Scheduling and Data Replication to Improve Tape Jukebox Performance

Bruce K. Hillyer, Rajeev Rastogi, Avi Silberschatz

Bell Laboratories
700 Mountain Ave.
Murray Hill, NJ 07974
{hillyer,rastogi,avi}@bell-labs.com

Abstract

An increasing number of database applications require online access to massive amounts of data. Since large-scale storage systems implemented entirely on magnetic disk can be impractical or too costly for many applications, tape jukeboxes can provide an attractive solution. This paper shows how the performance of tape jukeboxes can be improved across a broad parameter space via a new scheduling algorithm and schemes for the placement and replication of hot data. We substantiate our claim by an extensive simulation study that quantifies the improvements obtained over a wide variety of workload characteristics. Our experiments suggest that system throughput increases when replicas of hot data are placed at the tape ends (not in the middle or at the beginning). As a result, the proposed replication techniques can be used to fill existing spare capacity in a tape jukebox, thus improving the performance of the jukebox "for free".

1. Introduction

A growing number of database applications require online access to large volumes of data. For example, NASA's earth observing system (EOS) will store more than a petabyte of data [4]; telecommunication service providers store terabytes of phone call data for billing, data mining, and fraud detection; and digital libraries and video-on-demand servers may also hold enormous volumes of data.

For huge data storage systems, tape-based hierarchical storage is superior to disk-only systems in terms of price, physical volume, and power consumption [8]. Archive reliability may also be superior for tapes, because they generally degrade in a detectable way over time rather than failing catastrophically. One disadvantage of tape systems compared to disk arrays is the high access latency for random tape retrievals – measured in minutes rather than the milliseconds of random disk seeks. The combination of disk-based secondary storage with tape-based tertiary storage is very effective for (a) huge storage systems that exceed practical limits on the number of disk drives, and (b) cost-limited systems in which the price, power, and physical space advantages of a tape library are compelling.

We show how to improve the bandwidth and access latency of tertiary storage systems by clever algorithms for retrieval scheduling, data placement, and data replication. The data placement and replication schemes capitalize on skewed data-access patterns to reduce the number of tape switches and the average positioning time. The amount of performance improvement depends on the workload, but with our best scheduling algorithm, throughput and latency improvements in the range of 10–30% can be achieved for sufficiently skewed access patterns (i.e., exhibiting sufficient distinction between hot and cold data).

The paper is organized as follows. Section 2 describes our tape and jukebox performance model, Section 3 presents scheduling algorithms for tape jukeboxes and states formal results, and Section 4 presents a simulation study of scheduling algorithms, data placement, and replication of hot data. Related work is described in Section 5, and concluding remarks are offered in Section 6. An extended version of the paper, containing proofs of formal results and significantly more detail on the algorithms and experiments is found at http://www.bell-labs.com/∼rastogi/

2 System Model

This section describes the organization of the tertiary storage subsystem, the model of tape positioning time, and the service model that is the basis for the scheduling algorithms studied in the next section.

A typical tertiary storage subsystem comprises a collection of tape drives accessing a large pool of tapes. In high-end mass storage systems this takes the form of a large tape

library that contains several drives and thousands of tapes. We study an organization consisting of multiple small jukeboxes, each having one drive (future work could extend this to multiple drives) and several tapes, because this organization has certain performance and cost advantages [6, 4]. The small jukeboxes are scheduled independently and are accessed concurrently. They hold relatively cold data—hot data are placed or cached in semiconductor memory, and warm data are on magnetic disks.

In this study, we assume that the tape drive can read an entire tape in a single forward pass, which is true for helical-scan tape technologies such as DAT/DDS, 8mm, and 19mm formats from Exabyte, Sony, and Ampex. The algorithms in this paper would need to be modified for serpentine tapes such as Travan, Quantum DLT, and IBM 3950. We assume that a tape drive must rewind a tape before ejecting it.

The unit of storage is a data block of fixed size. Blocks are stored on tape in physical *positions* that are consecutively numbered from 0 at the beginning of the tape. This numbering induces block relationships *higher* and *lower* and tape directions *up* and *down*. The *locate* operation repositions the tape head to a requested block. The *transfer time* is the time to read a block after it has been located.

2.1 Tape Positioning Model

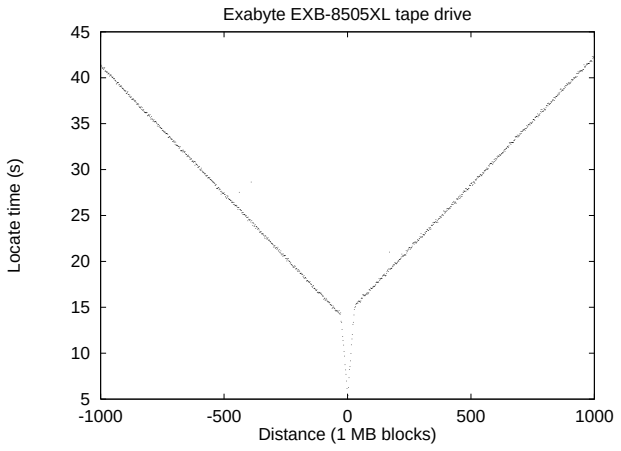


Figure 1. Locate Time as a Function of Distance (1 MB Logical Blocks)

For single-pass tape technologies such as helical-scan, the locate time is often modeled as a simple linear function of the difference in logical block numbers: a fixed startup time plus a term that is linear in the distance traversed [1]. Figure 1 illustrates our locate-time measurements on an Exabyte EXB-8505XL for a 1 MB block size. We see 4 linear functions, for short distance and long distance tape motion in the forward and reverse directions. Via a least squares linear fit over a set of 2130 random locates, we find that a forward locate past k logical blocks of size 1 MB takes $4.834 + 0.378k$ seconds for $k <= 28$, and $14.342 + 0.028k$ otherwise. A reverse locate past k logical blocks of size 1 MB takes $4.99 + 0.328k$ seconds for $k <= 28$, and $13.74 + 0.0286k$ otherwise. Locating to the physical beginning of tape takes an additional 21 seconds (the drive performs some overhead work whenever it fully rewinds). Averages over 2130 read-time measurements on the Exabyte EXB-8505XL indicate that reading k megabytes after a forward locate requires $0.38 + 1.77k$ seconds, and reading k megabytes after a reverse locate requires $1.77k$ seconds, although these measurements exhibit a significant variance. We estimate the time to switch the mounted tape in an Exabyte EXB-210 jukebox with the EXB-8505XL drive as 81 seconds: 19 seconds for the drive to eject the old tape, 20 seconds for the robotic arm to switch tapes, and 42 seconds for the drive to load the new tape and prepare for I/O.

