HEP Data Analysis based on an Object Database Store

D. Düllmann *CERN, Geneva, Switzerland*

Abstract

The RD45 project at CERN is aimed at providing a persistent data store for LHC experiments. More than 1PB of event data per experiment a year have to be stored and retrieved for analysis by physicists distributed at some 100 institutes world wide. This paper discusses the design for an event data store based on commercial object databases management systems (ODBMS) and presents first results from the ongoing RD45 prototyping work.

Keywords: ODBMS, Persistency, Analysis, Event Data, RD45

1 Introduction

The design of the data management system for LHC experiments will be a challenging task. On one hand this system has to cope with the vast amount of measured data of more than 1 PB (10^{15} bytes) per experiment per year that has to be stored in a complex hierarchy of different storage media. On the other hand also the distribution of the LHC collaborations with some 2000 physicist from some 100 institutes world wide poses a serious requirement on the storage system with respect to data distribution in a heterogeneous networked computing environment.

Since the LHC experiments will exploit OO technology as much as possible, a well designed binding of the storage system to OO languages is very important. The storage system should not impose it’s own data model on HEP applications, but rather be a natural extension of the underlying implementation language. The storage system should not enforce the choice of only one particular language on the whole software system. Even if major parts of the experiment software will be written in a single language (like C++), openness to new market trends in the computing industry is an essential requirement which allows us to exploit new developments in the computer industry in a long term software investment like the experiment software.

Traditional, sequential file base storage packages like ZEBRA do not meet any of the listed requirements. They do not scale to the data volume, they fail to provide a suitable language binding for modern OO languages and they do not comply to any commercially recognised de-jure or de-facto standard. This makes the use of third party software very difficult if not impossible.

2 The RD45 preferred Solution

The solution studied by the RD45 project consists of three major building blocks:

- A commercial Object Database Management System (ODMBS), which stores all experiment and user data and provides in a single logical view of the data for a multitude of platforms and languages.
- A commercial mass storage system like HPSS[1], which is integrated with the ODBMS in a way that allows transparent access to the data from user applications without the need to supply any details on the physical organisation like file, host and device names.
- A largely experiment independent analysis framework which is implemented as an extendable class library and provides HEP specific functionality which is needed

2.1 The ODMG Database Portability Standard

The Object Data Management Group (ODMG) - a group of object database vendors - has defined a portability standard[2], that allows application code to be written in a vendor independent way. The standard defines

- the semantics of a language independent object model;
- an object definition language which describes the object attributes and methods and the relations between different objects;
- a language binding to several OO languages like C++, SmallTalk and soon Java;
- an ad hoc query language (OQL/SQL++).

2.2. Objectivity/DB

On top of those features defined in the ODMG standard, most database vendors implement additional functionality aimed at specific market segments. Objectivity/DB for example provides heterogeneous access to a fully distributed database from more than 14 different operating systems among them all major UNIX and Windows variants together with additional functionality with respect to fault tolerance and high performance in a networked environment:

- Federated database architecture
  The federated database concept combines a multitude of database files - possibly residing on different machines - to a single logical view of related objects. This allows the user to access data from local or remote locations in a single transparent fashion without having to worry about physical storage details. On the other hand it allows the experiment to choose a physical storage layout which achieves the best possible performance for the current access pattern and later on to reorganise this data without compromising existing applications.
- Data Replication to remote sites
  Replication of key data to multiple regional centres to increase availability and performance by keeping local copies of the data. The database keeps those copies up to date also after temporary network problems.
- Schema Evolution
  Support for an evolving data model over time with minimal impact on existing applications.
- Object Versioning
  Support of history keeping on object level and concurrent access for example in cases like calibration data and working group event selections

These features make Objectivity currently a very well suited ODBMS with respect to the data management problems of the LHC era. All RD45 prototyping work is currently base on Objectivity/DB and C++.

2.3 Impact of an ODBMS on HEP Software

To study the impact of the use of an ODBMS on HEP software, we have ported several existing software packages to use the Objectivity database. Our main interest was to evaluate the amount of object model or code changes needed and to compare the transfer performance of the database with traditional I/O packages.

