

Random I/O Performance of a Tandberg MLR1 Tape Drive

Roger Midtstraum and Olav Sandst ¹

Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract

This paper presents an access time model for the Tandberg MLR1 serpentine tape drive and provides low-cost algorithms to characterize this model for each individual tape. The access time model is used to study the effects of scheduling of I/O requests. Simulations and experiments show that use of proper scheduling algorithms provides substantial improvement of the random I/O performance.

1 Introduction

Compared to magnetic disks, magnetic tape technology stores the same amount of data at a fraction of the price and storage volume. For emerging applications, like data mining, digital image databases and video archives, which require really huge amounts of data storage, these advantages can outweigh the problems related to the much longer access times, which is the main disadvantage of magnetic tapes.

When applications have a non-sequential access pattern to data stored on tapes, it is of foremost importance to minimize the random access delay to the data stored on tape and thereby maximizing the utilization of the tape drives. This can be achieved by careful *scheduling* of the I/O requests against the tape drives. A hypothetical image database, which stores high-quality images (10 MB each) on magnetic tapes, can serve as an example. A single 13 GB QIC tape will be able to store 1300 images. A user query requesting images of “Thor Heyerdahl” could for instance find ten such images on a single tape. To read these ten images, which are scattered around the tape, without any attempt to optimize the access time would take 22 minutes and 7 seconds in the worst case and 8 minutes and 57 seconds in the average case. Using the best scheduling algorithm, this access time can be reduced to 3 minutes and 45 seconds, which is only 42% of the average case without optimizing. 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 tape drive as an example. While a lot of work have been done on modelling and scheduling of random accesses on magnetic disks [8], little research have been performed for serpentine tape. A very notable exception is the work by Hillyer and Silberschatz [2, 3] which presents a detailed model of seek times for the Quantum DLT 4000 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. [6] have studied the problem of scheduling I/O requests for robotic libraries with tape drives but make no efforts to schedule the processing of requests for a given tape. Sarawagi [12]

has studied query processing in tertiary memory databases. She proposes to use average seek cost to model the performance of serpentine tape drives and hence misses 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.

To overcome the limitations of the model suggested by Hillyer and Silberschatz [2], we have deliberately designed a less detailed model of the access times for the Tandberg MLR1 tape drive, and we provide low-cost algorithms to characterize each individual tape. Based on this access time model, we have simulated the performance of a number of known scheduling algorithms, and of a new algorithm, MPScan*, which gives superior results. All simulation results are validated by extensive experiments on real tape drives. The main conclusion is that our low-cost model of access times gives estimates that are good enough to facilitate efficient scheduling of random I/O requests, while still being simple enough to be of practical value.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to magnetic tape technology. Section 3 presents a model of access times for the Tandberg MLR1 serpentine tape drive, which is the basis for scheduling of random I/O requests. Section 4 describes several algorithms that perform static scheduling of random retrievals from serpentine tape drives, and Section 5 presents simulations that reveal the relative performance of the scheduling algorithms. Section 6 presents measurements of schedule executions on the Tandberg MLR1 which validate the simulation results, and discusses the quality of the access time model and of the scheduling algorithms. Section 7 concludes the paper.

2 Magnetic Tape Technologies

There are three main 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 standards for serpentine tape drives, QIC and DLT. QIC – Quarter Inch Cassette – started as a standard for inexpensive tape storage with modest capacity and bandwidth. During the last years the specifications have improved, and QIC is now comparable to DLT regarding both storage capacity and data transfer bandwidth. The QIC standard [7] 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[5] – is a standard originally developed by Digital Equipment Corporation. DLT 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.

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 block addresses and physical positions on the tape, making it much harder to estimate the access times.

