Storage Management for High Energy Physics Applications

A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim
National Energy Research Scientific Computing Division
Lawrence Berkeley National Laboratory

Abstract

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, and high energy physics experiments. 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 that are managed by some mass storage system (MSS). A major bottleneck in analyzing the simulated/collected data is the retrieval of subsets from the tertiary storage system. This bottleneck results from the fact that the requested subsets are spread over many tape volumes, because the data are stored as files on tapes according to a predetermined order, usually according to the order they are generated. In this paper we describe the architecture and implementation of a Storage Manager designed to support the ordering of the data on tapes to optimize access patterns to the data. We also describe additional optimization opportunities to improve access time. The system is being built for a High Energy Physics experiment scheduled to go on-line in a year.

1. Introduction

Many applications (usually in scientific domains) generate large volumes of data that have to be stored on tertiary storage (typically robotic tape systems). Some examples are large scale simulations for climate modeling, combustion modeling, high energy physics experiments, and satellite data. In such applications, one of the major bottlenecks in analyzing the simulated/collected data is the retrieval of subsets from the tertiary storage system. This bottleneck results from the fact that the requested subsets are spread over many tape volumes, because the data are stored as files on tapes according to a predetermined order, usually according to the order they are generated. In this paper we describe the architecture of a Storage Manager designed to support the ordering of the data on tapes to optimize access patterns to the data. The storage manager takes advantage of the properties of the stored data, such as time and spatial location of satellite data, or the properties of particle collisions in high energy physics experiments. We describe in this paper a system being build for a high energy physics experiment scheduled to go on-line in a year, called STAR (http://www.rhic.bnl.gov/STAR). While we concentrate in this paper on the high energy physics application, the principles developed for this system apply to other application areas. As an example, we applied the same principles in the development of a prototype for Climate Modeling applications, and showed 10-50 factor access time improvement for a typical query mix [1,2]. We describe next in some detail the High Energy Physics application, and the properties of the data.

2. Brief description of 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 and "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 event collisions 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 are collected, they undergo an "event reconstruction phase". Each event is analyzed to determine its sub-particles and summary properties for each event are derived (such as the total energy of the event, momentum, and 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.

The access of such subsets from these tape systems is slow because of the inherent sequential nature of the devices (requiring long seek times to reach the desired events), 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. The approach is to organize the event data on tape storage in a way most appropriate to the probable access patterns of the data rather than the order in which they were generated. One aspect of our approach is to order the events on tapes according to their properties, so as to minimize the number of tapes that need to be mounted and the number of tape seeks made when subsets of the events are needed. Another aspect is to coordinate file caching and purging in order to maximize their sharing by multiple queries.

3. Optimization opportunities

3.1 Multidimensional clustering

As mentioned above, the main concept in optimizing access of data from tapes is clustering the data according to their expected retrieval. In the case of HENP data, one can express this problem as follows. Given a large set of objects (in this case physics "events"), each with multiple properties (such as "energy", "transverse momentum", and "number of muons"), we can view each object as a point in the multi-dimensional property space. A typical query is a "range query" in that space (for example, get all events with energy between 2-5 GeV's, and more that 10 muons). Such queries therefore, describe a localized subspace in the multi-dimensional property space. Therefore, it makes sense to organize the data according to multi-dimensional cell structure. Thus, our approach is to organize the space into cells by partitioning each dimension into bins. We then store cells that belong to the same multi-dimensional cluster close to each other on physical storage. This is handled by one of the main components of the system, called the "clustering analyzer", which is described in the next section.

We note that considering object clustering in the multi-dimensional space of their properties is applicable in many areas (including non-scientific applications). For example, one can characterize satellite images according to a set of properties, such as color, shape, time, and position. Similarly, sales transactions can be characterized by product type, location, time, etc. In the multi-dimensional property space each object (image, sales transaction) is a single point. A cluster in that space represents a phenomenon that might be of interest. If we organize the data according to spatial locality in that space, the request for data (e.g., images) that represent a phenomenon will tend to get data that are clustered physically.

3.2 Multi-query overlap

Another opportunity for access time improvement is to have users share files that were cached to disk whenever possible. Given a query, we use an index according to the properties associated with each object (event), in order to determine which events qualify for the query and which files they are stored in. Thus, given a query we can determine the files that we need to cache. If any of these files are in cache because another query is using them, we can immediately make these files available to the new query. This strategy pays off well if the set of queries overlap the multi-dimensional regions, which is the case if several investigators are working on similar phenomena. Indeed, it is the practice in the HENP community of users that predefined subsets are extracted, and if they are small enough, they are stored on disk cache for the long term. However, this practice has its limitations in that only a relatively small number of such subsets can be cached, and that the physicists cannot ask for ad-hoc subsets to be retrieved.

