

Analysis of Retrieval of Multimedia Data Stored on Magnetic Tape

Olav Sandst  and Roger Midtstraum*

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7034 Trondheim, Norway
{olavsa, roger}@idi.ntnu.no

Abstract

This paper discusses scheduling of random I/O requests for multimedia data stored on magnetic tape using serpentine data layout. Results from simulations and real experiments show that substantial improvements of retrieval performance can be achieved by proper I/O scheduling. A new algorithm, Multi-Pass Scan Star (MPScan), is presented and shown to give better results than any other known algorithm. The level of Quality of Service that can be reached by use of MPScan* is discussed in the context of retrieval of image and short video segment data.*

1. Introduction

For emerging multimedia applications, like large image databases and video archives, which require really huge amounts of data storage, the low cost and high storage density can make magnetic tape a feasible storage alternative. Considering use of magnetic tape, there are three important drawbacks. First, in order to realize the low storage cost, a high number of tape cartridges must be gathered in a *tape library* which consists of a limited number of tape drives, a store for the cartridges when they are not in use, and a robot mechanism which moves cartridges between the store and the tape drives. When applications access data stored on a cartridge which is not mounted in a tape drive, they will have to wait until a tape drive is available, the current cartridge is unloaded from the drive, and the requested cartridge is loaded into the tape drive. At best, this waiting time will be tens of seconds. Second, as magnetic tape is a sequential medium, the *seek times* needed to locate data on a tape are horrendously long, typically in the range from 10 to 100 seconds. Third, the *data transfer rate* is limited compared to magnetic disk, and transfer of large data objects can take considerable time to complete.

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

To improve the performance of tape based systems, one can improve the efficiency of bringing the cartridges to and from the drives, improve the efficiency of finding the relevant data when a cartridge is mounted in a tape drive, or improve the efficiency of data transfer to/from the tape. While improvements of the data transfer rate are entirely in the hands of the manufacturers, and a lot of work have been done on efficient use of tape libraries [9, 10], little has been done to optimize the access of multimedia data stored on a single tape. For applications which have a non-sequential access pattern, it is of great importance to minimize the random access delay to the multimedia data stored on tape and thereby maximize the utilization of the tape drives.

In this paper, we study the problem of scheduling I/O requests for multimedia data stored on serpentine tape drives, using the Tandberg MLR1 tape drive as an example. While a lot of research has been done on modelling and scheduling of random access on disks, less has been performed for serpentine tape. Hillyer and Silberschatz [6] have studied random I/O scheduling of conventional data, but no previous work has been done for multimedia data. An alternative approach to improve the performance of tape drives would be to study optimal placement of data on tapes in tertiary storage libraries, as has been done by Christodoulakis et al. [1] and Ghandeharizadeh et al. [4]. Several projects have studied use of tape in video servers [3, 8], but have used a very simple tape model.

In order to keep the time consumption within practical bounds, we have developed a moderately complex model for the access times of a serpentine tape drive, and we provide low-cost algorithms to characterize each individual tape. Based on this model, we have simulated the performance of several known scheduling algorithms, and of a novel algorithm, MPScan*, for retrieval of different kinds of image and video data. The results from the simulations are validated by experiments on real tape drives. The main conclusion is that clever I/O scheduling provides significant speed-up of retrieval of multimedia data from serpentine tape drives. As MPScan* is shown to be the best algorithm,

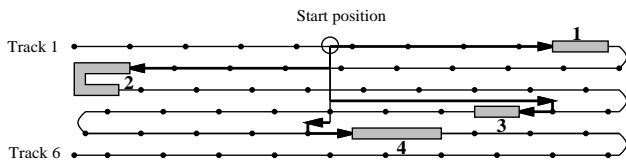


Figure 1. A simplified model of a serpentine tape with the key points marked on the tracks. From the current start position, possible seek patterns are indicated for four data requests.

we provide a detailed discussion of the Quality of Service that the use of this algorithm can provide for retrieval of image and video data.

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, and describes algorithms that perform static scheduling of random retrievals. Section 4 contains the results from simulations of scheduling tape requests for multimedia objects of different sizes. Section 5 presents results from experiments on real tape drives and discusses the merits of the scheduling approach. Section 6 gives the conclusions and outlines some further work.

2. Digital Tape Technology

There are three main tape technologies: helical-scan tape, parallel tape and serpentine tape. In this paper we focus on the *serpentine* tape model. Serpentine tape drives first read/write a track in the forward direction, then read/write the next track in the reverse direction, and so on, leading to a *serpentine pattern* for the data layout as seen in Figure 1. In the experiments reported in this paper we have used the Tandberg MLR1 tape drive [14]. This drive uses serpentine data layout and is based on the 13 GB QIC standard [11], making it possible to store 13 GB of data on each tape. 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. Scheduling of Tape Accesses

To optimize the utilization of tape drives, one must reduce the average access times of the objects stored on the tape. This can be done by using a scheduler to *re-order* random I/O requests to the tape. In this section, we present several such scheduling algorithms. To make these algo-

rithms perform well, one needs a fairly accurate model of the behavior of the tape drive. Such a model for a serpentine tape drive is presented as the first part of this section. This model is based on experiences and measurements from extensive use of the Tandberg MLR1 tape drive, but is general enough to be easily adaptable to other serpentine tape drives. A more detailed presentation and evaluation of the access time model can be found in [12].

