Efficient I/O Scheduling in Tertiary Libraries *

Abstract

With the recent improvements in network and processor speeds, several data intensive applications have become much more feasible than ever before. The only practical solution for storing such enormous amounts of data is tertiary storage. Automated access to tertiary storage is made possible through robotic libraries for tapes and optical disks. Due to the slow speeds of operation of the drives and library robotics, access times for such libraries are high resulting in the accumulation of I/O requests. In this paper we study the problem of scheduling these requests for efficient performance. We focus 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 drive case. The scheduling policies developed achieve significant performance gains over more naive policies. The algorithms developed are simple to implement and are not restrictive. The study is general enough to be applicable to any storage library handling removable media, such as tapes and optical disks.

1 Introduction

With the recent improvements in network and processor speeds, several data intensive applications have become much more feasible than ever before. Examples of such applications include digital libraries, large scale scientific modeling and other multimedia applications. These applications are characterized by very large computational and storage requirements. Applications such as the Earth Observing System Data and Information System (EOSDIS) [KBCH95] have estimated data generation rates of well over a terabyte a day. In the present commercial setting and in the foreseeable future, the only practical solution for storing such enormous amounts 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 high cost of secondary storage are forcing computer architects and designers to re-evaluate the role of tertiary storage[CHL93].

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

Commercially available automatic tape and disk libraries or jukeboxes (e.g. [EXA96, SON96, Qua96]) 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 drives or readers 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 starting location of requested data and then transfers the requested data. 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 be as high as a few minutes. For example, the SONY OSL-2001 jukebox can switch optical disks in about 5 seconds and the average seek time for optical disks is 28 - 40 milliseconds [SON96]. The DLT4700 mini tape library has a load/unload cycle of 29 seconds and the average seek time is about 68 seconds [Qua96]. Therefore for the optical disks the seek time is much smaller than the switch time and for the tapes the seek time is larger than the switch time. Processors, on the other hand, are many orders of magnitude faster than these switch and seek times, with the consequence that the potential for a large number of requests accumulating at the library is high. Processing requests in first come first serve (FCFS) order in such a system may result in poor performance. For example, consider three requests that arrive at the library: the first is for blocks 5 through 10 on tape A, the second is for blocks 1 through 20 on tape B and the third is for blocks 11 through 20 on tape A. If these are processed in FIFO order, we see that tape A will be loaded, then it is ejected and tape B is loaded which is then ejected and tape A is reloaded and the drive seeks over the first 10 blocks. In contrast, if we service the requests in the order first, third and second, we see a reduction in the total service time, since we eliminate one eject, one load and a seek on tape A. Thus, given a set of pending requests at the library, it is desirable to determine an optimal order of execution such that some performance metric is optimized. This metric could be the average waiting time, or the time to completion of the last request or the throughput of the system.

In this paper, we study the problem of scheduling I/O requests for robotic libraries to achieve better performance. Although the problem in its full generality is quite complex, we make some simplifying assumptions for the purposes of this study. These assumptions are described in the next section. 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 and discusses related work. Section 3 addresses the problem of scheduling on a single drive. In Section 4 the more general problem of scheduling with multiple drives is analyzed. In Section 5 the performance of the optimal and heuristic solutions for the scheduling problems based on simulations are presented. Section 6 concludes the paper.

2 Problem Description

In this section we present the problem of I/O scheduling for robotic libraries and relate it to work that has been done in other areas. It is shown that the problem of robotic library scheduling is different from the problems that have already been studied. However, due to its similarity to some known scheduling problems which are hard to solve, we expect that it too is a difficult problem. Therefore, as an initial step towards solving the general problem, we focus on a simplification of the problem as described in the subsection 2.3.

2.1 General Problem

We begin with an informal description of the problem. Given an automatic library of removable media (tape or optical disk) consisting of drives, robotic arms and media, and a set of requests for various data items stored on the media, what is the most efficient schedule for the requests? In order to compare the efficiency of potential scheduling policies, it is necessary to define a quantitative measure of performance. This measure, also known as the optimization function, could be one of several possibilities. Some of the popular optimization functions that have been studied in the literature include [Pin95, MP93]:

- *Makespan:* - the time at which the last request is completed.
- *TWCT:* - the Total Weighted Completion Time (each job has an assigned weight).
- *TWT:* - the Total Waiting Time for all jobs.
- *Throughput:* - the average number of requests serviced per second.

Since the major concern for tertiary storage performance is the slow speed, we are interested in reducing the amount of time that a given request has to wait before it is serviced. We have therefore focused on minimizing the average waiting time over all the requests as our optimization function. For I/O requests, the waiting time is taken to be the time from the arrival of the request (in our study all requests arrive together at time 0) to the beginning of the data transfer for the request. This includes the time to perform necessary seek operations. A more formal definition of the problem is now presented.

Definition 2.1 *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.*

The above scheduling problem is related to the problem of scheduling tasks on multiple processors but is more complex. Intuitively, there are two factors which need to be considered when deciding which item to service next once a drive becomes available. The first factor is that the cost of servicing a request depends upon which medium is currently loaded on the drive. If the medium which holds the requested data is not loaded, then the currently loaded medium has to be rewound (if the media are tapes) and ejected followed by the loading of the medium with the requested data and seeking to the desired location from the beginning of the medium. If on the other hand, the correct medium is loaded then the amount of time that the item waits is only the time to seek to the correct location from the current location of the head. Therefore, a reasonable heuristic is that requests for the currently loaded medium should be serviced before requests for other media. The second factor is that, if the seek time for the drive is much larger than the switch time (such as in the case of many tape drives), servicing requests for media that are not currently loaded before servicing a request that requires a large seek on the currently loaded medium may reduce the average waiting time. Consider the following example.

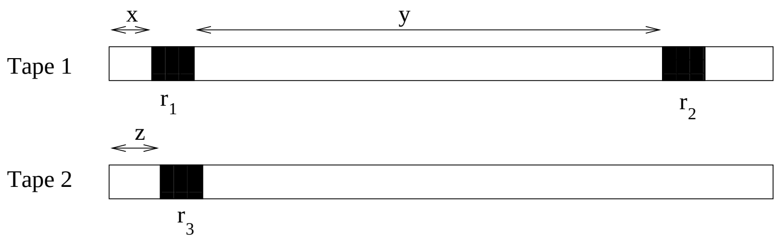


Figure 1: Distribution of Requests for Example 2.1

Example 2.1 Consider a library with a single drive and three requests. The first request, r_1, is for tape 1, starting at block x, the second, r_2, is for tape 1, starting y blocks after r_1 and the third, r_3, is for tape 2, starting at block z, as shown in Figure 1. Each of the requests is for a single block of data. Suppose that r_1 is serviced first, resulting in tape 1 being loaded on the drive and the head being positioned at the beginning of block 2. In this example we ignore the time required to seek or rewind over one block. If request r_2 for tape 1 is serviced next, followed by r_3, the waiting times will be:

