. The total requirements for storage are well over ten terabytes and will continue to grow as more patient data is collected in digital format. Even though the cost of disk storage has dropped significantly, and will likely continue to do so in the future, the storage requirements are also growing at a similar pace. The desire to store in digital format very high quality medical multimedia data for all patients, and automatically captured high quality images of the universe [25] are examples of applications with ever-growing storage needs. Such large volumes of data are typically stored on tertiary storage such as automated tape libraries [26] or CD/DVD jukeboxes [18]. Even with the availability of a large amount of disk space, the use of tertiary storage allows cheap scalability to even larger data volumes.

Tertiary storage offers much cheaper storage than magnetic disks. This is achieved through a large number of cheap media sharing a small number of expensive drives. On the flip side, data access on tertiary storage can suffer from large latencies if media need to be swapped on drives or tape needs to be rewound. Typical access times for magnetic disks are on the order of milliseconds whereas the access time for magnetic tapes can vary from a few milliseconds to a minute or more. If the tape holding the requested data is not loaded on a drive, it is necessary to rewind a currently loaded tape, eject it, place it back on the rack, pick up the requested tape, load it, and seek to the appropriate location before data transfer can begin. These operations are very slow due to the mechanical motion required. It should be noted that the streaming rate for tertiary storage is comparable to that of disks, however the latency for random access can be very much higher.

Large delays in accessing data can result in high *startup latency* (time that elapses between the submission of the request and the beginning of the retrieval) or *jitter* (delays in data after playback has begun). In order to reduce startup latency and jitter, careful management of storage is essential. This is especially important when multiple users access the repository concurrently. In this paper we present novel techniques for the efficient management of large volumes of multimedia documents on secondary and tertiary storage. Due to the large amount of data to be stored, data primarily resides on tertiary storage. The disks that make up the secondary storage layer typically serve as a cache. Data that is retrieved from tertiary storage is temporarily stored on disk. A replacement policy such as Least Recently Used (LRU) or Least Frequently Used (LFU) is typically used to make room for the new data.

We propose that secondary storage should not be used as a cache in the traditional sense – instead, most of the secondary storage should be used to permanently save parts of multimedia documents or objects. At the tertiary storage level, we propose the use of replication to avoid expensive media switching. In particular, we show that by saving the initial segments of documents permanently on secondary storage, and replicating on tertiary storage, the startup latency can be significantly reduced. Since we are effectively reducing the amount of secondary storage available for buffering the data from tertiary storage, an increase in jitter may be expected. However, we show that our technique reduces jitter in contrast to the expected behavior. Although advance knowledge of the access pattern is helpful, it is not essential for our techniques. We show how the observed access patterns can be used to determine and tune the placement.

The rest of the paper is organized as follows. In Section 2 we summarize the related work. Section 3 presents our new approaches for disk caching and tertiary placement. A description of the system model is presented in Section 4. Section 5 gives details experimental results and Section 6 concludes the paper.

2. Related Work

Although the issue of storing multimedia data on tertiary storage has been addressed by several researchers, the problem of reducing startup latency and jitter in a multi-user setting has not been studied. For example, Chervenak et al. [3,2] have investigated the use of tertiary storage in video-on-demand applications. A cache replacement technique for managing secondary storage buffers when multimedia objects are stored on tertiary storage has been developed by Ghandeharizadeh et al [9]. The study is limited to a single-user, single-disk personal computer system. In Section 5 we show that their scheme is not effective in a multi-user, multi-disk system – giving poorer performance than a simple LFU cache replacement scheme. The use of a pipelining mechanism that avoids the need for complete materialization of an object on disk before initializing playback is presented in [8]. The basic idea is to divide an object into multiple slices and overlap the retrieval of one slice with the playback of the previous slice. This reduces latency delays during playback but does not reduce the startup delay. This technique can be applied orthogonally to our technique to reduce jitter (note that in our experiments we find that jitter is negligible with our replication scheme). In order to mask network latency and loss, prefix caching of the initial segments of multimedia streams at proxy servers has been proposed [23]. The study addresses network issues such as workload smoothing through caching for multimedia data. The problems of latency and jitter for retrieval of data at the server are not addressed.

Storing video on hierarchical storage has also been studied in [28,29]. The study addresses I/O bandwidth issues at the various levels of the storage hierarchy. The problems of high startup latency for access to tertiary storage and jitter are not addressed. Scheduling schemes for tertiary storage libraries are discussed in [7,11,17,20] – any of these techniques can be applied in conjunction with our research to further improve performance. In [14] a prefetching algorithm based upon Markov-chain prediction of access is developed. Placement schemes for data on tertiary storage libraries have been proposed based upon independent document access probabilities and no replication [4,27]. Optimal arrangement of cartridges and file-partitioning schemes for carousel-type systems are investigated in [24]. Placement schemes for data on optical disks are developed in [6]. Both these studies do not address the issues of multimedia data. We show that use of replication can significantly reduce the need for expensive switching of media on tertiary storage resulting in significant improvements. The cost of replication on tertiary storage is minimal. Models of tape systems and tertiary storage system parameters can be found in [10,13].

3. Hot Objects and Prefix Caching

In this study we address the problem of multimedia storage not only for individual objects but also for multimedia documents that are composed by sharing a collection of objects. In this section we first explain the nature of multimedia documents. This is followed by a description of the proposed prefix caching, and tertiary placement schemes. Finally, a discussion of adaptive placement is presented.

3.1. *Multimedia Documents*

An example of a multimedia document is an online technical manual consisting of images, video, and audio clips. Similarly, a news report consisting of a sequence of several stories with clips is another example. A document specifies the layout of the multimedia objects as well as the timing relationships between them. For example a video segment is played after a previous video or animation is completed. Several approaches for describing multimedia documents have been developed including graph models, Petri-Net models, and object-oriented models [1,15]. An example of a Petri-Net description is shown in Figure 1. The document begins with the display of video1 followed by the simultaneous display of video2 and audio2, followed by video3. The objects that make up documents can be shared among multiple documents. The document may not be stored as a single object, rather it can be composed dynamically from its constituent objects at the time that it is to be retrieved. In order to playback a document, the physical objects that make up the document need to be retrieved in the order that they appear in the document. For ease of exposition, we will present the discussion in terms of documents in the remainder of the paper. However it should be noted that the ideas discussed are equally applicable to repositories that do not have the notion of documents.

