A Study on the Use of Tertiary Storage in Multimedia Systems*

Leana Golubchik†
Department of Computer Science
A.V. Williams Building, Room 4129
University of Maryland
College Park, MD 20742
Email: leana@cs.umd.edu
Phone: (301) 405-2751; Fax: (301) 405-6707

Raj Kumar Rajendran
Department of Computer Science
500 W. 120th Street
Columbia University
New York, NY 10027
Email: kumar@cs.columbia.edu

Abstract

In spite of the dramatic improvements in the cost of secondary storage, magnetic disks alone are inadequate for meeting the storage and delivery needs of many modern applications. Novel designs of storage hierarchies are needed if we wish to take advantage of the low cost of tape media but still provide reasonable performance characteristics, in the context of modern applications. Specifically, we are interested in multimedia storage server applications, which in addition to high storage and high bandwidth requirements must support display of continuous data streams (e.g., video) and thus satisfy real time constraints. In this paper we focus on the issues and tradeoffs involved in the design of multimedia tertiary storage systems that store and retrieve *heterogeneous* multimedia objects.

1 Introduction

In spite of the dramatic improvements in the cost of secondary storage, magnetic disks alone are inadequate for meeting the storage and delivery needs of many modern applications [6]. Tape media is still two orders of magnitude less expensive than magnetic disk storage; although, tape drives exhibit access latencies two to four orders of magnitude larger than that of disk drives. Novel designs of storage hierarchies are needed if we wish to take advantage of the low cost of tape media but still provide reasonable performance characteristics, in the context of modern applications. Specifically, we are interested in multimedia storage server applications (such as video-on-demand systems), which in addition to high storage and high bandwidth requirements must support display of continuous data streams (e.g., video) and thus satisfy real time constraints — that is, once a display of an object begins, it must continue at a specified bandwidth for the entire duration of that object's display.

*This work was supported in part by the NSF CAREER grant CCR-96-25013.

†This work was partially done while the author was with the Department of Computer Science at Columbia University.

Thus far, relatively little work has been done on server architectures that use tape storage as an online repository of data, especially in the context of multimedia systems that store and deliver *heterogeneous*[1] objects. The success of RAID technology, and the low bandwidths of then current tape-drive technology spurred some early work on striping of data over multiple tapes[2] [2, 1]. At the time, the conclusions were, to a large extent, that: (1) there was limited utility to striping, as contention for the drives and the large latency reduced the throughput of the system, (2) such a strategy was useful only for sequential access and backup, and (3) tertiary storage was not a viable delivery mechanism for continuous media data (such as video), as tape-striping performed poorly under concurrent access. In a later work [4] the authors showed that, depending on the system workload, tape striping can be a viable and attractive alternative. Some of the issues encountered in multi-level multimedia server design were addressed in [3].

Given the current tape drive technology, a single drive can simultaneously support multiple continuous streams of data, and thus it is important to devise schemes to efficiently share the bandwidth among many streams while satisfying the real-time constrains. The idea of servicing multiple streams from one tape system by buffering data on secondary storage was exploited in [7], where the notion of a *cycle* was introduced in the context of tertiary storage. During a cycle, data needed to serve each active stream for the duration of one cycle is read and buffered[3] on secondary storage and then delivered to users in the next cycle[4]. Only as many streams as can be serviced while still meeting the real-time constraints are admitted into the system.

The design of such multimedia tertiary storage systems raises some interesting questions, which include: (a) how much data should be read in each cycle — the larger the amount, the better the tertiary bandwidth utilization, but the smaller the amount, the better the response time of the system (or latency to starting service of a new request) and the smaller the required buffer (or secondary storage staging) space and (b) what is the latency of the system under different workloads and what are the factors that affect latency. In this paper we focus on addressing such questions in the context of tertiary storage systems that store and retrieve *heterogeneous* multimedia objects.

In devising a scheduling technique for servicing real-time requests in this periodic manner, it is important to address the following questions. Firstly, what latency can we expect for requests being serviced when there is a certain load on the system, i.e., how early in the reservation schedule can we find a sequence of slots that can satisfy a particular request. And secondly, what is the resulting cost of the storage subsystem. More specifically, one important tradeoff is between improved bandwidth utilization (on the tape subsystem) and increased buffer space requirements (on the disk subsystem). Since it is not immediately clear how to compare savings in I/O bandwidth with savings in buffer space, one approach is to assess this tradeoff through cost considerations, e.g., in this case a meaningful perfor-

[1]The heterogeneity is reflected in the different sizes and transfer rate requirements of the multimedia objects, as defined later in the paper.

[2]Due to space limitations, we will only mention a few representative existing works on this topic.

[3]In the remainder of the paper "buffering" will refer to caching of data, read from tertiary storage, on secondary storage, since this is the level of the storage hierarchy we are studying here.

[4]The "offset" between data retrieval and data delivery is needed in order to maintain the real-time constraints.

mance measure is $/stream.

In this paper, we first introduce some simple techniques for scheduling retrieval of heterogeneous multimedia objects on a tertiary storage system and show that these techniques suffer from poor utilization and unpredictable latency partly due to bandwidth fragmentation. We then present a novel strategy, termed *rounds*, which addresses the issue of bandwidth fragmentation and results in better bandwidth utilization and more predictable latency. We study the relative improvement in cost/performance of *rounds*, as compared to the simpler schemes, using simulation as well as cost and characteristics of existing commercial robotic tape storage libraries.

The remainder of the paper is organized as follows. Section 2 describes the architecture of the system, identifies the resource allocation problem that needs to be solved, and then describes several strategies that can be used to solve this problem. Section 3 presents analysis of the various strategies and their comparison using a set of performance as well as cost/performance metrics. Section 4 gives our concluding remarks.

2 System

In this section we describe the system we are considering, define some issues and problems involved in the design of such systems, and then present several schemes that address these issues.

2.1 Architecture and Terminology

The multimedia storage server considered here consists of a three level storage hierarchy, including main memory, disk storage, and tertiary storage. In this work we concentrate on the tertiary level of the system, from which data is retrieved and staged to the disk subsystem before delivery; as already mentioned, this is motivated by the need to meet real-time constraints. Briefly, the tertiary storage system under study is a robotic tape storage library, which contains a relatively large number of tapes, relatively few drives, and a robotic arm used to move tapes in and out of drives[5]. The entire database is stored on tertiary storage and data is retrieved on-demand. All data stored in the tape library is in units of constant size, termed *pages*.

