# Random I/O Scheduling in Online Tertiary Storage Systems

**Bruce K. Hillyer**
Bell Labs
bruce@research.bell-labs.com

**Avi Silberschatz**
Bell Labs
avi@research.bell-labs.com

## Abstract

New database applications that require the storage and retrieval of many terabytes of data are reaching the limits for disk-based storage systems, in terms of both cost and scalability. These limits provide a strong incentive for the development of databases that augment disk storage with technologies better suited to large volumes of data. In particular, the seamless incorporation of tape storage into database systems would be of great value. Tape storage is two orders of magnitude more efficient than disk in terms of cost per terabyte and physical volume per terabyte; however, a key problem is that the random access latency of tape is three to four orders of magnitude slower than disk. Thus, to incorporate a tape bulk store in an online storage system, the problem of tape access latency must be solved. One approach to reducing the latency is careful I/O scheduling. The focus of this paper is on efficient random I/O scheduling for tape drives that use a serpentine track layout, such as the Quantum DLT and the IBM 3480 and 3590. For serpentine tape, I/O scheduling is problematic because of the complex relationships between logical block numbers, their physical positions on tape, and the time required for tape positioning between these physical positions. The results in this paper show that our scheduling schemes provide a significant improvement in the latency of random access to serpentine tape.

## 1 Introduction

Database storage systems require an overhaul of the underlying storage media to satisfy the need for online access to terabytes of data. The need is particularly acute for two classes of data management applications. The first class consists of applications that require the storage and retrieval of volumes of data too large to fit on magnetic disk alone. An example of such an application is EOSDIS, which will require the storage of about 1 petabyte of data per year [KB91]. The other class consists of applications that require terabyte-scale storage, but cannot afford the cost, size, or system complexity of a computing system that includes hundreds of large disk drives.

Removable media is the key to affordable and practical high capacity storage. The two main choices today are optical disk and magnetic tape. Optical disk has reasonable access times and is durable, but the cost per terabyte and physical volume per terabyte aren't substantially better than for magnetic disk. Moreover, the technology for high capacity optical disk is vendor specific, and keeps changing, rendering previously recorded storage media obsolete. By contrast, magnetic tape has significant cost and density advantages over disk. For instance, 70 cartridges for the Quantum DLT7000 tape drive can hold 2.4 terabytes in one cubic foot, at a cost of less than $6300. The superiority of tape for bulk storage is confirmed by the prevalence of multi-terabyte mass storage installations that use tape (e.g., in silos) as a major component.

Tape technology was classically applied to strictly sequential processing such as batch updating of master files. Recently, tape technology finds its use in data mining applications, where tens of thousands of queries are aggregated, and satisfied during one complete sequential scan of the data. The number of concurrent queries can be made sufficiently large that the application becomes CPU bound, not I/O bound. Tape technology, however, is not being used for mainstream processing in online database management systems. Since tape will continue to dominate bulk storage for the foreseeable future, we need to develop ways to seamlessly incorporate tapes into online database systems. However, to gain tape's capacity and cost advantages for more randomly accessed data, numerous problems need to be solved. The one addressed in this paper is how to efficiently retrieve sparsely scattered data from a tape, which has a random access time three to four orders of magnitude higher than disk. The premise of this paper is that significant speedups can be obtained by scheduling batches of random I/O's. Unlike the broad literature on



SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

algorithms for disk I/O scheduling to minimize total retrieval time [WGP94, GD87], to the authors' knowledge, no applicable algorithms for random tape I/O scheduling have been published. Tape scheduling was studied in [Won80, KMP90] under the assumption that positioning time between logical blocks $S$ and $D$ is proportional to $|S - D|$, but this assumption may be too strong for helical-scan tape, and is false for serpentine tape.

Various issues concerning tape storage technology have recently been addressed in the database literature. [KAOP91], [SSU91], and [CHL93] discuss the importance of progress in tape storage. [Isa93] describes a system for hierarchical storage management of a scientific database that exports a relational database interface. Striped tape organizations are studied by [DK93] and [GMW95]. [Sto91] describes an architectural framework to incorporate multilevel storage management in a database management system and [MH94] identifies several properties needed by object-oriented database systems that operate with tertiary storage. [ML95] studies join algorithms for databases stored partly on tape and partly on disk, and [Sar95, SS96] deals with issues of caching, query optimization, and mount scheduling for relational database use of tertiary storage jukeboxes.

The remainder of this paper is organized as follows. Section 2 describes the two basic tape technologies in current use. Section 3 presents a random access model for a Quantum DLT4000 and states the results of measurements to assess the model accuracy. Section 4 describes several algorithms that perform static scheduling of random retrievals from the DLT4000, and Section 5 presents model-driven simulations that assess the speed of the algorithms and quality of the resulting schedules. Section 6 presents measurements of schedule executions on the DLT4000 validate the simulation results, and Section 7 gives additional measurements that assess the sensitivity to model errors. Concluding remarks appear in Section 8.

## 2 Tape Technology

The two main classes of tape technology are *serpentine* tape and *helical scan* tape. A serpentine tape drive records a track (or group of tracks) down the length of the tape, then reverses direction and shifts the heads sideways a small distance to record another track (or group) up the length of the tape, continuing back and forth until tens of track groups have been written. One example of 1995 technology is the Quantum DLT4000, which has a sustained transfer rate of 1.5 MB/s, a tape capacity of 20 GB, and a price near $5000. The DLT7000 is 5.2 MB/s and 35 GB. The IBM 3590 is 9 MB/s and 10 GB, with a price near $44,000. A helical scan drive uses rotary head technology like that of a home video cassette recorder. One example is the Exabyte 8505 (500 KB/s, 7 GB, $1500). Another example is the Ampex DST (15 MB/s, 165 GB, $80,000).

Random I/O to a helical scan tape is easy to schedule because the logical block numbers correspond directly with physical tape position. If logical blocks $a$, $b$, and $c$ are ordered such that $a < b < c$, and the tape head is initially at $a$, the fastest order to read all three blocks is $a, b, c$. Thus, the best scheduling algorithm is simply "sort by logical block number and retrieve in order."

