Scheduling Tertiary I/O in Database Applications *

Sunil Prabhakar Divyakant Agrawal Amr El Abbadi Ambuj Singh
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

We study the problem of scheduling I/O requests for tertiary storage libraries to improve performance. The focus is on scheduling policies that process all requests on a loaded medium before unloading it. For single drive settings an efficient algorithm that produces optimal schedules is developed. For multiple drives the problem is shown to be NP-Complete. Efficient and effective heuristics are presented for the multiple drives case. The scheduling policies developed achieve significant performance gains over more naive first come first server policies. The study is general enough to be applicable to any storage library handling removable media, such as tapes or optical disks.

1 Introduction

With the recent improvements in network and processor speeds, several data intensive applications have become more feasible than ever before. Examples of such applications include digital libraries, data warehousing, data mining, large scale scientific modeling and other multimedia applications. These applications are characterized by very large computational and storage requirements. In the present commercial setting and in the foreseeable future, the only practical solution for storing such large volumes of data is tertiary storage. Although tertiary storage, in particular magnetic tape, has historically been used primarily for archival or backup purposes, the exploding storage requirements and the higher cost of secondary storage are forcing computer architects and designers to re-evaluate the role of tertiary storage.

Commercially available automatic tape and disk libraries or juke boxes provide automated access to large amounts of tertiary storage. These libraries can hold hundreds or thousands of media. Throughout this paper we will use the term *medium* to refer to a tertiary storage unit such as a magnetic or optical tape or an optical or magneto-optical disk. A library typically has a small number of read/write drives and a few robot arms that load and unload media to and from these drives. Upon receiving a request, the robots will load the desired medium onto one of the drives (if there is already a medium loaded on the drive - it may need to be rewound in the case of tapes, ejected and replaced onto the shelf). The drive then loads the new medium, seeks to the requested location and then performs the I/O. The time required to switch a medium on a drive is usually of the order of a few seconds, and the time required to seek an entire medium may vary significantly. Typically, for optical disks the seek time is much smaller than the switch time while in the case of tapes the seek time may dominate the switch time. For example, the SONY OSL-2001 juke box can switch optical disks in about 5 seconds and the average seek time for optical disks is 28 - 40 milliseconds. The DLT4700 mini tape library has a load/unload cycle of 29 seconds and the average seek time is about 68 seconds. Processors, on the other hand, are many orders of magnitude faster than these switch and seek times. These differences in speed make it desirable to determine an optimal order of execution for a set of tertiary storage requests.

We study the problem of scheduling I/O requests for robotic libraries to improve performance. Although tapes and optical disks, the two leading tertiary storage technologies today, differ significantly in their access characteristics (one is sequential and the other is random access) our study is general enough to be applicable to both. The rest of this paper is organized as follows. Section 2 describes the problem

*Work supported by a research grant from NSF/ARPA-/NASA IRI9411330 and NSF instrumentation grant CDA-9421978.

and related work. Section 3 derives an optimal solution for the problem of scheduling on a single drive. In Section 4 the more general problem of scheduling with multiple drives is shown to be NP-Complete. In Section 5 the performance of the optimal and heuristic solutions for the scheduling problems is analyzed using simulation. Section 6 concludes the paper.

2 Problem description

The tertiary scheduling problem can be formally described as follows:

Definition 1 *Given a set of media, S, a number of drives, d, a number of robotic arms, A, a set of requests, R, each request $r \in R$ consisting of a medium number, a start block, a number of blocks and a read/write flag, what is the best schedule for servicing these requests such that the average waiting time for all requests is minimized.*

Note that there are many options for the choice of the optimization function [8]. We have chosen the average waiting time because the major concern for tertiary library performance is the high access latency. Thus reducing the latency (waiting time) seen by each request is important. It is assumed that requests do not span more than one medium. Such requests can be broken into multiple independent requests for single media.