$$\text{Wait time for } r_1 = T + S_x$$
$$\text{Wait time for } r_2 = T + S_x + X_1 + S_y$$
$$\text{Wait time for } r_3 = T + S_x + X_1 + S_y + X_2 + R_{x+y} + T + S_z$$
$$\text{The total wait time will be } 4T + 3S_x + 2X_1 + 2S_y + X_2 + R_{x+y} + S_z$$

where S_i is the time required to seek i blocks, X_i is the time required to transfer i blocks, R_i is the time required to rewind the tape i blocks and T is the time to switch tapes on the drive. On the other hand, if we service r_3 after r_1, followed by r_2, the wait times will be:

$$\text{Wait time for } r_1 = T + S_x$$
$$\text{Wait time for } r_3 = T + S_x + X_1 + R_x + T + S_z$$
$$\text{Wait time for } r_2 = T + S_x + X_1 + R_x + T + S_z + X_3 + R_z + T + S_{x+y}$$
$$\text{Therefore the total wait time is } 6T + 3S_x + 2X_1 + 2R_x + 2S_z + X_3 + R_z + S_{x+y}.$$

Comparing the two total wait times, we see that it is more beneficial to service r_3 before r_2 if

$$2S_y + X_2 + R_{x+y} > 2R_x + 2T + S_z + X_3 + R_z + S_{x+y}$$

Assuming that rewind and seek rates are almost the same, i.e. $R_{x+y} = S_{x+y}$ and $S_z = R_z$, we get $2S_y > 2R_x + 2R_z + 2T$ which implies that $S_y > R_x + R_z + T$. In other words, if the time to seek over y blocks is larger than the time to rewind over x blocks, plus the time to rewind over z blocks plus the switch time, then it is more beneficial to switch to tape 2 and service r_3 and then switch back to tape 1 and service r_2. We now substitute some numbers for a commercial tape drive, the Exabyte EXB-Mammoth [EXA96] using 170mm, 20GB tapes. The size of a block is chosen to be 512 KB, which gives almost 40,000 blocks per tape. Let x and z be 1 block each. The tape switch time for the mammoth drive is around 40 seconds. The drive is rated to have a seek rate of 188 MB/s which is the same as 376 blocks/second, with a 0.1 second startup time. Therefore, the right hand side of the above inequality, $R_x + R_z + T = 40.405$ seconds. In order for the left hand side to be larger than this we require that $y > (40.405 - 0.1) * 376$. Thus if y is greater than 15,200 blocks, it is preferable to switch tapes after processing request r_1, processing r_3 and then returning to do r_2. Since the length of the tape is 40,000 blocks, the above scenario is possible. Therefore in this example, servicing all requests for the loaded medium before switching to another medium results in sub-optimal performance. These two contradictory factors make the task of finding an optimal schedule for the requests very difficult.

2.2 Related Work

The problem of task or job scheduling has been extensively studied in the literature [Pin95, MP93, Par95]. Related areas are discussed here, with emphasis on the applicability of existing results to our problem. Scheduling tasks are traditionally categorized as processor scheduling or shop scheduling. Processor scheduling refers to problems where a set of tasks has to be executed by a set of processors. Any task can be executed on any processor (the rates may vary depending on the task and the processor). Shop scheduling is concerned with finding optimal schedules for tasks that need to be processed by several machines in order to be completed. There are many variations of both types of scheduling problems. The problem of shop scheduling is not related to the problem of robotic library scheduling because the essential element of jobs requiring processing by several different machines is absent since in robotic library scheduling any one of the available drives can service the request. The processor scheduling problem seems to be very similar to our library scheduling problem.

Several variations of the problem of scheduling tasks on processors have been investigated[Par95, GJ79]. The various parameters studied include single versus multiple processors, preemptive versus non preemptive tasks, tasks with and without release times and deadlines, precedence constraints between tasks, tasks with weights or priorities and tasks with setup times. Various optimization functions have also been studied such as *Makespan*, *Weighted completion time*, *Total weighted tardiness* (for tasks with deadlines), *Maximum lateness*, etc.

The problem of robotic library scheduling however, is not addressed by any of the existing studies. We now explore the similarities and underscore the differences between these two. The drives and processors play analogous roles, as do the requests and tasks. The processing time for tasks however, does not easily map onto any simple concept for the library. It is tempting to relate it to the transfer time, but then we do not capture the seek and switch latencies that delay the processing. We could include these latencies as part of the processing time for each request, however this leads to a further complication. This is because the latency of access for a request is not constant rather it depends largely on the state of the drives and the location of the requested data. We therefore have to work with variable task processing times which is a problem that has not been addressed in earlier studies.

One variation of the processor scheduling problem appears to address a similar problem - tasks with setup times. This problem addresses the case where a setup time is required before a certain job can be processed. The latencies can be viewed as setup times, however latencies for requests are dependent on the order in which requests are processed. Hence this variation is also not applicable. Another variation of the setup times problem was studied in [BD78] where tasks are grouped into classes and a setup time has to be paid for a job if the preceding job is not from the same class. This problem too does not apply because it is too restrictive. Problems with sequence dependent setup times have been studied wherein the setup time required for a given request is a function of the previous request. This formulation seems to be most applicable to the problem of robotic library scheduling. However, the studies have focussed on the single processor case and the optimization function used is the *Makespan*. In such a situation, the problem of minimizing the makespan reduces to the well known Traveling Salesperson Problem (TSP)[LLKS85]. This problem is known to be NP-complete and has been studied extensively. If multiple processors or drives are present, then the problem of minimizing the makespan reduces to the Multiple Traveling Salesperson Problem which is also NP-complete. If, however, the optimizing function considered is the total waiting time or the total completion time, as in our study, then the library scheduling problem does not reduce to the TSP problem.

The robotic library scheduling problem is also related to the problem of disk scheduling which has been studied extensively in the operating systems community. For disk scheduling, a number of requests need to be serviced on different locations. The drive has to seek between requests that are not contiguously located. Numerous algorithms have been proposed for efficiently servicing disk requests, such as LOOK, C-LOOK, C-SCAN [Teo72] etc. The major difference between disk scheduling and library scheduling is that tertiary storage media such as tapes or optical disks are removable as opposed to fixed magnetic disks. The algorithms developed for disks all assume that the medium is never switched which is not true for robotic libraries.

