Designing a future Conditions Database based on LHC experience

D Barberis\textsuperscript{1}, A Formica\textsuperscript{2}, E J Gallas\textsuperscript{3}, G Govi\textsuperscript{4}, G Lehman Miotto\textsuperscript{5}, A Pfeiffer\textsuperscript{5}
on behalf of the ATLAS and CMS collaborations

\textsuperscript{1} INFN Sezione di Genova and Dipartimento di Fisica, Università di Genova, Genova, Italy
\textsuperscript{2} CEA/Saclay IRFU/SEDI, 91191 Gif-sur-Yvette, France
\textsuperscript{3} Department of Physics, Oxford University, Denys Wilkinson Building, Keble Road, Oxford OX1 3RH, United Kingdom
\textsuperscript{4} Fermi National Accelerator Laboratory, Batavia, IL 60510, USA
\textsuperscript{5} CERN, CH-1211 Genve 23, Switzerland

E-mail: andrea.formica@cea.fr

Abstract. Starting from the experience collected by the ATLAS and CMS experiments in handling condition data during the first LHC run, we present a proposal for a new generation of condition databases, which could be implemented by 2020. We will present the identified relevant data flows for condition data and underline the common use cases that lead to a joint effort for the development of a new system.

Condition data is needed in any scientific experiment. It includes any ancillary data associated with primary data taking such as detector configuration, state or calibration or the environment in which the detector is operating. Condition data typically reside outside the primary data store for various reasons (size, complexity or availability) and are best accessed at the point of processing or analysis (including for Monte Carlo simulations). The ability of any experiment to produce correct and timely results depends on the complete and efficient availability of needed conditions for each stage of data handling. Therefore, any experiment needs a condition data architecture which can not only store conditions, but deliver the data efficiently, on demand, to potentially diverse and geographically distributed set of clients. The architecture design should consider facilities to ease conditions management and the monitoring of its conditions entry, access and usage.

1. Usage of Conditions Data in ATLAS and CMS experiments

During Run 1 of the Large Hadron Collider (LHC) at CERN (2009-2013), the ATLAS \textsuperscript{1} and CMS \textsuperscript{2} experiments each recorded and analyzed hundreds of petabytes of data and published many physics results. Achieving these results required extensive processing and analysis of massive datasets in a largely distributed computing environment. In addition to event data stored in distributed file systems, other data (called Conditions Data) related to the data taking and detector monitoring are essential at nearly every stage of data handling. ATLAS and CMS used different infrastructures and software layers to store and manage their \textasciitilde1.5 TB and \textasciitilde300 GB of Run 1 condition data, respectively.

In all cases, condition data are organized by folder, where each folder contains related condition data (usually from individual detector component) and each detector component may
have as many folders as needed. A folder identifies a coherent set of values for conditions (called payload), which is associated to a specific “Interval of Validity” (IOV) representing the time range over which that set of conditions is valid. For condition data which are mutable (may have more than one version in the same IOV, such as in the case of evolving calibration constants), each version is labeled with a Tag so that specific sets of conditions may be distinctly identified to be used in any data processing.

We give a brief overview of the ATLAS and CMS condition database systems in sections 1.1 and 1.2. For both experiments, primary condition database storage is within an Oracle DBMS [3], provided and administered by CERN IT department. The DB maintenance includes storage management, data replication and archiving. In addition, each experiment has a small group of database experts providing support for application development.

1.1. ATLAS Conditions Database
In Run 1, the ATLAS collaboration deployed a system based on the LCG Conditions Database infrastructure and the COOL API, both developed mainly by CERN IT [4]. This infrastructure will be used by ATLAS also during Run 2, with enhancements described elsewhere in this conference [5]. The chosen deployment consists of 32 schemata having between them over 2000 independent folders (logical sets of conditions entities). There are more than 10,000 underlying relational tables comprising the storage and organisation of the system. They are hidden from the general client by using the COOL API, a C++ library based on a software layer called CORAL (also developed by CERN IT). CORAL manages the actual database access and the queries that should be issued (partially hiding the SQL complexity) and supports multiple relational platforms: SQLite, Oracle, and MySQL. Using COOL, the client (in general a system expert) is able to manage his own condition data structures as well as the data they contain (their payload).