The problem of task or job scheduling has been extensively studied in the literature [6, 8]. The problem of robotic library scheduling however, is not addressed by any of the existing studies. If the optimization function considered is the *makespan* or total time to service all requests, then the scheduling problem is very similar to the well known Traveling Salesperson Problem. However, this is not the case when considering the average waiting time, as in our problem.

Recent work on tape scheduling [3, 4] has studied efficient processing schedules for I/O requests for single tapes. In [3], Hillyer and Silberschatz have analyzed algorithms for scheduling batched requests for a single serpentine tape that is loaded on the drive. Li and Orji [4] have studied efficient scheduling policies for linear tapes. Both these studies do not consider scheduling requests for more than one medium. Work has also been done on optimizing the performance of relational database management systems that incorporate tertiary storage [7, 12].

It has been shown that I/O traffic is bursty [11], i.e. a large number of requests arrive together followed by a long period during which no more requests arrive. Under these settings, the scheduling problem can be simplified by considering the requests in each burst independently. Further, it can be assumed that these requests arrive together, and thus the problem can be viewed as off-line scheduling with requests arriving together rather than online scheduling with requests arriving continuously. This technique has also been used for studying disk scheduling policies [1].

For robotic library scheduling there are two contradictory factors that come into play: reducing seek and switching costs by scheduling requests for the medium that is currently loaded onto a drive versus servicing short seek requests on other media before servicing requests which require large seeks on the currently loaded medium. The importance of the two factors is determined by the relative values of the switch and seek times. If the switch time is large compared to the expected or average seek time, as is the case for all optical disk and most tape systems, then it is unlikely that switching to a new medium before servicing all requests on the currently loaded medium will be beneficial. On the other hand, if the switch time is small compared to expected or average seek times, which is true for some tape systems, then switching media without servicing all requests and then later returning to service the remaining requests may result in lower waiting times. There are therefore two avenues along which the scheduling problem can be studied. The first, which we call *Full Medium* scheduling, considers only those scheduling policies that service all requests for a loaded medium before unloading it. The second, which we call *Partial Medium* scheduling, considers schedules that may unload a medium without servicing all the requests for the medium and then reloading the same medium to service the remaining requests after servicing requests on other media. In [9], we show that minimal gains are achieved by allowing extra media switching. In this paper we perform a complete study of the *Full Medium* scheduling problem.

3 Scheduling on a single drive

We begin by studying the problem of *Full Medium* scheduling for a single drive. For this version, the problem simplifies to determining the order in which the media are loaded onto the drive. This is because the minimum number of switches are made when each medium (for which there are requests) is loaded only once. When a given medium is loaded, all requests for that medium are serviced before it is ejected from the drive.

3.1 Optimization function

In order to understand the factors involved in the scheduling problem, we derive an expression for the average waiting time. All requests are assumed to arrive at the same time. The average waiting time is sim-

ply the total waiting time for all requests divided by the number of requests. Consequently, for any given set of requests, minimizing the average waiting time is the same as minimizing the total waiting time. In the remainder of the paper we use the total waiting time instead of the average waiting time since both reflect the same optimization function. Since all items arrive together (at time 0), we define the optimization function as follows:

Definition 2 *The total waiting time for a schedule of n requests is the sum of the waiting time of all requests. The waiting time for a request is the time (since arrival) at which the data transfer begins.*

The goal of our scheduling policies will therefore be to minimize the value of the total waiting time. The ordering of the media refers to the sequence in which they are loaded onto the drive. Throughout the paper, n_i denotes the number of requests for medium i, m is the number of media with requests and T is the time required to switch a medium. The switch time is the sum of the times to eject the currently loaded medium, the robot to pick the ejected medium, place it on shelf, move to the next medium, put the new medium into the drive and the drive to load the new medium (we assume that this time is constant, independent of the location of the drive or medium, as given in most product specifications). An expression for the total waiting time is given by Equation 1.

$$W = \sum_{i=1}^{m} \left(C_i + T \cdot i \cdot n_i + n_i \sum_{j=1}^{i-1} P_j \right) \qquad (1)$$

