Data Mining Using Light Weight Object Management in Clustered Computing Environments[*]

R. L. Grossman[†], S. Bailey and D. Hanley

Laboratory for Advanced Computing
University of Illinois at Chicago

Magnify, Inc.
Oak Park, Illinois

February, 1996

This is a draft of the following article: Data Mining Using Light Weight Object Management in Clustered Computing Environments, Proceedings of the Seventh International Workshop on Persistent Object Stores, Morgan-Kauffmann, San Mateo, 1997, pages 237-249.

Abstract

In this note, we describe the design, implementation and our initial experience with an object warehouse specifically designed for selecting, computing and filtering very large collections of objects, each of which has a large number of attributes. The object warehouse is built on top of a persistent object

[*]This research was supported in part by DOE grant DE-FG02-92ER25133, and NSF grants IRI-9224605, CDA-9303433, CDA-9413948, and the National Scalable Cluster Project. Please send correspondence to Robert Grossman, Laboratory for Advanced Computing, University of Illinois at Chicago, 851 South Morgan Street, Chicago, IL 60607, USA, 312 413 2176, 312 996 1491 (fax), grossman@uic.edu

[†]The author is also a member of the technical staff at Magnify, Inc. This work was also supported in part by the Massive Digital Data Systems Program through a contract to Magnify, Inc.

manager. We are especially interested in persistent object managers which are monotone, that is designed for data which is read-mostly, occasionally appended, and infrequently updated. These operations and access patterns are common when data mining large data stores, which provides the main motivation for our current work. For object warehouses to prove useful, they must scale as the number of objects increase, as the selectivity of queries increases, and as the computational complexity of queries increases. We show that our implementation scales in each of the dimensions over three orders of magnitudes: from queries taking seconds touching all the attributes on megabytes of data to queries taking hours touching a small fraction of the data on stores approaching one hundred gigabytes.

Keywords: persistent object stores, data mining, data warehouses, scientific computing, numerically intensive queries

1 Introduction

We are interested in the class of scientific and engineering applications which consist of numerically and statistically intensive queries on large amounts of data in which the access pattern consists of frequent reads, occasional appends, and infrequent updates. Examples include searching for new particles in high energy physics data [9], uncovering evidence for global climate change from satellite sensor data [6], searching collections of radar images for military threats [20], looking for similarity patterns in large amounts of time series data [1], and searching for patterns in large amounts of textural data [4].

The problem is difficult due to 1) the large amounts of data, and 2) the numerically and statistically intensive queries. On the other hand, the problem is made simpler due to the access patterns: the full functionality of a database is not needed, since the data is by and large read only, except when additional data is being added.

To be specific, consider a persistent object store containing high energy physics data consisting of objects representing particle collisions, which are called events. Data analysis requires analyzing from several thousand to several million events. Assume that each event has several hundred attributes and is about 0.5 megabytes in size. Some of the attributes are object-valued, such as leptons in the example below, and some are collection-valued. A typical query consists of the following steps:

Select. Look through all events and select those having two leptons such that the charge of the two leptons is equal but of opposite sign.

Compute. For each event select, compute the energy of the event that the leptons contribute.

Filter. Collect those events such that the energy contributed by the leptons is within a specified range.

Aggregate and Analyze. Compute a histogram of the lepton energy of those events passing through the filter.

These types of queries are becoming common in a variety of disciplines. As another example, consider a persistent object store containing laser radar images which are tagged with attributes and a query which *selects* those images in a specified region during the past two hours, *computes* the number of military threats in each image using a feature extraction algorithm, *filters* the results and retains those posing the greatest threats, and *analyzes* the results to provide a current estimate of the location of the threats [20].

Since these types of applications involve *Selecting*, *Computing*, and *Filtering* (SCF) large collections of objects, for lack of a better term, we say that these applications are all concerned with *SCF data mining.*

The purpose of this paper is to describe our experience designing, implementing and operating an object warehouse for SCF-queries built over a persistent object manager. We report on our experiences using the object warehouse with both synthetic and real data. We also identify some design elements which make persistent object stores useful for these types of queries and describe our preliminary work developing SCF benchmarks related to scalability.

We conclude this introduction with a brief summary of the paper. Section 2 contains background material and a description of related work.

Section 3 describes some of the characteristics of persistent object stores relevant to data mining. We call a persistent object store *monotone* (the term is due to Peter Buneman) in the case that it is designed to work with large amounts of read-only data, to which data is occasionally appended, but infrequently updated. We call a persistent object store *optimistic* in the case that it assumes that processes which access the data have information about how the data is physically layed out. Most persistent object stores and databases are optimistic, while most distributed object systems and wide area object systems are not. To further improve performance we exploited *parallelism* and *precompute* and *index* as much of the data as possible. These characteristics lead to the acronym MOPPI which we use to describe *M*onotone, *O*ptimistic object stores supporting *P*arallel selection, computing and filtering of *P*recomputed and *I*ndexed data. We view MOPPI Stores as a type of object warehouse.

