OBJECT ORIENTED DATABASES IN HIGH ENERGY PHYSICS

Pavel Binko
Lawrence Berkeley National Laboratory, California, U.S.A.

Abstract
High Energy Physics (HEP) community in Large Hadron Collider (LHC) era is in transition phase from FORTRAN to C++ and from data streams to persistent objects. Together with it new data management will be necessary, which would allow to leave filenames and tape numbers and access data in form of objects only selecting their required physical contents. One solution is using Object Database Management Systems (ODBMS). Due to vast volumes of data in HEP, an interface between ODBMS and Mass Storage Systems (MSS) is necessary.

1 INTRODUCTION

C++ [1] is an object oriented programming language, which is very suitable for description of complex data structures in HEP. Main building element in C++ are **classes**, they encapsulate data types and member functions.

Objects are variables of class types. **Transient objects** are typically created by a process and cease to exist when process terminates (or earlier). **Persistent objects** continue to exist upon process termination and may be accessed by other processes. Persistent objects can be stored in an ODBMS.

There are many types of objects to be stored, like detector calibration and geometry data, production control data, histograms, but the most important and most difficult is **event data**. All LHC experiments [2] expect to have enormous data volumes and rates.

TIME INTERVAL	VOLUME OF DATA	
	ATLAS [3] / CMS [4]	ALICE [5]
	1 MB / event, 100 Hz	40 MB / event, 40 Hz
1 second	100 MB	1.6 GB
1 minute	6 GB	100 GB
1 hour	360 GB	6 TB
1 day	8,6 TB	140 TB
1 week	60 TB	1 PB
1 month	260 TB	
	1 PB useful information per year	1 PB useful information per year

LHC Data Volumes and Rates

2 REQUIREMENTS FOR ODBMS

An ODBMS used in HEP has to satisfy these requirements:

- Standard compliance - Object Database Management Group (ODMG)
- Scalability – 100 GB to 1 TB per database file, total size many PB
- Heterogeneous environment
- Distributed application
- Data replication
- Schema evolution
- Object versioning
- Language heterogeneity
- MSS interface

3 OBJECT DATABASE MANAGEMENT GROUP (ODMG)

The ODMG [8] is a consortium of ODBMS vendors and interested parties working on standards to allow portability of customer software across ODBMS products. The ODMG-93 (current version 2.0) standard is based on existing SQL-92, OMG and ANSI programming language standards. Its functional components include:

- Object Definition Language (ODL) makes possible to describe a database schema independent of the programming language thereby making the schema portable between compliant databases. The ODL syntax is very similar to the C++ one.
- Object Query Language (OQL) is an SQL like declarative language that provides efficient querying of database objects, while retaining compatibility with the SQL-92 SELECT syntax.
- C++ and Smalltalk Bindings (Java Binding in development) extend the ANSI standards to support manipulation of persistent objects, OQL and transactions

From these standards HEP is currently using ODL and C++ language binding.

4 OBJECTIVITY/DB

There are currently nine voting members of ODMG – ODBMS vendors. The project RD45 [7] at CERN [2] investigates in comparison of best candidates and to solve persistence in HEP Objectivity Inc. [9] has been chosen. Its Objectivity/DB product group, which consists of Objectivity/DB, Objectivity/DB Fault Tolerant Option (Objectivity/FTO) and Objectivity/DB Data Replication Option (Objectivity/DRO), fulfills most of our requirements.

4.1 Objectivity/DB

Objectivity/DB is a distributed ODBMS, is ideal for applications that require complex data models, supports large numbers of users and provides high performance access to large volumes of physically distributed data. It integrates easily with application software, allows you to directly store and manage objects through standard language interfaces including C++, Smalltalk (Java soon), and ANSI SQL using traditional programming techniques and tools.

Objectivity/DB supports all major hardware platforms (Apple Macintosh, Digital, Hewlett-Packard, IBM, Intel, NEC, Silicon Graphics, and Sun Microsystems), and operating systems (MacOS, UNIX, VMS, Windows NT and 95). Full interoperability across these platforms is provided, allowing data created by applications on 32-bit NT platform can be read, updated, or deleted by applications on 64-bit UNIX platform.

Fundamental unit of storage is a **basic object**. It may contain scalar types, non-persistent C++ objects and structures, fixed-size and variable-size arrays, associations to other objects, and object references to persistent-capable classes. Each basic object is contained in a container.

Container is a collection of basic objects, they are physically clustered together in memory pages and on disk. Access to all basic objects in a single container is very efficient. Since locking takes place at the container level, right **object clustering** based on object access patterns may considerably improve the performance. Each container is contained in a database.

Database is a collection of containers. It contains also a system created default container and user-defined containers. The default container holds basic objects that you have not explicitly placed in user-defined containers. Databases physically map to files. Each database belongs to a federated database.