To run the experiments reported in this paper we have used the Tandberg MLR1 tape

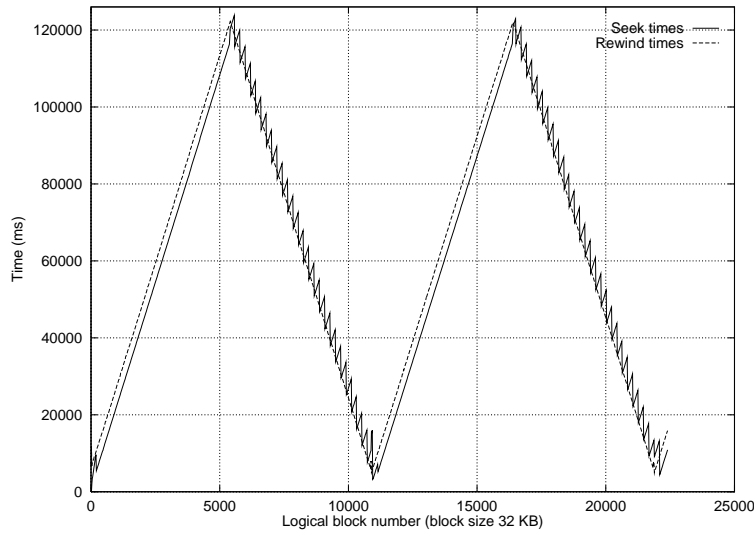


Figure 1 Seek and rewind times for the first tracks of a tape. The seek times are measured starting at the beginning of the tape and seeking to a given logical block number.

drive [13]. This tape drive uses serpentine data layout and is based on the 13 GB QIC standard [7], 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. All experiments are performed using two Tandberg MLR1 tape drives connected to a Fast SCSI-2 bus on a SparcStation 20 workstation.

3 Estimating Access Times

To reduce the access time and thus optimize the utilization of a tape drive by scheduling of the random I/O requests, one needs a fairly accurate model of the behavior of the tape drive. In this section we present a model for seek times on a serpentine tape drive. This model is used as a basis for the scheduling algorithms presented in the next section. A more thorough discussion of the model can be found in [11].

The *access time* is the time it takes from a memory device gets an I/O request until the data is made available to the requesting entity. For a tape drive, this is the time used to position the tape (*seek time*) plus the time used to read the data (*transfer time*). Considering a tape drive, there is nothing to be done with the transfer time, as the drive reads with a constant transfer rate until the end of the requested data is reached. Hence, the transfer time will be proportional to the size of the requested data. Contrary, the seek time is essentially wasted time and should be reduced as much as possible. Because of this, the focus of our access time model will be on modelling seek times. Modelling seek times for serpentine tape drives is non-trivial and provides opportunities for substantial optimization of the total access time.

An I/O request consists of the address of the first logical block requested and the number of consecutive blocks to be read. The purpose of the seek time model is to estimate the time that the tape drive would use to re-position the tape from the current logical position to the logical start position of the next I/O request. On a serpentine tape drive, this seek time is determined by the physical distance between the two positions on the tape and by whether the drive has to change track and/or direction, or not.

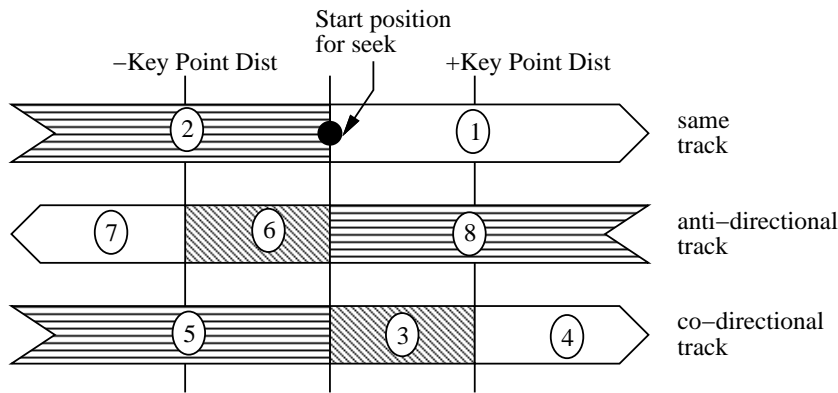


Figure 2 Model used for estimating seek times.

