Determining the Optimal File Size on Tertiary Storage Systems Based on the Distribution of Query Sizes*

Luis M. Bernardo Henrik Nordberg Doron Rotem Arie Shoshani
Scientific Data Management Research Group
NERSC Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

Abstract

In tertiary storage systems, the data is stored on multiple tape volumes where each tape is further divided into files. Since in many such systems the minimum unit of data transfer is a file, it is an important problem to match file sizes with the access patterns to the data. In general, if the file size is large relative to the query size it will lead to the transfer of large amount of irrelevant data whereas small file sizes will incur an overhead penalty associated with reading each new file. In this work, we analyze the relationship between file sizes and query response times and provide a methodology to compute the optimal file size given information about the distribution of query sizes. Exact closed form solutions for the cost function are given for two common distributions.

1. Introduction

In many scientific domains large volumes of data are often generated by experimental devices or simulation programs. Examples are atmospheric data transmitted by satellites, climate modeling simulations, high energy physics experiments, etc. The volumes of data may reach hundreds of terabytes and therefore it is impractical to store them on disk systems. Rather they are stored on robotic tape systems which are managed by some mass storage system (MSS). A MSS is a software system that manages a hierarchy of storage devices, including robotic tape libraries and disk cache systems (such as RAID) [5]. It is typical for scientific data to be stored as files on tapes according to a predetermined order, usually according to the order they are generated. In order to avoid searching all the files when we need to retrieve the data, the files are indexed externally to the MSS according to some attributes (properties) associated with each file. For example, satellite data may be indexed by time and spatial location, and high energy physics data indexed according to the properties of the particle collisions (we focus on this application below). When it is necessary to retrieve the information, the index is used to determine which tapes should be mounted for reading and which files should be read out of each tape.

As the time to retrieve data from a robotic tape library is greatly affected by the cost of mounting new tapes or having to rewind tapes for repositioning, there has been some research done to find the optimal way to break the data across tapes [3, 2]. Other research efforts were directed at the problem of reordering the reading of query results from the tape to avoid excessive rewindings and tape mounts [8, 9, 6, 7, 4].

One important problem associated with reading data from tapes is determining the file size to store data on each tape. Some MSSs permit only the reading of entire files, and therefore what is on a file relative to the query requesting information from the file is critical. Ideally, one would want to match the file size to the query requesting the file (i.e. the file contains only the information that the query needs). In practical shared systems this presents an optimization problem for determining the file size given a certain query mix.

Some MSSs permit partial file reads, *i.e.*, the system will read from tape only the requested portion of the file. One can incorrectly perceive that this feature permits files to be

*This work was supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

as large as one wishes, because one can read only the data needed. However, because tapes can only be read sequentially, reading data from the middle of a file requires getting to the data location at read speed. To see this point suppose that there is only a single file per tape (*e.g.* a single file of size 30 GBs). Suppose that a query submitted to the system needs only 50 MBs located in the middle of the file. A fairly fast tape system reads at a rate of 5 MBs/sec. Thus reading to the middle of the file will require 50 minutes! On the other hand, if the 50 MB where stored as a single file, the tape reader will move to that location at seek speed, which is about 100 times faster. It will take about 30 seconds instead.

Thus, it is essential that the "chunks" of data that go into a single file match well the queries being submitted to the system. This is the problem addressed in this paper. There are two conflicting considerations in determining the file size. At one extreme, one can choose the file size as the smallest chunk needed by the intersection of the query mix. In this case, there may be queries that span several files . Such queries pay the overhead of reading multiple files instead of a single larger file. We refer to this cost as the "file splitting overhead", and will explain this in more detail in Section 3. On the other hand, if we choose to combine several small chunks into a larger file, queries that need only small chunks pay the overhead of reading unnecessarily large files. The tradeoffs between these two overheads is the optimization problem we are addressing in this paper.

We explain below in some detail a typical high energy physics application as the context in which this problem was explored. This application domain is not unique. Similar considerations exist in other applications that access large datasets stored on robotic tape systems.

1.1. Background on high energy physics data

High Energy and Nuclear Physics (HENP) experiments consist of accelerating sub-atomic particles to nearly the speed of light and forcing their collision. A small part of the particles collide "head on" and produce a large number of sub-particles. Each such collision (called an "event") generates in the order of 1-10 MBs of raw data collected by a detector. The rate of data collected is a few such events per second, or about 10 MB/s on the average. This corresponds to $10^7 - 10^8$ events/year, and the total data volume amounts to about 300 TBs per year. A typical experiment may run for 3 years. Such volume of data needs to be stored on tapes, since it is impractical to store them on disk systems.

After the raw data is collected, it undergoes an "event reconstruction phase". Each event is analyzed to determine its sub-particles and to extract summary properties for each event (such as the total energy of the event, momentum, and the number of particles of each type). The amount of data thus generated is about a tenth of the raw data, which amounts to about 30 TBs per year. A typical analysis that physicists wish to perform on the reconstructed data involves the selection of some subset of the events according to some conditions over the summary information. For example, one may wish to select all events that are at a certain energy level and that have only 2 or less electrons or muons. Such a selection of events from the tapes may involve mounting a large number of the tapes, each containing a relatively small number of the desired events.

Data retrieval from these tape systems form a major bottleneck because of the inherent sequential nature of the devices (requiring long read time to reach the desired events in a file), and the time to mount and dismount tapes. The problem that needs to be addressed is how to retrieve desired subsets of the events for analysis from the tape storage system in the most efficient manner.