Federated database logically contains user-defined databases and the data model (schema) that describes all class definitions. Most administrative control is on the federated database level, including configuration information (where Objectivity/DB files physically reside) and concurrent access control.

Objectivity/DB provides three types of server processes – lock server, local database server, and remote database server. The **local database server** is typically linked with the application. The **remote database servers** use a page server to provide optimal performance over a wide range of data loads. Page servers typically outperform object servers by providing significantly higher transfer rates.

Objectivity/DB allows simultaneous access to multiple objects. To ensure data consistency, database access is managed by **lock server** through the use of locks. A process can obtain either a read lock, which allows other processes to read the objects, or an update lock, which prevents all other processes from reading or modifying the objects. To achieve high concurrency multiple readers one writer (MROW) mode can be used. For each federated database locks are managed by a lock server. For federated databases that are partitioned using Objectivity/FTO, there is one lock server per partition. Locking at three levels of granularity is supported – federated database, database, and container.

ODBMS support **data models** of high complexity. Data modeling with Objectivity/DB is similar to modeling in non-database applications. You create your data model and decide how those objects should behave and interact. In addition you specify, which objects you want to store in the database and relationship between objects.

Schema evolution allows changes to be done in the database schema. When the schema is changed existing objects of the changed class may need to be converted to reflect the changes. The object conversion can be dome in Deferred Mode, On-Demand Mode, or Immediate Mode with an upgrade application.

Object versioning allows to maintain multiple copies (versions) of the same basic object. The application can create separate versions of an object and track its genealogy over time. It can perform linear or branch versioning of objects, and can specify a particular version as the default version.

Objectivity/DB supports both persistent and transient **Standard Template Library (STL)** class libraries with Objectivity/C++ (based on the ObjectSpace Standards <ToolKit>). It enables rapid development and deployment of applications based on standard interfaces.

4.2 Objectivity/DB Fault Tolerant Option

Objectivity/FTO supports fault tolerance and improves performance for large database deployments. It improves data availability after network failures by distributing lock management, and replicating the database schema and catalog to multiple locations in a network. Using Objectivity/FTO federated database can be divided into autonomous partitions. Each partition is self-sufficient and insulated from network and other system failures.

4.3 Objectivity/DB Data Replication Option

Objectivity/DRO improves data availability and enhances read performance by replication a subset of data in multiple locations in network. It can also eliminate the danger of a single point of failure to the database. When an application accesses objects in a replicated database, Objectivity/DB locates the closest accessible replica. This improves the performance of the application because of reduced

network latency and disk contention. With Objectivity/DRO, it is possible to

- Create multiple replicas of a database across a heterogeneous LAN/WAN.
- Provide full read/write access to replicated databases. When one database image is updated, other replicas are automatically and transparently updated as well.
- Dynamically determine write privileges to replicated databases.
- Access another replicated database if part of the network becomes unavailable (it can be used also for automatic backup).

5 MASS STORAGE SYSTEMS

It is expected to have many PB of data per year in LHC era. Although significant increase of disk sizes can be expected, we cannot expect multi-PB disk farms soon. We therefore need some way of combining ODBMS and MSS [10]. The only one choice from existing MSS is currently High Performance Storage System (HPSS) [11]. HPSS has been developed in collaboration of IBM, LANL, LLNL, NERSC, ORNL, SNL and many others.

HPSS provides scalable parallel storage systems for highly parallel computers as well as traditional computers and workstation clusters. It is designed for large storage capacities, and to transfer data at rates up to multiple GB per second. The data can be transferred using these user interfaces:

- File Transfer Protocol (FTP)
- Parallel File Transfer Protocol (PFTP)
- Network File System Version 2 (NFS V2)
- IMB SPx Parallel File System (PIOFS) Import / Export

Integration between Objectivity/DB and HPSS will be realized at the end of 1997.

6 CONCLUSIONS

Object Database Management Systems open new era in data management in HEP. ODBMS allow different level of access to the experimental data. Users will stop to worry about file names and tape numbers. Users will select the data according its physical contents and characteristics. It will considerably improve the whole physical analysis process and will have big consequence for HEP.

REFERENCES

1. Ellis M. A., Stroustrup B., The Annotated C++ Reference Manual, Addison-Wesley, 1992, ISBN 0-201-51459-1
2. CERN, http://www.cern.ch/
3. LHC, http://wwwlhc01.cern.ch/
4. ATLAS, http://atlasinfo.cern.ch/Atlas/
5. CMS, http://cmsinfo.cern.ch/cmsinfo/
6. ALICE, http://www.cern.ch/ALICE/
7. RD45, http://wwwcn.cern.ch/asd/cernlib/rd45/
8. ODMG, http://www.odmg.org/
9. Objectivity Inc., http://www.objectivity.com/
10. MSS, http://wwwcn.cern.ch/~hepmss/
11. HPSS, http://sdsc.edu/hpss/