1.2. CMS Conditions Database
In Run 1, the CMS collaboration deployed a system independent of COOL but also based on CORAL for access to its Oracle based storage, with a similar number of schemata and folders. For Run 2, after a review of their scalability and use cases, and with a desire to simplify their architecture, CMS implemented a deep reorganisation of their system as described elsewhere in this conference [6]. While significant changes were made, the system used in Run 2 remains reliant for database access and management on CORAL methods (or via the Python wrapper of CORAL).

1.3. Conditions software and distribution architecture
Figure 1 gives an overview of the conditions distribution architecture along with the places in the dataflow where the conditions software API calls are located. In both experiments, the approach is to use C++ or Python-based software clients to read and write conditions. While this simple client-server architecture works fine for a small number of clients, it failed in scalability to deliver needed condition data to what can be ∼10K of simultaneously executing reconstruction jobs on the grid, each needing to retrieve some conditions data to succeed. To address this limitation, a Java application called Frontier [7], originally developed by CMS and now jointly maintained by experts from both experiments, was deployed to provide to clients efficient access to the conditions data via simple HTTP protocols. Frontier is deployed in Apache Tomcat servers, and uses JDBC to access (in read-only mode) the Oracle backend. Since many jobs can ask the system for the same conditions, a caching layer is deployed on top of the Frontier Tomcat server, based on a Squid proxy. The proxy caches the data response with the query encoded inside the URL so that any job sending an identical HTTP request can then be served the cached conditions data content. From the client side, a special plugin was developed in order to allow
Figure 1. Run 1 infrastructure for Conditions Data in ATLAS and CMS.

CORAL to send the query to Frontier instead of sending it directly to the Oracle server. By enhancing the conditions database architecture with the Frontier/Squids, the overall system has been found to effectively scale and provide all processing jobs with the conditions they need, at the same time limiting the accesses to the Oracle database to a minimum.

Exceptionally, the ATLAS online environment chose an alternative to Frontier called Coral Server (an extension to CORAL), which similarly caches requests and responses. In the system we propose for the future, we incorporate built-in mechanisms which include this ability to cache results by request since it is a critical functionality.

2. Feedback from Run 1 experience in ATLAS and CMS

We list here the major problems that were identified in the Run 1 infrastructure for both experiments.

1) **DB schema complexity**: A larger number of schemata and tables are generally more difficult to maintain, both for conditions management as well as for database administrators. In CMS during Run 1, this was considered a major weakness so in their redesign for Run 2, CMS decided to consolidate, using a single schema and just a few tables. Run 2 CMS payload data are stored in a single table with one payload column of BLOB type; the content of the BLOB is read and written using a Boost serialisation library for C++ objects. Other tables contain the metadata information for the Tags, IOVs and Global Tags (see below). In this collaboration between ATLAS and CMS conditions experts, we realized that the tables in the new CMS structure were found to be very similar to the metadata tables collected by ATLAS on their conditions database (which combines information over all the ATLAS schemata) [9]. This minimal set of tables is shown in Figure 2 and is used as a basis for the storage architecture for the new system.

2) **DB access layer, C++ API**: The API to access Conditions Data is, in both experiments, complex to install and use. The long term maintenance of these libraries from CERN IT could be an issue. Today many libraries are available inside the IT industry or in the open source market, which deliver more functionality in more programming languages (Java, etc. as well as Python and C++) and to a more diverse set of DBMS relational backends.

3) **Client - Server model**: The approach of a client-server (two tier) model has limitations in scalability. Furthermore, the usage of the client API directly inside the software framework of each experiment makes new development and deployment much more complicated: the validation cycle may stretch to take weeks because it requires multiple mandatory progressive validation steps in controlled test environments before the change can be fully deployed in
production software releases. We are exploring an alternative architecture which allows more
language independence on the client side and facilitates quicker validation and deployment.