The locate and read models were validated by comparing predictions with measurements in ten random walks on the tape, each random walk consisting of 100 locates and reads. Over these 10 trials, the largest error in the total predicted locate time was 0.6% and the mean error was 0.5%. For the read time, the largest error in the total predicted time was 4.6%, and the mean error was 2.6%.

Changing the locate, read, and tape switch functions to model a higher-performance system naturally improves the simulated system performance, but does not materially alter our results about choice of scheduling algorithm, the amount of replication, and the data placement.

2.2 Service Model

The unit of I/O is a data block of fixed size, where the blocksize is a parameter of this study. A data block may be replicated on multiple tapes, with at most one copy per tape.

A scheduling algorithm is specified by a *major rescheduler* that at tape switch time chooses a tape and forms a retrieval schedule, and an *incremental scheduler* that handles newly arriving requests, either scheduling them on the fly, or deferring them until the next invocation of the major rescheduler. Section 3 describes several scheduling algorithms. They form a retrieval schedule that can be executed in a single *sweep* over the tape, consisting of a *forward phase* (forward locates only) followed by a *reverse phase* (reverse locates only).

The service model repeatedly cycles through four steps.

1. Invoke the major rescheduler, which takes as input the pending list that contains all requests that have not yet been scheduled for retrieval. It selects the tape to service next, extracts the requests for that tape, and sorts those requests to form a retrieval schedule (termed the service list).

2. Switch to the selected tape if it is not already loaded.

3. Execute the service list by iteratively dequeuing the head of the list and reading the block at that tape position. If a new request arrives during the execution of the service list, invoke the incremental scheduler. It may insert the request into the service list, or defer the request by appending it to the pending list. The sweep ends when the service list becomes empty. At the end of the sweep, the tape head remains at the position of the last block retrieved, until the major rescheduler determines the necessity of a rewind and tape switch.

4. If the pending list is empty, wait for a request to arrive.

3 Scheduling Algorithms

This section describes simple algorithms that schedule retrievals from a tape jukebox, and proposes a more sophisticated algorithm. In Section 3.3, we state results showing the NP-hardness of this scheduling problem and bounding the deviation from optimality of the proposed algorithm.

3.1 Basic Scheduling Algorithms

The trivial scheduling algorithm is *FIFO*, which simply services requests in their order of arrival. This will give terrible performance for random requests to a tape jukebox because most retrievals will incur the overhead of a tape rewind, switch, and long locate. It would be much better to group the requests by tape, to amortize the tape-switch and locate overhead over multiple retrievals.

The *static* algorithms choose a tape to process, and form the service list by sorting all the requests for the selected tape from the pending list. During the execution of the service list, newly arriving requests are just appended to the pending list. The static algorithms only differ in the criteria by which the major rescheduler selects the next tape.

The term *jukebox order* denotes an arbitrary circular ordering on the tapes in a jukebox, such as "ascending order of slot number". The *effective bandwidth* of a schedule is the total number of bytes retrieved divided by the number of seconds to perform the retrieval—the time includes tape switch overhead (rewind, eject, robotic tape motion, and load) and schedule execution time (locate and read through the blocks in the service list). The locate and read times are calculated using the timing model of Section 2.1.

The static algorithms studied here are defined by the following tape selection policies. **Round robin** selects the next tape (in jukebox order after the currently mounted tape) that has a request in the pending list. **Max requests** chooses a tape having the maximal number of requests in the pending list, breaking ties by preferring the first one in jukebox order starting at the currently mounted tape. **Max bandwidth** is similar to Max requests, but it uses effective bandwidth rather than number of requests. **Oldest request, Max requests** chooses a tape that can satisfy the oldest request in the system (i.e., the request at the head of the pending list) by using the max requests method. **Oldest request, Max bandwidth** is similar, but it chooses by max bandwidth rather than by max requests.

The static algorithms defer every newly arriving request to the pending list, even if the request is for a block on the current tape. The corresponding *dynamic* algorithms insert such requests into the service list on the fly, provided that the requested block is ahead of the current position of the tape head in the existing schedule. Five dynamic algorithms are formed by combining this dynamic incremental scheduler with the five tape selection methods given above.

3.2 The Envelope-Extension Algorithm

The scheduling algorithms described in the previous section do not take a global view. They greedily service all possible requests from a tape, even if this impairs performance. For example, (see Figure 2) suppose that blocks A, B, C, and D are requested, and the tape head is at the beginning of tape 1. Notice that block D is replicated. After reading blocks A and B from tape 1, the algorithms described previously will proceed all the way to the end of tape 1 to fetch D, even though a better choice would be to retrieve the copy that follows C on tape 0.

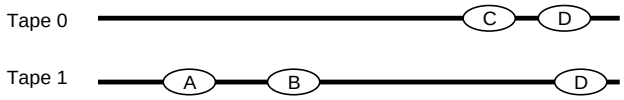


Figure 2. Tape jukebox simple example

The envelope-extension algorithm proposed in this section takes a global view. The basic idea is as follows. Consider the set of requests for non-replicated data blocks, and in particular, the distinguished non-replicated blocks that are farthest from the beginning of each tape (e.g., C and B in Figure 2). The collection of these distinguished blocks across all tapes defines an "envelope"—a set of tape prefixes that must be traversed. In general, this envelope may already contain some requested blocks that are replicated—retrieving these blocks does not require any additional portion of the tape to be traversed. To satisfy requests such as D that have no replicas inside the envelope, the envelope is incrementally extended to an "upper envelope" that satisfies all requests. In figure 2, extending the envelope from C to D is clearly less expensive than extending the envelope from B to D. The full algorithm is combined with a policy such as "max bandwidth" to choose which tape to visit first.

At tape switch time, the envelope-extension algorithm performs three jobs. First it calculates an upper envelope

for all tapes using all the requests in the pending list. Then it chooses a tape. Finally it extracts from the pending list all requests that can be satisfied by the portion of the chosen tape that lies within the upper envelope, and sorts those requests by position to form the service list. The upper envelope is calculated as follows.