3.1. Modelling Access Times

The *access time* is the amount of time it takes from a memory device getting an I/O request, until the data being made available to the requesting entity. An I/O request consists of the address of the first logical block requested and the number of consecutive blocks to be read. For a tape drive, the access time is the time used to position the tape (*seek time*), plus the time used to read the data (*transfer time*). On a serpentine tape drive, the seek time is determined by the physical distance between the two positions on the tape and whether the drive has to change track and/or direction, or not. The transfer time is determined by the amount of tape the drive has to read and the number of track changes.

Three important properties of the serpentine tape must be included in a model to make it able to estimate access times. First, the model must be able to estimate the physical position on the tape 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. Third, the model must be able to estimate the transfer time for a given tape interval.

An application which uses a Tandberg MLR1 drive accesses the blocks on the tape by logical addresses. The tape drive does not provide any information about the physical position on the tape for a given logical data block. For serpentine tapes, the amount of data that can be stored on a single tape varies from tape to tape. Thus, the number of blocks on each track varies. As a consequence of this, we have to characterize *each individual tape* to be able to precisely estimate the physical position of each logical block address. Hillyer and Silberschatz [5] have done this for a Quantum DLT 4000 drive. The DLT 4000 drive uses a set of predetermined positions, or *key points*, on the tape to locate the individual blocks. By locating each of the key points on the tape, they get a very accurate model. Unfortunately, it takes about twelve hours to locate all the key points for a single tape, making the model too time consuming in most practical applications. We use a much faster characterization of each tape, by locating only the start address of each track on the tape. These track addresses are then used to find the correct track number, and to estimate the physical position for any given logical data block. Finding the address of the first block on each track can be done at an insignificant cost

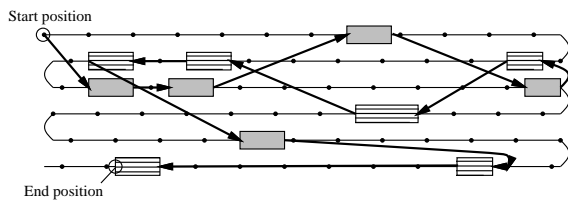


Figure 2. A MPScan schedule resulting in two full scans of the tape.

during the writing of the tape. If we are not able to log the writing of the tape, we can find the address of the first block on each track by reading a small interval on the end of each track and analyzing the read times, a process which takes about 13 minutes for each tape [12].

Experiments show that the Tandberg MLR1 drive also uses *key points* on the tape as starting points to locate the individual blocks on the tape. These key points are evenly spaced along the tracks of the tape as illustrated in Figure 1. When a tape drive gets an order for seeking a logical block address on the tape, it first seeks for the nearest key point preceding the requested block, then reads until it has found the requested block. This results in three main types of seeks as indicated by seek 1, 2 and 3 in Figure 1. For the first seek type, the time to perform the seek is determined only by the seek distance. For the second seek type, we have to include time to change track(s). For the third seek type, we have to take into account that the tape drive seeks longer than actually needed to find the key point, then changes direction and reads in the opposite direction to find the requested block address. To estimate the seek times between two physical tape positions, we have partitioned the possible seeks into nine disjunct cases. For each of these cases, we have run extensive practical experiments on tape drives to establish cost functions. These cost functions produce estimates for seek times between any two positions.

To estimate the transfer time of a tape access is much easier because the drive reads the tape at a constant data rate. Only in cases where the drive has to change track (as for seek 2 in Figure 1), the model has to include the average cost of a track change in the read time.

3.2. Scheduling Algorithms

Given a list R of I/O requests and an initial tape position I , the goal of *scheduling* the I/O requests for a serpentine tape drive 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 [6], and then propose a novel algo-

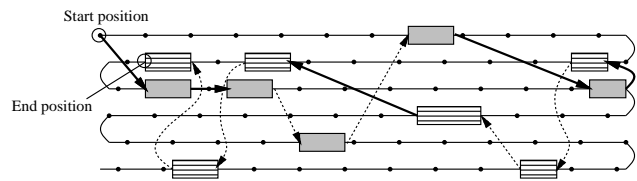


Figure 3. The MPScan schedule in Figure 2 reduced to one scan. New seeks introduced in the schedule are marked with dotted arrows.

rithm, Multi-Pass Scan Star (MPScan*), which makes good use of the streaming capability of the tape drive. To clarify the presentation, we have classified the algorithms based on the degree to which they take the physical geometry of the tape into consideration.

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 attempt to optimize the execution order. In cases where one does not have any knowledge of the properties of the storage device, this is the obvious method to schedule I/O requests.

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

One-dimensional algorithms

Algorithms in this class 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 know *elevator* algorithm used for disks. The requests in R are partitioned into two groups, one group with all requests on forward tracks and one group with all requests on reverse tracks. The requests in each group are sorted on *physical* tape position.

Two-dimensional algorithms

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 - as explained in [6], 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 smallest seek cost (locate time) as the first tape operation in the schedule. The current tape position is updated, and the next request to be included is the one with shortest seek time from this position. This is repeated until all requests in R are included in the schedule.

