Scheduling Queries for Tape-Resident Data[⋆]

Sachin More and Alok Choudhary

Department of Electrical and Computer Engineering
Northwestern University

Abstract. Tertiary storage systems are used when secondary storage can not satisfy the data storage requirements and/or it is a more cost effective option. 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.

1 Introduction

Tertiary storage systems are employed in cases where secondary storage can not satisfy the data storage requirements or tertiary storage is a more cost effective option [8]. NASA's Earth Observing System (EOS) Data and Information System (EOSDIS) is an example of the former [4, 13]. The latter case, can be found in data warehousing applications. Inmon [10] 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 without sacrificing performance.

Tertiary storage devices have traditionally been used as archival storage. The new application domains require on-demand retrieval of data from these devices [9]. While data archiving applications access large chunks of contiguous data, these new applications access data that is scattered on multiple media. Hence correct scheduling of data retrieval requests becomes important. For example, I/O time for a query that accesses data from two different media using a single robotic arm and two tape drives is minimized when the tape that needs more time to read is loaded first.

Many of the application domains that use tertiary storage access *multidimensional datasets* [2]. 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 [18], 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 [15]. 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

[⋆] This work is supported by DOE ASCI Alliance program under a contract from Lawrence Livermore National Labs B347875.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1292–1301, 2000.

automated tertiary storage, the total I/O time for the query is also influenced by the order in which media are accessed.

Various issues in tertiary storage management have been addressed by the database community. Carey et. al. [1] evaluate issues in extending database technology for storing/accessing data on tertiary storage. Stonebraker [23] proposes a database architecture that uses hierarchical storage. Livny et. al. [17] and Sarawagi [20, 21, 22] examine issues in query processing when data resides on tertiary storage. Data striping on tertiary storage has been evaluated in [3, 5]. Tertiary storage space organization issues are addressed in [2, 6]. 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. 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 proven results.

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

2 Background

The system model 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 the following set of operations: rewinding currently loaded tape; ejecting the tape; putting it back; fetching the tape to be read; loading the tape; searching and reading data inside the tape. Putting back a tape and fetching a new one are handled by the robotic arm and the rest of the actions are carried out by the tape drive. The time to do the arm operations is denoted by t_A (which we assume to be same for all tapes [6]) and time to do the drive operations is denoted by t_D.

Given a set of tapes and blocks from each tape that need to be accessed by a query, find the order in which the tapes should be read to minimize the total I/O time. The problem is cast as a special case of two-machine flow-shop scheduling problem [19]. The arm operations denote the first machine and the drive operations denote the second machine. There are n jobs to be scheduled. The optimality criteria is *makespan*, total execution time of the schedule. The distinctive features of our problem (in contrast to the traditional two-machine flow-shop scheduling problem) are:

- *Multiple instances of machines* More common system configurations have a single 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.

- *Buffer bound* $= T$ At the most T jobs can be in the shop simultaneously. The robotic arm can not load more tapes while all drives are busy accessing tapes loaded in them and must remain idle.
- If job i starts at time s_i on the first machine then it must be scheduled so that it finishes by $s_i + t_A + t_{D_i}$ on the second machine. The second machine is idle for time t_A before a job can be scheduled on it. This accommodates for the behavior of a tertiary storage system where a drive is empty while the robotic arm is loading the next tape. The case where $A = T = 1$ (a single tape drive serviced by a single robotic arm) is uninteresting under this condition. In the rest of this paper we assume $T > 1$.
- Practical considerations prevent use of scheduling algorithms that compare t_A and t_{D_i} values. The value of t_{D_i} can not be predicted correctly unless a very accurate analytical model of the tape drive is available. For example, Johnson's algorithm [12], which is optimal for traditional two-machine flow-shop scheduling, performs such a comparison.

3 Workload Characterization

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 $((k-1) \times t_A) < max_i(t_{D_i}) \leq (k \times t_A)$ where $0 \leq i < n$.

