CORAL Server and CORAL Server Proxy: Scalable Access to Relational Databases from CORAL Applications

A Valassi¹, R Bartoldus²,4, A Kalkhof¹, A Salnikov²,4, M Wache³,4

¹CERN, IT Department, CH-1211 Geneva 23, Switzerland
²SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA
³Institut für Physik, Universität Mainz, D-55099 Mainz, Germany

E-mail: andrea.valassi@cern.ch

Abstract. The CORAL software is widely used at CERN by the LHC experiments to access the data they store on relational databases, such as Oracle. Two new components have recently been added to implement a model involving a middle tier “CORAL server” deployed close to the database and a tree of “CORAL server proxies”, providing data caching and multiplexing, deployed close to the client. A first implementation of the two new components, released in the summer 2009, is now deployed in the ATLAS online system to read the data needed by the High Level Trigger, allowing the configuration of a farm of several thousand processes. This paper reviews the architecture of the software, its development status and its usage in ATLAS.

1. Introduction
The Large Hadron Collider (LHC), the world’s largest and highest-energy particle accelerator, designed to collide opposing beams of protons or lead ions, started its operations in September 2008 at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. Huge amounts of data are generated by the four experiments installed at different collision points along the LHC ring. The largest data volumes, coming from the “event data” that record the signals left in the detectors by the particles generated in the LHC beam collisions, are generally stored on files. Relational database systems are used by all four experiments to store several other types of data, such as the “conditions data” that record the experimental conditions at the time the event data were collected, as well as geometry data and detector configuration data. In three of the experiments, ATLAS, CMS and LHCb, conditions data and several other types of relational data are stored and retrieved from C++ or Python applications using the CORAL [1] software, one of the three packages jointly developed by these three experiments and the CERN IT Department within the Persistency Framework project [2].

1.1. Accessing the relational data of the LHC experiments using CORAL
CORAL provides a set of C++ libraries that support data persistency for several relational database backends and deployment models, including local access to SQLite files, direct client access to Oracle and MySQL servers, and read-only access to Oracle through the FroNTier/Squid web server/cache system [3]. The C++ API of CORAL consists of a set of SQL-free abstract interfaces that isolate the user code from the database implementation technology. Users write the same code for all backends,

⁴The author is a member of the ATLAS Collaboration.
as the SQL commands specific to each backend are executed by the relevant CORAL library, loaded at run-time by a special plugin infrastructure. The experiments use CORAL either directly in their applications, or indirectly through other common libraries like COOL [4], which is used by both ATLAS and LHCb for the management of the time variation and versioning of their conditions data.

Support for multiple backends makes CORAL especially suitable for accessing the relational data of the LHC experiments in the heterogeneous LHC Computing Grid (LCG) environment. In fact, different relational technologies and deployment policies may be in use at the many LCG sites, or for the many use cases involved in data processing and analysis, or even at different moments during the experiment lifetime. CORAL also provides components that simplify user authentication and replica lookup in this distributed environment. Presently, ATLAS, CMS and LHCb all store the master copy of their conditions databases on Oracle clusters at the CERN Tier0 site [5]. This is generally the only database that needs to be updated with new or modified conditions data, typically by applications running at CERN. Read-only access of conditions data, at CERN or remote Grid sites, for offline reconstruction or analysis is implemented in different ways, depending on the specific experiment, use case and site. These include replication of the Tier0 master databases to Tier1 sites via Oracle Streams [5] and partial replication to SQLite files for ATLAS [6] and LHCb [7]. Indirect access to the CERN Oracle database via the FroNTier/Squid web server/cache system is essentially the only mechanism used by CMS [3]; in late 2009, FroNTier was also adopted by ATLAS [8] for end-user analysis on the Grid. In all of these cases, CORAL is the software used on the client side by user applications.

1.2. Motivation for a CORAL server: a middle-tier for relational data access via CORAL

As described above, direct Oracle server access from CORAL client applications has been so far an essential ingredient for data processing in the LCG, in particular for all write access use cases and for some of the read access use cases of ATLAS and LHCb. A schematic representation of this client-server connectivity model is shown on the left side of figure 1. From the software point of view, this is triggered when the user application asks CORAL to connect to a URL starting with the ‘oracle’ prefix; the OracleAccess plugin of CORAL is then loaded, which internally uses the Oracle OCI client protocol to connect to the desired database server (‘dbserver’ in this example).