The experimental results in this paper use a MOPPI Store we designed and implemented as an object warehouse called PTool. The PTool used for this paper is the third generation in a series of systems sharing the same name. The first PTool (Version 0.4) was designed to work for single networked workstations [12] and was used for applications in aeronautics and high energy physics. The second PTool (Version 0.6) was designed for high performance shared nothing multi-computers, such as the IBM SP-2, and was mainly used for mining high energy physics data [9] and [13]. The current PTool (Version 2.0) is designed for local, campus, and wide area workstation clusters, and has been used for a variety of applications, including high energy physics [10], complex systems [15], and computational fluid dynamics [16]. Section 4 contains a brief description of Version 2 of PTool.

An important goal of this work was to understand issues effecting the scalability of SCF-queries when using object warehouses. The model we used to investigate scalability issues is described in Section 5. In this model we identify size, parallelism, selectivity and complexity as key factors limiting the scalablity of SCF-queries. Section 6 describes our experimental results involving these factors. In this section we report on SCF-queries on tens of Gigabytes of data. Using these results we are currently experimenting on scaling our system to work with hundreds of Gigabytes of data. Section 7 contains a discussion and our conclusions.

Although this work is very preliminary, we feel that it is a challenging and important problem to understand the critical issues effecting the scalability of persistent object stores to the Terabyte range.

We feel that this paper makes two contributions to the field of scientific data mining: 1) We have identified a variant of persistent object stores—MOPPI—which are useful for scientific data mining. 2) We have designed, implemented and used MOPPI stores for several scientific data mining applications demonstrating the practicality and scalability of approach. In particular we have demonstrated numerically intensive queries on synthetic and real data on persistent object stores ranging in size from megabytes to sizes approaching one hundred Gigabytes.

2 Background and Related Work

The focus in this paper is on systems appropriate for selecting, querying, filtering and mining scientific data. The data is complex in that attributes may be elementary data types, object-valued, or collection valued.

Recall that objects are called *persistent* in the case that their existence is orthogonal to the process which creates them. This means that the creating process writes them to disk or other permanent media in a format in which other processes can later access them; otherwise, objects are called *transient*. In other words transient objects are coterminal with the process that create them.

There are a variety of systems in use for working with persistent collections of objects.

Persistent Object Managers. Persistent object managers support persistence for objects and can be viewed as extensions to, or replacements to, file systems. The collections [5] and [2] provide a variety of perspectives on persistent object managers.

Object-oriented Databases. An object-oriented database provides a variety of functionality for persistent objects, such as a transactions, back up and recovery, concurrency control, and a query language. Object-oriented databases arose to manage the data for a variety of complex applications such as computed-aided design CAD, computer-aided software engineering (CASE), scientific databases, and knowledge bases. Object-oriented databases are generally designed to support transactions involving frequent updates on relatively small collections of objects, while object warehouses are designed for numerically intensive queries on relatively large collections of objects.

Object-Relational Databases. Support for user-defined objects and user-defined functions can be added to traditional relational databases. Object-relational databases are extensions to relational databases supporting some of the most important features of object-oriented databases.

Distributed Object Systems. Distributed object systems are designed to manage objects in heterogeneous environments. Examples include OMG's CORBA in the UNIX environment and COM/OLE for Win-32 environments. The central issue for these types of systems is how a heterogeneous system can access an object without necessarily knowing its object id (OID), where it is stored or how it is stored. To support this type of complexity incurs an added overhead.

Wide Area Object Systems. Wide area object systems are designed to manage objects in internet and intranet environments. Sun's Java is an example. Security is an overriding concern for this category since clients can easily access objects from untrusted hosts.

Object file systems. As traditional file systems evolve, their files are beginning to take on some of the attributes of objects, such as supporting attributes, hierarchical structures, and embedded files.

3 MOPPI Stores

Our goal was to design, implement and gain some experience with an object warehouse designed for scientific data mining. The system had five main requirements:

> R1. *Large data sets.* The most important goal was to support very large data sets, ranging in size from tens to thousands of gigabytes, consisting of objects, each of which had a large number of attributes. It was important that our system scale transparently from small stores to very large stores.
>
> R2. *Low overhead access to the data.* Another goal was to provide access to the data with very low overhead. This was essential since most interactions with the database were numerically or statistically intensive queries.
>
> R3. *Support for parallel queries.* A another goal was to support processing, computing and filtering of objects in parallel together with returning, and merging a *fixed object* return type.
>
> R4. *Transparent access to the data.* It was also important that access to the data be transparent in that the application programmer should not need to worry explicitly about transforming the data from one format to another just because the data set was large or involved a large number of attributes. For example, to gain efficiency with distributed object systems one often imports the data in one format and works with the data in another.
>
> R5. *Simple Design.* In addition, as a practical matter, our interest was to simplify the design of the system as much as possible in order to make our goal of scaling, maintaining and using the system to manage Terabytes of data as easy as possible.