All requests to the tertiary subsystem are for sets of pages which are randomly distributed among the tapes. We assume (given the large number of tapes and the relatively small number of drives) that each retrieval of a page requires a tape exchange, and more specifically, it requires a robot and drive load, a seek to the required page on the tape, the read itself, a rewind, and a drive and robot unload (refer to [1, 4] for details). The time required for all these operations to complete is termed *cycle time* (C_t).

Each request is represented by a tuple (b, p), where: (1) b is the requested transfer rate, bounded on the low side by a predetermined value, which we will refer to as *minimum bandwidth* (B_{min}), and on the high side by the combined effective capacities of all the tape drives in the system, which we will refer to as *maximum bandwidth* (B_{max}) and (2) p is the number of pages to be transferred at that rate. When a request for a certain transfer

[5] More than a single arm is possible, but for ease of exposition we will consider a single arm here, unless otherwise stated.

rate arrives, the following must be reserved in order to service the request and maintain real-time constraints: (a) *streaming bandwidth* on the tape[6] subsystem for reading the data, (b) *staging bandwidth* on the disk subsystem for writing the data (later to be delivered from the disk subsystem), and (c) *staging space* on the disk subsystem for caching the staged data. We focus on reservation of tertiary bandwidth first. Each reservation unit is made up of two parts: the reservation of a robot for the length of time it takes to load/unload a tape and the reservation of a drive for the length of time it takes to load, seek, read, unwind, and unload. This set of reservations is termed a *slot*. Slots corresponding to one drive are staggered from another drive by the "robot latency" — if the slots corresponding to the different drives were synchronized, all drives would require the robot to exchange tapes simultaneously (which is not possible with a single robot arm).

The main notation used in this paper is summarized in Table 1, for ease of reference. We will define the various terms in this table, as the need for it arises.

Notation	Definition
D	Number of tape drives
L_t	Robot latency
C_t	Cycle time
B_{max}	Maximum allowed request bandwidth
B_{min}	Minimum allowed request bandwidth
S_t	total stagger (due to contention for the robot arm)
R_l	Round length (number of slots in a round per drive)
R_s	Round size (number of slots in a round)

Table 1: Notation.

2.2 Problem

Given real-time (or continuity) constraints of multimedia data, when a new request arrives to the system, the tertiary subsystem needs to reserve *periodic* tape drive slots for the entire length of that request. The periodicity of slot reservation corresponds to the bandwidth of the request, e.g., a request that requires a bandwidth that is a quarter of the bandwidth of a single tape drive needs to be scheduled one slot out of every four, on a single drive. In devising a scheduling technique for servicing the real-time requests in this periodic manner, it is important to address the following questions. Firstly, what latency can we expect for requests being serviced when there is a certain load on the system, i.e., how early in the reservation schedule can we find a sequence of slots that can satisfy a particular request. And secondly, what is the resulting cost of the storage subsystem. More specifically, as

[6]Note that, streaming bandwidth is also needed on the disk subsystem to eventually deliver the data to the user, but, as already mentioned, we focus on the tertiary level here.

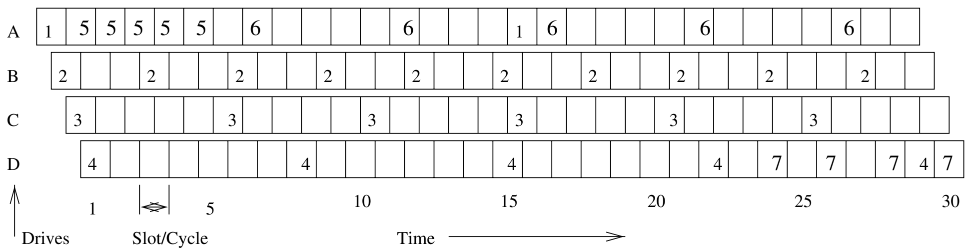

Figure 1: Schedule with *Simple* Scheme.

already mentioned, one important tradeoff is between improved bandwidth utilization (on the tape subsystem) and increased buffer space requirements (on the disk subsystem). Since it is not immediately clear how to compare savings in I/O bandwidth with savings in buffer space, one approach is to assess this tradeoff through cost considerations, e.g., in this case a meaningful performance measure is \$/stream. We elaborate on this later in the paper. Below, we first give a (fairly simple) strategy for scheduling of requests for multimedia objects in a (strictly) periodic manner in order to illustrate the basic problem. We term this strategy *simple*.

The *simple* scheduling strategy allocates each request to one drive[7]. The drive that can satisfy the request the earliest is chosen. We illustrate this strategy and the associated problems through an example. Consider a system with four drives, each with an effective bandwidths of 4 MB/s. Let all requests in the system have bandwidth requirements that range from 4 MB/s to 0.25 MB/s. Thus, a request with the maximum bandwidth requirement would need consecutive slots to service it, while a request with the minimum bandwidth requirement would need one slot on one drive every sixteenth cycle. Then, given the following sequence of request arrivals:

- Req. 1: $(b, p) = (0.25$ MB/s, 2 Pages), i.e., 1 slot in 16
- Req. 2: $(b, p) = (1.3$ MB/s, 10 Pages), i.e., 1 slot in 3
- Req. 3: $(b, p) = (0.8$ MB/s, 6 Pages), i.e., 1 slot in 5
- Req. 4: $(b, p) = (0.57$ MB/s, 5 Pages), i.e., 1 slot in 7
- Req. 5: $(b, p) = (4$ MB/s, 5 Pages), i.e., all consecutive slots
- Req. 6: $(b, p) = (0.8$ MB/s, 5 Pages), i.e., 1 slot on 5
- Req. 7: $(b, p) = (2$ MB/s, 7 Pages), i.e., 1 slot in 2

the corresponding reservation schedule is depicted in Figure 1. Note the periodicity of the various requests and how they correspond to the request bandwidth requirements.

The state of the reservation array in Figure 1 and the resulting schedule of requests illustrate a problem: even thought the array is quite sparse, delays are already becoming

[7]Although *simple* is a limited strategy, in a sense that it can not service requests that require more than a single drive worth of bandwidth, we use it in this paper to illustrate some of the issues and tradeoffs associated with delivery of heterogeneous multimedia objects from tertiary storage.

considerable. For instance, Req. 7 could be satisfied no earlier than time slot 24. This is due to the fact that, even though the schedule is relatively sparse, it is already fairly fragmented and thus finding a set of empty slots *with a certain periodicity* is "difficult". This problem also makes it more difficult to predict how far into the reservation schedule we would have to look to find the slots that will satisfy a new request, i.e., it is difficult to predict the latency to service a newly arrived request.

