EPJ Web of Conferences **295**, 01011 (2024) https://doi.org/10.1051/epjconf/202429501011
CHEP 2023

Calibration and Conditions Database of the ALICE experiment in Run 3

Daniel-Florin Dosaru[3,*], *Costin* Grigoraş[1,**], *Rafał* Mucha[2,***], and *Michał* Trzebuniak[2,****]

[1]CERN, Esplanade des Particules 1, 1211 Geneva 23, Switzerland
[2]AGH University of Krakow, al. Adama Mickiewicza 30, 30-059 Kraków, Poland
[3]"Politehnica" University of Bucharest, Splaiul Independenţei no. 313, sector 6, Bucharest, Romania

Abstract. The ALICE experiment at CERN has undergone a substantial detector, readout and software upgrade for the LHC Run 3. A signature part of the upgrade is the triggerless detector readout, which necessitates a real time lossy data compression from 1.1 TB/s to 100 GB/s performed on a GPU/CPU cluster of 250 nodes. To perform this compression, a significant part of the software, which traditionally is considered off-line, was moved to the front-end of the experiment data acquisition system, for example the detector tracking. This is the case also for the various configuration and conditions databases of the experiment, which are now replaced with a single homogeneous service, serving both the real-time compression, online data quality checks and the subsequent secondary data passes, Monte-Carlo simulation and data analysis.
The new service is called CCDB (for Calibration and Conditions Database). It receives, stores and distributes objects and their metadata, created from online detector calibration tasks and control systems, from offline (Grid) workflows or by users. CCDB propagates the new objects in real time to the Online cluster and asynchronously replicates all content to Grid storage elements for later access by Grid jobs or by collaboration members. The access to the metadata and objects is done via a REST API and a ROOT-based C++ client interface which streamlines the interaction with this service from compiled code while plain curl command line calls are a simple access alternative.
In this paper we will present the architecture and implementation details of the components that manage frequent updates of objects with millisecond-resolution intervals of validity and how we have achieved an independent operation of the Online cluster while also making all objects available to Grid computing nodes.

1 Introduction

With the paradigm shift of the ALICE experiment from triggered data taking to triggerless readout and real time data compression came a couple of new constraints for the conditions data collection and distribution. In LHC Runs 1 and 2 the calibration and condition data was collected by a service called Shuttle [1] and saved to files at the end of the experiment run (a

*e-mail: daniel.dosaru@upb.ro
**e-mail: costin.grigoras@cern.ch
***e-mail: rafal.mucha@cern.ch
****e-mail: trzebuniak.michal@gmail.com

contiguous period of unchanging conditions, usually lasting a few hours and limited by the accelerator fill duration). These calibration objects were used in asynchronous processing of the experiment data, run on Grid nodes anywhere from hours to years later. In LHC Run 3 part of the data processing was moved to a cluster of machines close to the experiment where real time data compression algorithms take as input the most recent calibration information, as described in the ALICE Upgrade Technical Design Report [2]. These processes also accumulate information and produce new calibration objects representing the updated state of the detectors [3]. The new Calibration and Conditions Database (CCDB) service was implemented as both an archival tool as well as a real time object distribution method, effectively implementing a calibration feedback loop in the Online environment.

CCDB's design was driven by a requirement to represent the interval of validity of objects with millisecond precision. It had to support user-defined metadata associated to objects and queries by it. Additional considerations in designing the new CCDB service were the number of entries expected to be stored by this service, the rate of queries in both the real time data compression stage as well as those coming from asynchronous data processing on the Grid and the availability of the services during the critical periods of data taking.

In the following sections we will address the architecture of the system, database structure and deployment, API and client implementation details, binary data replication to the distributed storage infrastructure as well as current status of the service after one year of operation in production.

2 Architecture

A key design choice of CCDB was to separate the data from the metadata. Most of the paper will focus on handling the metadata (object definition, matching and presenting it to clients) while the data itself is assumed to be a binary blob transported without any processing and handled as described in Sect. 6. While CCDB is content and format agnostic, in practice all files are ROOT[4]-serialized objects.

