EPJ Web of Conferences **295**, 01013 (2024) https://doi.org/10.1051/epjconf/202429501013
CHEP 2023

Towards a new conditions data infrastructure in ATLAS

Evgeny Alexandrov[1], *Luca* Canali[2], *Davide* Costanzo[3,*], *Andrea* Formica[4,**], *Elizabeth J.*Gallas[5,***], *Mikhail* Mineev[1], *Nurcan* Ozturk[6,****], *Shaun* Roe[2], *Vakho* Tsulaia[7], and *Marcelo* Vogel[6]

[1]Joint Institute for Nuclear Research, Joliot-Curie 6, 141980 Dubna (Russia)
[2]CERN, CH-1211 Geneva 23 (Switzerland)
[3]Department of Physics and Astronomy, University of Sheffield, Sheffield (United Kingdom)
[4]IRFU, CEA, Université Paris-Saclay, F-91191 Gif-sur-Yvette (France)
[5]University of Oxford, Denys Wilkinson Bldg, Keble Rd, Oxford OX1 3RH (United Kingdom)
[6]University of Texas at Arlington, 701 South Nedderman Drive, Arlington, TX 76019 (USA)
[7]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (USA)

Abstract. The ATLAS experiment is preparing a major change in the conditions data infrastructure in view of LHC Run 4. In this paper we describe the ongoing changes in the database architecture which have been implemented for Run 3, and describe the motivations and the on-going developments for the deployment of a new system (called CREST for Conditions Representational State Transfer, as a reference to REST architectures). The main goal is to set up a parallel infrastructure for full scale testing before the end of Run 3.

1 Introduction

The processing of ATLAS [1] event data necessitates the retrieval of a collection of auxiliary non-event data stored within database systems. This data, referred to as "conditions data," generally exhibits variations over time and encompasses elements such as detector alignment, calibration, and configuration information. The complexity escalates due to the requirement of disseminating this information across the global ATLAS computing grid, along with the sheer multitude of concurrently operating processes on the grid. Each process demands a distinct set of conditions to advance.

Our focus is directed towards the foundational database infrastructure, which underwent a redesign for ATLAS in Run 3. This redesign involved the consolidation of resources within the online Oracle cluster, coupled with the necessary developments to ensure secure access.

This reorganization resulted in an architecture resembling the one the experiment is preparing for Run 4, known as the CREST project [2]. Here we expound upon the architecture and the current development status of this project. The first significant milestone involves deploying a functional demonstrator by fall 2023, with the intention of testing segments of data processing workflows using real conditions data migrated from the existing system.

*e-mail: davide.costanzo@cern.ch
**e-mail: andrea.formica@cern.ch
***e-mail: elizabeth.gallas@cern.ch
****e-mail: nurcan.ozturk@cern.ch

2 Conditions Database infrastructure

The infrastructure for managing and accessing condition data within ATLAS consists of the following components:

- Database clusters: Oracle databases store the conditions data according to the LCG Conditions database infrastructure [3] which includes C++ and python methods within its COOL API for managing database content and for client read-only access. Over the years using this infrastructure, further methods have been developed on top of the COOL API to suit ATLAS-specific requirements.
- Database read-only copies: Database replicas kept in sync with the source utilizing Oracle technologies which include Active Data Guard (ADG, the Oracle-provided technology for physical replication) and Golden Gate (an Oracle solution for logical replication of selected schemas).
- Generic database access via a middle-tier server: This is achieved through the Frontier system [4] which mediates client data selection requests with the underlying database storage system. Benefits of this layer include the ability to monitor requests as well as to moderate intermittent spikes in load.
- Web Proxy: To ensure efficient access and optimal resource utilization, a series of Squid proxies are deployed. These proxies screen the Frontier server and the database, filtering requests from individual clients (jobs) that are accessing condition data.

2.1 Architecture before Run 3

The conditions data are organized in two different Oracle clusters, depending on their usage (online data taking and High Level Trigger processing, or any other "offline" workflow):

1. Online Oracle cluster (named ATONR) on the ATLAS technical network, for conditions data to be consumed in real time workflows.
2. Offline Oracle cluster (named ATLR) on the CERN GPN network, for conditions data to be consumed in offline workflows, from bulk processing to reprocessing and Monte Carlo simulation.

Conditions that are primarily stored in the online cluster are additionally replicated using Golden Gate streaming technology to the offline cluster. Once in the offline cluster, these conditions can be accessed in read-only mode. The figure 1 provides a simplified overview of the ATLAS Conditions Data infrastructure prior to Run 3.

