Experience using different DBMSs in prototyping a Book-keeper for ATLAS' DAQ software

I.Alexandrov[1], V. Amaral[2], A.Amorim[2], E.Badescu[3], D.Burckhart-Chromek[4], M.Caprini[3], M.Dobson[4], R.Hart[5], R.Jones[4], A.Kazarov[4,9], S.Kolos[4,9], V.Kotov[1], D.Liko[4], L.Lucio[2,4,7], L.Mapelli[4], M.Mineev[1], L.Moneta[6], M.Nassiakou[4], N.Parrington[7], L.Pedro[2], A.Ribeiro[2], Yu.Ryabov[8], D.Schweiger[4], I.Soloviev[4,9]

1) Joint Institute for Nuclear Research, Dubna, Russia
2) FCUL (Science University of Lisbon), Lisbon, Portugal
3) Institute of Atomic Physics, Bucharest, Romania
4) European Organization for Nuclear Research (CERN), Geneva, Switzerland
5) National Institute for Nuclear Physics and High Energy Physics (NIKHEF), Amsterdam, Netherlands
6) Physics Section, University of Geneva, Geneva, Switzerland
7) Sunderland University, Sunderland, England
8) Petersburg Nuclear Physics Institute (PNPI), Gatchina, St. Petersburg, Russia
9) On leave from 8.

Abstract

The Online Book-keeper (OBK) was developed to keep track of past data taking activity, as well as providing the hardware and software conditions during physics data taking to the scientists doing offline analysis. The approach adopted to build the OBK was to develop a series of prototypes, experimenting and comparing different DBMS systems for data storage. In this paper we describe the implemented prototypes, analyse their different characteristics and present the results obtained using a common set of tests.

Keywords: Online Book-keeper, ATLAS, Trigger/DAQ, ODBMS, RDBMS, Online Databases

1. Introduction

The Online Software in the ATLAS experiment [1] is the "glue" for all the other sub-systems of Trigger/DAQ in this LHC's detector, since it monitors/supervises all of their activity. It includes the OBK component that archives the data to be kept from the control of the online system and the information about the data recorded to permanent storage, storing information about each run that occurs. This information includes a selection of the messages passed between the Online Software components. The Online Software components may communicate via the Message Reporting System (MRS) and via the Information System (IS). Both MRS and IS provide a subscription mechanism that is used by the OBK to access and store messages online while they are being interchanged.
Besides the online data taking application, the OBK also makes available offline data retrieval tools. Users are provided with a Web-based data browser and a C++ API (Application Programming Interface), which is used to access the data collected in present and past ATLAS test beams.

2. Implementation

We developed several implementations of the OBK [2], based on different DBMS systems (Objectivity/DB [3] , OKS [4] and MySQL [5]). The aim was to study their different characteristics, evolving the design from prototype to prototype, and providing different solutions that could be used in different Online Software environments. For example, the Objectivity/DB based solution requires a commercial licence, while the OKS and MySQL implementations can be generally used. We attempted to use a similar structure for all implementations but variations in the facilities for schema definition, programming API' s and data clustering option, resulted in substantial differences in the prototypes.

2.1. Generic architecture

All the OBK prototypes implement a data acquisition application "obk_daq" (see Fig. 1) that subscribes to relevant MRS and IS servers in order to receive the MRS/IS messages exchanged between components via CORBA/IIOP [6] callbacks. It then stores these messages associated with each run. The information includes when did the run start and end, if it was successful, what were the basic physics parameters, etc. This selection and organization involves a significative amount of logic that has to be resolved without disturbing other Online Software components.
The stored data is then made available offline to users by means of a Web-based browser or a C++ API. The Web browser enables the user to go through different levels of abstraction of the data (Partition/Run-Header/Messages) and to visualize it in HTML tables. The data extraction for the web is accomplished using a C++ application "obk_dump" that outputs ASCII data to PHP scripts for HTML generation. The possibility of using URL links to navigate between different levels of abstraction makes the browser useful for quick searches.
The C++ API on the other side provides a much more detailed interface to the data, making it possible for an application, for example, to search for instances of a particular IS class parameter or to cycle through all the run headers in a partition.