By an object warehouse we mean a system designed for numerically and statistically intensive queries on persistent collections of objects. We now describe some of the strategies which are commonly used to improve the performance of these types of queries.

> *Precomputed.* The easiest way to lower the cost of numerically or statistically intensive computations is to precompute as many queries as possible.
>
> *Indexed.* Another easy way to improve performance is to provide a variety of indicies to the objects in the warehouse so that there is as direct access as possible to the underlying data.
>
> *Monotone.* In mathematics, a function is called monotone in case it is non-decreasing (or non-increasing). Following a suggestion of Peter Bunemann, we use the term monotone to refer to persistent object stores in which objects are appended, but not deleted. With monotone object stores, it is easier to cluster the objects in order to improve access times. Without this property, attributes which need to be analyzed together might not be stored together. Clustering data to improve access is easier for monotone object stores. Monotone object stores are also easier to design. There are several variants: For some applications, updates to objects may be accomplished by versioning: the updated object is appended and a link added to the prior version. When necessary, the warehouse as a whole may have to be rebuilt to make room for additional data.
>
> *Optimisitic.* Both object warehouses and object-oriented databases typically store their objects in a format particular to a specific environment. This means that little or no reformatting is required when accessing objects from that environment. On the other hand, there may be a significant cost incurred when accessing the objects from a heterogeneous environment.

The goal of these strategies is to obtain higher performance access to objects in the warehouse. Precomputing involves the trade off of obtaining higher performance by increasing the (one-time) cost required to populate the warehouse. Both precomputing and indexing involve the trade of off of obtaining higher performance by increasing the size of the store, since the derived data which is precomputed must be stored as must the indices.

Storing the data in an optimistic fashion involves the trade off of incurring additional costs when accessing the data from heterogeneous environments.

The acronymn MOPPI is derived from the initial letters of these characteristics: the warehouses are designed to be *M*onotone, are *O*ptimistic about data layout, support *P*arallel access, computing and filtering, and contain *P*recomputed data which is *I*dexed.

4 Design and Implementation

As described more fully in [13], PTool Version 0.6 provides persistence for instances of C++ classes through an application program interface (API) and class libraries. Persistent objects are grouped together into persistent container classes provided by the class libraries. This is one of the standard approaches to provide persistence to C++ objects.

To provide scalability, PTool Version 0.6 organizes physical collections of objects into segments, as is usual, and organizes physical collections of segments into larger units which we call folios. Segments are simply contiguous extents of virtual memory which are moved or mapped between disk and memory. Folios are moved between disks or between disks and tertiary storage devices. Folios are implemented as files to facilitate interfacing PTool with hierarchical and distributed storage systems [11]. In PTool, there are separate managers for objects, segments, and folios. For additional details, see [13]. The current Version 2 of PTool contains an ODMG93 compliant API and organization into object managers, segment managers and folio managers.

In addition, Version 2 is designed to work with local, campus, and wide area clusters of workstations, as well as high performance clusters of workstations. The logical model is of a distributed collection of nodes with one or more PTool processes running on each of the nodes. Nodes may run i/o, compute or query processes. The distinction between these different types of processes is discussed further in the next section. Physical data management and data transport in PTool Version 2 is divided into four layers:

> *Collection Layer.* PTool client processes access collections of objects, supporting application specific storage and access structures.
>
> *Object Layer.* Methods for creating, accessing, and updating objects are here. This is actually the simplest part of PTool and remains largely the same as in Version 0.6.

Container Layer. This layer manages segments and folios on single nodes, collections of nodes in high performance switching fabrics, nodes within a local area networks, and nodes within a wide area networks. Collective i/o is supported in this layer. This is the most complex part of PTool and is new in Version 2.

Transport Layer. This layer abstracts the process of communicating between nodes independent of whether sockets, circuits, message passing, RPCs, or some other protocol is used.

5 Computational Model

In this section, we describe a model for costing SCF-queries and for understanding their scalability. Input to these queries consist of a selection criterion and a filtering criterion. The output consists of a collection of objects or aggregated attributes of objects.

We assume that we have a collection of objects and that each object has one or more attributes. We assume that objects are stored on one or more nodes in a network. We distinguish between three types of nodes: i/o nodes, compute nodes, and query nodes, which we describe below. The model we use is illustrated in Figure 5. This section is based in part on [17].

Select. In this phase, objects satisfying a specified criterion are selected. There are two selectivities which arise in this phase. The first is called the *input-output selectivity* σ_{io}. The objects may have hundreds of attributes and not all of them may be required to determine whether the object satisfies the specified selectivity criterion. The input-output selectivity may be measured using either the ratio m_0/m, where m_0 is the number of the total number m of attributes used in criterion, or by s_0/s, where s_0 is the amount of space in bytes occupied by the attributes used in the criterion and s is the total size in bytes of the object. The second selectivity is called the *selection selectivity* σ_s and denotes the percentage of the total number of objects N in the store satisfying the specified criterion.