One of the causes of the above problem is the restriction that each request be serviced by a single drive. This restriction exists due to the problem of *stagger* caused by robot latency — if the slots corresponding to the different drives were "synchronized", then all drives would require the robot to exchange tapes simultaneously, which is not possible. If we could find a solution to the problem of stagger, we could gain additional flexibility in scheduling requests and consequently attain a better utilization of tertiary bandwidth.

One simple solution to this problem is use of additional buffer space. That is, data from tape drives that start their reads earlier can be buffered on secondary storage until the last tape drive is ready to read. This "synchronization" of slots essentially makes tape drives "interchangeable", allowing the use of different tape drives in retrieving the various pages of a request, while ensuring constant bandwidth allocation to a request. The motivation is that in such a system, requests can be fitted into the reservation schedule earlier (than for instance in a system using *simple*), utilizing some combination of drives rather than just one to satisfy the request. We will refer to this technique as the *buffered* scheme.

More specifically, the slots that correspond to different tape drives are staggered from each other by the robot latency L_r, i.e., the second drive is staggered from the first by L_r, the third from the first by $2 \times L_r$, and so on. Thus, the *total stagger* (S_t) is equal to $\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2}$ cycles (or slots), where D is the number of drives, L_r is the robot latency, and C_t is the cycle time.

We illustrate the *buffered* scheme through another example. Given the following sequence of request arrivals:

- Req. 1: $(b, p) = (0.25$ MB/s, 2 Pages), i.e., 1 slot in 16 on one dr ive
- Req. 2: $(b, p) = (1.3$ MB/s, 10 Pages), i.e., 1 slot in 3 on one drive
- Req. 3: $(b, p) = (0.8$ MB/s, 6 Pages), i.e., 1 slot in 5 on one drive
- Req. 4: $(b, p) = (0.57$ MB/s, 5 Pages), i.e., 1 slot in 7 on one drive
- Req. 5: $(b, p) = (2$ MB/s, 9 Pages), i.e., 1 slot in 2 on one drive
- Req. 6: $(b, p) = (16$ MB/s, 4 Pages), i.e., 1 slot on all four drives
- Req. 7: $(b, p) = (1$ MB/s, 5 Pages), i.e., 1 slot in 4 on one drive

the corresponding reservation schedule is depicted in Figure 2. Note that the "synchronization" of drives also increases the range of bandwidths supported. While the maximum bandwidth supported under the *simple* scheme was the bandwidth of a *single* tape drive, the maximum bandwidth that can be serviced under the *buffered* scheme has increased to the bandwidth of all the tape drives combined, e.g., Req. 6, which has a bandwidth requirement of 16 MB/s, is serviced by all four drives. However, the buffered scheme does not

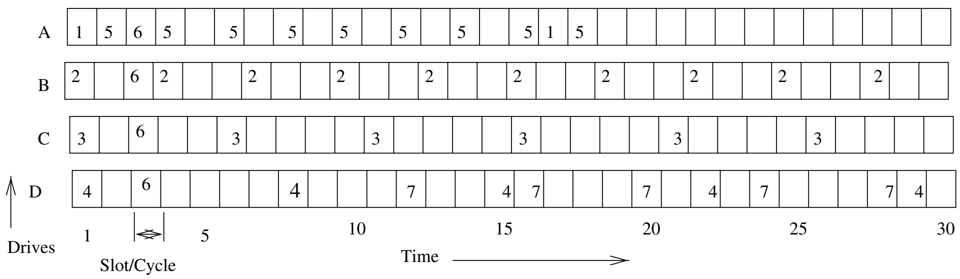

Figure 2: Schedule with *Buffered* Scheme.

(completely) eliminate bandwidth fragmentation. Note that, even though the reservation array is fairly sparse, Req. 7 could not be serviced until slot 12.

In summary, although *simple* and *buffered* strategies will satisfy the real-time constraints of multimedia data, one problem that results from such strategies is *bandwidth fragmentation*, partly due to the *heterogeneity* of user requests. This consequently results in less efficient bandwidth utilization and latencies in servicing requests. In order to design a more cost-effective system, we need a data retrieval scheduling technique which:

1. reduces bandwidth fragmentation by "packing" slots better; for instance, if we could relax the rigid periodic allocation of slots to requests, without incurring large buffering costs, we would be able to better utilize tape drive bandwidth

2. disassociates a request from a tape drive — if different tape drives could serve different pages of the same request, we could gain additional flexibility[8]

3. facilitates fairly simple latency prediction, i.e., given the current load, what is the expected latency for starting service of a newly arrived request; this is especially useful in systems where there is a possibility of a user reneging, where the probability of that occuring is a function of the expected latency to start of service.

2.3 The Rounds Approach

In this section, we introduce a strategy termed *rounds* that exhibits the characteristics stated above[9]. It is based on aggregating a set of slots into a round, where each slot is homogeneous in time and space, i.e., within a round one slot is the same as any other, irrespective of where it lies chronologically or to which tape drive it corresponds. All pages read from all slots in a round are staged before streaming begins, and each reservation is for a certain number of slots from a particular round. More precisely:

- A *round* is a set of slots, whose *size* is the ratio between the maximum bandwidth, B_{max}, and the minimum bandwidth, B_{min}, i.e., the number of slots per round, $R_s = B_{max}/B_{min}$. The round *length*, R_l, is the number of slots per round per drive. In our

[8]Part of the cost here will be due to tape drive stagger. Since slots corresponding to different drives are staggered by some multiple of the robot load/unload latency, requests serviced by multiple drives would require some additional buffer space to guarantee continuity in data delivery.

[9]The *buffered* technique introduced above only exhibits the second characteristic.