The total waiting time for all requests can be viewed as consisting of three components for each medium. The first component, C_i, is the waiting time incurred by requests for the ith medium once it has been loaded onto the drive. This time is independent of the order in which the media are loaded. The second component, $T \cdot i \cdot n_i$, is the waiting time for requests of the ith medium due to the media switches that precede medium i. This time is the same for all requests on medium i and is dependent only on the ordinal position of the medium in the scheduling order. The third component, $n_i \sum_{j=1}^{i-1} P_j$, is the waiting time due to the processing of the requests on the preceding media (P_k is the processing time for the kth medium, and is independent of the loading schedule). It is the same for each request of a given medium and is equal to the sum of the processing times for the preceding media.

3.2 Optimal solution

We now present an algorithm that determines an optimal schedule for a single drive. The proof of optimality of this schedule is available in [10].

Theorem 1 *Any schedule of m media which satisfies the following equation is optimal:*

$$\forall i \;\; 1 \leq i < m, \;\; \frac{n_i}{T + P_i} \geq \frac{n_{i+1}}{T + P_{i+1}}.$$

Finding an optimal solution for full medium scheduling on a single drive is therefore reduced to the problem of sorting the media in non-increasing order of the ratio of number of requests to the sum of the processing time and the switch time. This is easily accomplished by using any efficient sorting algorithm, such as *quicksort*. The time complexity for determining the optimal order is therefore $O(N + m \log m)$, where N is the number of requests and m is the number of media with requests. The $O(N)$ component is the time required to estimate the processing time, P_i, for each tape, for which the seek and transfer times for every request have to be determined. Note that we are assuming that the time to estimate the seek and transfer times for each request is constant. Note also that the order of processing of requests for a given medium is not restricted, i.e. any order can be used. This order can be as simple as the order of the block numbers for linear tapes or more complex such as those suggested by Hillyer and Silberschatz in [3] for serpentine tapes or any disk scheduling policy for optical disks. For medium scheduling it is necessary only to be able to determine the seek and transfer times for each request. We have not made any assumptions about the seek function or transfer rates for the media either (they can be arbitrarily distributed) we only require good estimates of the actual times to determine an optimal schedule.

4 Multiple drives

We now consider the scheduling problem in the presence of more than one drive. For the multiple drive case, the total waiting time can be viewed as the sum of the waiting times for each of the drives. For each drive, Equation 1, developed in Section 3 gives the total waiting times. The total waiting time for all requests processed on a single drive can be generalized from Equation 1. Assuming d drives, the total waiting time for all drives can be written as:

$$W = K + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (P_i^k + T) \right) \qquad (2)$$

where the superscript k identifies the drive, and $K = \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(C_i^k + T \cdot n_i^k \right)$ is a constant independent of the assignment of media to drives or the order of processing of media on drives.

We note that the problem of minimizing the total waiting time is the same as the problem of minimizing the total completion time. The only difference between the two measures is a constant amount, equal to the sum of the transfer times for all the requests. Since this amount is independent of the order of processing, we can say that the total completion time, W', for all requests is given by

$$W' = K' + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (P_i^k + T) \right) \qquad (3)$$

where K' is a constant independent of the schedule.

Theorem 2 *The problem of multidrive full medium scheduling is NP-Complete.*

The proof of Theorem 2 is presented in [10].

5 Heuristics and simulation results

In this section we present simulation results to validate the theoretical results discussed in the earlier sections. We begin by describing the experimental setup used for the simulations. Next the benefits of *Full Medium* scheduling are presented. These are followed by the experiments with single and multiple drive settings.

5.1 Experimental setup

The algorithms were evaluated using an event driven simulator for robotic libraries developed using *CSIM* [13], a simulation library package. A simulation model of a general robotic library consisting of a variable number of drives, robot arms and media has been developed. The numbers of each of these is configurable. The performance of the drives and robot arms and the characteristics of the media are also configurable. Table 1 lists some of the parameters that can be adjusted for the model. The model mimics an actual robotic library. The drives operate independently of each other, except that they share the robot arms to pick and put media.