Compute. In this phase, derived data is computed for each object. There are two types of derived data computed: The first type simply requires using data from a single object's attributes. The second type may, in addition, require data from other objects. In the latter case the computation would access certain global meta-data.

Filter. In this phase, those objects satisfying a criterion for filtering are retained. The combined cost of the compute and filter phase of a query is denoted κ and has units seconds per object.

Aggregate. In this phase, all the objects satisfying the filter criterion are collected.

Define the *i/o throughput* τ_{io} of a node as the amount of data that can be accessed, which is measured in bytes per second. We have immediately from the definitions above that the cost of the query, which is measured in seconds, is approximately equal to

$$\max \{S \cdot \sigma_{io}/\tau_{io}, N \cdot \sigma_s \cdot \kappa\},$$

where S is the total size of the store, measured in bytes and N is the total number of objects in the store.

The effective size of the store, which is dependent upon the particular query is $S \cdot \sigma_{io}$, where σ_{io} is the i/o-selectivity. The effective number of objects, which is again dependent upon the query, is $N \cdot \sigma_s$, where σ_s is the selection selectivity. The query cost is then a simple function of the effective size of the query and either the i/o-throughput or the cost per query per object. The computation is input-output bound in the case that the first term is dominant and compute-bound in the case that the second term is dominant.

Scalability. We can now return to the primary purpose of this paper. Specifically, this paper is concerned with our experience designing, implementing and working with object warehouses built using persistent object managers which scale over three orders of magnitude as the size, selectivity, and complexity of SCF-queries vary. Specifically:

1. Does the object warehouse scale as the size S increases?

2. Does the object warehouse scale as the complexity κ of the query increases?

3. Does the object warehouse scale as the selectivities σ_{io} and σ_s increase? The *apparent i/o throughput* τ_{io}/σ_{io} provides a measure of this scale-up.

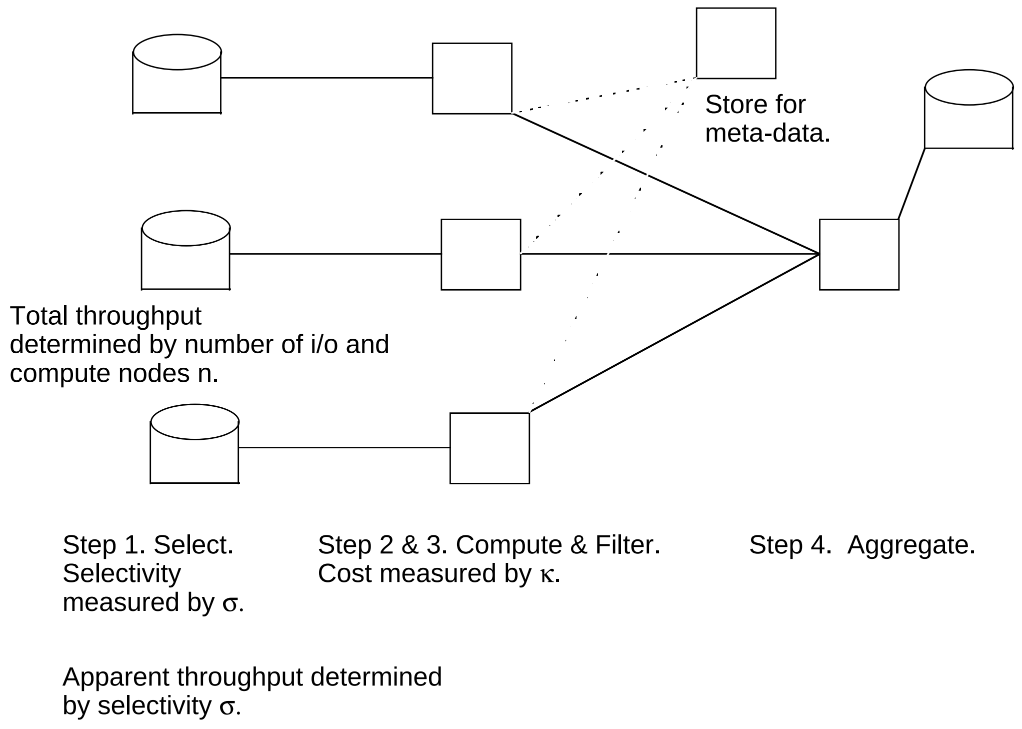

Figure 1: The computational model we use.

4. Does the object warehouse scale as the number of nodes n increases? Ideally, for i/o-bound queries, we expect the *total i/o throughput* to equal $n \cdot \tau_{io}$, where n is the number of i/o-nodes. A similar formula holds for compute-bound queries.