1: Form initial envelope. For each tape, the initial envelope is the position of the tape head after reading the highest non-replicated request (if any), or 0 otherwise. If the current head position for the currently mounted tape is not inside the initial envelope, the envelope is extended to include it.

2: Absorb unscheduled requests that are inside the envelope. Schedule each request that can be satisfied by some block inside the current envelope. If more than one replica for a request is inside the envelope, choose the currently-mounted tape if possible, or the tape having maximal number of scheduled requests that is first in jukebox order after the currently mounted tape.

3: Calculate incremental bandwidths. The remaining unscheduled requests lie outside the envelope. For each tape, form an "extension list" consisting of the unscheduled requests that can be satisfied by this tape, sorted by position. Now calculate an "incremental bandwidth" for each prefix of the extension list as follows. If the prefix is for the currently mounted tape, or for a tape for which the envelope is greater than 0, the incremental time cost of extending the envelope to include this prefix is the total of the locate and read times, starting from the current envelope, through all the requests in this prefix, plus the time to locate back to the position of the current envelope. If the prefix is for a tape whose envelope currently is 0 and this is not the currently mounted tape, an additional cost is incurred for eject, tape switch, and load. The incremental bandwidth of each prefix is defined to be the number of bytes retrieved by the prefix divided by the incremental time cost of the prefix.

4: Extend envelope. Choose the prefix having the highest incremental bandwidth, breaking ties by preferring the tape having the most number of scheduled requests inside the envelope, and then by using jukebox order. Extend the envelope to include the chosen prefix, thereby scheduling the requests in the chosen prefix.

5: Shrink envelope. It can happen that some replicated block that is currently scheduled at the outer edge of the envelope on tape a is also contained in the portion of tape b just enclosed by the envelope extension. In this case, alter the schedule to satisfy the request from the replica on b, and shrink the envelope on a back to the preceding request in a's schedule. If more than one tape can contract its envelope in this way, first shrink the tape having the fewest number of scheduled requests, and break ties by choosing the tape lowest in jukebox order. Repeat this envelope-shrinking step until the envelope can shrink no further.

6: Iterate. If unscheduled requests remain, go to Step 3.

When the preceding steps terminate, the resulting envelope is the upper envelope. Now choose a tape. For each tape, consider the set of all requests that can be satisfied by that tape within the upper envelope (in general, a superset of the per-tape schedule constructed above). Apply the tape switch policy (i.e., oldest request, or max requests, or max bandwidth) to these sets of requests. The resulting envelope algorithms are named oldest request envelope, max requests envelope, and max bandwidth envelope.

For the envelope-extension algorithm, the incremental scheduler is a modification of the dynamic incremental scheduler described earlier. If the request can be satisfied by the current tape while remaining within the upper envelope, it is inserted into the forward or reverse phase of the current sweep as before. Otherwise the envelope-extension algorithm (steps 3–5) is applied to calculate which tape and position should be used to satisfy the new request. This may extend or shrink the upper envelope, modifying the pending list and service list accordingly. If the resulting schedule would not retrieve the request from the current tape, then the request is just appended to the pending list.

3.3 Complexity Results

We now briefly state the time complexity of the envelope-extension algorithm, and the theoretical results that characterize the difficulty of obtaining an optimal schedule (i.e., a schedule with minimal total execution time), and that bound the distance from optimality of the envelope-extension schedule. These matters are discussed rigorously and at length in the extended version of the paper, available via http://www.bell-labs.com/∼rastogi/

First we state that the time complexity of the major rescheduler is dominated by the complexity of steps 3–5. Let the number of requests be n, and the number of tapes be t. Then the complexity can be shown to be $O(n^2 \cdot t^2)$, because of at most n iterations (each iteration schedules at least 1 request), with the cost per iteration dominated by step 5 (at most n requests and t replicas $\times$ t tapes).

We next state a result showing that this scheduling problem is intractable. Let $C(S)$ denote the time cost of a schedule S (as made rigorous in the extended version of this paper). Let S_1 be the schedule at the end of step 2, and S_2^{opt} be an optimal extension of S_1 that satisfies all the requests.

Theorem 1: The problem of determining whether there exists an extension S_2 of S_1 containing all the requests remaining unscheduled at the end of step 2 of the major rescheduler, such that $C(S_2) - C(S_1) \leq c$, is NP-hard. □

Although the final schedule S_2 at the end of step 6 of the major rescheduler may be sub-optimal, it can be shown that the additional cost of S_2 over S_1 is within a harmonic factor of the additional cost of S_2^{opt} over S_1. This result is a consequence of the following theorem.

Theorem 2: Let S_1 be the schedule at the end of step 2 and S_2 be the final schedule generated by the major rescheduler. If n is the number of requests remaining unscheduled at the end of step 2, then $C(S_2) - C(S_1) \leq H_n \cdot (C(S_2^{opt}) - C(S_1)) - n \cdot (H_n - 1) \cdot (C_s + C_r) + n \cdot C_d$, where C_s is the start up cost for a short distance forward locate, C_r is the time to transfer data for a block, C_d is the difference between the start up costs for short and long distance forward locates, and $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} harmonic number. □

4 Simulation Results

The simulation studies in this paper cover a broad territory, in terms of characteristics of the data access workload, the data layout on tape, and the algorithms for retrieval scheduling. The major parameters studied are the request arrival process, hot/cold skew in the request stream, the I/O transfer size, the scheduling algorithm, the placement of hot data, and replication of hot data. Rather than presenting raw measurements for a 6-dimensional parameter space, we distill the results to obtain answers for eight questions.

1: What I/O transfer size should be used?
2: What is a good scheduling algorithm (no replication)?
3: Where on the tapes should hot data be placed (no replication)?
4: How many replicas of hot data should be stored in a jukebox?
5: Where should hot data and replicas be placed (replication)?
6: What is a good scheduling algorithm (replication)?
7: How important is the amount of skew in the request stream?
8: Is it cost effective to replicate hot data?