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 [6].

MPScan - Multi-Pass Scan - as the SCAN algorithm presented in the previous subsection, this is also an elevator algorithm. The main idea behind this algorithm is that tape streamers run most smoothly if they are allowed to stream, i.e. to read the tape in one direction for longer periods of time. The SCAN algorithm orders the operations in sequential physical order on the tape. Unfortunately, this does not guarantee that the tape drive avoids rewinding when it seeks to the next position in the schedule (see for example seek 4 in Figure 1). MPScan aims to guarantee streaming by selecting, as the next request in the schedule, the request which is closest to the current position and still so far apart that no interruption of the scan can occur because of rewinding to a key point.

An example which shows how this algorithm works, is given in Figure 2. The number of scans produced by this scheduler is highly dependent on the physical distribution of the requests. If the requests are scattered evenly on the tape, the number of scans that the tape drive has to perform will be fairly low. If the requests are clustered in physically close groups, the number of scans can become unreasonable high.

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 a schedule only by minimizing the cost that this single operation will incur on the final schedule. As more requests are scheduled, fewer requests are candidates for being scheduled next, leading to rather long seeks in the last part of the schedule. The main idea behind MPScan* is to avoid these long seeks at the end

Case	Data size	Objects on a tape
Compressed JPEG image	160 kB	79924
Uncompressed 600x800 image	1.44 MB	8881
High quality image	30 MB	426
1.5 Mbit/s video clip, 62 seconds	11.6 MB	1092
6 Mbit/s video clip, 62 seconds	46.5 MB	264

Table 1. The five different storage cases.

of an MPScan schedule, by removing the last scans from the schedule and inserting the requests somewhere else in the schedule. These requests are inserted into the schedule in the positions where the extra costs incurred are minimum. MPScan* starts by performing an ordinary MPScan scheduling of the requests. It then repeatedly removes the requests constituting the last scan from the schedule, and inserts them into the remaining schedule. This is repeated as long as the estimated cost of the new schedule (without the last scan), is less than the previous schedule.

Figure 3 shows the result of applying MPScan* to the same I/O requests as used for MPScan in Figure 2. 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.

4. Simulations

In this section, we present the results from simulations of the scheduling algorithms. The reason for simulating the algorithms is to compare the properties of the different scheduling algorithms for media objects of different sizes. The three most important properties of the scheduling algorithms are the *average access time* for the scheduled requests, the *initial latency* until the first request has been completed, and the *CPU cost* for computing a schedule. After an initial presentation of the simulated storage cases, the discussion of these properties is the topic of this section.

4.1. Storage Cases

To analyse the effect of scheduling in different situations, we have used five different multimedia cases in the simulations. First, we have considered storage of images of low, medium and high quality. Second, we have considered storage of short video clips of medium and high quality. To have realistic data sizes for the video objects, we have analyzed television news from TV2 Norway and found that their typical news program consists of 12 news stories, with an average duration of 62 seconds each. In a television news archive application, a set of such news stories would be the typical result of a query [7]. The details of the five storage cases are presented in Table 1. The bandwidths of the two video cases have been chosen to resemble typical MPEG-1 and MPEG-2 video streams.

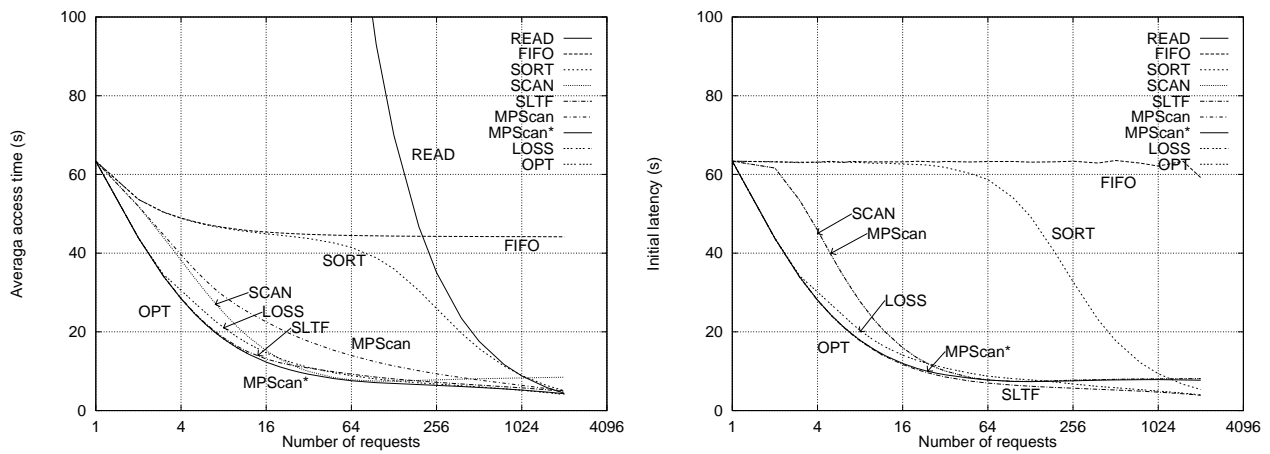


Figure 4. Average access time and initial latency for accessing images of 160kB from the tape.