3.3 File caching order

Given a query that has objects (events) in multiple files, it may be useful to have some policy that determined which files to cache first. In the case of physics data, each "event" is independent of another, and therefore analysis can proceed as soon as one file is cached. In practice, one can determine if the subset is of interest from analysis of some fraction of the data. If it is determined that this subset is of no interest, the query can be aborted. This implies that for such cases, it is better to cache first the files that have the largest number of objects that qualify for the query. Another consideration for file caching order is to prefer files that have the larger number of queries waiting for them, or files that are smaller in size. The ability to analyze a set of queries ahead of time presents such opportunities, and a caching policy can be customized for the mix of queries.

3.4 Selecting file size

In general, if the file size is large relative to the query size it will lead to the transfer of large amounts of irrelevant data whereas small file sizes will incur an overhead penalty associated with reading each new file. There are two conflicting considerations in determining the file size. At one extreme, one can choose the file sizes as the smallest chunks resulting from the intersection of the events needed by a typical set of queries. 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. 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 tradeoff between these two overheads is an optimization problem that needs to be considered when deciding the file size for storing multiple objects (events). In a previous paper [3] we analyzed 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. Some systems may permit some form of a “partial file read” where an efficient read of part of the file is supported by fast seek to the desired location in the file. In that case larger files can be used to combine many small files, but the burden of keeping track of the various “chunks” in the file needs to be done outside the system.

3.5 Query estimation

Retrieving large subsets of data from tape-based datasets is very expensive both in terms of computer and system resources, and in the length of time for a query to be satisfied. Therefore, it is quite useful to provide an estimate of the amount of data and the number of files that need to be retrieved for a potential query. Often, users start a query, get frustrated with the length of time to perform the analysis, and abort the query. An important optimization strategy is to prevent such non-productive activities by providing a quick estimate of the size of the returned data, and a time estimate of how long it will take to retrieve the data. As is discussed below we designed and developed such a “query estimator” by taking advantage of the index used to evaluate a query.

4. General System Architecture

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 events to cluster on, and the partitioning of each property into bins (called a binning strategy). The bins define a multidimensional cell structure over the properties. Each cell thus contains the events (points) that have properties define the cell. Cluster analysis consists of finding clusters of cells that are adjacent and the number of points in them is above a certain threshhold. The input to this module is the file that contains the properties of each event (referred to as the n-tuple file). The analyst can specify any subset of the properties, and binning specification for each property. For this selection the CA generates a distribution graph that shows the degree of clustering for this choice (it shows what percent of the events fall into what percent of the cells). Using this tool the analyst can choose the properties