More recent work on tape scheduling [HS96b, LO96] has studied efficient processing schedules for I/O requests for single tapes. In [HS96b] Hillyer and Silberschatz have analyzed algorithms for scheduling batched requests for a single serpentine tape that is loaded on the drive. Li and Orji [LO96] have studied efficient scheduling policies for linear tapes. Both these studies do not consider scheduling requests for more than one medium. Sarawagi and Stonebraker have worked on the problem of ordering I/O requests for tertiary storage media when tertiary storage is directly incorporated in a relational database [SS93]. The reordering is based upon knowledge of the state of the tertiary storage and the semantics of the requests that are made for performing join operations. Their work is not applicable outside the relational database framework. Optimization of the performance of join operations in databases has also been studied by Myllymaki and Livny [ML95] under the assumption that one relation is on tape and the other on disk. Other researchers have studied various aspects of tertiary storage systems. Myllymaki and Livny have investigated the benefits of executing tape and disk I/O in parallel [ML96]. In [FM96], Ford and Myllymaki have proposed a log structured file system for tertiary media. The benefits of striping in tape based systems has been studied by Drapeau and Katz [DK93] and also by Golubchik and Muntz [GM95]. In [GM95] a general open system model for striped libraries is developed. In [Sar95a, Sar95b], modifications to the architecture of database management systems are proposed for efficiently accessing tertiary storage directly. Other studies have looked at the problem of reorganization of data that is stored on tertiary media in order to improve retrieval performance [CDK+95, SS94]. Both studies focus on large arrays with known access patterns and make optimizations for these accesses.

2.3 Problem Specification

From the above discussion it is clear that the general problem of robotic library scheduling is difficult (possibly NP-complete). Instead of trying to solve the complete problem in one step, we first simplify it by restricting the problem domain. We make the following assumptions.

- Requests have no priorities (or they all have the same priority).
- All requests arrive at the same time (Non real-time).
- No deadlines for requests.
- No dependencies between requests.

As explained earlier, there are two contradictory factors that come into play for robotic library scheduling: reducing seek and switching costs by scheduling items for the medium that is already loaded onto the drive versus servicing short seek requests on other media before servicing requests on the currently loaded medium which require large seeks. The importance of the two factors is determined by the values of the switch and seek times. If the switch time is large compared to the expected or average seek time, 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 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 this paper we perform a complete study of the *Full Medium* scheduling problem.

For certain media, such as optical disks, the switch time is much larger than seek times. The average switch time for optical disks is of the order of a few seconds and the maximum seek times are of the order of tens or hundred of milliseconds. In the case of tapes where the seek time is usually larger than the switch time, *Full Medium* scheduling policies may be sub-optimal. However, even these schedules result in great savings when compared to more naive scheduling policies as described in later sections. In the remainder of the paper, we consider only those schedules that service all requests on a medium once it is loaded before ejecting it. The discussion is applicable to both tape libraries and optical disk libraries. For tapes, we assume that the tapes need to be rewound completely before they are ejected. There are two common formats in which tapes are commercially available, cassettes and cartridges. Cassettes, similar to audio or video tapes, consist of two spools between which the tape is reeled. Cartridges have a single spool on which the tape is stored and the second spool onto which the tape is reeled during access is usually available at the drive. Cartridges must be rewound before they are ejected from the drive. Cassettes could potentially be ejected without rewinding to the beginning, as can be done with audio tapes. However, in all commercial tapes, when data is stored, some control information needs to be written in order to enable the drive to locate the data when the tape is loaded again. For most current technologies, this data is written at the beginning of the tape. Some tape technologies with multiple load/eject zones do exist, where the data can be written and the tape ejected, but they are not yet very common.

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. Once a given medium is loaded, all requests for that medium are serviced before it is ejected from the drive. When a tape is loaded the head is positioned at the beginning of the tape, for optical disks, there is a similar initial position of the head. The order of processing the requests optimally on a medium is not important for our discussion. 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 [HS96a] for serpentine tapes or any disk scheduling policy for optical disks. When all requests have been serviced, the medium is rewound (in the case of tapes) to the beginning before it is ejected.

3.1 Optimization Function

In order to understand the various factors involved in the scheduling problem, we first derive an expression for the average waiting time. Since all items arrive together, the number of items in the problem is fixed. The average waiting time is therefore simply the total waiting time for all items divided by the number of items. Any schedule that minimizes the average waiting time, must also minimize the total waiting time. Therefore 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.

The goal of our scheduling policies will therefore be to minimize the value of the total waiting time. The following notation is used:

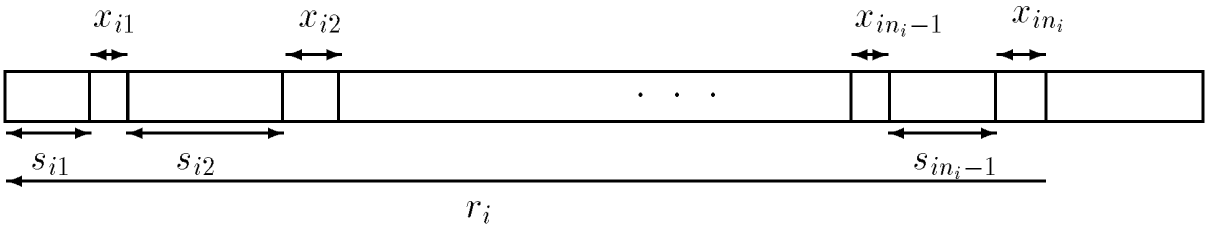


Figure 2: Notation used in the development for Tapes

- w_{ij} denotes the amount of time that the jth request (in the order of execution on the medium, as determined by the scheduling algorithm) on medium i waits before transfer begins.
- s_{ij} denotes the time required to seek from the end of the $(j-1)$th request to the first block of the jth request on medium i. For $j = 1$, it is the time required to seek from the initial position of the head after loading to the first request.

- T is the time required to switch a medium. It includes the time to eject the currently loaded medium, time for the robot to pick the ejected medium, place it on shelf move to the next medium, put the new medium into the drive and time for 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).
- n_i denotes the number of requests for medium i.
- x_{ij} is the time required to transfer data for the jth request on medium i.
- r_i denotes the time required to rewind medium i after all requests for the medium have been serviced. For optical disks this time may be absent or negligible. For tapes it is the time required to rewind to the beginning of the tape and the time to write the control information.
- W_i denotes the total waiting time for all requests on medium i.
- m is the number of media with requests.

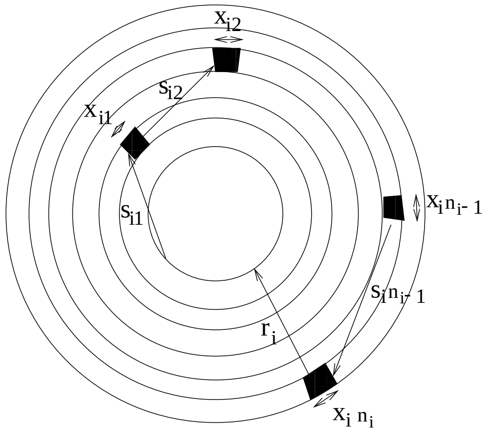


Figure 3: Notation used in the development for Optical Disks

Figure 2 shows the ranges on tape over which the above times are defined. Figure 3 shows the ranges on disk over which the above times are defined. In both figures, it is assumed that the order of processing of requests is $1, 2, \cdots, n-1, n$. In the following discussion the ordering of the media refers to the sequence in which they are loaded onto the drive. The waiting times for requests of the first medium are as follows:

$$\begin{aligned} w_{11} &= T + s_{11} \\ w_{12} &= T + s_{11} + x_{11} + s_{12} \\ &\vdots \\ w_{1n_1} &= T + s_{11} + x_{11} + s_{12} + x_{12} + \ldots + x_{1n_1-1} + s_{1n_1} \end{aligned}$$