But current helical scan technology is unsuitable for intensive random I/O because of relatively rapid tape wearout. For instance Exabyte tapes can only withstand 1500 tape head passes under excellent conditions [Exa93], and in the less-than-pristine conditions of practical use, can develop unacceptable error rates after a few hundred passes. By contrast, the serpentine DLT tapes are rated for 500,000 tape head passes [Qua95], which equates to more than 3.5 years of continuous reading. Given a reasonable caching strategy, it is likely that a DLT tape will survive thousands of hours of intensive use under any access pattern.

Since database management workloads have a large component of random I/O, serpentine tape technology appears to be the better choice for a database system's online tertiary storage component; therefore, helical scan tape will not be considered further in this paper.

## 3 A Model of Locate Time

The *locate* primitive is the tape function analogous to the "seek" primitive on a disk. The DLT4000 [Qua95] manual does not provide any information concerning the locate time except that the maximum is 90 seconds and the average is 45 seconds. To gain an understanding of how the locate time behaves, we filled a DLT4000 tape with fixed-sized chunks of size 32 KB, termed *segments*, and measured the time of numerous locate operations.[1] The measurements show that the maximum locate time is about 180 seconds, the expected locate time from the beginning of tape to a random segment is 96.5 seconds, and 72.4 seconds between 2 randomly chosen segments. Moreover, the measurements demonstrate that the locate time is not a simple function of the source and destination segments. Indeed, for most source segments `x`, there exist approximately 300 destination segments `y` such that `locate_time(x,y-1)` exceeds `locate_time(x,y)` by about 25 seconds.

Figure 1 depicts the result of numerous measurements of locate times from segment 0 to other segments on the tape. In this figure, the solid curve shows the locate time from the beginning of the tape, in seconds, as a function of the destination segment. The dotted curve shows the corresponding rewind time. The vertical dashed

[1] The largest power of 2 that can be handled by the SCSI device driver in Solaris 2.3 is 32 KB. Writing a DLT4000 tape takes about 3 hours and 53 minutes for 622,102 segments.

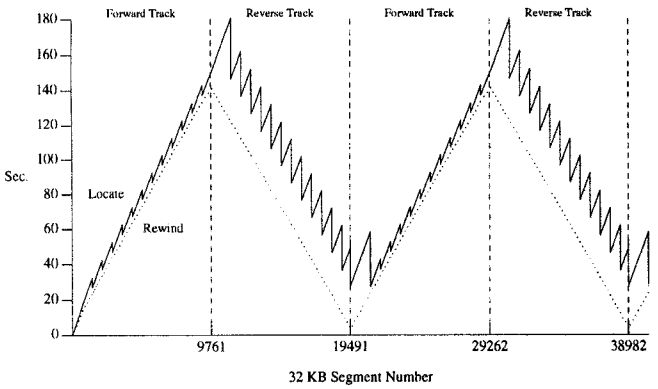

Figure 1: Detailed Locate Time Measurements.

lines in the figure denote track boundaries. The local minima in the sawtooth pattern are *dips*. Each dip is exactly one segment beyond a peak: the drop from peak to dip is abrupt. A *section* is the portion of a track from a dip to the following peak. Collectively, the segment numbers of the beginning and end of each track and the segment numbers of the 13 dips per track are the *key points* of a tape. Even numbered tracks are termed *forward tracks* because the tape motion is from the physical beginning of the tape toward the end, and odd numbered tracks are *reverse tracks*. The term *forward direction* is relative to the current location: it is always toward higher numbered segments; that is, toward the end of the tape in forward tracks, and toward the beginning of the tape in reverse tracks. Two tracks are *anti-directional* if one is a forward track and the other is a reverse track, *co-directional* otherwise.

The canonical identifier for a segment on tape is its absolute segment number; that is, the logical block number, which is 0 for the first chunk of data written to the tape, 1 for the second, etc. It is useful to define a physical coordinate system for serpentine tape that consists of `(track, section, segment)`, similar to the `(cylinder, track, sector)` coordinate system for disks. Section 0 within a track and segment 0 within a section are defined to be the ones closest to the physical beginning of the tape. The track numbers on a DLT4000 are 0–63 and section numbers are 0–13. Algorithms to determine the precise segment numbers of the key points are given in [HS96]. In essence, each dip is found by measuring locate times from the preceding dip.

Measurements indicate that tracks have differing lengths, perhaps reflecting differing amounts of space lost to bad spots. Similarly, the section boundaries in different tracks are at different physical distances from the beginning of the tape. Sections contain approximately 704 segments, except section 13 is significantly shorter. Because of the serpentine writing pattern, the first segment written on a forward track `t` is `(t,0,0)`, but the first segment written on a reverse track `t'` is `(t',13,k)`, where `k` has a typical value of 600 or so. The `(track, section, segment)` coordinate system has the property that `(t,a,b)` and `(t',a,b)` are physically nearby on tape, whether `t` and `t'` are co- or anti-directional.

The complete locate time model is described in [HS96]. The model is a function that has 8 major cases with 9 additional subcases, each of which is discontinuous and nonmonotonic, but piecewise linear. Below, we present an intuitive description of the model. The description refers to "read" and "scan", which are the two tape transport speeds of the DLT4000. "Read" is the slower speed (15.5 seconds per section) used for I/O transfers and short-distance tape motion, and "scan" is the higher speed (10 seconds per section) used for rewind and long-distance tape motions. The model cases are stated in terms of the location of the destination segment with respect to the source segment.