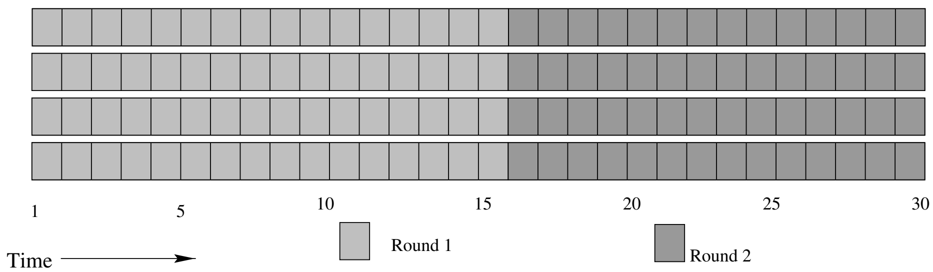

Figure 3: Rounds

example, where $B_{min} = 0.25$ MB/s and $B_{max} = 4 \times 4$ MB/s, $R_s = 64$ and $R_l = 16$. Intuitively, R_l is the time, in slots, it takes to stream a page at B_{min}. This is illustrated in Figure 3 where each set of highlighted slots constitutes a different round.

- A round is the fundamental unit of reservation. All slots in a round are identical (or *homogeneous*). Reservations are done at the granularity of rounds rather than slots. If a request needs a bandwidth of b MB/s, then $\frac{b}{B_{min}}$ slots are reserved per round for the number of rounds required. Since all slots in a round are identical and scheduling is performed at the level of rounds, slots within a round can be packed without concern for periodicity or for drive association – this results in a more efficient bandwidth utilization of the tertiary subsystem.

- Every page in a round is staged to disks before streaming to the user begins. That is, streaming begins after the last page corresponding to a round has been staged. Thus, the average *latency* is half the round length, i.e., $\frac{R_l}{2}$ cycles, given that there is available capacity in the round.

- The required secondary storage staging space should be sufficient to buffer two rounds worth of data (in pages) plus the stagger (which is due to the contention for the robot arm), i.e., $(2 \times R_s) + (\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2})$ pages (we elaborate on this in Section 3.1).

The *rounds* strategy exhibits the following advantages:

- *Near complete utilization.* As long as there are slots left in a round, i.e., as long as there is unused bandwidth available, and the bandwidth of a request fits (*somewhere* in a round), it will be satisfied. In the earlier strictly periodic model, the data access *pattern* had to fit the *pattern of holes* created by unoccupied slots.

- *More predictable quality of service.* As will be described later, *rounds* exhibits smaller variances in latency to starting service of a new request, which facilitates a more predictable quality of service.

- *Simplicity of reservation.* Slots in a round are completely equivalent, so there is no need to allocate blocks in certain patterns or worry about which drive a slot is associated with. Given the *heterogeneity* of applications expected to utilize a mass storage system, this is an attractive feature.

- *Opportunities for optimization.* Given the architectural differences of various tertiary storage systems, flexibility in the order of block and tape retrieval will provide opportunities for scheduling optimization *within* a round (or customization of scheduling to different architectures, e.g., as in [5]).

However, it also exhibits the following disadvantages:

- *Potentially higher average latency under a small class of workloads.* Because streaming only begins at the end of rounds, there is a potential for higher average latency as compared to a strategy without rounds where streaming can begin as soon as a page is staged. However, our experiments indicate that this occurs *only* under *light* loads, and that in those cases the latency penalty is not significant.

- *Higher secondary storage.* In the earlier strict periodic model, each page was streamed as soon as it finished staging, while in *rounds*, all pages in a round are staged before streaming begins; thus, an additional secondary storage cost is incurred.

These advantages and disadvantages are quantified in the following section. Before we proceed, we would also like to mention that other variations on the three schemes given here are possible; however, we do not discuss them here as one of our main goals is to illustrate the issues and tradeoffs involved in the retrieval of heterogeneous multimedia objects from tertiary storage, using simple strategies.

3 Analysis

To determine the usefulness of the rounds technique, we evaluate its performance and cost (which includes both tertiary storage system cost as well as secondary storage system cost needed for staging of data from tertiary storage and streaming it to users) and compare it to the two other strategies, namely *simple* and *buffered*. We study the performance of all three strategies using simulation[10]. Specifically, we consider their behavior using the following performance metrics: (a) *latency*, time to start service of new requests, (b) *misses*, the fraction of requests rejected (based on a maximum latency system requirement, i.e., requests that must wait for more than a prespecified maximum amount of time are rejected), and (c) *cost/stream*, which includes the cost of the robotic tape storage library and the disk subsystem needed to support a specified level of performance and quality of service[11].

3.1 Secondary Storage Usage

We begin our analysis by examining the secondary storage requirements of the three strategies described in Section 2, since secondary storage usage constitutes one of the main differences between these techniques. More specifically, secondary storage is required by the different strategies for three purposes: staging, streaming, stagger-smoothing. All three

[10] In our evaluation, we consider existing robotic tape storage technologies, such as Ampex, Exabyte, etc.

[11] It is simple to show that the maximum amount of secondary storage is required (for staging) when all requests are for the lowest bandwidth available; thus, in our evaluation, we use this worst case possibility to compute the secondary storage cost.

strategies require space for staging and streaming while only *buffered* and *rounds* require space for stagger-smoothing, where the space required to compensate for the stagger is a function of the number of drives, as described earlier, and is equal to $(\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2})$.

Furthermore, each tape drive requires buffer space to write the page that it is currently reading. This is the required staging space and is equal to D pages. Once a page has been staged to secondary storage, it will be resident there until it has been streamed (to the user). The length of time it is resident is a function of the required streaming rate — the slower the streaming rate, the longer it will reside on secondary storage; therefore, the peak demand for streaming buffer space occurs when all requests are for the minimum bandwidth. For instance, in our earlier example where $B_{max} = 16$ MB/s and $B_{min} = 0.25$ MB/s, the maximum demand for streaming buffer space occurs when 64 requests for 0.25 MB/s are being serviced simultaneously, where each page, which is staged at 4 MB/s, remains in secondary storage for 16 cycles. Contrast this to a situation where 4 requests, each with a 4 MB/s bandwidth requirement, are being serviced where each page is resident in secondary storage for only one cycle while being streamed[12]. We thus use the worst case scenario, where all requests are for the lowest bandwidth, to calculate the peak demand for streaming buffer space for each scheme. Hence, the amount of storage required varies according to the strategy used and is given below:

- **Simple**: in the worst case, each page takes a cycle to be staged and R_l cycles to be streamed. Thus, each page remains in secondary storage $R_l + 1$ cycles; furthermore, each set of D such pages are offset by one slot. Thus, at any point in time there are D pages being staged, and $D \times R_l$ pages being streamed making the peak total demand $R_s + D$ pages.