Objects are all associated a Version 1 UUID, issued by the first receiving party and encoding the request timestamp, IP address of the client plus random bits. These unique object identifiers are later used in the communication protocol to identify recently used objects that clients have in memory and can be potentially reused. See Sect. 4.1 for more details.

Considering the high rate of new calibration objects created during the data taking of the experiment, we have designed the entire system to use absolute timestamps expressed in epoch milliseconds. Each object has an Interval of Validity (IoV) that is defined as an interval that includes the Start of Validity (SoV) but excludes the End of Validity (EoV). Thus $IoV = [SoV, EoV)$.

In addition to the IoV and an unique identifier, a creation timestamp and other metadata key-value pairs are automatically assigned to each object, extracted from the request itself (like original file name and content type) or explicitly specified by the creator.

2.1 Object matching and snapshot access

CCDB's main role is to locate the correct calibration or condition object needed by a processing entity. For that it needs two key pieces of information:

- a path – string with slash separators organizing the objects in a tree-like structure. It is defined as a constant in the client code, by convention setting the first token to the name of the detector that defines it. 350 paths for 25 detectors and 10 global areas were defined so far and are accessed when needed by various tasks. In particular the online compression processes use 250 of them. Examples: *TPC/Calib/Align*, *CPV/PedestalRun/DeadChannels*;

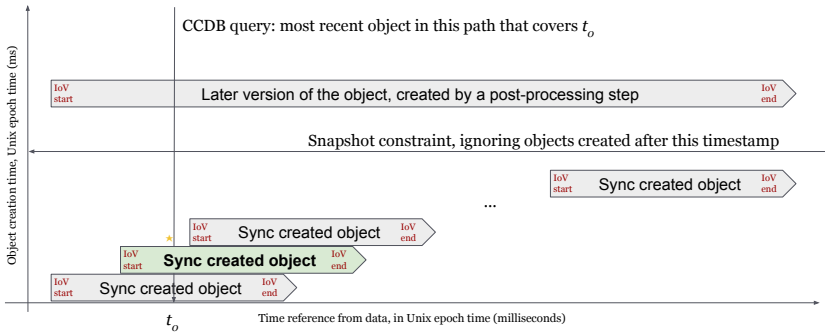

Figure 1. IoV and snapshot queries in a path; the most recently created matching object is returned, unless a snapshot timestamp is indicated.

- a reference timestamp t_0 – epoch millisecond at which the object must be valid at, thus $SoV <= t_0 < EoV$. The client code will infer this from the chunk of data it is processing.

Given these two parameters, CCDB returns the most recently created object in that path who's IoV contains t_0. Optionally, clients can specify a cutoff on the object creation time, again in epoch milliseconds, represented on the vertical axis above. That allows creating snapshots, reducing the namespace to what objects were available at the indicated timestamp in order to reproduce results from a previous data processing, for the same reference timestamp t_0. An illustration of this logic is presented in figure 1 where the later version is ignored to go to the original object created during data taking when a snapshot timestamp is indicated.

3 Database backing

The choice of a database backend to store the metadata was driven by the IoV queries described above. Considering a simplified matching query below:

```
SELECT * FROM ccdb WHERE
    path='TPC/Calib/Align' AND
    validity CONTAINS data_timestamp AND
    creation_timestamp < snapshot_timestamp
ORDER BY creation_timestamp DESC LIMIT 1;
```

the main issue is the *CONTAINS* condition, that could be expanded to something like:

```
start_of_validity <= data_timestamp AND data_timestamp < end_of_validity
```

for which only one index would be used efficiently while the other comparison would be sequentially checked. In this case and with an estimated 100 million objects created in a large path in LHC Run 3 and Run 4 data taking, an implementation using independent columns for *SoV* and *EoV* does not scale, in either MySQL or PostgreSQL.