1. In the same track, in the same section or one of the following two sections: read forward.
2. More than two sections forward in the same track, or more than one section forward in a co-directional track: switch to the destination track, scan forward to the key point that is two before the destination, then read forward.
3. Backwards in the same track or a co-directional track, but not all the way to the first or second section, or forwards up to one section: switch to the destination track, scan backward to the key point that is two before the destination, then read forward.
4. Backwards in the same or a co-directional track, in the first or second section: switch to the destination track, scan backward to the beginning of the track, then read forward.
5. In an anti-directional track, reached by switching tracks and then proceeding forwards two sections or more: switch to the destination track, scan forward to the key point that is two before the destination, then read forward.
6. In an anti-directional track, reached by switching tracks and then proceeding forwards zero or one section, or reversing but not all the way back to the first or second section: switch to the destination track, scan backward to the key point that is two before the destination, then read forward.
7. In an anti-directional track, reached by switching tracks and then by reversing to the first or second section: switch to the destination track, scan backward to the beginning of the track, then read forward.

To assess the accuracy of the model, we performed a test of a random sequence of 3000 locate operations on the original tape used to develop the model, and compared the times with the model estimates. The difference exceeds 2 seconds for 7 of the 3000 locates.

A test of 1000 locates on a different tape gives an error exceeding 2 seconds for 24 locates. It may be possible to tune the model to obtain better agreement across a variety of tapes, but the sensitivity testing reported in section 7 confirms that the current model is sufficiently accurate to support the scheduling of random I/O.

Given the locate time model for the DLT4000 parameterized by the key points of a tape, it is possible to estimate how long it will take the DLT4000 to read a sequence of segments. This is the essential ingredient for scheduling: numerous possible rearrangements of a list of desired segments can be evaluated to predict which ordering will execute most quickly.

## 4 Scheduling Random I/O

We are now left with the problem of finding scheduling algorithms that produce schedules that minimize the time to retrieve a set of requested segments from a tape. We present eight algorithms. Some are analogous to disk scheduling algorithms, while others have been developed to take advantage of properties of the model of locate time for the DLT4000.

The algorithms and their time complexity are as follows. Let $I$ denote the initial position of the head before the execution of the schedule, $R$ denote the list of requests presented to a scheduling algorithm, and $S$ the reordered list of requests produced by the algorithm.

**READ (read the entire tape).** Read the entire tape sequentially and then rewind. This avoids the need to schedule the I/O's, and avoids using the locate operation. For a DLT4000 tape, a typical time to read an entire tape and rewind is 14,000 seconds (just under 4 hours).

**FIFO (first in, first out).** Perform the locates and reads as they are presented, without reordering them to improve the execution time.

**OPT (the optimal algorithm).** Scheduling a set of tape I/O's can be modeled as a variant of the traveling salesman problem that has a specified starting city, and that does not return to the original city after the last city has been reached. On the DLT4000, `locate_time(x,y)` typically differs from `locate_time(y,x)` by tens of seconds, so the *asymmetric* version of the traveling salesman problem applies.

For simplicity, it is assumed that each read operation retrieves a single segment. The extension to multi-segment reads is trivial. The input to the problem is a list $R$ containing $n+1$ segments (i.e., requiring $n$ locate and read operations); the first segment is the initial tape head position $I$. Note that the final tape head position is unconstrained; in an optimal schedule it will be $x+1$ for some segment $x \in R$. The output is a reordering $S$ of $R$ having the property that the total time to read all $n$ segments, starting with the tape head at $I$, is minimal.

An instance of this scheduling problem is represented by the following digraph $G$. Each segment $x_i \in R$, for $(i > 0)$, is represented by a pair of cities $x\_in$ and $x\_out$. The former corresponds with the position of the tape head when it has been positioned to read segment $x$, and the latter is the position after reading $x$, which is $x+1$ under the simplifying assumption of single segment reads. A *read edge* is directed from $x\_in$ to $x\_out$. The weight of this edge is the time to read a segment. After reading $x_i$, the tape could be positioned to any of the remaining $x_j$. Thus, each $x\_out_i$ in graph $G$ has $n-1$ out-directed *locate edges*, one to each point $x\_in_j$ $(j \neq i, j > 0)$. The weight of the edge from $x\_out_i$ to $x\_in_j$ is `locate_time(`$x_i+1$`,`$x_j$`)`. In addition, $x_0$ has $n$ locate edges to the $x\_in_i$.

An optimal schedule starts at $I$ and reads all the segments in $R$ in an order that minimizes the total locate time. This corresponds to a path in graph $G$ that starts at $I = x_0$, and passes through each point $x\_in_i$ and $x\_out_i$ via an alternating sequence of locate edges and read edges. This is uniquely represented by an ordering on the $x\_in_i$ (i.e., an ordering on $R$.)

The traveling salesman problem is known to be NP-hard in general cases. This is such a case: it is evident that the retraction of $G$ by collapsing $x\_in_i$ with $x\_out_i$ for all $i$ is a fully connected digraph on $n$ nodes, together with one in-directed edge from $I$. Thus, the computational complexity of OPT is exponential in $n$ (if $P \neq NP$). The implementation of OPT is an exponential algorithm that calculates the minimal locate time over all permutations of $R$ starting at $I$.

**SORT (order the requests by segment number).** For helical scan tape, the optimal schedule is $R$ sorted by segment number. This scheduling algorithm has computational complexity $O(n \log n)$. For a serpentine tape, it will be seen that SORT is poor for small $n$ but reasonably good when $n$ is large enough that nearly every section of the tape contains a request.

**SLTF (shortest locate time first).** This algorithm is similar to the shortest seek time first (SSTF) algorithm for disks, which is a greedy algorithm that starts at $I$ and iteratively proceeds to the nearest element of $R$ that has not yet been read.

The computational complexity of calculating an SLTF schedule for the DLT4000 is $O(n^2)$, because from each segment $x_i$ in $R$ it is necessary to find the segment $x_j$ (not yet in the partial schedule) that has the minimum distance from $x_i$. Fortunately, two facts about the locate model permit an improvement. First, reading ahead within a section is fast. In particular, given section $X$, and segments $x_i, x_j \in X$ such that $x_j > x_i$, and given segment $y \notin X$, `locate_time(`$x_i, x_j$`)` < `locate_time(`$x_i, y$`)`. Second, given a section $X' \neq X$, the segment $x_k \in X'$ that has minimum locate time from $x_i$ is the lowest numbered segment in $X'$.

