

Improving the Access Time Performance of Serpentine Tape Drives

Olav Sandst  and Roger Midtstraum
Department of Computer and Information Science
Norwegian University of Science and Technology
{olavsa, roger}@idi.ntnu.no

Abstract

This paper presents a general model for estimating access times of serpentine tape drives. The model is used to schedule I/O requests in order to minimize the total access time. We propose a new scheduling algorithm, Multi-Pass Scan Star (MPScan), which makes good utilization of the streaming capability of the tape drive and avoids the pitfalls of naive multi-pass scan algorithms and greedy algorithms like Shortest Locate Time First. The performance of several scheduling algorithms have been simulated for problem sizes up to 2048 concurrent I/O requests. For scheduling of two to 1000 I/O requests, MPScan* gives equal or better results than any other algorithm, and provides up to 85 percent reduction of the total access time. All results have been validated by extensive experiments on Tandberg MLR1 and Quantum DLT2000 drives.*

1. Introduction

A few decades ago when the blue dinosaurs still ruled the computer rooms, magnetic tape was actively used in data processing. Input (program and data) as well as the results from the processing were stored on tape. The use of the tape was highly optimized as most of the processing was done as batch jobs which read and wrote the tape sequentially. Since then, magnetic disks have taken over as the premier storage medium for program and data. The main advantage of disks over tape is the shorter random access latency, a few milliseconds rather than tens of seconds. The main advantages of magnetic tape are *cost* and *storage density*. Today, a high-end disk drive stores about 10–20 GB. A magnetic tape stores the same amount of data at a fraction of the price and storage volume. For emerging applications, like scientific databases, digital image databases and video archives, which require vast amounts of data storage, these two factors can be of sufficient importance to make magnetic tape a possible choice.

When applications have a non-sequential access pattern

to data stored on tapes, it is of foremost importance to minimize the random access delay and thereby maximizing the utilization of the tape drives. When more than one data item is requested on a single tape, this can be achieved by careful *scheduling* of the concurrent I/O requests against the tape drive. In a hypothetical image database storing high-quality images (10 MB each), a single 13 GB QIC tape would be able to store 1300 different images. A query for images of “Bill Clinton” could for instance find ten such images on a single tape. To read these ten images, which are scattered around the tape, would normally take 8 minutes and 47 seconds in the average case and 22 minutes and 7 seconds in the worst case. Using the best scheduling algorithm, the average access time can be reduced to 3 minutes and 45 seconds. As this simple example shows, the *random access performance* of tape drives can be significantly improved by means of proper I/O scheduling.

In this paper we study the problem of scheduling random I/O requests for serpentine tape drives, using the Tandberg MLR1 and the Quantum DLT2000 tape drives as examples. While a lot of work has been done on modelling and scheduling of random accesses on magnetic disks, little research have been performed for serpentine tape. A very notable exception is the work by Hillyer and Silberschatz [4, 5], which presents a detailed model of seek times for the Quantum DLT4000 tape drive and evaluates several algorithms for scheduling of random I/O requests. The strength of their work lies in the very accurate model of seek times, but unfortunately, the time necessary to characterize each tape is so long (twelve hours of processing), that it significantly reduces the practical value of their work. Prabhakar et al. [7] have studied the problem of scheduling I/O requests for robotic libraries with tape drives, but made no efforts to schedule the processing of requests for a given tape. Sarawagi [11] has studied query processing in tertiary memory databases. She proposed to use average seek cost to model the performance of serpentine tape drives and hence missed opportunities to optimize the random I/O performance. Taking an alternative approach to improve the performance of tape drives, Christodoulakis et al. [1]

have studied optimal placement of data in tertiary storage libraries.

Scheduling of I/O requests requires an access time model for the storage device. To reduce the time-consuming characterization process incurred by the previous efforts, we deliberately designed a less detailed access time model for serpentine tape drives and provide low-cost algorithms to characterize each individual tape. This model has been used to schedule I/O requests with a number of known algorithms. From these experiences, we came up with a new scheduling algorithm, Multi-Pass Scan Star (MPScan*), which make efficient utilization of the streaming capability of serpentine tape drives. Using the access time model, the performance of MPScan* and several other scheduling algorithms have been simulated, and the results have been validated by measurements on tape drives. The main conclusion is that our low-cost model gives access time estimates that are good enough to facilitate efficient scheduling of random I/O requests, while still having a resource consumption well within practical limitations. Further, for all reasonable problem sizes, MPScan* performs better than the other algorithms, giving up to 85 percent reduction of the access time.

The remainder of this paper is organized as follows. Section 2 introduces serpentine tape technology, Section 3 describes our model of access times, Section 4 describes scheduling algorithms, Section 5 presents results from simulations of the scheduling algorithms, Section 6 presents measurement from executions on tape drives, and, finally, Section 7 gives the conclusions and outlines some remaining problems.

2. Magnetic tape technologies

There are three important tape-technologies: helical-scan tape, parallel tape and serpentine tape. *Helical-scan* tape drives read/write vertical or diagonal tracks on the tape using a rotating read/write head. *Parallel* tape drives read/write all tracks in parallel during one scan through the tape. *Serpentine* tape drives first read/write a track in forward direction, then read/write the next track in reverse direction, and so on, leading to a *serpentine pattern* for the data layout.