Several limitations of this basic two-tier connectivity model have become apparent over time with the growing use of the deployed Oracle servers. To start with, the only authentication mechanism available for experiment jobs consists in providing user names and passwords at connection time (as X509 proxy certificate authentication is not supported by the database vendors). This poses many
security risks, including password vulnerability and the need to expose database ports on the public network for remote users. When the experiment is accessed through shared database accounts available to all members of the collaboration, this also forces the experiments to maintain a complex infrastructure for retrieving the credentials for these accounts from individual user jobs, as reported by LHCb [7]. A second limitation of this model is that different client jobs must establish separate physical connections to the Oracle server, even when they all access the same schemas using the same credentials. This may lead to an inefficient use of server resources and performance bottlenecks when connection rates are high; this is especially true when many jobs are launched at the same time and all try to access the database simultaneously, as seen in ATLAS [6]. A third limitation of this model is the dependency of user applications on the Oracle client software, as this is linked to the CORAL plugin that is loaded at runtime when accessing Oracle servers directly. Any update to a new version of the Oracle client implies the redistribution to all Grid nodes of this software, and in most cases also of new CORAL libraries rebuilt against it. For instance, a crash of the Oracle 11g client libraries reported by ATLAS [9] on a specific hardware model deployed on some of its Grid sites caused the urgent rebuild of a new CORAL release, temporarily downgrading to the Oracle 10g client.

The project described in this paper was initially proposed in 2006 [10] precisely to overcome those limitations of the simple two-tier connectivity model. The basic idea is that a middle-tier server should be introduced between CORAL clients and the Oracle database servers. Client applications could then connect to this “CORAL server” through a new CoralAccess plugin and a new custom protocol (via a URL starting with the ‘coral’ prefix), while the CORAL server, itself a CORAL application, would connect to the Oracle database server inside the site firewall. The Oracle credentials could be stored on the CORAL server, which would use them for clients authenticated and authorized using their X509 proxy certificates. Access from several concurrent clients to the same Oracle schemas could be multiplexed through a single connection from the CORAL server to the database, reducing the load on the latter. In this model, shown in the central part of figure 1, client applications would also no longer depend on the Oracle client software, which is only needed in the CORAL server.

The model above was then extended by introducing an additional “CORAL server proxy” between the clients and the CORAL server, to provide data caching and further multiplexing for read-only use cases, as represented on the right side of figure 1. This is particularly relevant to the specific read-only use case of database access for the ATLAS High-Level Trigger (HLT) configuration, described in the next section. In fact, development on this project started in 2007 as a joint effort of the CORAL team in CERN IT and the ATLAS HLT team in SLAC, with the goal of developing a system that could address both the specific ATLAS requirements for read-only data access in an online environment and the more general requirements of scalable and secure database access for distributed offline processing on the Grid. The deployment of the software in the ATLAS HLT was set as the first priority, as this was an urgent need for the HLT team at the time of the LHC startup. A first phase of development in 2007 and 2008 [11] was very useful to complete the analysis of the HLT requirements and understand some of the issues involved. Following some departures and arrivals in the CORAL team, development was restarted at the beginning of 2009 according to a brand new software architecture design [12], described in section 3. After less than one year, the software was installed in the ATLAS HLT system in October 2009, where it has been used for LHC data taking without any major issues.

2. CoralServer and CoralServerProxy for database access in the ATLAS HLT

Within the ATLAS Trigger and Data Acquisition (TDAQ) system, shown schematically in figure 2, the Level2 (L2) and Event Filter (EF) triggers of the HLT [13] are meant to provide the final selection to reduce event rates to approximately 200 Hz for data storage. These software-based triggers take a decision based on the partial (L2) or full (EF) reconstruction of raw events: as in the offline environment, this requires the HLT processes to retrieve other types of data from relational databases, including COOL conditions data, geometry data and trigger configuration data. Events are processed in parallel on the L2 and EF farms, which presently consist of approximately 1100 nodes in total (though this is only about 40% of the final expected setup). As 8 HLT processes run in parallel on the
dual-CPU quad-core nodes in the farms, almost 9000 clients in total need to retrieve tens of MB of data from the database simultaneously within a few minutes at most.