4.2. Results

For each of the five storage cases, we consider a 13 GB MLR1 tape filled with corresponding media objects (see Table 1). To perform the simulations, we have made request lists containing from 1 to 2048 requests for distinct media objects on the tape. For schedules of lengths from 1 to 192 requests, we have repeated the simulation for 100000 different request lists, for schedules from 256 to 2048 requests we have gradually reduced the number of request lists from 25000 for 256 requests, to 500 for 2048 request. For the OPT algorithm, the largest request 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 4 shows the average access times for tape requests for images of 160 kB. With only one request, there is nothing scheduling can do to improve the performance, and the average access time for one 160 kB image will be 65 seconds. For requests of more media objects, no scheduling (i.e., use of the FIFO strategy) would result in an average access time of 45 seconds. In these cases, the average access times can be greatly reduced by using one of the better scheduling algorithms. For schedules with less than 12 requests, the average access time curves for OPT, SLTF and MPScan* overlap, and any of them could be used. For longer schedules, it is not feasible to use the OPT algorithm, and as long as the schedule contains less than 2100 requests, MPScan* gives the shortest access times. For longer schedules, the READ algorithm should be used.

Figure 4 also shows the initial latency until the first image is made available to the application. As for access times, the best scheduling algorithms provide substantial benefits compared to the FIFO approach. For large request sizes, SLTF gives the shortest initial latency. This is due to the

greedy behavior of always selecting the request with lowest seek time first.

To illustrate how the performances of the schedulers are influenced by the size of the requested media objects, Figure 5 shows average access times and initial latencies for the 6 Mbit/s video clips, which is the largest object size that has been investigated. Due to the much larger data size, the average access times and the initial latency have increased. Comparing the different schedulers, there are three points worth noting. First, MPScan* now performs better than SLTF over the entire simulation range. Second, when a high percentage of the objects on the tape is requested, LOSS performs better than MPScan*. Third, SCAN performs much worse than for smaller objects, because it quite often has to rewind long distances to get to the next request.

The simulations for the intermediate object sizes show similar results. MPScan* gives the best results for all object sizes, and should be preferred over the other scheduling algorithms. In the remaining of this section we present more detailed results for this scheduler. Figure 6 shows the average access times for all the five storage cases in Table 1, using the MPScan* scheduler. The access time curve for the READ scheduler is included to indicate the point where it is sensible to change to this approach. The figure shows that the MPScan* scheduler provides substantial improvements for all data sizes, compared to not using a scheduler. Note that larger data sizes increase the access times, and reduce the number of requests that are necessary to make READ the best solution.

The MLR1 tape drive has a maximum sustained data rate of about 1.5 MB/s. Figure 7 shows the effective data rates that can be achieved for different object sizes and numbers of requests, using the MPScan* scheduler. For the small images of 160 kB, the effective data rate is quite low, from 2.5 KB/s for schedules of one image to about 30 KB/s for sched-

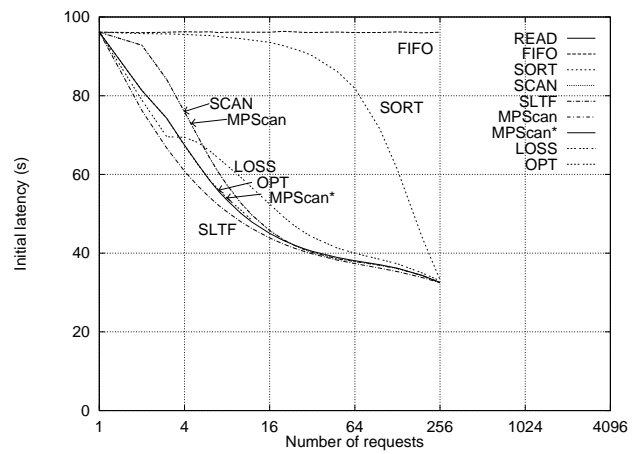
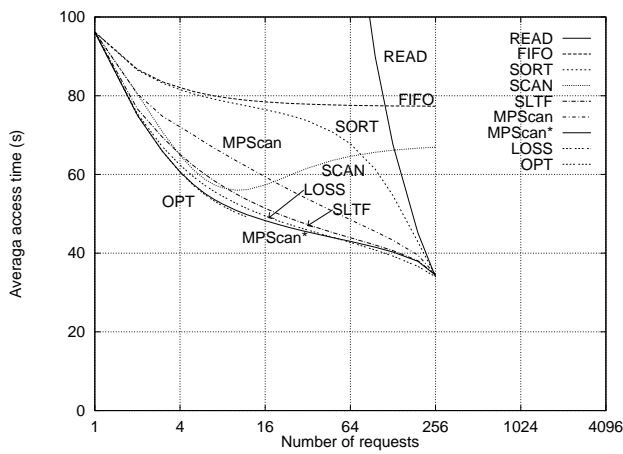


Figure 5. Average access time and initial latency for accessing 6 Mbit/s video sequences of 62 seconds from the tape. The average object size is 46.5 MB.