2.2 Architecture during Run 3

For Run 3 a major operation of databases consolidation has been prepared. Two main aspects were considered in this plan:

- Oracle license model: the license model until 2023 was covering all Tier-1s Oracle nodes used by ATLAS to keep a copy (via Golden Gate) of the conditions data. After April 2023 a new license model was adopted by CERN IT, based on a "per-core" license for Oracle nodes.
- ATLAS was the only client inside CERN of the Oracle Golden Gate replication technology: its usage was discouraged by CERN IT, considering the licensing costs, the additional load on the support team, and the redundancy with the other available replication technology: Oracle Data Guard.

Subsequently, the ATLAS Database and Metadata (ADAM) team made the decision to shift real data conditions workflows from the offline to the online cluster in order to phase out the need for the Golden Gate replication. The architecture that was implemented is depicted in figure 2. A few major alterations in the architecture were required to eliminate the Oracle Golden Gate replication (from the online cluster to the offline cluster and Tier-1s): the centralization of all real data conditions into the Oracle ATONR cluster and the establishment of an intermediary service (called COOL-Proxy) designed to manage user requests for the storage of new conditions data. In this freshly devised infrastructure, all conditions data situated outside the online ATLAS network (ATCN) can be accessed solely through read-only (ADG) replicas of the ATONR nodes.

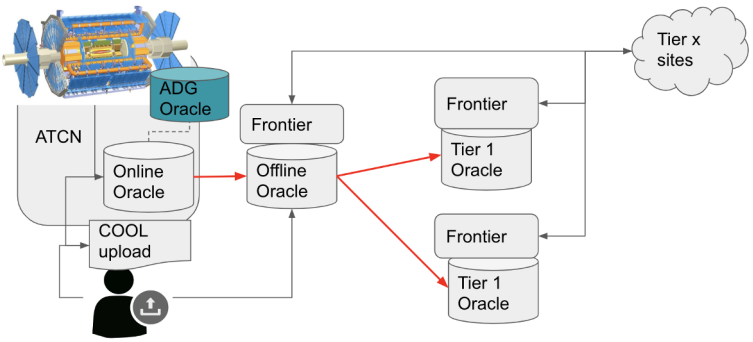

Figure 1. Conditions Data system architecture before Run 3; the red arrows depict data copies using the Oracle Golden Gate technology.

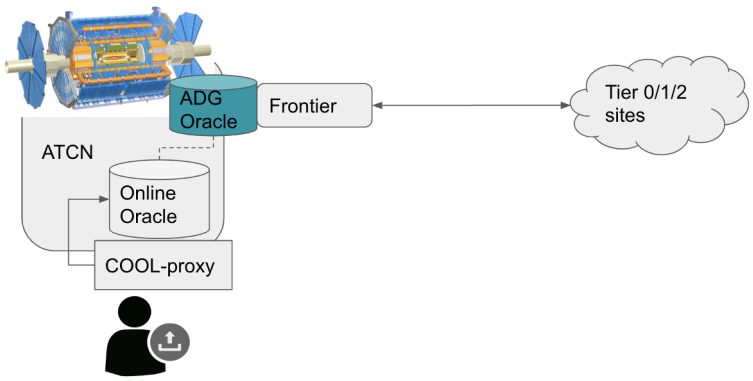

Figure 2. Conditions Data system architecture for Run 3; the COOL-proxy is accessible from CERN General Public Network (GPN).

This novel architecture brings about the ability to optimize Oracle licensing costs. Through the consolidation of all conditions data usage within the online cluster (ATONR), a distinct and dedicated environment for all data processing workflows is established. This separation prevents the mingling of conditions usage with other applications, as the ATLR cluster supports a wide array of applications ranging from detector construction to authorship

and metadata. Additionally, this approach has led to a reduction in Oracle administration burden and associated expenses for Tier-1s.

To ensure secure access to the ATONR cluster from the CERN GPN network, we devised an intermediary server that acts as a custom proxy system. This system, known as COOL-Proxy, is intended for use by experts responsible for uploading new conditions data. Notably, the COOL-Proxy system employs the new CERN SSO authentication mechanism. Conditions data experts are linked to specific e-groups, and a token mechanism is utilized to grant them writing privileges solely within the Oracle schema corresponding to the e-group(s) to which they belong.

On the hardware front, enhancements were made to the online cluster, involving the addition of an extra node. This improved configuration ensures extra capacity to critical online system to process all new conditions upload workflows, while maintaining the same load on the rest of the online environment as experienced during Run 2. Moreover, this extra node improves the available redundancy in case of cluster node failures.