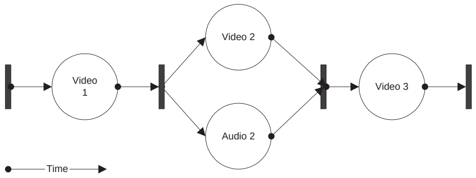

Fig. 1. Example of a Document

Information about the access patterns for multimedia data is a very important input for efficient storage and retrieval of data. Popularity of documents can be captured simply by the probability of access. In addition to direct access to documents (such as by identifying the document directly), users may access documents based upon links from other documents (e.g HTML documents with links to other documents, or hyperlinks between manual pages). Such access is also very common in a browsing scenario whereby users simply follow links of interest. A user would typically begin by accessing a document and then possibly following some number of interesting links. If none of the links in the document are interesting, the user may access a document not connected by links from the current document.

A *Browsing Graph* (BG) [16] can be used to capture such access patterns. The browsing graph consists of labeled nodes and labeled edges. Each node represents a document and the label of the node gives the probability that the node will be accessed independent of the previously visited document. A directed edge between two nodes represents a link from one document to the other and the edge label gives the probability that the edge would be followed. The sum of the probability of all edges going out of a document is not necessarily 1.0, since it is possible that none of the edges will be followed. This model is similar to that used by the Google search engine for assigning weights to documents in the world-wide-web [12].

3.2. *Hot Prefixes and Disk Caching*

Due to the large volume, data resides primarily on tertiary storage. Typically, the disks are used as a cache to temporarily hold data after it has been retrieved from tertiary storage. The disks also act as a buffer for holding data that is to be played later. When the disk cache is full, documents need to be replaced in order to make room for newly requested ones. A document replacement policy such as LRU or LFU can be used to choose which documents to replace. These policies however, are not well suited for multimedia documents. In [9] a cache replacement policy is proposed for caching continuous media data on secondary storage. Instead of replacing entire objects, the tail ends of objects are replaced from the disk cache when space is needed.

We propose an alternative use of the secondary storage. The total disk space is divided into two sets – the *HOT CACHE* and the *BUFFER*. The buffer is used as above to temporarily store data that has been retrieved from tertiary storage. A replacement policy such as LFU is used to manage the buffer. The hot cache is used to permanently hold a special subset of objects: those having a high temperature or *HOT OBJECTS*. In the context of documents, an object refers to each multimedia component that makes up the document, e.g. a video or audio clip. The ideas can be easily applied even if no documents are defined on the objects. For large multimedia objects, only a subset (prefix) of the object needs to be stored on disk. The entire object is stored on tape. Thus only the prefixes of hot objects would be stored in the *HOT CACHE*.

The intuition behind permanently saving hot objects in disk is to mask the high access latency of tertiary storage. A request for a document can suffer a large startup latency if the document is not available on disk. Due to the large size of documents, it is not possible to save most documents on disk. The high startup latency can be masked by having only a small initial portion of the document stored on disk. When the document is requested, playback can begin immediately from disk with very little delay. Concurrently, the document is retrieved from tertiary storage. The playback of the portion of the document saved on disk overlaps with the access latency before the requested document can be read from tertiary storage.

The "heat" of an object is determined using prior information about the access of the objects. This could simply be the observed frequency of access of each document. Given the probability of access of each document, we can compute the heat of an object as the sum of the access probabilities of all documents that contain the object. However, for the purposes of hot caching, we only want to save on disk those objects that occur early in the document. Therefore an object's heat is calculated as the sum of access probabilities of only those documents in which it occurs early. An object is considered to occur early in a document if it lies within the initial segment of the document. The initial segment, or *DELTA*, can be defined as a fixed amount of time, or as a fixed fraction of the document's total playing time. Delta is a parameter that can modeled to suit a system based on its resources. In theory it is possible to use a different value of *DELTA* for different objects of classes of objects. However, the main purpose of caching the *DELTA* prefix is to mask the latency of tertiary storage access. This latency is dominated by the exchange of media and seek times. Hence it is likely to be relatively constant independent of the nature of the data items or the workload. For this reason, we propose the use of a constant value of *DELTA* governed by the nature of the tertiary storage system and disks. In Section 5 we consider the choice of *DELTA* as a fraction of the length of an object (i.e. the length of the prefixes for objects are chosen to be proportional to their entire length). From the results we see that a single choice of *DELTA* gives similar results to the variable choice alternative. For the case of individual multimedia objects with no notion of documents, only the initial *DELTA* segment of the object is saved on disk. The "heat" of an object is simply the cumulative access to the object.

In the proposed scheme, the heat of each object is calculated as explained above.

The hot cache is then filled with prefixes of objects in the order of their heat, beginning with the hottest. An important point is that objects that are shared by several documents are saved only once in the hot cache. The fraction of disk storage reserved for hot objects is denoted by B. The remainder of the storage is used as a buffer between secondary and tertiary storage. Any of the traditional cache management schemes can be used to manage this buffer.

3.3. *Tertiary storage placement*

Tertiary storage is characterized by cheap storage with high access latency. The goal of placement on tertiary storage is to reduce latency. The major component of latency is the time for switching media on drives. In [27] it is shown that a placement whereby the objects are placed sequentially in decreasing order of their access probabilities is optimal. This result, however, is based upon the assumption that objects are accessed independently. This assumption is not true in practice. The access is based upon documents, not independent objects. The popularity of an object is determined by the access to all documents in which the object is contained. Thus it is possible if we follow the placement of [27], the objects for documents get distributed among multiple media resulting in extremely poor performance due to multiple switches.

We avoid this problem by ensuring that the access of a document incurs at most a single media switch. This is achieved by replication of objects. Instead of saving a single copy of each object on tertiary storage, we replicate objects so that a complete document is stored on a single medium. Thus each object is replicated as many times as the number of documents it occurs in. Replication on tertiary storage has a low overhead because storage is cheap. Note that on secondary storage, there is no replication of objects. The entire set of objects needed for a document can now be found placed together on a single medium. Of course, multiple documents can be stored on the same medium. In fact, we use the algorithm of [27] to determine which documents to place on which media using the access probability of documents. Documents are placed in decreasing order of their access probabilities.