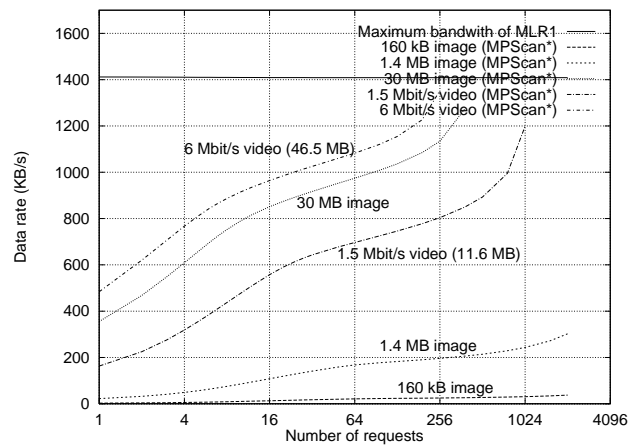
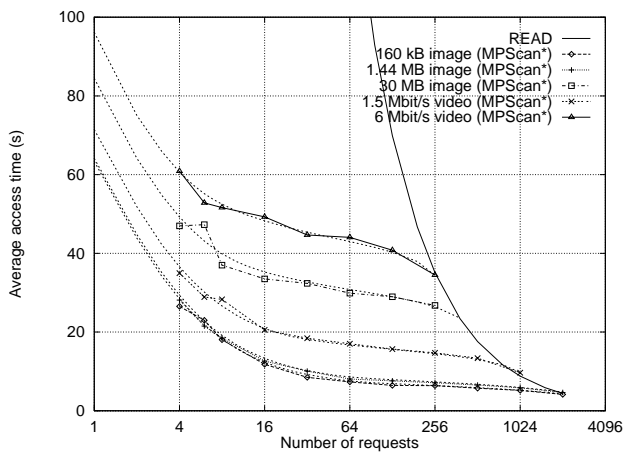


Figure 6. Simulated access times (dotted lines) using MPScan* compared to measured average access times.

Figure 7. Effective data rates using the MP-Scan* scheduler.

ules of 1000 images. As the object size increases, the effective data rate increases. As an example, accessing sixteen 6 Mbit/s video clips produces a high sustained data rate of almost 1 MB/s, which is a 66% overall utilization of the maximum data bandwidth of the tape drive.

4.3. CPU Cost

As always, there is a cost involved in computing better schedules. The simulations of the schedulers were run on a SparcStation 20 workstation with a 125 MHz HyperSparc processor. Each run of the algorithms was timed, and the average times to compute the different schedules are presented

in Figure 8. Considering only the feasible algorithms, we see that LOSS and MPScan* use most CPU time to compute the long schedules. Still, the extra CPU time is less than the saved execution time, compared to the other algorithms. For MPScan*, the problem of high startup delay, due to computational cost, can be reduced by starting execution of the first requests immediately after the initial MPScan-step of the algorithm has finished.

5. Experiments and Discussion

To validate the simulation results, we have run schedules of varying problem sizes on a MLR1 tape drive. By doing this, we were able to verify that the behavior of the al-

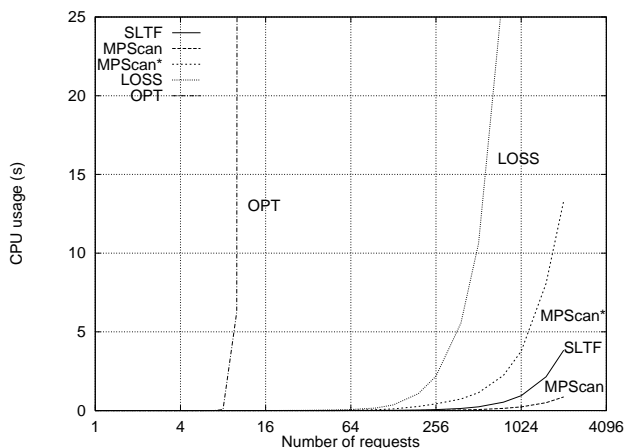


Figure 8. CPU usage for scheduling of requests.

gorithms are similar to the simulation results, and that the estimated execution times are approximately equal to the measured execution times. Since the MPScan* scheduler is the preferred scheduler to use for most problem sizes, we only present results from executions of schedules produced by MPScan*. The experiments show similar results for the other scheduling algorithms.

In the experiments, we used the same storage cases as during the simulations. The tapes were filled with fixed sized data objects, according to the sizes given in Table 1, and rewinded to the beginning of the tape before the first tape operation was started. Each tape operation consisted of reading an object of a given size from the tape. For each schedule, the total time used to execute all the operations was measured.

Figure 6 presents measured average access times for schedule lengths from 4 to 2048 requests for each of the five storage cases. Comparing the measured access times to the simulated access times for MPScan* shows that the simulated access times are good approximations of the measured access times.

To get a better apprehension of the accuracy of the estimated times, we have calculated the difference between the execution times estimated by the MPScan* scheduler and the measured execution times for each of the schedules. The average difference between estimated and measured times in our experiments was 3.2 %, with 95 % of the schedules having a difference less than 8.9 %. This shows that the estimated scheduling time is a good estimate for the actual time to perform the schedule, and is good enough to facilitate efficient scheduling of queries for multiple media objects stored on a tape.