Theorem 1. [14] *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.*

The above result provides an interesting insight to the problem. Given a workload, it tells us when the first machine is the bottleneck and when it is 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 $T \times t_A$. 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

A workload containing small jobs represents 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 a 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.

Theorem 2. [14] *$\forall i$, if $t_{D_i} \leq (T-1) \times t_A$ and $A = 1$, then longest-t_D-first (LtF) schedule is optimal.*

5 Workloads Consisting of Big Jobs

For queries that access large amount of data from each tape (where $t_{D_i} > (T - 1) \times t_A$), 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.

Proposition 3. [14] *$\forall i$, 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.*

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. For the problem under consideration, the length of the head is independent of the scheduling algorithm. The StF schedule puts the job with largest second machine time last. This results in bigger 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. [14] shows that when number of jobs is less than or equal to number of instances of the second machine LtF produces optimal makespan. But LtF is not necessarily optimal when number of job exceeds number of instances of the second machine.

Proposition 4. [14] *$\forall i$, 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.*

We propose a new heuristic that combines properties of StF and LtF:

1. Sort the jobs using StF strategy.
2. Pick the last T (number of instances of the second machine) jobs and reverse their order. If there are n jobs, the last T jobs are numbered $n-T, n-T+1, \ldots, n-2, n-1$ at the end of previous step. And $t_{D_{n-T}} \leq t_{D_{n-T+1}} \leq \cdots \leq t_{D_{n-2}} \leq t_{D_{n-1}}$. We reverse their order so that $t_{D_{n-T}} \geq t_{D_{n-T+1}} \geq \cdots \geq t_{D_{n-2}} \geq t_{D_{n-1}}$
3. Repeat the above step for jobs $n-2T, n-2T+1, \ldots, n-T-2, n-T-1$. Keep repeating step 3 moving towards the start of the schedule.

Below is the illustration explaining working of our heuristic algorithm:

Jobs	a	b	c	d	e	f	g	h	i	j	k	l	m	n
t_D	6	10	7	13	8	5	4	1	9	2	0	12	11	3
Applying StF (Step 1)	k	h	j	n	g	f	a	c	e	i	b	m	l	d
Reversing order of the last 4 jobs (Step 2)	k	h	j	n	g	f	a	c	e	i	d	l	m	b
Repeated application of step 3	k	h	j	n	g	f	i	e	c	a	d	l	m	b
	k	h	f	g	j	n	i	e	c	a	d	l	m	b
	h	k	f	g	j	n	i	e	c	a	d	l	m	b

The workload consists of 14 jobs (a, b, . . . , n). The configuration of the flowshop is $A = 1, T = 4$. The jobs are first sorted using StF algorithm. Then the order of the last 4 jobs is reversed. The algorithm then works its way towards the start of the schedule, reversing orders of 4 consecutive jobs. In the final step, only two jobs remain, jobs h and k. Their order is reversed too.

6 Performance Evaluation

So far we assumed that the values of t_A and t_{D_i} for each job (tape to be read) are known. 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, since it requires accurate modeling of the tape drive(s). On the other hand, tape drive manufacturers do provide peak/average search (seek) rate and peak/average read rate. These values can be used to estimate t_{D_i}. The estimated value of t_{D_i} is denoted by $t_{D_i}^{estimated}$. Ideally, if $t_{D_j} < t_{D_k}$ 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 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 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 combines the above two estimation methods. $t_{D_i}^{estimated} = \text{seek rate} \times \text{maximum offset} + \text{read rate} \times \text{blocks read} + \text{seek rate} \times (\text{maximum offset} - \text{blocks read})$.

Our experiments revealed that *data volume estimates* and *full estimates* help scheduling algorithms perform better than using *maximum offset estimates.* We also found that scheduling algorithms perform equally well whether *data volume estimates* are used or *full estimates* are used. This is because read times dominate seek times for the workloads we considered. We use *data volume estimates* for all scheduling algorithms since it has lower computing requirements.