One aspect of this problem is determining how many events to put into each file. The focus of this paper is on the optimal file size that should be used to store events.

This paper is organized as follows. In Section 2 we describe the system architecture we are in the process of implementing. In Section 3 we formulate an optimization problem that determines the optimal file size given that the distribution of query sizes is known and that queries are independent of the location on tape. In Section 4, we solve this optimization problem using two of different query size distributions. Finally, in Section 5 we summarize our conclusions and comment on possible generalizations and future work.

2. General system architecture

We describe briefly below the system architecture in order to point out when the file size analysis is used. The system has several components that correspond to the different phases of the system operation: analysis, restructuring, and access.

The architecture of the system is depicted in Figure 1. It has three main modules. At the top left of the figure is the Clustering Analyzer. The Clustering Analyzer (CA) is the module that facilitates the selection of properties of the events to cluster on. The input to this module is a file that contains the properties of each event. The module was designed so that the data administrator (the physicist responsible for the data organization on the system) can choose the properties that exhibit the highest clustering level. The result of this module is a specification on how to reorganize the events on the tapes for efficient access. We note that queries for this application tend to be expressed as range queries on several properties that characterize the physics phenomena of interest. In practice it may be necessary to observe the actual queries in order to determine the most

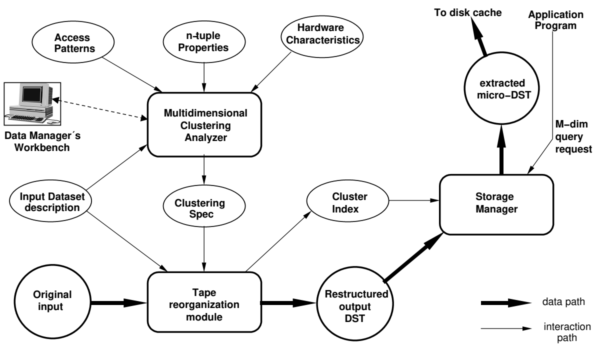

Figure 1. Architecture of system for event reorganization and access.

useful properties for the cluster analysis. The selection of properties to cluster on is done by physicists using the "data manager's workbench". Once a selection is made according to the most likely dimensions to be used in queries and the clustering properties, the CA generates a "clustering spec". This serves as the input to the second main component: the Tape Reorganization module.

The Tape Reorganizer (TR) is the component responsible to reorganize the tapes according to the reordering specification generated by the Clustering Analysis (CA) module. It has three main components. 1) The Reorganization Analyzer - the component that produces an execution plan to minimize the mounting of the tapes that need to be read and written into. 2) The Reorganization Monitor - the component that monitors the reading and writing of tapes by the mass storage system (we use HPSS [1] for this project). Because the reorganization process may be very long (many hours) this module keeps track of the status of the reorganization (flow control), so that in case of a device or system failure, it can resume the operation from the last point of interruption. 3) The Directory Generator - the component that generates the index of the location of events that were clustered into files, and the location of files on tapes. The Index is used by the third main component, the Storage Manager. It is in this TR module that the file size analysis is used. The events are organized into files by this module according to the optimal file size selected. The file size analysis is the subject of this paper.

The third component, the Storage Manager (SM) is responsible for determining, for each query request, which events and files need to be accessed. A specialized index is used for quick (real-time) estimation of the number of events that qualify for a given query. The SM keeps track of which queries are executing at any time, which files are cached on behalf of each query, and which files still need to be cached. It is also responsible for interfacing to the mass storage system (HPSS) to perform all the actions of caching files to disk and removing files from the disk cache.

Next, we focus on the formulation of the cost functions.

3. The optimal file size problem

A central aspect in the formulation of the optimal file size problem is an understanding of the behavior of the robotic tape storage system, and the mass storage system (MSS) that manages the tape system. The MSS we have been working with is HPSS [1]. In our laboratory the robotic tape system managed by HPSS is the IBM 3590 tape system, and thus experimental figures obtained refer to this configuration. HPSS is a general purpose storage system designed to provide an order of magnitude performance improvements over previous MSS software. The HPSS characteristics and behavior relevant for the problem reported here are also common to many other storage systems, and therefore the analysis we discuss here also applies to them as well.

A tape system is usually characterized by several parameters. The ones of interest to us are: the read speed r (in MB/s), the seek speed s (in MB/s), and the tape size L (in MB). In addition, we are interested in the overhead of reading multiple files instead of a single larger file. This parameter, o, which we call the "file splitting overhead" (measured in seconds) is determined by both the characteristics of the robotic tape device, and the MSS software implementation choice. We describe below more precisely the file splitting overhead, and the reason it is an important parameter for our analysis.

3.1. The file splitting overhead

Before deciding on file size, a linear order of all the "chunks" of data to be put on tapes is determined. In the case of the HENP application this is a linear order of all the events (as mentioned above, the data for each event is about 1-10 MB). This linear order can be simply the order of the data as it is generated, or the result of a preferred ordering that provides better clustering for the anticipated queries. This preferred ordering problem is not the topic of this paper. It is assumed that the order has been predetermined.