In addition, we are interested in the scalability of the system as the number of attributes of an object increases, as the individual size of an object increases, and as the number of objects in a collection class or store increases. For this preliminary work, we have concentrated on the four measures above.

6 Experimental Results

6.1 Synthetic Data

Experiments were conducted on a cluster of eight workstations connected with Asynchronous Transfer Mode (ATM) technology. Each workstation is a Hewlett-Packard Apollo 9000 Model 725/100 with 128 Megabytes of RAM, and a 9 Gigabyte, external, Fast-Wide, single-ended, SCSI disk. The workstations are connected by a FORE Systems ASX-200 ATM switch utilizing Hardware version 1.0 and Software version ForeThought 3.4.2 (1.3). Each workstation is connected to the switch via a Hewlett-Packard J2802B EISA/ATM adapter card rated at OC-3 (155Mbps) and two multi-mode fibers. The communication protocol was HP implemented TCP/IP over ATM (i.e. Classical IP).

All of our experiments involved a simple Event class consisting of an event header and an event body containing five hundred attributes. Size was varied by using more events. Selectivity was varied by selecting events using more of the attributes. Computational cost was varied by executing a fixed series of floating point operations from one (complexity one) to a thousand times (complexity one thousand). For further details, see [19].

Database Size. In this experiment we varied the database size between 4 and 40 Gigabytes while fixing the selectivity at 5% and the query complexity at 1. Five different database sizes where chosen: 4 Gig, 12 Gig, 20 Gig, 32 Gig, and 40 Gig. Six trials were run for each database size; the mean results of the experiment are summarized in Figure 6.1. This experiment showed that query times increase in a linear fashion as database size progresses from 4 Gigabytes to 40 Gigabytes. This is as expected since the overhead of the PTool management scheme does not increase in proportion to the database size.

Selectivity. In this experiment we varied the selectivity of the query between 0.2% and 100% while fixing the database size at 4 Gigabytes and the query complexity at 1. Seven selectivities were chosen: 0.2%, 1%, 5%, 20%, 50%, 85%, 100%. Six trials were conducted at each selectivity. The mean results of the experiment are summarized in Figure 6.1. The results show an increase in query time as selectivity approaches 100% from 0%.

Further analysis shows a non-linear increase in query time as selectivity progresses from 0% to around 40% and a linear increase as selectivity continues to progress from around 40% to 100%. This dual characteristic of the query time increase is a signature of the impact of PTool's fetching and caching scheme. When selectivity is low, each Event Body that is pulled for computation is relatively far apart from the next. At extremely low selectivities this is not a factor since very few Event Bodies are fetched. However at moderate selectivities (between 5% and 40%), only a relative few selected Event Bodies are present in each PTool segment. This causes many caching misses and creates a quickly increasing overhead. However, at around 40% selectivity, the cache misses decrease as the data selected is closer together and this results in lower order increase in query time.

Complexity. For this experiment, complexity was scaled from 1 to 1000. This represents relative number of floating point operations on each object "selected." Six trials were conducted for each of 12 different complexities; the mean results are indicated in Figure 6.1. As is evident from the graph, the query time scaled linearly to a complexity rating of 1000. It is important to note that the query becomes compute bound (i.e. the query time depends on the computational performance of the machine) almost immediately (somewhere around 100).

Parallelism. To test parallelism, we created 8 i/o-nodes containing 6 gigabytes of data each. The same 8 nodes were also used as compute nodes for the compute and filter portion of the SCF-query. One of the 8 nodes was also used as the query node. The i/o-throughput τ_{io} was approximately 2.3 megabytes per node. We worked with low complexity queries so that the queries were i/o-bound. The total throughput scaled linearly from 1 to 8 nodes, provided a total throughput of approximately 19 megabytes per second over a total store size of 48 gigabytes. One does not expect this scale up to continue indefinitely since the query node which collects the returned objects will eventually create a bottleneck. Unfortunately, our experimental facility did not allow us to measure the total throughput provided by parallel SCF-operations over a greater range.

6.2 Experimental Data

This section summarizes the results from [18]. For this series of experiments we used data from the CDF experiment at Fermi National Accelerator Laboratory. We created 3 PTool stores using four i/o-nodes. Each i/o-node was allocated 2 file systems of 2 Gbytes each. We used about 10.6 Gbytes for these 3 stores. The analysis used one PTool compute and query node running on the least loaded workstation of the SP-2 system (with a TB0 switch) at the University of Pennsylvania.