In this paper we focus on the *serpentine* tape model. There are two important technologies for serpentine tape drives, QIC and DLT. The QIC – Quarter Inch Cassette – standard [8] uses tape which is a quarter inch wide, have cartridges with both wheels inside the cassette, and provides standard tape formats covering the storage range from 60 MB to 25 GB. DLT – Digital Linear Tape [6] – uses a half inch tape which is stored in a cartridge with only one reel, the second reel is part of the tape drive. When inserting a DLT tape into a drive, the tape first has to be mounted onto this reel. The DLT family of drives supports storage capac-

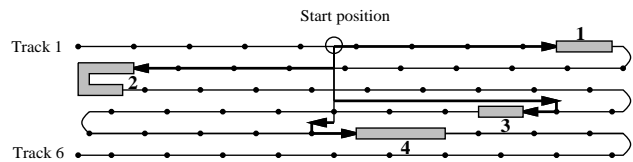


Figure 1. A simplified model of a serpentine tape with the key points marked on the tracks. From the current start position, possible seek patterns are shown for four data requests.

ities from 2.6 GB to 35 GB per tape.

While DLT and QIC drives are slightly different, their access time characteristics are similar and to a high degree dictated by the serpentine data layout. Contrary to parallel and helical-scan drives, serpentine drives do not provide a direct relationship between logical address and physical position on the tape, making it much harder to estimate the access times.

In the experiments reported in this paper, we have used the Tandberg MLR1 [12] and the Quantum DLT2000 [3] tape drives. The Tandberg MLR1 is based on the 13 GB QIC standard [8], making it possible to store 13 GB per tape (without compression). The drive can deliver (read/write) a maximum sustained data rate of 1.5 MB/s to/from the host computer. Each tape has 72 logical tracks, 36 in the forward direction and 36 in the reverse direction. The DLT2000 drive stores 10 GB per tape (without compression), and has a maximum transfer rate of 1.25 MB/s. All experiments are performed using two Tandberg MLR1 tape drives and one DLT2000 drive connected to a Fast SCSI-2 bus on a Sparc-Station 20 workstation.

3. Estimating access times

An I/O request to a tape consists of the address of the first logical block, and the number of consecutive blocks to be read. The *access time* for an I/O request is the time it takes from the tape drive gets the request, until the data is made available to the requesting entity. When a tape drive receives multiple I/O requests, the total access time often can be reduced by the use of a scheduler, which reorganizes the retrieval order of the tape requests. In order to do intelligent reorganizing, the scheduler must be able to compute quite accurate estimates of the access times.

For a tape drive, the access time is made up of the time used to re-position the tape drive (*seek time*) and the time used to read the requested data (*transfer time*). Regarding the transfer time, there is nothing that can be done to reduce this, as the drive reads with a constant transfer rate until the end of the requested data is reached. The transfer time then depends on the size of the requested data and on the number of necessary track changes. This is straightforward to

model, and the relevant model parameters can easily be determined for each type of tape drive [10].

On the other hand, the seek times are essentially wasted time and should be reduced as much as possible. Because of the complex data layout on a serpentine tape, it is a non-trivial task to provide a model which makes it possible to compute precise estimates for the seek times. The rest of this section describes the task of providing such a model, starting with a brief analysis of the relevant characteristics of serpentine tape.

3.1. Characteristics of serpentine tapes

Figure 1 shows a principal sketch of a serpentine tape. The tape is organized into a fixed number of tracks which run in alternating directions, producing a serpentine pattern. From a users point of view, however, the tape is seen as a linear sequence of logical data blocks, numbered from zero and up to the number of blocks on the tape. A data item is stored in a number of logical data blocks, written to subsequent positions along one or more tracks.

When a data item is requested, the tape drive first has to re-position the tape from the current position to the position of the first logical block of the data item, and then the requested block can be read in one uninterrupted read operation. The re-positioning of the tape normally consists of two parts. The head has to be positioned on the correct data track (latitudinal position, see case 2, 3 and 4 in Figure 1), and the tape has to be wound to the correct longitudinal position (all cases in Figure 1). In order to determine the longitudinal position of each individual block, the tape drive uses a set of predetermined positions, or *key points*, which are evenly spaced on each track. When the tape drive changes track and/or reading direction, it has to seek to the nearest preceding key point before it can position itself on the requested data block (see case 2, 3 and 4 in Figure 1).

3.2. Model of seek times

The purpose of a seek time model is to be able to estimate the seek time between two *logical* block addresses on a tape. Two important properties of the serpentine tape have to be included in such a model. First, as the seek time is largely determined by the difference in longitudinal position on the tape, the model must be able to estimate the *physical* position for each logical block address on the tape. Second, the model has to include information about how to compute the seek time between two given *physical* positions.

As most tape drives do not provide any information about the physical position of the logical data blocks, this information has to be estimated. Our experiments show that the amount of data that can be stored on one single tape varies from tape to tape. For a single tape, the number of data blocks also varies from track to track. As a consequence of

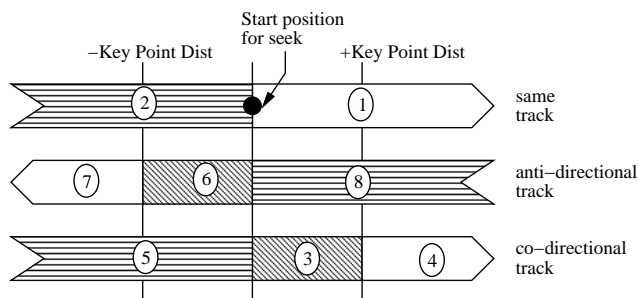


Figure 2. Model used for partition seeks into disjunct seek classes. The *Key Point Distance* is the longitudinal distance between two adjacent key points on a track.

this, the mapping from logical to physical position is bound to be rather complex and has to be instrumented on a per tape basis. Hillyer and Silberschatz [4] have done this by locating each of the key points on the entire tape. This gives a very accurate model. Unfortunately, the computational cost is about twelve hours to locate all the key points on a single tape.