3.1 Database backend and structure

The solution was to use PostgreSQL's *tsrange* data type, that stores a time interval as a single column with dedicated operators [5], including the *CONTAINS* one (@>). This operator is supported by a GiST index on the column for an efficient matching of both sides of the IoV.

Furthermore, PostgreSQL implements a generic key-value store data type, *hstore*, that can be used to store the generic metadata associated with each object. While a few frequently

used details are stored as separate columns (id, path, validity, creation time, object size, checksum), other implicit or user-defined metadata are stored in a single *hstore* column.

To prove the suitability of this solution we have run synthetic benchmarks, inserting the target 100 million objects in a single namespace and querying random timestamps from it. A single database server could serve 20'000 requests per second, which covers the foreseen uses in both the real time data compression as well as accesses from Grid workloads.

3.2 Online-Offline separation

A design requirement of the ALICE experiment was for it to be able to operate even when the experimental area is disconnected from the campus area where the Grid services are hosted.

For independent operations we need a full replica of the metadata in each location while allowing new objects to be created on either side of the connection. We have implemented this using Bucardo [6], a tool that tracks the database changes and applies them asynchronously to the other side of the connection for an eventual consistency between the two primary instances.

In addition to the metadata, binary data might also not be available on the experiment side. A local disk cache reduces the impact of network interruptions and we have a large (20 TB) area dedicated to caching binary content. Any file that is created during data taking first goes to this area and then is replicated out, and any file that is accessed is also going to the same area. Since all necessary objects are loaded at the very beginning of any data taking period, they will be present on the local disk for further accesses, if needed.

3.3 Deployment

For increased reliability and capacity to serve requests we have implemented PostgreSQL's native primary/standby replication on two servers on each side of the experiment. Each server also runs an instance of the CCDB service and connects to a Pgpool-II [7] instance on *localhost* that in turn load balances the queries between the available PostgreSQL instances.

CCDB clients are only given a DNS alias of the service and thus randomly connect to one of the servers. They have to be ready to follow HTTP or Xrootd [8] redirects to the actual location of the data, because in the Grid environment the best location to retrieve the data from is the closest storage element to where the client runs and not the central services.

In the Online environment as a trigger on new objects being created we also distribute the binary content to the entire cluster via multicast. Dedicated CCDB services listen for these multicasts on each data processing node and cache the object content in memory, ready to serve these objects to real-time compression workflows running on the respective nodes with minimal latency. All these components can be seen in figure 2.

4 API and C++ client code

We chose a RESTful design for the new implementation, relying on standard protocols, libraries and tools to handle the communication. CCDB [9] services are Java applications embedding a Tomcat engine to handle the HTTP requests. They also link to our Grid middleware JAliEn [10] to handle client authentication and interactions with the Grid infrastructure.

Authentication to the services is built on top of the same X.509 scheme that we use for Grid users and jobs. Depending on the instance, path and operation, clients might be redirected to the HTTPS endpoint where they have to present a Grid certificate and an internal ACL is further applied to restrict users' write permissions to parts of the namespace.

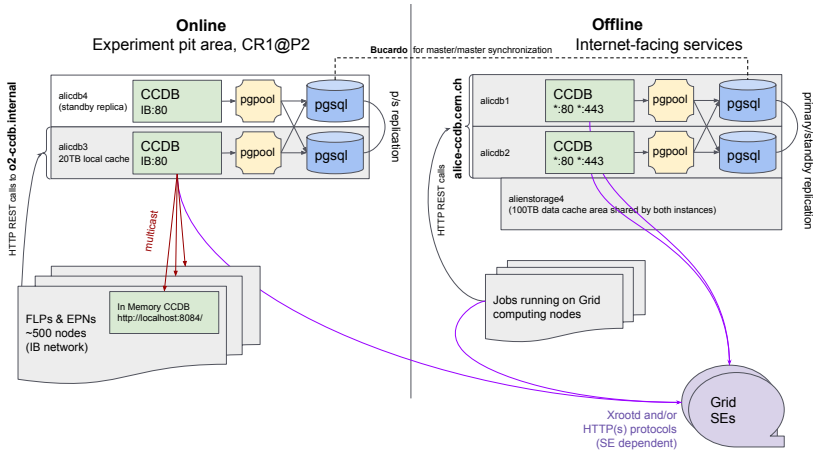

Figure 2. Deployment diagram showing the redundant services deployed on both sides of the experiment, the real time multicast distribution of new objects in the Online environment. All objects are copied to Grid storage elements and Grid jobs are redirected to this resource for read operations.