3 CREST: a Conditions Database infrastructure after Run 3

The system implemented during Run 3 closely resembles the architecture that is being tested for the ATLAS runs commencing from Run 4. The new initiative for managing conditions data is named CREST, originating as a progression from the CMS conditions database. It inherits fundamental concepts for the data model and the design of relational tables from its predecessor. The development of the CREST system also benefited from discussions within the HEP Software Foundation [5] working group on cross-experiment conditions data management systems [6]. The intention behind CREST is to replace the existing COOL conditions database in satisfying the conditions data requests of all offline data processing and Monte Carlo simulations from Athena [7] jobs (Athena is the the ATLAS software framework for event processing). This comprehensive system is composed of several integral components:

- Relational Database: Data in the CREST database is stored in relational tables utilizing a straightforward schema. Conditions data payloads are stored within the database as Large Objects (LOBs) and referenced through unique keys stored as related metadata in CREST. In-depth information regarding the data model can be found in references [2] and [8].
- A REST API for the conditions data management system, accompanied by an implemented web server and corresponding client libraries.
- A web proxy system designed to offer a caching layer, thereby diminishing the utilization of the web server and database by clients (Athena jobs).

While the architecture closely mirrors that of Run 3, there are notable improvements (refer to figure 3). Apart from a significant simplification in the data model, resulting in a substantial reduction in the number of tables, the CREST system introduces a REST API for conditions data management. This innovation entirely decouples client code from the underlying storage implementation. Consequently, clients are no longer obligated to understand how the storage system is internally structured.

The most significant distinction lies in the capability to maximize the utilization of the caching layer by establishing a clear demarcation between metadata, such as the validity intervals for each individual conditions data payload, and the conditions payload itself. This segregation is achieved at the level of the REST API definition through access to identical conditions data sets via a unique *key*.

To grasp the benefits of such a data model design, we can examine the current utilization of database servers in both ATLAS and CMS. We assume that both experiments possess a comparable amount of conditions data.

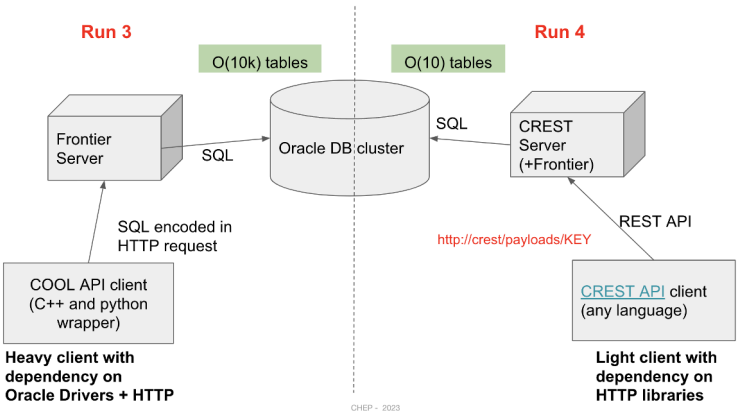

Figure 3. Conditions Data: comparison between present and proposed architectures.

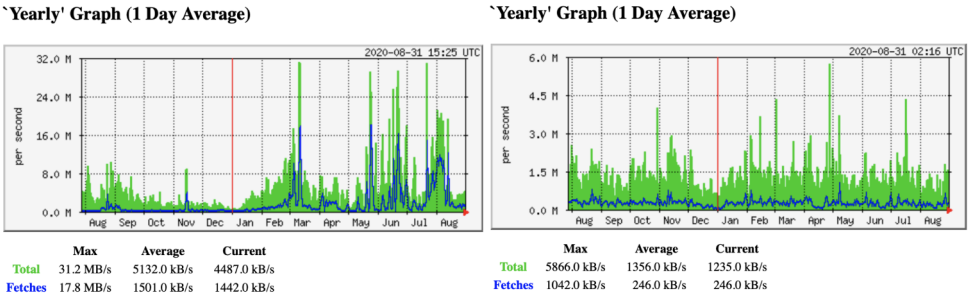

Figure 4. Frontier/Squid usage in ATLAS and CMS.

We have extracted the volumes of data retrieved from Oracle ("Fetches") and from the Squid system ("Total") for both the ATLAS and CMS experiments, utilizing official monitoring plots. These figures are presented in table 1. Additionally, the yearly (single day average) graph is displayed in figure 4. The considerable variability observed in the case of ATLAS suggests a less optimal utilization of the caching system, likely stemming from the manner in which clients request the necessary data. A more comprehensive investigation into the underlying causes of these inefficiencies has been conducted within ATLAS, leveraging the logging data from the Frontier servers [9].