3.4. *Adaptive Placement*

A key component of the proposed storage management schemes is knowledge of the access pattern. Although it is useful to know this a priori, it is not critical to the success of the proposed approach. Such information can easily be gathered from the system by keeping track of document requests. Based upon the observed access pattern, the choice of hot objects can be altered accordingly. In Section 5 we show the effectiveness of this adaptive placement in response to changes in the access pattern. In the complete absence of access information, the placement can begin with an initial guess for the hot objects followed by progressive refinement as user requests are serviced.

4. System Model

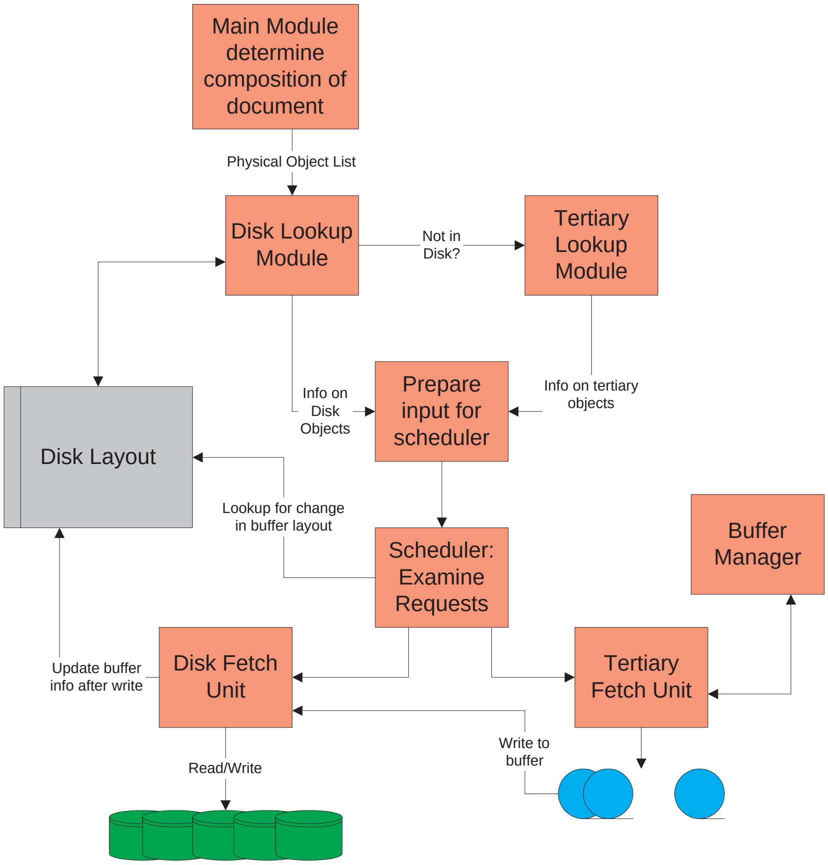

Fig. 2. Block Diagram of System

The model of our system is shown in Figure 2. The functionality of each module and its relationship with the other modules in the system is explained below. Every request for a document is decomposed into requests for the component objects. The *Disk Lookup* module performs a lookup of all the objects currently residing on disk to determine if any of the requested physical objects are presently in secondary storage. This includes objects in the hot cache as well as those in the buffers. Note that the disk buffer handles objects not documents. Thus it is possible that some objects are retained in the buffer while other objects from the same document are replaced. Based on the results of the disk lookup, all the objects not found on disk are searched for in tertiary storage. The *Tertiary Lookup* module determines the location of the requested objects on tape. The information about the location of the constituent objects on disk and tertiary storage is passed on to the *Scheduler*.

The scheduler orders the requests for fetching the objects into main memory in the order of their occurrence in the document. This is done taking into consideration the buffer space available. Each user has an allocated buffer space in main memory to hold the requested objects before they are sent on the network. As objects are fetched into main memory, the buffer space allotted to the corresponding user decreases. Unavailability of buffer space results in the request being kept on hold until an object from the same document is played and the space occupied by that object is released. The scheduler also takes into account the time at which an object is required during the playback using a delay estimation module. The delay estimation module takes into consideration the current status of the system resources and produces an estimate of the time required to fetch the data from disc and tapes.

The request is sent to a drive chosen on the basis of the request queue on each drive. If the required tape is already loaded and is currently in use, the estimation module does not factor in the load time but estimates the delay based on the length of the queue for the tape and the size of each waiting request. The playing of the requested document is delayed until the data that is readily available from disk can mask the fetch time of the data from tertiary storage. Delaying the start increases the startup latency but reduces jitter that would be observed if there is a break between the consumption of data on disk and the arrival of data from tertiary storage. On completion of the delay estimation, the scheduler sends the requests to the *Disk Fetch* unit and the *Tertiary Fetch* unit.

The *Buffer Manager* keeps track of the data stored on the buffer disks and also the amount of space available to buffer data from tertiary storage. The buffer manager uses an LFU policy for object replacement from the buffers. Once the data is available in main memory it is ready to be sent to the user over the network. The system assumes the availability of a fixed bandwidth network connection out of the server. Based on the playing time of each object and its size, the bandwidth requirement for that object is determined. If sufficient bandwidth is available the object is transmitted. Otherwise, the playback of the document is delayed until sufficient bandwidth is available.

5. Experimental Results

In this section we demonstrate the effectiveness of prefix caching and replication towards reducing startup latency and jitter. The results are based upon a detailed CSIM [22] simulation model of the system described above. The disk specifications for the model are based on the HP 97560 disk drive [21]. The tape library is modeled on the Exabyte EXB-480 tape library configured with Exabyte Mammoth drives [5]. Further details of the tape simulator can be found in [19]. The secondary storage is configured with 20 disks each of capacity 2GB, giving a total of 40GB of disk storage. The division of the disk storage into hot prefix cache and buffer is achieved by dedicating entire disks to either of the two uses. The tertiary storage component is modeled on a robotic tape library with four Exabyte drives. Some of the important parameters for the disks and tape simulation are provided in Table 1. The experiments were conducted on a synthetic collection of 10,000 multimedia objects of average size 100 Megabytes and a playback rate of 8MB/second. The tape library is configured with 1000 tapes each of size 10GB, giving a total of 10TB of tertiary storage. It should be noted that the capacity of each disk is deliberately chosen to be small compared to currently available disk drives. This is done to compensate for the small number of multimedia objects considered in the experiments. Experiments with larger numbers of objects took too long to complete. Therefore the amount of disk or cache capacity was reduced accordingly. In practice, larger disks would be used for caching larger volumes of tertiary-resident data.