The answer to each question is illustrated by a graph that explores the effect of a few parameters when the remaining parameters are fixed at their "best" values. Many of these graphs are *parametric* graphs. In such a graph the independent variable is not shown on either axis. As the independent variable changes, it traces out a curve in two output variables. For instance, in Figure 4 the independent variable is the queue length, and the output variables shown on the axes are the mean throughput and the mean delay. A second independent variable can be set to a few different values to generate a family of curves. In most of our parametric graphs, the independent variable that traces out each curve is the intensity of the request load, and the independent variable that gives rise to a family of curves is the variable under study, such as the choice of scheduling algorithm, or the placement of hot data, or the amount of skew in the request stream.

The following workload assumptions define the scope of this study. First, the workload consists of random logical block reads. (Writes would be directed to disk-resident delta files, occasionally written to tape during idle time or piggybacked on the read schedule.) Second, the requested logical block numbers are independent of one another. (We do not exploit performance gains from clustered or Markov-type data dependencies.) Third, we study the effect of skew (non-uniform access probabilities) in the block requests seen by the jukebox. We use a simple hot/cold model of skew that is characterized by two parameters: the percent of tape-resident data that are hot (PH), and the percent of tape requests directed to hot data (RH). A hot request selects one of the hot blocks with a uniform random distribution. A cold request selects one of the cold blocks with a uniform random distribution.

Each request is for one logical block. The logical block size is a parameter of the study, and is examined in Section 4.1. The mapping between the logical block number and the physical location on some tape is a consequence of the layout and placement policies, which are parameters studied in sections 4.2–4.8.

Two request generation scenarios are considered. The first scenario considers a fixed number of I/O-bound processes that are accessing a storage system. This is studied via a closed-queuing simulation model that maintains a queue of requests, and that generates a new request immediately after the completion of each request, thus maintaining a constant request queue length. (Note that any improvement to the service rate causes a corresponding increase in the request generation rate, thereby increasing the system throughput.) The second scenario involves a storage system serving a large pool of clients that make sporadic requests. This is studied via an open-queuing simulation that models the request stream as a Poisson arrival process with some relatively large mean interarrival time between requests. In this case, the rate of request arrivals is largely independent of the service rate, so a slow server will cause a lengthy queue of outstanding requests to accumulate.

The intensity of the workload is set by the queue length in the closed-queuing model, and by the mean interarrival time in the open-queuing model. The two models give qualitatively similar answers to the eight questions studied (with a few interesting differences noted in the text), so we show graphs only from closed-queuing experiments. Each experiment involves numerous simulation runs that model 10 million seconds of jukebox activity using the tape model described in Section 2.1. Similar to the Exabyte EXB-210 tape library with an EXB-8505XL drive, we assume that a tape jukebox has 10 tapes each with a capacity of 7 GB.

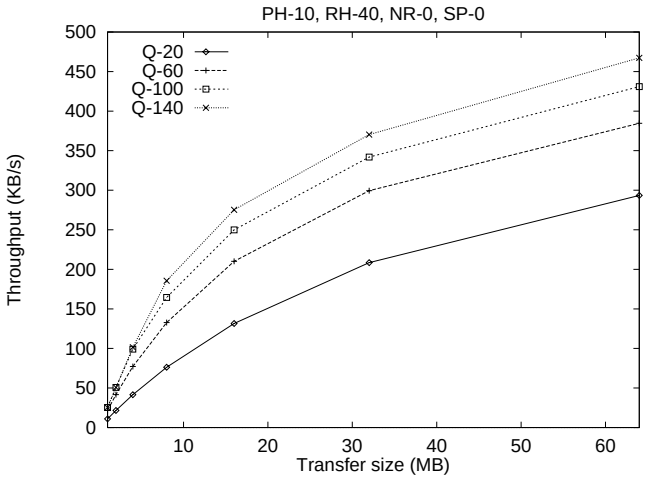


Figure 3. The Effect of Transfer Size

4.1 The Effect of Transfer Size

The first workload factor studied is the I/O transfer size. The transfer size is an important design characteristic for a random-access workload—too small a transfer size causes the effective

transfer rate of the tape jukebox to become so small that the processor starves for data upon which to compute.

The graph of Figure 3 shows the average throughput (KB per second) of a jukebox containing an Exabyte EXB-8505XL tape drive as a function of the transfer size. Curves are shown for four different workload intensities (queue length 20, 60, 100, 140). The performance is based on a good scheduling algorithm (the dynamic max-bandwidth algorithm—see Section 4.2) with moderate skew in the request stream. The notation above the graph describes the values of four parameters, as follows. PH-10 means 10% of the logical blocks are hot, RH-40 means 40% of the requests are directed to hot data, NR-0 means that the number of replicas of hot blocks is zero, SP-0 (Start Position 0) means that the hot data are placed starting at the beginning of the tape (Section 4.3 describes and evaluates this data placement).

From the experiments shown in this graph and numerous others that investigate issues of skew, we can assert that the I/O size should be at least 16 MB— this enables our scheduling algorithm to obtain an effective transfer rate above 30% of the streaming transfer rate of the drive. All subsequent experiments in this paper will use a 16 MB data block size. Below 16 MB the effective data rate of the system falls off very rapidly. For instance, Figure 3 illustrates that reducing the request size from 16 MB to 8 MB costs nearly a factor of 2 in performance—the positioning time per request remains about the same, but the amount of data transferred per request is divided by 2. This answers Question 1. It is qualitatively independent of the skew and intensity of the workload, and remains true for a wide range of tape drive streaming data rates and positioning speeds.

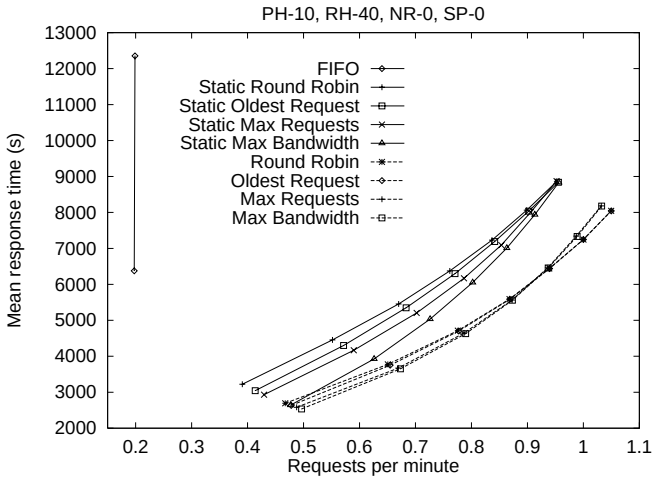


Figure 4. Relative Performance of Scheduling Algorithms (No Replication)

4.2 Scheduling Without Replication

Figure 4 shows a graph depicting throughput/delay curves for nine scheduling algorithms. From the notation above the graph it is seen that 10% of the data are hot, 40% of the requests are directed to hot data, the number of replicas is 0, and hot data are placed at the beginning of the tape.