The sum of the waiting times for requests for the first medium, W_1, is therefore:

$$W_1 = n_1 * T + \sum_{j=1}^{n_1} (n_i - j + 1) * s_{1j} + \sum_{j=1}^{n_1 - 1} (n_i - j) * x_{1j}$$

The waiting times for the second medium are:

$$\begin{aligned} w_{21} = &\quad Q_1 + T + s_{21} \\ w_{22} = &\quad Q_1 + T + s_{21} + x_{21} + s_{22} \\ \vdots & \\ w_{2n_2} = &\quad Q_1 + T + s_{21} + x_{21} + \ldots + x_{2n_2-1} + s_{2n_2} \end{aligned}$$

where $Q_1 = T + \sum_{j=1}^{n_1} s_{1j} + \sum_{j=1}^{n_1} x_{1j} + r_1$, is the time spent before the second medium is scheduled to be loaded. It includes the switch time for the first medium, the seek and transfer times for each request of the first medium and the time for rewinding the medium. The total time for the second medium, W_2 is:

$$W_2 = n_2 * (Q_1 + T) + \sum_{j=1}^{n_2} (n_2 - j + 1) * s_{2j} + \sum_{j=1}^{n_2 - 1} (n_2 - j) * x_{2j}$$

$$W_2 = 2Tn_2 + n_2 (\sum_{j=1}^{n_1} s_{1j} + \sum_{j=1}^{n_1} x_{1j} + r_1) + \sum_{j=1}^{n_2} (n_2 - j + 1) s_{2j} + \sum_{j=1}^{n_2 - 1} (n_2 - j) x_{2j}$$

Similarly, for the ith medium,

$$W_i = n_i * (Q_{i-1} + T) + \sum_{j=1}^{n_i} (n_i - j + 1) * s_{ij} + \sum_{j=1}^{n_i - 1} (n_i - j) * x_{ij}$$

where $Q_{i-1} = (i-1)T + \sum_{k=1}^{i-1} \sum_{j=1}^{n_k} s_{kj} + \sum_{k=1}^{i-1} \sum_{j=1}^{n_k} x_{kj} + \sum_{k=1}^{i-1} r_k$

$$W_i = iTn_i + n_i \left(\sum_{k=1}^{i-1} \left(\sum_{j=1}^{n_k} (s_{kj} + x_{kj}) + r_k \right) \right) + \sum_{j=1}^{n_i} (n_i - j + 1) s_{ij} + \sum_{j=1}^{n_i - 1} (n_i - j) x_{ij}$$

The total waiting time for all requests, W, is therefore:

$$W = \sum_{i=1}^{m} W_i = \sum_{i=1}^{m} \left(\sum_{j=1}^{n_i} (n_i - j + 1) s_{ij} + \sum_{j=1}^{n_i - 1} (n_i - j) x_{ij} \right) + \sum_{i=1}^{m} iTn_i + \sum_{i-1}^{m} \left(n_i \left(\sum_{k=1}^{i-1} \left(\sum_{j=1}^{n_k} (s_{kj} + x_{kj}) + r_k \right) \right) \right)$$

Let $P_k = \sum_{j=1}^{n_k} (s_{kj} + x_{kj}) + r_k$, the processing time for tape i, and
$C_i = \sum_{j=1}^{n_i} (n_i - j + 1) s_{ij} + \sum_{j=1}^{n_i - 1} (n_i - j) x_{ij}$, the total waiting time incurred by the requests on medium i after medium i has been loaded onto the drive.
Therefore,

$$W = \sum_{i=1}^{m} C_i + \sum_{i=1}^{m} Tin_i + \sum_{i=1}^{m} \left(n_i \sum_{k=1}^{i-1} P_k \right)$$

i.e. the total waiting time for all requests consists of three components. The first component is the waiting time incurred by requests for a given medium once the medium has been loaded onto the drive, which is constant independent of the order in which the media are loaded. The second component is the waiting time for requests of a medium due to the media switches that precede the request. This time is the same for all requests on a given medium and is dependent only on the ordinal position of the medium in the scheduling order. The third component is the waiting time for requests due to the processing of the requests on the preceding media. It is constant for each request for a given medium and is equal to the sum of the processing times for the preceding tapes. Therefore, the total waiting time for all requests can be expressed as:

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

3.2 Optimal Solution

We will now develop an algorithm that determines the optimal schedule for a single drive. For this purpose we need to establish the following lemmas.

Proposition 1 *For any given schedule, switching any two media in the order, affects the waiting times for requests on the media that are switched and all media that lie between them only. The requests for media that precede or follow both switched media will have the same waiting times.*

The above property is a direct consequence of the impact of the three components in the waiting time when the media are reordered.

Lemma 3.1 *In any schedule, if two adjacent media, i and $i+1$ are switched, the change in the total waiting time for the schedule is given by:*

$$\Delta W = n_i T + n_i P_{i+1} - n_{i+1} T - n_{i+1} P_i$$

Proof: By Proposition 1 only the requests on media i and $i+1$ will change as a result of the switch. Each request on medium i sees an increase in the waiting time due to the switch. The increase is an extra medium switch and the processing time of medium $i+1$. Therefore the increase in waiting time is: $n_i T + n_i P_{i+1}$. Similarly, each request on medium $i+1$ sees a decrease in the waiting time. The decrease is equal to one medium switch and the processing time for medium i (since medium i no longer precedes it in the new order). The amount of the decrease for medium $i+1$ is: $n_{i+1} T + n_{i+1} P_i$. Hence the total change in the total waiting time is:

$$n_i T + n_i P_{i+1} - n_{i+1} T - n_{i+1} P_i.$$

□

Lemma 3.2 *In any optimal schedule, $O_1, O_2, \cdots, O_m$, the following condition must hold:*

$$\forall i \ \ 1 \leq i < m, \ \ \frac{n_i}{T+P_i} \geq \frac{n_{i+1}}{T+P_{i+1}} \tag{2}$$

Proof: Let us assume that for an optimal solution, there exists k, $1 \leq k < m$, such that the above condition is not true, i.e.,

$$n_k(T+P_{k+1}) < n_{k+1}(T+P_k)$$

Consider a schedule which differs from $O_1, O_2, \cdots, O_m$ only in that the media O_k and O_{k+1} are switched. Therefore by Lemma 3.1 the total waiting time for such a schedule would differ from that of the optimal schedule by

$$\Delta W = n_k(T+P_{k+1}) - n_{k+1}(T+P_k)$$

which is negative. Hence we see that the new schedule has a lower total waiting time than the optimal schedule $O_1, O_2, \cdots, O_m$. Therefore this schedule could not be optimal, leading to a contradiction. Hence our assumption that there exists k which satisfies
$n_k(T+P_{k+1}) < n_{k+1}(T+P_k)$ must be false. Therefore $\forall k \ \ 1 \leq k < m, \ \ n_k(T+P_{k+1}) \geq n_{k+1}(T+P_k)$
Rearranging, we get $\forall k, 1 \leq k < m, \ \frac{n_k}{T+P_k} \geq \frac{n_{k+1}}{T+P_{k+1}}$. □

We now develop a sufficient condition for an optimal solution.