To address the scalability issue implied by these numbers, it was observed early on by the HLT team that all L2 or EF clients need exactly the same data. It is then enough to fetch these data only once from the database and distribute them to all processes in the HLT farms. This model, shown in figure 3(a), provides the required scalability by caching data and multiplexing connections through a tree of proxy servers. A first implementation, shown in figure 3(b), was successfully deployed in ATLAS in 2007-2008: data were retrieved by the HLT processes from a MySQL database server using the relevant CORAL plugin, while a “dbProxy” component, based on the open-source MySQL protocol, took care of caching and multiplexing. It should be noted that FroNTier/Squid system also implements the model in figure 3(a) and had been evaluated as a possible solution before dbProxy was developed, but its performance in 2006 had been found inadequate for the ATLAS HLT requirements.

In 2008, following the decision to move the data from the ATLAS-managed MySQL server to the Oracle clusters maintained by CERN IT, this system ceased to work. As the dbProxy could not be re-implemented for the proprietary Oracle protocol and the CORAL server was not yet mature enough to be used in production, a temporary solution was found: as shown in figure 3(c), this involved a “MySQL-to-Oracle” (M2O) translator proxy between the Oracle server and the dbProxy tree. This was only a short term solution, since the translation between MySQL and ORACLE cannot be made universal and has to be validated for every new type of data, which involves a significant overhead.

In October 2009, the CoralServer and CoralServerProxy software was finally deployed in the ATLAS online, as shown in figure 3(d). The commissioning phase was very smooth: the software worked almost immediately, providing the required read-only access functionalities with an adequate performance, comparable to that ensured by the M2O-based system. This is largely due to the emphasis placed on tests during the development process: in particular, a stripped down version of the HLT software, packaged as a standalone test reproducing the HLT access to all three relevant databases (COOL, geometry, trigger configuration), had systematically been used prior to deployment to test the software functionality and performance. During one year with the new system, only one incident has occurred, where the CORAL server hanged and had to be restarted; this is understood as

Figure 3. Connection models of the ATLAS HLT clients to the relevant database servers: (a) generic database server and a tree of caching proxies; (b) MySQL server and dbProxy tree, as in 2007-2008; (c) Oracle server, M2O translator proxy and dbProxy tree, as in 2008-2009; (d) details of the deployment since October 2009, with one CoralServer connected to three Oracle servers, and a hierarchical tree of CoralServerProxy instances.
being caused by a bug in CORAL handling of network glitches, not entirely specific to the CORAL server [2]. As a positive bonus, the new system has also simplified authentication, since it is now no longer necessary to distribute the Oracle passwords to all HLT client processes, as it was in the past.

3. CoralServer software: architecture design and development outlook

This section briefly summarized the design principles of the CoralServer software. More details about the architecture design are beyond the scope of this paper and are available in the unpublished internal documentation [12]. The design takes into account both the specific read-only requirements of ATLAS HLT and the more general needs of potential offline users for secure write access. The software has three deliverables, the client library and the server and proxy executables, which communicate over TCP/IP via a dedicated binary protocol. Only one package (CoralServerBase), including the protocol subset needed by the proxy to know which packets are cacheable, is used by all three components: the development of the proxy (at SLAC, partly based on dbProxy) and that of client and server (at CERN and Mainz) have been largely independent otherwise. The development of the client and that of the server are instead tightly coupled and follow a joint design based on a Remote Procedure Call model. As shown in figure 4, client and server are both split into three components: the client and server ‘halves’ of each component form a single unit that can be developed and tested independently from the others, to which they are loosely coupled using abstract C++ interfaces. One package (CoralSockets) sends and receives packets over the network; another (CoralStubs) marshals and unmarshals packets as method calls of a single relational “façade”; the last package pair (CoralAccess and CoralServer) bridges this façade from/to the relevant CORAL calls on the client or server side. Testing at all levels was an essential ingredient of the successful and rapid deployment of this new software in ATLAS; the modular design described in this paragraph also allowed the incremental test of database access through a variable subset of the three components, as shown in figure 5. This was done for instance on the standalone test reproducing the HLT access to all three databases (COOL, geometry, trigger).

While read-only access is now used successfully in production by the ATLAS HLT, work on the other functionalities foreseen for more general use is still ongoing. SSL sockets and X509 proxy certificate authentication with VOMS role authorization are in the final testing stage before they can be released, while update functionalities have only been prototyped. A monitoring infrastructure and performance tests (including comparisons to FroNTier) are also underway. At the moment, however, this work is delayed as the priority is the maintenance and support of CORAL during LHC data taking.