Parameter	*Value(s)*	*Meaning*
DISK SIMULATION PARAMETERS		
ROT_SPEED	4002	Rotational speed RPM
SEC_SIZE	512	Size of sector in bytes
SEC_TR	72	No. of sectors per track
CYLINDERS	1962	No. of cylinders
TR_CYL	19	No. of tracks per cylinder
TRKSKEW	8	Track skew in sectors
CYSKEW	18	Cylinder skew in sectors
CNTRL_TIME	1.2	Controller overhead (ms)
CAPACITY	2 GB	Disk storage capacity
TAPE SIMULATION PARAMETERS		
RWD_OVHD	0.0083 seconds	Rewind Overhead
SEEK_OVHD	0.0083 seconds	Seconds
SEEK_SPEED (RWD_SPEED)	103 (103) MB/s	Tape seek (rewind) rate
EJECT_TIME	2 seconds	
LOAD_TIME	4 seconds	Time to load a tape on a drive
PICK_TIME	1 second	Time for robot to grab a tape
PUT_TIME	1 second	Time for robot to drop a tape
MOVE_TIME	1 second	Time for robot to move
XFER_SPEED	3.0 MB/s	Tape transfer speed
NUM_TAPES	1000	Total number of tapes
TAPE_CAP	10 GB	Tape cartridge capacity
Num of Drives	4	

Table 1. Table of Parameters

The set of documents and the access pattern is generated as follows. The number of component objects in each document is chosen from a uniform distribution between 3 and 20. The corresponding number of objects are chosen following the access probability of the objects. Since we are dealing with multimedia objects, the access probability of objects follows a Zipf distribution. The document access probabilities are also assigned following a Zipf distribution. In order to capture the effects of links between documents, we introduce the notion of edges between documents. To determine the edges, the documents are divided into clusters. The number of documents in a cluster is uniformly distributed between 2 and 20. Some (5%) of the documents are considered to be outliers that do not belong to any cluster. For each document, a *death probability*, p_d, is picked uniformly distributed between 0.05 and 0.2. This is the probability that the user does not follow any of the links from this document. Edges to other documents within the cluster are created and assigned probabilities that are uniformly distributed so as to add to 1 - p_d.

It is important to note that although the access pattern is an input to the placement algorithm, it is not crucial that this pattern be accurate. As we have mentioned earlier, if the access pattern is unknown or changes after the placement, the system can adapt to the observed access pattern by adjusting which objects get placed in the hot cache. Experimental evidence to support this claim is presented in Subsection 5.7.

Based upon the structure of the documents, and their access probabilities a placement of data on tertiary and secondary storage is generated. In each experiment, we run multiple concurrent streams of requests, each corresponding to a different user. Each stream begins by requesting a starting document following the access probability for the documents. As soon as this document is retrieved, the user chooses to either follow one of the edges or to pick another document following the document access probabilities. This choice is based upon the edge probabilities and the death probability of the currently accessed document. In each test we first warm up the caches by running 1000 requests. Following this, we run another 1000 requests based upon which we compute the average startup latency or average jitter observed by the requests.

In the following experiments we study the performance of prefix caching and the impact of the following parameters: $DELTA$, number of hot object vs. buffer disks, number of simultaneous users in the system, available network bandwidth, and the access pattern. The performance of the PIRATE cache replacement scheme designed for a single-user, single-disk environment is also presented.

5.1. *Impact of Hot Object Caching*

We begin by studying the effectiveness of the hot object technique in reducing the startup latency. Figure 3 shows the average latency as the number of concurrent users is varied for three different choices of $DELTA$ (the size of the “prefix”). The graph for $DELTA = 0$ represents the performance for no hot object caching where all disks are used as buffers. The other two graphs show the performance with hot

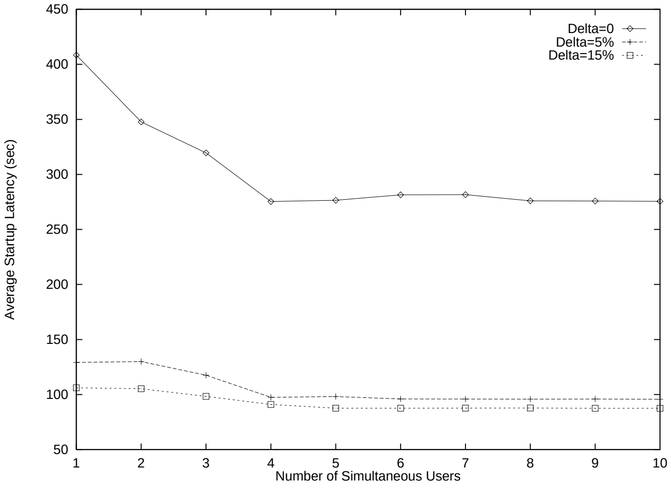

Fig. 3. Startup Latency for Delta as Percentage

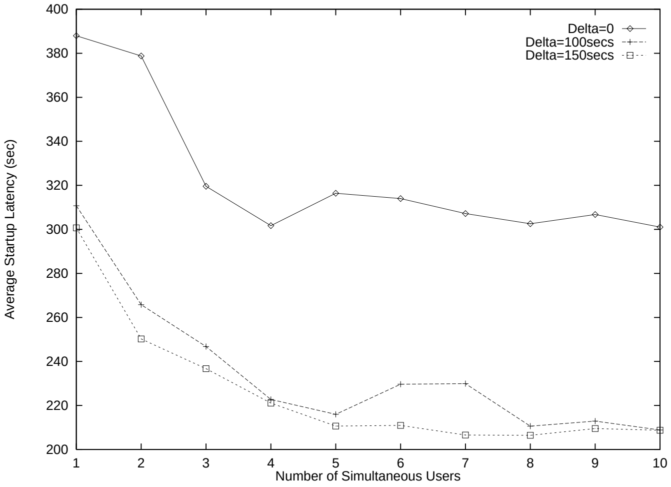

Fig. 4. Startup Latency for Delta as Time

object caching for $DELTA$ equal to 5% and 15% of the total time of each document (i.e. an object is considered to be in the prefix of the document if it occurs within the first 5% or 15% of the document). The number of users was varied from 1 to 10 . As can be seen from the graph, prefix caching considerably reduces the startup delay. The difference in performance between the 5% and 15% values of $DELTA$ is not significant. The number of cache disks was maintained at 8 and the number of buffer disks was 12. The alternative choice of $DELTA$ as a fixed amount of time was also studied. Figure 4 shows the results for $DELTA$ as 100 seconds and 150 seconds. Similar results are seen, except that these values are not as effective as the 5% or 15% choices for $DELTA$. This is easily explained by the fact that with $DELTA = 5\%$, the corresponding average value in seconds is about 250. For the remainder of the experiments, we fix $DELTA$ to be 150 seconds, unless specified otherwise.

