Database Computing in HEP—Progress Report

C. T. Day, S. Loken, J. F. MacFarlane

Lawrence Berkeley Laboratory [1]

E. May, D. Lifka, E. Lusk, L. E. Price

Argonne National Laboratory

A. Baden

Department of Physics
University of Maryland

R. Grossman, X. Qin

Department of Mathematics, Statistics, & Computer Science
University of Illinois at Chicago

L. Cormell, P. Leibold, D. Liu, U. Nixdorf, B. Scipioni, T. Song

Superconducting Supercollider Laboratory

The major SSC experiments are expected to produce up to 1 Petabyte of data per year each. Once the primary reconstruction is completed by farms of inexpensive processors, I/O becomes a major factor in further analysis of the data. We believe that the application of database techniques can significantly reduce the I/O performed in these analyses. We present examples of such I/O reductions in prototypes based on relational and object-oriented databases of CDF data samples.

Introduction

The SDC and GEM experiments at SCCL are sized to produce up to 1 Petabyte (10^{15} bytes) of data per year each. The dominant CPU load of primary reconstruction is expected to be handled by farms of inexpensive processors. However, subsequent analysis represents an I/O load about four orders of magnitude beyond our current experience. Since I/O performance historically has not kept up with CPU performance gains, we have proposed [1] moving from the traditional, serial access, tape-based analysis systems of the past to more sophisticated database approaches. The database organization will allow a program to read only the portions of an event it actually needs, thereby significantly reducing the amount of I/O required.

To test these ideas, we have built prototype systems [2] based on 100–200 Megabytes of CDF data. We have constructed systems using a conventional relational database system, an object-oriented database system and Ptool, an home-brew object storage system. We report on the performance of each prototype. In addition, we describe some of the data modeling considerations involved in adapting to the two types of system. Finally, we discuss some of the design considerations raised by the need for a flexible data structure in a physics research

[1]Supported by the U. S. Department of Energy under Contract No. DE-AC03-76SF00098.

environment coupled with the enormous size of the database and consequent expense of restructuring it.

Databases

Commercial relational database systems organize information into tables with each row in the table representing one datum and each column describing a feature of that datum. For example, a table of Employees might have columns for Taxpayer ID, Name, Job Class ID, and salary. A table of Jobs might have columns for Job Class ID, Job Title and salary range. To find the Job Title of the Employee with Taxpayer ID 343-88-2134, the system finds the row in the Employee table with matching Taxpayer ID, picks out the Job Class ID from that row, then finds the row in the Jobs table with the matching Job Class ID and returns the Job Title.

We have also worked with a commercial object-oriented database system. It functions by making otherwise ordinary C++ objects persistent. That is, data placed in such an object can be recovered by another program even after the creator has exited. This is true even of links between structures. In the example above, the joining of the tables on Job Class ID would be replaced by a simple link from an Employee structure to the corresponding Job structure.

PTool [3] is a persistent object storage system developed at the University of Illinois at Chicago. It provides persistence by using the mmap() function to map persistent files to regions of virtual memory.

Data Modeling

We have taken ψ data from CDF's 1988–89 run, analyzed the event Ybos bank structures and recast them into an Entity-Relationship model suitable for both kinds of databases. The E-R model was then coded by hand into SQL to create relational tables and into C++ to create persistent objects.

Traditionally, all the banks representing an event are stored together. Programs analyze events by reading in all banks, deciding if the event is of interest and plotting some meaningful quantity. The filtering and plotting calculations rarely require all of the event banks, but they are read anyway. Since many analyses require limited calculation once reconstruction is done, this produces a great deal of unnecessary I/O.

In contrast, we have stored all of the entities of a given type for *all* events together, either in one table per entity type in a relational system or in one object database per entity type in an object-oriented one. As much or as little of any given event as needed can be analyzed by doing table joins on the event ID field, or by following links; neither system fetches data until it is actually touched. This way, analysis programs can read in only those portions of events that they will actually use.