Figure 2. Table structure for new new architecture.

3. Common use cases for Conditions Data management in ATLAS and CMS

We describe in the following paragraph the use cases which we have identified in both
experiments. We believe this commonality is not isolated to just ATLAS and CMS; other
experiments are expected to have conditions data workflows which would be satisfied by the
new system. The proposal for a common development, in addition to improving the present
infrastructure, has the great advantage of allowing a broader maintenance by a wider range of
developers from different experiments deploying it (as we already do today in the case of Frontier
development).

3.1. Consuming Conditions Data

This section enumerates the main data flows for conditions data in both experiments (shown
schematically in Figure 3).

1) **Online data processing**: This is the first level of processing executed by the trigger software
to determine which events are selected for offline processing. In this step the event data are
processed using the conditions data that were declared as valid in the past. By definition,
during this step, we do not have the time to recompute new conditions. In this environment,
changes are deployed less frequently. Major conditions updates can enter in the system only
after the experts have fully checked the data.

2) **Tier-0 data processing (Express and Bulk)**: The first offline data processing is executed
by CERN Tier-0 computing farms against all data accepted by the online trigger system. The
first stage is called Express Stream (ES) processing where a representative sample (about
10%) of all the data is analyzed. Subsystem experts are given a time window (about 36 to 48
hours) to study the results and deduce refined calibrations for the bulk processing stage. The
bulk processing then commences, processing all of the data using the best known conditions
in calibration and alignment at this point. Its output is then ready and usable for offline
physics analysis (or refinements to online data taking conditions).

3) **Reprocessing Campaign**: More detailed offline analysis of Tier-0 reconstructed
data inevitably reveals further optimizations in conditions as well as improvements in
reconstruction algorithms. In reprocessing campaigns, these optimized conditions data are used to re-reconstruct all the data providing improved parameters for physics analysis. These campaigns typically reprocess large volumes of data on the grid, rather than at Tier-0 centers.

4) **User analysis**: In some cases, certain types of condition data are needed for specific physics or detector studies which require more detailed conditions than needed in a typical data analysis. Also, physics calibrations (e.g. jet energy scales and b-tagging performances) should be available at this level. These data are used typically by all analyses that involve given physics objects. While this processing may not be required on all data streams, even the subset of data required may be quite large. These jobs, like reprocessing, generally are deployed using grid resources.

5) **Isolated processing**: The commercial market and some institutional organizations may have computing resources (HPC or supercomputing centers) which may be made available to experiments. The possibility to execute jobs in these centers may be exploited only if experiments are able to adapt their processes to be able to execute in these environments. For example, worker nodes may not have external connections to the grid, so delivery of conditions via Frontier is not available: the needed conditions for these jobs must be provided (file based) along with the input event-wise data files. These nodes may have a further constraint of limited memory, so minimizing the size of prepared conditions data becomes mandatory (since all conditions volume is too large). The new conditions architecture should include the capability to generate conditions files containing only the conditions expected to be needed for processing. Having this capability would also satisfy use cases such as the analysis of data in the context of Data Preservation Projects outside the experiment’s working environment, or users running analysis on machines that are not connected to the network.

4. **Proposal for a new Architecture to manage Conditions Data**
Combining our Run 1 experience with the varied requirements of the infrastructure including its distribution, we can define the basis for a new architecture to overcome the mentioned limitations. Sections 4.1 to 4.3 describe the main features foreseen for the new system.
4.1. Conditions Data Model

The minimal set of tables needed for conditions storage and tagging is shown in Figure 2. As illustrated, a Global Tag is a collection of specific subsystem Tags. Every global tag is in turn associated to a set of IOVs (versioning is implemented via insertion time), where each IOV corresponds to a time interval as well as to a hash reference into the Payload table. The payload is thus uniquely identified via a hash (MD5 or SHA) of its content, which avoids duplication of identical information in the relational database. These tables should be defined within a single database schema (as is the case for the CMS implementation for Run 2).

