Database Computing in High Energy Physics

Andrew BADEN
Department of Physics, University of Maryland
and
Robert GROSSMAN[1]
Laboratory for Advanced Computing and Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago

Abstract

We describe preliminary work concerning the design and prototyping of an extensible, object-oriented database designed to analyze data in high energy physics. These databases are motivated by the data processing and analysis needs of a generic high energy physics experiment to be proposed for the Superconducting SuperCollider (SSC), and requires the collection and analysis of between 10 and 100 million sets of vectors (events), each approximately one megabyte in length. We sketch how this analysis would proceed using an extensible, object-oriented database which supports the basic data types used in HEP.

1 Introduction

In this paper, we describe preliminary work concerning the design and prototyping of an extensible, object-oriented database designed to analyze data in high energy physics (HEP) experiments at the Superconducting SuperCollider (SSC) laboratory. These experiments will require collecting and analyzing approximately 10 to 100 million "events" per year during proton colliding beam collisions. Each "event" consists of a set of vectors with a total length of approximately one megabyte. This represents an increase of approximately 2–3 orders of magnitude in the amount of data accumulated by present large HEP experiments. To effectively store and analyze this large amount of data will require that HEP focus on new techniques from data engineering. In Section 2, we describe the requirements of the database. In Section 3, we review the methods used to analyze data in a large present-day state-of-the-art high energy physics experiment which will serve as our "prototype" for the SSC. In Section 4, we review data models available for analyzing scientific data. In Section 5, we sketch how this analysis would proceed using an extensible, object-oriented database, which supports the basic data types used in HEP, including runs, events, muons, tracking data, *etc.* Section 7 contains further discussion of a HEP database. Section 8 describes some design considerations.

Acknowledgements. Section 2 is based in part upon some working notes of Irwin Gaines. Sections 3, 5 and 6 are based in part upon [2] and [3]. We wish to thank Chris Day and Larry Price for several helpful discussions about this paper. Finally, this paper is a preliminary: a more complete version will be published elsewhere.

2 Requirements

In this section, we review the computing requirements of a present day HEP experiment to establish a baseline. We then consider the analogous requirements for a generic HEP experiment at the SSC. Expectations at the SSC are that there will be an increase relative to the largest ongoing HEP

[1]This research is supported in part by NASA grant NAG2-513. Address: Department of Mathematics, Statistics, and Computer Science, m/c 249, University of Illinois at Chicago, Box 4348, Chicago, IL 60680, grossman@uicbert.eecs.uic.edu.

COMPUTING IN HIGH ENERGY PHYSICS '91

experiments to date of about ~ 1 order of magnitude in the size of each event (to 1 Mbyte/event), ~ 2–3 orders of magnitude in the number of events collected for analysis (to 10–100 Hz), and ~ 1 order of magnitude in the number of physicists who will be accessing the data (to ~ 1000 physicists).

At present, high energy physics (HEP) colliding beam experiments at the Fermi National Accelerator Laboratory (FNAL) measure the radiation products from the collision of particle beams at rates of up to 285 kHz. The requirements can be broken down into the following categories:

Event recording. In the *event recording* stage, events which warrant scientific investigation are identified through a sequence of several decisions (a mixture of hardware and software processing), each requiring greater execution time. A decision to keep a particular event, called a *trigger*, results in the corresponding raw data being collected and written to magnetic tape. A small fraction (~ 1 Hz) of the total number of events is recorded on magnetic tape for data analysis.

Mass Storage. At 1 Hz, with each event requiring $\sim$ 150 KBytes of digital information, $\sim$ 15 GBytes/day are recorded on 8 mm tape. Over the course of a year approximately 1 TByte of data is recorded.

Reconstruction. In the *reconstruction* stage, the raw data is read from tape and processed by production codes which reconstruct the digital data in order to identify the number and type of the particles which characterize each event. Specifically, the result of the production codes are dimensional quantities such as energy, mass, momentum, orbits, event topology, *etc.* These data summaries are written to some medium, usually magnetic tape, for further analysis, referred to as *data summary tapes* (DSTs). The reconstruction is done using a farm of workstations and ....

Analysis. In the *analysis stage*, computations are performed on the data in the data summary tape. For instance, such computations can be used to discover (or verify) correlations between certain types of events, or measure an average of a quantity over a particular set of events.

Computing environment Approximately one hundred physicists at ten institutions require access to the data.

