EPJ Web of Conferences **214**, 04052 (2019) https://doi.org/10.1051/epjconf/201921404052
CHEP 2018

Conditions evolution of an experiment in mid-life, without the crisis (in ATLAS)

Lorenzo Rinaldi[1,*], *Andrea* Formica[2], *Elizabeth J* Gallas[3,**], *Nurcan* Ozturk[4], and *Shaun* Roe[5], on behalf of the ATLAS Collaboration

[1]Dipartimento di Fisica e Astronomia, Università di Bologna and INFN Sezione di Bologna, Via Irnerio 46, I-40126 Bologna, *IT*
[2]Université Paris-Saclay, CEA/Saclay IRFU, 91191 Gif-sur-Yvette, IRFU/CEA, FR
[3]University of Oxford, Denys Wilkinson Bldg, Keble Rd, Oxford OX1 3RH, UK
[4]University of Texas at Arlington, Arlington 76019 Texas, USA
[5]CERN, CH - 1211 Geneva 23, Switzerland

Abstract. The ATLAS experiment is approaching mid-life: the long shutdown period (LS2) between LHC Runs 1 and 2 (ending in 2018) and the future collision data-taking of Runs 3 and 4 (starting in 2021). In advance of LS2, we have been assessing the future viability of existing computing infrastructure systems. This will permit changes to be implemented in time for Run 3. In systems with broad impact such as the conditions database, making assessments now is critical as the full chain of operations from online data-taking to offline processing can be considered: evaluating capacity at peak times, looking for bottlenecks, identifying areas of high maintenance, and considering where new technology may serve to do more with less.
We have been considering changes to the ATLAS Conditions Database related storage and distribution infrastructure based on similar systems of other experiments. We have also examined how new technologies may help and how we might provide more RESTful services to clients. In this presentation, we give an overview of the identified constraints and considerations, and our conclusions for the best way forward: balancing preservation of critical elements of the existing system with the deployment of the new technology in areas where the existing system falls short.

1 Introduction

The ATLAS experiment [1] is operating at the Large Hadron Collider (LHC) at CERN. ATLAS has been successfully collecting collision data for the past 8 years, and in the next few years there will be significant upgrades of the LHC machine and of ATLAS subdetector systems, bringing an increase in luminosity and in the rate of collected data. The processing of the experimental data collected by ATLAS requires a wide variety of auxiliary information from many systems (e.g. Detector Control Systems (DCS), Trigger and DAQ, Data Quality, the LHC accelerator and ATLAS sub-detectors) stored in the ATLAS Conditions Database. Such pieces of information are heterogeneous both in data type (e.g. standard integer, floating

*e-mail: rinaldi@bo.infn.it
**e-mail: gallas@cern.ch

point and string types, database-standard BLOBs, CLOBs, and types special to the Conditions database infrastructure like references to external POOL and ROOT files) and in time granularity (spanning intervals from minutes to hours in duration).

During the LHC Run 1 and Run 2, the ATLAS Collaboration deployed and used a system based on the LHC Computing Grid (LCG) Conditions Database infrastructure and the COOL API [2], a C++ library based on a software layer called CORAL which manages the actual database access and the queries that should be issued (hiding the SQL complexity from clients) and supports multiple relational platforms (SQLite and Oracle). We found that this architecture worked well so far, but is showing signs that it will not scale to cope with the processing of the increasingly complex and intense data flows coming with the growing LHC luminosity.

For this reason, the ATLAS Conditions Database management is evolving to a new Conditions Database system [3] which is based on RESTful [1] client-server interaction and has an architecture which uses an intermediate server that disentangles the business components dealing with the database management aspect of the client. This allows the client and server to evolve separately via well-identified interfaces. In the new framework, the database access layer is implemented at server level and the exchanged network traffic will be conditions-database specific (instead of the present generic SQL), allowing to profit in a better way of the parameters used to retrieve the data, which are today completely invisible inside the SQL statement; this element can be relevant for caching optimization. During the intermediate transition phase, we are developing a set of new tools to permit RESTful access to COOL/CORAL for preserving the functionality of the present system.