Table 1. ATLAS and CMS Frontier/Squid monitoring.

Experiment	Type	Fetches (MB/s)	Total (MB/s)	Ratio
ATLAS	Year Avg	1.5	5.1	30%
ATLAS	Year Max	17	31	54%
CMS	Year Avg	0.25	1.35	20%
CMS	Year Max	1	5.8	17%

4 CREST architecture

The CREST system follows a multi-tier model architecture. In this arrangement, the back-end remains a relational database that employs a concise collection of tables to oversee the management of conditions data metadata and payloads. Simultaneously, a web server is fashioned as the front-end. This web server actualizes a REST API, abstracting the direct interaction with the database. A collection of client libraries has been prepared to facilitate the utilization of the REST API from various programming languages. Notably, a C++ client has been meticulously developed for usage from Athena clients. The existing state of the system is elaborated upon in this section.

4.1 CREST REST API

The REST API is documented using OpenAPI specifications [10] in a YAML format. This API essentially outlines the URL paths made accessible through the CREST server, as well as the data objects exchanged between the server and the client (JSON is employed for data transmitted via HTTP). Opting for a standardized set of specifications for API description offers the advantage of being compatible with a diverse range of tools, enabling the generation of code for both server stubs and clients across various programming languages and frameworks.

The API description encompasses a comprehensive array of metadata elements essential for conditions data management, including tags, intervals of validity (IOVs), and global tags. Remarkably, these metadata components exhibit high similarity between the current ATLAS system (COOL) and the CMS data model.

4.2 CREST server

The CREST server is constructed using established Java technologies [11], specifically relying on specifications like JAX-RS and JPA, alongside the Spring Boot [12] framework.

The description of the REST API via OpenAPI facilitates the generation of server stubs within the Jersey framework [13], employing standard generation tools [14].

The robust support and seamless interoperability within the Java ecosystem contribute to the stability of the server code over time. This ecosystem's flexibility allows for effortless transitions between various sets of implementation libraries. For instance, a switch between web servers such as Tomcat [15] and Undertow [16] can be accomplished through a simple adjustment in the CREST server's build file, without necessitating internal code alterations.

Utilizing JPA implementations, such as Hibernate [17], for standardized database access provides the advantage of concise object-relational mapping syntax, while still retaining the option for deeper optimization of specific queries.

The project's source code (for the server and the related libraries) is hosted on GitLab at CERN [1].

4.3 CREST client libraries and tools

Interactions with the CREST server occur through utilization of the REST API. The official client takes the form of a C++ implementation, and it is integrated into Athena conditions data services.

To assess the functionality of the current software for a specific subsystem, we can carry out trials by migrating the conditions data for that given subsystem into the CREST database.

[1] https://gitlab.cern.ch/crest-db

To facilitate this migration from the existing COOL DB, a dedicated tool has been crafted. This converter tool can be configured to selectively copy "tags" from COOL to CREST. This operation involves employing the COOL API for reading and the CREST C++ client for data insertion via the CREST server into Oracle. The tool also provides a set of logging information, offering the added benefits of profiling and debugging the copying process.

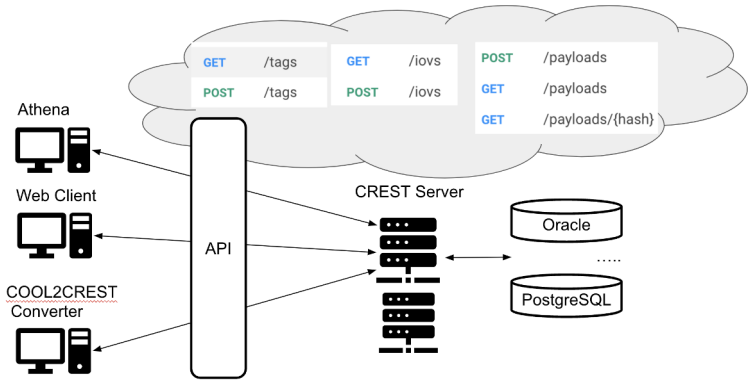

Figure 5. CREST deployment and test infrastructure

4.4 CREST deployment and test infrastructure

To assess the functionality of the CREST system, we have employed a cloud-based deployment approach. A range of distinct CREST servers, each employing different underlying schemes, are accessible to developers and Athena testers. This setup allows us to experiment with various database platforms, including Oracle and Postgres, ensuring that the server code remains well separated from the specifics of the underlying storage technology. A depiction of this deployment scheme is presented in figure 5.

