(Object) Relational Databases for High Energy Physics Data [1]

Michael Athanas [a,b]

Representing the Nile[1] Collaboration

[a] *Department of Physics, University of Florida, Gainesville, FL 32611*

[b] *Department of Physics, Cornell University, Ithaca, NY 14853*

We explore representation and population of commercial databases in conjunction with real high energy physics (HEP) applications. We have developed a unified object representation of a database based upon STL. We describe two implementations of this object interface for the Illustra Object Relational database and the Informix Online Data Server relational database. Due to the multi-dimensional nature of HEP data we have found data representation in an object relational database convenient. Throughput benchmark measurements have been performed comparing the two databases. We discuss how the conventional database client/server architecture adapts to a parallel distributed environment as well as how function shipping may be used in the open architecture provided by Illustra. We describe our overall experience with the Illustra and Informix products.

Key words: database; object; C++; relational; benchmark; performance; world peace

1 Introduction

Commercial databases have been around in one form or another for quite some time. Their evolution has been driven primarily by the needs of the business community. The information explosion caused by the internet has motivated new features from database manufacturers, such as the ability to manage Gbytes to Tbytes of data and the ability to provide advanced abstract

[1] work supported by the NSF under grant ASC-9318151

data representation. The relevant question addressed by this paper is whether the commercial database solution is viable for HEP.

Efficient data throughput is only one aspect to consider when dealing with large amounts of data. Data management and organization for large and complex collaborations are also an important consideration.

The target processing environment we are applying for is hundreds of networked workstations or PC's with several hundred users.

2 Illustra Object-Relational Database

The Object-Relational aspect of the Illustra database is derived from a new object-id (oid) basetype which may be used to refer to other records (objects) in the database. Illustra augments every record with additional hidden attributes, such as a unique oid. Although it is admitted by Illustra technical staff that their object referencing is a pretty facade - there is nothing you can do with Illustra that you cannot do with any traditional relational databases. However, Illustra provides a cleaner and simple way of expressing relations. For example, in Illustra it is possible to have an array of integers appear as a single attribute in a table. Behind the scenes, that attribute is actually an object-id referring to a side table of integers, which is how it would be done using a traditional relational database.

The real power of the Illustra database derives from its open architecture. It is possible and remarkably easy to extend the database server's vocabulary of basetypes and corresponding operators. This allows developers to build an intuitive representation of their data. For example, we have built a set of basetypes which are appropriate for physics data, such as *event_t*, *track_t*, *shower_t*, and so on. This allows queries of the form:

$$\begin{aligned} Select\ \ & header,\ track,\ shower,\ from\ Run(34456)\ where \\ & getNTRKCD(header) > 4\ and \\ & getKINCD(track) > 0\ and \\ & abs(getZ0CD(track)) < 0.05; \end{aligned}$$

The function getKINCD(header_t) takes a *track_t* object and is evaluated at the server. This permits the server to take advantage of its efficient data fetching mechanism and test the data selection criteria before the data are shipped to the client.

A collection of basetype definitions and corresponding functions are called a *datablade*. Illustra and 3rd party venders offers a host of datablades including

2d, 3d, statistics, and timeseries.

3 Informix Online Database

The Informix Online database[2] is advertised as a high performance scalable relational database. One of the more interesting features of the database, which has not been thoroughly explored at the time of writing this paper, is its scalable distributed architecture. That is, a database maybe implemented (and replicated) across many nodes. A query is decomposed in a way that distributes the service work appropriately.

In the Spring of '96, Informix acquired Illustra. The Univeral Server is a result of the merging of the two database technologies. Universal Server was only recently released and has not been evaluated at the time of this writing.

4 Object Application Programmer's Interface

A physics analysis program is generally highly complex and requires the speed and flexibility of a high-level language such as C++ or Fortran. Data in the form of calibration constants, geometry, event data (detector response and Monte Carlo generated) must be made available as input to the physics analysis. Similarly, an analysis program produces data in the form of data summaries. Our approach is to present a unified and intuitive interface for all data access.