Given some linear order of the chunks of data (events), a query over this linear order selects some subset of the events, as shown in Figure 2. As can be seen, the shaded chunks are the ones selected. The chunks of data are indivisible: a query either needs the entire chunk or none of it. The query therefore spans some section of the linear order of chunks. We refer to that as the "query span". It is obvious that it is better that a query span has a high percentage

Figure 2. A query spans some section of the linear order of chunks even though only a subset of them (the shaded chunks) were selected.

of chunks selected. This is achieved if there is a good match between queries and the organization of chunks in the linear order. However, as will be seen below, our analysis of the file size is independent of the degree of matching. The degree of matching only affects the parameters that characterize the query distribution.

To illustrate the concept and importance of the file splitting overhead, consider the simple example shown in Figure 3. Suppose that we have 3 queries: query 1 spans 300 MBs, query 2 partially overlaps query 1 and spans 100 MBs, and query 3 partially overlaps query 1 and spans 200 MBs. We also assume that these queries apply to a section of the tape located 10 GBs from the beginning of the tape (about 1/2 way for reasonably large tapes). If we do not allow duplication of data, then we have two options here. The first option, which we call the "no split" option, is to have a single file F1 of size 300 MBs. The second option, which we call the "split" option, is to split this file into two files F2 and F3 of 100 MBs and 200 MBs respectively. In the second option, to satisfy query 1, it is necessary to read F2 and F3. The extra time necessary to read the two files instead of reading F1 is the "file splitting overhead" (FSO). This time depends on the policy used by the MSS. For example, the policy may be to rewind to the beginning of a tape after each file is read. Alternatively, the policy may be to leave the read head at the end of the file, and wait to see what the next command is (which may be "read another file from this tape" or dismount this tape). Accordingly, the FSO may vary from system to system. We have measured the FSO as well as other characteristics for the three real systems shown in Table 1.

The measurements shown in Table 1 were performed on an Exabyte Carousel Tape System running under Unitree (an MSS still in use that precedes HPSS), an IBM 3590 tape system running under HPSS, and an AMPEX D2 Tape Library running under Unitree. The table shows measurements for the mount time, tape size, transfer rate (read speed), seek speed (this is the maximum seek speed achieved after initial ramp up), and the file splitting overhead. Note that the file splitting overhead is measured in seconds, but it can also be expressed in megabytes (the equivalence is made by the transfer rate). The measurements also showed that the file splitting overhead did not depend on where a file was located on the tape. This can be explained by fact that all tape systems come to a stop at the end of reading one file before setting to read the next file.

We cannot account for the effect of the MSS on these figures as the measurements were performed treating the system as a "black box". However, we know that the FSO is affected by the MSS policy as discussed above. In the IBM system, HPSS stops at the end of a file, and does not rewind the tape until it is dismounted. In the Ampex system, the Unitree system rewinds to a nearby "station" before it reads the next file. This accounts for the larger FSO on the Ampex. We speculate that the low FSO on the Exabyte is due to the slow read speed, so the system can start reading the next file very quickly. Regardless of the reasons, given a particular MSS and robotic system, we show in this paper that the FSO plays a major role in determining the file size in applications where multiple files (or part of files) are requested by a single query.

We note that newer technology uses "serpentine" layout of files on a tape. For example the IBM 3590 tape system uses 8 tracks. When writing files, if it reaches the end of one track it continues to write in the next track backwards, etc. in a serpentine fashion. This not only increases the capacity of the tape, but also reduces seek time on the average to 1/2 of a track. For example, if the tape can hold 24 GB and has 8 tracks, the average seek is only 1.5 GB. In our analysis, this effect only changes the parameter used for the average seek distance.

Another effect that cannot be predicted is the hardware compression of files. Some systems (the IBM 3590 included) apply a compression to files before they are stored on tape. The compression rate is data dependent - a file with a large number of repeating data, such as lots of zeros, is highly compressed. We did not take into account this effect in our analysis, assuming that all the files are compressed to an equal degree.

3.2. The effect of partial file reads

Another consideration is whether "partial file reads" are supported by the MSS. As mentioned above, a partial file read means that when a portion of the file is needed, the system opens a file, and moves at read speed until the needed portion is reached, then reads the requested portion and stops. If partial file reads are not supported, then files have to be read in their entirety when any part of the file is requested. We will consider now the effect of partial file reads.

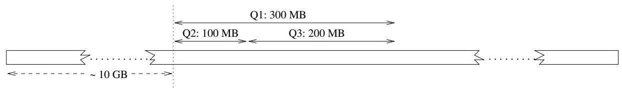

Figure 3. An example of file splitting overhead. The preferred solution depends on the distribution of queries.

Measured Characteristics	Exabyte	IBM	Ampex
Mount time (sec)	315	47	39
Tape size (GB)	5	24.5	25
Transfer rate (MB/s)	0.265	4	12.9
Seek speed (MB/s)	31.25	52	503
File Split Overhead (sec)	0.25	3	11
File Split Overhead (MB)	0.064	12	141

Table 1. Comparison between different storage systems.

We compare below the cost of executing the queries shown in Figure 3, for the two options discussed previously, i.e. the "no split" and "split" options. We consider these for two cases: the availability or lack of partial file reads. We assume read speed of 5 MB/s, seek speed of 500 MB/s, and file splitting overhead of 30 seconds. Note that to keep the example simple, we only consider the case of a single file split. In general, a query can span many file splits, and incur the extra cost of multiple file split overheads. Since in this example we have only a single split, we needed to choose a relatively high file splitting overhead of 30 seconds (measured values were up to 11 seconds), in order to illustrate its effect.