A prototype physics SCF-analysis, representing the particle decay $Z^0 \rightarrow e^+e^-$ was coded in C++. We achieved an i/o-throughput τ_{io} of approxi-

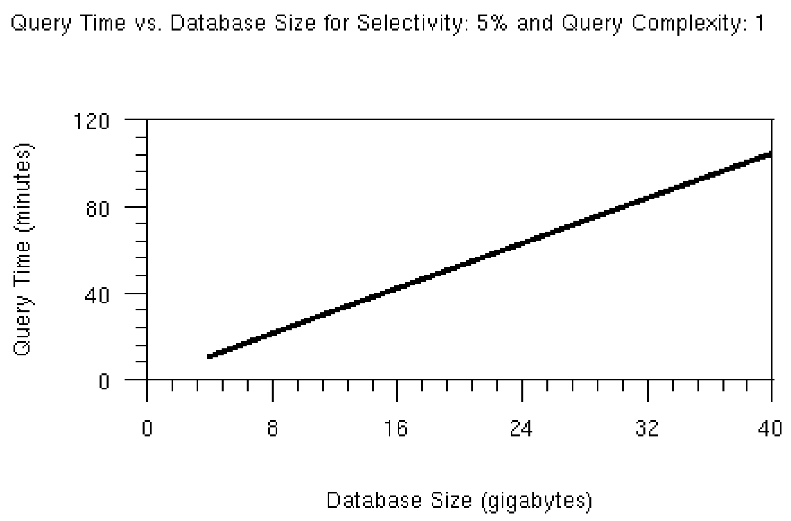

Figure 2: This graph shows the result of varying the size S of the store using synthetic data.

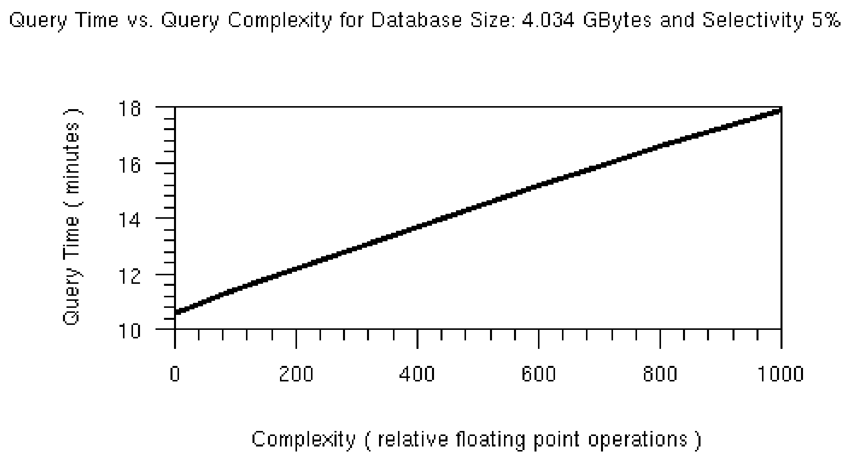

Figure 3: This graph shows the result of varying the complexity κ of the query using synthetic data.

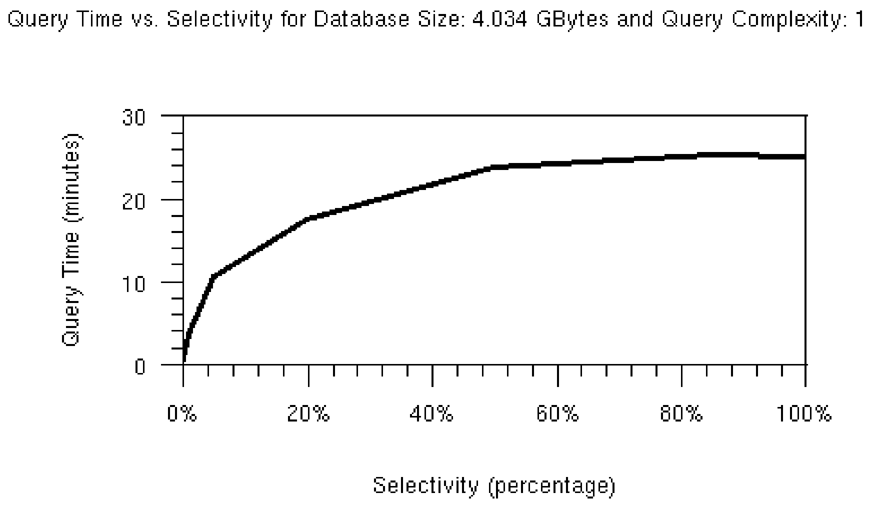

Figure 4: This graph shows the result of varying the selectivity σ_{io} of the query using synthetic data.

mately 2.9 Mbytes/seconds, while the production code at Fermi Laboratory (written in Fortran) delivered approximately 0.88 Mbytes/seconds when reading the corresponding data in the legacy file format in which it was stored.

7 Discussion and Conclusion

In this paper we have described the design, implementation and usage of a specialized object warehouse for data mining queries which is built using a persistent object manager. Our interest is in using these object warehouses to support data mining queries involving numerically and statistically intensive analyses on large data stores, containing large numbers of objects, each of which has large numbers of attributes.