The ODMG object model is in most respects an extension of the implementation language objects model (in this case the C++ object model). We have shown by porting several data stores from existing experiments (among them H1, OPAL and L3) to an ODBMS, that the ODMG object model and the ODL language in particular are very suitable to describe HEP data models.

It should be noted, that the main access method in typical HEP applications is navigational access. An event object typically references other objects like sub-detector data directly, without any associative lookups. In the C++ language this is implemented through the use of pointers or references. A ODMG compliant database extends this model introducing smart pointer classes which implement on demand object I/O in a way transparent to the application. Objects are automatically brought into the application memory as needed whenever they are referenced. We have shown, that the transfer performance (object transfer between disk and application memory) of the database reaches for a suitable object model[1] more than 90 % of the raw device performance.

Since the ODMG approach to implement the access to persistent data through smart pointers is much less intrusive than traditional systems with explicit I/O statements, the amount of code changes needed to make transient C++ applications use an ODBMS was generally relatively small. Some minor changes were needed to establish and control the database session and to migrate from transient container classes to their persistent replacements provided by Objectivity/DB.

In addition to navigational access object databases provide powerful associative access methods like B-trees and hashed dictionaries which speed up the selection of relevant data from the data store. The query language OQL, a superset of SQL extended with respect to the richer object model of ODBMS, is expected to be a useful tool for interactive data selection and to provide an important interface to third party products.

3 Efficient Event Selection based on a ODBMS Data Store

Within RD45 we started to prototype some first ideas for an experiment independent analysis framework. The central problem specific to HEP analysis is the selection of suitable event as input to further analysis. Since traversing the complex data structure within each event in a large data store is a time consuming process, traditionally techniques like “Event Directories” and “Ntuples” have been employed to make the selection problem easier to manage.

The “Event Directory” approach has the important disadvantage to be very sensitive against storage reorganisation (e.g. moving the event data to different hosts or reprocessing the underlying data). The Ntuple technique enforces a completely new data model - a single table - and introduces a new naming scheme. The user thereby loses any system-supported logical relation to the complete event data. Ntuples are also frequently invalidated by new calibrations or reprocessing the event data or just the need for one additional field from the full event data. In all those cases a time consuming recreation of the Ntuple has to happen.

In the RD45 project we have implemented a prototype system which overcomes most of these problems by exploiting the features of an Object Database. The prototype consist of a few

[1] The performance of a particular data model depends to a large extend on model parameters like mean size of the objects and number of relations between them. An more detailed analysis of the object model dependency of the transfer performance will therefore be very important.

experiment base classes which should be specialised to adapt to an experiment specific data model.

- Event and Run
  Mostly abstract base classes which implement the 1-to-n relation between runs and events.
- EventTag
  Extracted copy of important event summary data condensed in a single database file to speed up the selection performance, but still logically attached to the main event data via an association.
- EventCollections
  Collections store a set of events by references and provide an iterator like access.
- EventSelections and SelectionPredicates
  EventSelections are specific EventCollections, which store a set of event defined by boolean selection function encapsulated in a SelectionPredicate object.

Copying the selection relevant information from the main event data hierarchy to a relatively small EventTag objects (<100 bytes) allows us to achieve a very good physical data clustering on disk. The EventTag therefore is used as an important tool to significantly speed up selections that have to be performed very frequently or that access larger amounts of input data. An important benefit of isolating the selection relevant data is that the database file containing the tag objects because of it’s moderate size can always be kept on disk and can easily be kept up to date through the database replication feature. Even though the tag data is physically well clustered, it is in contrast to Ntuple data still logically attached to the main model and can therefore be kept in sync with calibration or reconstruction changes in an automated way. Since EventTags are normal objects of the implementation language, they can contain complex types like four-vectors or matrices, which enables the user to define selection in terms of standard operation on this types.