Theorem 3.1 *Any schedule of m tapes which satisfies Equation 2 is optimal.*

Proof: Suppose that there is a schedule,$A_1, A_2, \cdots, A_m$ which satisfies Equation 2, but is not optimal. Let $O_1, O_2, \cdots, O_m$ be an optimal schedule. If both schedules are the same, then both are optimal, otherwise, let i be the smallest index for which the two schedules differ. That is $\forall i, \ \ 1 \leq j < i, A_j = O_j$, and $A_i \neq O_i$. Since both schedules satisfy Equation 2, $O_i \geq O_k, \ \forall k, \ \ i < k \leq m$, and $A_i \geq A_k, \ \forall k, \ \ i < k \leq m$. In other words, A_i must have the largest ratio of number of requests to processing time plus switch time for all media that follow A_i, and O_i must also have the largest ratio for all media that follow O_i. This can only be true if either $A_i = O_i$ or the ratio for A_i is equal to the ratio for O_i. Thus we observe that the only difference in the two sequences can be the order in which media that have the same ratio are ordered. Next we show that rearranging the order of media that have the same ratio has no effect on the overall waiting time.

Consider a sequence of media, $C_l, C_{l+1}, \cdots, C_n$, all of which have the same ratio of number of requests to processing time plus switch time. If we switch any two of these media, say, i and j (C_i precedes C_j in the above sequence), then the change in the total waiting time is:

$$\Delta W \quad = \sum_{k=i}^{j} (change\ in\ waiting\ time\ for\ medium\ k)$$

$$
\begin{aligned}
\Delta W &= \Delta W_j + \Delta W_{i+1} + \cdots + +\Delta W_{j-1} + \Delta W_i \\
\Delta W_j &= n_j(-P_i - T - P_{i+1} - T \cdots - P_{j-1} - T) \\
\Delta W_{i+1} &= n_{i+1}(P_j + T - P_i - T) = n_{i+1}(P_j - P - i) \\
&\vdots \\
\Delta W_i &= n_i(P_{i+1} + T + \cdots + P_j + T)
\end{aligned}
$$

Therefore,

$$\Delta W = n_i(P_{i+1} + T) - n_{i+1}(P_i + T) + \cdots + n_i(P_j + T) - n_j(P_i + T)$$

Since all these media have the same ratio, it is clear that the change $\Delta W = 0$. Hence we can interchange any pair of media in the sequence without affecting the total waiting time. In other words, any permutation of media with the same ratio has the same waiting time.

Therefore, since the two sequences $A_1, A_2, \cdots, A_m$ and $O_1, O_2, \cdots, O_m$ differ only in the sequence of media with the same ratios, they must have the same total waiting times. In other words the sequence $A_1, A_2, \cdots, A_m$ is also optimal. □

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 processing time plus 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. For example for serpentine tapes, the scheduling policies developed by Hillyer and Silberschatz [HS96a] can be used for scheduling the requests on each tape. For the 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

In the previous section, we studied the problem of I/O scheduling for a single drive. We were able to derive an efficient algorithm that determines the optimal schedule. 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, the formula developed in the earlier section gives the total waiting times. The total waiting time can be computed as the sum of the waiting times for each drive. The times for each drive are given by

Equation 1. The total waiting time for all media processed on drive l is:

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

where

- m^l is the number of media that are processed on drive l
- C_i^l is the constant component of the wait time for items on the ith medium on drive l
- n_i^l is the number of items for the ith medium on drive l
- Tin_i^l is the wait time for items of the ith medium on drive l due to the switching of preceding media in drive l
- P_j^l is the processing time for the ith medium on drive l
- $n_i^l \sum_{j=1}^{i-1} P_j^l$ is the wait time for items on the ith medium on drive l due to the processing of preceding media on drive l.

Therefore, assuming d drives, the total waiting time for all media is given by,

$$W = \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(C_i^k + Tin_i^k + n_i^k \sum_{j=1}^{i-1} P_j^k \right) \tag{3}$$

This can be rewritten as:

$$\begin{aligned}
W &= \sum_{k=1}^{d} \sum_{i=1}^{m^k} C_i^k + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k T i + n_i^k \sum_{j=1}^{i-1} P_j^k \right) \\
&= \sum_{k=1}^{d} \sum_{i=1}^{m^k} C_i^k + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (T + P_j^k) + T n_i^k \right) \\
&= \sum_{k=1}^{d} \left(\sum_{i=1}^{m^k} C_i^k + \sum_{i=1}^{m^k} (T n_i^k) \right) + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (P_i^k + T) \right) \qquad (4)
\end{aligned}$$

The first term in Equation 4 is the sum of the constant times for all media ($\sum_{k=1}^{d} \sum_{i=1}^{m^k} C_i^k$) and the product of the switch time, T and the sum of the number of requests for all media ($T \sum_{k=1}^{d} \sum_{i=1}^{m^k} n_i^k$). Both these factors are independent of the order of processing of media. Therefore, the total waiting time can be written as:

$$W = K + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (P_i^k + T) \right) \tag{5}$$

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

We now show that the problem of finding a scheduling policy which minimizes Equation 5 is difficult by showing that the problem is the same as a known NP-complete problem. Let us first 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 for all items is given by

$$W = K' + \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(n_i^k \sum_{j=1}^{i-1} (P_i^k + T) \right) \tag{6}$$

where $K' = \sum_{k=1}^{d} \sum_{i=1}^{m^k} \left(C_i^k + T n_i^k + \sum_{j=1}^{n_i^k} x_{ij}^k \right)$ is also a constant independent of the schedule.

Since NP-complete problems are usually posed as a Yes or No answer, we restate the above problem as:

Definition 4.1 Multidrive Full Medium Scheduling (MFMS) GIVEN: *A set S of media, $|S| = m$, number d of drives, a medium switching time, T, number of requests, n_i and processing time, P_i for each medium $i \in S$, and a positive integer J.*
QUESTION: *Is there a d-drive schedule for the media such that the value of W defined by Equation 6 is no more than J?*

Theorem 4.1 *The problem of multidrive full medium scheduling is NP-complete.*

Proof: We will prove the above theorem by showing that a known NP-complete problem can be reduced to the above problem ($MFMS$). We will reduce the problem of *Scheduling to Minimize Weighted Completion Time (MWCT)* [LKB77], which is defined as follows, to $MFMS$.