Thus, to find the segment $x_j$ nearest to $x_i$, it suffices

to consider (1) the smallest segment further ahead within the same section, and if there is none, then (2) the smallest request in each of the non-empty sections. The first fact guarantees that whenever a non-empty section is chosen, the algorithm will consume all the requested segments in that section, in order of increasing segment number.[2] The second fact enables the algorithm to restrict its attention to the smallest segment number requested from each non-empty section. Thus, the computational complexity of SLTF scheduling for the DLT4000 need not be quadratic in n. The worst-case computational complexity is $O(n \log n + k^2)$ where $k \leq 896$ is the number of non-empty sections.[3] The first term accounts for sorting the requested segments within each non-empty section, and the second term accounts for searching based on the smallest element of each non-empty section.

It is fruitful to capitalize further on the idea of coalescing multiple segments into a single representative. The criterion for coalescing above was that segments within a single section are coalesced. A more aggressive approach coalesces segments whose difference is less than some constant. Given a distance threshold $T$, coalesce segments $s_i$ to representatives $r_k$ in the following way. (1) Sort the requested segments. (2) The initial representative $r_0$ is defined to be the first segment $s_0$. (3) Now iterate over the segments $s_i$ in increasing order: if $s_i - s_{i-1} < T$ and $s_i \neq I$ then coalesce $s_i$ into $r_k$, otherwise $s_i$ is the next representative $r_{k+1}$, and the iteration over $i$ continues. Experiments show that 1410 (the size of 2 sections) is a good choice for $T$, and that the quality of the schedule is not highly sensitive to $T$.

**SCAN (elevator algorithm).** SCAN is an important disk scheduling algorithm that scans the disk arm back and forth across the width of the disk, performing I/O operations at each requested cylinder when the head passes over. The analogous algorithm for serpentine tape is as follows. Each track has 14 sections. They are numbered 0 to 13, where 0 is physically closest to the beginning of the tape. Assume for simplicity of exposition that the starting location $I$ is at the beginning of the tape, and let `request(T,X)` denote the list of requests in track `T`, section `X`, sorted by increasing segment number. The algorithm alternately shuttles up the tape, reading sections in forward tracks, and then back down the tape, reading sections in reverse tracks. Inspection of the pseudocode (see Figure 2) shows the time complexity of the SCAN algorithm to be linear in the number of sections containing requests.

The resulting schedule differs from SORT in that

```
partial_schedule = empty.
while (some request has not yet been scheduled) {
  for X=0 to 13 {
    if (some forward track T has request(T,X)) {
      append request(T,X) onto partial_schedule, and
      set request(T,X) to empty.
    }
  }
  for X=13 to 0 {
    if (some reverse track T has request(T,X)) {
      append request(T,X) onto partial_schedule, and
      set request(T,X) to empty.
    }
  }
}
```

Figure 2: The SCAN Algorithm

the SCAN schedule tends to switch tracks more often, but take fewer passes up and down the length of the tape. For instance, given 3 requests having `(track, section)` coordinates (16,2), (17,12), and (18,3), the SORT schedule is (16,2), (17,12), (18,3) which takes two long passes over the length of the tape, whereas the SCAN schedule is (16,2), (18,3), (17,12) which traverses the length of the tape only once.

**WEAVE (a predefined relative ordering of the sections).** The WEAVE algorithm is an approximation to SLTF that requires no calls to `locate_time()`. It applies a pre-defined order to the sections containing requests, such that nearby sections are considered before far-away sections. The *weave pattern* is an ordering of the sections on tape relative to a given starting section. From a given section $X$, the first segment in the weave pattern is the segment immediately following $X$ in the same track (if one exists); the expected locate time is about 15.5 seconds, with a range of 0–31. The second section in the weave pattern is two sections forward in the same track (expected locate 31s, range 15.5–46.5), and the third is two sections ahead in any co-directional track (expected locate 40.5s, range 28–53). The overlapping locate time ranges make WEAVE only an approximation to SLTF.

The weave pattern is described as follows. Let `CT` denote the set of tracks co-directional with track `T`, and `AT` denote the set of tracks anti-directional with `T`. Let `flip(S)` be a function defined on section numbers 0..13 that flips the section numbers at the ends of the tape by mapping 0..13 to 1,0,2..11,13,12. Let `fwd(S,n)` be a function on section numbers that moves forward `n` sections from `S` in the same track. Similarly, `rev(S,n)` moves in the reverse direction. In the following specification of the weave pattern, ignore section numbers less than 0, greater than 13, or that repeat a section already seen. The complete weave pattern from `(T,S)` begins with: `(T,S)`, `(T,fwd(S,1))`, `(T,fwd(S,2))`, `(CT,fwd(S,2))`, `(AT,rev(S,1))`, `(CT,fwd(S,1))`,

[2] A minor exception is the section containing $I$. The first steps in the schedule will read $I$ and any larger segments in that section; segments before $I$ in that section can appear elsewhere in the schedule.

[3] The tape has a total of 896 sections: 64 track groups each containing 14 sections.

(AT,rev(S,2)). It continues with the following iteration:

```
FOR i=0..13 {
   (AT,flip(fwd(S,i))), (T,fwd(S,i+3)),
   (CT,fwd(S,i+3)),     (T,flip(rev(S,i))),
   (CT,flip(rev(S,i))), (AT,rev(S,i+3))     }.
```

Suppose a partial schedule $S_i$ has been constructed, and that $X$ is the section containing the last request in $S_i$. The WEAVE algorithm considers all the sections on tape, one at a time, in the order specified by the weave pattern starting from $X$. It stops at the first considered section $X'$ that contains a request not yet in $S_i$, sorts all the requests from $X'$ into increasing segment order, and appends this list to $S_i$ to form $S_{i+1}$. If some unscheduled requests still remain, it sets $X = X'$ and repeats the process. The time complexity of the WEAVE algorithm is $O(n)$ because for each section that contains a request, it iterates through every section on the tape, stopping at the first one containing a request that has not been scheduled.