Experiments were conducted with both single and multiple drive settings. Two sets of input request patterns were used for our experiments. The first set consisted of requests distributed randomly across all the media in the library. The second set consisted of a skewed "hot-cold" distribution where 80% of the tapes were accessed 20% of the time and the remaining 20% of the tapes were accessed 80% of the time. All requests generated were of the same size, 5 blocks, where a block is 512 KB. For each experiment, the number of tapes was varied from 10 to 100, and the number of requests was always set to be ten times the number of tapes, e.g. for 20 tapes, we generated 200 requests. The values of the model parameters used for the experiments are shown in Table 1. These are based on the Exabyte EXB-480 Tape Library configured with Exabyte EXB Mammoth drives [2].

Parameter	*Value*
Number of Drives	1 or 4
Number of Robotic arms	1
Number of media in the library	10-100
Number of requests	100-1000
Time to remove medium from drive	10 sec
Time to place medium on drive	10 sec
Time to move between media	2 sec
Seek(Rewind) overhead (sec)	0.1 (0.1) sec
Seek(Rewind) rate (MB/s)	188 (193) MB/s
Transfer rate of the drive (MB/s)	3 MB/s
Time to eject a medium (sec)	8 sec
Time to load a medium (sec)	10 sec

Table 1: Table of Model Parameters

5.2 Validation of *Full Medium* scheduling

In order to validate our hypothesis that *Full Medium* scheduling is beneficial, we conducted experiments to compare the performance of variants of the naive First Come First Serve (FCFS) policy. Three versions of FCFS algorithms were examined. The first, called FCFS, schedules requests in the order in which they are received. Note that even though we assume that the arrival time for all requests is the same for the purposes of our waiting time derivation, we do have the notion of ordered arrival. The reason for this is that the order of magnitude of the average waiting time for tertiary storage devices is a few tens or hundreds of seconds, whereas the requests will arrive within a few milliseconds of each other due to the I/O burst. The second variation, called FCFS_II, is a full medium scheduling algorithm which determines the order of media according to the order of the requests. That is, the first medium is the one on which the first request is located, the second medium is the one on which the earliest request not for the first medium is located, and so on. FCFS_II serves all requests on a given medium before ejecting the medium, however it serves these requests in the order in which they arrive. The third algorithm, called FCFS_III, optimizes over FCFS_II by processing the requests on a given medium in a more efficient manner - in the order of ascending tape blocks. The results for single drive and multiple drives were very similar and therefore only the results for the multiple drive setting are shown. Figure 1 shows the performance of the three FCFS policies with the hot-cold request patterns. The results for the random distribution were very similar and are there-

fore not shown [10]. It can be observed that the algorithms that service all requests for the loaded medium before those for other media (FCFS_II and FCFS_III) reduce the average waiting time for requests significantly. Reordering the requests for a given medium results in even further reductions in the average waiting time (FCFS_II versus FCFS_III). This confirms our hypothesis that *Full Medium* scheduling is beneficial even though it may not be optimal. It is important to note that these experiments were based upon data for tape libraries for which the seek time dominates the switch time. Hence, we expect that *Full Medium* scheduling will result in even greater gains when the switch time dominates the seek time such as optical disk libraries.