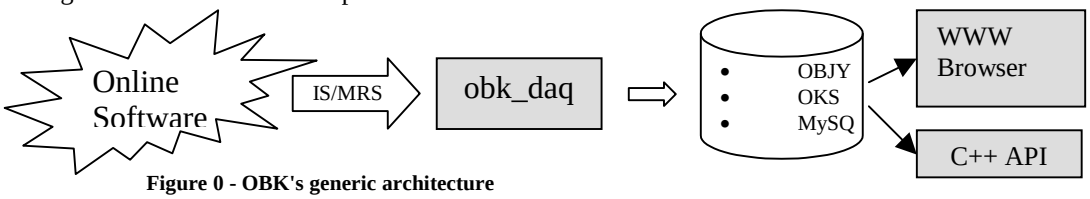

Figure 0 - OBK's generic architecture

2.2. Implemented prototypes

The OBK component of the online software is an interface to the offline analysis environment and is embedded in the TDAQ distributed system. It provides a laboratory to test the use of different Database Management Systems that are candidate to general solutions for the HEP community. The Objectivity and OKS prototypes are using the ATLAS standard offline and online database solutions. A third prototype was build using an Open Source solution to evaluate the use of the common functionality of the relational database systems.

2.2.1. Prototype based on Objectivity/DB

The Objectivity based prototype was the first to be developed as it uses the DBMS chosen inside the ATLAS Software for long-term data management. The Objectivity/DB databases for OBK are stored centrally in database server and accessible by means of an AMS (Advanced Multithreaded Server) daemon running on the server machine. The approach for storing book-keeping data in Objectivity/DB was to "break" the messages in pieces and store them in associated but distinct persistent objects (e.g. each IS message parameter is stored in a different object). The online partitions are mapped to different databases and runs are associated to containers.

2.2.2. Prototype based on OKS

The second OBK prototype to be implemented uses OKS as a persistency system. OKS is a light-weight C++ based in-memory persistent object manager developed as a package within the Online Software framework. It is primarily used within the Online Software for managing the configuration databases. The fact that it is *Open Source* software, avoids the problems related with licensing faced with Objectivity. Also some of the advantages of the OKS approach are that this DBMS is lighter and

more oriented for real-time processing than Objectivity. The disadvantages are that it is not a full scale DBMS and doesn't feature all the functionalities, such as support for transactions. Another interesting point of the OKS approach is that the data can be stored in XML format, therefore making it human readable and highly portable. Given that OKS is also an Object Oriented DBMS, the persistent object schema follows the one implemented for the Objectivity version.

2.2.3. Prototype based on MySQL

The most recent implementation of the OBK prototype uses the MySQL *Open Source* DBMS. MySQL is a relational database system as opposed to the Object Oriented model used before. The mapping of the OBK in MySQL took some rethinking of the interface with the DBMS, that was changed considerably to be entirely SQL based. We have achieved a mapping between an OO and a relational database schema that was suitable for the OBK needs.

2.3. Design and implementation experience

While designing and implementing the various prototypes we discovered both positive and negative aspects of each of the DBMS and could appreciate some of the benefits/disadvantages of using an OO or a relational approach. Using Objectivity/DB and OKS benefited from a natural integration of the application code with the object database system. The schema mechanisms of both DBMS are powerful and flexible, and allow a direct mapping of application objects with the database objects. What was found less attractive was the complexity of the code involved, as well as the difficulty to tune DBMS accesses. The code is very dependent of the specific object database implementations.
While the Objectivity based prototype does not cluster together the run summary information but spreads it over all the run containers, the OKS prototype defines a special XML file containing the run summary information for all runs. This overcomes the need to parse many XML files to select runs based on certain properties. The results presented are positively influenced by this optimisation.
The advantages of using MySQL are that it is very easy to develop a database application – easy to define the schema, easy to make SQL queries – and the code quantity/complexity involved in implementing the queries was much reduced.
We found less flexibility in mapping the software objects to the database schema provided by the relational model. This was partially overcome by storing complex objects in “XML like” format as strings in the database. All the parameters that are foreseen to be used in selection criteria were made explicitly available as table attributes. This is also influencing the results in a positive fashion.