5.1. 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 tape drive(s), which are often a bottleneck in tape libraries. Figure 9 shows the relative and absolute improvements that the best scheduling algorithm, MPScan*, gives compared to the best non-scheduling approach, which is an optimal combination of FIFO and READ. From the figure, one should make the following observations:

1. For all object sizes, the advantage of MPScan* first increases with the number of request, then reaches a maximum gain at the point where READ starts to get better than FIFO, and from that point on, the relative advantage decreases until READ finally takes over as the best approach.
2. As the object size increases, the relative advantage of MPScan* decreases, and READ takes over as the best algorithm at a lower number of requests. This is intuitive, since the transfer time takes a higher share of the total access time as the object size increases.
3. Until the points where READ takes over from FIFO, the absolute improvements (in seconds) are largely independent of the object size and only a function of the number of requests. This implicates that the seek times for MPScan* are determined by the number of requests only, and do not depend on the media object size.

For small image files, MPScan* more than halves the total execution time of all schedules containing from 5 to 768 requests. The best results are achieved for schedules of 196 request, which are executed in 1/7 of the time, saving more than 7300 seconds (more than two hours). For large video files, the savings are more modest. The total execution times are reduced with more than 15% for all schedules containing from 2 to 192 requests. The best results are for schedules of 96 requests, which are executed in half the time, saving more than 3500 seconds (slightly less than an hour).

5.2. Quality of Service

When magnetic tapes are used to store multimedia objects, the Quality of Service (QoS) parameters for the retrieval of the objects are: 1) *Initial latency*, which is the time from the data request is sent, until the first requested object is made available to the application, 2) *Inter-arrival times*, which are the times between delivery of two consecutive requests, 3) *Total execution time* for retrieving all requested objects, and 4) *Resulting data rate*. A fifth QoS parameter

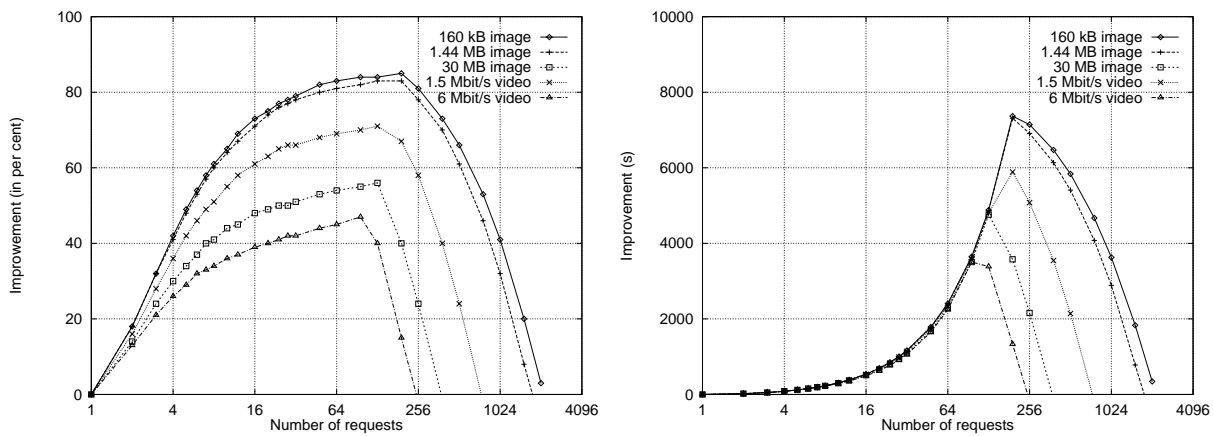


Figure 9. (a) Reduction in percentage of total execution time by using MPScan*, compared to an optimal combination of FIFO and READ scheduling. (b) Reduction in seconds of total execution time by using MPScan*, compared to an optimal combination of FIFO and READ scheduling.

being the degree of *variance* of the other QoS parameters. As the use of image and video data have different characteristics, we have organized the discussion of QoS in two parts, first considering QoS in the context of image objects, and then discussing QoS in relation to video data. The rest of this section discusses QoS as perceived from a single user. For more users, the QoS will be lower and fairness of service has to be considered.

Image Objects

Assume a user of an application retrieving a number of images from an image collection. For her, the important QoS parameters will be the initial latency, the inter-arrival times, and the total execution time. Figure 10a shows the number of images that will be retrieved from a tape during the first four minutes using FIFO and MPScan*. The figure contains the retrieval rates for small (160 kB) images and large (30 MB) images when the user has requested 128 images for retrieval. The user will experience much lower initial latencies if the application uses MPScan* instead of FIFO, in average 7 versus 64 seconds for 160 kB images, and 25 versus 83 seconds for 30 MB images. Also, the inter-arrival times are much lower for MPScan* than for FIFO, in average 7 versus 44 seconds for the 160 kB images, and 28 versus 65 seconds for the 30 MB images. As a result, the total time to retrieve 128 images of size 160 kB will be reduced from about one and a half hour, to less than 15 minutes by changing from FIFO to MPScan*.

Another quality of MPScan* is that the variations in both initial latency and inter-arrival times are much lower than for FIFO. For larger request sizes, these typically vary with a few seconds using MPScan*, while for FIFO these times

vary uniformly between a few seconds and up to two minutes.