- **Buffered**:

  the worst case is the same as in the *simple* strategy; thus, $D \times (R_l + 1)$ pages are required for staging and streaming. In addition, secondary storage is required to account for the stagger smoothing, as described in Section 2.2, where the total stagger was given as $\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2}$ cycles.

- **Rounds**: a page takes a cycle to be staged; it is then resident in secondary storage until the end of the round and is then streamed in R_l cycles. The peak secondary storage demand occurs during the last cycle of each round when R_s pages from the previous round are still being streamed, while the last D of the current round's pages are being staged, giving a total of $2 \times R_s$ pages. In addition, secondary storage is required to account for the stagger smoothing, as described in Section 2.2, where the total stagger was given as $\frac{L_r}{C_t} \times \frac{D \times (D-1)}{2}$ cycles.

[12]In both cases secondary storage space can be reclaimed earlier, but for simplicity of exposition, we assume that space is not reclaimed until the whole page has been streamed.

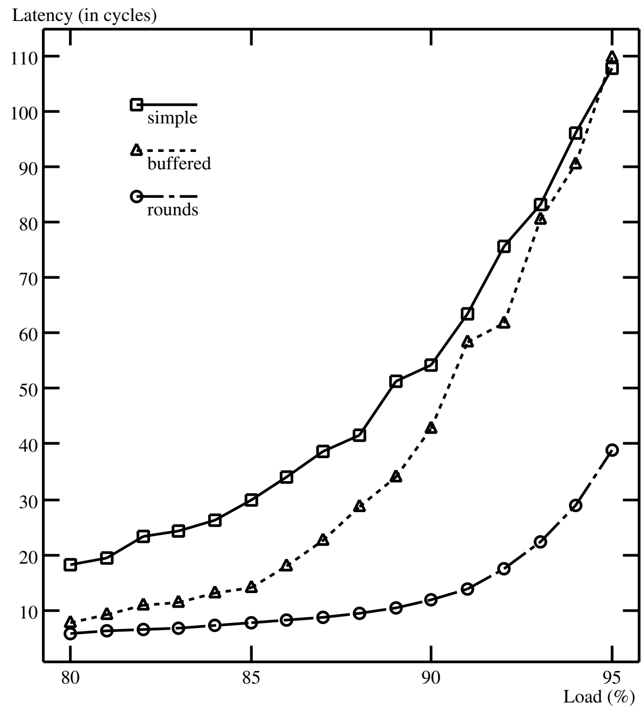

Figure 4: Latency.

3.2 Simulation Results

In our simulation[13], data requests consisting of a tuple (p, r), where p is the request size in number of pages and r is the periodicity[14], are generated using a Poisson processes where the inter-arrival times are exponentially distributed[15] and p and r are uniformly distributed. The simulation system then attempts to fit each request into the reservation schedule as early as possible using one of the strategies described in Section 2. If a fit cannot be found within a *specified maximum latency* (which is a design parameter), a “miss” is recorded. Unless stated otherwise, in the following analysis, we use a system with: (a) 4 drives, (b) request sizes that are uniformly distributed between 1 and 15, (c) periodicities that are uniformly distributed between 1 and 5, and (d) specified maximum latency of 400 cycles.

We first investigate the performance of the different schemes using the *latency* and *miss* metrics — Figures 4 and 5 depict latency and misses, respectively, as a function of system load for the three strategies described earlier. As expected, at light workloads, all three strategies exhibit similar latency and miss percentages. For moderate to high workloads, the inefficiencies of *simple* and *buffered* strategies result in much higher latencies and miss

[13]Note that, since *simple* is not able to service requests with bandwidth requirements exceeding the bandwidth capacity of a single drive, to make the comparison between the three schemes fair, we limit the experiments presented in this section to such request types. This is not limiting, in the sense that it still allows us to illustrate the basic issues and tradeoffs involved in retrieval and delivery of heterogeneous multimedia objects from tertiary storage.

[14]That is, to service a request with periodicity r, we must retrieve a page for that request every r^{th} slot of a single tape drive.

[15]The mean will be specified on per-graph basis, as needed.

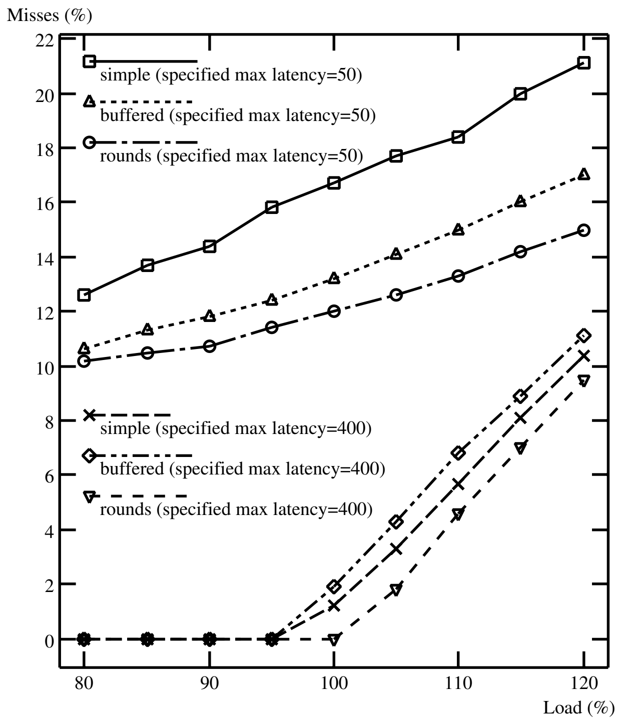

Figure 5: Misses.

percentages, as compared to *rounds*. That is, *rounds* exhibits a more graceful performance degradation as the load increases, i.e., *rounds* can handle higher loads before system saturation occurs — this is due to lower tertiary bandwidth fragmentation which results in better tertiary bandwidth utilization. Note, of course, that the difference in miss percentages is more significant under a relatively smaller specified maximum latency (refer to Figure 5).

We next investigate the effect of *number of drives* and *request sizes* on system performance, using *latency* as the performance metric — Figures 6 and 7 depict system latency as a function of mean request size[16] and number of tape drives, respectively, for all three strategies.

