Large Scale Access Tests and Online Interfaces to ATLAS Conditions Databases

A Amorim\textsuperscript{1}, L Lopes\textsuperscript{1}, P Pereira\textsuperscript{1}, J Simões\textsuperscript{1}, I Soloviev\textsuperscript{2}, D Burckhart\textsuperscript{3}, J Von Der Schmitt\textsuperscript{4}, M Caprini\textsuperscript{5} and S Kolos\textsuperscript{6}

\textsuperscript{1} SIM and FCUL, University of Lisbon, Campo Grande, 1749-016 Lisbon, Portugal
\textsuperscript{2} Petersburg Nuclear Physics Institute, San Petersurg, Russia (PNPI)
\textsuperscript{3} CERN - Geneva, Switzerland
\textsuperscript{4} Max-Planck-Institut für Physik, Garching, Germany
\textsuperscript{5} Horia Hulubei National Institute for Physics and Nuclear Engineering (IFIN/HH Bucharest), Măgurele, Romania
\textsuperscript{6} University of California, Irvine, California 92697-4575, USA

E-mail: Antonio.Amorim@sim.fc.ul.pt

Abstract. The access of the ATLAS Trigger and Data Acquisition (TDAQ) system to the ATLAS Conditions Databases sets strong reliability and performance requirements on the database storage and access infrastructures. Several applications were developed to support the integration of Conditions database access with the online services in TDAQ, including the interface to the Information Services (IS) and to the TDAQ Configuration Databases.

The information storage requirements were the motivation for the ONline ASynchronous Interface to COOL (ONASIC) from the Information Service (IS) to LCG/COOL databases. ONASIC avoids the possible backpressure from Online Database servers by managing a local cache. In parallel, OKS2COOL was developed to store Configuration Databases into an Offline Database with history record. The DBStressor application was developed to test and stress the access to the Conditions database using the LCG/COOL interface while operating in an integrated way as a TDAQ application.

The performance scaling of simultaneous Conditions database read accesses was studied in the context of the ATLAS High Level Trigger large computing farms. A large set of tests were performed involving up to 1000 computing nodes that simultaneously accessed the LCG central database server infrastructure at CERN.

1. Introduction

The technological trend to exploit the parallel nature of event processing in High Energy Physics using very large processing farms, not only implies a growing usage of database management systems but also poses very demanding problems to the database server infrastructure.

In the ATLAS offline system, both prompt and delayed reconstruction is to be performed using the Athena reconstruction framework [14] on a GRID-immersed infrastructure. While processing the events, not only the Conditions data, but also the detector description and all data-taking control and monitoring data, should be accessed from the database server infrastructure by selecting the right versions for each specific run and event. The offline access to the databases is simplified by the fact that a common Athena framework is to be used for all applications. This framework internally defines the data schema through the mapping to the
transient classes performed by the Athena Conversion Services. Another useful feature results from the fact that the reconstruction jobs will typically not run in a synchronous manner, thus smearing over time the load to the database servers.

The ATLAS online DAQ and Trigger processing, while also involving a huge number of processing nodes, is rather different in nature, since different approaches were necessary to cope with both the fast processing of the first and second trigger levels and the complex processing of the Event Filter. The online infrastructure has been developed on top of a common distributed control and configuration system. A state machine is part of the online software, bringing some level of synchronization to all the infrastructure.

Figure 1 includes a representation of the online interfaces to the ATLAS Conditions Databases.

The definition of “configure” and “running” states makes it natural for the online configuration database infrastructure to be created in the memory layer of each node, in the beginning of data taking, both concentrating all possible delays before the system is running and becoming extremely fast to access during the running period. The configuration databases also provide an object interface to the data that is much more powerful than the one provided by the typical relational database systems. This infrastructure can be applied to all data that is infrequently updated during each run. We describe in Section 2 the tools to make the configuration information accessible offline.