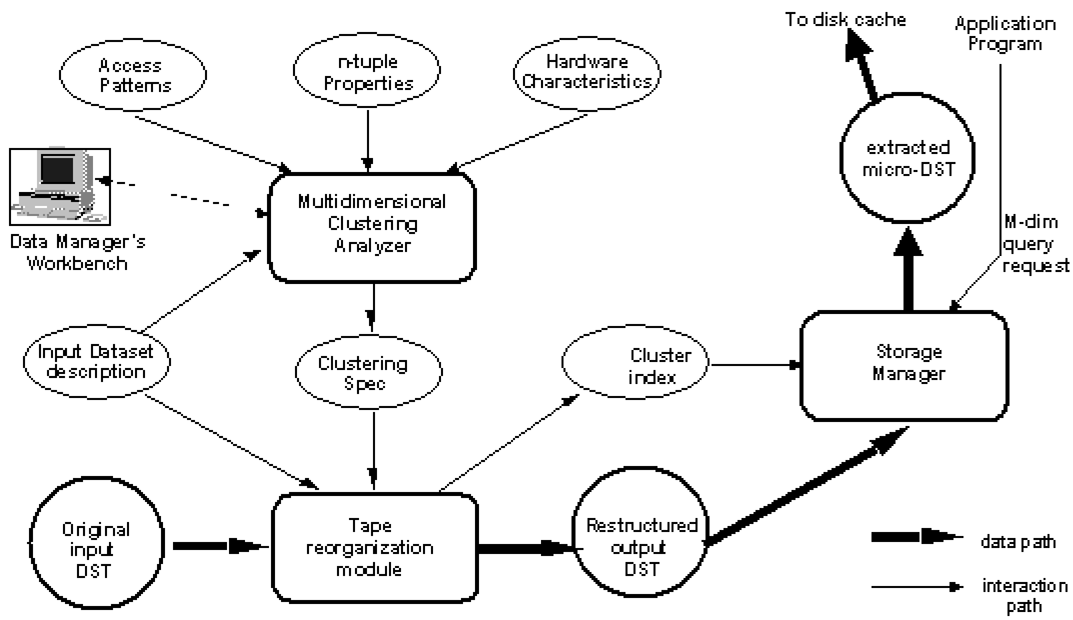

Figure 1: Architecture of system for event reorganization and access

and degree of binning to be used for reordering the events in files stored on tapes. The selection of properties to cluster on and the binning specification is done by physicists using an interface which we call 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 to. 2) The Reorganization Monitor – the component that monitors the reading and writing of tapes by the mass storage system (we use the High Performance Storage System (HPSS) 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 directory that contains the order in which events were clustered into files. The directory contains one entry for each file that points to sections of an event table for the events in that file. The table has one row for each event, and represents the reorganized table of events and their properties that is generated by the event reconstruction process. This directory is used to generate the Cluster Index used by a third main component, the Storage Manager.

The Storage Manager (SM) is responsible for determining, for each query request, which events and files need to be accessed. A specialized index (called a "compressed bitmap" index) is constructed to be used for quick (real-time) estimation of the number of events that qualify given a query. The SM has 3 main components. 1) The Query Estimator, that uses the index to determine what files and what events are needed to satisfy a given range query. 2) The Query Monitor, that keeps track of what queries are executing at any time, what files are cached on behalf of each query, what files are not in use but are still in cache, and what files still need to be cached. The QM consults an additional module, called the caching policy module, that determines what file to cache next according to the policies selected by the system administrator. 3) The Cache Manager, that is responsible for interfacing to the mass storage system (HPSS) to perform all the actions of staging and purging files from the disk cache.

5. Implementation issues

The Clustering Analyzer starts by counting the number of events that fall into each cell, and then uses a near-neighbor algorithm to determine the clusters. Potentially, there can be a very large number of cells (e.g., 100 bins for each dimension in a 4-dimensional space amounts to 100 million cells). However, if the data is clustered in this space most of the cells are empty. Indeed, we found that for sufficiently fine binning of physics events (about 1000 cells or more) 85-95% of the cells are empty. Therefore, we used a hashing table to store only the cells that are not empty while constructing the counts. We found that this saved space and speeded up the analysis program.

We mentioned above that the Query Estimator uses a “bitmap index”. The reason for this choice is that a multi-dimensional index over 100 million elements is very large, and we needed a quick estimation of the results of a query. The bit sliced index is based on the binning of each dimension. For each bin we generate a bit-vector that has a 1 in the position of an event that falls into this bin, and 0 otherwise. It is possible to compress such bit slices, and perform boolean operations over the compressed form. It is beyond the scope of this paper to describe the details of this indexing technique. However, it is worth mentioning that bit-sliced indexes were found to be useful for multi-dimensional indexes, and are employed in data warehouse applications [4].

Another implementation issue we encountered was module interconnections. The Storage Manager has three components. Since it may be advantageous to have each component reside on another machine (e.g., the Cache Manager may be on the same machine that support the Mass Storage System), we chose to use CORBA for interfacing between the modules. A critical issue was that each of these components needs to deal with multiple requests concurrently, it was useful to take advantage of an ORB that supports multi-threading. This limited us with the choice of such products.

Acknowledgement

This project is funded by the Grand Challenge program at the Department of Energy, in 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.

We thank all the collaborators of this project who are developing other components of the system, and who provided the realistic requirements for the storage management component of this project.

References

[1} L.T. 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, Pergammon Press, April 1995, vol. 20, (no. 2): 155-83. (http://www.lbl.gov/~arie/papers/optimass.Info.Sys.ps).

[2] L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani, Optimizing Tertiary Storage Organization and Access for Spatio-Temporal Datasets, NASA Goddard Conference on Mass Storage Systems, March 1995. (http://www.lbl.gov/~arie/papers/optimass.goddard.ps).

[3] L. Bernardo, H. Nordberg, D. Rotem, and A. Shoshani, Determining the Optimal File Size on Tertiary Storage Systems Based on the Distribution of Query Sizes, Tenth International Conference on Scientific and Statistical Database Management, 1998. (http://www.lbl.gov/~arie/papers/file.size.ssdbm.ps).

[4]   P. O’Neil, "Informix Indexing Support for Data Warehouses," Database Programming and Design, v. 10, no. 2, February, 1997, pp. 38-43.