Option "no split", no partial file reads

- read Q1 = seek (10GB) + read (300 MB) = 80 sec
- read Q2 = seek (10GB) + read (300 MB) = 80 sec
- read Q3 = seek (10GB) + read (300 MB) = 80 sec

(Thus: average = 80 sec)

Option "split", no partial file reads

- read Q1 = seek (10GB) + read (100 MB + 200 MB) + FSO = 110 sec
- read Q2 = seek (10GB) + read (100 MB) = 40 sec
- read Q3 = seek (10.1GB) + read (200 MB) = 60.2 sec

(Thus: average = 70 sec)

As can be seen, if all queries have the same frequency, then the average time for option 1 is 80 seconds and for option 2 about 70 seconds, making option 2 the preferred one. This is because the extra read times dominate in option 1, even though the FSO is fairly large. However, in reality queries can span multiple files, and the file splitting overhead can add up. To see this suppose that query 1 is 5 times as likely to be issued as queries 2 and 3. In this case the average is still 80 seconds for option 1, but it is about 93 seconds for option 2 $((5 \times 110 + 40 + 60)/7)$. Thus, option 1 is preferable.

Finally, we consider the effect of having partial file reads.

Option "no split", with partial file reads

- read Q1 = seek (10GB) + read (300 MB) = 80 sec
- read Q2 = seek (10GB) + read (100 MB) = 40 sec
- read Q3 = seek (10GB) + read (300 MB) = 80 sec

(Thus: average = 67 sec)
Note that for query 3, we still need to read the entire file, because the desired portion is at the end of the file.

Option "split", with partial file reads

- This option is exactly the same as the one with no partial file reads because the split files are read fully.

(Thus: average = 70 sec)

In this case we get the reverse result. The average time for option 1 is about 67 seconds and for option 2 still about 70 seconds, making option 1 the preferred one. Thus, partial file reads mitigate the effect of the extra reads necessary. In practice, the ability to perform partial file reads permits larger file sizes than without this capability. Larger file sizes are generally preferable because the MSS system needs to manage less files in its directories.

In this paper we develop the cost functions only for the case of no partial file reads. Indeed, in many systems partial file reads are not available. However, as can be seen from the above example and discussion, the effect of having partial file reads permits larger file size. The precise analysis of this case will be considered in future work.

We now proceed with the formal investigation of deriving the optimal file size given various query distributions.

3.3. Cost function and optimal file size

From the discussion in the previous section we see that there are two opposing requirements: on the one hand we want small files (or many files per tape) to avoid reading or caching unnecessary data. On the other hand we want few files per tape (or large files) to avoid paying the file splitting overhead too many times.

For our analysis, we assume that the following 2 parameters are known: the tape length L and the file splitting overhead o. Although the file splitting overhead is not usually provided by vendors, it can be easily measured experimentally. For our purposes both quantities will be measured in seconds. We also make use of a third parameter, the read speed (transfer rate), to express the tape length in seconds. In our analysis, we will ignore the seek time, the argument being that it contributes an overall constant, and so doesn't play a role in our analysis. It contributes to the minimum retrieval time but it does not affect the optimal file size. Finally, we assume that we also know the distribution of query sizes.

The distribution of query sizes used in our analysis is determined by the queries processed by the MSS. This may not necessarily be the same distribution as the original queries presented by the user as the system may decide to split large queries into multiple subqueries. For that reason, we assume that a query is satisfied only if all the data requested by it is retrieved and presented to the user in one chunk; that is, if a query can be satisfied one part at a time, then it will be represented as multiple queries, one for each part. Similarly, if a query requests data from t different tapes where $t > 1$, it will be considered as t separate and independent queries as the system cannot guarantee that the required tapes will be mounted without interruption by other users and system events. This means that the maximum query size is equal to the size of a tape. As mentioned previously, we also assume that the data was laid out on tape beforehand and that we have no control over that. The implicit assumption here is that some sort of clustering was done to the data, but the question of whether a better clustering could be found is of no concern to us. Given the above parameters and assumptions, we now proceed to determine the cost function to retrieve the data for a particular class of query distributions.

We restrict this paper to the simple case where the distribution of query location is uniform; that is, for a given query size the query is equally likely to fall anywhere on the tape. More general situations, where some parts of the data are more actively retrieved than others, is left for future work. Because of this uniform distribution of the query positions, and our treatment (explained below) of seek time as an overall constant, all the files have the same size and the optimization problem has only one free parameter, n, the number of files. This follows from symmetry arguments: if all the tape locations are equivalent from the "queries" perspective, then the files should have the same size everywhere. In reality, this approach ignores the "edge effects" as it can be shown that queries of size q will hit data at a distance of less than q from the edges of the tape less frequently then data anywhere else on the tape. However, if the average query size is much smaller than the tape size the error resulting from this simplification will be very small, and treating the files as if all of them have the same size is a very good approximation.

As mentioned before, both the tape length L and the file splitting overhead o will be measured in seconds. The value of L is obtained by dividing the tape capacity (in MB, say) by the read speed (in MB/sec). In general, o is independent of the file size S.

Since there is a finite number of "chunks" per tape the number of possible different queries is also finite and consequently the query size will take discrete values. However for analysis purposes we assume that the query size is a continuous quantity. This approximation is justified if a file of optimal size consists of a large number of chunks. As will be shown, this is indeed the case in our analysis.