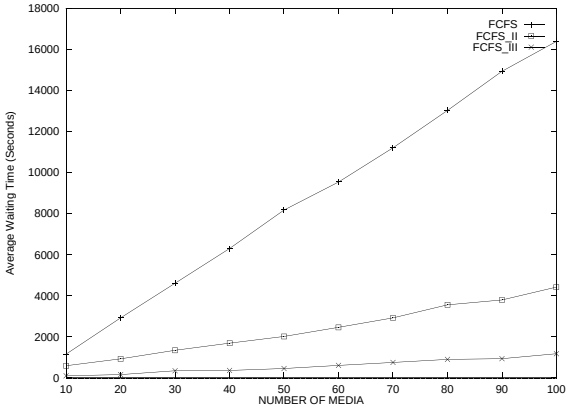

Figure 1: Variations of FCFS for multiple drives with Hot-Cold distribution

5.3 Experimental results

For a single drive, we have shown that an optimal algorithm exists. The algorithm, which we call OPT, schedules media in non-increasing order of the ratio of number of requests to the sum of the processing time and the switch time. Although the optimal algorithm for single drives, OPT, is simple – it requires knowledge of the processing time for each medium. It may not always be possible to determine this time. For example, the scheduling algorithm for requests on a given medium may not be known or the exact layout of the data on the medium may not be known. In the absence of such information, we propose the following heuristic: *Schedule the media in the order of non-increasing number of requests.* This heuristic, which we call *Number*, does not guarantee optimal solutions but is very easy to implement as it requires no knowledge of the processing or switch times. Several experiments were conducted to evaluate the effectiveness of this heuristic. It was observed that the heuristic performs very well and is within 1% of the optimal [10]. It should also be noted that OPT and *Number* perform better than FCFS_III which does not reorder the media.

For multiple drive scheduling no efficient optimal solution was found. Given the difficulty of scheduling on multiple drives, we look for approximate solutions that give reasonable performance. The simple algorithm, OPT, that generates an optimal solution for the single drive case (order tapes in non-increasing order of the ratio of number of requests to the sum of the processing time and switch time) does not guarantee optimal results for the multiple drive case. An analogous algorithm for the minimum weighted completion time problem ($MWCT$) is guaranteed to produce a solution that is within $(1+\sqrt{2})/2$ (approximately within 20%) of the optimal [8]. This bound does not necessarily hold for the multiple drive problem due to the other constant factors in the cost. Given its good performance guarantee for the $MWCT$ problem, we evaluated this algorithm for the multiple drive problem. We also evaluated the simpler heuristic based solely on the number of requests per medium, *Number*.

Since no efficient optimal algorithm was developed for the multiple drive case, we used a brute force algorithm to generate the optimal solution for the purpose of comparison. Given a set of m media and d drives, there are d^m different allocations of media to drives. Given a set of media to be processed on a drive, the optimal schedule of these media can be determined using OPT, the optimal algorithm for single drives. Therefore, for each allocation of media to drives, we can determine an optimal schedule. The optimal schedule for the multiple drives is taken to be the allocation with the least average waiting time. Since determining the optimal schedule requires exponential running time, we were limited to small numbers of tapes. Several experiments were conducted with 10 tapes using both request distributions to compare the performance of OPT and the brute force algorithm. With 10 tapes, we found that the OPT algorithm for single drives performs surprisingly well in the multiple drive setting as well. In most cases, OPT produces schedules that are within 1% of the brute force algorithm for multiple drives. The performance of the simpler *Number* heuristic was compared with that of OPT for larger numbers of tapes with multiple drives. The results for the random and the hot-cold distribution, which are very similar to those for a single drive, are shown in Figure 2. It can be seen that the *Number* heuristic generates solutions that are very close to those produced by OPT. As for the single drive case, both OPT and *Number* perform better that FCFS_III with multiple