A large part of the information that flows on the online software control bus, in particular the Information Service data, is also necessary for the complete offline interpretation of the event stream, and must be copied to the central database infrastructure. Minimizing the possible delays during data taking also requires developing a special solution to store the flow of online information to the COOL [15] databases. The ONASIC package, described in Section 3, collects this information in an asynchronous way, avoiding the backpressure from the central database servers on the online communication bus.

In spite of the mechanisms introduced above, a substantial fraction of the data required by the trigger event processing involves calibration and detector description information that must be accessed online from the central database infrastructure in a quasi-synchronous way. A benchmark tool was developed to reproduce these data access patterns and to evaluate the performance of the developed interfaces and the central database management infrastructure. This tool is described in further detail in Section 4, together with some operation results obtained.

2. Storing Online Configuration Information
TDAQ partitions, which are instances of the Configurations database [2], are expected to change often in the beginning of forthcoming ATLAS data-taking activity. Partitions will be generated

Figure 1. Online interfaces to Conditions database context diagram
on-the-fly to be used in a set of runs, and can be made persistent through the Online Kernel System (OKS) [3] XML files. The use of COOL Conditions Database model to give OKS partitions persistency other than XML files was long standing.

The first tool that was developed, **oks2cool**, is a direct interface that used the OKS and COOL API's and mapped data from OKS directly into COOL. Later, **oks2tidb2** or **oks2cool2** was developed, this new solution is enhanced by using the Temporal Instrumental Database (TIDB2) [1] as intermediate layer. The TIDB2 Plugin architecture allows not only a wider range of operation modes for **oks2cool2** but also making it much more simple in terms of implementation of further functionalities.

### 2.1. OKS2COOL and OKS2COOL2

**oks2cool** used the OKS API to load and manage the tdaq partition (set of XML files), it first gets the OKS schema (list and definitions of OKS Classes) to define the COOL FolderSet Schema (definition of tables in the database), after this it loads the OKS objects, prints a Time Stamp and Channel ID (set of rows in a COOL table, each having a unique Interval of Validity) attributes for each object, maps OKS Data Types into COOL Data Types, and starts to create the COOL data payload. When the list of OKS objects reaches the end, the payload is flushed into COOL database. **oks2cool** used the simplest OKS/COOL data mapping (see Fig. 2). The FolderSet (directory of tables in COOL) name is user defined for allowing the user to define its own database organization. A COOL Folder (database table) is created per each OKS Class, and a Folder row is created per each OKS Object. The COOL Channel ID is the MD5 of the string containing the “user_name” plus “partition_name”, thus avoiding data overlapping in the database.

**oks2cool2** was successfully used since tdaq-01-06-00 release (May 2006).

The new application **oks2cool2** has a different architecture from **oks2cool**, as it uses TIDB2 Plugin architecture: COOL Connection Plugin, OKS Data handling Plugin and TypeLink Plugin; **oks2cool2** became much simpler, letting TIDB2 do all the underlying work. **oks2cool2** works by using TIDB2 to open and load the partition XML files, a TIDB2 database is temporary built in memory, and when all OKS objects have been loaded, TIDB2 flushes all Tables into COOL.

### 2.2. Dealing with Schema evolution

**osk2cool2** uses a similar data mapping as **oks2cool** but is has an extra feature which allows to deal with the schema changes problem which arises from the need to move objects with different data schemas into the same database.
oks2cool2 allows for each object to be also stored as a binary, using TIDB2 OKS Extended Type Plugin. The OKS object integrity is always preserved in the contents of its binary record in the database. Figure 3 shows the oks2cool2 data mapping where this binary object is present in the COOL Folder.