Let q be the query size, a continuous quantity in the range $(0, L]$. In our analysis we assume no partial file reads. Hence, if a query needs some part of a file the whole file has to be retrieved. Since there are n files there are also n different query ranges ($k = 0, 1, \ldots, n-1$):

$0 < q \leq \frac{L}{n}$:

$$\frac{L}{n}\left(\frac{L}{n} - q\right) n + \left(2\frac{L}{n} + o\right) q(n-1)$$

$\frac{L}{n} < q \leq 2\frac{L}{n}$:

$$\left(2\frac{L}{n} + o\right)\left(2\frac{L}{n} - q\right)(n-1) + \left(3\frac{L}{n} + 2o\right)\left(q - \frac{L}{n}\right)(n-2)$$

$$\vdots$$

$k\frac{L}{n} < q \leq (k+1)\frac{L}{n}$:

$$\left((k+1)\frac{L}{n} + ko\right)\left((k+1)\frac{L}{n} - q\right)(n-k) + \left((k+2)\frac{L}{n} + (k+1)o\right)\left(q - k\frac{L}{n}\right)(n-k-1)$$

$$\vdots$$

For each possible query range we indicate the corresponding average time to execute such queries. To understand how we compute the average execution time let's look at the first case, *i.e.*, when $0 < q \leq L/n$. Then, as shown in Figure 4, two things can happen: (a) either the query falls inside just one file (of size L/n) or, (b) it falls across two files in which case it also incurs the file splitting overhead penalty. In order to compute the probability of (a) and (b) we will first find the weight of each event in terms of the possible locations for the left endpoint of the query. A query of size q will fall inside a given file if its left endpoint falls within a distance of $L/n - q$ from the beginning of the file. The total weight of (a) is therefore $(L/n - q)n$, where the factor n is due to the fact that there are n such files. Similarly, in case (b) where a query falls across two files, a query of size q will cross a boundary of a given file if its left endpoint is within a distance of q from the end of that file. The total weight of (b) is therefore $q(n-1)$ where the $(n-1)$ factor is due to the fact that there are $(n-1)$ pairs of consecutive files in the tape. The associated costs are L/n and $(2L/n + o)$. The last aspect to take into account is a normalization issue: the sum of probabilities for every query range has to add up to 1, *i.e.*, we need to divide the above weights by

$$\left(\frac{L}{n} - q\right)n + q(n-1) = L - q.$$

Although we considered the case $k = 0$ here, this normalization factor is in fact independent of k as can be easily

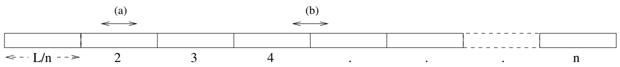

Figure 4. The two possible query patterns when $0 < q \leq L/n$: (a) the query falls inside a file - the cost is L/n; (b) the query falls across a file boundary - the cost is $(2L/n + o)$.

seen.

The average cost of executing a query of size q on a tape of length L and n files and file splitting overhead o is then

$$C(q|L,n,o) = \frac{1}{L-q}\sum_{k=0}^{n-1}\Theta_k(q) \times \times \left[\left((k+1)\frac{L}{n}+ko\right)\left((k+1)\frac{L}{n}-q\right)(n-k) + \left((k+2)\frac{L}{n}+(k+1)o\right)\left(q-k\frac{L}{n}\right)(n-k-1)\right], \quad (1)$$

where

$$\Theta_k(q) = \theta(q-k\frac{L}{n})\theta((k+1)\frac{L}{n}-q),$$

and $\theta(x)$ is the following step function

$$\theta(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}.$$

Now if the distribution of query sizes is also known then we can evaluate the average cost of a query execution easily

$$C(L,n,o) = \int_0^L D(q)C(q|L,n,o)dq, \quad (2)$$

where $D(q)$ is the normalized query density

$$\int_0^L D(q)dq = 1.$$

The optimal file size (or optimal number of files per tape, $\hat{n}$) is obtained from

$$\left.\frac{\partial C(L,n,o)}{\partial n}\right|_{\hat{n}} = 0.$$

To the above average query execution cost we should add a seek cost, *i*.e., an extra cost to seek to the beginning of the first file touched by the query. Since the read head can be anywhere on tape when the query is issued, it is reasonable to assume that the seek cost is the same on average because the queries fall anywhere on tape with the same likelihood. Hence, seek costs are an overall constant (excluding possible small edge effects) that can be neglected for our purposes.

4. File size determination

As examples of the usefulness of the formula derived in the previous section we consider now two cases. These two cases, although simple enough to allow an exact solution, at least of the cost function, capture nevertheless the relevant behavior that may be displayed by more complex cases. The first one, where the query density function is a delta function, can be solved exactly in the limiting case where o vanishes and its simple final form serves as a check of the validity of the above formula. The other example, a Zipf like query density function, is more realistic. Although it can also be solved exactly in the same limiting case, the analytical solution is unnecessarily complicated, and numerical methods turn up to be preferable, with solutions easier to grasp.