Definition 4.2 Scheduling to Minimize Weighted Completion Time (MWCT) GIVEN: *Set S of tasks, number $m \in Z^+$ of processors, for each task $t \in S$ a length $l(t) \in Z^+$ and a weight $w(t) \in Z^+$, and a positive integer J'.*
QUESTION: *Is there an m-processor schedule σ for S such that the sum, over all $t \in S$, of $(\sigma(t) + l(t)) \cdot w(t)$ is no more than J'?*

The $MWCT$ problem has been shown to be NP-complete in [LKB77]. Given any instance of the $MWCT$ problem, we create an instance of the $MFMS$ problem as follows. For each task $t \in S$ of $MWCT$, we create a medium in the $MFMS$ problem such that the number of requests for the medium is equal to the weight of t and the processing time for the medium is equal to the length of the task t. The number of drives for the $MFMS$ problem is set equal to the number of processors in the $MWCT$ problem. The value of the constant J for the $MFMS$ problem is set to be equal to the constant for the $MWCT$ problem, J' plus K', where K' is defined above as the constant factor in the total waiting time in Equation 6. Now we see that any schedule that satisfies the $MFMS$

problem will generate a schedule that results in a value of W which is smaller than or equal to J. If from this schedule we derive a schedule for the $MWCT$ problem by simply substituting the corresponding tasks and processors for the media and drives, we will get a schedule which has a total weighted completion time of no more than $J - K'$, which is equal to J'. Similarly, any schedule for the $MWCT$ problem which results in a weighted completion time no more than J' can be easily converted into a schedule for $MFMS$ which has a total waiting time of no more than $J' + K'$, which is equal to J. Hence, a solution for $MWCT$ exists if and only if a solution for $MFMS$ exists. Clearly, $MFMS$ is in NP because we can guess a schedule and determine it's total completion time and compare it with J in polynomial time. Since $MFMS$ is in NP and an NP-complete problem ($MWCT$) can be reduced to $MFMS$, we conclude $MFMS$ is an NP-complete problem. $\square$.

5 Performance Simulation and Heuristics

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* [Sch86], 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 need to gain exclusive access to one of the robot arms in order to load or eject media. The drives operate independently of each other, except for the contention for robot arms.

Experiments were conducted with both single drive 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 possible request locations were accessed 20% of the time and the remaining 20% of the locations 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 2. These values are based on the Exabyte EXB-480 Library configured with Exabyte EXB Mammoth drives [EXA96]. Most of the values have been chosen to

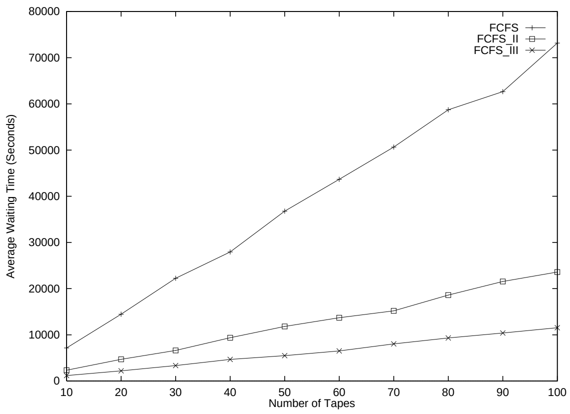
Parameter	*Meaning*
NDRIVES	Number of Drives
NROBOTS	Number of Robotic arms
NUM_TAPES	Number of media in the library
NUM_REQ	Number of requests for the experiment
PICK_TIME	Time required by robot to remove medium from drive
PUT_TIME	Time required by robot to place medium on drive
MOVE_TIME	Time required by robot to move between media on shelf
SEEK_OVHD (REW_OVHD)	Constant overhead for seek(rewind) (sec)
SEEK_RATE (REW_RATE)	Rate at which the drive can seek(rewind) (MB/s)
XFER_SPEED	Transfer rate of the drive (MB/s)
EJECT_TIME	Time required by the drive to eject a medium (sec)
LOAD_TIME	Time required by the drive to load a medium (sec)

Table 1: Table of Model Parameters

closely match the published figures for the products modeled. For some experiments, however, the value of one parameter was altered. The number of robot arms for most libraries is usually 1 or 2 whereas we have set the number of arms to 4 for some of the experiments with multiple drives. The reason for this is that if the number of robots is less than the number of drives, then contention between the drives for the robot arms affects the performance. To evaluate the performance of the heuristics for multiple drives relative to the optimal solution independent of contention for the robots we have set the number of robots to 4. The experiments were repeated with a single robot to determine the performance of these heuristics under contention.

Parameter	*Meaning*
NDRIVES	1 or 4
NROBOTS	1 or 4
NUM_TAPES	10 - 100
NUM_REQ	100 - 1000
PICK_TIME	10 sec
PUT_TIME	10 sec
MOVE_TIME	2 sec
SEEK_OVHD (REW_OVHD)	0.1 (0.1) sec
SEEK_RATE (REW_RATE)	188 (193) MB/s
XFER_SPEED	3 MB/s
EJECT_TIME	8 sec
LOAD_TIME	10 sec

Table 2: Parameter values used for the experiments



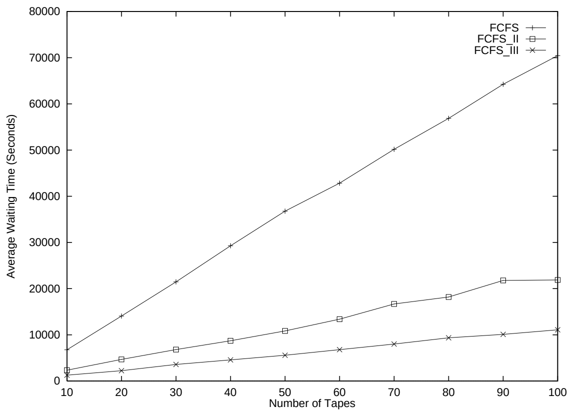


Figure 4: Variations of FCFS for Single drive with (a) Random and (b) Hot-Cold request distributions

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 or seconds of each other due to higher processor and network speeds. Hence, even though the requests do not arrive at the "same" time, for the purposes of our evaluation they can be considered to have arrived at the same time. 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 (such as in the order of ascending tape blocks) depending upon the best algorithm for the media type (tape or disk etc.). The results for the experiments are shown in Figures 4 and 5. Figure 4 shows the results for single drive with both input patterns and Figure 5 shows the results for multiple drives with both input patterns. It can be observed that the full medium algorithms reduce the average waiting time for requests significantly. Reordering the media results in even further reductions in the average waiting time. This confirms our hypothesis that full medium scheduling is beneficial even though it may not be optimal as seen in Example 2.1.