**LOSS (a greedy algorithm for asymmetric traveling salesman).** Recall that SLTF is a greedy traveling salesman algorithm that that starts at the initial city and iteratively proceeds to the closest city that has not yet been visited. It is too greedy. It goes astray because it is oblivious to the fact that choosing the closest city *now* may force the path to traverse a very long edge *later*. The LOSS algorithm [LLKS85] is better. At each step it chooses the shortest out-edge from some city, or the shortest in-edge to some city. But the edge it picks is incident on a city where the difference between the shortest edge and second shortest edge is maximal, so choosing this short edge also avoids the future use of a much longer edge.

In more detail, the LOSS algorithm is as follows. The *out-loss* for a city is defined to be the length difference between the shortest and second shortest out-directed edges. Similarly, the *in-loss* of a city is the difference in length between the shortest and second shortest in-directed edges. The *loss* of a city, i.e., the "lost opportunity" if an edge incident on the city is not chosen at this iteration, is defined to be the larger of its in-loss and out-loss. At each iteration, the algorithm focuses on the city $C$ with highest loss. Suppose, for example, its out-loss is the cause of $C$'s high loss value. Then the loss algorithm chooses $C$'s shortest out-edge to be a part of the path. Suppose this edge connects $C$ to $C'$. Then all other in-edges at $C'$ are deleted from the graph, as are all other out-edges at $C$, and the loss values are recalculated for all cities affected by edge deletion. The time complexity of the LOSS algorithm is quadratic: for $n+1$ cities it iterates $n$ times to build a path containing $n$ edges, and at each iteration it visits all remaining cities to delete edges and recalculate the loss. As with SLTF, the LOSS algorithm can be improved by coalescing nearby segments into a single representative. The resulting reduction in problem size is significant given the quadratic time complexity.

Although the LOSS algorithm is sufficiently fast to be practical for all schedule sizes for which scheduling is superior to reading the entire tape, an idea for future work is to speed it up by starting with a sparse graph. A preprocessing step forms the coalesced representative cities, each of which covers all the requested segments in one or more consecutive sections of the tape. Then the graph construction starts at the out-end of each city, and explores the neighboring sections in weave order, adding edges to the graph until a logarithmic number of out-edges have been generated for each city. The resulting graph contains $O(n \log n)$ edges, so the running time of the LOSS algorithm is $O(n \log^2 n)$ because of $n$ iterations to select $n$ edges in the final path, each requiring $O(\log^2 n)$ time to delete a logarithmic number of other edges from $G'$, each deletion requiring at most a log-time rearrangement of a heap of loss values for the remaining cities.

Unfortunately, experiments demonstrate that the final schedules generated by the LOSS algorithm operating on the complete digraph contain a considerable number of very long edges, and so the LOSS algorithm will be unable to form a completely connected path if only the short edges are available. A possible solution is to run the $O(n \lg^2 n)$ LOSS algorithm until it can proceed no further, producing a disconnected collection of partial paths, then contract each of these partial paths to be a single city, forming a smaller graph, and repeat. This algorithm runs on a succession of reduced problems, iterating until a single connected path remains.[4] Evaluating a more sophisticated algorithm, such as that in [CDT95], remains as future work.

## 5 Simulation Studies

The algorithms presented in Section 4 have been evaluated by simulation. Numerous sets of segment numbers were randomly generated, and for each set, all scheduling algorithms were applied. The execution time of the resulting schedules was simulated using the DLT4000 locate time model. This section describes the simulation experiments and presents the results. Section 6 validates the simulation by measuring the actual run time of selected schedules, and Section 7 perturbs the locate time model to assess the effect of model errors.

The initial tape head position is chosen to be either at a random location on the tape, or at the beginning of the tape. The first models a scenario in which a tape is scheduled repeatedly, executing retrievals in batches.

[4] Colleague David S. Johnson notes that some modern algorithms for the traveling salesman problem have this flavor, producing partial paths of short edges, and then linking the partial paths into a single connected path.

```
initialize random number generator with seed.
for N in (1,...,10,12,16,24,32,48,64,96,128,
  192,256,384,512,768,1024,1536,2048) {
  for t from 1 to T[N] {
      generate a set of 1 + N segment numbers.
      for each scheduling algorithm {
         schedule the 1 + N locates.
         estimate the schedule execution time.
      }
  }
  for each scheduling algorithm, print mean &
    standard deviation of the total schedule
    execution time and the time per locate.
}
```

Figure 3: Model-Driven Simulation Experiment

In this case, at the beginning of each schedule execution the tape head is in the position of the last read in the previous batch. The second scenario applies to a robotic tape changer that has just loaded a new tape, so the tape head is at the beginning of the tape.[5]

Except for OPT, experiments that schedule from 1 to 192 requests are repeated 100,000 times to give good values for the mean and standard deviation of the expected schedule execution time. Because of CPU time limitations for the simulation, for schedules of length 256, 384, 512, 768, 1024, 1536, and 2048, the number of trials are 25,000, 12,000, 7,000, 3,000, 1,600, 800, and 400, respectively. OPT iterates 100,000 trials for up to 9 requests, 10,000 trials for 10 requests, and 100 trials for 12 requests.

Given a seed value for the Solaris `lrand48()` pseudo-random number generator, the simulation experiment is given by the pseudocode of Figure 3. `N` is the schedule length, and `T[ ]` is a table that states the number of trials for each value of `N`. The pseudorandomly generated segment numbers range from 0 to 622057, corresponding to the logical block numbers of the 622058 segments of size 32 KB on one particular DLT4000 tape.