4.1. A delta query size distribution

As a first example we consider the case where the query has a fixed size

$$D(q) = \delta(q - \frac{L}{N}),$$

where $N \geq 1$. This just means that all queries have size L/N (less than L since $N \geq 1$, with N an arbitrary number not necessarily an integer). Then the average query execution time is

$$C(L,n,o) = C(\frac{L}{N}|L,n,o),$$

and from (1) we see that only one term contributes, the one that satisfies

$$\Theta_k(\frac{L}{N}) = \theta(\frac{L}{N}-k\frac{L}{n})\theta((k+1)\frac{L}{n}-\frac{L}{N}) \neq 0.$$

This happens for

$$\frac{n}{N} - 1 < k < \frac{n}{N},$$

with slightly different conditions if n/N happens to be integer. This has solution $[n/N]$ where $[x]$ represents the largest integer not greater than x. The average query execution time can then be written as

$$C(L,n,o) = \frac{1}{L-\frac{L}{N}}\left[\left(\left(\left[\frac{n}{N}\right]+1\right)\frac{L}{n}+\left[\frac{n}{N}\right]o\right) \times \times \left(\left(\left[\frac{n}{N}\right]+1\right)\frac{L}{n}-\frac{L}{N}\right)\left(n-\left[\frac{n}{N}\right]\right) + \left(\left(\left[\frac{n}{N}\right]+2\right)\frac{L}{n}+\left(\left[\frac{n}{N}\right]+1\right)o\right) \times \times \left(\frac{L}{N}-\left[\frac{n}{N}\right]\frac{L}{n}\right)\left(n-\left[\frac{n}{N}\right]-1\right)\right]. \quad (3)$$

The goal now is to find the n that minimizes this quantity. Although one can in principle write a closed form solution, it would certainly be very complicated and not very illuminating. So we will resort instead to some numerical examples.

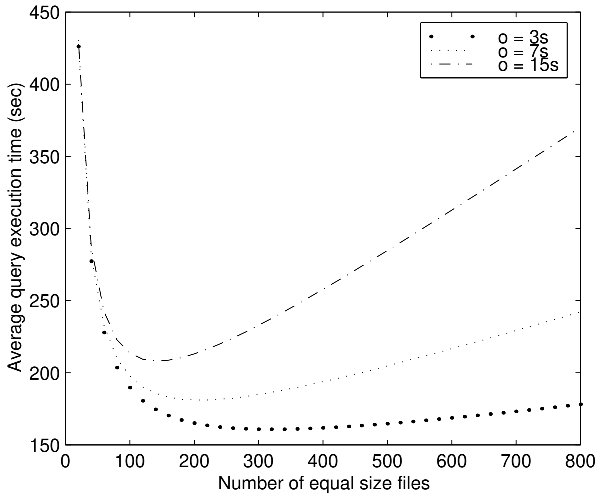

Figure 5. Query execution time as a function of n for three different file splitting overheads, $o = 3, 7, 15$. Here the query size is fixed and equal to $L/N = 122.5$.

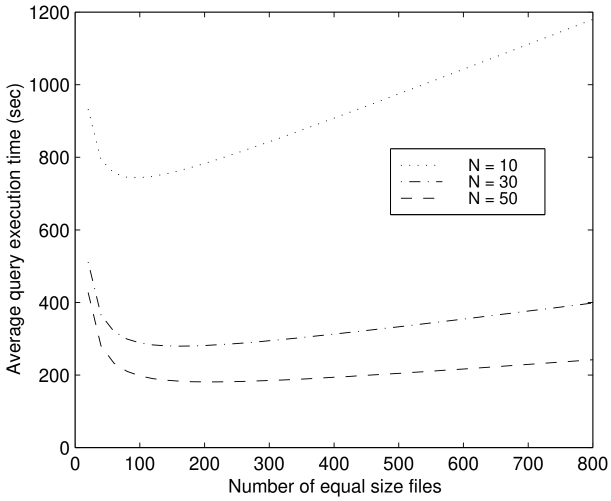

Figure 6. Query execution time as a function of n for three different "delta" query sizes, L/N, $N = 10, 30, 50$. Here the file splitting overhead is fixed and equal to $o = 7$.

Figure 5 shows the results obtained for the average query execution time as a function of the number of files on tape. The values used were $L = 6125$ seconds, $N = 50$ and three different values were used for the file splitting overhead, $o = 3, 7, 15$ seconds. The choice of values was suggested by the experimental tests we performed in HPSS with the IBM 3590 tape library. We found out that the tapes in the system we tested can store around 24500 MBs, the retrieval speed is close to 4 MB/sec, and the file splitting overhead was measured as 3 seconds. The value of N was chosen so that the size of the fixed size query (L/N) was $6125/50$ (or about 500 MBs), a value that seemed reasonable to try for illustration purposes. In Figure 6 we show the results obtained for fixed $o = 7$ and three different values of L/N, $N = 10, 30, 50$. As we can see (Figure 5), the optimal number of files $\hat{n}$ increases as the file splitting overhead o decreases. This is an expected result, and in fact we can prove that $\hat{n}$ becomes infinite in the limit where o vanishes.

From (3) it is easy to show than when $o = 0$,

$$C(L, n, 0) < \frac{L^2}{L - \frac{L}{N}} \left[\frac{1}{N} \left(1 - \frac{1}{N} \right) + \frac{1}{n} + \frac{1}{Nn} + \frac{1}{n^2} \right]$$

and the minimum is obtained in the limit $n \to \infty$ and is $C(L, \infty, 0) = L/N$, which is just the query size. Of course, L/N is always a lower bound of the cost function for any number of files. This expected result serves then as a check of the above formula.

Another point worth mentioning about Figure 5 is the fact that the larger the file splitting overhead o, the more important it becomes in determining the optimal file size.