The curve for the FIFO algorithm is a vertical line at the left of the graph. A longer queue does not improve the FIFO service rate (each retrieval switches to a random tape and positions to a random block), so longer queues create proportionally longer response times.

The other curves show increasing throughput and delay as the queue length increases. The plotted points, moving from left to right along a curve, correspond to queue lengths of 20, 40, 60, ..., 140. The static algorithms are generally inferior to the dynamic algorithms both in throughput and delay. But for a light workload (small queue size), it is unlikely that a new request for a tape will arrive during the execution of the retrieval schedule from that tape, so the static max-bandwidth algorithm is comparable to the dynamic algorithms. At heavy workloads, the dynamic algorithms are significantly better. Among the dynamic algorithms, low to moderate workloads favor the greedy policies of switching to the tape with highest bandwidth or most number of requests. Heavy workloads favor the fair tape switching policies of round-robin and oldest request, which tend to prevent unlucky requests from incurring excessive delays waiting for their tape to be processed.

The results are qualitatively similar for a wide variety of skews and placements of hot data. The results are also qualitatively similar for the open-queuing model, with one exception: At high workloads, the choice of algorithm only affects the delay. It has little effect on the throughput in the open-queuing model because faster service doesn't cause faster generation of new requests.

So the answer to Question 2 is that the dynamic max bandwidth algorithm is good for all workloads when data are not replicated, and the simpler max requests algorithm is nearly as good.

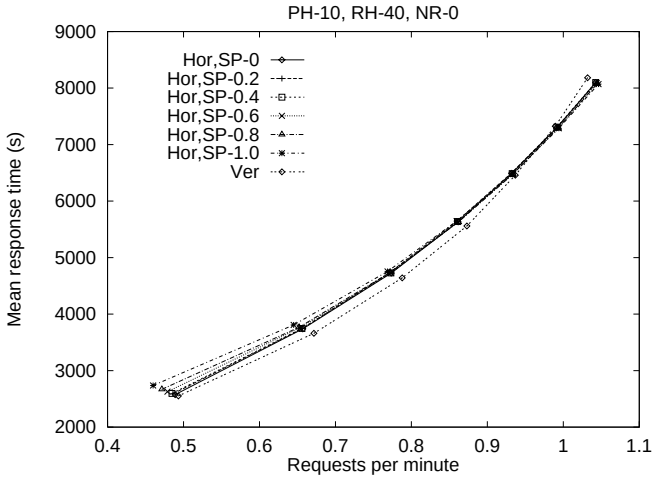


Figure 5. Throughput and Latency as a Function of Hot Data Placement (No Replication)

4.3 Placement of Hot Data

The next experiment addresses the issues of layout and placement of hot data in the case of no replication. With respect to the layout, one alternative is to collect all the hot data onto one tape (a "vertical" layout), and the other alternative is to distribute the hot

data over multiple tapes (a "horizontal" layout). The placement issue concerns where hot data should be located within a tape—at the beginning of the tape, at the end, or somewhere between.

Figure 5 shows a graph that has a family of curves for various horizontal placements of hot data ranging from the beginning of the tape to the end, and one curve for a vertical layout. "SP" is a normalized variable indicating the position of the hot data: SP-0 means the hot data are placed at the beginning of the tape, and SP-1.0 means at the end of the tape. From the notation above the graph it is seen that 10% of the data are hot. Since the jukebox contains 10 tapes, this means that for a vertical layout, one tape is completely filled with hot data, so there is no choice of placement. We did not experiment with quantities of hot data larger than the capacity of one tape, but we suspect that a vertical layout in that case would lead to excessive tape switching. The schedules are from the dynamic max-bandwidth algorithm—the best algorithm from the previous experiment.

The graph indicates that a vertical layout is the best choice except for very intense workloads that cause too-frequent switches to the hot tape, hence long waiting times for cold requests. Given a horizontal layout and no replication, the hot data should be placed at the beginning of the tapes, thereby decreasing the mean locate time. Thus, we have answered Question 3, and the answer remains true for open queuing and for a wide range of skews.

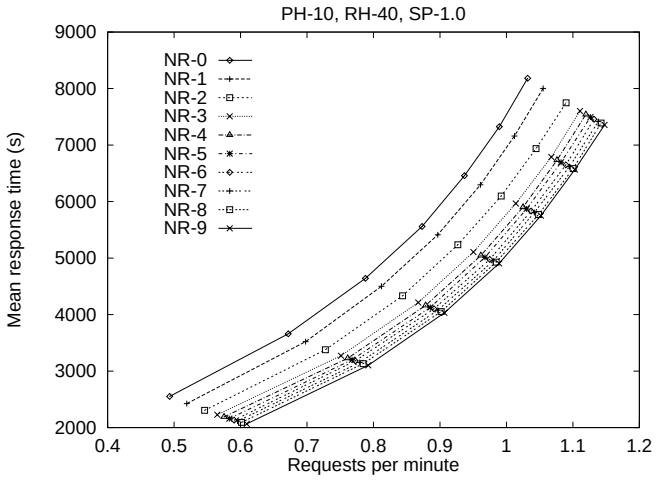


Figure 6. Throughput and Latency as a Function of Number of Replicas of Hot Data

4.4 Replication of Hot Data

Should hot data be replicated? It seems intuitive that placing copies of hot items on more than one tape will increase the number of items that can be retrieved from any given tape, thus reducing the frequency of tape switching, and decreasing the expected locate time between items retrieved from a single tape.

Figure 6 shows a graph for moderate skew that displays a family of curves as the number of replicas varies from 0 to 9. Replicas are placed at the end of the tape because this placement is better when data are replicated (see Section 4.5 below). The curves in the graph are for vertical placements, i.e, placements in which one tape contains all the hot data, and replicas are distributed round-robin across the remaining tapes. Similar results are obtained for horizontal placements (in which both hot data and the replicas are distributed across all the tapes).

