Scheduling Queries on Tape-resident Data

Sachin More, Alok Choudhary

Technical Report No. CPDC-TR-2000-01-001

January 31, 2000

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston IL-60208

Scheduling Queries on Tape-resident Data

Sachin More Alok Choudhary

Department of Electrical and Computer Engineering
Northwestern University

Abstract

Advances in storage technology have made near-line tertiary storage a viable alternative for database and data warehouse systems. Tertiary storage systems are employed in cases where secondary storage can not satisfy the data handling requirements or tertiary storage is more cost effective option. Tertiary storage devices have traditionally been used as archival storage. The new application domains require on-demand retrieval of data from these devices. This paper investigates issues in optimizing I/O time for a query whose data resides on automated tertiary storage containing multiple storage devices. We model the problem as a limited storage parallel two-machine flow-shop scheduling problem with additional constraints. Given a query, we establish an upper bound on the number of storage devices for an optimal I/O schedule and provide experimental proof for it. For queries that access small amounts of data from multiple media, we derive an optimal schedule analytically. For queries that access large amount of data we derive a heuristics based scheduling algorithm using analytically proved results. We also discuss practical aspects of about using the theoretical results in a real environment and demonstrate our applicability of our work using an accurate robotic tape library simulator.

1 Introduction

Advances in storage technology has made near-line tertiary storage a viable alternative for database and data warehouse systems [Inm]. Tertiary storage systems are employed in cases where secondary storage can not satisfy the data handling requirements (for example, amount of data is too large) or tertiary storage is more cost effective option [HS96b]. NASA's Earth Observing System (EOS) Data and Information System (EOSDIS) is an example of the former [FPS96, KBCH95]. It collects about 0.6 terabytes of data daily and will run for about 15 years. Clearly such large amount of data can not be stored on secondary storage. The other case, where tertiary storage is more cost effective than secondary storage, can be found in data warehousing applications. [Inm] shows that substantial monetary savings can be achieved using a hierarchical data storage containing comparatively small amount of secondary storage and vast amounts of tertiary storage. These savings can be gained without sacrificing performance, sometimes even gaining in performance.

Tertiary storage devices have traditionally been used as archival storage. The new application domains require on-demand retrieval of data from these devices ([HS98]). These applications access a *slice* of the dataset. The data belonging to this slice may not be stored contiguously on a single media. This is in contrast to data archiving applications which access large chunks of contiguous data. When data needed by application lies scattered on multiple media, correct scheduling of data retrieval requests becomes important. For example, consider a query that accesses data from two different media using a single robotic arm and two tape drives. The total I/O time of the query is minimized when the tape that needs more time to read is loaded first.

1.1 Motivating example

Many of the application domains that use tertiary storage access a *multidimensional dataset*. In a multidimensional dataset, each data item occupies a unique position in a n-dimensional hyper-space. A query selects a subset of the data items by selecting a subset of the domain in each dimension. Given the wide variety of expected queries ([OLA98]) it is not possible to store data accessed by each query contiguously without high amount of data replication. Hence, a query on a multidimensional dataset stored on tertiary storage, accesses data from multiple media ([MC99]). The answer set for a query consists of a set of two-tuples, media id and a block number inside the media. The I/O time of a query is determined not only by the amount of data it accesses, but the location of the data as well. Time needed to read the query data from a media depends on the amount of data and location of the data inside the media. When using automated tertiary storage like robotic tape libraries, the total I/O time for the query is influenced by the order in which the media are accessed. ■

Various issues in tertiary storage management have been addressed by the database community. [CHL93] evaluates issues in extending database technology for storing/accessing data on tertiary storage. [Sto91] proposes a database architecture that uses hierarchical storage. [Sar95a, Sar95b, ML95, SS96] examines issues in query processing when data resides on tertiary storage. Data striping on tertiary storage has been evaluated in [DK93, GMW95]. Tertiary storage space organization issues are addressed in [CDK$^+$95, HRS99]. This paper investigates issues in optimizing I/O time for a query whose data resides on automated tertiary storage containing multiple storage devices. We model the problem as a limited storage parallel two-machine flow-shop scheduling problem with additional constraints. Given a query, we establish an upper bound on the number of storage devices for an optimal I/O schedule and provide experimental proof for it. For queries that access small amounts of data from multiple media, we derive an optimal schedule analytically. For queries that access large amount of data we derive a heuristics based scheduling algorithm using analytically proved results. We also discuss practical aspects of about using the theoretical results in a real environment and demonstrate our applicability of our work using an accurate robotic tape library simulator.

The rest of this paper is organized as follows. Section 2 introduces the problem. Sections 3, 4 and 5 analyze the problem and provides theoretical results for the problem. Section 6 discusses important practical considerations for the problem. Section 7 presents performance evaluation of our approach. Conclusions are presented in section 8.

2 Background

2.1 System model

The system consists of $A \geq 1$ robotic arms and $T > 1$ tape drives. A query needs data from n tapes. Reading data from a tape consists of following operation:

1. Rewinding previously loaded tape
2. Ejecting the previously loaded tape
3. Putting back previously loaded tape
4. Fetching new tape
5. Loading new tape
6. Searching and reading data inside the tape

Operations 3 and 4 are done by the robotic arm and the rest are carried out by the tape drive. For tape i, the time to do the arm operations is denoted by t_{A_i} and time to do the drive operations is denoted by t_{D_i}. t_{A_i} consists of time to put away tape $i-1$ and fetching tape i. Since t_{A_i} is same for all the tapes we denote it by t_A. t_{D_i} consists of searching and reading data from tape i, rewinding tape i and ejecting it from the drive. We assume that fairly accurate estimates of t_A and t_{D_i} are available. Section 6 discusses practical aspects of estimating these values.

2.2 Problem Statement

> Given a set of tapes and blocks from each tape that need to be accessed, find the order in which the tapes should be read to minimize the total I/O time is minimized.

The problem can be cast as a special case of two-machine flow-shop scheduling problem. The arm operations denote the first machine and the drive operations denote the second machine. There are n jobs to be scheduled one for each tape that contains the data needed to answer the query. The distinctive features of our problem (in contrast to the traditional two-machine flow-shop scheduling problem) are:

1. Multiple instances of machines. More common system configurations have a single (or maybe two) robotic arms servicing a number of tape drives. In this paper, we consider the case where there is one instance of the first machine and multiple instances of the second machine.

2. Buffer bound $= T$. At the most T jobs can be in the shop simultaneously. Since the number of tape drives in the system is T, not more than T tapes can be read simultaneously.

