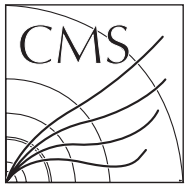
Available on CMS information server

CMS CR -2009/106



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

14 May 2009 (v2, 15 May 2009)

CMS Offline Conditions Framework and Services

G.Govi, V.Innocente,Z.Xie

Abstract

Non-event data describing detector conditions change with time and come from different data sources. They are accessible by physicists within the offline event-processing applications for precise calibration of reconstructed data as well as for data-quality control purposes. Over the past years CMS has developed and deployed a software system managing such data. Object-relational mapping and the relational abstraction layer of the LHC persistency framework are the foundation; the offline conditions framework updates and delivers C++ data objects according to their validity. A high-level tag versioning system allows production managers to organize data in hierarchical view. A scripting API in python, command-line tools and a web service serve physicists in daily work. A mini-framework is available for handling data coming from external sources. Efficient data distribution over the world-wide network is guaranteed by a system of hierarchical web caches. The system has been tested and used in all major productions, test-beams and cosmic runs.

Presented at *International Conference on Computing in High Energy and Nuclear Physics,21-27 March 2009,Prague,Czech Republic,15/05/2009*

CMS Offline Conditions Framework and Services

G. Govi
Department of Physics, Northeastern University, Boston, MA. 02115, USA

V. Innocente
European Laboratory for Particle Physics (CERN), Geneva, Switzerland

Z. XIE*
Department of Physics, Princeton University, Princeton, NJ. 08542, USA
E-mail: Zhen.Xie@cern.ch

(for the CMS collaboration)

Abstract. Non-event data describing detector conditions change with time and come from different data sources. They are accessible by physicists within the offline event-processing applications for precise calibration of reconstructed data as well as for data-quality control purposes. Over the past years CMS has developed and deployed a software system managing such data. Object-relational mapping and the relational abstraction layer of the LHC persistency framework are the foundation; the offline condition framework updates and delivers C++ data objects according to their validity. A high-level tag versioning system allows production managers to organize data in hierarchical view. A scripting API in python, command-line tools and a web service serve physicists in daily work. A mini-framework is available for handling data coming from external sources. Efficient data distribution over the worldwide network is guaranteed by a system of hierarchical web caches. The system has been tested and used in all major productions, test-beams and cosmic runs.

1. Introduction

This paper gives an overview on the offline software framework and services for conditions data management in the CMS experiment. The CMS model distinguishes two types of time variant non-event data: those taken during the operation of the experiment for monitoring purposes and those used in offline reconstruction and analysis or results from calibration and alignment algorithms. In this paper we focus on the data of the latter type. The data model and the base technology choices will be discussed. For the system to be most effective, different usages are identified and the core software components and services are designed to satisfy requirements from each different type of user activity. How such a design goal is achieved will be explained along with the description of the software components and services. Though not the focus of this paper, the computing and deployment model of the system will be briefly described.

2. Data model and base technology choices

Conditions data change with time and require frequent lookups by time in the data store. Time in this context is represented by either run number, universal timestamps or luminosity section id. A Relational Data Base Management System (RDBMS) is suitable for handling concurrent random access to a centralized storage. CMS online system uses RDBMS as non-event data store. For these reasons RDBMS is the choice for conditions data persistency at offline level as well. On the other hand, the Object-Oriented nature of the offline software requires the data to be delivered as objects. The relational storage component of the LHC persistency framework POOL [1] has not only the capability of mapping objects to relational tables, but also the required relational data abstraction through a database abstraction layer CORAL [2]. Therefore POOL and CORAL constitute the basic components for CMS offline handling of conditions data.