This solution is possible through the use of a mechanism present in the TIDB2 OKS Extended Type Plugin which contains a streamer. This streamer first converts the OKS object into a TIDB2 BLOB object. In fact it is encoding the OKS object (all its attributes and relations) into a string in such a way that the XML formatting is easily interpreted by front end application such as KTIDBExplorer [16] (TIDB2 database browser) (see Fig. 4). After the TIDB2 BLOB object has been created, the streamer goes through all of the object’s attributes, and in case the TIDB2 Table Schema has a column for such attributes, the streamer also writes the corresponding
attribute or relation value into the respective column. This way it is possible to have in the same database tables a collection of objects of different schema versions. Later it is possible to reconstruct the table to explicitly show whatever version one wants, that is, to rebuild the table including columns for all new attributes or relations of a particular schema version. The objects from older schemas get these columns of newer attributes or relations blanked.

In Fig. 5 is presented a snapshot of KTIDBE Explorer showing a binary (OKS_data) object opened.

So, the main advantages of oks2cool2 are:

- Schema can change any time causing no trouble to the storing procedure.
- It keeps historical evolution easy to track, because data with different schema can be stored on the same folder.
- It simplifies the implementation of any further functionalities.

3. ONASIC

The Online Asynchronous Interface to COOL is a 2-tier interface from the Online Information Service to Conditions database whose reliability lies in avoiding backpressure from database servers, i.e. in case the database servers shut down, ONASIC keeps collecting data from the Online infrastucture and manages it’s own cache of files until the servers are up again, then ONASIC will start to move this collected data into the database. This solution enables ATLAS/TDAQ to be up and running for at least 24 hours (TDAQ Configurations and Controls Working Group requirement to all Online software tools) disconnected from the outside world without danger of Conditions data loss.

It’s architecture uses 2 interfaces, onasic_is2oks (interface to the Online Services which belongs to the Online infrastructure) and onasic_oks2cool2 (interface to the Conditions database server which can be non dependent from Online Infrastructure), having between them a cache of OKS XML files which allows both interfaces to work independently, see Fig 6.

3.1. ONASIC Operation

The onasic_is2oks application works by creating callbacks for each IS Object for which it was configured to collect data from. Once a new such object is published by the Online Information Service, onasic_is2oks parses the object and creates the correspondent OKS class, in case it doesn’t exist already, after which the OKS object of this class is created.

After a user-defined timeout or a maximum total of 10000 objects present in onasic_is2oks memory, all these objects are dumped into a OKS XML file whose name gets a Time Stamp to

Figure 6. ONASIC Context Diagram.
The data mapping model is the most generic one, taking advantage of the semblance of OKS and IS Data Types, see Fig 7.

The ONASIC package includes the SetConditions tool which allows the user to configure ONASIC on-the-fly, thus enhancing ONASIC integration with Monitoring Data interested users. The Configuration options are as follows:

- Selection of the IS Servers and Objects being collected.
- Possibility of defining a new COOL FolderSet to store data into, thus allowing a better database organization.
- Selection of the IOV (Interval of Validity) type, either by Time Stamp or RunNumber plus Luminosity Block number, per IS Object.
- Option to specify from which IS Object's attribute to get the Luminosity Block number (by default, Luminosity Block number is got from "RunParams.RunParams" IS Object).
- Option to specify from which IS Object's attribute to get the Channel ID number. (By default, Channel ID is the MD5 out of a constant string)
- Possibility do specify a template form of the name of the IS Object in order to get any of: "Channel ID", "Luminosity Block Number" or "Run Number" from the name of the Object itself.
- Possibly to send a command to get a table with the present ONASIC configuration.

The ONASIC user can at any time check the present configurations, see Figure 8.

4. DBStressor

For the Large Scale Tests November 2006 edition, our group was asked to develop a testing tool that was able to simulate the sub-detectors access to Conditions database during the TDAQ
Configuration phase, or Athena jobs accessing Conditions Data. To achieve this task, we have developed the DBStressor as a package of the Online software.

The DBStressor package contains four applications, namely (see Fig. 9 and 10):

- **dbs_create_cool_table** - A tool to create large COOL tables with random data, to be used in the tests.
- **dbs_configure** - The tool used to configure on-the-fly the dbstressor_controllers
- **dbsstressor_controller** - The online controller itself, that accesses and fetches data from a COOL database.
- **dbs_get_results** - The tool that gathers the individual results of all dbstressor_controllers and dumped them into an ASCII file.