3. If job i starts at time s_i then it must be scheduled so that it finishes by $s_i + t_A + t_{D_i}$. This implies that the second machine is idle for time t_A before a job can be scheduled on it. The first machine is processing the job during this mandatory idle time. This accommodates for the behavior of a tertiary storage system where the a drive is empty after the tape is unloaded and remains empty while the robotic arm is loading the next tape.

4. Practical considerations (section 6) prevent use of scheduling algorithms that compare t_A and t_{D_i} values. For example, Johnson's algorithm ([Joh54]), which is optimal for traditional two-machine flow-shop scheduling, performs such a comparison to split the workload.

3 Workload characterization

In this section we study the behavior of the system and relate workload characteristics with the number of machines in each stage. The workload characterization is based on comparative values of time taken by the jobs on the first machine and the second machine. More formally, a job is characterized by a k *value of the job.* For any job i, its k value is governed by the inequality $(k-1) \times t_A < t_{D_i} \leq k \times t_A$. The k value of a workload is defined as $(k-1) \times t_A < max_i(t_{D_i}) \leq k \times t_A$. Given a workload, we derive the upper bound on the number of instances of the second machine based on its k value. We back the proof with experimental results. The experimental results suggest that a more tighter bound on number of instances of the second machine exists. Finally we use the result to characterize jobs (and workloads) given the k value of the job (workload) and number of instances of the second machine in the system.

Theorem 1 *For $A = 1$, if $(k-1) \times t_A < max_i(t_{D_i}) \leq k \times t_A$, then increasing number of instances of the second machine beyond $min(n, k+1)$ does not improve the makespan of any schedule.*

Proof. For $n \leq T$: Trivial. For $n > T$: The first T jobs are started on the first machine at time $s_i = i \times t_A$ where $0 \leq i \leq (T-1)$. Job T starts on machine one no earlier than $max(T \times t_A, min_{j=0}^{T-1}(j \times t_A + t_A + t_{D_j}))$ which is equal to $max(T \times t_A, (k+1) \times t_A)$. In general, job i starts on the first machine no earlier than $max(i \times t_A, (k+1+i-T) \times t_A)$ where $i \geq T$. For $T \geq (k+1)$, all jobs start at the same time. So we conclude that increasing number of instances of second machine beyond $k+1$ does not improve makespan for any schedule. ■

3.1 Experimental validation

We verified the above theorem using simulation. The value of t_A was set to 1. The value of k was varied between 1 and 16. The value of T_{D_i} were randomly generated between 0 and $k \times t_A$. For example, the workload with $k = 4$ contained jobs whose T_{D_i} values varied between 0 and 4. Each workload consisted of 100 jobs. Since the workloads were randomly generated, we created 1000 distinct instances of each type of workload and averaged the results over the instances.[1] For the rest of this paper, *a workload* refers to a set of random workloads of the same type and *makespan of a workload* means averaged makespan of multiple random instances of the workload. Number of instances of the second machine were varied between 1 to 32 and number of instances of the first machine is 1.

Fig. 1 plots the results. Each curve corresponds to a workload for a distinct value of k. Note that increasing the value of k results in jobs with bigger values of T_{D_i}. Hence, a workload with a higher k value has bigger jobs in it. For each workload, the curve remains flat after the number of instances of the second machine exceed $k + 1$.

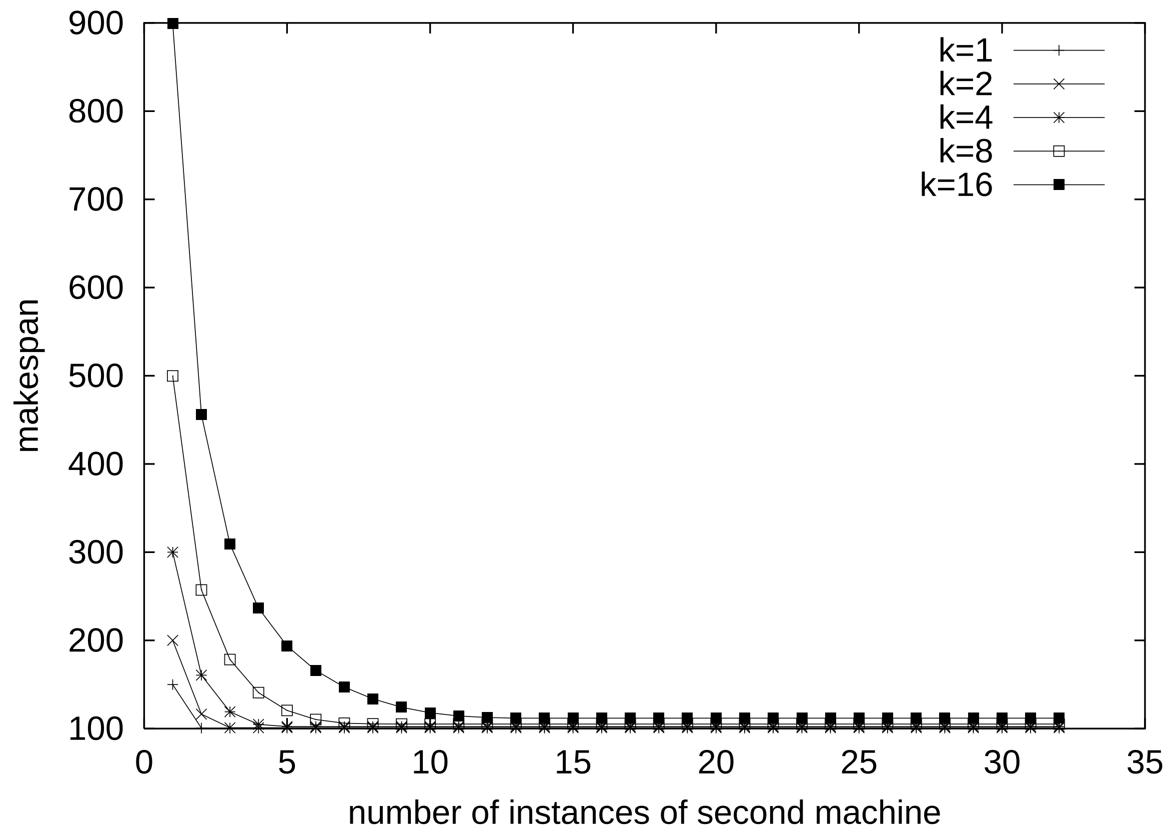


Figure 1:

Our results also reveal that the makespan for a workload does not show significant improvement even before the number of instances of the second machine exceed $k + 1$. This suggests that the scalability (in terms of number of instances of the second machine) is below $k + 1$. Based on our results we hypothesize that:

[1]Simulations with other values of number of jobs in the workload and number of random instances of a type of workload did not did not differ qualitatively from the results presented here. This observation holds for results for random workloads presented in this paper.