---

**Figure 4.** Component decomposition of the CORAL server executable and client plugin.

**Figure 5.** Incremental tests of the software: from direct Oracle access (left) to access via the CoralServer (right).
4. Conclusions
The new CORAL server and proxy components have been used successfully for LHC data taking in the ATLAS HLT since October 2009. Their rapid and successful commissioning, in less than one year since the start of their development, is largely due to the emphasis on testing and the excellent cooperation in the development team, which was facilitated by the modular design of the software.

References
[1] I. Papadopoulos, R. Chytracek, D. Duellmann, G. Govi, Y. Shapiro and Z. Xie. Proc. CHEP 2006, Mumbai. CORAL, a software system for vendor-neutral access to relational databases, http://indico.cern.ch/contributionDisplay.py?contribId=329&sessionId=4&confId=048
[2] A. Valassi et al. Proc. CHEP 2010, Taipei. LCG Persistency Framework (CORAL, COOL, POOL): status and outlook, http://cdsweb.cern.ch/record/1327680
[3] D. Dykstra and L. Lueking. Proc. CHEP 2009, Prague. Greatly improved cache update times for conditions data with FroNTier/Squid, http://frontier.cern.ch/dist/Paper_CHEP09_Frontier-newcaching.pdf
[4] A. Valassi, R. Basset, M. Clemencic, G. Pucciani, S. A. Schmidt and M. Wache. Proc. NSS 2008, Dresden. COOL, LCG conditions database for the LHC experiments, http://cdsweb.cern.ch/record/1142723
[5] M. Girone. Proc. CHEP 2009, Prague. Distributed database services, a fundamental component of the WLCG service for the LHC experiments, http://cdsweb.cern.ch/record/1177864
[6] A. Vaniachine et al. Proc. CHEP 2009, Prague. Advanced technologies for scalable ATLAS conditions data access on the Grid, http://iopscience.iop.org/1742-6596/219/4/042025
[7] M. Clemencic. Proc. CHEP 2007, Victoria. LHCb distributed conditions database, http://iopscience.iop.org/1742-6596/119/7/072010
[8] D. A. Smith et al. CHEP 2010, Taipei. FroNTier use in ATLAS, http://cdsweb.cern.ch/record/12998655
[9] A. Vaniachine. Private communication (2010), https://savannah.cern.ch/bugs/?762194
[10] D. Duellmann. LCG AA 2006 Internal Review. Persistency Framework overview (unpublished), http://indico.cern.ch/conferenceDisplay.py?confId=a063165
[11] Z. Molnar, R. Chytracek, D. Duellmann, G. Govi, A. Kalkhof and A. Valassi. Proc. NSS 2008, Dresden. Distributed database access in the LHC computing grid with CORAL, http://cdsweb.cern.ch/record/1142728
[12] A. Valassi. CORAL Server project: software architecture design proposal (unpublished, 2008), http://indico.cern.ch/materialDisplay.py?materialId=2&materialId=0&confId=46850
[13] W. Wiedenmann. Proc. CHEP 2009, Prague. The ATLAS online High Level Trigger framework, http://iopscience.iop.org/1742-6596/219/2/022024

Acknowledgments
This project has benefitted from the work and suggestions of several other members, former and current, of the CORAL, COOL and POOL teams, all of whom we are grateful to. We especially thank Dirk Duellmann and Zsolt Molnar for their contributions to the early discussions and developments on the CORAL server project in 2008 and previous years. We are grateful to the CORAL user community in the LHC experiments for their continuous feedback about the software and their help in defining the user requirements for the CORAL server. We particularly thank our colleagues in the ATLAS online teams for their help in testing and deploying the software at Point1. All the former and current SPI team members deserve special thanks for their constant help in maintaining the CORAL development infrastructure. Several useful discussions with the developers of the VOMS software, particularly Akos Frohner and Vincenzo Ciaschini, are also kindly acknowledged. Finally, our colleagues from the Physics Database Team in CERN IT, together with the DBAs of the ATLAS experiment, have provided invaluable consultancies for understanding and overcoming the issues in the Oracle database servers they operate: we warmly thank them for their help and their support to this project.