The bulk data, also called the payload, are C++/POOL objects and can be accessed via their Interval-Of-Validity(IOV). In the CMS model, the IOV sequence is always consecutive with no holes nor overlap in time. An IOV sequence is a C++/POOL object which contains the mapping of time to externalized POOL tokens of the payload objects of the same type. In the more recent performance enhanced design, an IOV object can contain also the summary of the payload it refers to. IOV can be versioned and the name of the version is called the IOV tag. IOV tags are the lowest level of access to the data. Knowing the tag, one arrives at the valid payload object for a given point in time by finding the corresponding time period in the IOV and following the POOL token as shown in the lower part of figure 1.

In organized large scale data production, production managers need to manage a collection of IOV tags and configure different jobs to access different subsets of them. Having identified such a user case, we build a tree structure on top of the basic IOV tags where each tree node is a data access point called the "global tag". Such tree structure is illustrated in the upper part of figure 1. The leaves of the trees are linked to the base level IOV tags described previously. The collection of all the IOV tags is called the "Tag Inventory".

There are many ways of modeling trees in RDBMS. The most common ones use either multiple self-joins or recursions to traverse the tree. To avoid performance penalties introduced by self-joins and recursions we have chosen the not-so-common "nested set" [3] model where tree nodes are represented by nested sets with number pairs. With this model, all the data are contained in one table. Queries are based on the numbering of the nodes. This model does not introduce multiple self-joins and there is no limit on the depth of the tree.

3. Core Software and Services

For the software to be effective and fit the genuine needs of users, the overall design of the system focuses differently on different types of user activities.

In CMS offline processing, users access the detector conditions through the generic EventSetup part of the EDM framework.[4]. EventSetup provides a snapshot of the detector conditions at an instant in time. PoolDBESSource is a specific EventSetup data provider which delivers the data objects to the EventSetup and guarantees the time validity of the delivery. PoolDBOutputService is a load-on-demand EDM service that writes out conditions data and associates IOV to the data. As the names of these modules imply, they are built on top of POOL and CORAL libraries. POOL is responsible for the mapping of objects to tables as well as the object input/output while CORAL is the generic RDBMS abstraction layer. These components together, as shown in figure 2, ensure the conditions data I/O is an integral part of the event processing and the user awareness of the difference in non-event and event data handling is minimal. No database APIs nor queries are exposed and thanks to the EventSetup design, users do not need to learn any conditions or IOV management API. The data dictionary and reflection mechanism are identical to that of the event data. PopCon is a mini-framework

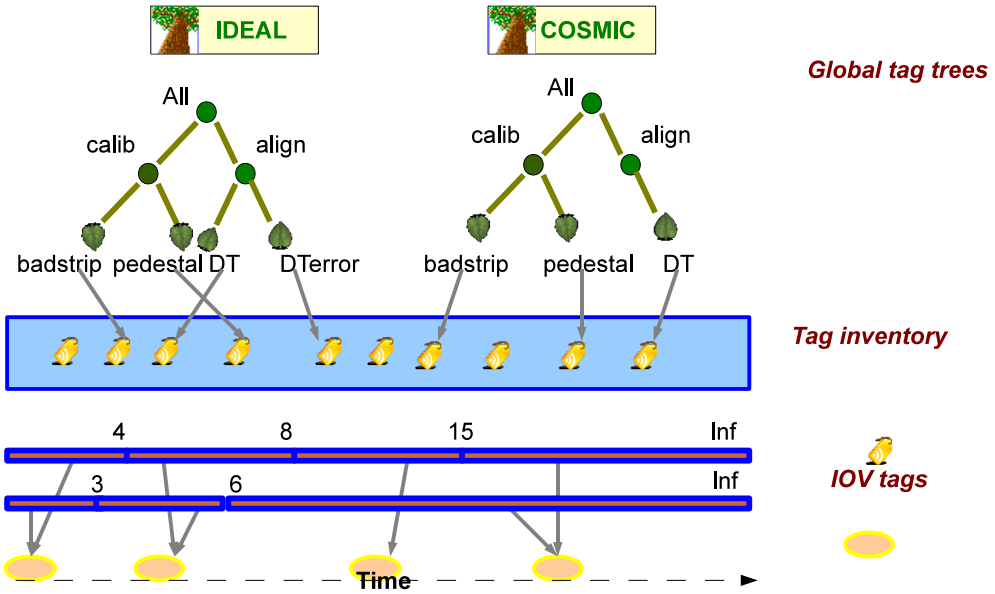


Figure 1. Conditions database data model

based on PoolDBOutputService. It manages data transformation from external to the POOL format and data transfer from external sources to the offline databases. And it has the integrated capability of data transfer logging and IOV consistency check.