Hypothesis 1 *For $A = 1$, if $(k-1) \times t_A < \frac{\sum_i t_{D_i}}{n} \leq k \times t_A$, then increasing number of instances of the second machine beyond $min(n, k+1)$ does not improve the makespan of any schedule.*

3.2 Categorization of jobs

The above result provides an interesting insight to the problem. Number of robotic arms in a tertiary storage system is far less than number of tape drives. For example, its not uncommon to find a single robotic arms servicing four (or even more) tape drives. Since the number of instances of the first machine is equal to the number of robotic arms, the result becomes significant. Given a workload, it tells us when the first machine is the bottleneck and when its not. Given the system configuration, makespans of workloads with k values less than or equal to T will be constrained by the first machine, that is idle times can be introduced on the second machine because the first machine is always busy. The jobs in these kind of workloads have their execution time on the second machine bounded above by $T \times t_A$, where T is the number of instances of the second machine. Jobs in these workloads are *small jobs.* Execution time on the second machine for a *large job* is more than $T \times t_A$.

4 Workloads consisting of small jobs

Small jobs have their execution time on the second machine bounded above by $T \times t_A$. They represent a situation where after loading a media in a drive, the drive finishes reading data off that media before the robotic arm can finish loading media in other drives. In such situation, we find that the robotic arm is busy all the time (except at the end when there are no more media to access) irrespective of the order in which the media are loaded. This case occurs when the data layout does not conform to the access pattern exhibited by the query and hence one needs to retrieve smaller pieces of data from a large number of media.

We first prove optimality of the schedule that schedules the jobs in the decreasing order of their execution time on the second machine. We then validate the result experimentally.

Theorem 2 *$\forall i$, if $t_{D_i} \leq (T-1) \times t_A$ and $A = 1$, then longest-t_D-first (LtF) schedule is optimal.*

Proof. The LtF schedule sorts the jobs in the decreasing order of their time on the second machine. Let S be the optimal schedule. If S differs from LtF, then there exist two jobs i and j such that $i < j$ and $T_{D_i} < T_{D_j}$. Since $t_{D_i} \leq (T-1) \times t_A$, jobs i and j can start at times $i \times t_A$ and $j \times t_A$ respectively, irrespective of the values of t_{D_i} and t_{D_j}. Hence jobs i and j can be swapped without affecting legality of the schedule. Further, if j is the last job to finish ($s_j + t_{D_j} = makespan(S)$), the swap will reduce the makespan by $T_{D_j} - T_{D_i} > 0$. This process can be continued to arrive at a schedule identical to LtF. ■

Experimental Validation The experiments test effectiveness of LtF scheduling against naive scheduling strategy which does not change the order of the jobs. T, number of instances of the second machine, was varied between 1 and 32. For a given value of T, we set $k = T - 1$ when generating random workload. The value of t_A was set to 1. A, number of arms, is set to one. Hence a workload with $T = 8$ will contain jobs with t_{D_i} between 0 and 7. A workload for a higher value of T, has bigger jobs in it.

The results are shown in figure 2. The graph consists of two curves, one for the naive scheduling algorithm and one for the LtF scheduling algorithm. Each point on the curve plots the makespan of the schedule generated by the algorithm for the workload of the type $T = k + 1$. For the naive scheduling algorithm, the makespan increases as the $T = k + 1$ value of the workload increases. Naive scheduling algorithm is not able to compensate the increase in the size of the jobs using additional instances of the second machine that are available. Schedules generated using LtF algorithm on the other hand, are able to maintain scalability by making efficient use of additional instance of the second machine that are available and hence makespans of schedules generated using LtF algorithm are hardly affected when number of instances of the second machine increases. In fact, LtF tends to perform better with higher value of T. Hence we conclude that LtF is an optimal and scalable algorithm for workloads where $(k-1) \times t_A < \frac{\sum_i t_{D_i}}{n} \leq k \times t_A$, i.e., queries that access small amount of data from each tape.

