Objectivity/DB Performance and Scalability

Dirk Düllmann
RD45 Collaboration
CERN IT/ASD
Geneva, Switzerland

In response to the LCRB milestones, the CERN RD45 project has made a detailed study of the performance and scalability of the Objectivity/DB ODBMS product, including a comparison based upon the use of the ODBMS as input to physics analysis version the traditional approach of using PAW and Ntuples. Given that the LHC experiments will eventually require a system capable of storing many tens of PB of physics data, and early demonstration of the scalability of any proposed solution is fundamental. This paper discusses the results obtained in a first one to one comparison with PAW.

Key words: RD45, Object Persistency, ODBMS, Objectivity/DB

1 Introduction

The RD45 project at CERN is aimed to provide an persistent data store for LHC experiments. About 1PB of event data per experiment and year have to be stored and retrieved for analysis by physicists distributed at some 100 institutes world wide.

The solution proposed by RD45 consists of three major building blocks[1]:

- A commercial Object Database Management System (ODBMS) compliant with the ODMG[2] portability standard which stores the experiment data and provides in single logical view of the data on a multitude of platforms and languages.
- A commercial mass storage system like HPSS, which is transparently integrated with the ODBMS without the need for user applications to supply any details on the physical organisation like file, host and device names.
- An experiment independent analysis framework which is implemented as an extendible class library and provides HEP specific functionality.

The main component of the analysis framework which provides tools for efficient event selection is the tag database (TagDB). This database contains an extracted copy of important event summary data condensed in a single physical file. In contrast to conventional event directories or Ntuple files the TagDB approach uses the services of the ODBMS to maintain a bi-directional, logical association to the main event data hierarchy.

A complete overview of the RD45 project can be found in [3–6].

2 Performance Comparison with PAW and Ntuples

Below, we describe performance comparisons between PAW and Ntuples and simple TagDB implementations. The approach has been to perform one to one comparisons between Ntuples and TagDB implementations using Objectivity/DB. To this end, we have used a standard NA45/CERES Ntuple from their 1995 production data. The same analysis has been performed using both PAW+Ntuples

Preprint submitted to Elsevier Preprint *4 March 1997*

and Objectivity/DB, under a variety of different cache conditions. The benchmark environment used in both cases was identical and is described in detail in [6].

In all cases, we have measured the both first-pass (”cold”) and second-pass (”hot”) cases. At the time of writing, we have more confidence in the hot measurements - issues such as the file system cache can strongly influence these performance measurements and, short of rebooting a CS-2 node between each measurement, it is hard to be certain that no caching is taking place for the first-pass measurements.

The NA45 Ntuple used contains 302 columns (all floats) and some $21K$ rows, giving a total size of around $25\,MB$. This is seen to be a somewhat typical size for Ntuples today, although they are often combined into larger logical units using the PAW chaining facility. The main time during the PAW-based analysis is spent in a single command, namely:

```
ntuple/loop [ntuple] ana.f
```

`ana.f` is a single, compiled, Fortran function that performs all section cuts and histogramming. In the current analysis, some 15 columns are typically used to make selections, although this is expected to rise so that eventually 80 columns are used. Both the original Fortran version of this function and the equivalent C++ code are reproduced in [6].

In the tables below, the time shown is that spent on performing the event selection, including the time to access the attributes used in the selection and those attributes used for the histogramming. The time spent in filling and displaying histograms has not been measured, as this is independent of the database.

2.1 Full Tag Comparisons

The first tests were based upon an implementation whereby each row in the NA45 Ntuple was converted to a separate ”tag” object, i.e. an object with one attribute corresponding to each column in the Ntuple. This means that no traversals are required to access additional objects to perform the query or to fill the histograms. All the tags were stored in a single container in an Objectivity/DB database, and clustered according to insertion time. A compiled, user-written selection function was used in both cases. The time taken to compile and load these functions, and the time taken to fill the histograms, has been subtracted from the values shown.