In order to make a model which is less costly to establish, but still accurate enough to facilitate scheduling of I/O requests, we have developed a model which does not rely on the location of each of the key points. The model we propose is based on the two following strategies:

- To estimate the physical position of each logical block on the tape, we locate the logical address of the first block on each track. These track addresses can then be used to find the correct track number and to compute an estimate for the longitudinal position of any given logical block.
- To estimate the seek time between any two physical tape positions, we partition the seek space into eight disjunct seek classes, with regard to the work that is incurred on the tape drive. For each seek class, we provide analytic cost functions to compute the seek time.

The start addresses of each track has to be determined once for each tape, because these addresses vary from tape to tape. As shown in [10], this can be done at virtually zero cost by logging and analyzing the write times for the blocks on the tape, using the *Write-Turn* algorithm. If the writing of the tape can not be monitored, the start addresses can be determined quite quickly by reading selected parts of the tape and analyzing the read times, using the *Read-Turn* algorithm. As an example, use of the Read-Turn algorithm involves only 13 minutes of analysis per 13 GB QIC tape.

The seek time between two positions on a tape is determined by four factors:

- The longitudinal distance between the two positions.
- The need of a track change.
- The need to change winding direction.
- Whether locating the closest key point influences on the seek, or not.

Based on these factors, the seek space can be partitioned into the eight seek classes shown in Figure 2. From a given position, we can have seeks to positions further down the same track, which incur no track change, no change of winding direction and is not influenced by a key point (the area marked 1 in the figure). The second seek class contains the seeks to preceding positions on the same track, which requires two changes of winding direction, location of a key point, but no change of track (area marked 2 in the figure). The third and fourth seek classes are seeks to positions further down a co-directional track, which require a track change, has no change of winding direction, and may be influenced by a key point (area marked 3) or not (area marked 4). The remaining four seek classes can be described in a similar way.

For each seek class, we have determined analytic cost functions which have to be instrumented once for each type of tape drive. The instrumentation of the cost functions is done by statistical analysis of measured seek times for a large number of actual seeks on a tape drive. A more elaborate description of every aspect of the access time model can be found in [10].

4. Scheduling algorithms

When a tape cartridge is mounted into a tape drive, several users may have issued any number of I/O requests for data on this single tape. The problem of scheduling such random access I/O requests for a serpentine tape drive can be stated as follows:

Given a list of I/O requests and an initial tape position, the goal is to produce a possibly reorganized list, containing the same requests, which will result in a minimum total access time when the requests are executed in this new order.

In this section, we first describe a number of known scheduling algorithms and then propose a novel algorithm, Multi-Pass Scan Star (MPScan*), which makes clever utilization of the streaming capability of the tape drives. To clarify the discussion, we have classified the algorithms based on the degree to which they take the physical geometry of the tape into consideration.

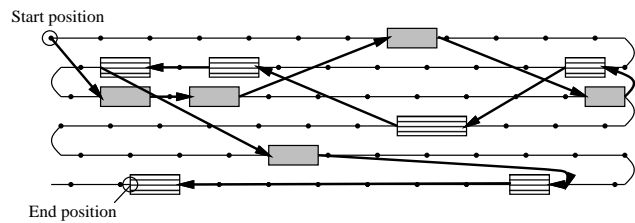


Figure 3. A MPScan schedule resulting in two full scans of the tape.

4.1. Zero-dimensional algorithms

These algorithms perform the “scheduling” of the I/O requests without any consideration of the physical properties of the tape.

READ - reads the tape sequentially until all requests are served. On a Tandberg MLR1 tape drive it takes about two and a half hours to read an entire tape.

FIFO - reads the requests in the order in which they are found in the initial schedule, without any attempts to optimize the execution order. This is the obvious method to schedule I/O requests for storage devices in cases where one does not have any knowledge of the properties of the device.

SORT - re-organizes the requests such that they are executed in order of increasing *logical* addresses.

4.2. One-dimensional algorithms

In this class we find the algorithms that take into account the physical positions (longitudinal dimension) of the requests on the tape, but neglect to take into consideration that requests to close positions, but on different tracks, might incur relatively high seek times.

SCAN - All requests are executed in a single scan back and forth the entire tape – this algorithm is similar to the well known *elevator* algorithm used for disks. The requests are partitioned into two sets based on the read direction of the corresponding track. The requests in each set are sorted on *physical* tape position. If we assume that the tape drive is positioned at the beginning of the tape, SCAN reads all the requests on forward tracks as it *scans* through the tape, and then all the requests on reverse tracks as it scans back to the start of the tape. The complexity of this algorithm is $O(n \log n)$.

4.3. Two-dimensional algorithms

The algorithms in this class take into account both the physical tape distance between two tape blocks (longitudinal dimension) and the cost of changing between tracks (latitudinal dimension).

OPT - this is the *optimal* scheduler which always (if the model is exact) gives the shortest possible total execution

```

Schedule = MPScan(Request_list) // produce initial
// MPScan schedule
N = number of scans in Schedule
MinCost = cost_of(Schedule)
Final_Schedule = Schedule

while ( N > 1 ) {
  remove last scan of requests from Schedule.

  while (some request r in last scan) {
    compute insertion cost for all possible positions to
    insert r into the Schedule.
    insert r where the insertion cost is least.
    remove r from last scan.
  }

  if (cost_of(Schedule) < MinCost) {
    MinCost = cost_of(Schedule)
    Final_Schedule = Schedule // best schedule
  } // so far

  N = N - 1;
}

```

Figure 4. Pseudo code for the MPScan* algorithm. The cost_of function estimates the execution cost of the schedule on a tape drive.

time. The problem with this algorithm is that it reduces the scheduling problem to the familiar *Traveling Salesman problem*, which is known to have an exponential time complexity for computing the optimal solution. Because of this, OPT is hardly useful for schedules containing more than ten requests.