3.1 Characteristics of the MLR1 Tape Drive

To find the characteristics of the MLR1, the tapes were first written with 32 KB data blocks. By performing random seeks on a number of tapes, we found the maximum seek time for accessing a random block to be 126 seconds. The average seek time is 65 seconds for seeks starting from the beginning of the tape, and 45 seconds for seeks between any two random positions. In Figure 1 we have plotted the seek and rewind times for seeks from the start of the tape to blocks on the first four tracks of a typical tape. From the figure, we see that for forward tracks the curves for seek times are straight and increase proportionally with the block address of the sought block, while for reverse tracks the curve for seek times have a *sawtooth* pattern.

By studying the seek times for the reverse track, we have found that both the number of blocks within a sawtooth and the number of sawtooth along a track are varying. This is probably due to bad areas on the tape which the drive will skip. Other experiments have shown that the number of blocks on a track vary from track to track, resulting in a non-trivial mapping from logical block addresses to physical tape positions. The reason for the sawtooth pattern on the reverse track is that the read direction is opposite of the seek direction. When the tape drive tries to locate a position on the reverse track, it first has to seek past the sought position, then change winding direction and start reading in the opposite direction until it has found the sought block. Figure 1 indicates that the tape drive uses a set of predetermined points to decide where to stop the seek in forward direction and start seeking in the opposite direction. These points correspond to the first block in each sawtooth. Hillyer and Silberschatz [2] experienced similar sawtooth patterns for the DLT 4000 drive. They defined the points where the seek time has a big dip from one sawtooth to the next as the *key points* of the tape. This suggests that there will be key points along the forward tracks too, and by doing seek operations which start on a different position than at the beginning of the tape, we find sawtooths on the forward tracks too.

3.2 Model of Seek Times

Two important properties of the serpentine tape must be included in a model to make it able to estimate seek times between tape positions. First, it must be able to estimate the physical position for each logical block address on the tape. Second, it must be able to compute the seek time between every pair of physical positions. Hillyer and Silberschatz [2] have done this by locating the start address of each sawtooth on the entire tape. This gives a very accurate model. Unfortunately, it takes about twelve hours to locate all the sawtooths for

each tape, making their model too costly in most practical applications.

To make a model which is less costly to establish, while still accurate enough for most applications, we have proposed a model which do not rely on locating each of the sawtooths. Our model is based on the two following strategies:

- To be able to estimate the physical position of logical blocks on the tape, we locate the start address (the logical address of the first block) of each track on the tape. These track start addresses can then be used to find the correct track number and an estimate for the physical position, for any given logical block.
- To be able to estimate the seek times between two physical tape positions, we partition the possible seeks into eight disjunct cases. For each of these cases, we have established analytical cost functions.

There are four variables which influences the seek time for a given seek. These are 1) the physical distance between the two tape positions, 2) if the drive has to change track, 3) if the drive has to change winding direction, and 4) whether the sawtooths influences on the seek time. The eight different search cases are illustrated in Figure 2. These cases are based on how the different variables influences on a given seek. The cases are 1) seeks forward on the current track (no change of direction nor sawtooths), 4) long seeks on co-directional tracks where there is no change of direction nor sawtooths, 2) and 5) seeks where the drive has to change direction and there is always a sawtooth, 3) short seeks where there is a certain probability the seek time will be influenced by a sawtooth (i.e., in order to find the correct key point), 6) short seeks where the drive has to change winding direction and there is a certain probability that the seek time will be influenced by a sawtooth, 7) seeks where the drive has to change winding direction, and finally 8) seeks where the seek time will be influenced by locating a sawtooth.

For each of these cases of seeks, we have run extensive practical experiments with the tape to establish cost functions. These cost functions compute estimates for the seek time between two given positions. A more detailed description of the cost functions is given in [11].

The cost of establishing this model is low. The cost function for each of the eight regions in Figure 2 can be established once for each type of tape drive. To find the start addresses of the tracks has to be done once for each tape because these vary from tape to tape. As shown in [11], this can either be done by measuring and analysing the writing time for each block to the tape by using the *Write-Turn* algorithm, or by actively reading parts of the tape and analysing the read times by the *Read-Turn* algorithm. The cost of using the Write-Turn algorithm for finding the addresses of the start block on each track is virtually zero. To use the Read-Turn algorithm takes in average 13 minutes per tape.