Other tables will be added related to monitoring: keeping track of (e.g.) the number of IOVs per tag, the evolution of payload size for each subsystem and so on. Such monitoring capabilities were add-ons in both experiments since it was missing from their core architectures: we believe that this structural metadata for the monitoring of the conditions storage and its evolution is an essential capability which should be included in any such system.

4.2. Conditions Distribution Architecture

A core component of the new system in an intermediate server (the logic tier in a classic three-tier architecture) which provides clients with conditions retrieval as well as insertion. The simplicity of the conditions storage architecture significantly reduces structural complexity on the database side which thereby simplifies the nature of interactions with it.

Its retrieval capability is similar to that used by the Frontier/Squid system (associating resources with a unique URI and caching that result to satisfy further identical requests). The resource representation is transmitted to the client in JSON or XML format.

The usage of the intermediate server disentangles completely the business components dealing with the database management aspect of the client, allowing the two systems to evolve separately via well identified interfaces. We can imagine that server code can then be modified without any changes in the client side, allowing much more flexibility in the code maintenance with respect to the present architecture.

4.3. Prototyping the new architecture

A prototype system is now in development incorporating existing open source WWW-based technologies shown schematically in Figure 4. We began with technologies in use in the Frontier servers, so that we can incorporate the successful deployment of systems already in place for the installation, distribution and monitoring of existing servers.

The prototype is a Java based application, developed under a Spring framework[10]. Inside Spring, several modules are available to profit from JPA specifications (Java Persistence API[11]).

![Figure 4. Prototype for the new architecture.](image-url)
and to implement, in an easy way, REST (Representational State Transfer) services. The prototype consists of a package containing the data model expressed in Java objects mapped to a generic relational backend by means of a JPA implementation (we chose Hibernate[12] which is widely used in the Java community). The business code for insertion and retrieval of data is implemented in another package using Spring repositories: here all queries are implemented, by means of a generic Java Query Language (very close to SQL) and converted into the appropriate database dialect depending on the database driver which is declared in configuration packages. A Web package delivers higher level functionalities and resources dedicated to administration (using authentication) or to any client (without authentication mechanisms) for data manipulation. This prototype is hosted at CERN (managed in a Git repository) and deployed on a dedicated machine which is accessible to a restricted group of ATLAS and CMS collaborators. It is deployed inside a Tomcat server, but the packaging of the application (which uses Maven[13]) allows to set up (by means of Spring profiles configuration), alternative deployments for JBoss WildFly[14] or Jetty[15] servers. A Python client which access the RESTful services via PycURL[16] has also been developed to start testing the prototype.

5. Work Packages

We have identified a set of 6 work packages which each develop a component of the new infrastructure: WP1-4 are essential components for a functionally complete architecture. WP1-3 deliver basic functionality, while WP4 is strongly suggested for storage optimization and cross checks, WP5 is encouraged to optimize server configuration for performing core services (WP3) and WP6 is highly useful to disentangle usage (understanding which conditions are needed for particular specific or generalized use cases).

5.1. WP1 - Conditions storage

This package investigates possible solutions for the payload storage and the optimisation of the indexing of the storage. In this context several technologies are considered and the performances evaluated. Object serialisation libraries are available today towards several destination formats: text (xml/json) or binary. The choice in the serialisation can have deep impact on the backend solutions that can be adopted for the payload storage. NoSQL technologies can be explored in this work package. It is important to take into account the long term access of the payload, and face any question related to backward compatibility at an early stage.

5.2. WP2 - Conditions metadata catalogue

This package defines the condition metadata catalogue and includes administrative services, which, for example, allow experts to create folders in the storage system. The catalogue describes the experiment sub-systems and contains defining qualities of each of its folders. It collects structural (per folder) metrics from the storage system for use in WP4 (storage monitoring). It contains the infrastructure for global tagging and instantiates the information needed by the Core Services (WP3) such as for access authentication.