LOSS - is an heuristic for solving the *Asymmetric Traveling Salesman Problem* [2]. It solves the same problem as the OPT scheduler, but in linear time. The reason for including this scheduling algorithm, is to compare our scheduling strategies with the work by Hillyer and Silberschatz [5].

SLTF - Shortest Locate Time First [5] - Starting on the start position of the drive, it first selects the request with the least seek cost (locate time) as the next tape operation to be performed. It then selects the remaining request with the least seek cost from the new current position. This step is repeated until all requests are scheduled. The cost of this algorithm is $O(n^2)$.

MPScan - Multi-Pass Scan - as the SCAN algorithm, this is an elevator algorithm, but this algorithm allows multiple passes of the tape. The main problem with the SCAN algorithm is that it does not take into account the cost of track changes. As a result, the tape drive might have to rewind to find the closest key point, when the next request in the schedule is physically close to the current tape position. MPScan avoids such interruptions of the streaming by careful selection of the next request to be included in the schedule. At each step in the production of the schedule, the algorithm considers only the requests which are further down the current track and the requests on co-directional tracks which are more than the key point distance further down these tracks. These are the requests in tape regions 1 and 4 as shown in Figure 2, relative to the current position. The physically

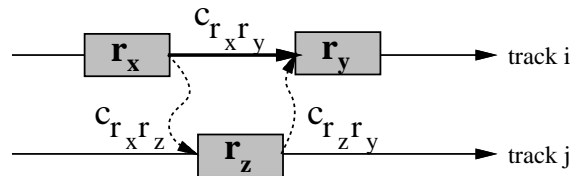


Figure 5. The insertion cost for inserting request r_z between request r_x and request r_y is computed as $C_{r_x r_z} + C_{r_z r_y} - C_{r_x r_y}$.

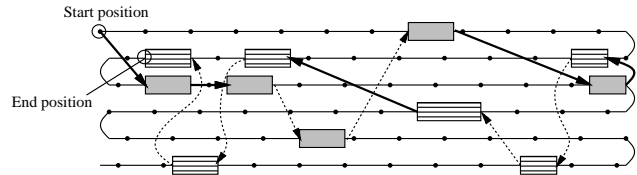


Figure 6. The MPScan schedule in Figure 3 reduced to one scan by using the MPScan* algorithm. New seeks introduced in the schedule are marked with dotted arrows.

closest of these requests is chosen as the next request to be included into the schedule. When no more requests are fulfilling the requirements for being included in a scan, a scan in the opposite direction is started. This strategy guarantees that the tape drive does not have to rewind when seeking to the next request in the schedule, at the cost of possibly having to make multiple passes through the tape. The complexity of the MPScan algorithm is $O(n^2)$.

An example, which shows how MPScan schedules 11 requests in two full scans of the tape, is given in Figure 3.

MPScan* - Multi-Pass Scan Star is an improved algorithm based on the MPScan algorithm. In schedules produced by MPScan, the number of requests per scan tends to decrease in the last scans of the schedule. As a result, the seeks in the last part of the schedule will be rather long. The MPScan* algorithm avoids these long seeks at the end of a MPScan schedule, by reducing the number of scans in the schedule and inserting the affected requests into the remaining scans.

The pseudo code for MPScan* is given in Figure 4. The algorithm starts by creating an initial schedule, using the MPScan algorithm. It then proceeds by reducing the number of scans that the tape drive has to perform, by repeatedly removing the last scan from the schedule. For each of the requests in the removed scan, an *insertion cost* is computed for all positions where the request can be inserted into the remaining schedule. The *insertion cost* is the extra seek time which the drive will use if a request, r_z , is inserted into the schedule between two other requests, r_x and r_y . Figure 5 illustrates how the *insertion cost* is computed. Each of the

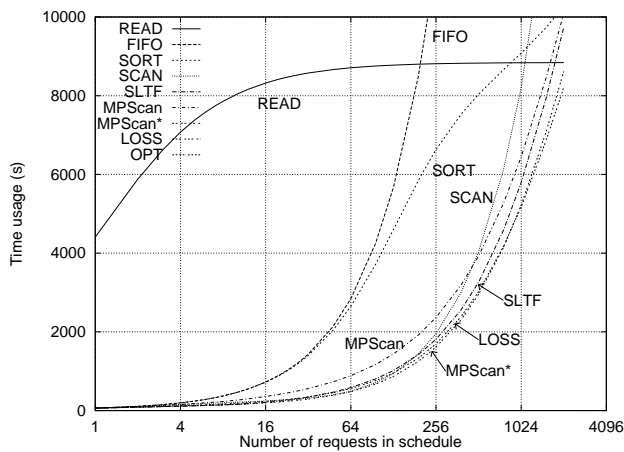


Figure 7. Total time for performing a request list for different request list sizes.

requests is inserted into the remaining schedule at the position with the least insertion cost. This process is repeated by removing the next last scan from the schedule, until the schedule consists of only one scan. For each scan removed, the total seek time cost of the new schedule is computed. Finally, the schedule with the least total seek time cost is chosen as the result of the algorithm. The worst case complexity of the MPScan* algorithm is $O(n^3)$.

Figure 6 shows the result of applying MPScan* to the same I/O requests as used for MPScan in Figure 3. As shown, the number of full scans of the tape is reduced from two to one by inserting the requests of the last scan into the first scan, possibly leading to a shorter total execution time.

5. Simulations

The algorithms presented in the previous section have been implemented. This section presents results from simulation of the algorithms. The reason for simulating the algorithms is to be able to compare the properties of the different algorithms using the tape model presented earlier in this paper. In the next section the simulation results will be validated by comparing them to results from execution of schedules on real tape drives.