2 The ATLAS Conditions Database during LHC Run 1 and Run 2

ATLAS conditions data are stored in a relational database. The database design is based on the LCG Conditions Database and is accessed by clients using the COOL API, both of which were developed by the CERN IT department for the LCG [2]. COOL is a C++ API library based on the CORAL access layer. It provides high level functionality which allows users (the expert scientists that manage the conditions data for any ATLAS sub-detector) to create their own COOL Database and fill it with payload data corresponding to a given time range over which that data is valid (the IOV, or Interval of Validity). Using the COOL terminology, a COOL Database for a given system is called a *schema*, and the database tables dedicated to a given set of parameters inside each schema are called *folders*. A folder containing data which can only have a fixed, unchangeable value in each time interval is defined as a single-version (SV) folder: IOVs may only be appended and data cannot be overwritten. A folder is defined as a multi-version (MV) folder when the payload data can have multiple versions in any given IOV: each new version of the data is entered under a distinct *folder tag*. For data processing involving many folders, a higher level *global tag* is defined which contains all the folder tags to be used: this simplifies task configuration because event processing generally requires the conditions data from over 150 different folders.

Conditions data are concurrently accessed by a large number of clients: the majority are event processing jobs using Athena [4] the ATLAS event processing software framework. Database access is managed via the COOL API using the intermediate Frontier/Squid [5] services, as shown in Figure 1.

Database queries which are created via the COOL/CORAL stack are propagated as parameters inside a URL and sent as HTTP requests to the Frontier server. Access to the Oracle Database is managed via a Java DB Connectivity (JDBC) layer. Since each request with an

[1] Representational State Transfer (REST): https://en.wikipedia.org/wiki/Representational_state_transfer

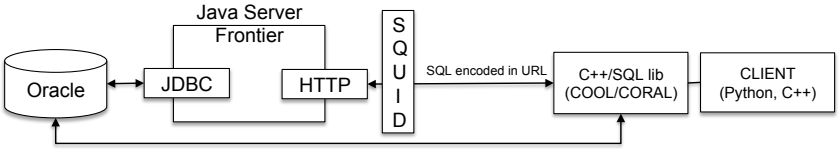

Figure 1. Schematic of the COOL access mechanism

identical URL will retrieve the same data from Oracle, a simple web caching layer is implemented via squid proxy.

The database design and the data access model have been generally stable through LHC Run 1 and for most of Run 2, but in the later stages of Run 2 dips in data access efficiency were observed. This led to a general survey of the system to assess how to improve current operations as well as the viability of the current infrastructure for future operations. We found they current system is problematic in a number of areas:

- Global management of the system is made more difficult because the conditions data are contained in over 30 schemas and stored in over 10,000 underlying folder tables in the database. The implementation of global tags in the LCG infrastructure proved cumbersome in ATLAS because of the way in which it is implemented at the database level.
- The granularity of the conditions data in the current system is not well suited to the caching mechanisms of Frontier.
- The support from CERN IT of the underlying software stack (COOL and CORAL) is decreasing and will stop at the beginning of Run 3. Maintenance of the complexity of the current implementation will be magnified as new data is added to existing schemas and as new detector systems come online. We also are concerned about the possible implications in terms of data preservation.

These considerations lead us to evaluate new database models and architectures on the time scale of the end of Run 3.

3 The new Conditions REST infrastructure

A new REST-based architecture for the management of ATLAS conditions data is now being developed. The new Conditions REST (CREST) architecture enhances the role of the Frontier servers, as shown in Figure 2. CREST development started in collaboration with the CMS experiment [6] who have been successfully using this underlying database model for storing and accessing conditions data since the beginning of LHC Run 2. In this architecture, the client is not aware of the underlying persistence technology used, and interacts with the storage via functions implemented at server level, providing only the set of conditions data or metadata that are needed directly inside the URL (or requests body or headers) of the HTTP method (GET, POST,...) used. Having the client compose requests via an abstraction layer above the SQL also allows alternative storage systems to be swapped on the server side in the future without changes needed on the client side. We call this set of functions the CREST API; the API has been written using OpenAPI [7] specifications (in short, a JSON file describes the URLs and their parameters as well as the Request and Response content). This enables us to take advantage of code generation tools: in our case, the client library (in

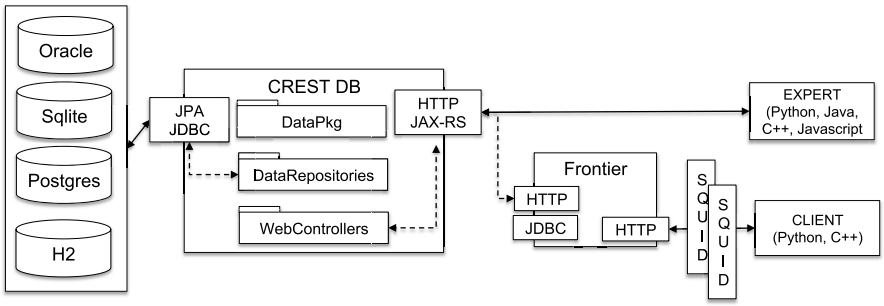

Figure 2. Schematic of the CREST database architecture.