With today's technology, it makes sense to modify this strategy in two important ways:

- by making use of parallel processing, it is now possible to reconstruct data in real time;
- by making use of database technology, it is possible to provide random access to some fraction of the archived events.

With these changes in mind, reasonable requirements for a generic HEP experiment at the SSC are:

Event recording. Approximately 10–100 Hz events, each 1 MByte is size, are expected to pass the last (level three) trigger and be recorded.

Reconstruction. If each event requires 100–1000 instructions to process, then to reconstruct the 10–100 events/sec in real time, requires approximately 10,000 MIPs. Providing one workstation per event should make this possible.

Mass storage. Approximately 10–100 events/sec, each 1 MByte in size, requires recording approximately 1–10 TBytes/day, or approximately 1 PByte/year. Using a hierarchical storage model, it is reasonable to keep approximately 0.1–1% of the data in secondary storage, providing a 1–10 TByte "online" database.

Computing environment. The goal is provided approximately 1000 individuals at approximately 100 institutions access to the data.

3 Traditional Data Analysis in HEP

Most HEP computing schemes store data in sequential records using memory management software systems written at HEP laboratories. Data are analyzed via computer programs which are written mostly in Fortran, access events from storage as records, read it into common memory, and provide hooks for users to apply custom-written

subprograms. Each of these programs is compiled and linked to a standard set of libraries. Linking, at present, is not usually done interactively.

Once the analysis program is built, it is made to loop over a number of events. For each event the entire record is read from storage, and those parts which are relevant to the particular analysis (and very few analyses use all of the data in a single event) are used by the user subprogram. For instance, a typical session would consist of:

1. Read in each event. If the events are on tape, access the tape first. For each event:
   - require certain global event characteristics;
   - calculate quantities from each event (*e.g.* the presence of a certain number of a certain type of particle);
   - require the particles to have certain characteristics;
   - fill histograms or scatter plots of distributions.
2. If desired, save events passing requirements onto disk or tape.
3. Report results of calculations, histograms, plots, regressions, *etc.* at the end of session.

This procedure is very straight forward, and there are no obvious inefficiencies to be trimmed away. However, there are inefficiencies which involve the repetition of this procedure many times due, for example, to debugging the code or the application of additional cuts which result in smaller and smaller data sets. Events with a record size of 1 Mbyte (or more) can easily result in an unacceptable I/O load on the system, necessitating either a drastic reduction in the size of each event through either a compression or trimming and/or a drastic cut in the number of events. The critical point is that access to the data is via entire event records, and in a sequential manner, *i.e.* events are read as atomic.

Recently, the data analysis stage has been made easier by using an interactive data analysis program developed at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. The program, called PAW (Physics Analysis on Workstations), resembles a spread-sheet program and incorporates an inline Fortran interpreter. The Fortran interpreter eliminates the need to need to link the large Fortran programs previously used to analyze the data. The interpreter has been found to be fast enough, since most of the time spent when running in the interpreter mode is for I/O. Data analysis with PAW must be preceeded with a transformation of the data to a limited relational tuple. However, once the data is transformed, the program allows one to interactively identify events with computed quantities passing certain thresholds (*cuts*) and to study correlations and anti-correlations of the particles and events. The program also supports many standard statistical operations, such as constructing histograms, scatter plots, curve of best fit, and *etc.* However, the system has well-defined data structures which do not allow dynamic changes, and require a preprocessing to transform existing HEP data (which is by nature dynamic).

4 Available Data Models

We are proposing that database technology be used to analyze data from HEP experiments. In this section, we review some basic data models and how they apply to HEP. In the next section, we describe a HEP data model.

Relational model. The relational database was developed in the early 1970's, sparked by the influential paper of Codd [7]. Prototypes emerged in the middle to the late 1970's, including Ingres [17] and System R [1]. To visualize a relational database, consider a set of variables, or equivalently a multi-dimensional system space. In database parlance, each dimension (or variable) is referred to as a *domain*. Now consider that in this system, there are relationships, or conditions, between domains which tend to group certain values of each domain across domain boundaries. Such relationships group the domains into tables where the columns are domains and the rows (called *tuples*) are the grouping of domains. As an example, consider a HEP event at the high level where one is concerned with "objects" such as the 4-vectors

Run	Event	Pte (p_t e^+/e^-)	Ptmu (p_t μ^+/μ^-)
1001	126390	74.1	3.5
1006	69372	0	78.0
1007	574930	62.3	215.2
1020	6320	99.2	0
	⋮		

Figure 1: A HEP table called *HEPdata*.

of the electrons, muons, jets, *etc.* One domain is the set of all electron transverse energies, another might be the same for muons. Each row is an event, which is the *relation* which causes particular electrons and muons to be grouped, and the entire set of relations forms a *table* (or *relation*) in database parlance. See Fig. 1 for an example of a HEP table.