To instrument the access time model, we have used the Tandberg MLR1 drive as the target drive. All simulations were performed on a set of request lists containing from one to 2048 requests. Each request was for one 32 KB block on the tape. The block addresses were drawn from a uniform distribution in the interval $[0..MaxBlocksOnTape)$. For schedules of length from 1 to 192 requests, we used 100,000 different request lists, for longer schedules we gradually reduced the number of list from 25,000 request lists for 256 requests per schedule, to 500 request lists for schedules of 2048 requests. For the OPT algorithm, the largest request

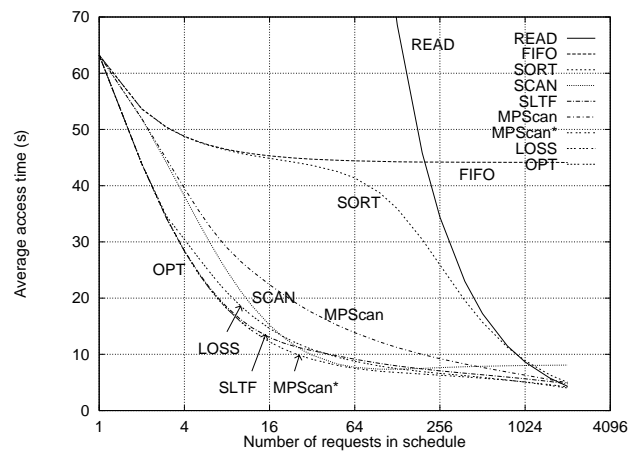


Figure 8. Average access time for each I/O request using different scheduling algorithms and for different problem sizes.

lists contained 12 requests, and was run only 100 times due to the high CPU usage. All simulations started with the tape drive positioned at the beginning of the tape.

Figure 7 shows estimated execution times for the different schedule lengths using the different scheduling algorithms. The corresponding average access times per request are presented in Figure 8. As can be seen from Figure 8, with only one request there is nothing that can be done to improve the performance, and in average the access time for one tape request will be 63 seconds. If we do no scheduling of the requests (i.e., using the FIFO strategy) the average seek time stabilizes on 45 seconds. This can be greatly reduced by using one of the better scheduling algorithms. For schedules with less than 12 requests, the curves for OPT, SLTF and MPScan* overlap and any of them can be used. For longer schedules, it is not feasible to use the OPT algorithm. As long as the length of the schedule is less than 1000 requests, MPScan* produces the best schedules. For schedule lengths from 1000 requests to 2100 requests, LOSS is marginally better than MPScan*. For even longer schedules, the READ strategy gives the shortest total execution time for performing the schedule.

The simulations were run on a SparcStation 20 workstation with a 125 MHz HyperSparc processor. Each run of the algorithms was timed, and the average times used to compute the different schedules are presented in Figure 9. If we disregard the OPT scheduler, LOSS and MPScan* are the only schedulers which use more than five seconds for computing the long schedules. Still, the higher CPU time usage of MPScan* and LOSS is much less than the reduction in total execution time achieved by use of these schedulers. Unfortunately, due to this higher CPU time usage, the initial latency for retrieving the first data object from the tape will be longer. This can be a problem for some applications. For

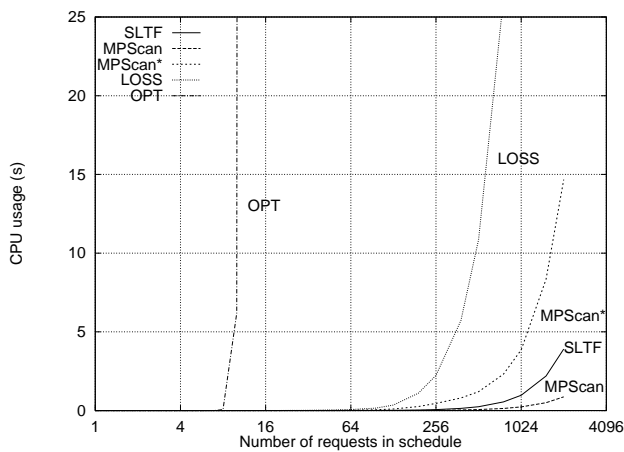


Figure 9. CPU usage for scheduling of request lists of different sizes. The cost for scheduling of 12 request using OPT is 912 seconds. To schedule 1024, 1536 and 2048 request using LOSS takes 63, 119 and 227 seconds.

MPScan*, this problem is readily solved by starting the execution of the first few requests in the first scan, as soon as the initial MPScan step of the algorithm has finished.

6. Experiments and discussion

To validate the results from the simulations, we have run schedules of varying problem sizes produced by the different schedulers on a Tandberg MLR1 tape drive. As for the simulations, the tape drive was positioned at the beginning of the tape before the first tape operation was started, and each request was for one 32 KB block from a random position on the tape. For each schedule, the total execution time was measured.

The measured access times per request, for the different scheduling algorithms, are presented in Figure 10. By comparison of these results with the corresponding simulated access times given in Figure 8, one can see that the relative performance of the scheduling algorithms are the same in the simulations and the real experiments. The experiments confirm MPScan* as the best algorithm for problems with less than about 1000 requests, and LOSS as the best algorithm for problem sizes between 1000 and 2000 requests.