5.2. *Impact on Jitter*

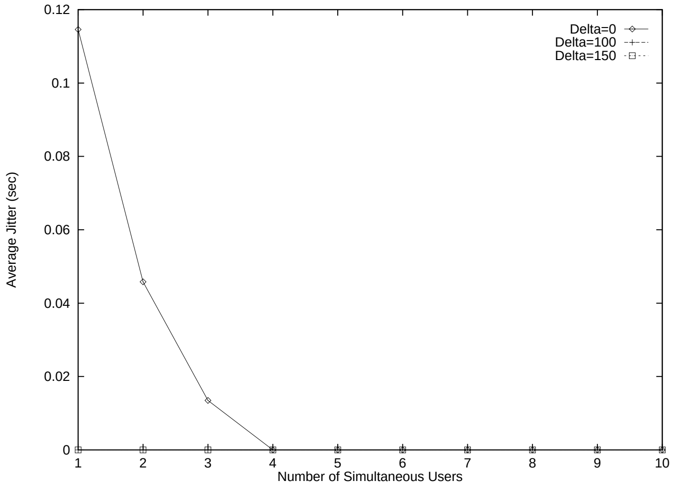

Fig. 5. Average Jitter for Delta as Time

While a reduction in the average startup latency due to hot object caching is not unexpected, the impact on jitter is not obvious. By designating some of the disks as hot object disks we effectively reduce the number of disks available as buffers for saving data fetched from tertiary storage. This reduction could adversely affect the jitter. Figures 5 and 6 show the average jitter observed for the same settings as the above experiments. We see that for both choices of $DELTA$, the observed jitter is in fact lower than that without hot object caching. In fact, there is no observed jitter with hot object caching. The combination of hot objects caching and replication of objects on tertiary storage is the primary reason for this reduction. Under our scheme the playback of a document is not started until the disk resident objects for the objects can completely mask the latency of bringing the document onto a drive in the tape. Once this happens, the entire document is retrieved from tape in a sequential read resulting in no jitter. Note that startup latency could be further reduced as the cost of some increase in jitter if we begin the playback of disk resident components earlier without regards to completely masking the tertiary latency.

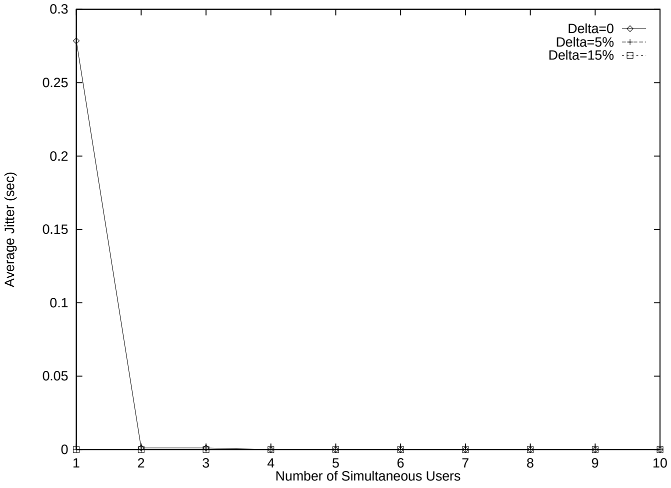

Fig. 6. Average Jitter for Delta as Percentage

5.3. *Comparison to PIRATE*

The Partial Replacement Technique (PIRATE) cache management scheme proposed in [9] is specially designed for the management of multimedia objects on a secondary, tertiary storage hierarchy. The PIRATE scheme is developed and tested for a single user environment with a single buffer disk. In order to test the performance of this scheme for the multi-user, multi-disk environment, it was necessary to adapt the scheme.

In our implementation of PIRATE, we choose the granularity of replacement as blocks of size equal to tape blocks. The original scheme proposes that each object be divided up into fixed sized units called blocks. The replacement occurs in block units. Since we need to migrate the scheme to a set of disks rather than a single disk, the choice of the disk becomes a factor that comes into play. The original scheme takes into consideration the frequency of access, called the "heat", to choose victims. We also use the same parameter to choose a victim. We scan through the disk resident objects and choose the object having the lowest access value as the victim. This victim determines our choice of the disk that will provide the set of victims to be partially replaced to accommodate the incoming object. This may not be the best choice because in an environment with multiple disks the objects are scattered across the buffers and frequency of access of a single object may not be sufficient indication of the nature of the objects on that disk. Another approach could be to determine the average access frequency of objects in each disk and choose the disk having the lowest average frequency of access of objects . Then we select replacement objects from this disk. However this would involve considerable overhead in the presence of a number of disks. So with our choice of the disk storing the LFU object, the objects on the disk are scanned to select victims. Victims are selected in ascending order of access frequency starting with the least frequently accessed object and the tail-end block of each victim is replaced. The notion of "slice" in the original PIRATE algorithm is taken to be a set of 10 blocks. If the disk resident portion of the object is less than 10 blocks than that object is not a candidate for replacement.