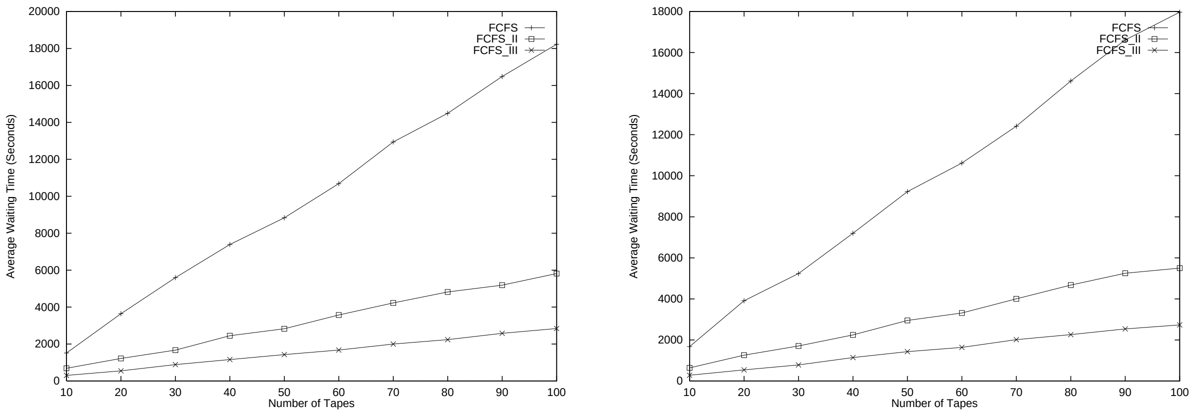


Figure 5: Variations of FCFS for Multiple drives with (a) Random and (b) Hot-Cold request distributions

5.3 Single Drive Experiments

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. Figure 6 shows some of the results comparing the performance of the heuristic with the optimal algorithm. The graphs in Figure 6 also show the results for the FCFS_III heuristic described earlier. We note that the hueristic performs very well and is within 1% of the optimal. It should also be noted that OPT and *Number* perform better than FCFS_III.

5.4 Multiple Drive Experiments

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 multidrive case. The analogous algorithm for the minimum weighted completion time problem ($MWCT$)[LKB77] has been shown to have an upper bound. In particular the algorithm is guaranteed to produce a solution that is within

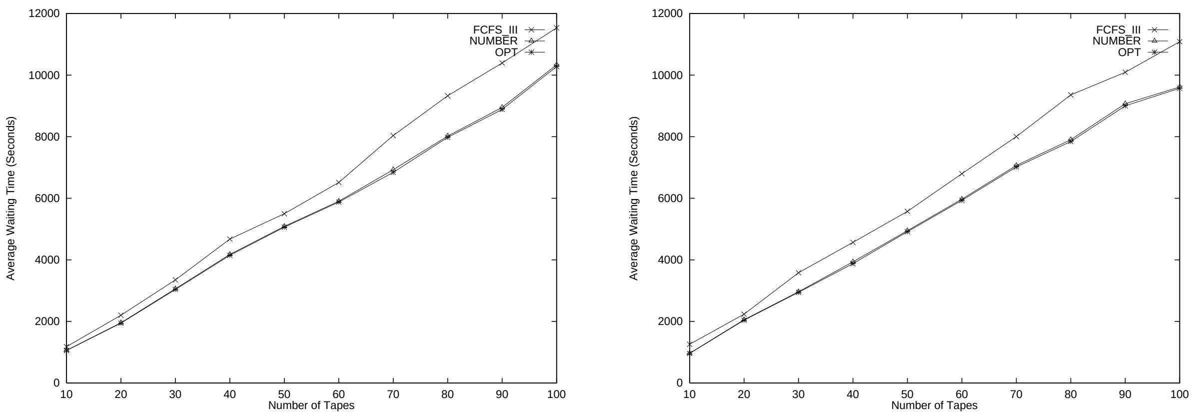


Figure 6: Comparison of the Number heuristic with the Optimal for single drives with (a) Random and (b) Hot Cold request distributions

$(1+\sqrt{2})/2$ (approximately within 20%) of the optimal for $MWCT$ [Pin95]. This bound does not necessarily hold for the $MFMS$ problem due to the other constant factors in the cost. Due to its good performance guarantee for the $MWCT$ problem, we evaluated this algorithm for the $MFMS$ problem. This algorithm requires knowledge of the processing time for media which may not always be known. Hence 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. Once the set of media that are to be processed on a drive is determined, the optimal schedule for the drive can be determined using the optimal algorithm, OPT, for single drives. Therefore, for each allocation of media to drives, we can determine the optimal schedule. The optimal schedule for the multiple drives is taken to be the schedule with the least average waiting time. Since this algorithm has 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 are shown in Figure 7. We note that the *Number* heuristic generates solutions that are very close to those produced by OPT.

In order to study the performance of the algorithms when the number of robot arms is small, we repeated the experiments with multiple drives using only a single robot arm. Figure 8 shows the results for the FCFS-III, OPT and *Number* heuristics with a single robot. It can be seen that even in the presence of contention, the OPT and *Number* heuristics perform well.

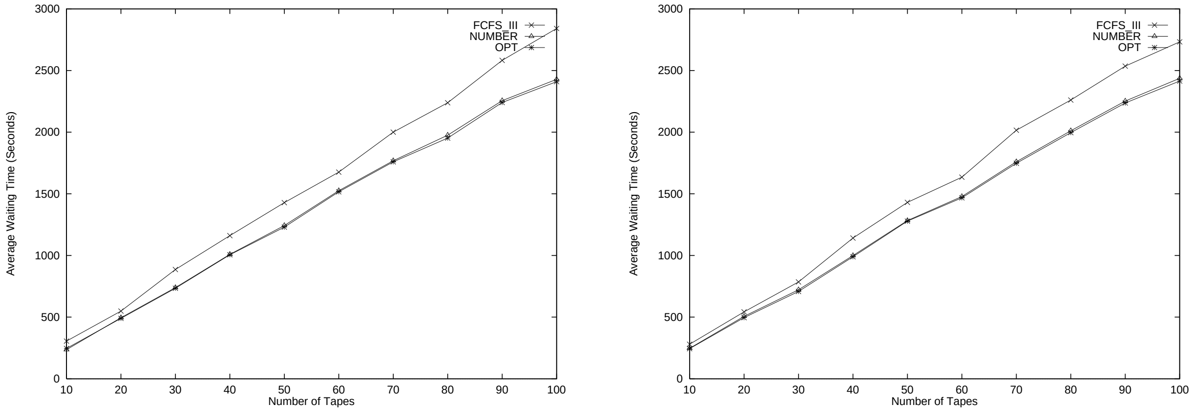


Figure 7: Comparison of the Number heuristic with OPT for multiple drives with (a) Random and (b) Hot-Cold request distributions