As of now, the official deployment resides in a Kubernetes [18] cluster, utilizing machines within the CERN openstack infrastructure. This cluster also serves to deploy a caching system that relies on Varnish. This caching system plays a crucial role in validating the architecture during our initial large-scale tests.

ATLAS has laid out plans to introduce a demonstrator for CREST utilization by the conclusion of 2023. This demonstrator is set to undergo testing within the High-Level Trigger (HLT) workflow. This choice stems from the fact that online workflows, such as the HLT, impose more demanding requirements in terms of caching. Notably, conditions data like luminosity and beam-spot need to be refreshed regularly, sometimes even at the granularity of each luminosity block [2].

5 Conclusions

We have detailed the modifications made to the ATLAS conditions data management infrastructure in preparation for the Run 3 data acquisition phase. These adjustments are geared towards readying both the experiment and conditions data users for an enhanced architecture set to be employed in the upcoming data acquisition (during Run 4). Additionally, we have

[2] A luminosity block is defined as a period with stable luminosity (generally about one minute in duration).

elucidated the novel architecture known as the CREST project and highlighted its distinctions from the current system. We have emphasized the core differences and enhancements that the new architecture aims to tackle.

References

[1] ATLAS Collaboration, JINST **3**, S08003 (2008), https://dx.doi.org/10.1088/1748-0221/3/08/S08003

[2] P.J. Laycock, D. Dykstra, A. Formica, G. Govi, A. Pfeiffer, S. Roe, R. Sipos, Journal of Physics: Conference Series **1085**, 032040 (2018), https://dx.doi.org/10.1088/1742-6596/1085/3/032040

[3] A. Valassi, R. Basset, M. Clemencic, G. Pucciani, S.A. Schmidt, M. Wache, *COOL, LCG conditions database for the LHC experiments: Development and deployment status*, in *IEEE Nuclear Science Symposium Conference Record, 2008. NSS '08* (2008), pp. 3021–3028, https://doi.org/10.1109/NSSMIC.2008.4774995

[4] D. Dykstra, J. Phys.: Conf. Ser. **331**, 042008 (2011), http://iopscience.iop.org/1742-6596/331/4/042008

[5] *HEP Software Foundation*, https://hepsoftwarefoundation.org/

[6] M. Bracko, M. Clemencic, D. Dykstra, A. Formica, G. Govi, M. Jouvin, D. Lange, P. Laycock, L. Wood, TBD (2019), https://www.osti.gov/biblio/1527431

[7] G.A. Stewart, et al., J.Phys.Conf.Ser. **762**, 012024 (2016), https://iopscience.iop.org/article/10.1088/1742-6596/762/1/012024

[8] L. Rinaldi, A. Formica, E.J. Gallas, N. Ozturk, S. Roe, EPJ Web Conf. **214**, 04052 (2019), https://doi.org/10.1051/epjconf/201921404052

[9] A. Formica, N. Ozturk, M. Si Amer, J.L. Bahilo, E.J. Gallas, I. Vukotic, EPJ Web Conf. **245**, 04032 (2020), https://doi.org/10.1051/epjconf/202024504032

[10] *OpenAPI Initiative. (2023). OpenAPI specifications, version 3.1.0.*, https://www.openapis.org

[11] *Oracle corporation. (2017). Java Platform. Enterprise Edition (Java EE) Specification, version 8*, https://javaee.github.io/javaee-spec/

[12] *Spring. Spring Boot (Version 2.3.0). Available from*, https://spring.io/projects/spring-boot

[13] *Jersey 2 JAX-RS API (Version 2.35). Available from*, https://eclipse-ee4j.github.io/jersey

[14] *OpenAPI Tools. (2020). OpenAPI Generator (Version 4.3.1). Available from*, https://openapi-generator.tech/

[15] *Apache Software Foundation. (2022). Apache Tomcat (Version 9.0.59). Available from*, https://tomcat.apache.org/

[16] *JBoss Community, WildFly Project. (2019). Undertow (Version 2.2.24). Available from*, https://undertow.io/

[17] *Hibernate Community. (2021). Hibernate ORM (Version 5.6.15.Final). Available from*, https://hibernate.org

[18] *Cloud Native Computing Foundation. (2022). Kubernetes (Version 1.23.0). Available from*, https://kubernetes.io