The replication of hot data across the tapes means that fewer cold items can fit on each tape. This causes a small increase in the probability that a tape switch will be required to satisfy each cold request. The resulting performance loss is more than compensated for by the performance improvement in satisfying hot requests.

From the graph it is uniformly the case that more replicas give better performance. In terms of requests per minute, an increase of about 18% is obtained by going from no replication to full replication; similarly, full replication can result in improvements in response time as high as 13%. These improvements are mainly attributed to a 20% decrease in the number of tape switches as a result of fully replicating hot data. We see diminishing returns with increasing numbers of replicas.

Open queuing shows a similar benefit from replication, with the same exception seen in Section 4.2: with open queuing at high workloads only the latency improves, the throughput does not. Thus, the answer to Question 4 is that for best performance one should prefer to have a replica of each hot item on every tape.

The performance improvement from replication has an associated cost; this tradeoff is discussed in Section 4.8.

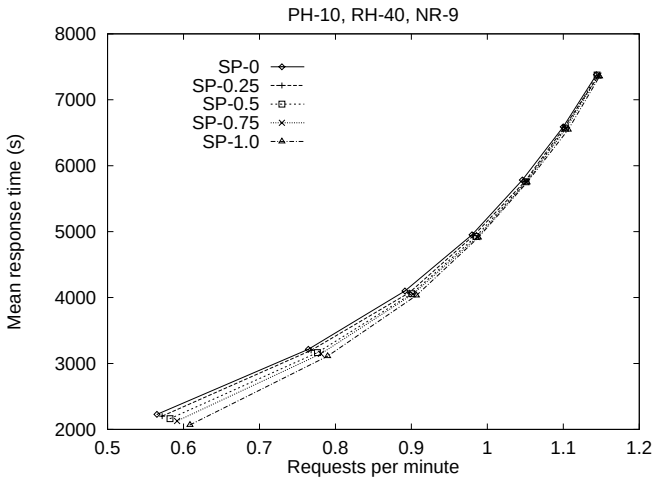


Figure 7. Throughput and Latency as a Function of Replica Placement

4.5 Placement of Replicas

Where should replicas of hot data be placed within a tape? Figure 7 shows a graph for moderate skew and full replication (original plus 9 copies of each hot item in a 10-tape jukebox) that displays a family of curves as the placement of replicas ranges from the beginning of tape (SP-0) to the end of the tape (SP-1.0).

It is seen that when replicating, hot data and replicas should be placed at the end of the tape (giving throughput and response time improvements of 4% and 3%, respectively). This contrasts with

the no-replication case of Section 4.3, in which the hot data should be placed at the beginning of the tape. We explain this difference is as follows. In the case of full replication, a significant portion of each tape is filled with replicas—nearly half if 10% of the data are hot in a 10 tape jukebox. If 40% of the requests are directed to hot data, placing the hot data and replicas at the end of the tape means that 40% of the requests are directed to the second half of the tape, and 60% of the requests are directed to the first half of the tape, decreasing the average locate and rewind distance. Furthermore, it is not necessary to locate into the second half of every tape: full replication makes it possible to retrieve hot data only when there are enough pending hot requests to amortize the time cost of locating into the second half of the tape and rewinding back.

In additional experiments that explore this question for varying amounts of skew and numbers of replicas, some "rare" intermediate values of skew and replication give marginally better performance when the replicas and hot data are placed at an intermediate position in the tape. These parameter settings are "rare" in that with no replication the best placement for hot data is at the beginning of the tape, and as the amount of replication and skew increases, the best placement very quickly moves to the end of the tape. This result holds for both open and closed queuing.

Thus, the answer to Question 5 is that the hot data and replicas should be placed at the end of the tapes if the hot data are replicated (either partial or full replication). If the tape jukebox has excess capacity, storing replicas of hot items at the ends of tapes will improve the performance "for free".

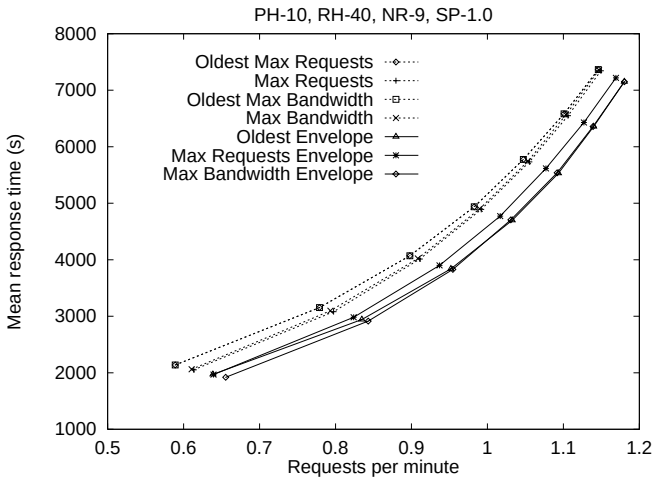


Figure 8. Relative Performance of Scheduling Algorithms With Replication

4.6 Scheduling With Replication

Figure 8 shows a graph depicting throughput/delay curves for a variety of algorithms given full replication at the end of the tape and moderate skew.

The globally optimized schedules produced by the envelope algorithms are superior. Max-bandwidth envelope can give throughput and response time improvements of 6% and 5% respectively over the (non-envelope) dynamic max bandwidth algorithm. The relative performance remains similar for a wide variety of skews, placements, and amounts of replication, so the answer to Question 6 is that the max-bandwidth envelope algorithm is the best choice when hot data are replicated. When there are no replicas, this algorithm degenerates into the dynamic max-bandwidth algorithm (the best choice for no replication) so we can say that the max-bandwidth envelope algorithm is always preferred.