### 4.1. DBStressor Operation

DBStressor was used in 2 phases: The offline phase, which consists of the use of **dbs_create_cool_table** to create the COOL Folders for the testing purposes. This tool works by simply defining the schema of the Folder as decided by the user, and then, by generating random data, it creates and flushes the data payload into the Folder.

The online phase consists on the measurement of the times to fetch data from the Folders described above, which is done with a tdaq partition running through states. Moreover, the DBStressor controllers work in 3 steps:

---

**Figure 9.** DBStressor Creating Databases for tests.

**Figure 10.** DBStressor in the TDAQ Partition Context Diagram.
(i) - After booting the partition, the `dbs_configure` is used to configure all the dbstressor controllers, i.e. which connection links to which database server, which Folder to get data from, which interval of validity within the Folder to fetch and which Channel ID’s to get.

(ii) - During Configure Action, all the dbstressor controllers start a timer, use their configuration information previously received to access COOL, and fetch data into a local file, and then stop the timer and publish an IS object with their name, number of rows fetched and time they had taken to do it.

(iii) - After Start Action, when all the dbstressor controllers have completed their task (condition imposed by the state machine), `dbs_get_results` is used to collect all the IS objects published by the dbstressor controllers, and to compile them into an ASCII file which will serve as source data to produce histograms with timing measurements.

4.2. Tests ran with DBStressor in Large Scale Tests, 2006.

It was decided to follow a set of four tests:

(i) **Measure Oracle/COOL and LXBATCH bandwidth.**
Each controller fetched the same and complete table.

(ii) **Test capability of data Indexing by Oracle/COOL.**
Each controller fetched 25 percent of a large table, but each controller was assigned to get data from a random Since IOV.

(iii) **Measure times to build tables in COOL.**
Timing of the `dbs_create_cool_table` process for several different tables created.

(iv) **Test COOL performance to query by Channel ID.**
Each controller fetched a defined number of rows (10, 100, 1000 and 10000) but each of them was given a random Channel ID to get data from.

These tests have been performed for the scale steps 50, 100, 150, 250, 500, 750 and 1000 nodes.

The overall result of each test is a histogram with timing counts for all controllers. (see Fig. 11)
4.3. Results for Test 1

As example of DBStressor operation

The collection of results for Test 1 allowed to plot a nice curve of the measured bandwidth, even though there is a spread in bandwidth values according to the size of data being fetched at each test run. (see Fig. 12 left)

Fig. (12 right) shows the extrapolated bandwidth calculated from the measurements made in Test 1. It is quite evident that the bandwidth drops rapidly with the cluster size, indicating that 1000 nodes will get data from the database server at 100KB/s rate.

For each scalability phase a different partition was used, for 1000 nodes (see Fig. 13) the partition was organized as follows:

40 farms, each containing 25 sub-farms.
Each sub-farm containing 1 dbstressor controller.

5. Conclusion

Mainly we have developed and tested 3 interfaces, 2 that store data in COOL and another to test read access in a large scale.

The OKS2COOL2 solution to schema versioning changes seems to be a consistent solution that will persist and, what is more important, it will allow for Conditions databases to be better organized. This will also prevent some hard scripting work to transport data from databases with previous versions into new updated ones.

The ONASIC asynchronous solution is proved to be safe, and no better solution is foreseen. Also its user interface is now in shape which makes this a tool available and interesting for all Monitoring and Conditions Data users that want to store their data in COOL, as it now meets
all requirements previously asked by other groups.

DBStressor has given us the scope of what to expect from COOL Databases’ performance in the context of sub-detectors configuration and Athena jobs needs to access Conditions data. We can now extrapolate these results and be more precise when assuming how much time sub-detectors will need to configure. DBStressor provides a wide range of results that are still being analyzed, and it will be possible to understand more precisely some bottlenecks in relational database operation in the context of the TDAQ state machine operation.