From these figures, one can also see that the estimated and experienced access times are approximately equal. To get a better comprehension of the accuracy of the estimated execution times, Figure 11 shows the difference in percent between estimated and measured execution time for schedules produced by MPScan* and run on the MLR1 drive. A positive value in the figure indicates that the estimated execution time is a number of percent higher than the actual measured execution time. As the figure shows, most of

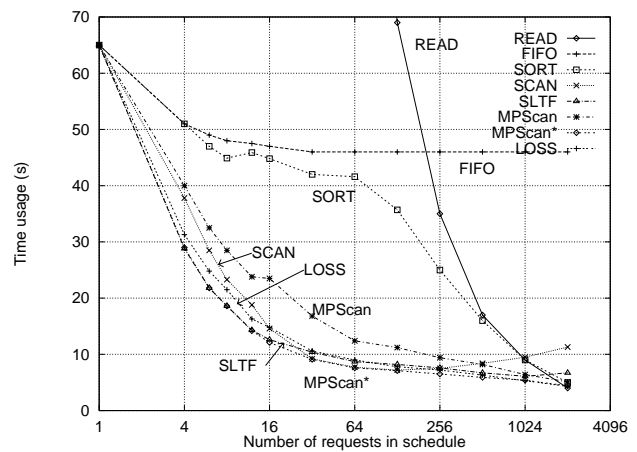


Figure 10. Average access time in seconds for each I/O request using different scheduling algorithms and for different problem sizes.

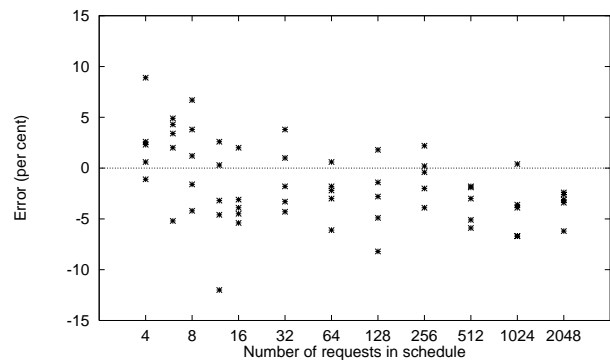


Figure 11. Percent error in estimated execution times for schedules produced by MPScan*. For each problem size, the results from performing five schedules on a Tandberg MLR1 are included.

the estimated execution times are within +/-5 percent of the measured execution time. As this is achieved without determining the important key points, we consider this to be a good result. Determining every key point, Hillyer and Silberschatz [5] experienced differences within +/-1 percent for the Quantum DLT4000 drive, but at a preventive high cost.

6.1. Access time improvements

The purpose of random I/O scheduling is to reduce the total execution time for a given combination of I/O requests. This will minimize the waiting time for the requesting application(s) and maximize the utilization of the (expensive) tape drives, which frequently are a bottleneck in tape systems. Figure 12 shows the relative and absolute improvements that the best scheduling algorithm, MPScan*, gives

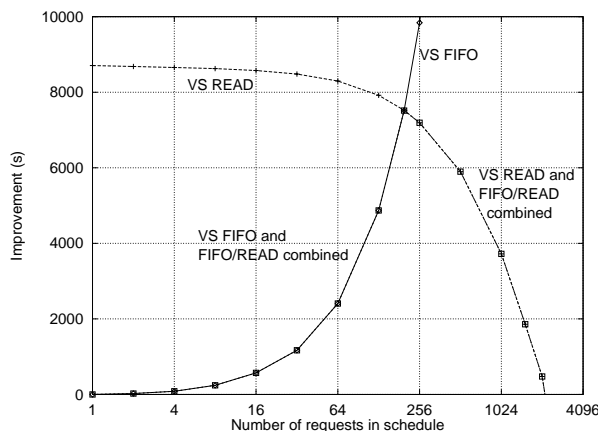
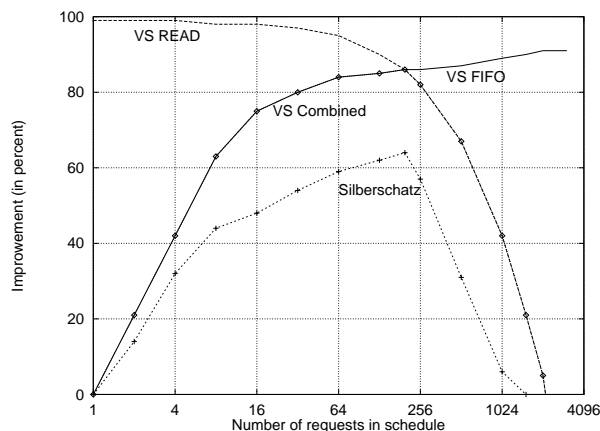


Figure 12. (a) Reduction in percent of total execution time using Tandberg MLR1 for MPScan* compared to FIFO scheduling, compared to READ and compared to an optimal combination of FIFO and READ. Results from Hillyer and Silberschatz [5] show their best results compared to an optimal combination of FIFO and READ. (b) Reduction in seconds of total execution time for MPScan* compared to FIFO scheduling, compared to READ and compared to an optimal combination of FIFO and READ.

compared to the non-scheduling approaches, which are random order (FIFO) execution of the requests, reading the entire tape (READ), or a favorable combination of FIFO and READ. As shown in the figure, MPScan* scheduling gives substantial benefits for all problem sizes ranging from two I/O requests and up to 2100 I/O requests. The maximum gain, compared to an optimal combination of FIFO and READ, is for a schedule of 196 requests, which is the point where FIFO and READ have the same performance. At this point, a MPScan* schedule executes in 20 minutes and 47 seconds, compared to an execution time of 2 hours and 27 minutes for the corresponding FIFO/READ schedule, saving more than two hours and seven minutes.

Comparing the results for MPScan* to the results for the other scheduling algorithms, we find that LOSS is the only scheduler that give better results and only for schedules containing more than 1000 requests. For 1024 requests, the average improvement by use of LOSS is less than one percent. For 2048 request, the improvement increases to five percent. Unfortunately, the computational cost of LOSS is rather high for large schedules (see Figure 9), leading to a high initial latency. Unlike MPScan*, there is no easy way to start execution of any requests until the entire schedule has been computed.