Apart from the standard HEP event processing usage described above, we recognize the need of software utilities for data management users. Different from the event processing jobs, production managers do not deal with payload data, they organize production jobs via global tag management and data transfer between different database instances and different backend technologies. Since tools for the data management are independent of the event processing, Python is chosen as the base language because its simplicity and flexibility. All the global tag management software are written directly in Python while at lower level, database connection and transaction management and query abstractions are exposed to Python via PyCoral which is the Python binding layer of CORAL.

Finally, the third kind of use case is similar to data mining where physicists browse tags, payloads and summary data searching for patterns and anomalies. We satisfy the requirements from such usages by providing various services. From the conditions database web service, one can browse the tags and make plots of the payload and the summary data. Similar to data management tools, the web service is written in Python. IOV, payload and summary data inspection capabilities are exposed to Python via a C++/Python binding layer of the core components. Given the nomadic nature of this kind of user, the web service is packaged in the way that it can be deployed centrally as well as on standalone personal computers. We also provide a conditions ROOT browser supporting interactive IOV manipulation, data inspection and display from the ROOT prompt. Figure 3 shows one example ROOT plot of EcalPedestals value as a function of run number.

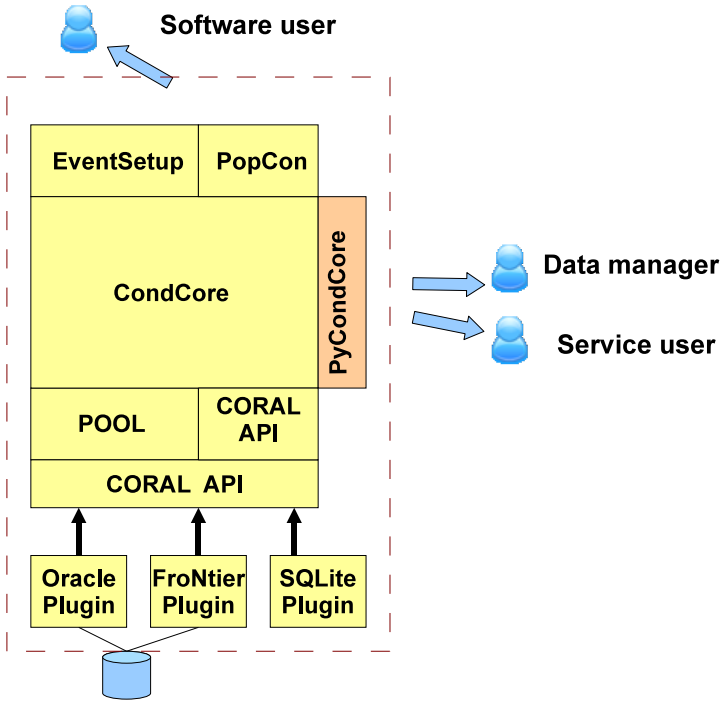


Figure 2. Software Components

4. Deployment Model

The computing model for the offline conditions data can be summarized as a distributed read-only access to centrally managed databases. Distributed access is achieved through FroNTier [5] which is a multi-tier web cache system deployed at CMS computing centers around the world.