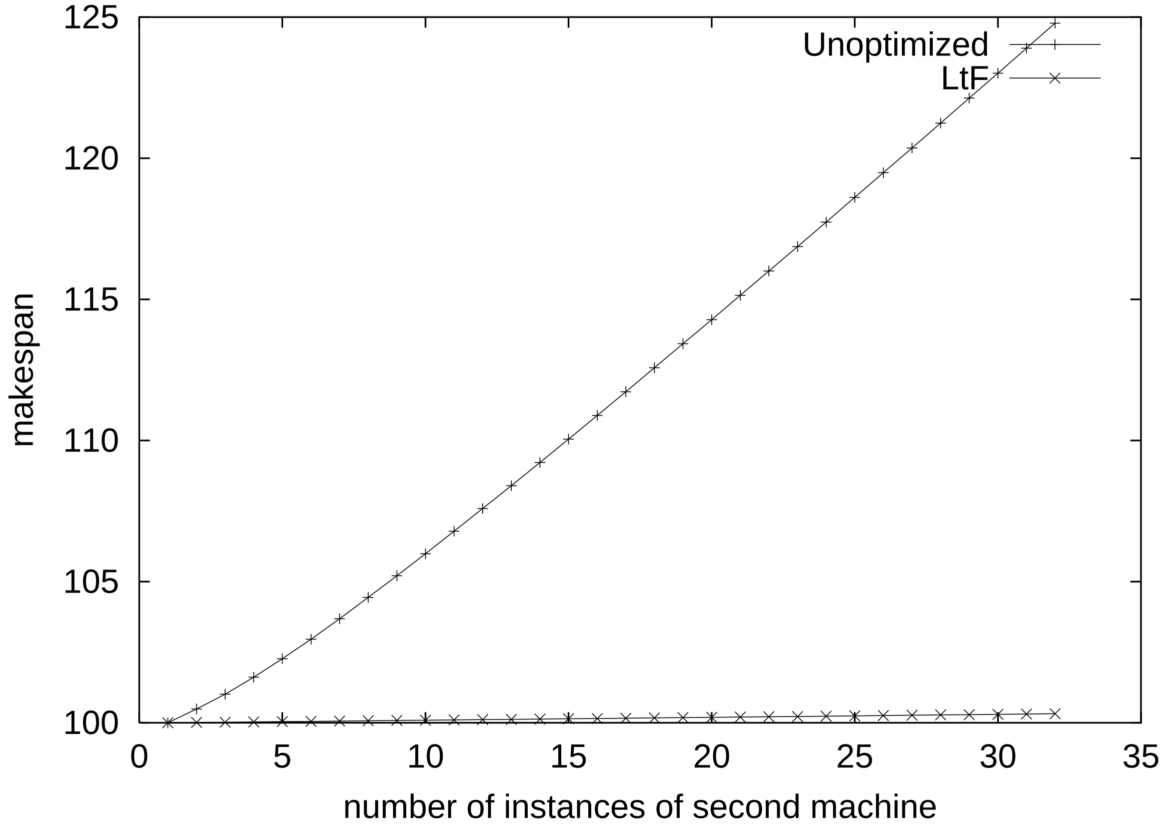


Figure 2:

5 Workloads consisting of big jobs

For that access large amount of data from each tape (where $t_{D_i} > (T-1) \times t_A$) the tape drives, the first machine is not a bottleneck. This leads one to believe that eliminating idle times on the second machine will lead to an optimal schedule.

Lemma 1 *$\forall i$, if $t_{D_i} > (T-1) \times t_A$ and $A = 1$, then shortest-t_D-first (StF) schedule is optimal in terms of idle time for the second machine.*

Proof. The first T jobs are started on the first machine at time $s_i = i \times t_A$ where $0 \leq i \leq (T-1)$, irrespective of the scheduling algorithm used. Job T is scheduled at time $s_T = max(T \times t_A, min_{j=0}^{T-1}(s_j + t_{D_j})) = max(T \times t_A, min_{j=0}^{T-1}(j \times t_A + t_A + t_{D_j}))$. When using StF, $t_{D_i} \leq t_{D_j}$ if $i \leq j$. Hence, $s_T = max(T \times t_A, t_A + t_{D_0}) = t_A + t_{D_0}$, since $\forall i$, if $t_{D_i} > (T-1) \times t_A$. Note that, job T is scheduled on the same instance of the second machine where job 0 was scheduled. Also, job T is scheduled immediately after job 0. In general, job i is scheduled immediately after job $i - T$ on the same instance of the second machine. Hence no instance of the second machine is idle as long as there are more jobs to process. ■

However, StF schedule does not necessarily produce the optimal schedule. Apart from idle times of machines, the length of the *head* and *tail* of the schedule determine the optimality of a schedule. When a schedule is executed in a two-machine flow-shop environment, the second machine starts later than the first machine, producing idle time for the second machine. The length of this idle time depends on the execution time of first T jobs on the first machine, where T is the number of instances of the second machine. For the problem under consideration, the length of the head is independent of the scheduling algorithm. After the first machine is finished processing all the jobs, the second machine is still busy and requires some time before it finishes processing all the jobs. The length of the tail is at least as big as the execution time on the second machine of the last job scheduled. The StF schedule puts the job with largest second machine time last. This results in higher value for the length of the tail, producing a suboptimal makespan.

Schedules generated by longest-t_D-first (LtF) algorithm on the other hand can produce idle time on the second machine but are successful in reducing the length of the tail. This is because the LtF algorithm puts smallest jobs at the end of the schedule. But as we will see later, reducing the tail also does not guarantee optimal schedule. We start by showing that scheduling one job per instance of the second machine can lead to optimal schedule when number of jobs equals number of instances of the second machine.

Lemma 2 *$\forall i$, if $t_{D_i} > (T-1) \times t_A$ and $A = 1$ and $n = T$, then there exists an optimal schedule which scheduled exactly one job on each instance of the second machine.*

Proof. Consider an optimal schedule that has $m > 1$ jobs scheduled on an instance of the second machine. Since $n = T$, this implies there are at least $m - 1$ instances of the second machine that do not have any jobs scheduled on them during the entire makespan. This job can be scheduled on any of the idle machines while maintaining a legal schedule. If this was the last job to finish according to the original schedule, then we get a schedule with a shorter makespan, contradicting our assumption that the original schedule was optimal. If it was not the last job to finish in the original schedule then the new schedule has equal makespan. In both cases, the instance of the second machine that had m jobs scheduled on it, now has $m - 1$ jobs scheduled on it according to the new schedule with equal or less makespan than the original schedule. This process can be repeated until there is exactly one job scheduled on each instance of the second machine. ∎

Lemma 3 *$\forall i$, if $t_{D_i} > (T-1) \times t_A$ and $A = 1$ and $n = T$, then longest-t_D-first (LtF) algorithm schedules exactly one job on each instance of the second machine.*

Proof. Trivial. ∎

Next we show that LtF produces optimal schedule when number of jobs equals number of instances of the second machine.

Lemma 4 *$\forall i$, if $t_{D_i} > (T-1) \times t_A$ and $A = 1$ and $n = T$, then longest-t_D-first (LtF) schedule is optimal.*

Proof. There exists an optimal schedule (OPT) that will schedule exactly one job on each instance of the second machine. Since LtF also schedules exactly one job on each instance of the second machine, if OPT schedules jobs in the same order as LtF, then LtF is optimal. If not, in OPT, there exist two jobs scheduled consecutively, jobs i and $i + 1$, such that $t_{D_i} < t_{D_{i+1}}$. Let s_i and s_{i+1} be the time jobs i and $i + 1$ start on the first machine, respectively. Then $s_i + t_A \leq s_{i+1}$ holds. Note that job i will never finish after job $i + 1$. Since all the jobs are scheduled on distinct instances of the second machine, we can swap jobs i and $i + 1$ while maintaining legality of the schedule. The new schedule will have a makespan no more than OPT. Repeated application of this process will transform the OPT schedule into LtF without any increase in the makespan. ∎

But LtF does not necessarily produce the best makespan when number of job exceeds number of instances of the second machine.