To verify the generality of our approach, we have used MPScan* to schedule random I/O requests on a Quantum DLT2000 tape drive. As shown in Table 1, the results are similar, but not quite as good as the results for the MLR1 tape drive. The best result is for 150 I/O requests, where 53 percent reduction of total execution time is achieved. As the DLT2000 has fewer key points on each track, the key point

No. of requests	MLR1	DLT2000	DLT4000
1	63.2	66.8	95
2	43.9	51.7	72
4	29.0	40.2	54
16	12.1	29.0	38
64	7.6	24.8	31
256	6.5	21.7	24
1024	5.4	11.0	13

Table 1. Average access time in seconds for the Tandberg MLR1 and DLT 2000 with the MP-Scan* algorithm, and DLT4000 with the LOSS algorithm. The access times for DLT4000 are found in [5].

distance is longer and the cost of many short seeks is going to be higher. The longer key point distance also decreases the accuracy of the access time model, introducing more errors into the schedules.

It is interesting to compare our results for the Tandberg MLR1 to the results that Hillyer and Silberschatz [5] achieved for the Quantum DLT4000 tape drive. In Figure 12 (a) we have shown their best results compared to an optimal combination of FIFO and READ for the Quantum drive. For any number of requests, we get significantly better results with the Tandberg drive. On average, we reduce the total execution time with 23 percent more than they do for the DLT4000 drive.

The average access times are much higher for the DLT4000 drive than for the MLR1 drive. Table 1 shows the access times for the DLT4000 drive with their best algo-

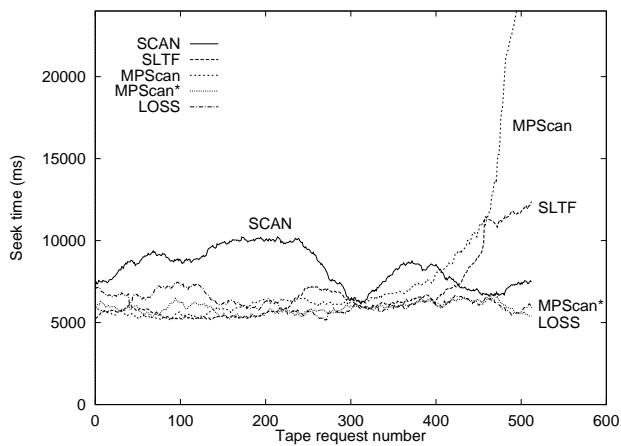


Figure 13. Measured seek times for each request in a schedule of 512 requests. The schedules are produced using SCAN, SLTF, MPScan, MPScan* and LOSS. The seek times have been run through a smoothing filter of width 60.

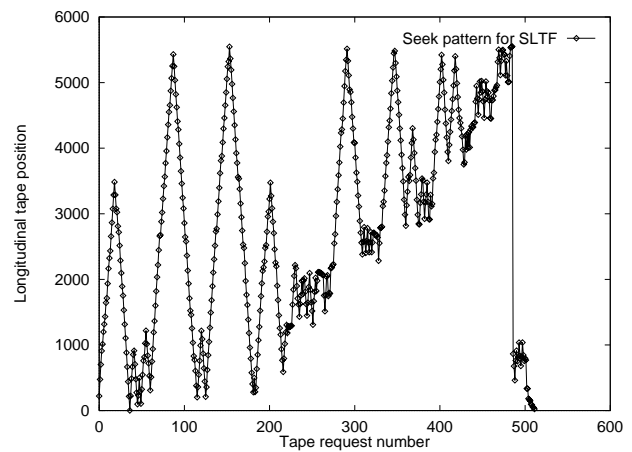


Figure 14. Physical seek pattern on the tape for a schedule of 512 requests produced by SLTF. The graph shows the longitudinal tape position for as the drive executes the requests in the schedule.

rithm, LOSS, and the access times for the MLR1 drive with the MPScan* algorithm. A schedule of 196 I/O requests would for instance have a total execution time of 1 hour and 24 minutes on the DLT4000 drive with the LOSS algorithm, compared to the less than 23 minutes on the MLR1 with MPScan*.

The Quantum DLT4000 drive and the Tandberg MLR1 drive are comparable, except that the DLT stores 54 percent more data (20 GB versus 13 GB) and has a maximum seek time that is 43 percent longer (180 seconds versus 126 seconds). Even if we take the longer maximum seek time into full effect (which is more than would be fair), the Tandberg drive still performs a schedule of 196 requests in less than half the time needed by the DLT4000. The main reason why the MLR1 performs so much better, is that the Tandberg drive has many more *key points* (25 versus 13 on each track). The higher number of key points leads to a much shorter key point distance, which significantly reduces the cost of many of the shorter tape movements. Another factor is that the DLT seeks with 30 percent less speed when searching for a position between two key points, while the MLR1 performs all operations at full speed. These results should probably be taken into consideration when new generations of serpentine tape drives are designed.

6.2. Analysis of tape behavior

This section presents observations from studying the behavior of the scheduling algorithms in practical experiments with the Tandberg MLR1 drive. These results will help the understanding of some of the results presented earlier in this paper.