5.3. WP3 - Conditions core services

This package comprises the middle-tier of the conditions data infrastructure, interacting with the Metadata Catalogue (WP2) and the Storage (WP1) to provide services for all use cases for data retrieval, exposing a REST interface to the clients. The data model for pure conditions data description is delivered by this package, with the appropriate tools for folder creation. The system uses catalogue information to impose authentication and authorisation. The data caching solutions are explored in this middle-tier level using modern web caching technologies.
5.4. WP4 - Conditions storage monitoring
The storage monitoring component of the project interacts with the catalogue (WP2) (collecting structural metrics including data volume, row counts, etc.) and the storage WP1 (interval coverage and gaps in conditions of specific tags or across global tags). This is critical to understand conditions volume growth and conditions completeness, which represents an important integrity check.

5.5. WP5 - Conditions server monitoring
Monitoring of the server running the Core Services is needed to understand throughput (for optimisation of server configuration) and identify and address bottlenecks due to slow queries or unexpected job workflows.

5.6. WP6 - Transactional storage and monitoring
Transactional data are collected by the Core Services servers (read-access throughput and specific conditions accessed) and from jobs (conditions retrieved and actually used by jobs). This type of data might best be handled using non-relational storage or Big Data related technologies. This component would also include interfaces to mine information from this collected data.

6. Summary
We have presented here a proposal for investigating a new Conditions Database architecture for the ATLAS and CMS experiments which could replace their present infrastructures on the timescale of LHC Run 3 (2020). This new architecture takes into account the experience accumulated in both experiments and the evaluation of use cases which leads us to a much simplified data model with respect to Run 1. We believe we can deliver the necessary functionality in a system which is much easier to translate for use in other experiments, keeping the best aspects of present architectures but simplifying the client aspect and the multi-language support by means of standard WEB technologies based on the REST architecture.

References
[1] The ATLAS Collaboration, “The ATLAS Experiment at the CERN Large Hadron Collider”. 2008 J. Inst. 3 S08003 ISSN 1748-0221
[2] The CMS Collaboration, “The CMS experiment at CERN LHC”. 2008 J. Inst. 3 S08004 ISSN 1748-0221
[3] Oracle URL http://www.oracle.com/index.html
[4] Valassi A, Basset R, Clemencic M, Pucciani G, Schmidt S A and Wache M 2008 IEEE Nuclear Science Symposium Conference Record, 2008. NSS ’08 pp 3021–3028
[5] Boehler M, Borodin M, Formica A, Gallas E J and Radescu V 2015 “Evolution of ATLAS conditions data and its management for LHC run-2”. Proceedings of the CHEP conference J. Phys.: Conf. Ser.
[6] Govi G, Di Guida S, Pfeiffer A and Ojeda Sandonis M 2015 “The CMS Condition Database system”. Proceedings of the CHEP conference J. Phys.: Conf. Ser.
[7] Dykstra D 2011 J. Phys.: Conf. Ser. 331 042008 ISSN 1742-6596 URL http://iopscience.iop.org/1742-6596/331/4/042008
[8] Apache Tomcat URL http://tomcat.apache.org/
[9] Gallais E J, Albrand S, Borodin M and Formica A 2014 J. Phys.: Conf. Ser. 513 042020 ISSN 1742-6596 URL http://iopscience.iop.org/1742-6596/513/4/042020
[10] Spring URL http://spring.io/
[11] The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 317 URL https://www.jcp.org/en/jsr/detail?id=317
[12] Hibernate URL http://hibernate.org/
[13] Apache Maven URL http://maven.apache.org/
[14] WildFly URL http://wildfly.org/
[15] Jetty URL http://eclipse.org/jetty/
[16] PycURL URL http://pycurl.sourceforge.net/