We focused on data mining queries involving *selecting* objects based upon a selection criterion involving one or more of their attributes, *computing* additional derived data, *filtering* to retain those objects satisfying a filtering criterion, and aggregating the results for further analysis. We call these types of queries SCF-queries. They are common in scientific data analysis, scientific computing, and related areas.

More precisely our interest was designing, implementing and operating an object warehouse in which SCF-queries scale as the data size, selectivity,

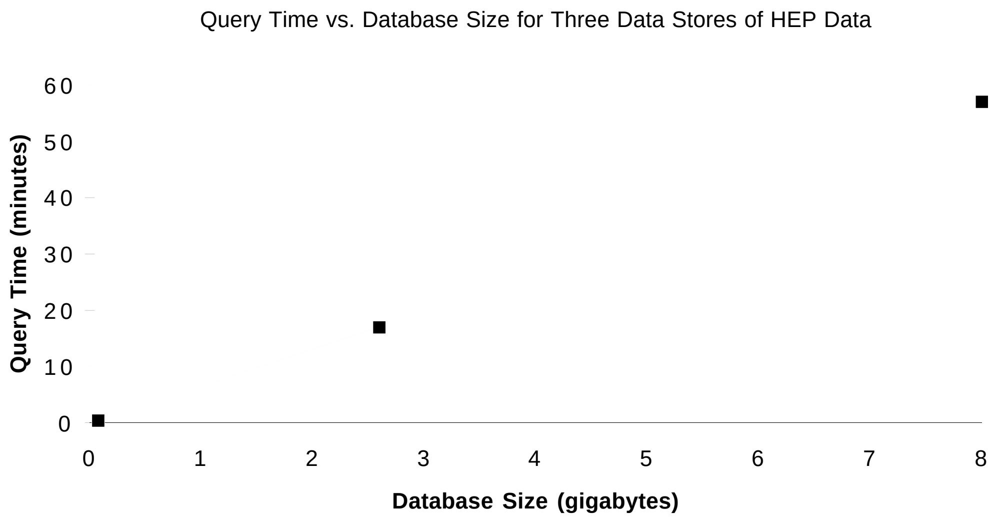

Figure 5: This graph shows how the query time scales linearly with the size of the store using high energy physics data. Some of the data for the last two points are estimates. This graph is from [18], where additional details can be found about this work.

and complexity of the queries range over three orders of magnitude. Specifically, we desired the ability to execute transparently queries lasting seconds on megabytes of data touching all of the attributes to queries taking hours on tens to hundreds of gigabytes of data touching 0.01% of the data. In this paper, we reported on results using actual data over most of this range and using synthetic data with the same structure over the entire range.

To provide this range of scalability we worked with persistent object stores with the following characteristics:

> *Monotone.* We assume that the stores are read-mostly, with data appended, and infrequently updated.
>
> *Optimistic.* We assume that the persistent object manager can provide low overhead access to data by exploiting knowledge about how the data is layed out on disk.
>
> *Parallel Access & Computing.* We assume that the object warehouse supports parallel access, computation, and filtering. To simplify the design we assume that the return type of a computation is a collection of homogeneous objects.
>
> *Precomputed and Indexed.* To further gain performance object warehouses precompute and index as much information as possible.

We use the acronym MOPPI to describe object stores with these characteristics.

It is usually sufficiently difficult to work with large data sets of actual data, much less to test different object store designs using real data. Although we have done so in this paper, we also have relied heavily on synthetic data having the same structural characteristics and have not been able to test as many alternate designs as may have been ideal.

In this work, we have provided a brief description of a MOPPI store we designed and implemented called PTool and uncovered some short-comings in our current implementation (Version 2). We mention two of the most important:

First, although read-only access works well for numerically and statistically intensive data access during data mining queries, it is much less satisfactory when populating persistent object stores that may be hundred gigabytes in size. For this reason, we are currently exploring a variety of means of supporting back up, recovery, and concurrency during populating very large object stores.

Second, with one hundred gigabytes of disk spread over a few dozen nodes managing the physical resources of the system becomes an important problem. Nodes, disks, and other peripherals may come and go. Distributing i/o-processes, compute processes, and filtering processes over the appropriate nodes is also a problem. For this reason, we are experimenting with different resource management algorithms, strategies and tools for handling these and related problems.

In this paper, we feel that we 1) have made the case for using MOPPI stores for data mining queries which involve selecting, computing and filtering large collections of objects, each of which has a large number of attributes, and 2) presented evidence that this approach scales over a wide range of data sizes, selectivities, and query complexities.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, "Efficient Similarity Search in Sequence Databases," *Lecture Notes in Computer Science,* Volume 730, Springer-Verlag, pp. 69–84, 1993.

[2] M. P. Atkinson, V. Benzaken and D. Maier, editors, *Persistent Object Systems (Proceedings of the Sixth International Workshop on Persistent Object Systems),* Springer-Verlag and British Computer Society, 1995.