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 random access I/O requests for a serpentine tape drive can be stated as follows:

Given a list R of I/O requests and an initial tape position I, the goal is to produce a possibly reorganized list S, containing the same requests as in R, which will result in a minimum total execution time when the requests are executed in the order dictated by S.

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.

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 R , 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 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 distance on the tape 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 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.

SLTF - Shortest Locate Time First - this is similar to the Shortest Seek Time First (SSTF) algorithm used for disk scheduling. Starting on position I , it selects the request from R with the least seek cost (locate time) as the first tape operation to be performed. It then selects from the remaining unscheduled requests, the request with the least seek cost from the new current tape position as the next tape operation to be performed. This step is repeated until all requests in R 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 through the tape. The main problem with the SCAN

```

Schedule = MPScan(R)
N = number of scans in Schedule
MinCost = cost_of(Schedule)
S = Schedule

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

    while (some request r in last scan) {
        go through all requests in Schedule and
        insert r where the added cost to the Schedule is least.
        remove r from last scan.
    }
    if (cost_of(Schedule) < MinCost) {
        MinCost = cost_of(Schedule)
        S = Schedule           // the best schedule so far
    }
    N = N - 1;
}

```

Figure 3 Pseudo code for the MPScan* algorithm.

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, and the closest of these candidates are chosen as the next request to be included into the schedule. This guarantees that the tape drive does not have to rewind to seek 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 $O(n^2)$.

MPScan* - Multi-Pass Scan Star is an improved algorithm based on the MPScan algorithm. SLTF and MPScan have the drawback that they are too greedy and select the next operation to be included in the final schedule only by minimizing the cost that this single operation will incur on the final schedule. SLTF do this by selecting the next operation to be the one with minimum seek time, while MPScan do it by selecting the physically closest operation which will not make necessary to change direction. As more requests are scheduled, less request will be candidates for being scheduled next and the seek times for each request will increase, leading to rather long seeks in the last part of the schedule.

The main idea of MPScan* is to avoid these long seeks at the end of an MPScan schedule by removing the corresponding requests from the schedule and inserting them somewhere else in the schedule. The place where to insert a request is selected by computing how much *extra cost* this request will introduce to the total schedule. The details of the algorithm are found in the pseudo code in Figure 3. It starts by performing an ordinary MPScan scheduling of the requests. It then removes the requests constituting the last scan from the schedule. For each of these requests, the added cost is computed for all possible positions where this request can be inserted in the schedule, and the request is inserted in the position where it does least harm. This process is repeated by removing and inserting the next last scan until the schedule consists of only one scan. For each repetition, the total cost of the new schedule is computed and finally the schedule with the best cost is chosen

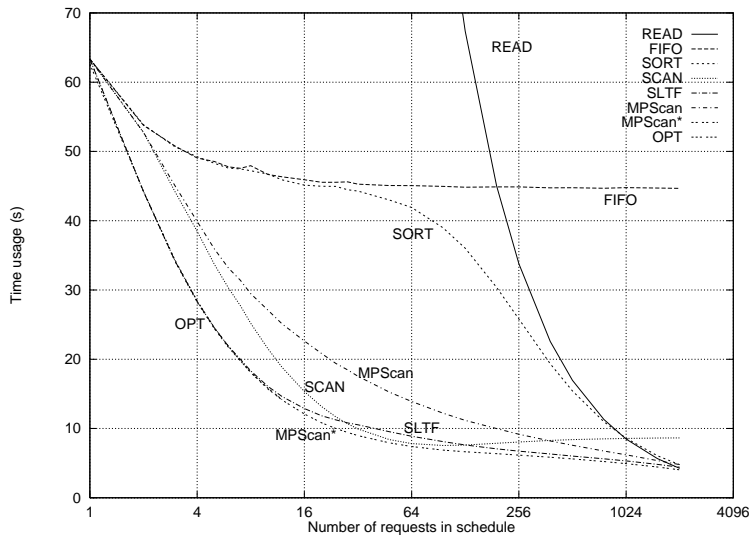


Figure 4 Simulated average access time in seconds for each I/O request using different scheduling algorithms and for different problem sizes.

as the result of the algorithm. The worst case complexity of this schedule is $O(n^3)$, but for most schedule sizes the cost of the algorithm is acceptable.