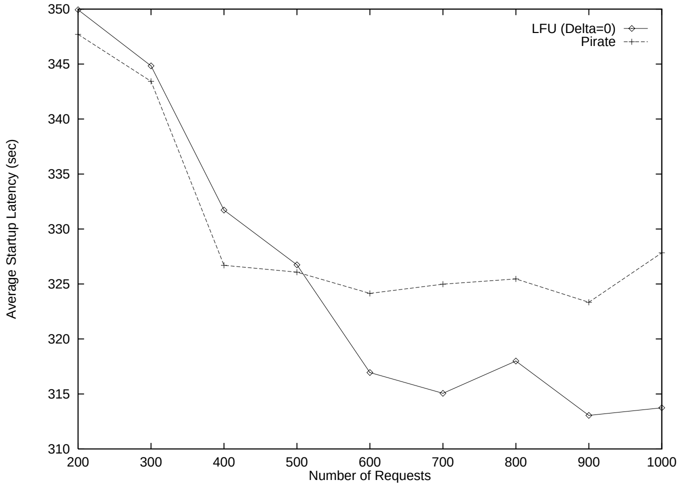

Fig. 7. Performance of *PIRATE* versus LFU

While servicing user requests, if the scheduler detects a request to an object that is partially disk resident then a fetch for the remaining portion of the object from the tape is scheduled as a fetch consecutive to the fetch from disk. The tape placement is considered and the fetch start position on tape is calculated from the start of the entire object and the portion resident on disk. Since the documents on tape are stored in full replication, there may be multiple occurrences of the same object on tape. A bad selection of the object from tape can result in the overhead of unloading and loading of a new tape. To avoid this we use the document ID as a parameter in addition to the object ID to select the current tape. This increases the probability of selecting a tape that is already loaded or one that will be required to be loaded for other accesses too.

In Figure 7 the average startup latency of the PIRATE replacement scheme versus the simple LFU replacement is presented. The experiment is conducted with 10 concurrent users. The results for varying numbers of requests is shown. Surprisingly, we find that for even as few as 600 requests, the PIRATE scheme does not outperform the simple LFU scheme. This is in contrast to the results presented in [9] for a single user environment. The poor performance of PIRATE can be traced to the increased tertiary storage accesses as the number of requests increases. Since with increased requests it is necessary to replace objects on disk, PIRATE replaces small sections of several objects instead of replacing entire objects. Consequently, most objects in the cache are incomplete resulting in the need to access tertiary storage for the remainder, no matter how small it is. As the number of objects accessed increases, the performance degrades even more. Since the PIRATE scheme did not perform better than LFU, it will clearly give poor performance as compared to our hot object caching scheme too. Consequently, no direct comparison is necessary.

5.4. *Number of Hot Prefix Disks*

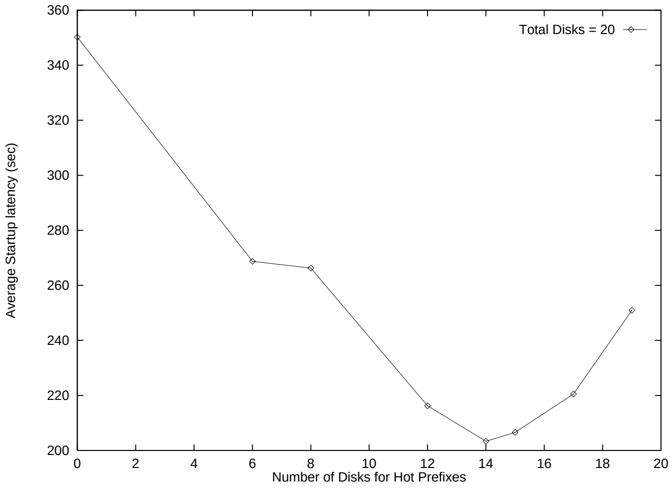

Fig. 8. Impact of Number of Hot Cache Disks, (B)

The fraction of disks used for hot object caching is an important parameter. In this experiment we study the impact of this parameter. Figure 8 shows the average startup latency as the fraction of disks used as hot cache buffers in increased. Initialy, as the number of hot object disks is increased, there is a reduction in the average latency due to the benefit of latency masking. However, as we go beyond 14 disks, the latency begins to increase again. This increase is due to the greatly reduced amount of space available for buffering leading to delays. From the graph we see that a choice of 14 out of a total of 20 disks is optimal for caching hot objects.

5.5. *Choice of $DELTA$*

In this experiment we study the impact of $DELTA$ for different numbers of hot cache disks. Figure 9 shows the average latency as a function of $DELTA$. The value

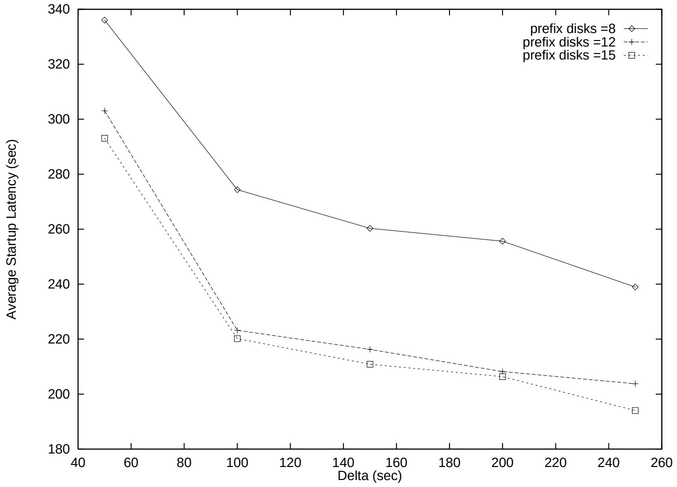

Fig. 9. Impact of $DELTA$

of $DELTA$ is varied from 0 to 250 in steps of 50. The number of simultaneous users in the system was maintained at 10. Three sets of graphs are shown for the number of hot cache disks as 8, 12, and 15. The plot shows that with the increase in the value of $DELTA$ there is a considerable decrease in the startup delay. This can be attributed to the fact that with a larger delta the number of objects cached in the hot cache increases resulting in larger document prefixes being available for fast retrieval and transmission. We can see that the performance for different choices of hot cache disks is very similar with respect to $DELTA$. Thus $DELTA$ can be chosen independently. If the number of disks is hot cache disks is chosen to be 15 (as suggested by the previous experiment), a choice of $DELTA$ to be around 250 seconds gives good performance.