4.2. A Zipf like query size distribution

As a second example we consider the Zipf [10] like distribution

$$D(q, Q, \alpha) = A \frac{1}{\alpha Q + q}, \tag{4}$$

where α is a small number, Q is largest value that q can take, which we assume as L, and A is a normalization constant,

$$A = \left(\int_0^L \frac{1}{\alpha L + q} dq \right)^{-1} = \frac{1}{\log\left(1 + \frac{1}{\alpha}\right)}. \tag{5}$$

The offset αL is introduced to avoid singularities at $q = 0$. Below we provide the exact closed form solution for the cost function followed by a few graphs that show the behavior of this function.

We recall (equation (1)) that the average cost of executing a query of size q on a tape of length L and n files and file splitting overhead o is

$$C(q|L, n, o) = \frac{1}{L - q} \sum_{k=0}^{n-1} \Theta_k(q) \times$$
$$\times \left[\left((k+1)\frac{L}{n} + ko \right) \left((k+1)\frac{L}{n} - q \right) (n - k) \right.$$
$$\left. + \left((k+2)\frac{L}{n} + (k+1)o \right) \left(q - k\frac{L}{n} \right) (n - k - 1) \right],$$

and the average cost of a query execution is then

$$C_\alpha(L, n, o) = \int_0^L D(q, L, \alpha) C(q|L, n, o) dq.$$

The integrals involved are straightforward. Now define $F_1(L,\alpha,k,n)$ and $F_2(L,\alpha,k,n)$ as

$$F_1(L,\alpha,k,n) \equiv \int_0^L \frac{1}{L-q} D(q,L,\alpha)\Theta_k(q)dq$$
$$= A\int_0^L \frac{1}{L-q}\frac{1}{\alpha L+q}\Theta_k(q)dq$$
$$= \frac{A}{(\alpha+1)L} \times$$
$$\times \left[\log\left(\frac{\alpha L+(k+1)\frac{L}{n}}{\alpha L+k\frac{L}{n}}\right)+\log\left(\frac{L-k\frac{L}{n}}{L-(k+1)\frac{L}{n}}\right)\right],$$

and

$$F_2(L,\alpha,k,n) \equiv \int_0^L \frac{q}{L-q} D(q,L,\alpha)\Theta_k(q)dq$$
$$= A\int_0^L \frac{q}{L-q}\frac{1}{\alpha L+q}\Theta_k(q)dq$$
$$= \frac{A}{(\alpha+1)} \times$$
$$\times \left[\alpha\log\left(\frac{\alpha L+k\frac{L}{n}}{\alpha L+(k+1)\frac{L}{n}}\right)+\log\left(\frac{L-k\frac{L}{n}}{L-(k+1)\frac{L}{n}}\right)\right].$$

Hence,

$$C_\alpha(L,n,o) = \sum_{k=0}^{n-2}\Bigg[\left((k+1)\frac{L}{n}+ko\right)(n-k)\times$$
$$\times\left((k+1)\frac{L}{n}F_1(L,\alpha,k,n)-F_2(L,\alpha,k,n)\right)$$
$$+\left((k+2)\frac{L}{n}+(k+1)o\right)(n-k-1)\times$$
$$\times\left(F_2(L,\alpha,k,n)-k\frac{L}{n}F_1(L,\alpha,k,n)\right)\Bigg]$$
$$+A\log\left(\frac{1+\alpha}{\alpha+\frac{n-1}{n}}\right)(L+(n-1)o).$$

The optimal file size (or optimal number of files $\hat{n}$ per tape) is obtained by finding the n that minimizes the above expression.

As we can see from Figures 7 and 8, the choice of an optimal file size is not very relevant if the file splitting overhead is small and the query size distribution is skewed to represent smaller queries.

Comparing the graph of Figure 5 with that of Figure 8 we see that in the latter one (Figure 8) the number of files per tape is smaller and hence the optimal file size is larger. This result can be verified by looking at the average query size of the Zipf like distribution with $\alpha = 0.001$. In fact a simple integration gives

$$\bar{q} = \int_0^L qD(q,L,\alpha)dq = A\int_0^L \frac{q}{\alpha L+q}dq$$

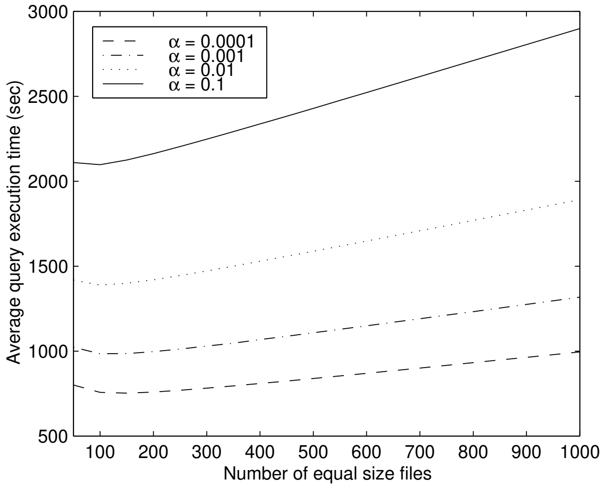

Figure 7. Average query execution times as a function of the number of files per tape for different values of α. A value of $o = 3$ was used.

$$= L\frac{1-\alpha\log\left(1+\frac{1}{\alpha}\right)}{\log\left(1+\frac{1}{\alpha}\right)},$$

where $\bar{q}$ stands for the average of q. Substituting now for the values $L = 6125$ and $\alpha = 0.001$ gives $\bar{q} \simeq 880.43$, a value significantly larger than the value of $L/N = 122.5$ considered in Figure 5.