5 Simulations

The algorithms presented in the previous section have been implemented. This section presents the results from simulations 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 we will validate the simulation results by comparing them to the results from execution of schedules on a MLR1 streamer.

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 128 requests, we used 2000 different request lists, for schedules in the interval 192 to 768 we used 1000 request lists and for schedules with more than 768 we used 400 different request lists. For OPT the largest request lists contained 12 requests. All simulations started with the tape drive positioned at the beginning of the tape.

In Figure 4, we show estimated average access times per request for the different schedule lengths using different scheduling algorithms. As can be seen from the figure, with only one request there is nothing that can be done to improve the performance, and in average the seek time for one tape operation will be 65 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 about 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 since it costs too much to compute the schedule, and as long as the length of the schedule is less than about 2100 requests MPScan* produces the best schedules. For even longer schedules, the READ strategy gives the shortest total execution time for performing the schedule.

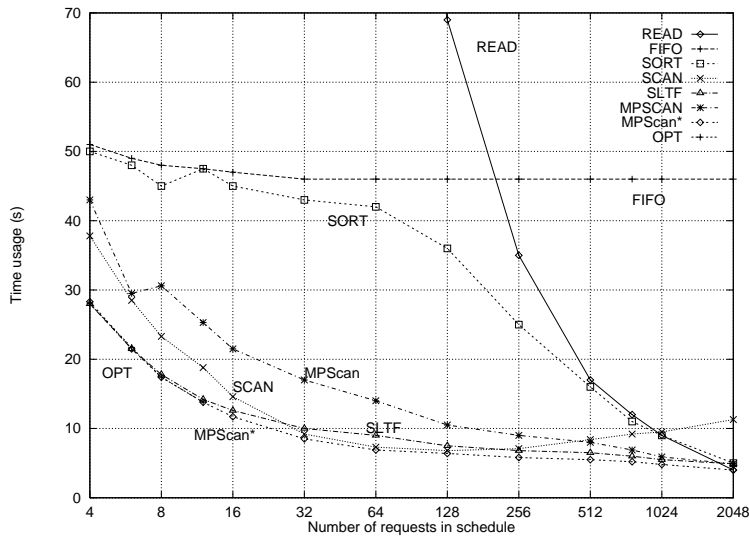


Figure 5 Measured average access time in seconds for each I/O request using different scheduling algorithms and for different problem sizes.

	4	8	16	32	64	128	256	512	1024	2048
SCAN	3.88	3.12	3.35	5.49	6.44	12.19	12.17	-1.54	-9.18	-23.39
SLTF	4.26	4.62	3.14	3.98	-0.65	-1.19	-7.09	-7.36	-5.88	-8.40
MPScan*	3.99	5.05	2.75	6.15	3.96	4.17	2.89	2.33	1.50	-0.10

Table 1 Difference between estimated and measured times for performing a schedule of a given size. The error shows how many percent the estimated time differs from the measured execution times.

6 Experiments and Discussion

To be able to validate the different scheduling algorithms and compare how they perform we have run schedules of varying problem sizes produced by the different schedulers on a MLR1 tape drive. Just as during the simulations, the tape drive was positioned at the beginning of the tape before we started the first tape operation. Each operation consisted of reading a 32 KB block from a random position. For each schedule, the total time that the tape drive used to execute all the operations was measured.

Different criterias can be used to validate and compare the algorithm. First, we should be able to verify that the algorithms behave in the same way as in the simulations. Second, we should verify that the estimated execution times are approximately the same as the actual execution times. Third, we should be able to detect if any of the schedulers have any peculiar behavior which make them better/worse than predicted by the simulations.

The measured execution times per request for the different scheduling algorithms are presented in Figure 5. This figure should be compared to the corresponding simulated access times in Figure 4. By comparing these two figures, we see that the relative ordering of the schedulers are the same in both the simulated and in the real experiments, and confirms that MPScan* is the best algorithm.