Figure 4 depicts the average time per locate for the schedule produced by each algorithm, as a function of the schedule length, assuming a random starting position. The experiments are repeated with the initial starting point set to 0 (the beginning of the tape) to produce Figure 5. The entire set of experiments is repeated for 5 different initial pseudorandom number seeds. The mean schedule execution time varies by less than 0.5% across these 5 sets of experiments, except for the OPT algorithm on schedules of length 12, which has only 100 trials, where the mean varies 2.5%.

To assess the amount of CPU time required to generate a schedule using each algorithm, the simulation

[5] Single-reel cartridge tape technologies, including DLT and the IBM 3590, must rewind to eject.

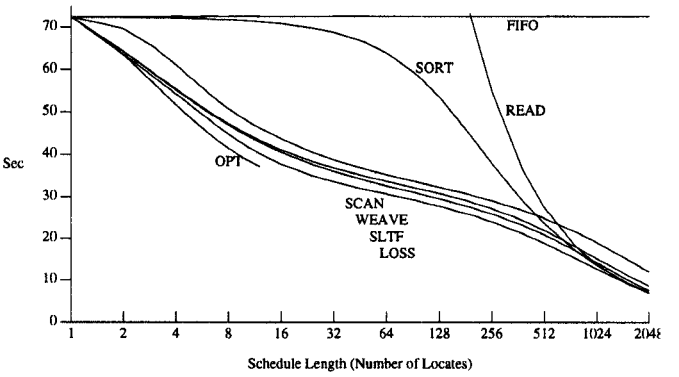


Figure 4: Mean Time Per Locate, Random Starting Point.

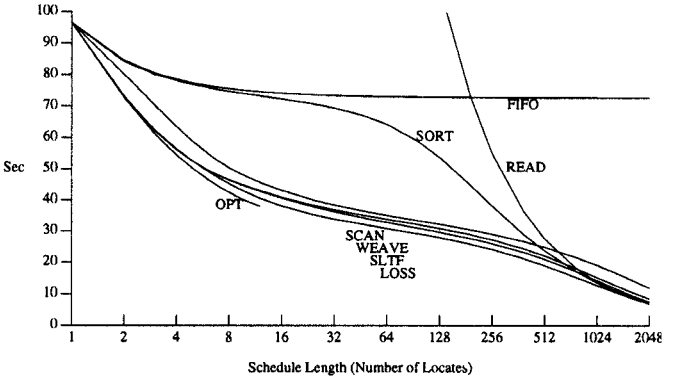


Figure 5: Mean Time Per Locate, Starting Location at Beginning of Tape.

experiments described above are timed on a lightly loaded Sun SparcStation 20/61 computer. Dividing the CPU time consumed by a simulation by the number of trials gives an approximation of the scheduling time. It is a slight overestimate, as it includes the overhead of random number generation and statistics calculation. The amount of CPU time required to generate a schedule is shown in Figure 6. Times too large to appear on the graph are as follows. OPT takes 0.6 seconds

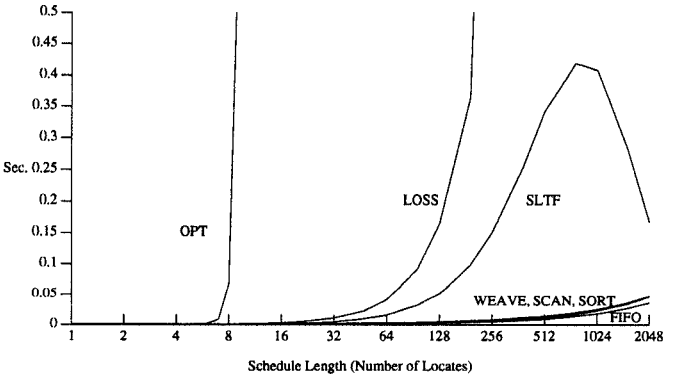


Figure 6: CPU Seconds to Generate a Schedule

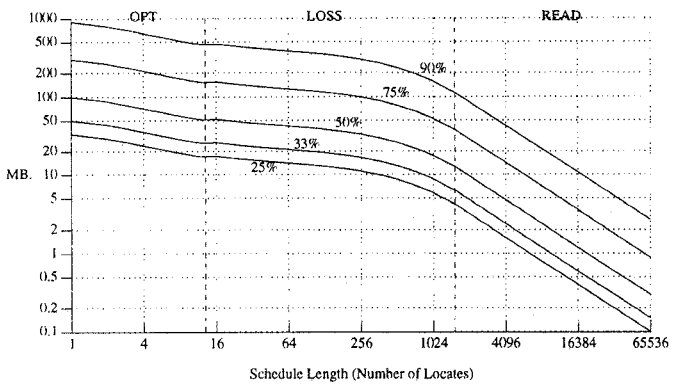


Figure 7: DLT4000 Utilization Curves per Schedule Length and Transfer Size

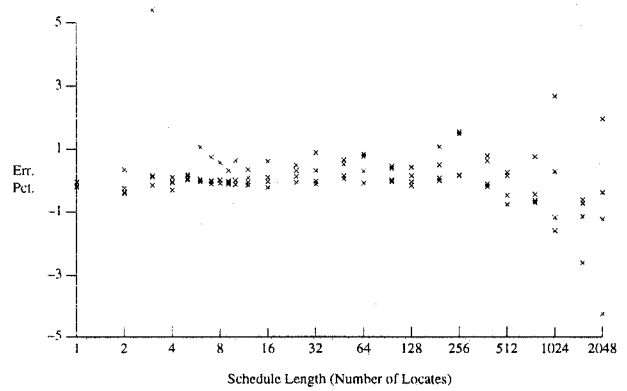


Figure 8: Percent Error in Estimated Schedule Execution Times, LOSS Algorithm

to schedule 9 locates, 6 seconds to schedule 10 locates, and 936 seconds to schedule 12 locates. For the LOSS algorithm on 256, 384, 512, 768, 1024, 1536, and 2048 locates, the CPU times are 0.6, 1.4, 2.3, 4.9, 8.1, 17.8, and 30.5s respectively.