3. Test Procedure

To evaluate the performance of the different prototypes we have defined a set of tests that cover both storing the information in the OBK databases and performing analysis queries. The considered examples either retrieve the information in one run (Query 1) or selects a set of runs with a given criteria, fulfilled by 5% of the runs (Query 2). The tests for the three OBK prototypes were performed using PIII PC client and server machines running Linux. The databases were populated with 1000 Run’s for each prototype.

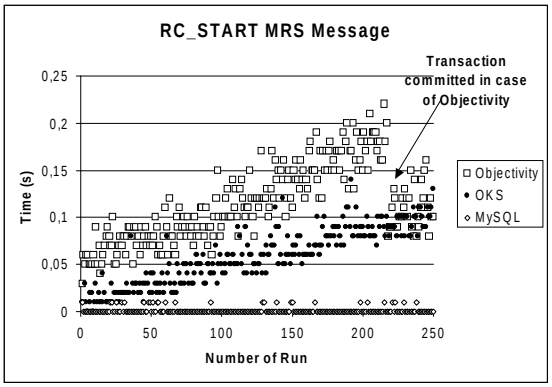

Figure 2 - The time taken to store and process the start of run MRS Message.

4. Results

The time taken to store and process the start of run MRS Message is presented in Fig. 2. The start of run is used to create the containment for the new run and setting the appropriate variables. The Objectivity and OKS prototypes show an increase of the storing time with successive runs. In the Objectivity case this trend can be overcome by committing the transaction.
The Table 1 presents the necessary disk space and shows the average query time for queries performed directly in the database server machine and remotely using a different computer.

	Space in Mbytes	**Elapsed Time (s)**				**User + system Time (s)**			
		Query 1- Loc./Rem		Query 2- Loc./Rem		Query 1- Loc./Rem		Query 2- Loc./Rem	
Objectivity	63.4	0.13	0.94	18.13	116.08	0.12	0.12	8.74	8.7
OKS	3.7	0.38	1.42	0.35	1.18	0.38	1.42	0.34	1.18
MySQL	1.1	0.39	0.70	0.02	0.08	0.03	0.07	0.02	0.05

Table 1: The space required in Mbytes and the average query time in seconds for different prototypes for a sample database of 1000 runs.

5. Conclusions

The ATLAS OBK was used to perform a comparative study of different DBMS solutions. The performance results can be seen as favourable to the MySQL prototype and less optimal for the Objectivity solution. One must however take into account that the queries performed benefited from the fact that no iteration was required through the collections of XML files/strings since they were made available in MySQL table attributes or in the OKS run summary file. The rather large remote access time for the query 2 in Objectivity can be seen as resulting from downloading information from all run containers through the network. Implementation of the query using predicate queries and indexed parameters seemed not to improve the situation. In contrast, browsing a single container for the query 1 is very fast for this object database, especially taking into account that it is the only prototype to implement transactions for database processing. Another relevant point is the big storage space required by Objectivity, as opposed to that required by OKS and MySQL. We interpret this difference as being the overhead introduced by all the extra functionality and scalability possibilities available in Objectivity, at least in what concerns the OKS DBMS.
The OBK is work in progress. We plan in the long term to be able to test all the prototypes more thoroughly and in heavier memory/disk usage situations, in order to better check their scalability possibilities.

6. Acknowledgements

We would like to thank Beniamino Di Girolamo for his useful comments and RD Schaffer for his help in the Objectivity implementation.

7. References

1. "ATLAS High-Level Trigger, DAQ and DCS Technical Proposal", CERN/LHCC/2000-17
2. "Design of the Run Bookkeeper System for the ATLAS DAQ prototype –1", A. Amorim, H.Wolters, 1997. See http://atddoc.cern.ch/Atlas/Notes/049/Note049-1.html
3. Objectivity home page, see http://www.objectivity.com
4. "OKS User's guide", I. Soloviev. See http://atddoc.cern.ch/Atlas/Notes/033/Welcome.html
5. MySQL home page, see http://www.mysql.com
6. "Use of CORBA in the ATLAS DAQ Prototype", A. Amorim et al., IEEE Transactions on Nuclear Science, vol.45, No 4, August 1998