We have chosen the C++ Standard Template Library (STL) to provide the basis for our interface. Collections of records (i.e., a set of detector tracks) are represented as a STL container. An iterator is obtained from this container and the objects from the container are brought into the application address space by dereferencing the iterator. It is completely intuitive to the STL programmer and flexible enough for our purposes. We have found that this interface accommodates a variety of persistence methods such as a persistent object manager[4], a file system[3], and databases.

The data portion of any object may make use of this interface and become persistent through inheritance and providing a few methods. As an example, the following code fragment produces a STL container by opening a connection to the "nile" store.

```
ODBstore<MyObject>  nileStore("nile");
ODBstore<MyObject>::iterator it = nileStore.begin();
```

The object *MyObject*, is the user constructed object that represents data records in a particular table. *MyObject*'s class definition includes an inheritance derivation from an ODBstore base class which implements the object communication to the underlying database implementation. The interface is lightweight and has been demonstrated not to interfere with the performance of the underlying implementation.

5 Data Model

One of the more challenging aspects of building a database is designing the data model, that is, specifying how the data are conceptually organized. A data model that yields good performance and easy maintenance results from careful design. The relational database language Structured Query Language (SQL) provides a native way of expressing a design.

Because of the challenging task of joining tables in a relational database, it is necessary to distinguish a physical layout from a logical layout. In the physical layout, all attributes that have the same multiplicity are grouped together in the same table. For example, there are some non-repeating attributes which are valid for an entire collision, such as the number of tracks, total visible energy, and so on. All of the attributes are part of a single table. Track momenta, error matrix, quality of fit, etc., all are grouped in a different table.

As we build an intuitive object representation of the data, we build a logical layout. Here, we take advantage of the database's ability to create virtual tables, or *views*, that contain attributes from tables from the physical layout. This provides optimal performance while maintaining an intuitive representation of the data.

6 Processing Architecture

Once a database is populated, information is requested in the form of a query. The query is sent off to the (distributed) database server and is processed. The resultant set of information corresponding to the query is delivered to the client process that requested it. That is, all the data are funneled into a single client process.

Conceptually, a centralized monolithic data repository is very appealing from a user's perspective. It would be unnecessary for a user to maintain data location information as in a distributed data configuration. Unfortunately, this type of architecture does not work well when there are many cpu-intensive jobs which perform better in a distributed processing environment.

The Informix+Illustra solution is to take advantage of function shipping; the user's analysis, written in Fortran, C++, or other high level language, is compiled and linked into a dynamic link library and registered with the database system. A user submits the analysis dll along with their data query and it is processed by the distributed servers. Unfortunately, all scheduling control is then relinquished to the database system.

Depending upon the kind of data served, a more practical processing architecture would implement multiple data servers on various nodes in a distributed processing environment. This permits a parallel links architecture which better adapts to the distributed processing paradigm, but is only scalable in a clumsy manner.

7 Conclusion

General purpose commercial databases are not a high performance solution for specific HEP data storage. However, high performance is only one aspect of the problem physicists are facing with the current information explosion. High quality data management and organization are necessary for maintaining coherence among large collaborations and complex data collection apparatus. The database community have developed time tested tools to this end. Unfortunately, it may be difficult to depend upon the survival of a particular database vendor over the lifetime of a physics experiment. However, the database language standards will survive. As long as the high level language interface is clean and the underlying database design is independent of vendor specific attributes, it is easy to migrate to the database vendor du jour.

The STL object interface with the Illustra database implementation is currently employed in the Nile Fast-Track system for system administration and in the CLEO III data acquisition *slow control* system.

References

[1] The Nile Project, http://www.nile.utexas.edu/

[2] For more information on the Informix Online Dynamic Server, see http://www.informix.com/informix/products/techbrfs/servconn/2073177/2073177.htm.

[3] See the talk by K. Jeong *A High Performance Data Server Optimized for HEP Data* presented at this conference.

[4] See the talk by M. Ogg *Experience with a Persistent Object Manager for HEP data* presented at this conference.

[5] See the talk by K. Honscheid *The CLEO III Data Acquisition System* presented at this conference.