There are two central Oracle databases at CERN for offline conditions data. The instance at the experiment site, named ORCON is the master instance connected to the restrictive online network. The main source of the ORCON data are those selected from the online database and those written directly from the High-Level-Trigger application. Oracle streaming makes data in the online master instantly available to the CMS general public users in the Oracle instance at the CERN computing center, named ORCOFF. Frontier caches are connected to both oracle instances ensuring read-only access in CMS computing centerers. Data are written to the online Oracle master (ORCON) through the PopCon mini-framework or imported from sqlite files in a controlled manner.

The system provides three levels of Oracle services: production, integration and development. Additionally, users can also use SQLite databases for testing and development. Thanks to the CORAL abstraction layer, importing/exporting data across different database backends is fully supported.

5. Status and Plans

The core component development started in 2006. A prototype system was successfully tested for the first time in Computing-Software-Analysis challenge 2006. While evolving, the system has been used routinely in all the global runs, cosmic run and Monte Carlo productions subsequently. By now the core components have reached complete functionalities. In order to improve

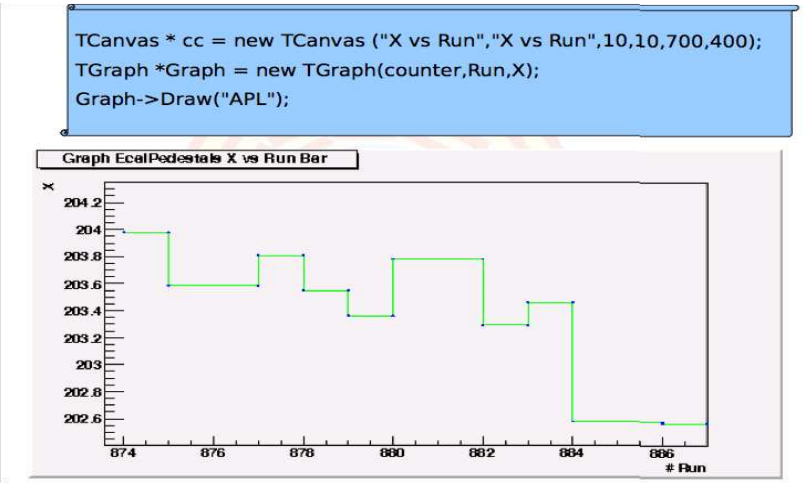


Figure 3. ROOT plot of Pedestals value versus run number

performance and to increase the speed and improve the memory usage for interactive payload data inspection, some enhancements in the data model are introduced in recent releases together with corresponding modifications and new data types in POOL. The service development started in 2007. In production as standard cms web service since 2008 and more functionality is being added.

6. Conclusions

CMS has developed a software system to manage conditions data as objects stored in a relational database taking advantage of the object-relational mapping capability of POOL and the RDBMS abstraction of CORAL. The core component plugs on one side into the CMS offline framework and to the POOL and CORAL persistency layer on the other, yet to the end users the differences in event and non-event handling seem minimal. High level tag collections are organized as trees with a light-weight relational design. A full set of python-based tools are provided to the production managers for the global tag management and data transfer. Services such as the web service and the conditions data ROOT browser facilitate interaction with data by physicists.

References

[1] D. Duellmann 2003 *The LCG POOL Project - General Overview and Project Structure*, Proc. Conf. for Computing in High Energy and Nuclear Physics (San Diego).

[2] I. Papadoupolos *et al.* 2006 *CORAL a software system for vendor-neutral access to relational databases*, Proc. Conf. for Computing in High Energy and Nuclear Physics (Mumbai) .

[3] J. Celko 2004 *Joe Celko's Trees And Hierarchies In SQL for Smarties* (San Francisco: Elsevier) .

[4] C.D. Jones 2006 *Access to non-event data for CMS*, Proc. Conf. for Computing in High Energy and Nuclear Physics(Mumbai).

[5] L. Leuking *et al.* 2004 *FroNTier: High Performance Database Access Using Standard Web Components in a Scalable Multi-tier Architecture*, Proc. Conf. for Computing in High Energy and Nuclear Physics (Interlaken).