The results of the experiments can be summarized as follows. OPT is recommended for scheduling up to 10 locates. Then, use the LOSS algorithm for up to 1536 uniformly randomly distributed requests. For more than 1536 requests just read the entire tape.

In the execution of a retrieval schedule, the percentage of time spent transferring data, as opposed to positioning the tape, is a function of the length of the schedule and the size of the retrieval requests. Figure 7 shows a family of curves that indicate combinations of schedule length and request size to achieve 25%, 33%, 50%, 75%, and 90% of the 1.5 MB/s sequential bandwidth of the DLT4000. It shows that by executing a schedule of 10 requests, each retrieving about 30 MB, the DLT can achieve a data rate comparable to that of a disk drive doing random retrievals of 8 KB pages.

## 6 Validation

As discussed in Section 3, the agreement between the DLT4000 locate time model and measured locate times is reasonably good. It remains to be determined whether this implies that the estimated schedule execution time is a good approximation of the actual schedule execution time on the DLT4000. To assess this, the estimated and measured execution times of schedules generated by the LOSS algorithm are compared for 4 trials at each schedule size. The results are depicted in Figure 8. The percent error is calculated as estimate less measurement, divided by measurement.

The results are that the model estimates are very accurate for schedules of fewer than 384 requests, typically with a difference much less than 1%. With large schedules the error increases to 5%, which is still sufficiently accurate to be useful. The reason for the decreasing accuracy with increasing schedule length is that a schedule of many requests contains numerous short locates near the physical track ends, and this region of the locate time model is less accurate.

## 7 Sensitivity

It is important to assess whether the estimated schedule execution time is sensitive to errors in the locate time model. Two kinds of experiments were conducted to test this. The first experiments investigate how the estimates deteriorate when the key points of a tape are determined inaccurately. The underlying question is whether it is really necessary to characterize each individual tape. The second set of experiments explores the impact of inaccuracies in the locate time model itself by systematically introducing errors into the model. The question here is how accurate the locate time model needs to be to avoid the generation of poor schedules.

To assess the importance of accurate determination of the key points of a tape, a set of experiments were performed to measure the schedule execution time on tape $A$ for schedules produced by the LOSS algorithm given the actual key points for tape $B$. These experiments replicate the validation experiments in the previous section using key points for the wrong tape. The consequence is disastrous, with the typical difference between estimated and measured time about 20%, as depicted in Figure 9.

To assess the impact of errors in the locate time model itself, the simulation study of the LOSS algorithm as depicted earlier in Figure 5 was repeated several times with different amounts of error inserted into the locate time function. Specifically, given the original function $locate\_time(source, destination)$ and an error amount $E$ that takes the values 1, 2, 3, 5, and 10 seconds, the function is altered to return $locate\_time(S, D) + E$ if $D$ is even, and $locate\_time(S, D) - E$ if $D$ is odd.

The effect of the error is seen in the graph of Figure 10, which shows the mean percentage increase in the

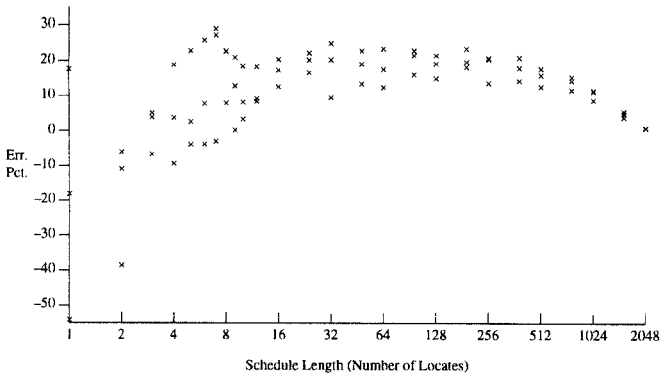


Figure 9: Percent Error in Estimated Schedule Execution Times, Wrong Key Points

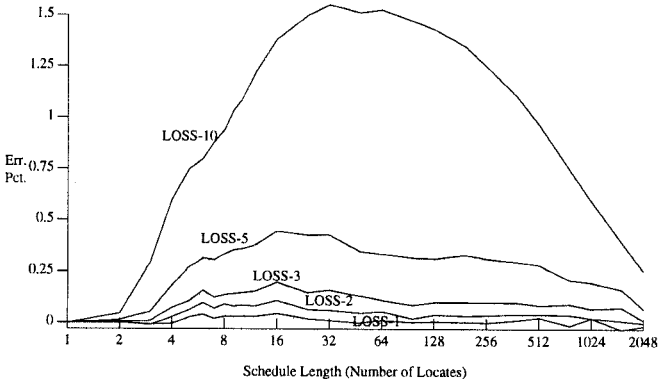


Figure 10: Execution Time % Increase, LOSS Algorithm, Perturbed Locate Model

estimated execution time of schedules generated by LOSS for schedules having from 1 to 2048 locates and the starting position at the beginning of the tape. The curve labeled LOSS-1 is the estimated percentage increase in execution time with $E$ set to 1 second. Similarly, LOSS-2, LOSS-3, LOSS-5, and LOSS-10 show the effect of the locate model error $E$ taking the values 2, 3, 5, and 10 seconds. The number of trials for this experiment is the same as for Figure 4.

A similar experiment for the OPT algorithm with the number of locates ranging from 1 to 12 shows no estimation errors even for $E = 10$. A possible explanation is that LOSS is greedy and can be lead astray by a single bad edge length, whereas OPT chooses a schedule based on the total estimated time, and this error model has an average error of 0.

The conclusion drawn from this set of experiments is that, given a locate model that has errors as described above, errors of 2 seconds or less have little effect, and errors of 10 seconds can degrade the execution time of a schedule by 1 or 2 percent. For schedules having fewer than 4 locates the error is small because the requests are far apart, so the locate times differ by more than the error. For schedules having more than 700 requests, the requests are becoming dense on the tape (i.e., approaching 1 per section). In this case, the LOSS algorithm generates a schedule dominated by sequential access from one section to the next.