An important attribute of a relation is that it abstracts a certain type of file. A file of this type would be a sequence of records, one for each tuple in the database. Each record would consist of a sequence of fields, one for each column in the table. The definition of a relation implies that all records in the file would have the same number of fields, that no duplicate records be allowed, and that each field be atomic and have no additional structure.

Relational databases support special languages (called *query languages*) which provide for the analysis of the data in the database. For instance, in the HEP example, if you wanted to collect all events with an electron AND a muon with p_t above a cut (p_{cut}), the query language may take the form as in Fig. 3 and produces a *new* table as in Fig. 4. A query is simply a function on the database acting on relations and producing other relations. It is an essential feature of such queries that they produce other relations, which can then be queried in exactly the same way. Relational databases also support the taking of intersections and unions of relations, which are called joins and meets, to produce other relations. In this way, a relational database provides a powerful means of completing queries on large databases containing containing numeric and alphabetic fields.

Extensible model. By the mid 80's, it was recognized as a good idea to relax the requirement that domains consist of atomic objects and extend databases to support a collection of data types richer than merely numbers and strings. By the late 80's, a number of such systems have been prototyped, including Genesis [4], Exodus [6] Dasdbs [16] and Postgress [18].

```
Range of t is HEPData
Retrieve into TOPTable (t.Run, t.Event)
Where Number (TopCandidates (t.Run)) ≥ 0
```

Run	Event
1007	574930
	⋮

Figure 2: A query and the relation it produces.

Imagine a relational database which is extended to support not only numbers and characters but also physical (and logical) objects such as events, muons, calorimetry tower lists, *etc.* In other words, the table entries are no longer assumed to be atomic, but rather to have an internal structure. For example, in Fig. 1, imagine that there is a column (domain) called "TOP" which is a basic data type known to the system. Queries such as in Fig. 2 would then be possible.

See Carey [6] for a recent survey covering some of these issues and detailing other approaches.

The statistical and scientific data model. Beginning in the late 70's and early 80's, there has been increasing interest in statistical and scientific databases. These types of databases differ in a number of important ways from conventional relational databases.

- The datatypes are different: datatypes in a statistical database include time series, matrices, and usually make provisions for missing data, outliers, *etc*; datatypes in a conventional relational database are usually limited to numbers and strings.
- Operators computing averages, standard deviations, regressions, *etc.* are important in a

statistical database, while operators computing meets and joins are important in a conventional relational database.
- Updating and modifying the data in a conventional relational database is frequent; on the other hand, in a statistical database, data may be added to the database, but is rarely changed.

Much of the data in a statistical or scientific database is static: updates are infrequent, but the queries are usually computationally very intensive. For example, the queries of interest typically cannot be answered from the stored relations and data; rather, some information typically needs to be computed, such as a regression. In other words, a typical query requires extracting features or some type of summary information from the database. In the past few years, there has been increasing interest in statistical and scientific databases. See for example, Hammond [10] and Rafanelli [15] and the references cited there.

The object-oriented data model. Beginning in the early 1980's, object-oriented concepts began influencing a variety of computing disciplines, including programming language design, artificial intelligence and knowledge representation, and databases. There have been several research projects prototyping object-oriented databases, including Iris [20], O_2 [8] and Orion [12].

Just as a relational database is organized according to a relational data model, consisting of sets, domains, elements and relations as described above, an object-oriented database is organized according to an object-oriented data model. There have been several attempts to define an object-oriented data model. We follow Kim [13] by requiring the data model support the following concepts:

1. Entities modeled in the database correspond to *objects*. Each object has a unique identifier.
2. An object has *attributes*, and each attribute assumes a value or set of values. Each such value is an object in its own right.
3. An object and its attributes may be modified only by explicitly specified *methods*. In this fashion, the object is encapsulated.
4. Objects are grouped together into *classes*, which are similar to abstract data types. An object is an instance of a class.
5. Classes belong to a *hierarchy*. A class may have zero, one or several subclasses. A class may *inherit* attributes and methods from zero, one or several superclasses.