As the mean request size increases, latency increases as well (partly due to the fact that it's more difficult to fit longer requests into a retrieval schedule), although again, *rounds* exhibits a more graceful degradation. Note that, beyond a certain mean request size, latency can start to decrease again (as in the *buffered* and *simple* curves here); this is partly due to the fact that, as the mean request size increases, so does the corresponding miss percentage, which effectively begins to lower the "actual" load on the system (since some requests do not get serviced due to misses). On the other hand, latency decreases as the number of drives in the system increases. This is due to the fact that with more drives, we have greater "flexibility", and thus the fragmentation of tertiary bandwidth has a lesser effect on latency. This is also evident by the fact that *rounds* performs significantly better than the other two schemes in systems with fewer drives (i.e., the better bandwidth utilization of *rounds* is of a (relatively) lesser importance in systems with many drives). In summary, these figures

[16]That is, a mean request size of x means that the request sizes are uniformly distributed between 1 and y, where $x = 1 + \frac{y-1}{2}$.

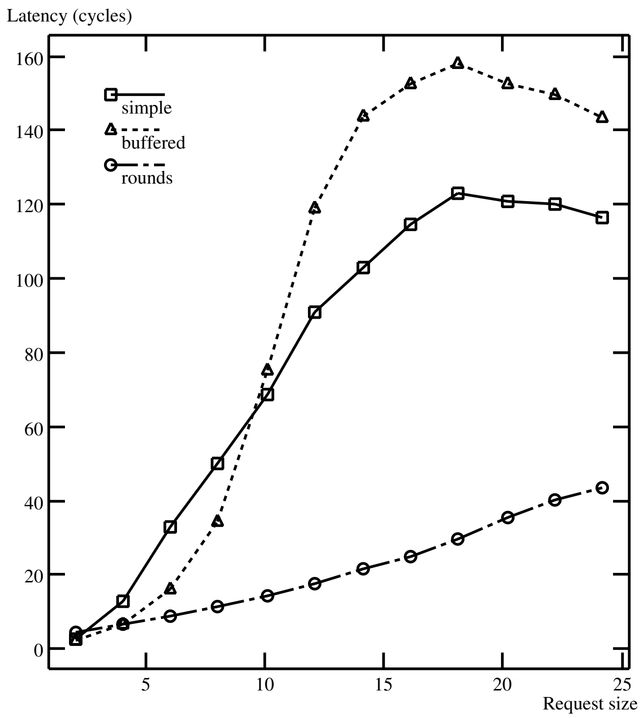

Figure 6: Effect of request sizes on latency (90% load).

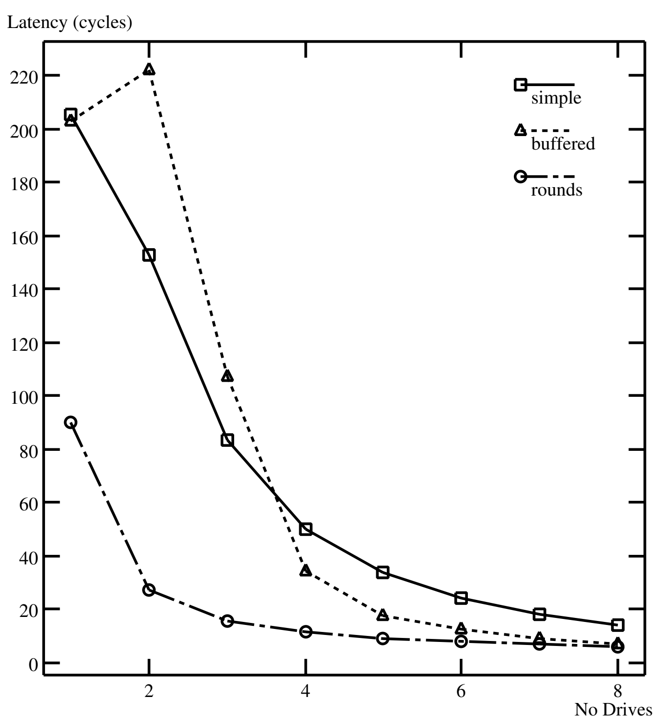

Figure 7: Effect of number of drives on latency (90% load).

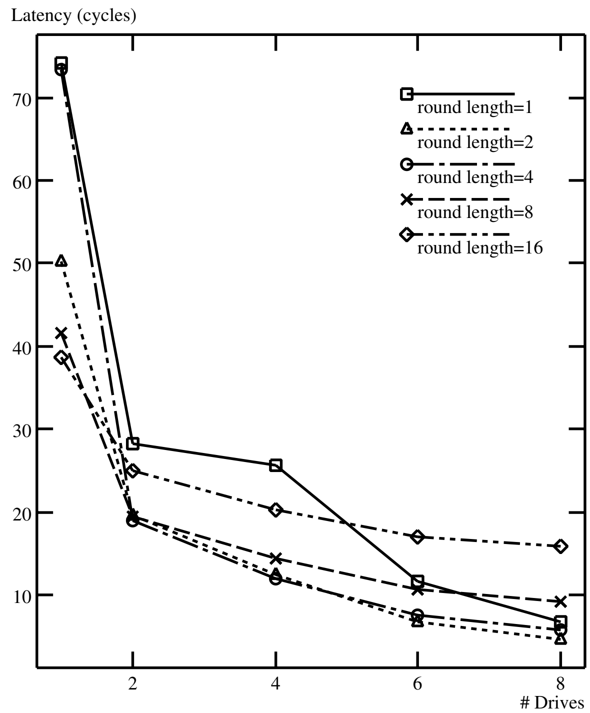

Figure 8: Effect of round length on latency (as a fnc of number of drives at 90% load).

demonstrate that, as request sizes grow larger and the number of tape drives in the system decreases, *rounds* begins to dramatically outperform the other two strategies.

Finally, we examine the effect of *round length* on system performance; specifically we use *latency* as the performance metric — Figures 8 and 9 illustrate the effect of round length[17] on the performance of the *rounds* strategy. Intuitively, the longer the round, the more efficiently we can use tertiary bandwidth, but the potentially higher the latency. Figure 9 illustrates that up to a certain point, as the request sizes increase, larger round lengths become more advantageous; however, after a certain point, it is difficult to compensate, with better bandwidth utilization, for the fraction of latency caused by a long round. Similarly, Figure 8 illustrates that larger round length can compensate for lack of drives, but only up to a certain point.