5.6. *Network Bandwidth*

In this experiment the impact of the available total network bandwidth available for transmission is studied. The bandwidth was varied from 1MB/sec to 10MB/sec – which is a reasonable value for a 10/100 Ethernet node. The number of simultaneous users in the system was maintained at 10. Figure 10 shows the startup latency as a function of the bandwidth. As expected, for low bandwidth, the latency is very high as the network becomes a bottleneck. However with increase in the total bandwidth available, the latency drops sharply. Clearly for larger numbers of users, the 10MB/s bandwidth will be inadequate. We can safely assume that with a Gigabit Ethernet, the network will not be a bottleneck even for larger numbers of users.

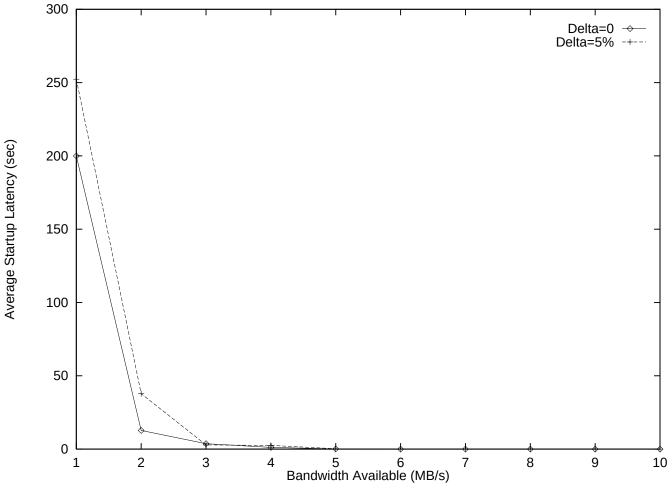

Fig. 10. Impact of available network bandwidth

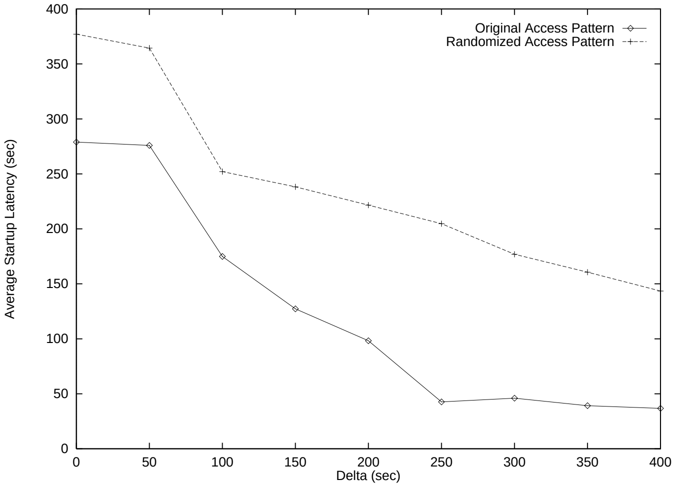

Fig. 11. Impact of Random Changes in Browsing Graph

5.7. *Adapting to Variations in Access Pattern*

In the preceding experiments it is assumed that the access probabilities of documents are known a priori. Based upon this information, the hot cache placement is determined. We now investigate the impact of variations in the access pattern and also the ability of the adaptive placement scheme to adjust to these variations. We begin by considering a drastic change in the access pattern. Figure 11 the average latency is plotted versus $DELTA$ is shown as the access pattern is changed randomly. We observe that there is an increase in the access latency as a result of the change. However, it is interesting to note that even with a very different access pattern than the one used to determine the placement, the use of hot object caching is effective in reducing latency.

In Figures 12 and 13 we study the impact of limited random changes in the document access probabilities and the edge probabilities respectively. In each experiment the placement is generated based upon an initial access pattern. Next, a random subset of 10% of the nodes (edges) are chosen and their probabilities are altered by 10%, 20%, etc. The performance is tested using this altered access pattern. The frequency of access to documents based upon this altered graph is captured and a new placement is made based only upon these observed frequencies (with no other knowledge of the changed access pattern). Using this adapted placement, the performance is again measured. This is repeated for varying degrees of changes from the original access pattern.

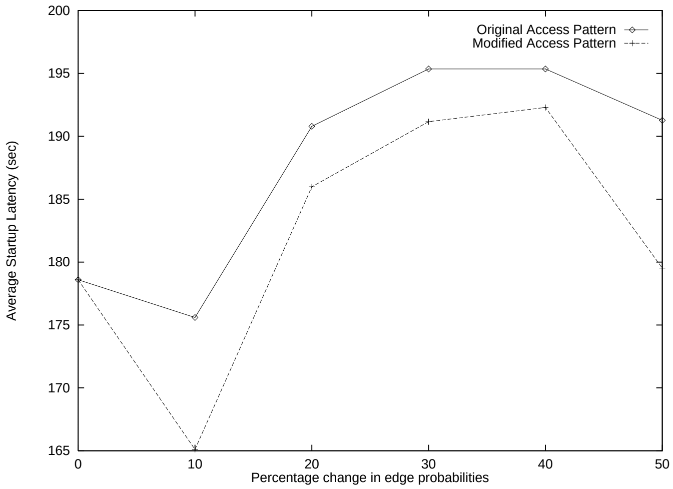

Fig. 12. Impact of Changes in Edge probabilities

In each graph we observe that by adapting to the observed pattern of access, we are able to reduce the latency. It is interesting to note that the increase in the latency is not large, even with 50% change in the probabilities.

6. Conclusion

In this paper we address the important problem of reducing startup latency and jitter for very large multimedia document repositories. The study explores a multi-user, multi-disk environment. To the best of our knowledge, this is the first study

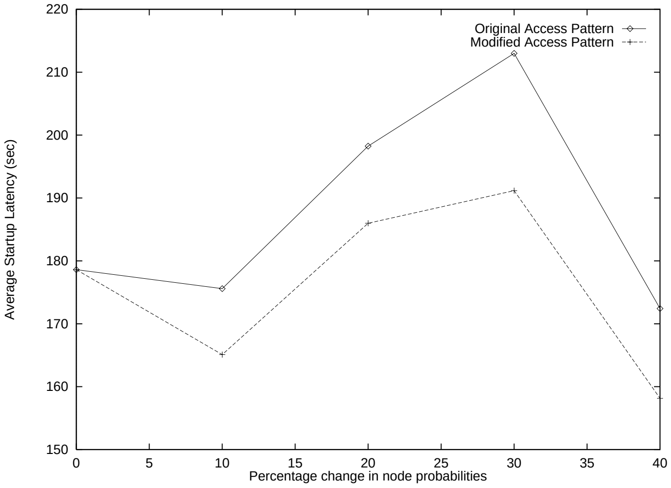

Fig. 13. Impact of Changes in Node probabilities

to explore these issues. We proposed the use of a large portion of the secondary storage as a permanent store for document prefixes in contrast to its customary use as a buffer. We also propose the use of replication on tertiary storage to avoid expensive media exchanges. The effectiveness of these approaches in reducing both startup latency and jitter is shown through extensive experimentation using a detailed simulator. The hot prefix placement scheme is also shown to easily adapt to variations in the access parameters. In our experiments the startup latency is reduced by as much as 75% and jitter is practically eliminated. Our results show that by reserving a large portion of the disk cache for the prefixes of the hottest objects, we are able to achieve very significant improvements in startup latency. Moreover, despite the reduction in available disk buffers, there is no increase in jitter due to replication on tertiary storage.