Even though this framework can make use of EventTag objects, it is not limited to use only tag data. To define a selection, the user simply creates an object which inherits from the abstract SelectionPredicate class. This base class defines the interface to a boolean selection function, which has access to a particular event when the selection is performed. The user is expected to override this function with his specific code and can at the same time define important cut values used by the selection as data members of the concrete predicate. Currently the implementation of such a predicate is possible either as a C++ object, via a selection expression in SQL or - limited to the tag object - via a simple predicate language defined by Objectivity/DB.

Since predicate objects can be made persistent as well as the container with the references to all matching events, physics working groups can share their analysis samples (e.g event classes like “all events with two reconstructed photons”) through the database in a consistent way. Since the persistent selection object retains the information on its input data and its selection predicate, the system is capable of revalidating selections automatically after reprocessing of the input data or even to classifying new data as it arrives in the event data store.

3.1 Implementation of Version Control for Event Selections

A common problem during the iterative refinement of selection cuts in HEP analysis is to keep track of different versions of cuts and the corresponding selected event sets. Lacking other solutions the user typically had to maintain multiple versions of the selection code together with multiple copies of more or less the same data which corresponds to particular versions of the selection code.

Within RD45 we have developed a prototype that uses the object version capability provided by Objectivity/DB to solve this problem: If the user decides to change the selection cut parameters, he can specify to retain the old version of the predicate object and optionally also the references to the selected events. The database will create a new predicate and a new event collection with new cuts and optionally make it the default version of the "physics event class" seen by analysis applications. At the same time other applications can still explicitly choose an older version of the class (e.g. to do consistency checks with old cuts) or they can use a newer version than the default (e.g. to check the quality of the new selection).

4 RD45 co-operation with experiments and research projects

RD45 is working in close contact to the LHC experiments and also with experiments that are currently taking data to understand their data management needs and to help to define their computing model in a way that exploits the services provided by an Object Database. The status of the different projects is summarised in the following.

4.1 Alice

The object model for the ALICE experiment is currently defined on the basis of Objectivity/DB. Persistent classes describing the raw data for the 7 main detectors have been defined comprising an typical amount of data of 40MB per event. A "mathematical event generator" has been developed, which fills the experiment data structure with simulated data of the expected amount. The main aim of this project is to check the consistency of the model and to do detailed performance studies with the database based on this data model.

4.2 CERES/NA45

The CERES/NA45 collaboration has recently redesigned their complete reconstruction and filtering software based on C++. The package consisting of more than 30 k lines has been ported to use an ODBMS store. The system has been used to write from multiple processing nodes in parallel into one store. In the next step 60 GB of reprocessed data will be stored in an Objectivity database.

4.3 GEANT 4

A first proof-of-concept prototype has been developed that stores persistent hits in an Objectivity database. More detailed object model design studies will follow to optimise the storage and runtime usage.

4.4 Atlas & CMS

RD45 works in close contact with the ATLAS and CMS computing model working groups to help defining the experiment data model and software infrastructure.

4.5 Future Plans

In a joint project together with Digital and Objectivity we will set up a performance test-bed consisting of multiple high-end Alpha servers and up to 100TB of disk. The main goal is to perform

- a detailed analysis of data transfer and query performance,
- a comparison with conventional Ntuple based analysis tools like PAW
- a detailed analysis of object model dependency of performance,
- scalability tests up to 100 TB.

5 Conclusion

Commercial Object Databases provide a coherent, logical view to a distributed data store expected to scale up to the PB region. The abstraction from the physical storage layout and the implementation language will be an important tool for optimising the data clustering and for adapting to changes in the analysis practice. ODBMS do not impose unacceptable performance penalties and do not constrain the choice of language, platform and object model. A HEP specific layer on top of an ODBMS can implement an efficient data selection that is applicable to batch and interactive analysis and might completely remove the need to produce Ntuples.

References

1. The *High Performance Storage System - HPSS*, **`http://www.sdsc.edu/hpss/hpss.html`**.
2. ODMG-93,*The Object Database Standard*, Edited by R.G.G. Catell, Morgan Kaufmann (1996).