[3] R. G. G. Cattell, editor, *The Object Database Standard ODMG-93*, Morgan Kaufmann Publishers, San Francisco, California, 1994.

[4] M. Damashek, "Gauging Similarity with n-Grams: Language-Independent Categorization of Text," *Science,* Volume 267, pp. 843–848, 1995.

[5] A. Dearle, G. M. Shaw, S. B. Zdonik, editors, "Implementing Persistent Object Bases: Principles and Practice," Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[6] J. Dozier, "Access to Data in NASA's Earth Observing System," in *Proceedings of the 1992 ACM SIGMOD, SIGMOD Record*, Volume 21, ACM, New York, 1992, page 1.

[7] C. T. Day, S. Loken, J. F. MacFarlane, E. May, D. Lifka, E. Lusk, L. E.Price, A. Baden, R. Grossman, X. Qin, L. Cormell, P. Leibold, D. Liu, U. Nixdorf, B. Scipioni, T. Song, "Database Computing in HEP — Progress Report," *Proceedings of the International Conference on*

Computing in High Energy Physics '92, C. Verkerk and W. Wojcik, editors, CERN-Service d'Information Scientifique, 1992, ISSN 0007-8328, pp. 557-560.

[8] D. R. Quarrie, C. T. Day, S. Loken, J. F. Macfarlane, D. Lifka, E. Lusk, D. Malon, E. May, L. E. Price, L. Cormell, A. Gauthier, P. Liebold, J. Hilgart, D. Liu, J. Marstaller, U. Nixdorf, T. Song, R. Grossman, X. Qin, D. Valsamis, M. Wu, W. Xu, A. Baden, "The PASS Project: A Progress Report," *Proceedings of the Conference on Computing in High Energy Physics 1994,* edited by S. C. Loken, pp. 229–232, 1995.

[9] N. Araujo, R. Grossman, D. Hanley, W. Xu, S. Ahn, K. Denisenko, M. Fischler, M. Galli D. Malon and E. May, "Some Remarks on Parallel Data Mining Using a Persistent Object Manager," *Proceedings of the Conference on Computing in High Energy Physics 1995,* to appear.

[10] S. Bailey, R. Grossman, and D. Hanley, D. Benton and B. Hollebeek, "Scalable Digital Libraries of Event Data and the NSCP Meta-Cluster," *Proceedings of the Conference on Computing in High Energy Physics 1995,* to appear. *Laboratory for Advanced Computing Technical Report,* Number LAC 96-R3, University of Illinois at Chicago, 1995.

[11] R. L. Grossman D. Hanley, and X. Qin "Caching and migration for physical collections of objects: Interfacing persistent object stores and hierarchical storage systems," in *Proceedings of the 14th IEEE Computer Society Mass Storage Systems Symposium,* S. Coleman, editor, IEEE, 1995, to appear.

[12] R. L. Grossman and X. Qin, "Ptool: a scalable persistent object manager," *Proceedings of SIGMOD 94,* ACM, 1994, page 510.

[13] R. L. Grossman, N. Araujo, X. Qin, and W. Xu, "Managing physical folios of objects between nodes," *Persistent Object Systems (Proceedings of the Sixth International Workshop on Persistent Object Systems),* M. P. Atkinson, V. Benzaken and D. Maier, editors, Springer-Verlag and British Computer Society, 1995.

[14] R. L. Grossman, D. Hanley, and X. Qin, "PTool: A Light Weight Persistent Object Manager," *Proceedings of SIGMOD 95,* ACM, 1995, to appear.

[15] R. L. Grossman, D. Valsamis and X. Qin, "Persistent stores and hybrid systems," *Proceedings of the 32nd IEEE Conference on Decision and Control,* IEEE Press, 1993, pp. 2298-2302.

[16] R. L. Grossman, A. Lifschitz, and Z. Likoudis, Persistent Object Stores of Two-Dimensional Point Vortices, *Laboratory for Advanced Computing Technical Report,* Number LAC 96-R4, University of Illinois at Chicago, 1995.

[17] R. L. Grossman, "A Computational Model for Data Mining Queries," *Magnify Technical Report,* Number 96-R6, February, 1996.

[18] S. Bailey, D. Benton, R. L. Grossman, D. Hanley, B. Hollebeek, R. Oliveira, "Mining Persistent Object Stores of High Energy Physics Data: A Case Study Using PTool," *Laboratory for Advanced Computing Technical Report,* Number LAC 96-R6, University of Illinois at Chicago, 1996.

[19] R. L. Grossman, S. Bailey, and D. Hanley, "Selecting, Computing and Filtering Large Object Stores, in preparation.

[20] J. G. Verly and R. L. Delanoy, "Model-based Automatic Target Recognition (ATR) System for Forwardlooking Groundbased and Airborne Imaging Laser Radars (LADAR), *Proceedings of the IEEE,* Volume 84, pp. 126–163, 1996.