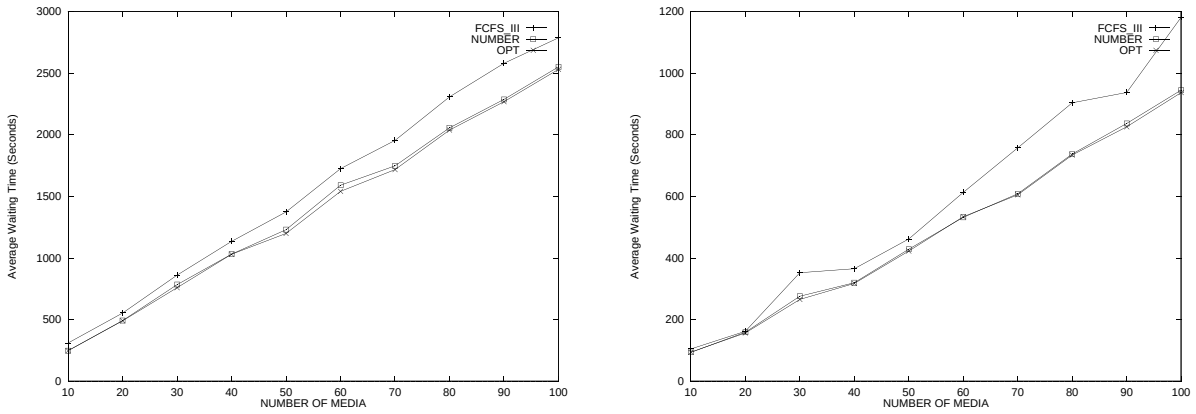

Figure 2: Comparison of the *Number* heuristic with OPT for multiple drives with (a) Random and (b) Hot-Cold request distributions

drives. Moreover, in all our experiments with single as well as multiple drives, we observed that the OPT algorithm achieved an average of 85% (random distribution) and 94% (hot-cold distribution) reduction in the average waiting times over the FCFS policy. It is interesting to note that despite the inefficiency of the naive FCFS policy, it is used in commercial systems [5].

6 Conclusion

We have investigated the problem of efficient I/O scheduling for tertiary libraries. The focus of the study has been on scheduling policies that do not switch media before servicing all requests for the loaded medium, (*Full Medium* scheduling policies). We have developed optimal (for a single drive) and near optimal solutions for the scheduling of I/O requests for robotic libraries. The algorithms are efficient, simple to implement and achieve significant improvements (85% - 94% for our workload) over the naive FCFS policy. They make no restrictions on the requests, the distribution of these requests, the seek or rewind times for the drives or the switch time for the library. Most importantly, the algorithms are applicable to any storage library that handles removable media, such as tapes or optical disks.

References

[1] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In *Foundations of Computer Science*, 1996.

[2] EXABYTE. Products. http://www.Exabyte.COM:80/Products/, Oct. 1996.

[3] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Canada, 1996.

[4] J. Li and C. Orji. I/O scheduling in tape-based tertiary systems. *Journal of Mathematical Modelling and Scientific Computing*, 6, 1996.

[5] R. Moore. Private Communication, 1996.

[6] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems with applications to Production Systems and Project Management.* John Wiley and Sons, Inc., 1993.

[7] J. Myllymaki and M. Livny. Disk-tape joins: Synchronizing disk and tape access. In *Joint International Conference on Measurement and Modeling of ComputerSystems. SIGMETRICS '95/PERFORMANCE '95*, pages 279–90, Ottawa, Canada, 1995.

[8] M. Pinedo. *Scheduling Theory, Algorithms and Systems.* Prentice-Hall International Series in Industrial and Systems Engineering. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[9] S. Prabhakar, D. Agrawal, and A. El Abbadi. Impact of media exchanges on robotic libraries. Technical Report TRCS97-07, Dept. of Computer Science, Univ. of Calilfornia, Santa Barbara, 1997. http://www.cs.ucsb.edu/TRs /TRCS97-07.ps.

[10] S. Prabhakar, D. Agrawal, A. El Abbadi, and A. Singh. Efficient I/O scheduling in tertiary libraries. Technical Report TRCS96-26, Dept. of Computer Science, Univ. of Calilfornia, Santa Barbara, 1996. http://www.cs.ucsb.edu/TRs /TRCS96-26.ps.

[11] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *USENIX*, pages 405–420, Winter 1993.

[12] S. Sarawagi. Query processing in tertiary memory databases. In *Proc. of the 21st Int. Conf. on Very Large Data Bases*, pages 585–596, San Francisco, California, 1995. Morgan Kaufmann.

[13] H. D. Schwetman. CSIM: A C-based, process-oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, pages 387–396, December 1986.