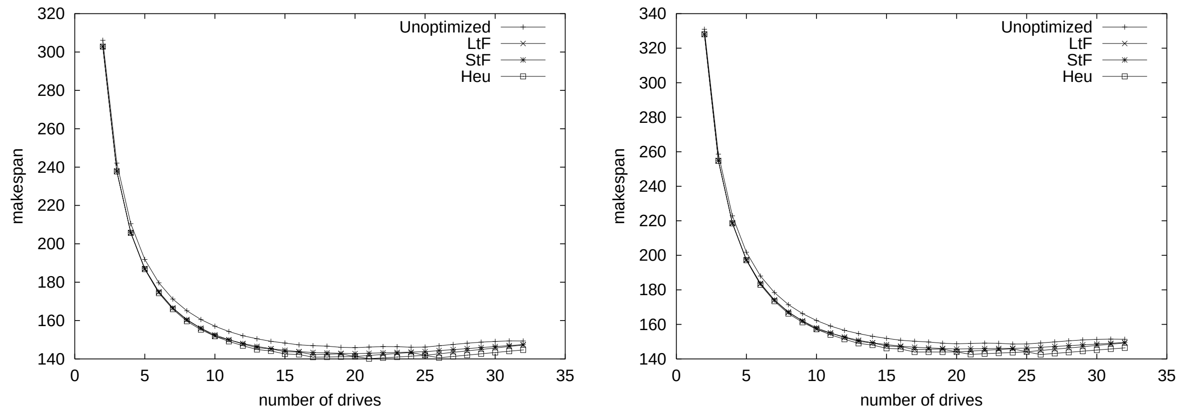


Figure 3:

Lemma 5 *$\forall i$, if $t_{D_i} > (T-1) \times t_A$ and $A = 1$ and $n > T$, then longest-t_D-first (LtF) schedule can be suboptimal.*

Proof. We prove this by giving a counter example. Let $n = 2T$. The jobs are numbered 1 to $2T$ such that $t_{D_i} \geq t_{D_{i+1}}$. Let $t_{D_i} = t_{D_1} - (t_A \times (i-1)), 1 < i \leq T$ and $t_{D_i} = t_{D_{T+1}}, T < i \leq 2 \times T$. Scheduling the jobs in LtF order (jobs are scheduled in the order $1, 2, \ldots, 2T$) will result in a makespan of $(T+1) \times t_A + t_{D_1} + t_{D_{T+1}}$. If the order in which the jobs are scheduled is changed to $T+1, T+2, \ldots, 2T, 1, 2, \ldots, T$, the makespan turns out to be $2 \times t_A + t_{D_1} + t_{D_{T+1}}$. ∎

Heuristic approach We saw earlier that StF produces schedules optimal in terms of idle times of second machine. Also LtF is optimal when number of jobs is equal to number of instances of the second machine. This suggests that a schedule which simultaneously exhibits properties of LtF and StF schedules should perform better. Based on this observation, we propose a new heuristic:

1. Sort the jobs using StF strategy.
2. Starting at the end of the StF schedule, pick the last T (number of instances of the second machine) jobs and sort them using LtF in place.
3. Repeat the above step moving towards the start of the schedule.

We evaluated the heuristic experimentally. The results are shown in figures 3 to 7.

6 Estimating drive operation times

So far in this paper we assumed that the values of t_A and t_{D_i} for each job (tape to be read) are known. It turns out that the value of t_A is not needed since all the scheduling algorithms we have seen so far use t_{D_i} values to generate the schedule. In general, its hard if not impossible to calculate T_{D_i} accurately given the set of blocks on the tape that are to be read. This is because it requires accurate modeling of the tape drive(s). On the other hand, tape drive manufacturers do provide performance numbers. These numbers include peak/average search (seek) rate and peak/average read rate. These values can be used to estimate t_{D_i}. Furthermore, the scheduling algorithms need to only compare values of t_{D_i} and never make use of the absolute values of t_{D_i}. We denote the estimated value of t_{D_i} by $t_{D_i}^{estimated}$. Ideally, if

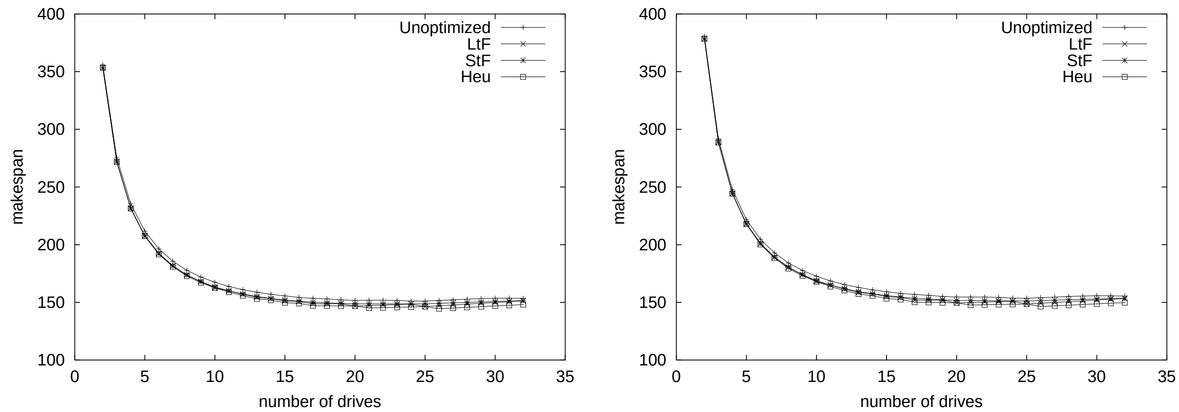


Figure 4:

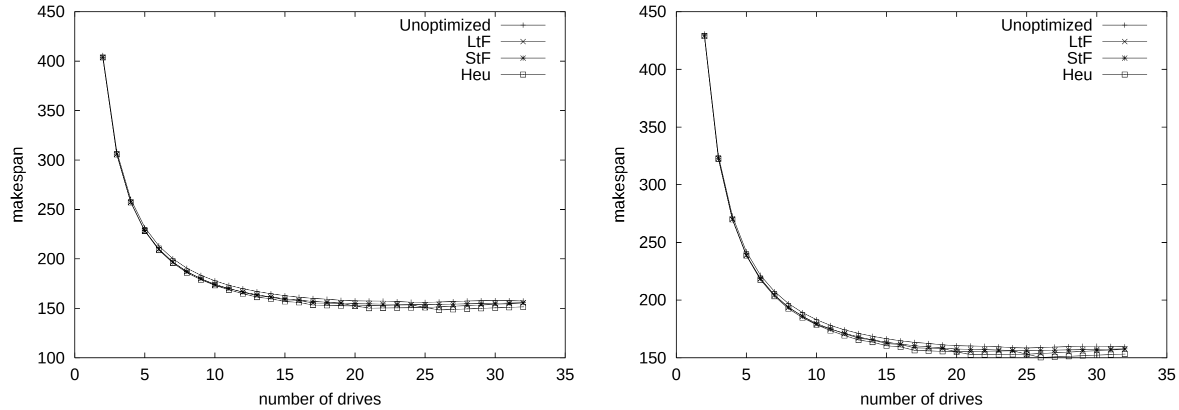
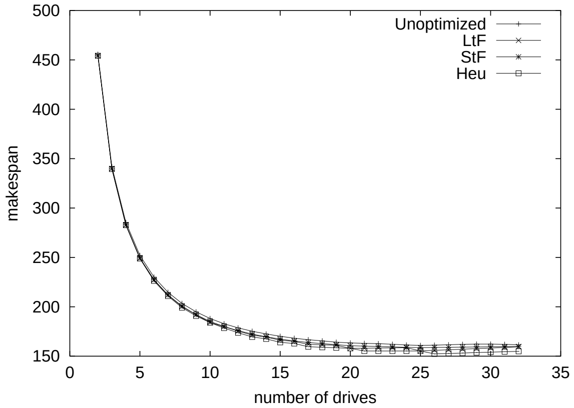