6. Acknowledgements
The authors wish to thank the support from their respective institutes. This work is partially funded by the Portuguese Foundation for Science and Technology (FCT) grants number POCI/FP/63410/2005, POCI/FP/63948/2005 and POCI/FP/81940/2007.

References
[1] Ricardo Neves and Joao Simoes. TIDB2 User's Guide. 2007.
[2] Igor Soloviev. Configuration Databases User's Guide. 2002.
[3] Igor Soloviev. OKS User's Guide, Tools Manual and Reference Manual. 2002.
[4] Serguei Kolos. Information Service User's Guide. 2005.
[5] Doris Burchhart. Report on the ATLAS TDAQ Large Scale Tests 2006. 2007.
[6] A. Amorim et al. "Conditions Databases: the interfaces between the different ATLAS systems". "Proceedings of the International Conference on Computing in High Energy and Nuclear Physics" CHEP2004, Interlaken, Switzerland, 2004.
[7] N. Barros et al. "The ATLAS experience with databases for structured data with temporal and hierarchical order." In "Proceedings of 14th IEEE Real Time Conference", Stockholm, Sweden, 2005.
[8] D. Burchhart-Chromek et al. "Testing on a Large Scale: Running the ATLAS Data Acquisition and High Level Trigger Software on 700 PC Nodes". In: Computing In High Energy and Nuclear Physics (CHEP 2006), Mumbai, India, 13-17 Feb 2006, 6 pages. ATLAS notes: ATL-COM-DAQ-2006-008.
[9] B. Gorini et al. "The ATLAS Data Acquisition and High-Level Trigger: Concept, Design and Status". In: Computing In High Energy and Nuclear Physics (CHEP 2006), Mumbai, India, 13-17 Feb 2006, 6 pages. ATLAS notes: ATL-COM-DAQ-2006-030.
[10] A. Amorim, J. Lima, L. Pedro, D. Klose, C. Oliveira, N. Barros (University of Lisboa). "An Implementation for the ATLAS Conditions Data Management Based on Relational DBMSs". IEEE Transactions on Nuclear Science, Volume: 51, Issue: 3, page(s) 591- 595 IEEE, June 2004.
[11] A. Amorim, J. Lima, L. Pedro, C. Oliveira, N. Barros (University of Lisboa). "An Open Source based implementation for ATLAS Conditions Data". In "Proceedings of 2003 IEEE Real Time Conference", Montreal, Canada, 2003.
[12] A. Valassi, D. Duellmann (CERN); A. Amorim, N. Barros, T. Franco, D. Klose, L. Pedro (University of Lisboa); S. Schmidt (University of Mainz); V. Tsulaia (University of Pittsburgh). "LCG Conditions Database Project Overview". In "Proceedings of the International Conference on Computing in High Energy and Nuclear Physics" CHEP 2004, Interlaken, Switzerland, 2004.
[13] A. Amorim, D. Klose, L. Pedro, N. Barros, T. Franco (University of Lisboa). "Addressing the persistency patterns of the time evolving HEP data in the ATLAS/LCG MySQL Conditions Databases". In "Proceedings of the International Conference on Computing in High Energy and Nuclear Physics" CHEP2004, Interlaken, Switzerland, 2004.
[14] ATLAS Collaboration. "Computing Technical Design Report - TDR (CERN-LHCC-2005-022)" http://atlas-proj-computing-tdr.web.cern.ch/atlas-proj-computing-tdr/Html/Computing-TDR-4.htm#pgfId=570
[15] "LCG COOL - Large Computing Grid Conditions Database Project". http://lcgapp.cern.ch/project/CondDB/
[16] Joao Batista. KTIDBExplorer User's Guide. 2007. http://sim.fc.ul.pt/sim_en/KTIDBExplorer_documentation