We use a tape library simulator to execute the schedules created by various scheduling algorithm. Most of the literature [3, 17, 21, 22] uses a linear approximation of the locate time for tape drives. [7] 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 [6] in our tape library simulator. We use the *SORT* algorithm described in [9] for I/O scheduling when fetching data from the same tape.

Random Workload Fig. 1 shows the performance of various scheduling algorithms over a set of randomly generated workloads. The set contains 1000

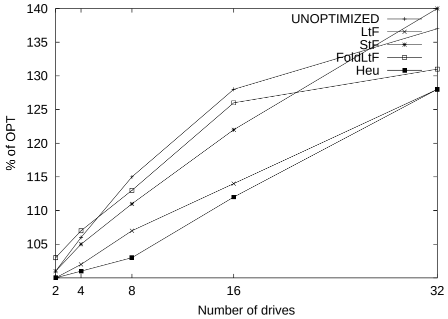


Fig. 1. Performance of various scheduling algorithms for random workloads.

workloads[1]. For each workload, we determine how many blocks to read from each tape by generating a random number[2] between 0 and the total number of blocks on the tape. Then for each tape, we generate N distinct block numbers randomly, where N is the number of blocks to be read from this tape.

The UNOPTIMIZED algorithm loads the tapes in random order. The FoldLtF algorithm is a heuristic proposed in [16] for job scheduling in a limited floor-space flow-shop 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. 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. Since LtF always outperforms StF, we conclude that the length of the tail of the schedule is more important than amount of idle time in the schedule for reducing makespan, 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 always performs well 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 when fewer tape

[1] We found that varying number of instances of the workload does not change the results qualitatively.

[2] We use an inversive congruential generator for generating random numbers.

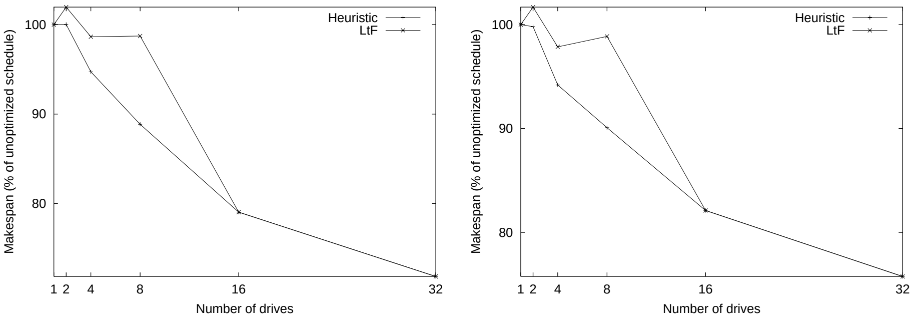
drives are used. As we saw earlier, LtF is optimal for large jobs when number of instances of the second machine are equal to or greater than number of jobs. When the number of drives is high, number of tapes to be loaded is close to/equal to number of drives available, making LtF optimal. But our algorithm 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. Since the performance of UNOPTIMIZED, StF and FoldLtF is considerably worse than LtF and our heuristic algorithm, we did not consider these algorithms for further performance studies.

Experimental Verification Using Sequoia 2000 Storage Benchmark We use the national dataset from the benchmark over a period of four years (200 weeks) and is about 64GB in size. 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 image was captured. The *location* attribute is the bounding box for the raster data. *band* is the wavelength band at which the data was captured. *data* is a two dimensional array of size $10240X6400$ of two byte integers at a spatial resolution of $0.5kmX0.5km$. 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.