Figure 10b shows how the retrieval rate for FIFO and MPScan* is influenced by the number of images included in a schedule. As the figure shows, FIFO is not influenced by the number of requested images. For MPScan*, both the initial latency and the inter-arrival time are reduced as the number of images in the schedule increases.

Video Objects

Contrary to image data, it is possible to estimate the time a user is going to spend “consuming” a video object. If VCR operations like *fast forward* are ruled out, a video object is consumed at the *playback rate* of the video stream, e.g. 1.5 Mbit/s. This facilitates computation of the production to consumption ratio, *PCR*, and introduces a sixth QoS parameter, *Waiting time*. Waiting time is the total time that the user spends waiting for video data to play.

Figure 11 shows the retrieval of four video clips. After the data request has been issued, the user has to wait the initial latency period, until the first video clip has been delivered from the tape drive. The user then starts playback of the video stream, while the tape drive goes on to fetch the next video clip. The figure shows what happens next for different PCR values. If $PCR < 1$, a user, which performs one uninterrupted playback of all the video clips, will have to wait for every video clip that is retrieved (e.g., a total waiting time of 5 time units for $PCR=0.5$ in the figure). If $PCR = 1$, a user will only have to wait for the first video clip, and the maximum buffer space equals the size of the largest video clip. If $PCR > 1$, a user experiences only the initial wait period, and the amount of video in buffer grows with the number of

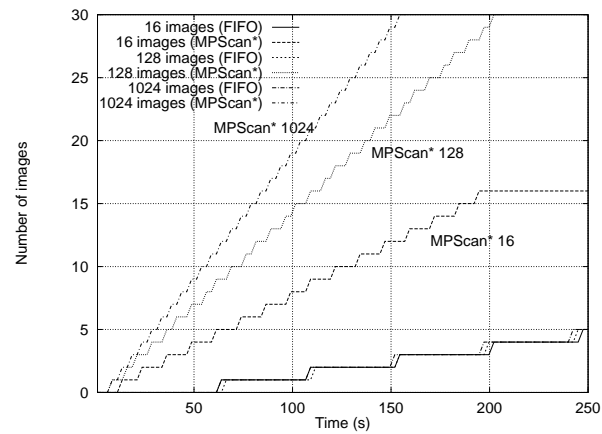
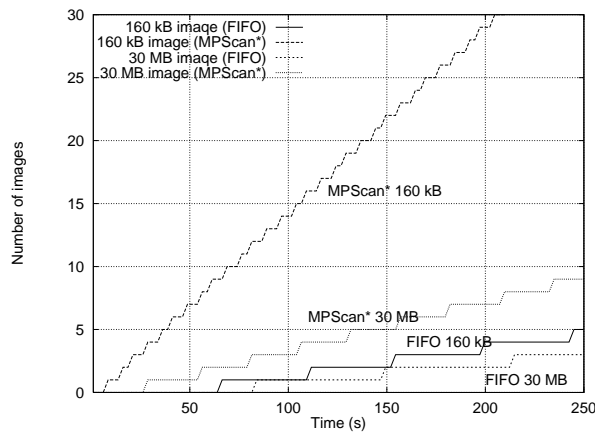


Figure 10. (a) Number of retrieved images for FIFO and MPScan* for different image sizes. The scheduled request was for 128 images. (b) Number of retrieved 160 kB images for FIFO and MPScan* for different schedule lengths.

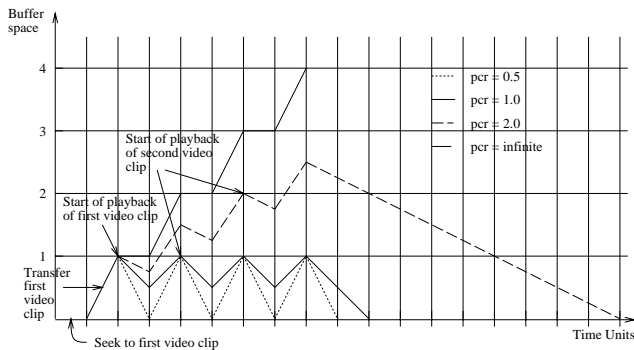


Figure 11. Retrieval and playback of four video clips for different PCR values.

retrieved video objects.

If buffer space is of little concern, one would want the PCR to be as high as possible, leading to the least total execution time and a minimum waiting time. If $PCR \geq 1$, it would be possible to start video playback as soon as the tape drive starts to deliver video data. This would reduce the initial latency, and possibly the total waiting time, but to simplify the discussion, this possibility will not be discussed any further in this paper.

Table 2 shows QoS parameters for retrieval of 1.5 Mbit/s and 6.0 Mbit/s video streams, respectively. The "Must see" column gives the fraction of each video clip that a user has to play in order to have a PCR of 1.0, and thus avoid waiting for the next video clip. A "Must see" value greater than one means that the user has to play the clip more than once (or rather, spend more time than one full playback). Without any scheduling (FIFO), the tape drive barely reaches a PCR

of 1.0 for video clips of medium quality, and the resulting QoS is going to be just acceptable. For a high quality stream, quite an amount of waiting time has to be expected. Utilizing MPScan* scheduling, the QoS improves to be quite good for medium quality video clips, and just acceptable for the high quality video. These QoS values are for video clips of 62 seconds. As long as the video data rate is less than the data transfer rate of the drive (1.5 MB/s or 12 Mbit/s), longer video clips would improve the QoS parameters (except total time), while shorter clips would make them worse (except total execution time).