References

1. E. Bertino and E. Ferrari. Temporal synchronization models for multimedia data. *Transactions on Knowledge and Data Engineering*, 10(4), 1998.
2. A. L. Chervenak. Challenges for tertiary storage in multimedia servers. *Parallel Computing Journal*, 24(1):157–176, 1998.
3. A. L. Chervenak, D. A. Patterson, and R. H. Katz. Storage systems for movies-on-demand video servers. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, pages 246–56, Los Alamitos, CA, September 1995.
4. Stavros Christodoulakis, Peter Triantafillou, and Fenia Zioga. Principles of optimally placing data in tertiary storage libraries. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 236–245. Morgan Kaufmann, 1997.
5. Exabyte. Products. http://www.Exabyte.CO M:80/Products/, Oct. 1996.

6. D. A. Ford and S. Christodoulakis. Optimizing random reterievals from clv format optical disks. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 413–22, Barcelona, Spain, September 1991.
7. C. Georgiadis, P. Triantafillou, and C. Faloutsos. Scheduling and performance of robotic tape libraries in video server environments. Technical report, Multimedia Systems Institute of Crete (MUSIC), Technical University of Crete, Crete, Greece, 1997.
8. S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. *Computer Communications*, 18:170–184, march 1995.
9. S. Ghandeharizadeh and C. Shahabi. On multimedia repositories, personal computers, and hierarchical storage systems. In *Proc. of ACM Int. Conf. on Multimedia*, 1994.
10. B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape. In *SIGMETRICS*, pages 170–9, Canada, 1996.
11. B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Canada, 1996.
12. Urs Hölzle. Google: Fun with linux and clustering. Seminar, Purdue University, September 2001.
13. Theodore Johnson and Ethan L. Miller. Performance measurements of tertiary storage devices. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 50–61. Morgan Kaufmann, 1998.
14. Achim Kraiss and Gerhard Weikum. Vertical data migration in large near-line document archives based on markov-chain predictions. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 246–255. Morgan Kaufmann, 1997.
15. Y.-M. Kwon, E. Ferrari, and E. Bertino. Modeling spatio-temporal constraints for multimedia objects. *Knowledge and Data Engineering*, 1999.
16. T. D. C. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *Journal on Selected Areas in Communication*, 8(3):413–4237, 1990.
17. S. More, S. Muthukrishnan, and E. Shriver. Efficiently sequencing tape resident jobs. In *Proc. ACM Symp. on Principles of Database Systems*, 1999.
18. Powerfile. Products. http://www.dvdchanger.com, Jun. 2001.
19. S. Prabhakar. An overview of current tertiary storage technology and research. Master's thesis, University of California, Santa Barbara, 1998.
20. S. Prabhakar, D. Agrawal, A. El Abbadi, and A. Singh. Scheduling tertiary I/O in database applications. In *Proc. of the 8th International Workshop on Database and Expert Systems Applications*, pages 722–727, Toulouse, France, September 1997.
21. C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, March 1994.
22. H. D. Schwetman. CSIM: A C-based, process-oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, pages 387–396, December 1986.
23. S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. of Infocomm*, 1999.
24. S. Seshadri, D. Rotem, and A. Segev. Optimal arrangements of cartridges in carousel type mass storage systems. *The Computer Journal*, 37(10):873–887, 1994.
25. A. S. Slazay, P. Z. Kunst, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. In *Proc.*

ACM SIGMOD Int. Conf. on Management of Data, pages 451–462, Dallas, Texas, May 2000.
26. StorageTek. Automatic tape libraries. http://www.storagetek.com/products/tape, Jun. 2001.
27. P. Triantafillou, S. Christodoulakis, and C. Georgiadis. Optimal data placement on disks: A comprehensive solution for different technologies. Technical report, Multimedia Systems Institute of Crete (MUSIC), Technical University of Crete, Crete, Greece, 1996.
28. P. Triantafillou and T. Papadakis. Exploiting tertiary storage for performance improvement in video-on-demand servers. Technical report, Multimedia Systems Institute of Crete (MUSIC), Technical University of Crete, Crete, Greece, 1998.
29. Peter Triantafillou and Thomas Papadakis. On-demand data elevation in hierarchical multimedia storage servers. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 226–235. Morgan Kaufmann, 1997.

7. Acknowledgements

This work was supported by NSF CAREER grant No. IIS-9985019, and NSF Grant 0010044-CCR.

Photo and Bibliography

Sunil Prabhakar is an Assistant Professor of the Department of Computer Sciences at Purdue University. He received the Bachelor of Technology in Electrical Engineering from the Indian Institute of Technology, Delhi in 1990, and M.S. and Ph.D. in Computer Science from the University of California, Santa Barbara in 1998.

Dr. Prabhakar's research interests are in large-scale data management, parallel and multimedia databases, and digital watermarking. His research has been supported by NSF, Microsoft Corp., IBM Corp., and the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University.

Dr. Prabhakar is a recepient of the NSF CAREER award. He is a member of the Editorial Board for the Journal of Database Management and a member of the IEEE and ACM.

Rahul Chari holds a Masters degree in Computer Science from Purdue University. At Purdue University, he has worked closely with Professor Sunil Prabhakar in the area of Multimedia data caching and retrieval. He is currently working for Andiamo Systems Inc. as a Software Design engineer in the area of Storage Area Networks (SAN switching). His current interests lie in the areas of volume management and virtualized storage systems.