A query type represents an access pattern on the dataset. A query is an instance of a query type. In general multiple access patterns are observed on a typical dataset. Access patterns are executed with different frequencies [2, 11]. In order to capture this phenomenon, we first define variety of access patterns (query types) on the dataset. Then we create different query mixes using these query types by manipulating number of different queries for each query type in the mix. We use two types of queries. *Query Type 1* selects 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* selects 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 mix is generated using two parameters: *Number of queries* denotes the total number of queries that this mix will consist of. *Query type percentages* represent the percentage of query instances that belong to each query type. The number of queries determines the accuracy of the query mix generation process. 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 mix 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 query mixes: *Query mix 1* consists of majority (90%) of queries from query type 1. *Query mix 2* has equal mix of queries from query type 1 and query type 2.



(a) Query mix 1

(b) Query mix 2

Fig. 2. Experimental results for Sequoia 2000 storage benchmark

Fig. 2 shows the performance results of our heuristic algorithm against LtF algorithm. The time taken to execute a query mix 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.

7 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

[1] Carey, M. J., Haas, L. M., and Livny, M. 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., 1993), ACM Press, pp. 413–417.

[2] Chen, L. T., Drach, R., Keating, M., Louis, S., Rotem, D., and Shoshani, A. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. *Information Systems 20*, 2 (April 1995), 155–183.

[3] Drapeau, A. L., and Katz, R. H. Striping in large tape libraries. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference* (San Diego, CA, 1993), IEEE Computer Society Press.

[4] Fox, S., Prasad, N., and Szezur, M. NASA's EOSDIS: an integrated system for processing, archiving, and disseminating high-volume earth science imagery and associated products, July 1996.

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

[6] Hillyer, B. K., Rastogi, R., and Silberschatz, A. Scheduling and data replication to improve tape jukebox performance. In *Proceedings of the 15th International Conference on Data Engineering* (Sydney, Australia, 1999), IEEE Computer Society Press, pp. 532–541.

[7] Hillyer, B. K., and Silberschatz, A. 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, 1996), ACM Press, pp. 170–179.

[8] Hillyer, B. K., and Silberschatz, A. Random I/O scheduling in online tertiary storage systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Canada, 1996), ACM Press, pp. 195–204.

[9] Hillyer, B. K., and Silberschatz, A. 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, MD, March, 1998), IEEE Computer Society Press.

[10] Inmon, B. The Role of Nearline Storage in the Data Warehouse: Extending your Growing Warehouse to Infinity. Technical white paper, 1999. Provided by StorageTek. http://billinmon.com/library/whiteprs/st_nls.pdf.

[11] Jagadish, H. V., Lakshmanan, L. V. S., and Srivastava, D. Snakes and sandwiches: Optimal clustering strategies for a data warehouse. In *Proceedings ACM SIGMOD International Conference on Management of Data* (Philadephia, Pennsylvania, 1999), ACM Press, pp. 37–48.

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

[13] Kobler, B., Berbert, J., Caulk, P., and Hariharan, P. C. 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* (Monterey, CA, 1995), IEEE Computer Society Press, pp. 65–76.

[14] More, S., and Choudhary, A. Scheduling Queries on Tape-resident Data. Tech. Rep. CPDC-TR-2000-01-001, Center for Parallel and Distributed Computing, Northwestern University, January 2000. http://www.ece.nwu.edu/cpdc/TechReport/1999/11/CPDC-TR-2000-01-001.html.

[15] More, S., and Choudhary, A. Tertiary storage organization for large multidimensional datasets. In *8th NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies and 17th IEEE Symposium on Mass Storage Systems* (College Park, MD, March 2000), IEEE Computer Society Press, pp. 203–209.

[16] More, S., Muthukrishnan, S., and Shriver, E. Efficiently sequencing tape-resident jobs. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pennsylvania, June 1999), ACM Press, pp. 33–43.

[17] Myllymaki, J., and Livny, M. Disk-tape joins: synchronizing disk and tape accesses. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Ottawa, Canada, 1995), ACM Press, pp. 279–290.

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

[19] Pinedo, M. *Scheduling Theory, Algorithms and Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[20] Sarawagi, S. Database systems for efficient access to tertiary memory. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems* (Monterey, CA, 1995), IEEE Computer Society Press, pp. 120–126.

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

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

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