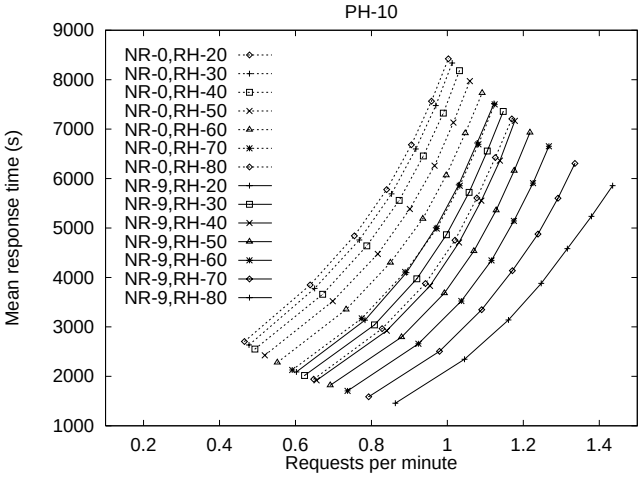


Figure 9. The Relationship Between Skew and Performance Improvements

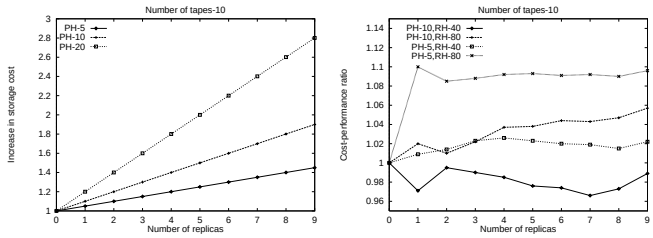
4.7 Importance of Skew

Figure 9 shows a graph for 10% of data hot that indicates how the throughput and delay change as the percent of requests to hot data varies from 20% to 80%. Curves drawn with dotted lines are for the non-replicated case, and curves drawn with solid lines are for full replication of hot data. The best data placements are chosen, namely at the beginning of the tape for no replication, and at the end of the tape for full replication. The best algorithm (max-bandwidth envelope) is used.

The graph shows that increasing skew is uniformly associated with better throughput and delay. It also gives an illustration of the performance advantage obtained by replication. For a wide range of skews and workloads, full replication improves the throughput and response time up to 25% and 19%, respectively, over no replication. As in previous experiments, given an intense workload in the open-queuing model, more skew is associated with improved response time but constant bandwidth.

4.8 Cost Effectiveness of Replication

With increasing replication of hot data, decreasing amounts of jukebox capacity remain for the storage of cold data. Thus improving performance via replication has an associated cost. This section first analyzes the storage overhead, then introduces the notion of the cost-performance ratio to compare systems, and finally presents results concerning the cost-performance ratio of replication compared with non-replicated tape jukeboxes.



(a) (b)

Figure 10. Cost-Performance

The storage overhead due to replication is proportional to the number of replicas of each hot item, and to the fraction of data that are hot. In general, the storage for PH% hot data with NR replicas grows by an expansion factor $E = 1 + \text{NR} \times \text{PH}/100$. Figure 10(a) shows this expansion factor as a function of both the number of replicas and the percent of data that are hot.

The cost-normalized performance ("cost-performance" for short) of a system is the ratio of some performance measure to the total dollar cost of the system. The performance metric considered here is the throughput in bytes per second. We assume that the cost of a farm of n tape jukeboxes equals cn where c is the cost of one jukebox, so the cost-performance of a farm of n jukeboxes is simply the ratio of aggregate throughput T to aggregate cost cn, i.e., T/cn. (A farm of tape jukeboxes grows in capacity one jukebox at a time, so the storage cost as a function of capacity is discontinuous. Our smooth approximation suffices for the conclusions that we draw at the end of this section.)

We define the *cost-performance ratio* of schemes a and b to be the ratio $\frac{T_a/cn_a}{T_b/cn_b}$, i.e., $\frac{T_a/n_a}{T_b/n_b}$. But T_a/n_a is just the performance per jukebox of scheme a, so the cost-performance ratio of schemes a and b is simply the ratio of their performance per jukebox. Suppose a is a replication scheme, and b is the non-replicated scheme. If the cost-performance ratio is greater than 1, then the additional performance due to replication offsets the additional storage cost.

We assume that the total workload applied to a farm is spread evenly over the jukeboxes. Since a replicated scheme uses E times more jukeboxes than a non-replicated scheme, the workload of requests applied to each jukebox in the replicated farm is smaller by a fraction $1/E$. This is significant because the simulation results presented in earlier sections show that the throughput per jukebox decreases when the queue is smaller.

Figure 10(b), depicts the cost-performance ratio of replication vs. no replication as the number of replicas increases from 0 to 9 in a 10 tape jukebox. The curves show the cost-performance ratio for four different skews. The workload assumes a queue length of 60 per jukebox in the non-replicated case, and thus $60/E$ in the replicated case.

The graph indicates that for workloads with moderate skew in a 10 tape jukebox, replication degrades the cost-performance ratio by as much as 3%—the space consumption expands faster than the performance improvement. The graph also shows that in case of a very high skew, replication gives a cost-performance benefit of 8% with as few as 2 replicas, and 10% with full replication. With a less intense workload (queue length 20), full replication has a cost-performance advantage that reaches 14% in the very high skew case. Additional experimentation based on jukeboxes holding 5 tapes rather than 10 show similar results. We conclude that although replication improves performance, it improves the performance per dollar only in high-skew cases.

Suppose that instead of replicating hot data within each jukebox, the space that would have been occupied by replicas is simply left empty—the data blocks are packed into as few tapes as possible within each jukebox, using a vertical layout that separates the hot data onto a single tape. Numerous experiments indicate that the cost-performance of this scheme is within one or two percent of the non-replicated scheme. Thus it may be desirable to use existing spare capacity in a jukebox to hold replicas, thereby obtaining a performance improvement "for free".

The cost-performance tradeoff can be summarized as follows. Using space in a jukebox to hold replicas of hot data will improve both the performance and the cost-performance if the workload exhibits a high skew. If the workload exhibits moderate or low skew, the tradeoff is that replication will improve the performance, but may degrade the cost-performance by a few percent. With existing spare capacity, one can improve retrieval latency significantly, and (particularly in a closed-queuing model) also the throughput by filling the spare capacity with replicas of hot data.

Thus, we offer the following recommendation. When gradually filling a jukebox with data, use one tape to hold a small percentage (say 10%) of data that are hottest, because the vertical layout of hot data is preferred. When writing other tapes, fill them only part way with original data, and append replicas of hot data to the ends of these tapes (per Section 4.5) when convenient. The performance overhead of this tape writing may be small if it is piggybacked on a schedule that already loads the tape to satisfy a schedule of read requests. When the jukebox nears overflow, the tape dedicated to hot data can be overwritten with base data. This incurs little loss in performance because a horizontal layout with full replication is nearly as good as a vertical layout with full replication. Finally, the space given to replicas at the ends of the tapes can be recaptured by overwriting the replicas with base data.