Figure 5:



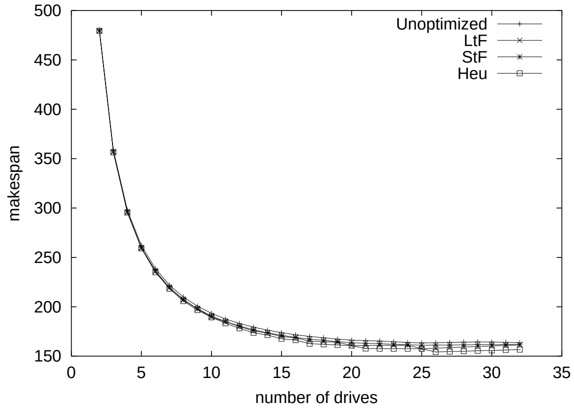


Figure 6:

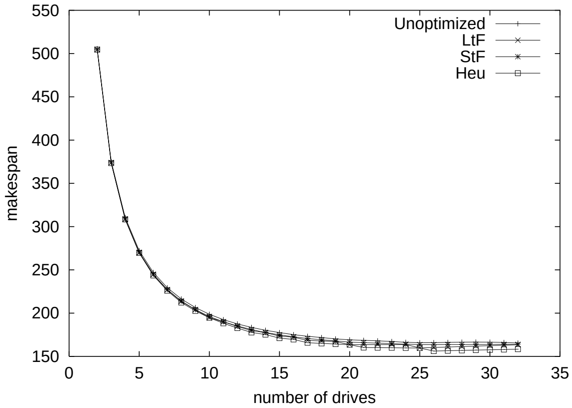


Figure 7:

Forward seek past k blocks	$4.834 + 0.378k$ seconds	$k \leq 28$
	$14.342 + 0.028k$ seconds	$k > 28$
Reverse seek past k blocks	$4.99 + 0.328k$ seconds	$k \leq 28$
	$13.74 + 0.0286k$ seconds	$k > 28$
Reverse seek to the start of the tape	21 seconds	extra
Reading k blocks after a forward seek	$0.38 + 1.77k$ seconds	
Reading k blocks after a reverse seek	$1.77k$ seconds	
Ejecting a tape	19 seconds	
Fetching a new tape from library	20 seconds	
Loading a tape	42 seconds	

Table 1: Analytical model for EXB-8505XL tape drive and EXB-210 tape library from [HRS99]

$t_{D_j} < t_{D_k}$ then $t_{D_j}^{estimated} < t_{D_k}^{estimated}$ should hold. We evaluated three different schemes to compute $t_{D_i}^{estimated}$:

1. *Maximum Offset Estimate* For each tape we find the offset of the farthest block to be read inside that tape. For each tape i, $T_{D_i}^{estimated}$ = maximum offset. This value is approximately proportional to the time it will take to rewind this tape under the tape drive model (see section 7.1) we use.

2. *Data Volume Estimate* For each tape i, $T_{D_i}^{estimated}$ = number of blocks read from the tape. This value is approximately proportional to the time it will take to read the blocks from this tape.

3. *Full Estimate* This estimation method combine the above two estimation methods. $T_{D_i}^{estimated} = seek\ rate \times maximum\ offset + read\ rate \times blocks\ read + seek\ rate \times (maximum\ offset - blocks\ read)$. The first term estimates the rewind time, the second term estimates the time to read the tape and the third term estimates the time to forward seek inside the tape while searching for the next block to be read.

Our experiments revealed that *data volume estimates* and *full estimates* help scheduling algorithms perform better than using *maximum offset estimates*. We found that scheduling algorithms perform equally well whether *data volume estimates* are used or *full estimates* are used. We use *data volume estimates* for all scheduling algorithms since it has lower computing requirements.

7 Performance evaluation

We first describe the analytical tape model that we use for presenting experimental results later in this section. Next we evaluate various scheduling algorithms using synthetic workloads. We then describe the Sequoia 2000 benchmark and evaluate the scheduling algorithms using the benchmark.

7.1 Analytical model for the tape library

Most of the literature ([Sar95b, SS96, ML95, DK93]) uses a linear approximation of the locate time for tape drives. [HS96a] found that such linear approximation is inaccurate. We use the analytical models of Exabyte's EXB-8505XL tape drive and EXB-210 tape library described in [HRS99]. The model uses a logical blocks size of 1MB and is described in table 1. We use the *SORT* algorithm described in [HS98] for I/O scheduling when fetching data from the same tape.