4.1 API

Standard tools such as *cURL* or *wget* can be used to perform operations and in the following we show a few examples of storing and retrieving data from the command line. A command to upload an object from a local file to the *TPC/Calib/Align* path with an interval of validity between 1 and 100000 and a user-defined metadata *quality* tag with the value of *2* is:

```
curl -F blob=@/tmp/file \
http://alice-ccdb.cern.ch/TPC/Calib/Align/1/100000/quality=2
```

to which the server returns as headers the confirmation code as well as the local object path:

```
HTTP/1.1 201
Location: /download/a329fcc6-9818-4d2e-a5af-16ca73686cf2
```

Retrieving an object can be done with a simple GET request, indicating a timestamp within the IoV of the previously created one, for example *50000*:

```
curl http://alice-ccdb.cern.ch/TPC/Calib/Align/50000
```

to which the server will return all metadata as HTTP headers, together with the content locations, pointing by default to the Grid data management URL (stating with *alien://*).

```
HTTP/1.1 303
Location: alien:///alice/.../a329fcc6-9818-4d2e-a5af-16ca73686cf
ETag: "a329fcc6-9818-4d2e-a5af-16ca73686cf2"
Valid-From: 1
Valid-Until: 100000
quality: 2
Content-Location: alien:///alice/.../a329fcc6-9818-4d2e-a5af-16ca73686cf
Content-Location: /download/a329fcc6-9818-4d2e-a5af-16ca73686cf2
Content-Disposition: inline;filename="o2-tpc-IDCZero_1681052400217.root"
ObjectType: o2::tpc::IDCZero
runNumber: 534275
```

The last three lines are from a production object showing examples of real file names, run number associated to the object and the serialized object type. Requests can further filter the objects by arbitrary metadata values specifying additional */Key=Value* URL suffixes, e.g.

```
curl http://alice-ccdb.cern.ch/TPC/Calib/Align/50000/quality=1
```

that will return a *HTTP 404* since this constraint is not matched by any existing object.

Other functionality can be triggered by client-specified headers. A very important one is *If-None-Match*, indicated by clients that already have a version of the object in memory and would like to reuse it, if possible. In that case the server does not have to provide the content any more and only confirms the validity with a *HTTP 304 Not Modified* response code. Otherwise the server can return a different object to use. Exemplifying in a command line a client that has moved on with the processing to timestamp *76543* and would like to reuse the existing object with id *a329fcc6-9818-4d2e-a5af-16ca73686cf2*, the call would be:

```
curl -H 'If-None-Match: a329fcc6-9818-4d2e-a5af-16ca73686cf2' \
http://alice-ccdb.cern.ch/TPC/Calib/Align/76543
```

Similarly the clients can specify the snapshot timestamps with *If-Not-After* and *If-Not-Before* HTTP headers, restricting the creation time of the objects to arbitrary time intervals.

4.2 C++ client and connection pooling

While command line tools may serve as a temporary solution, it is crucial for our processing framework, developed in C++, to seamlessly interact with objects within its native environment. For that we have implemented helper functions [11] that interface with the framework to receive the path, timestamp and other constraints and build the equivalent HTTP request. In turn the utility class uses *libcurl* to perform the actual request and, when needed, the JAliEn ROOT plugin [12] to pass the X.509 credentials and to interact with the Grid storage infrastructure. The content is streamed over the network from the closest storage node holding it, by default using the Xrootd protocol.