3.3 Cost-based Comparison of Schemes

Since rounds achieves its better performance (specifically, better utilization of tertiary bandwidth) by utilizing more secondary storage, we need a metric that will allow us to make a comparison between improved bandwidth utilization (on the tape subsystem) and increased storage space requirements (on the disk subsystem). As already mentioned, since it is not immediately clear how to compare savings in I/O bandwidth with savings in storage space, one approach is to assess this tradeoff through cost considerations — in this case a mean-

[17] In varying round length we assume that we have flexibility to vary B_{min}; otherwise, round length would be fixed, as described in Section 2.

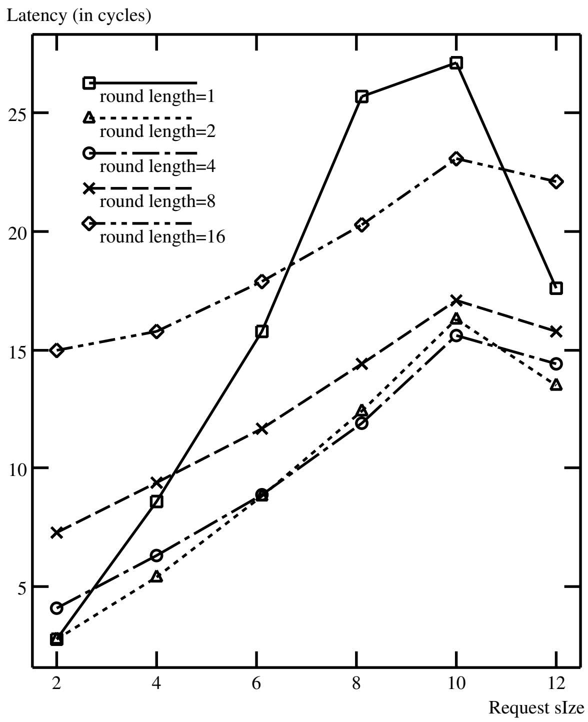

Figure 9: Effect of round length on latency (as a fnc of request size at 90% load).

ingful performance measure is \$/stream.[18] Specifically, we will use this metric to compare the three strategies at points where they provide similar performance.

Capacity (GB)	RPM	Cost($)	Cost/GB
1	5200	600	600
2	5200	800	400
4	5200	1,400	360
9	7800	2,300	255
23	7800	6,000	255

Table 2: Secondary storage.

Tables 2 and 3 give characteristics and price[19] information for some typical disk drives and robotic tape storage libraries. For each system, related characteristics, such as cycle length, secondary bandwidth required, stagger, and so on[20] are listed in Table 4.

[18]We will calculate this by computing the maximum bandwidth that a system can support and the corresponding cost of that system.

[19]These are *list* prices which were compiled from quotes, web-sites, and catalogues during the Spring of 1997.

[20]Cycle time is chosen such that the page size is "reasonably" large, i.e., results in fairly high tertiary bandwidth efficiency, while incurring "reasonably" low secondary storage costs. The number of secondary

Characteristic	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Max. number of Drives	4	6	4	2
Number of robots	1	1	1	1
Robot Latency (s)	20	20	6	10
Drive Latency (s)	190	190	28	185
Total Latency (s)	210	210	34	195
Max. tertiary bandwidth (MB/s)	20	30	60	6
Tertiary capacity (GB)	1,820	6,020	6,400	400

Table 3: Tape libraries.

The secondary storage required and the effective tertiary bandwidth of each system under the different strategies is given in Table 5, for the *simple*, *buffered*, and *rounds* schemes, respectively. To compute the "effective" bandwidth of each system, an "operating load" is chosen for each strategy by noting the load at which some performance metric remains below a constant threshold value. For example, using latency as the metric, 25 cycles as the threshold value, and looking up the load supported in Figure 4 (where the mean request size is 8, mean periodicity is 3, and 4 drives are used), we get operating points of 0.8 for the *simple* strategy, 0.87 for the *buffered* strategy and 0.93 for *rounds*. This is the fraction of the theoretical maximum bandwidth the system can maintain under the given constraints for the parameters chosen. (Refer to Section 3.1 for computation of secondary storage requirements.)

The resulting unit stream costs of the three strategies for the chosen operating points are listed in Tables 6 and 7, for the different systems used. The first operating point corresponds to a system with 1 drive and a mean request size of 18, where the desired performance metric (that is held constant among the three schemes) is a mean latency of 40 cycles. (The $\$/stream$ results for this operating point are given in Table 6.) The second operating point corresponds to a system with a mean request size of 8 and the maximum possible number of drives (i.e., 4, 6, 4, and 2 drives for the ATL4, ATL6, Ampex, and Exabyte AME systems, respectively). The desired performance metrics (that is held constant among the three schemes) in this case is a mean latency of 25 cycles, and the corresponding "operating load" is computed for each scheme and each tape library as described in the example above. (The $\$/stream$ results for this operating point are given in Table 7.) The costs in all cases are computed as the sum of tertiary and secondary storage costs, as described in Tables 4 and 5 as well as Section 3.1.

At the first operating point, rounds consistently shows an approximate *factor of two* improvement over the other two strategies, while at the second operating point, it varies from an $\approx 0.2\%$ degradation, as compared to *buffered* for the ACL 6/176 library, to a $\approx 40\%$ improvement, as compared to *simple* for the Exabyte AME library. Thus, depending on the required operating point, different schemes may be advantageous.

storage devices needed is chosen such that the secondary storage system bandwidth (needed for staging) matches the bandwidth of the tertiary system.

Characteristic	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Read time	243	243	106	310
Secondary Devices Needed	3	5	9	1
Cycle time	453	453	140	505
Cycle size (MB)	1,215	1,215	1,590	930
Round length	4	4	4	4
Round size (max)	16	24	16	8
Tertiary efficiency	0.54	0.54	0.76	0.6
Eff. tertiary BW (w/ 1 drive) (MB/s)	2.7	2.7	11.4	1.8
Eff. tertiary BW (w/ max drives) (MB/s)	10.8	16.3	45.6	3.6
Total Stagger (GB)	0.6	1.5	0.54	0.03
Total Stagger (in cycles)	0.26	0.66	0.26	0.02
Cost of Library (w/ 1 drive)	24,000	62,000	280,000	19,375
Cost of Library (w/ max drives)	57,000	117,000	610,000	26,000

Table 4: Tape library characteristics.