5 Data Analysis Using a HEP Database

We are currently designing and prototyping an extensible, object-oriented database specifically designed to handle the data analysis of HEP experiments. The HEP database supports the basic *objects* of high energy physics, including events, electrons, muons, jets; *etc.*, *relations* expressing relationships between the objects, and *methods* or *operators* acting upon the objects and relations.

Complex objects [11] are supported. For example, the object Event consists of several objects, including the objects RawData and ComputedData. The object RawData consists of several objects, including CalorimeterData and TrackingData, while the object ComputedData consists of several objects, including CellAddress and Helix.

A typical relation consists of rows corresponding to events and columns containing the various measured and computed quantities derived from events. See Fig. 1 for an example of a HEP table.

Objects have attributes, corresponding to columns in a relational database. For example, the object CalorimeterData has the attributes CellAddress and TotalEnergy. Methods act upon objects, including user-defined methods written in C or Fortran. For example, the method ComputeSpatialHelix acts upon the object TrackingData and returns the five parameters determining the spatial helix corresponding to the tracking data. There are many methods which statistically analyze the events. For example, the object Run consists of a sequence of Events, and the method MuonCandidates acts upon the object Run, analyzes the individual events, and returns the events with candidate muons for further analysis.

The database design is object-oriented as defined by Kim [13]: The state of an object is determined by the set of values for its attributes and objects are modified by methods which act upon

```
Range of t is HEPData
Retrieve into TOPTable (t.Run, t.Event,t.Pte)
Where t.Pte> p_cut and t.Ptmu> p_cut
```

Figure 3: A typical query.

Run	Event	Pte (P_t e^+/e^-)
1007	574930	62.3
	⋮	

Figure 4: Result of query on HEPdata.

them. Each object is given a unique object identifier. Objects belong to classes, and subclasses are supported. Subclasses inherit the attributes and methods of their superclasses.

The analysis of the HEP data can now proceed by querying the database. For instance, if you wanted to collect all events with an electron *and* a muon with the transverse momentum p_t above a cut (p_{cut}), the query in Fig. 3 may be used to generate the new relation in Fig. 4.

In summary, this reduces the $t\bar{t}$ $e\mu$ analysis discussed in Section 3 to a single query. The query interpreter for the database must perform the following actions:

1. Retrieve for each event in the HEPData table the data needed to calculate the number of electron and muon candidates.
2. Using this data, determine whether the event contains an electron or muon candidate. Unlike queries in a traditional relational database, queries of like this may require elaborate computations.
3. Retain those events which have at least one candidate which passes the cut determined by the threshold value.
4. Save the resulting events into a table that can be used as a basis for further queries, requiring additional computation of candidates and cuts.

6 Discussion of HEP Database

We begin by listing some of the advantages of analyzing HEP data using database computing compared to traditional HEP data analysis, as it is done in the colliding beam experiments at FNAL, for instance.

- Data access is more efficient, since
  - queries always produce other tables, which can themselves be made the subjects of queries;
  - when queries produce tables, the underlying data itself is not copied, but rather appropriate pointers are introduced pointing to the original objects, which may be quite large;
  - indices can be introduced to speed up subsequent queries.
- Analysis of the data is immediate and interactive.
- Many users can have simultaneous access to the data and create their own "views" of the data without modifying the underlying data or expensive rereading and recomputation of the entire underlying data set. Different collaborating institutions can efficiently share data in this way. With the appropriate queries, any subset of the data can be used to construct a table, which can be accessed by a collaborating institution. This can be done without necessarily requiring sequential access of all events, since now each "event" carries a unique object identifier.
- This approach "modularizes" the data analysis, lessening the hardware and software requirements and significantly reducing the the total amount of input/output.
- Data can be selected via simple queries using simple extensions of the standard query languages.
- Data can be analyzed with built in functions without the need for Fortran programming.
- Unlike the PAW program, this approach does not require the use of unstructured, device-independent binary files.

The inclusion of new data types in a HEP database requires that new query operators be supported, new storage and access methods be developed, and new methods of optimizing queries be found. In the case of HEP data, for example, it is important that the database support the storage and retrieval of events of an arbitrarily large size. It is also important that query operators defined on entire objects be supported, such as operators returning various statistical quantities. It is essential for HEP applications that arbitrary FORTRAN (and C) user-written subprograms interface easily into the query language, *i.e.*that there be a "hook" which the user can use to input customized operators to perform the queries. This is simply a logical extension of existing traditional HEP analysis methodology.