Python) and server stubs (in Java JAXRS [8]) are generated via the Swagger Codegen [9] library.

The other important simplification of the new system is the data storage: only a small number of relational tables are needed. The relevant metadata like global tags and tags are very well mapped in this structure. Another important aspect is the separation of the Payload data from the IOVs, which allows two clearly separated steps for loading the IOVs (considered as metadata as well) and then the relevant associated payload via a unique hash. This separation is essential in the caching: when we refresh the request for new IOVs, we can still access old payload data by their unique URL which will be in principle already cached by the Squids. We do not have a way to retrieve the same payload via different URLs as is the case today in the COOL API. Even though we could try to use COOL system in a similar manner, the existing COOL API does not allow for this separation (we provide the range in time we want in the request for Payload data). We may try to emulate the CREST data model approach by registering in the Payload only a reference to a file stored somewhere else, but this implies that we need then another software components to access and distribute those file (this may cause synchronisation problems between the information in the database and in the file distribution).

3.1 Athena CREST access

The Athena component that handles accesses to COOL has been modified to instead issue HTTP requests to the CREST server and data is provided in JSON format. Simple (single IOV) jobs run transparently (client/subsystem code is kept unchanged) switching between COOL/CREST, with the possibility to produce or consume local data files in JSON format.

The existing prototype handles single IOVs demonstrating the proof of concept, so this is now being extended to handle multiple IOVs.

4 The transition from COOL to CREST

The full CREST architecture will be ready by the end of Run 3. In the transition phase we need to preserve the client COOL access. To facilitate a smooth migration, a new access mode called "COOLR" (a COOL REST access method) is in development. The COOLR architecture is shown in Figure 3. In the CREST architecture, the permitted queries are implemented in the Java server, and the client sends only the parameters. The server is not meant to be generic (can only “read” COOL). The query parameters appear ’in the clear’ in

the URL, so identical query URLs can be cached by the squid (as in Frontier), with only one HTTP call.

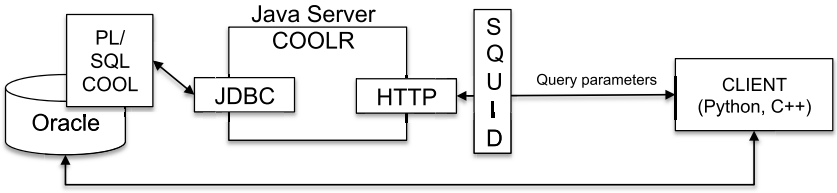

Figure 3. Schematic of the COOL REST (COOLR) architecture. Direct line indicates that the client can also directly access Oracle.

4.1 Gatling stress tests

A first estimate of the COOLR access performance has been carried out using Gatling [10], a highly capable load testing tool. The Gatling test machine has been set up with an official Frontier launchpad installation, as shown in Figure 4. The Frontier launchpad and the

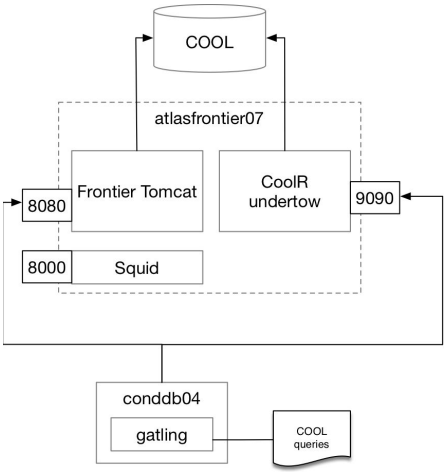

Figure 4. Schematic of the Gatling test engine.