It should be stressed that the QoS considerations made in this section are based on average behavior. The media object sizes and the seek times of tape drives are likely to vary within quite wide bounds. In order to obtain more reliable results, extensive simulation studies will be necessary.

6. Conclusion

In this paper, we have investigated static scheduling of random I/O requests for multimedia data stored on magnetic tape using serpentine data layout. The results showed that in many cases, clever scheduling provides substantial savings in both average access times and total execution times. As a result, we get much better utilization of the tape drives, which can give an improved Quality of Service for the users of applications which retrieve data from tapes. When used in hierarchical storage systems, the improved utilization of the tape drives can reduce the need for disk buffer, and hence reduce the total cost of the system.

Different scheduling algorithms have been implemented, and evaluated by simulations and by practical experiments

Video bit rate	Scheduler	Number of requests	Initial latency (s)	Inter arrival time (s)	Total time (s)	Production rate (KB/s)	PCR	“Must see”
1.5 Mbit/s	FIFO	1	72.0	–	72	161	0.86	–
		4	71.3	57.5	243	202	1.08	0.93
		16	71.3	53.8	878	216	1.15	0.87
		128	71.7	52.8	6777	220	1.17	0.85
	MPScan*	1	72.0	–	72	161	0.86	–
		4	36.9	36.5	146	319	1.70	0.59
		16	21.3	20.6	330	565	3.01	0.33
		128	15.0	15.2	1945	763	4.07	0.25
6 Mbit/s	FIFO	1	95.5	–	96	488	0.65	–
		4	95.8	81.4	340	571	0.76	1.31
		16	94.7	77.9	1263	597	0.80	1.26
		128	95.6	76.8	9849	605	0.81	1.24
	MPScan*	1	95.3	–	95	488	0.65	–
		4	66.0	59.9	245	776	1.03	0.97
		16	45.0	47.1	752	987	1.32	0.76
		128	36.0	39.2	5014	1187	1.58	0.63

Table 2. Quality of Service Parameters for a 1.5 Mbit/s and a 6 Mbit/s video stream.

on Tandberg MLR1 tape drives. Our new algorithm, MP-Scan*, is shown to give better retrieval performance than any other algorithm. Still, even use of the MPScan* algorithm can not provide wonders. Magnetic tape continues to be a rather slow random access storage medium, especially compared to magnetic disks. However, while one waits for high capacity, low cost and quick random access devices, possibly (re-)writable DVD disks [13], to become available, MPScan* provides a convenient way to make the most out of serpentine tape drives.

Three unsolved problems are the dynamic scheduling of requests which arrive during the execution of a schedule, the possible mismatch between the schedulers ordering of the requests and the user’s priorities, and fairness of service in case of multiple users.

Acknowledgement

We would like to thank Karel Babicky and Bernt Breivik, both Tandberg Data, for commenting on the technical questions we have had about the MLR1 tape drive.

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, 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] Y. N. Doğanata and A. N. Tantawi. Making a cost-effective video server. *IEEE Multimedia*, 1(4):22–30, Winter 1994.
- [4] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. A pipelining mechanism to minimize the latency time in hierarchical

multimedia storage managers. *Computer Communications*, 18(3):170–184, March 1995.

- [5] 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, May 1996.
- [6] 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, June 1996.
- [7] R. Hjelsvold, R. Midtstraum, and O. Sandstå. Searching and browsing a shared video database. In K. C. Nwosu, B. Thuraisingham, and P. B. Berra, editors, *Multimedia Database Systems*, chapter 4, pages 89–122. Kluwer Academic Publishers, 1996.
- [8] M. G. Kienzle, A. Dan, D. Sitaram, and W. Tetzlaff. Using tertiary storage in video-on-demand servers. In *Proceedings of COMPCON ’95*, pages 225–233, March 1995.
- [9] S.-W. Lau and J. C. S. Lui. Scheduling and data layout policies for a near-line multimedia storage architecture. *Multimedia Systems*, 5(5):310–323, September 1997.
- [10] 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, September 1997.
- [11] QIC Development Standard. *QIC-5010-DC, Serial Recorded Magnetic Tape Cartridge For Information Interchange, Revision E*. Quarter-Inch Cartridge Drive Standards, Inc., December 1994.
- [12] O. Sandstå and R. Midtstraum. Low-cost access time models for a serpentine tape drive. Technical Report IDI-3/98, Norwegian University of Science and Technology, 1998.
- [13] V. Shastri, P. V. Rangan, and S. Sampath-Kumar. DVDs: much needed “shot in the arm” for video servers. *Multimedia Tools and Applications*, 5(1):33–63, July 1997.
- [14] Tandberg Data, Oslo, Norway. *Tandberg MLR1 Series Streaming Tape Cartridge Drives Reference Manual*, 1 edition, August 1996.