4 Conclusions

In this paper we studied retrieval of *heterogeneous* multimedia objects from a tertiary storage system. We proposed the *rounds* retrieval scheme and compared it to two simpler strategies, namely *simple* and *buffered*. Briefly, some of the main results of our study are as follows[21]:

- *Latency.* For light workloads, rounds exhibits a slightly higher mean latency than the buffered strategy; however, for moderate to high workloads, rounds results in a significantly lower mean latency than the other two techniques, which is due to the fact that it is able to utilize bandwidth better — this becomes more pronounced as the load on the system grows. Furthermore, the variance in latency exhibited by the rounds strategy is smaller than that of the other two strategies, which facilitates a more predictable quality of service.

- *Misses. Rounds* consistently outperforms the other two strategies under this performance metric, again due to its more efficient bandwidth utilization. The differences become more apparent as the maximum allowed latency becomes more stringent and as the number of drives in the system decreases, i.e., either as the quality of service requirements become more stringent or as the amount of resources (which experience high contention) decreases.

- *Round length.* We also experimented with "proper" round lengths (since this is one of the system design parameters). As is probably expected, very small rounds become

[21]That is, we give a summary of all experiments that we have conducted thus far — most of these results are illustrated through the figures and tables given in Section 3; however, due to lack of space, we were not able to include all of these results in that section.

	ACL 4/52			ACL 6/176			Ampex			Exabyte AME		
Staging (Cycles)	20	20	30	30	30	48	20	20	32	10	10	16
Stagger (Cycles)	0	0.26	0.26	0	0.66	0.66	0	0.26	0.26	0	0.02	0.02
Disk space GB (1 drv)	24.3	24.3	26.7	36.5	36.5	55.9	31.8	31.8	49.8	9.3	9.3	14.9
Disk space GB (max)	24.3	24.6	27	36.5	37.3	56.7	31.8	32.3	51.3	9.3	9.3	14.9
Disk Cost k$ (1 drv)	7.2	7.2	7.8	10.9	10.9	15.5	11.8	11.8	16.2	3.0	3.0	4.3
Disk Cost k$ (max)	7.2	7.3	7.9	10.9	11.0	15.6	11.8	11.9	16.3	3.0	3.0	4.3
Ef. Bw. MB/s (1 drv)	3.0	3.0	6.4	4.6	4.6	9.6	12.8	12.8	26.9	1.0	1.0	2.1
Ef. Bw. MB/s (max)	8.6	9.4	10	14.7	15.2	15.7	36.5	39.7	42.4	2.3	2.7	3.4

Table 5: Effective bandwidth and secondary storage requirements.

	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Simple Strategy ($/MB)	$8,000$	$13,480$	$21,870$	$19,375$
Buffered Strategy ($/MB)	$8,000$	$13,480$	$21,870$	$19,375$
Rounds ($/MB)	$3,350$	$6,460$	$10,409$	$9,226$

Table 6: Unit stream cost (operating point 1).

inefficient at higher workloads, and very large rounds incur too high of a secondary storage penalty (as well as contribute to higher latencies) while only marginally improving bandwidth utilization.

- *Efficiency.* The rounds strategy is more efficient at utilizing available resources (specifically, tertiary storage bandwidth). Given the same architectural configuration, a system using *rounds* is able to maintain relatively low latencies at higher throughputs, i.e., a system using *buffered* (or *simple*) approach can become unstable and result in high latencies at significantly lower levels of workload than a system using *rounds*. Our experiments indicate more than a 100% improvement in throughput for systems with fewer drives and larger request sizes, i.e., systems with high contention for resources and high workloads. Thus, *rounds* is a more attractive scheme for systems with wider ranges of workloads.

- *Cost ($/MB/s).* In order to make this a more fair experiment, we compared the cost characteristics of the three strategies at points where they provide the same level of performance. Our experiments indicate that, depending on the architectural configurations of the tertiary libraries and the offered workloads, the cost/stream of *rounds* can vary anywhere from a $\approx$ 3% degradation to a *factor of two* improvement, as

	ACL 4/52	ACL 6/176	Ampex	Exabyte AME
Simple Strategy ($/MB)	$7,435$	$8,699$	$17,044$	$12,590$
Buffered Strategy ($/MB)	$6,841$	$8,425$	$15,664$	$10,725$
Rounds ($/MB)	$6,460$	$8,443$	$14,768$	$9,032$

Table 7: Unit stream cost (operating point 2).

compared to *buffered*[22]. Conversely, depending on the architecrual configuration and the offered workload, for an additional cost of $\approx 3 - 5\%$, *rounds* can reduce latency[23] by nearly a *factor of five*. These improvements are achieved without (possible) scheduling optimizations within a round and with the worst case assumptions for the secondary storage requirements (see above).

In summary, our experiments indicate that in most cases *rounds* outperforms *buffered* and *simple*, often resulting in significant improvements. However, as is usually the case, one retrieval scheme is not absolutely better than another, but rather one must understand the issues and tradeoffs involved and choose a scheme accordingly, where the appropriate scheme is, to a large extent, a function of the architectral configuration used, the expected system workload (which is a function of the mix of applications expected to use the system), and the desired operating point.

References

[1] A. L. Chervenak. Tertiary Storage: An Evaluation of New Applications. *Ph.D. Thesis, UC Berkeley*, 1994.

[2] A. L. Drapeau and R. Katz. Striped Tape Arrays. In *Proc. of the 12th IEEE Symposium on Mass Storage Systems*, pages 257–265, Monterey, California, April 1993.

[3] L. Golubchik and S. Marcus. On Multilevel Multimedia Storage Systems. In *Proceedings of the 2nd Intl. Workshop on Multimedia Information Systems*, September 1996.

[4] L. Golubchik, R. R. Muntz, and R. W. Watson. Analysis of Striping Techniques in Robotic Storage Libraries. *Proc. of the 14th IEEE Symposium on Mass Storage Systems*, pages 225–238, September 1995.

[5] B. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of Sigmetrics*, pages 170–179, 1996.

[22] We do not give the comparison with *simple*, as the difference is even greater.

[23] Because *rounds* is significantly more efficient at processing workloads, it is not always simple to make the comparison by maintaining the same level of performance.

[6] B. Hillyer and A. Silberschatz. Storage Technology: Issues and Trends. June 1996.

[7] S. W. Lau, J. C. S. Lui, and P. C. Wong. A Cost-effective Near-line Storage Server for Multimedia System. In *Proceedings of the 11th International Conference on Data Engineering*, March 1995.