COOLR server are listening on different ports. Gatling framework and scripts (automatically generated via swagger) are deployed on another VM at CERN. A set of 2000 "real" COOL queries was provided from the current system as input. The test queries are randomly taken by Gatling to simulate a load on the servers. In this way, Frontier and the COOLR server are using the same input (encoded SQL) and this guarantees that we see a performance similar to that of the Frontier launchpad. For both systems we measured the number of responses as a function of time, in two different configuration (with and without squid cache). We observed that both systems show a similar behaviour. In particular for COOLR, without the squid

cache, the test shows that we can reach a peak level of 500 requests per second, without any errors arising. With the squid cache appended to the launchpad, we managed to smoothly reach a rate of 1.5K requests per second. Gatling test results for COOLR are shown in Figure 5. That result gives a first indication that the COOL REST access can sustain a rate of

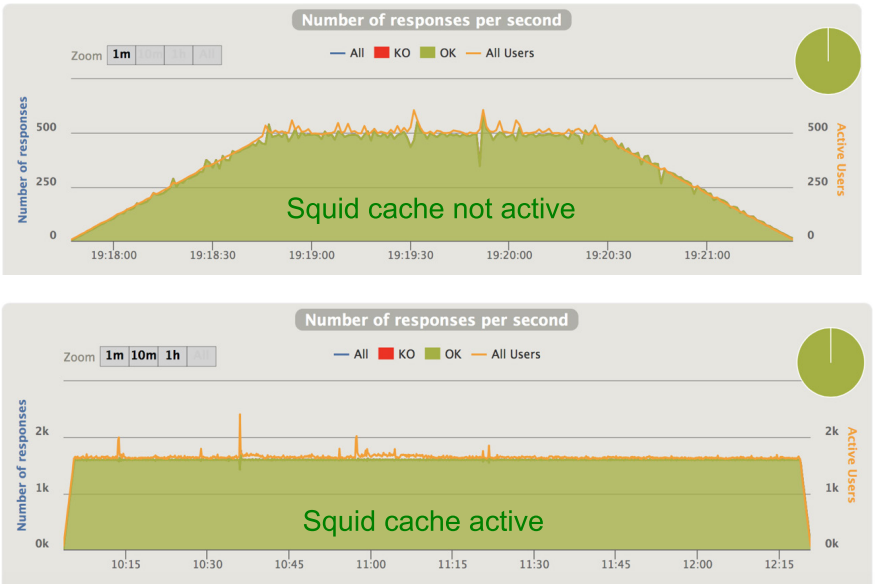

Figure 5. Gatling test results: number of COOLR responses as a function of time, without using squid cache (upper plot), and using the squid cache (lower plot). COOLR performance is similar to Frontier.

requests comparable to that of the current Frontier throughput.

5 Summary and conclusions

The COOL database has been successfully used for storing conditions data during LHC Run 1 and Run 2. The architecture is solid but it is not scalable with the increasing number of ATLAS data processing jobs expected in the future. A new Conditions REST Database infrastructure (CREST) is under development: It is based on a server-client architecture that allow a simpler and more lightweight management of condition payload and IOV data requests. The full deployment of CREST is foreseen by the end of LHC Run 3.

During the transition phase, we are evaluating COOLR, a prototype for COOL REST access, which uses an extended IOVDbSvc method enabling the new REST access mode in Athena. Preliminary Gatling stress tests using Frontier-like access shows that with the new system, COOL REST access can sustain loads comparable to that currently sustained by Frontier.

References

[1] The ATLAS Collaboration, JINST **3** S08003 (2008).

[2] Trentadue R et.al. *"LCG Persistency Framework (CORAL,COOL, POOL): status and outlook in 2012"*, J. Phys. Conf. Ser. **396** 053067 (2012).

[3] Formica A and Gallas E J *"A JEE RESTful service to access conditions data in ATLAS"*, J. Phys. Conf. Series **664** 042016 (2015).

[4] Calafiura P et.al. *"Running ATLAS workloads within massively parallel distributed applications using Athena multi-process framework (AthenaMP)"*, J. Phys. Conf. Ser. **664** 072050 (2015).

[5] Barberis D et.al. *"Evolution of grid-wide access to database resident information in ATLAS using Frontier"*, J. Phys. Conf. Series **396** 052025 (2012).

[6] Roland Sipos et.al. *"Functional tests of a prototype for the CMS-ATLAS common non-event data handling framework"*, J. Phys. Conf. Ser. **898** 042047 (2017).

[7] OpenApi: https://www.openapis.org [accessed 2018-12-03]

[8] JAVA JAX-RS: https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services [accessed 2018-12-03]

[9] Swagger: http://swagger.io [accessed 2018-12-03]

[10] Gatling: https://gatling.io [accessed 2018-12-03]