Looking at the graphs for the cost functions of the two distributions considered here, we see that they are very similar. In fact the same would happen with any other distribution. This can be explained by the fact that the overall shape of the distribution is not as relevant as the distribution's first moment (average query size). This follows from the fact that there is only one parameter in the problem (the file size, which from our assumptions is independent of the tape position).

5. Conclusion

In this paper we showed how to find the optimal file size for a system of tapes if the distribution of query sizes is known. The two main assumptions were 1) The queries fall anywhere on tape (or system of tapes) with the same probability, *i.e*, all parts of the data are equally active; and, 2) All queries span a single range of the tape. By optimal file size we mean the size that minimizes average retrieval times.

We found, as expected, that the optimal file size is determined by the value of the file splitting overhead and the shape of the query size distribution (or in more practical

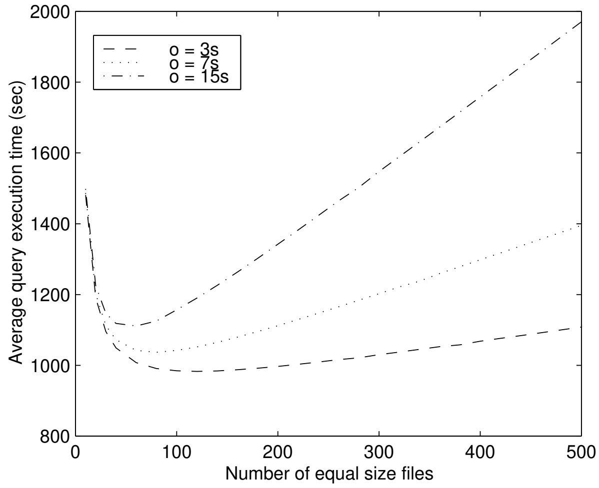

Figure 8. Average query execution times as a function of the number of files per tape for different values of o. A value of $\alpha = 0.001$ was used.

terms, by the average query size). Also as expected, smaller file splitting overheads and distributions skewed to smaller query sizes (small average query sizes) were shown to lead to faster response time.

We implicitly assumed the average query size to be much smaller than the tape length when we considered that all files would have the same size. This requires that some "good" clustering was done to the data before laying it out on tape, so that similar data points ("events") are physically close to each other on tape. Only then can we assume that most of the queries will be much smaller than the length of a tape. If this assumption doesn't hold and every query touches chunks at random the method described here cannot be applied. However one can show that in those situations the "optimal solution" would be to have as many files as chunks (assuming file splitting overheads of a few seconds, as the values we measured). Timewise, this would be far from optimal though and the need for some kind of clustering would become very clear.

We didn't discuss queries that consist of two or more disjoint ranges. However it is clear how to extend the analysis to those cases. A query consisting of say, j disjoint ranges, can be considered as j differents queries instead. The only extra thing to take into account then is the seek or mounting time between those j queries. This may require a deeper knowledge of the behavior of the storage system though and will be left for future work.

Finally, the precision of all this rests on our capability to correctly estimate or measure the file splitting overhead as well as the distribution of query sizes. While the measurements of file splitting overheads are relatively easy to perform, the distribution of query sizes may require more involved procedures such as query monitors that collect statistics about query sizes and frequencies or simulations based on some knowledge of the access patterns used in specific application domains.

References

[1] *HPSS User's Guide, High Performance Storage System*, Release 3.0, June 1996. See also the HPSS homepage at the San Diego Supercomputer Center (`www.sdsc.edu/hpss/`) and the homepage of the Cornell Theory Center (`www.tc.cornell.edu/UserDoc/HPSS/`).

[2] L. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. *Information Systems Journal*, 20(2):155–83, April 1995.

[3] L. Chen and D. Rotem. Optimizing storage of objects on mass storage systems with robotic devices. In *Advances in Database Technology - EDBT '94. 4th International Conference on Extending Database Technology*, pages 273–286, Cambridge, UK, 28–31 March 1994. M. Jarke, J. Bubenko, and K. Jeffrey, editors, Springer-Verlag.

[4] D. De Witt and J. Yu. Query pre-execution and batching in paradise: A two-pronged approach to the efficient processing of queries in tape-resident data sets. In *Ninth International Conference on Scientific and Statistical Database Management*, Olympia, Washington, USA, August 1997.

[5] S. Ranade, editor. *Jukebox and Robotic Libraries for Computer Mass Storage*. Meckler, London, 1992.

[6] S. Sarawagi. Database systems for efficient access to tertiary memory. In *Fourteenth IEEE Symposium on Mass Storage Systems. Storage - At the Forefront of Information Infrastructures*, pages 120–6, Monterey, CA, USA, 11–14 September 1995. IEEE Comput. Soc. Press.

[7] S. Sarawagi. Query processing in tertiary memory databases. In *21st International Conference on Very Large Data Bases*, pages 585–96, Zurich, 11–15 September 1995. U. Dayal, P. Gray, and S. Nishio, editors, Morgan Kaufmann.

[8] S. Sarawagi. Execution reordering for tertiary memory access. *Data Engineering Bulletin*, 20(3):46–54, 1997.

[9] S. Sarawagi and M. Stonebraker. Reordering query execution in tertiary memory databases. In *22nd International Conference on Very Large Data Bases*, pages 156–167, Mumbai (Bombay), India, September 3–6 1996. T. M. Vijayaraman et al., editors, Morgan Kaufmann.

[10] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison Wesley Press, Inc., Cambridge, 1949.