7.2 Random Workload

Figure 8 shows average performance of various scheduling algorithms over a set of randomly generated workloads. We use analytical model of Exabyte's EXB-8505XL tape drive and EXB-210 tape library (see section 7.1) containing 32 tapes for these experiments. The FoldLtF algorithm is a heuristic proposed in [MMS99] for job scheduling in a limited floor-space flows-hop environment. The algorithm first generates a list using LtF algorithm and then schedules jobs from both the ends of the list. The figure plots the makespan of each scheduling algorithm as percent of an OPT value. The OPT value is a lower bound on the makespan of the optimal schedule. We calculate OPT as follows: For each tape from which data is to be read, we compute the time taken to load that tape, read the necessary data, rewind the tape and place it back in the library. The OPT value is then sum of the times to access each tape divided by the total number of drives in the system. Note that we use the same set of workloads for all the data points in the figure. Hence the OPT value is inversely proportional to the number of drives in the system. We find that performance of the scheduling algorithms is not within a constant factor of OPT (for the expected range of value of number of drives in the system), its a function of the number of drives in the system. When comparing performance of LtF and StF, we find that the length of the tail of the schedule is more important than amount of idle time in the schedule for reducing makespan, since LtF always outperforms StF. The FoldLtF algorithm performs only slightly better than the UNOPTIMIZED case that too when number of drives in the system is comparatively higher. Our heuristic based algorithm is always performs well. This is due to a careful balance between idle times and length of tail achieved by our algorithm. Performance of LtF algorithm approaches performance of our algorithm when number of drives in the system is very low or very high. The reason LtF (and the UNOPTIMIZED case too) perform on par with our algorithm when number of drives is small is because there is very little scope for optimization in that case due to limited choice available to scheduling algorithms due to limited number of tape drives available. When the number of drives is high, number of tapes to be loaded is close to/equal to number of drives available. As we saw earlier, LtF is optimal for small jobs and it is optimal for large jobs when number of instances of the second machine are equal to or greater than number of jobs. But our algorithm clearly outperforms LtF when number of drives in the system is moderate (between 4 and 16). Note that this range of values of number of drives, is commonly found in a typical data management system handling large amounts of data.

7.3 Experimental verification using Sequoia 2000 benchmark

We evaluated our technique using Sequoia 2000 Storage Benchmark ([SD92]. We first provide description of the relevant portions of the benchmark and queries. We then describe the workload used for the experiments and give experimental results.

7.3.1 Description of the Sequoia 2000 benchmark

We base our experiments on the Sequoia 2000 Storage Benchmark ([SD92]) for the results presented in this section. The benchmark provides three separate datasets, *regional*, *national*, and *world*. We use the national dataset over a period slightly less than four years (200 weeks) which is about 64GB in size. The spatial resolution of the dataset is $0.5km X 0.5km$.

The schema for the tables used in these queries is:

```
create RASTER( location = box,
time   = int4, band     = int4,
data   = int2[][])
```

time is a four byte integer and denotes the half-month over which the raster data was accumulated. The data set contains data for 200 weeks, with the values $1, 2, \ldots, 100$. The *location* attribute specifies the bounding box for the raster data and is represented by the coordinates of its top-left and right-bottom corners represented by four 4-byte integers. *band* specifies the wavelength band at which the data was captured; the values range over $1, 2, 3, 4, 5$. *data* is a two dimensional array of size $10240 X 6400$. Each point is a two byte integer.

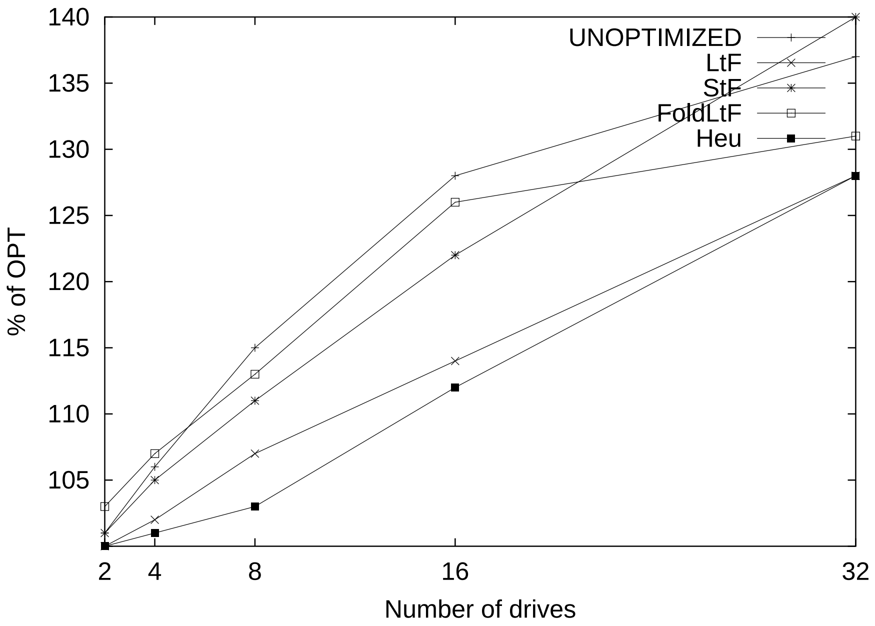


Figure 8: The figure shows performance of various scheduling algorithms for random workloads using Exabyte's EXB-8505XL tape drive and EXB-210 tape library (see section 7.1). Number of robotic arms used is 1.

This is a four dimensional dataset. The dimensions are *time*, *band*, X and Y, where X and Y are the two dimensions from the raster image. All the raster images are stored chronologically sorted, since they were captured such. Raster images for a half-month are not sorted in any particular order.

7.3.2 Workload description

A query represents an access pattern on the dataset. In general multiple access patterns are observed on a typical dataset. Access patterns are executed with different frequencies. In order to capture this phenomenon, we first define variety of access patterns (queries) on the dataset. Then we create different workloads using these queries by manipulating number of different queries in the workload.

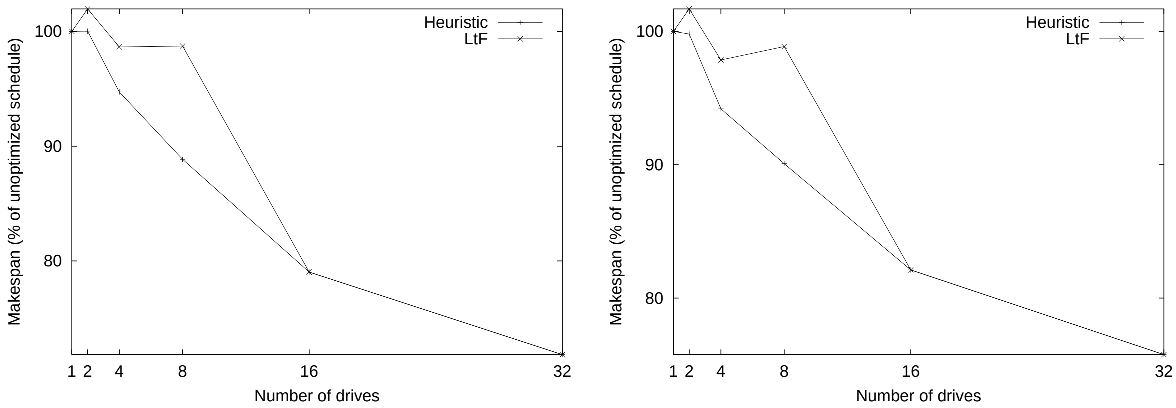
Queries We use two types of queries.