From these figures, we also see that the estimated and experienced access times are approximately equal. To get a better apprehension of how good the estimated times to per-

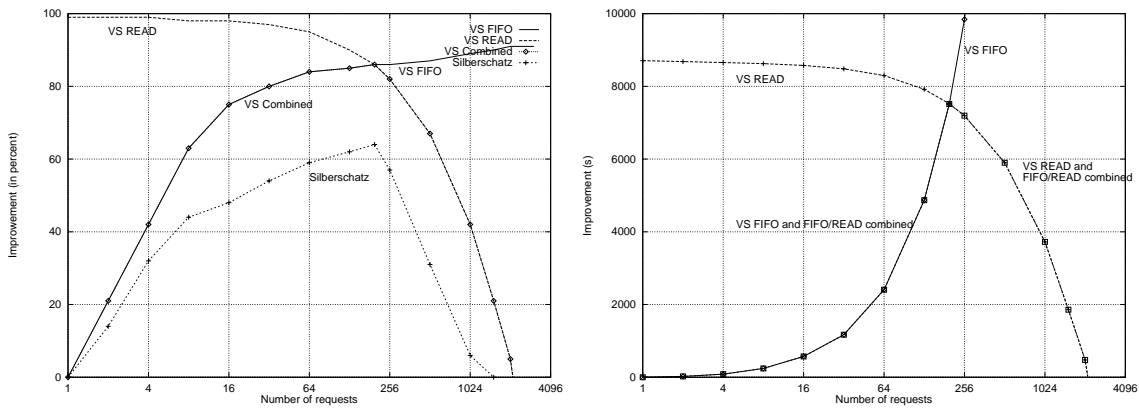


Figure 6 (a) Reduction in percent of total execution time for MPScan* compared to FIFO scheduling, compared to READ and compared to an optimal combination of FIFO and READ. Results from Hillyer and Silberschatz [3] show their best results compared to an optimal combination of FIFO and READ. **(b)** Reduction 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.

form a schedule are, Table 1 gives the average difference compared to simulations for the three best algorithms. These numbers are calculated by finding the difference in percent between the estimated and measured execution time for each of the schedules run on the MLR1. A positive value means that the estimated execution time is a number of percent higher than the actual measured execution time. Of course this error is not only a result of the particular scheduling algorithm, but also an estimate for how well the tape model estimates seek times. From the table, we see that most of the measured times are within +/-5 percent for schedules with less than 128 requests for SCAN and SLTF, and for all schedule sizes for MPScan*. The reason for the difference between measured and estimated execution times for long schedules produced by SCAN and SLTF is that these schedulers produce a seek pattern on the tape with a high number of changes of winding direction. For long schedules, these rapid changes of direction can impose slack in the tape, which makes the tape drive start to perform sub-standard.

We consider this to be reasonable good estimates for the actual execution times. Hillyer and Silberschatz [3] experiences less difference between estimated and measured times. This is probably due to their much more detailed tape model.

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 in order to minimize the waiting time for the requesting application(s) and to maximize the utilization of the (expensive) tape drive, which often is a bottle neck in tape systems. Figure 6 shows the relative and absolute improvements that the best scheduling algorithm, MPScan*, gives compared to the non-scheduling approaches, which are random order (FIFO) execution of the requests, reading the entire tape (READ), or a combination of FIFO and READ. MPScan* scheduling gives substantial benefits for all problem sizes from two I/O requests and up to 2100 I/O requests. The maximum gain is for a schedule of 196 requests where a MPScan* schedule executes in 20 minutes and 47 seconds, compared to an execution time of 2 hours, 5 minutes and 15 seconds for the

No. of requests	DLT	MLR1
1	95	62,6
2	72	44,2
4	54	28,5
16	38	12,1
64	31	7,4
256	24	6,2
1024	13	4,9

Table 2 Average access time in seconds for the DLT 4000 with the LOSS algorithm and for the MLR1 with the MPScan* algorithm.

corresponding FIFO schedule, saving more than one hour and 45 minutes.