Hand conversion of the E-R model to code was a laborous process, largely due to the variable lengths of the original Ybos banks. Neither database handles repeated substructures within a table/object well, so additional entities were created. We are working with automated tools to help with this process. For the commercial relational system, we have used ERDraw

Database	Object-oriented	Relational	PTool	Ybos	units
Ybos Data Loaded	215	42	42	109	MBytes
Database Size	241	115	72	109	MBytes
Time to Load	3.75	2	2	0	hours
ψ Mass Query	40	640	30	840	seconds

Table I: Prototype Database Performance Results

[4] to generate SQL from a graphical version of the E-R model. This tool shows considerable promise and works with several commercial databases. For the commercial object-oriented system, we are looking at the schema construction tool supplied with it. Currently, this tool has rather limited capabilities.

Performance

We have compared our prototype databases with native Ybos operations by selecting events with a di-muon mass in the range of the ψ. All tests began with the same 215 MByte set of Ybos data which contained 12 different bank types. By historical accident and limits on resources, the tests did not all include the same Ybos banks into their databases. The commercial object-oriented test did include all 12 banks. For the other three tests, an intermediate Ybos file was made by dropping three of the banks not involved in the event selection. For the Ybos test, this file was read sequentially and completely. For the remaining tests, eight of the remaining nine banks were loaded into databases from this file; the query was then run against the resulting databases. We have measured the sizes of the resulting databases, the time to fill the databases and the time to perform the event selection. All tests were performed on a SPARCstation I. The results appear in Table I.

For the Ybos case, we consider the intermediate file to be the "database", hence it takes no additional time to fill. The code which loaded the commercial object-oriented database also had to swap the data from VAX byte order and floating point format to the SPARC equivalent;this task alone takes 1.75 hours. The other two databases loaded from a hardware-neutral ASCII format file.

The ψ mass query benchmark in the Ybos case was a FORTRAN program which used the standard Ybos routines. These routines treat the file sequentially and read in complete events. Approximately 2000 events satisfied the selection. In the commercial object-oriented and PTool cases, the queries were C++ programs which only referenced three banks making up less than 3% of the database. The relational query was written in SQL and similarly only touched the relevent data.

Both object-oriented databases show a dramatic improvement in analysis time over the native Ybos. The commercial system achieves this with only a small space overhead of 12%, about half of which appears simply to be incompletely filled database segments. PTool uses more space since it was decided to populate the store using unpacked data, rather than the packed data used by the other object-oriented database and by Ybos.

The relational performance is disappointing, but we have not exhausted all avenues of optimization. In particular, the very large space expansion must be controlled if this database system is to remain as a viable option.

Design Considerations

The tree structure of banks in an event can reach depths of four or more. In simple relational designs, leaves of the tree need unique keys obtained by concatenating keys from all the layers above them. This can lead to considerable storage overhead in very large databases. Furthermore, piecing together an event by doing joins at run time could lead to the time for a single event growing with the size of the entire database.

For object databases, keys are replaced by fixed size pointers, regardless of depth of the event structure. Furthermore, run time joins are not needed, so the overall scaling behavior may be better than relational systems. However, the scale up for SSC is so enormous that more study is needed for both systems.

As analysis proceeds, events often have new banks attached to them. For relational systems, one just defines a new table with the proper keys; all the old data is undisturbed. For object systems, however, if the event has a fixed set of pointers to its component banks, adding a bank could lead to a complete restructuring of the database, a prohibitively expensive proposition. A Lisp-like list structure may be more stable and is under investigation.

References

[1] A. Baden and R. Grossman, "Database computing and high energy physics," *Computing in High-Energy Physics 1991*, edited by Y. Watase and F. Abe, Universal Academy Press, Inc., Tokyo, 1991, pp. 59–66.

[2] R. Grossman, A. Baden, C. Day, D. Lifka, E. Lusk, E. May, and L. Price, "Analyzing High Energy Physics Data Using Database Computing: Preliminary Report," *Laboratory for Advanced Computing Technical Report, Number LAC91-R17,* University of Illinois at Chicago, December, 1991.

[3] R. Grossman and X. Qin, "PTool: A Software Tool for Working with Persistent Data", *Laboratory for Advanced Computing Technical Report Number 92-11,* University of Illinois at Chicago, 1992. To appear.

[4] E. Szeto and V.M. Markowitz, "ERDRAW 2.2. Reference Manual", *TR LBL-PUB-3084*, Lawrence Berkeley Laboratory, May 1991.