In Figure 13, we show the measured seek time for each request in one schedule containing 512 tape requests for some of the schedulers. From this figure, it is worth noting that only the MPScan* and LOSS schedulers have approximately constant average seek times for all the requests in the schedule. MPScan and SLTF have about the same average seek times for the first part of the schedule, but for the last part of the schedule the seek times increase significantly. For MPScan the reason is that it is too *lazy*, only including requests which will not interrupt the streaming of the tape drive in the schedule. This leads to few requests in the last scans of the schedule, and to too many passes of the tape. For SLTF the reason is that it is too *greedy* and only considers one step at a time. The effect of the greedy behavior of SLTF can also be seen in Figure 14, where we have plotted the longitudinal seek pattern on the tape for the same schedule as in Figure 13. Note that although the SLTF algorithm does not impose any particular seek pattern on the tape, it still produces a seek pattern where the tape drive is mostly streaming for the first part of the schedule. As the number of non-scheduled requests gets lower, the algorithm starts to produce more and more costly seeks (see Figure 13). This is not the case for MPScan* where the long, costly seeks have been removed from the schedule. As can be seen in Figure 15, MPScan* manages to keep the streaming pattern for the entire schedule. The figure also shows “clusters of requests” along the tape scans. These are the positions in the schedule where the requests of the last scans of the initial MPScan schedule have been inserted.

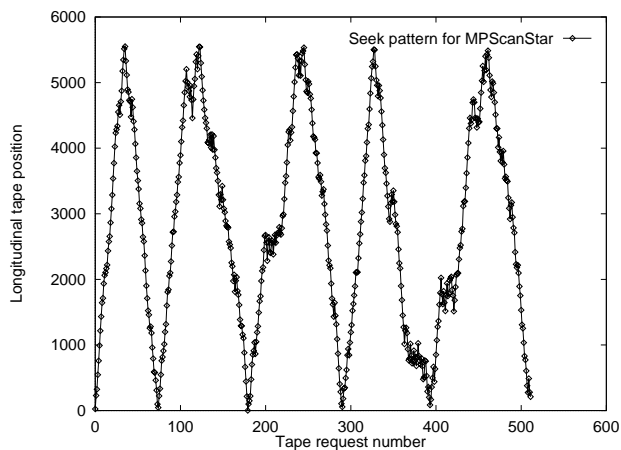


Figure 15. Physical seek pattern on the tape for a schedule of 512 requests produced by MPScan*. In this particular schedule, all requests are read in five scans through the tape.

7. Conclusion

In this paper we have described a general model to estimate access times for serpentine tape drives, which balances the need of accuracy with the time needed to characterize each individual tape. When used for the Tandberg MLR1 tape drive, the accuracy of the estimates are within +/- 5 percent for most cases, and in the worst case, the model requires no more than thirteen minutes of analysis to characterize each tape.

The access time model has been used to schedule random I/O requests against tape drives. Simulations and executions on tape drives demonstrated results that are fully comparable to results that have been achieved with far more elaborate approaches. Further, we have proposed a novel scheduling algorithm, MPScan*. Compared with other scheduling algorithms, MPScan* gives shorter access times for all reasonable problem sizes and produces access patterns that are favorable to keep the drives in "streaming" operation. Use of the MPScan* algorithm have clearly demonstrated the usefulness of I/O scheduling, as up to 85 percent savings in execution time have been experienced, compared to no scheduling.

The applicability of our approach has been demonstrated by practical experiments on the Tandberg MLR1 drive which follows the QIC standard and on the Quantum DLT2000 tape drive which follows the DLT standard. An interesting side effect of our work is the difference in random I/O performance that has been experienced for the Tandberg and Quantum drives. From the specifications, the Quantum DLT4000 and Tandberg MLR1 tape drives seem to have very similar performance characteristics. For random ac-

cess, however, the Tandberg drive has demonstrated superior performance.

The work presented has been used to schedule requests for multimedia data stored on tape [9]. Three unsolved problems are the dynamic scheduling of requests which arrive during the execution of a schedule, the possible mismatch between the scheduler's ordering of the requests and the user's priorities, and fairness of service in case of multiple users.

References

- [1] S. Christodoulakis, P. Triantafillou, and F. A. Zioga. Principles of optimally placing data in tertiary storage libraries. In *Proceedings of the 23rd VLDB Conference*, pages 236–245, Athens, Greece, August 1997.
- [2] P. V. D. Cruyssen and M. J. Rijckaert. Heuristic for the asymmetric travelling salesman problem. *The Journal of the Operational Research Society*, 29(7):697–701, 1978.
- [3] Digital Equipment Corporation. *Digital DLT2000/DLT2700 Cartridge Tape Subsystem Product Manual*.
- [4] B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 170–179, Philadelphia, Pennsylvania, May 1996.
- [5] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 195–204, Montreal, Canada, June 1996.
- [6] D. Lignos. Digital linear tape (DLT) Technology and product family overview. In *Proceedings of Fourth NASA Goddard Conference on Mass Storage Systems and Technologies*, Maryland, USA, March 1995.
- [7] S. Prabhakar, D. Agrawal, A. E. Abbadi, and A. Singh. Scheduling tertiary I/O in database applications. In *Proceedings of Eight International Workshop on Database and Expert Systems Applications*, pages 722–727, Toulouse, France, September 1997.
- [8] QIC Development Standard. *QIC-5010-DC, Serial Recorded Magnetic Tape Cartridge For Information Interchange, Revision E*. Quarter-Inch Cartridge Drive Standards, Inc., December 1994.
- [9] O. Sandst  and R. Midtstraum. Analysis of retrieval of multimedia data stored on magnetic tape. In *Proceedings from 1998 IEEE International Workshop on Multimedia Database Management Systems*, pages 54–63, Dayton, Ohio, August 1998.
- [10] O. Sandst  and R. Midtstraum. Low-cost access time model for a serpentine tape drive. *To be presented at the 16th IEEE Symposium on Mass Storage Systems*, March 1999.
- [11] S. Sarawagi. Query processing in tertiary memory databases. In *Proceedings of the 21st VLDB Conference*, pages 585–596, Zurich, Switzerland, September 1995.
- [12] Tandberg Data, Oslo, Norway. *Tandberg MLR1 Series Streaming Tape Cartridge Drives Reference Manual*, 1st edition, August 1996.