With the above implementation requests are independently handled and a new connection is established for each object download or validation request with the corresponding new TCP socket overhead. Which is a particular burden on the real time data compression that has tight execution constraints since about 200 paths have to be validated at every new data block. To speed this up we have developed an enhanced C++ utility [13] that can reuse existing HTTP sockets on Keep-Alive requests, reducing the overhead of managing high frequency socket create and destroy cycles. It has the additional functionality of reusing existing TCP connections, with the ability to perform multiple requests in parallel. Furthermore, the new utility is prepared to receive arrays of requests, for now processing them in sequence internally but ready for HTTP 1.1 pipelining and eventually move to HTTP 2 asynchronous operations.

5 Data management

5.1 Grid storage infrastructure

Offloading the object content to the distributed storage infrastructure is an optional mechanism in CCDB, enabled on the production instances. It is asynchronous, meaning that objects are first stored on the local disk on the receiving server and then uploaded to several endpoints. Our current policy is to replicate to eight distinct endpoints, geographically distributed so that any client can find a nearby copy of the calibration objects. On all these instances both the Xrootd as well as HTTP access protocols are enabled, for native access for our ROOT-based C++ applications while also allowing third-party applications access to the same data.

Reading back data is also transparently falling back to downloading the missing local file from a Grid endpoint if it is missing from the local disk. This mechanism is used in particular

by the Online CCDB instance, to ensure a local copy of each needed calibration file is available in case of network interruptions. The download mechanism is triggered automatically by real time compression clients requesting objects, which in turn triggers their multicast distribution to all other services in the local network as described in the next section.

The vast majority of the CCDB clients are Grid jobs, accessing the Offline CCDB instance. The example request in Sect. 4.1 shows how a client is pointed to an *alien:∕∕* URL by default, while also shown the alternative *Content-Location* of a local copy on the CCDB server. For scalability reasons clients should only use the Grid replica which is done by calling a *TFile::Open* on the *alien:∕∕* path. The custom *TGrid* ROOT plugin [12] handles this protocol and contacts the experiment file catalogue to locate the physical replicas of these files. The returned URL list is sorted function of the client network location, with the client trying to open them in turn in this order until one of them works. By default the Xrootd endpoint of each storage is accessed, thus URLs with a *root:∕∕* protocol. This is handled natively by ROOT, which relies on the Xrootd client library to open and stream the content of the file to memory, deserializing and casting it to the templated CcdbApi [11] API call.

5.2 Real-time object distribution in the Online environment

Real-time data compression workflows both consume and produce calibration objects. The updated calibration information has to be quickly propagated to all other instances of the workflow on nearby machines. To bring the updated calibration data close to the processing cores we run dedicated CCDB server instances on each data compression node that keep all recently seen objects in memory and subscribe to a multicast address for updates. As it can be seen in figure 2 the CCDB server in the Online environment pushes the content of new objects to all nearby nodes. Clients query their *localhost* CCDB in-memory instances. If no data is found they are redirected to the authoritative CCDB server that, with the assumption that this object is going to be needed by everybody else, will multicast its content. To warm up the caches, at the beginning of any data taking activity, one client goes through all paths in the namespace and requests the most recent object in each. With this mechanism we reduce the latency of accessing the most recent calibration object to a *localhost* HTTP request while also bringing all Grid objects to the Online instance, if not there yet, to keep its operations independent from the external network to the ALICE experiment.

Particular attention [14] was given to the possibility of losing UDP packets. For that we fragment the content ourselves and wrap data and metadata in packets that each include the ID, path, IoV and byte range in them. That way the CCDB multicast receivers can invalidate outdated objects from memory from the very first received packet as well as recover any missing byte ranges with vector HTTP requests from the authoritative CCDB server.

6 Current status

During LHC Run 3 in the production system we have accumulated so far (up to 2023.07.28) 6.8 million objects for a total of 1.56 TB of data. Most of the content (1.48 TB of data in 2.57 million objects) is created by a single detector, TPC, on average 2.2 new objects per second during experiment data taking.