It is interesting to compare the results for the Tandberg MLR1 to the results that Hillyer and Silberschatz [3] achieved for the Quantum DLT 4000 tape drive. In Figure 6 (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 DLT drive.

The average access times are much higher for the DLT drive than for the MLR1 drive. Table 2 shows the access times for the DLT drive with their best algorithm, 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 DLT drive with the LOSS algorithm, compared to the less than 23 minutes on the MLR1 with MPScan*. In effect the Tandberg MLR1 would be able to execute four schedules of 196 requests in the same time that the Quantum DLT 4000 processes one such schedule.

The Quantum 4000 drive and the Tandberg MLR1 drive are comparable, except that the DLT stores 54 percent more data (20 GB to 13 GB) and has a maximum seek time that is 43 percent longer (180 seconds to 126 seconds). Even if we take the longer maximum seek time into full effect (which is not reasonable, but still), the Tandberg drive performs three times as effectively for a schedule of 196 requests. The main reason why the MLR1 performs so much better, is not that MPScan* is a so much better algorithm than LOSS, but rather that the Tandberg drive has many more *key points* (25 to 13 on each track). The higher number of key points lead to much shorter *sawtooths*, 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 within the same sawtooth, while the MLR1 performs all operations at full speed.

7 Conclusion

In this paper we have proposed a model of access times for the Tandberg MLR1 serpentine tape drive, which balances the need of accuracy with the time needed to characterize each individual tape. The access time estimates are used to schedule random I/O against the tape drive and give results that are fully comparable with results that have been achieved with far more elaborate approaches. Further, we have proposed a novel scheduling algorithm, MPScan*, which gives very good results for all problem sizes and produces access patterns that are favorable to avoid certain irregularities in tape drive performance that can occur

with some other algorithms.

The simulations and practical experiments confirm earlier work by Hillyer and Silberschatz [3] and demonstrate that clever scheduling of I/O requests gives great performance improvements for a wide range of problem size in number of I/O requests. The random I/O performance achieved for the Tandberg MLR1 is much better than achieved by Hillyer and Silberschatz (op.cit.) for the otherwise comparable DLT 4000 serpentine tape drive. The main reason why the Tandberg MLR1 drive outperforms the DLT 4000 drive for random I/O requests is the higher number of *key points* used by the Tandberg drive.

The work presented is done as part of our research on storage and delivery of digital video [4, 9]. The access time model has been used for scheduling of requests for multimedia objects [10].

Notes

¹The order of the authors is random and conveys no information of unequal contributions to the contents of this paper.

References

- [1] S. Christodoulakis, P. Triantafyllou, 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] 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.
- [3] 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.
- [4] R. Hjelsvold, S. Langørgen, R. Midtstraum, and O. Sandstå. Integrated Video Archive Tools. In *Proceedings of ACM Multimedia '95*, pages 283–293, San Francisco, California, November 1995.
- [5] 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.
- [6] 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.
- [7] QIC Development Standard. *QIC-5010-DC, Serial Recorded Magnetic Tape Cartridge For Information Interchange, Revision E*. Quarter-Inch Cartridge Drive Standards, Inc., December 1994.
- [8] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.
- [9] O. Sandstå, S. Langørgen, and R. Midtstraum. Video server on an ATM connected cluster of workstations. In *Proceedings of XVII International Conference of the Chilean Computer Science Society*, pages 207–217, Valparaso, Chile, November 1997.
- [10] O. Sandstå and R. Midtstraum. Analysis of retrieval of multimedia data stored on magnetic tape. In *Proceedings from 1998 IEEE International Workshop on Multi-Media Database Management Systems*, pages 54–63, Dayton, Ohio, August 1998.
- [11] O. Sandstå 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*, San Diego, California, March 1999.
- [12] S. Sarawagi. Query processing in tertiary memory databases. In *Proceedings of 21st VLDB Conference*, pages 585–596, Zurich, Switzerland, September 1995.
- [13] Tandberg Data, Oslo, Norway. *Tandberg MLR1 Series Streaming Tape Cartridge Drives Reference Manual*, 1st edition, August 1996.