A comparison of the two sets of experiments in this section reveals that the locate model is much more sensitive to erroneous key points than it is to modest errors in the linear functions incorporated in the cases of the model. The reason is that errors in the key points cause the model to misidentify which section of tape a segment is in, and the difference in locate time between adjacent sections is large, typically 5 seconds in forward tracks and 25 seconds in reverse tracks.

## 8 Concluding Remarks

The work described in this paper is a first step toward the ultimate objective of extending database management systems to smoothly integrate online tertiary storage. We note that the earlier problem of bridging the 5 orders of magnitude performance gap between main memory and disk storage has been dealt with by a combination of algorithmic techniques, including prefetching, scheduling, and caching, together with hardware approaches such as disk farms. The analogous problem exists for tertiary storage—how to bridge the 4 orders of magnitude gap in random I/O latency between tape and disk. The work reported in this paper applies I/O scheduling techniques to narrow the gap.

The results in a nutshell are as follows. For the DLT4000 tape drive, the random retrieval rate without scheduling is 50 I/O's per hour. With a schedule of length 10, the OPT algorithm improves the retrieval rate to 93 I/O's per hour. The LOSS algorithm provides 124 I/O's per hour with schedule length 96, and 285 I/O's per hour with schedule length 1024. For a batch of 1536 requests, a LOSS schedule is no faster than reading the entire tape, and the rate is 391 I/O's per hour. The absolute improvement is significant: without scheduling, 192 random I/O's requires 3.87 hours. The LOSS algorithm improves this to 1.37 hours, saving 2.5 hours.

Another view of the results is suggested by Figure 7, which shows DLT4000 utilization curves for a workload that does not exhibit locality or sequentiality in the reference pattern. Because of the large random access latency of tape, solitary I/Os need to transfer contiguous chunks of at least 50–100 MB to get good device utilization. Scheduling reduces the access latency, giving acceptable utilization with significantly smaller transfer sizes, in the range 10–25 MB.

These results indicate that scheduling can reduce the latency gap between tape and disk by nearly one order of magnitude. To bridge the remaining 3 orders of magnitude is the subject of our ongoing research.

## References


[CDT95] G. Carpaneto, M. Dellamico, and P. Toth. Exact solution of large scale, asymmetric traveling salesman problems; algorithm-750. *ACM Transactions on Mathematical Software*, 21(4):394–415, December 1995.

[CHL93] Michael J. Carey, Laura M. Haas, and Miron Livny. Tapes hold data, too: challenges of tuples on tertiary store. In *Proceedings SIGMOD International Conference on Management of Data*, pages 413–417, Washington, DC, May 26–28 1993.

[DK93] Ann L. Drapeau and Randy H. Katz. Striped tape arrays. In *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems* [MSS93], pages 257–265.

[Exa93] Exabyte Corporation, 1685 38th St., Boulder, CO. *Media Guide: Digital 8mm Media*, mkt-122-01 edition, January 1993.

[GD87] Robert Geist and Stephen Daniel. A continuum of disk scheduling algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, February 1987.

[GMW95] Leana Golubchik, Richard R. Muntz, and Richard W. Watson. Analysis of striping techniques in robotic storage libraries. In *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, pages 225–238, Monterey, CA, September 11–14 1995.

[HS96] Bruce K. Hillyer and Avi 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*, Philadelphia, PA, May 23–26 1996.

[Isa93] David Isaac. Hierarchical storage management for relational databases. In *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems* [MSS93], pages 139–144.

[KAOP91] Randy H. Katz, Thomas E. Anderson, John K. Ousterhout, and David A. Patterson. Robo-line storage: low latency, high capacity storage systems over geographically distributed networks. Technical Report Sequoia 2000 Technical Report S2K-91-3, University of California, Berkeley, October 1991.

[KB91] Ben Kobler and John Berbert. NASA earth observing system data information system (EOSDIS). In *Proceedings of the Eleventh IEEE Symposium on Mass Storage Systems*, pages 18–19, Monterey, CA, October 7–10 1991.

[KMP90] John G. Kollias, Yannis Manolopoulos, and Christos H. Papadimitriou. The optimum execution order of queries in linear storage. *Information Processing Letters*, 36(3):141–145, November 1 1990.

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

[MH94] David Maier and David M. Hansen. Bambi meets Godzilla: object databases for scientific computing. In *Proceedings of the seventh International Working Conference on Scientific and Statistical Database Management*, pages 176–184, Charlottesville, VA, September 28–30 1994. IEEE Computer Society Press.

[ML95] Jussi Myllymaki and Miron Livny. Disk-tape joins: synchronizing disk and tape access. In *Proceedings of the 1995 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 279–290, Ottawa, Canada, May 15–19 1995.

[MSS93] *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 26–29 1993.

[Qua95] Quantum Corporation, 715 Sycamore Ave., Milpitas, CA 95035. *Quantum DLT4000 / DLT4500 / DLT4700 S Tape Mini-Library Product Manual, 81-108336-01*, 1995.

[Sar95] Sunita Sarawagi. Query processing in tertiary memory databases. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 585–596, Zurich, Switzerland, September 11–15 1995.

[SS96] Sunita Sarawagi and Michael Stonebraker. Execution reordering in tertiary memory databases. 1996. http://http.cs.berkeley.edu/šunita/.

[SSU91] Avi Silberschatz, Michael Stonebraker, and Jeff Ullman. Database systems: achievements and opportunities. *Communications of the ACM*, 34(10):110–120, October 1991.

[Sto91] Michael Stonebraker. Managing persistent objects in a multi-level store. In *Proceedings SIGMOD International Conference on Management of Data*, pages 2–11, Denver, CO, May 29–31 1991.

[WGP94] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 241–251, Nashville, TN, May 16–20 1994.

[Won80] C. K. Wong. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. *ACM Computing Surveys*, 12(2):167–178, June 1980.