7 Design Considerations

Architecture. We begin by describing how the database might fit into the general computing environment at the SSCL. One network architecture is to connect three backbones by high speed links:

1. a *production backbone* residing at the SSCL and responsible for the reconstruction and archiving of the data;
2. a distributed *analysis backbone* consisting of workstations and responsible for program development, testing and physics analysis of the data;
3. a *database backbone*, consisting of a primary database server residing at the SSCL and several secondary database servers distributed world-wide, and responsible for the distribution of the appropriate data to the analysis backbone.

Computing would be done using a client-server model, with various, distributed analysis and database servers providing client applications the necessary information required for the analysis and processing of the data.

We turn now to the design of the HEP database itself: our strategy is to design the database using modular components with well-defined interfaces. The components include a storage manager, an object manager, a table manager, a recovery manager, a concurrency manger, an interpreter, a query manager, a query optimizer, a report manager, and an adminstrative manager. We will pay particular attention to emerging standards in two areas: the IEEE Mass Storage Model [14] and the X3/SPARC/DBSSG OODB Task Group on the Standardization of Object Database Systems [19].

References

[1] Astrahan, *et al.*, "System R: A relational approach to database management," *ACM Transactions on Database Systems*, 1 (1976).

[2] D. Baden and R. Grossman, *A Model for Computing at the SSC*, Superconducting Super Collider Laboratory Technical Report No. 288, 1990.

[3] D. Baden and R. Grossman, "Database computing in high energy physics," *Laboratory for Advanced Computing Technical Report*, Number LAC90-R32, University of Illinois at Chicago, December, 1990.

[4] D. Batory, T. Leung and T. Wise, "Implementation concepts for an extensible data model and data language," *ACM Transactions on Database Systems*, Vol. 13, 1988.

[5] R. Bertl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E.H. Williams, and M. Williams, "The GemStone Data Management System," *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F. H. Lochovsky, editors, ACM, New York, 1989, pp. 283–308.

[6] M. Carey *et al.*, "The Exodus extensible DBMS project: An overview" *Readings in Object-Oriented Database Systems*, S. Zdonik and D. Maier, editors, Morgan-Kaufmann, 1989.

[7] E. Codd, *A relational model of data for large shared data banks*, Com. ACM, Vol. 13, pp. 377–387.

[8] O. Deux, *et al.*, "The Story of O_2," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, pp. 91–108, 1990.

[9] D. Fishman *et al.*, "Overview of the Iris DBMS," *Object-Oriented Concepts, Databases, and Applications*, W. Kim and F.H. Lochovsky, editors, ACM, New York, 1989, pp. 219–250.

[10] R. Hammond and J.L. McCarthy, editors, *Proceedings of the Second International Workshop on Statistical Database Management*, Springer-Verlag, New York, 1983.

[11] R. Haskin and R. Lorie, "On extending the functions of a relational database systems," in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, June 1982, pp. 207–212.

[12] W. Kim, J.F. Garza, N. Ballou and D. Woelk, "Architecture of the Orion next generation database system," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, 1990, pp. 109–124.

[13] W. Kim, "Object-oriented databases: definition and research directions," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, 1990, pp. 327–341.

[14] "Mass Storage System Reference Model, Version 4" edited by Sam Coleman and Steve Miller, IEEE. to appear.

[15] Maurizio Rafanelli, "Research Topics in Statistical and Scientific Database Management," in *Statistical and Scientific Database Management*, M. Rafanelli, J.C. Klensin, and P. Svensson, editors, Springer-Verlag, Berlin, 1989.

[16] H. Schek *et al.*, "The Dasdbs project: objectives, experiences, and future perspectives," *IEEE Transactions on Data and Knowledge Engineering*, Vol. 2, 1990.

[17] Michael Stonebreaker, ed., *The Ingres Papers*, Addison–Wesley, 1986.

[18] M. Stonebreaker, L. Rowe and M. Hirohama, "The Implementation of Postgres," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, pp. 125–142, 1990.

[19] "Object Data Management Reference Model," draft report prepared by the Object-Oriented Databases Task Group (OODBTG) for the Database Systems Study Group (DBSSG) of the Accredited Standards Committee X3.

[20] K. Wilkinson, P. Lyngbaek and W. Hasan, "The Iris architecture and implementation," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, pp. 63–75, 1990.