Analysis using	Time	Comments
PAW + RWN	$11.3\,s$ cold $2.5\,s$ hot	First pass - 0% cache efficiency Second pass - 100% percent cache efficiency
PAW + CWN	$16.4\,s$ cold $2.6\,s$ hot	Converted using htonew
TagDB	$6.2s$ cold $1.4s$ hot	

Table 1
Comparison of NA45 Ntuple with the “FullTag” TagDB Implementation

At the time of writing, the reasons why the column-wise Ntuples show worse performance in the first-pass case are not fully understood, but are being investigated in collaboration with the PAW support team in IT/ASD. Although, in this particular case, the TagDB implementation based upon Objectivity/DB shows better performance, we interpret these results as showing comparable performance, pending further, more detailed, investigations. It is fair to say, however, that these first, unoptimised results, are encouraging, particularly when one considers the significant amount of effort that has been spent on optimising the performance of PAW.

2.2 Reduced Tag

A slightly more complex implementation than the case described above is one where the subset of the data used for the selection is stored separately from that used in the analysis of the selected events. In this case, two objects are used - one containing the 15 "columns" used in the selections and the other containing the remaining data. These objects were clustered separately and were stored in different physical databases in the same federation. Only in the case that an event is selected is a traversal made from the reduced tag object to the remaining event data.

Tag Implementation	Time	Comments
Full Tag	6.2 s cold 1.4 s hot	No traversal
Reduced Tag - 0% selectivity	1.3 s cold 1.0 s hot	No traversal
Reduced Tag - 2% selectivity	5.5 s cold 1.2 s hot	One traversal per selected tag
Reduced Tag - 20% selectivity	7.5 s cold 1.5 s hot	One traversal per selected tag

Table 2
Comparison of Different TagDB Implementations

In the above table, the performance of the reduced tag, in the case of low selectivity, improves with respect to that of the full tag simply due to the decrease in I/O that is required. As no events are selected, only the tag objects are read in. As the selectivity rises, we see the effect of object clustering. The objects correspond to a selected event may well have been brought into the client cache as a result of a previous I/O. The potential benefit is very dependent on the object size and page size. Results from more detailed tests using different object and page sizes can be found in [6].

3 Queries Using Indices

Further optimisations can be achieved by introducing indices on the tag objects, either for the full or reduced tags. Objectivity/DB uses a B-tree with short OIDs, which are 4 bytes rather than the usual 8 bytes. This implies that they can only refer to objects within the same container, although Objectivity/DB V4 also provides federation-wide indices, based upon normal OIDs.

The storage overhead of a single index entry is the size of the attribute together with a 4 byte overhead. Thus, a single 8KB database page can store 1000 object references indexed on a single 32 bit field.

Unfortunately, it was not possible to include performance measurements using indices in this paper. However, detailed measurements can be found in [6], which is available via the Web.

4 Conclusions

An ODBMS approach offers the possibility of revolutionising our approach to physics analysis - offering not only more efficient access to the data, but also permitting more complicated analyses to take place. Initial tests show that comparable performance to today's systems can be achieved by even naive approaches, and that separation of data into the part required for selection and that used for analysis (i.e. reduced tag) offers performance improvements for sufficiently selective queries. Only minor performance enhancements have been made so far and it is expected that significant performance optimisations can be made with time. The ease of access and transparency to all of the event data has been demonstrated, and the cost of traversing associations to the event objects shown to be small. The above results have been based on initial measurements and only preliminary interpretations can be made at the time of writing. More complete information is available in the document [6].

References

[1] The RD45 Collaboration, *RD45 - A Persistent Object Manager for HEP*, LCB Status Report, March 1996, CERN/LHCC 96-15

[2] ODMG-93, *The Object Database Standard* Edited by R.G.G. Catell, Morgan Kaufmann (1996).

[3] The RD45 Collaboration, *RD45 - A Persistent Object Manager for HEP*, LCB Status Report, March 1997, CERN/LHCC 97-6

[4] The RD45 Collaboration, *Object Database Features and HEP Data Management*

[5] The RD45 Collaboration, *Object Databases and their Impact on Storage-Related Aspects of HEP Computing*

[6] The RD45 Collaboration, *Using and Object Database and Mass Storage System for Physics Analysis*