- **Query Type 1** *Select all images belonging to a band.* The data of interest is spread over the entire set of tapes that store the dataset.
- **Query Type 2** *Select all images belonging to a half-month.* The data of interest is localized in a few tapes of the set of tapes that store the dataset.

A query type is not an actual query, but rather a set of queries that define an access pattern. Each query in the set is an instance of the query type. For example, a query that retrieves all the images belonging to band 3 is an instance of the query type 1.

Workload generation A workload is generated using the following parameters:

- *Number of Queries* This denotes the total number of queries that this workload will consist of.



(a) Workload 1

(b) Workload 2

Figure 9: Experimental results for Sequoia 2000 storage benchmark

- *Query Type Percentages* These represent the percentage of query instances that belong to each query type.

The number of queries determines the accuracy of the workload generation process. If too few query instances are generated, it can result in mis-representation of query types in the workload. We use the rule thumb: For all query types, if the number of distinct query instances that belong to a query type is n and the query type percentage is p, the workload should contain at least $\frac{n}{p}$ queries. This assures that the expected number of occurrences for any query instance for a query type is at least 1. We evaluate two different workloads:

- **Workload 1** consists of majority (90%) of queries from query type 1.

- **Workload 2** has equal mix of queries from query type 1 and query type 2.

7.3.3 Performance evaluation

Figure 9 shows the performance results of our heuristic algorithm against LtF algorithm. The time taken to execute a workload by an algorithm is plotted as a percent of the time taken by a naive scheme that does not do any scheduling. The results show that our algorithm performs consistently well. Note that for 16 and 32 drives case, the number of tapes from which data is read for a query is less than the number of drives in the system. Since LtF has already been proved to be optimal in that case, LtF performs equally well when compared to our algorithm. When number of drives in the system is moderate, our algorithm clearly outperforms LtF. The gains in performance are due to a balanced optimization of both drive idle times and the size of the tail of the schedule.

8 Conclusions

This paper investigated issues in optimizing I/O time for a query whose data resides on automated tertiary storage containing multiple storage devices. We modeled the problem as a limited storage parallel two-machine flow-shop scheduling problem with additional constraints. The paper presented analytical results that provide insight to the problem. We presented a heuristic algorithm for scheduling data from a tape library. Our performance results show impressive gains for synthetic as well as real workloads.

References

[CDK+95] Ling Tony Chen, R. Drach, M. Keating, S. Louis, Doron Rotem, and Arie Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. *Information Systems*, 20(2):155–183, 1995.

[CHL93] Michael J. Carey, Laura M. Haas, and Miron Livny. Tapes hold data, too: Challenges of tuples on tertiary storage. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C.*, pages 413–417. ACM Press, 1993.

[DK93] Ann L. Drapeau and Randy H. Katz. Striping in large tape libraries. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference, San Diego, California, USA*. IEEE Computer Society Press, 1993.

[FPS96] S. Fox, N. Prasad, and M. Szezur. NASA's EOSDIS: an integrated system for processing, archiving, and disseminating high-volume earth science imagery and associated products. Technical Report 215-TP-001-001, NASA/Goddard Space Flight Center, July 1996.

[GMW95] Leana Golubchik, Richard R. Muntz, and Richard W. Watson. Analysis of striping techniques in robotic storage libraries. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems, Monterey, California, USA*, pages 225–238. IEEE Computer Society Press, 1995.

[HRS99] Bruce K. Hillyer, Rajeev Rastogi, and Avi Silberschatz. Scheduling and data replication to improve tape jukebox performance. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia*, pages 532–541, 1999.

[HS96a] Bruce K. Hillyer and Avi Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Philadelphia, Pennsylvania, U.S.A.*, pages 170–179, 1996.

[HS96b] Bruce K. Hillyer and Avi Silberschatz. Random i/o scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada*, pages 195–204. ACM Press, 1996.

[HS98] Bruce K. Hillyer and Avi Silberschatz. Scheduling non-contiguous tape retrievals. In *Proceedings of Sixth NASA Goddard Conference on Mass Storage Systems and Technologies and Fifteenth IEEE Mass Storage Systems Symposium, University of Maryland, College Park, Maryland*, March, 1998.

[Inm] Bill Inmon. The Role of Nearline Storage in the Data Warehouse: Extending your Growing Warehouse to Infinity. Technical white paper. StorageTek.

[Joh54] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, March 1954.

[KBCH95] Ben Kobler, John Berbert, Parris Caulk, and P. C. Hariharan. Architecture and design of storage and data management for the nasa earth observing system data and information system (eosdis). In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems, Storage, Monterey, California, USA*, pages 65–76. IEEE Computer Society Press, 1995.

[MC99] S. More and A. Choudhary. Tertiary Storage Organization for Large Multidimensional Datasets. Technical Report CPDC-TR-9911-018, Center for Parallel and Distributed Computing, Northwestern University, November 1999.

[ML95] Jussi Myllymaki and Miron Livny. Disk-tape joins: synchronizing disk and tape accesses. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Ottawa, Canada.*, pages 279–290, 1995.

[MMS99] Sachin More, S. Muthukrishnan, and Elizabeth Shriver. Efficiently sequencing tape-resident jobs. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, pages 33–43. ACM Press, 1999.

[OLA98] APB-1 OLAP Benchmark, Release II, November 1998. OLAP Council.

[Sar95a] Sunita Sarawagi. Database systems for efficient access to tertiary memory. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems, Monterey, California, USA.*, pages 120–126. IEEE Computer Society Press, 1995.

[Sar95b] Sunita Sarawagi. Query processing in tertiary memory databases. In *Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland*, pages 585–596. Morgan Kaufmann, 1995.

[SD92] M. Stonebraker and J. Dozier. SEQUOIA 2000: Large Capacity Object Servers to Support Global Change Research. Technical Report SEQUOIA 2000 Technical Report No. 1, Electronics Research Lab, March 1992.

[SS96] Sunita Sarawagi and Michael Stonebraker. Reordering query execution in tertiary memory databases. In *Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India*, pages 156–167. Morgan Kaufmann, 1996.

[Sto91] Michael Stonebraker. Managing persistent objects in a multi-level storage. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado.*, pages 2–11. ACM Press, 1991.