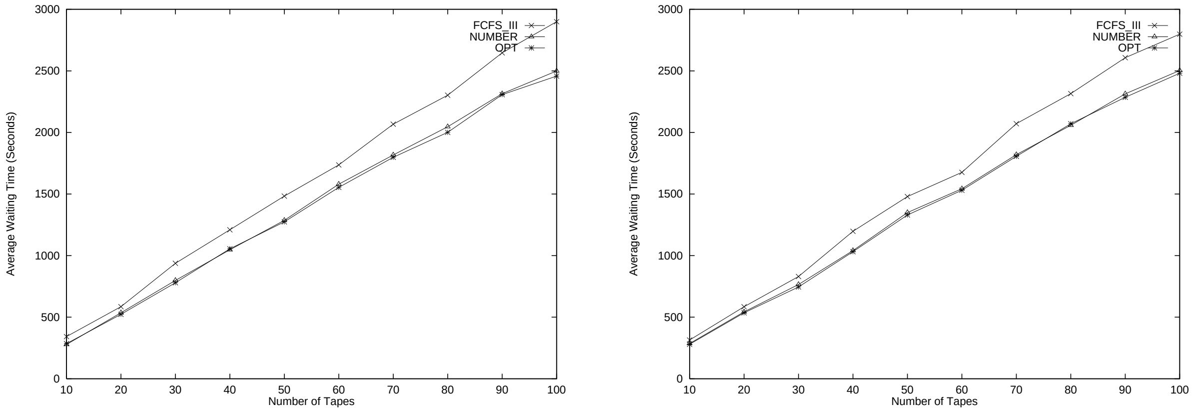


Figure 8: Comparison of the Number heuristic with OPT for multiple drives and single robot with (a) Random and (b) Hot-Cold request distributions

6 Conclusion

We have investigated the problem of I/O scheduling for tertiary libraries. The focus of the study has been on scheduling policies that do not switch media before completing all requests on the loaded medium, (*Full Medium* scheduling policies). For technologies such as optical disks which have switch times that dominate seek times, such schedules are likely to be optimal. For technologies that have a more dominant seek times, such as tapes, it may be possible to get more efficient schedules by unloading media to service requests on other media before reloading the previous medium, as seen in Example 2.1. However, even for such media, the *Full Medium* scheduling policies show great improvement over more naive scheduling policies. We expect that the more complicated scheduling policies needed to achieve optimal scheduling will not achieve significant gains over the simpler *Full Medium* scheduling policies.

In the case of a single drive, we have developed a scheduling policy, OPT, that produces optimal schedules in $O(m \log m + N)$ time, where m is the number of media and N is the number of requests to all the media. OPT does not depend upon the order of processing of requests within a single medium. We can therefore chose the best available scheduling policy within a given medium independently. It does however, require knowledge of the total processing time required for each medium and the time that the library takes to switch media. In the absence of such information, an even simpler heuristic, *Number* has been proposed which also takes $O(m \log m + N)$ time and produces schedules that are very close to the optimal (within 1% for all our experiments).

In the prescence of multiple drives, the problem of *Full Medium* scheduling becomes NP-Complete. Since no efficient solutions are expected to exist for NP-complete problems, we proposed two heuristics. The OPT algorithm which gives optimal performance for the single drive problem, was found to be a very good heuristic for the multiple drive problem. It gave results within 1% of the optimal for multiple drives. As in the single drive case, the even simpler *Number* heuristic was found to give schedules that are very close to those given by OPT for multiple drives. It is important to note that both heuristics perform well even in the presence of contention for the robot arms.

We have therefore developed optimal and near optimal solutions for the scheduling of I/O requests for robotic libraries. The algorithms are efficient and simple to implement. 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 and optical disks.

References

[BD78] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal of Computing*, 7(4):393–404, November 1978.

[CDK+95] L. T. Chen, R. Drach, M. Keating, S. Louise, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. In *Information Systems*, pages 155–83. Elsevier Science, 1995.

[CHL93] M. J. Carey, L. M. Haas, and M. Livny. Tapes hold data, too: Challenges of tuples on tertiary store. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 413–7, Washington, DC, 1993.

[DK93] A. L. Drapeau and R. H. Katz. Striping in large tape libraries. In *Proc. of Supercomputing*, pages 378–387, Portland, Oregon, 1993. ACM.

[EXA96] EXABYTE. Products. http://www.Exabyte.COM:80/Products/, Oct. 1996.

[FM96] D. A. Ford and J. Myllymaki. A log-structured organization for tertiary storage. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 20–7, New Orleans, Louisiana, 1996.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability - A guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, 1979.

[GM95] L. Golubchik and R. Muntz. Analysis of striping techniques in robotic storage libraries. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, pages 225–38, Monterey, CA, 1995.

[HS96a] B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape. In *SIGMETRICS*, pages 170–9, Canada, 1996.

[HS96b] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertary storage. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Canada, 1996.

[KBCH95] B. Kobler, J. Berbert, P. Caulk, and P. C. Hariharan. Architechture and design of storage and data management for the NASA Earth Observing System Data and Information System (EOSDIS). In *Proceedings of the 14th IEEE Symposium on Mass Storage Systems*, pages 65–76, Los Alamitos, CA, USA, Sept 1995. IEEE Computer Soc. Press.

[LKB77] J.K. Lenstra, A. K. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem*. Wiley, Chichester, 1985.

[LO96] J. Li and C. Orji. I/O scheduling in tape-based tertiary systems. *Journal of Mathematical Modelling and Scientific Computing*, 6, 1996.

[ML95] 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.

[ML96] J. Myllymaki and M. Livny. Efficient buffering for concurrent disk and tape I/O. In *Proceedings of Performance '96, Int. Conf. on Performance Theory, Measurement and Evaluation of Computer Communication Systems*, Lausanne, Switzerland, October 1996.

[MP93] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems with applications to Production Systems and Project Management.* John Wiley and Sons, Inc., 1993.

[Par95] G. R. Parker. *Deterministic Scheduling Theory.* Chapman & Hall, 2-6 Boundary Row, London SE1 8HN, UK, 1995.

[Pin95] M. Pinedo. *Scheduling Theory, Algorithms and Systems.* Prentice-Hall International Series in Industrial and Systems Engineering. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Qua96] Quantum. Products and technology. http://www.quantum.com/products, Oct. 1996.

[Sar95a] S. Sarawagi. Database systems for efficient access to tertiary memory. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, pages 120–6, Monterey, California, 1995. IEEE Comput. Soc. Press.

[Sar95b] 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.

[Sch86] H. D. Schwetman. CSIM: A C-based, process-oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, pages 387–396, December 1986.

[SON96] SONY. Recording media products. http://sony.sosin.com.sg/record-media/, Oct. 1996.

[SS93] S. Sarawagi and M. Stonebraker. Single query opimization for tertiary memory. Technical Report s2k-94-45, Computer Science Div. U.C. Berkeley, December 1993.

[SS94] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *IEEE Int. Conf. on Data Engineering*, pages 328–336, Houston, TX, USA, Feb. 1994. IEEE Comput. Soc. Press.

[Teo72] T. J. Teorey. Properties of disk scheduling policies in multiprogrammed computer systems. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 1–11, 1972.