Accesses from the distributed Grid workloads average to 550 requests per second over the last month of activity, the servers answering on average in 7 milliseconds per request. With synthetic tests (requesting objects in a loop from Grid jobs) we have observed capacity in excess of 10000 requests per second per server, so we are confident the current setup can support the expanding Grid infrastructure. Moreover, the chosen solutions (PostgreSQL replicas and load balancing with Pgpool-II) allow rapid deployment of more capacity.

7 Conclusions

The new ALICE CCDB service is now in production, with separate instances for Online (real time data compression) and Offline (Grid workloads) loosely coupled via a Bucardo service. The chosen database backend and database structure are supporting well the current workloads with plenty of spare capacity while the multicast distribution mechanism in the Online environment provides an efficient calibration loop between the more than 500 nodes involved in this activity.

Our focus is now shifting to the client side, optimizing the number of requests (changes in client logic to keep objects in memory) and the reuse of sockets and switching HTTP protocol version for a reduction of the global turnaround time of large batches of CCDB queries.

References

[1] J.F. Grosse-Oetringhaus, C. Zampolli, A. Colla, F. Carminati, for the Alice Collaboration, *The ALICE online-offline framework for the extraction of conditions data*, in *Journal of Physics: Conference Series* (IOP Publishing, 2010), Vol. 219, p. 022010, http://cds.cern.ch/record/1269925?ln=en

[2] P. Buncic, M. Krzewicki, P. Vande Vyvre, Tech. rep. (2015), http://cds.cern.ch/record/2011297?ln=en

[3] M.O. Schmidt, EPJ Web Conf. **245**, 01003 (2020)

[4] P. Canal, B. Bockelman, R. Brun, *ROOT I/O: The fast and furious*, in *Journal of Physics: Conference Series* (IOP Publishing, 2011), Vol. 331, p. 042005, https://iopscience.iop.org/article/10.1088/1742-6596/331/4/042005

[5] PostgreSQL, *Range functions and operators*, https://www.postgresql.org/docs/13/functions-range.html, last accessed: June 15, 2023.

[6] Bucardo, *Web site*, https://bucardo.org/, last accessed: June 15, 2023.

[7] Pgpool-II, *Web site*, https://www.pgpool.net/, last accessed: June 15, 2023.

[8] M. Kompaniets, O. Shadura, P. Svirin, V. Yurchenko, A. Zarochentsev, *Integration of XRootD into the cloud infrastructure for ALICE data analysis*, in *Journal of Physics: Conference Series* (IOP Publishing, 2015), Vol. 664, p. 022036, http://cds.cern.ch/record/2134539?ln=en

[9] C. Grigoraş, *Source code*, https://gitlab.cern.ch/grigoras/ccdb-local/, last accessed: June 15, 2023.

[10] M.M. Pedreira, C. Grigoras, V. Yurchenko, *JAliEn: the new ALICE high-performance and high-scalability Grid framework*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 03037, https://cds.cern.ch/record/2701497?ln=en

[11] S.W. Barthelemy von Haller, *Source code*, https://github.com/AliceO2Group/AliceO2/blob/dev/CCDB/src/CcdbApi.cxx, last accessed: June 15, 2023.

[12] A. Grigoras, C. Grigoras, M. Pedreira, P. Saiz, S. Schreiner, *JAliEn–A new interface between the AliEn jobs and the central services*, in *Journal of Physics: Conference Series* (IOP Publishing, 2014), Vol. 523, p. 012010, http://cds.cern.ch/record/2026281?ln=en

[13] M. Trzebuniak, *Source code*, https://github.com/AliceO2Group/AliceO2/blob/dev/CCDB/src/CCDBDownloader.cxx, last accessed: June 15, 2023.

[14] D.F. Dosaru, *Master thesis*, https://github.com/dosarudaniel/ReliableMulticastForALICE/blob/master/Diploma_project_documentation.pdf, last accessed: June 15, 2023.