5 Related Work

Several recent articles describing tape technology and schemes to improve tape performance have appeared [10, 11, 9, 7, 8, 5, 12, 16, 2, 1, 13, 14, 15], but none of them study replication to increase tape performance, and only [14, 15] address the scheduling of requests over multiple tapes in a jukebox.

In [16], query pre-execution is used to determine the set of tape blocks to be retrieved from a single tape so that scheduling can minimize random tape accesses while returning data in the order of request. In [13], algorithms for joining a tape-resident relation with one on disk are presented. In [5, 2, 1], tape striping is used to improve tape data transfer rates and response times for large requests, and techniques for overcoming striping-related problems like synchronization and increased number of tape switches are discussed. The authors of [8] propose and evaluate schemes for scheduling random I/Os on a single serpentine tape.

In [14], the authors present policies for fetching and evicting relation *fragments* involved in queries to reduce tape switches and seek times. The problem of retrieving fragments is complex, because the cache may not be large enough to store all the fragments

for the queries, and dependencies between various fragments induced by queries may require related fragments to be in the cache at the same time. In [15], the execution of subqueries in a query plan is dynamically reordered to minimize the retrieval overhead.

The problem of optimal data placement for different tape library technologies is studied in [3]. For tapes that rewind to the nearest zone, it is shown that the optimal placement of objects within each tape is the one that performs an organ-pipe arrangement of the object access probabilities; that is, placing hot objects toward the middle of the tape is the best strategy. In contrast, for tapes that rewind to the beginning before ejecting, placing the hot objects toward the start of the tape is the optimal policy. This is consistent with our experimental results in Section 4.3 where for horizontal layout and no replication, placing hot data at the beginning of tapes resulted in the best performance.

6 Concluding Remarks

As a step toward the effective use of tertiary storage for databases, this paper studies ways to improve the efficiency of data retrieval from a tape jukebox. A sophisticated scheduling algorithm is proposed, and it is compared with several simpler static and dynamic algorithms. The problem of scheduling retrievals in this setting is NP-hard, but we bound the maximum deviation of the proposed algorithm from optimal. Numerous simulation experiments compare the performance of the scheduling algorithms, and quantify the performance benefits of schemes for replication and data placement. The experiments cover a broad parameter space involving workload aspects such as skew, I/O size, and the intensity of the request arrival process. The simulation results are qualitatively independent of the particular bandwidth and capacity of the tape system modeled, but have been obtained under the assumption of a single-scan technology, such as helical-scan tape.

The principal conclusions supported by the experimental results are as follows. The proposed envelope-extension scheduling algorithm gives significantly better performance than the simple algorithms. Replication of hot data across the tapes in a small jukebox improves the performance. Replicas of hot data should be placed at the tape ends (not in the middle or at the beginning). For high skews, increasing the aggregate storage capacity in order to make room for replicas will improve the cost-performance by several percent, but for low skews, replication improves performance while degrading the cost-performance by a few percent. The proposed replication techniques can use existing spare capacity to improve the jukebox performance "for free".

References

[1] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD dissertation, University of California at Berkeley, Computer Science Department, Dec. 1994. Technical Report UCB/CSD 94/847, http://www.cc.gatech.edu/fac/Ann.Chervenak/papers/ChervenakThesisNov94.ps.Z.

[2] T. Chiueh. Performance optimization for parallel tape arrays. In *Proceedings of the 1995 International Conference on Supercomputing*, pages 385–394, Barcelona, Spain, July 3–7 1995. ACM Press.

[3] S. Christodoulakis, P. Triantafillou, and F. A. Zioga. Principles of optimally placing data in tertiary storage libraries. In *Proceedings of the 23rd International Conference on Very Large Databases*, pages 236–245, Athens, Greece, Aug. 1997. Morgan Kaufmann, San Francisco.

[4] F. Davis, W. Farrell, J. Gray, R. Mechoso, R. Moore, S. Sides, and M. Stonebraker. EOSDIS alternative architecture. Technical Report Sequoia 2000 TR 95/61, University of California at Berkeley, Apr. 1995. http://www.research.microsoft.com/research/BARC/Gray/EOS_DIS/default.htm.

[5] L. Golubchik, R. R. Muntz, and R. W. Watson. Analysis of striping techniques in robotic storage libraries. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, pages 225–238, Monterey, CA, Sept. 11-14 1995. IEEE Computer Society Press.

[6] J. Gray. Superservers. *http://www.research.microsoft.com/research/BARC/Gray/SuperServers.doc*, 1994.

[7] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 195–204, Montreal, Canada, June 3–6 1996.

[8] B. K. Hillyer and A. Silberschatz. Storage technology: Status, issues, and opportunities. Technical report, Bell Laboratories, Mar. 1996. http://www.bell-labs.com/∼hillyer/papers/tsurv.ps.gz.

[9] B. K. Hillyer and A. Silberschatz. Scheduling non-contiguous tape retrievals. In *Sixth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 113–123, College Park, MD, Mar. 1998.

[10] T. Johnson and E. L. Miller. Benchmarking tape system performance. In *Sixth Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 95–112, Mar. 1998.

[11] T. Johnson and E. L. Miller. Performance measurements of tertiary storage devices. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 50–61, New York, Aug. 1998. Morgan Kaufmann, San Francisco.

[12] R. H. Katz, T. E. Anderson, J. K. Ousterhout, and D. A. Patterson. Robo-line storage: low latency, high capacity storage systems over geographically distributed networks. Technical Report Sequoia 2000 TR S2K-91-3, University of California, Berkeley, Oct. 1991.

[13] J. Myllymaki and M. Livny. Disk-tape joins: synchronizing disk and tape access. In *Proceedings of the 1995 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 279–290, Ottawa, Canada, May 15–19 1995. ACM Press.

[14] S. Sarawagi and M. Stonebraker. Query processing in tertiary memory databases. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 585–596, Zurich, Switzerland, Sept. 11–15 1995. Morgan Kaufmann, San Francisco.

[15] S. Sarawagi and M. Stonebraker. Reordering query execution in tertiary memory databases. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 156–167, Bombay, India, Sept. 1996. Morgan Kaufmann, San Francisco.

[16] J. Yu and D. J. DeWitt. Query pre-execution and batching in paradise. Technical report, University of Wisconsin, Madison, Mar. 1996.