The CMS ECAL Non-Event Data Handling

Thomas Punz on behalf of the CMS Collaboration

Institute of Particle Physics, ETH Zurich, 8093 Zurich, Switzerland

Abstract. The Electromagnetic Calorimeter of the CMS experiment at the LHC includes about 76000 lead tungstate ($PbWO_4$) scintillating crystals. The detector properties must be continuously monitored in order to ensure the extreme stability and precision required. This leads to a very large volume of non-event data to be accessed continuously by shifters, experts, automatic monitoring tasks, detector configuration for trigger and data acquisition systems and offline data reconstruction programs. This paper describes the measurements taken by the Detector Control System, the calibrations, the data handling strategy and the workflow as well as the architecture of the configuration and conditions databases and the web interface to access the data. An overview is given on the experience with detector commissioning, which have allowed a system test in realistic run conditions.

Keywords: CERN, CMS, ECAL, Database
PACS: 01.30.Cc

INTRODUCTION

The Compact Muon Solenoid (CMS) [1] is one of two general purpose experiments at the Large Hadron Collider (LHC). It consists of different layers, each a dedicated subdetector, specialized in detecting and measuring a given class of particles. The layers are organized in the following way: Starting from its center the pixel and silicon trackers, the electromagnetic calorimeter (ECAL) and the hadronic calorimeter are placed inside a superconducting solenoid, which provides a uniform magnetic field of 3.8 T parallel to the beam axis. This is followed by four muon stations integrated into the return yoke of the magnet. Due to different requirements for the various subdetectors each subdetector group developed its own database schema and services for the subdetector control and monitoring. In this paper the requirements on the ECAL database will be discussed. The ECAL [2] is composed of 75848 lead-tungstate ($PbWO_4$) scintillating crystals. The crystals are tapered and arranged to form an approximate cylindrical barrel closed by two end-caps, whose axis coincides with the beam axis. A Preshower detector, based on two layers of silicon sensors, is placed in front of the crystals in the end-caps. This paper concentrates on the database aspects of the crystal parts of the ECAL. The light produced by the crystals is read out by avalanche photodiodes (APDs) in the barrel and vacuum phototriodes (VPTs) in the endcaps. The photodetector bias is provided by 1276 high-voltage lines, while the power to the front-end boards is fed by 844 individual channels. Moreover, there are about 7300 temperature sensors distributed among both the barrel and end-caps, which are regularly monitored. This paper is meant to give an overview on the complex topic of the ECAL data handling. A more detailed description can be found in reference [3].

AIM OF THE DATABASE PROJECT

The aim of the ECAL database project is to store the data obtained from quality control tests during the ECAL's construction, the detector conditions measured during operation and the configuration data needed to configure the hardware and software. The construction database contains information like the position of single objects in the final setup as well as the association between crystals, photodetectors and read-out channels. The conditions database contains information about the condition of the detector, e.g. currents, voltages, temperatures and humidities. There are two run modes to operate the detector: local and global runs. Local runs are those runs involving only the ECAL (or part of it). These runs are specially dedicated to take data relevant for ECAL control and monitoring. In the global run the ECAL is operated together with other subdetectors from CMS.

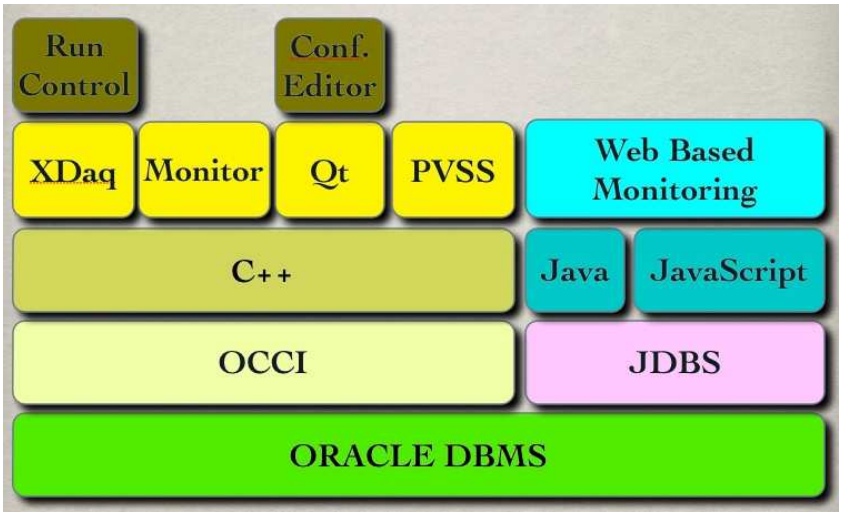

FIGURE 1. The software abstraction layer between the Oracle database and the frontend applications.

Software Architecture

The ORACLE database 10g Enterprise Edition is used as the main Database Management System (DBMS). There are two database clusters for CMS, the online and the offline. The online database is located at the CMS experiment to store all the data for the construction, configuration and conditions databases, which are necessary to operate the CMS detector. A fraction of the data are relevant for physics event reconstruction. This data are automatically copied to the offline database located in the CERN IT DEPARTMENT via a dedicated link. This copy process is performed by using the built-in streaming functionalities of the ORACLE DBMS. We will focus now on the online database, which can be accessed mainly via three mechanisms: C++ compiled programs, PVSS applications, and Java servlets. There are three main use cases for the online database: detector control and configuration, execution of local runs and browsing of its content. The detector control and configuration is provided by a graphical user interface (GUI), which can be used by the user to load, modify and store configurations as well as apply them to the detector hardware. This Detector Control System (DCS) is handled by a PVSS project [4]: a commercial SCADA (Supervisory Control And Data Acquisition) tool for automation produced by the ETM company, which is currently owned by Siemens. The DCS system is using the *PVSS RDB library* (developed at CERN) to communicate and operate on the DBMS. The data obtained by the DCS are written to the database each time a variation from previous values exceeds a given threshold, the dead band. This dead band is adjusted to the needs of the values to be archived. In order to check that the whole monitoring system itself is working the data are, in addition to the dead band archiving, written every 30 minutes to the database. The run control interface adopts the Qt graphic library [5] for the rendering of the interface and adopts C++ to connect to the database. For a detailed view of the hierarchy of all abstraction layers between the DBMS and the frontend applications we refer to figure 1.

Speed is one of the top requirements for the local run interface of ECAL. This can be achieved with compiled C++ programs. The communication between C++ and the DBMS is handled by the Oracle C++ Call Interface (OCCI) interface. The C++ programs read parameters from the database and acquire data from ECAL, which are then analyzed and the results stored back into the database. Data related to the detector configuration (DAQ and power status) as well as environmental data can have a simple, well defined and immutable structure, which can be handled by simple (key, value) pairs tables. Monitoring tasks can produce a very heterogeneous and complex set of data and its structure may change over time, due to a better detector knowledge or new monitoring tasks. To manage this one needs a complex table structure, which is able to adapt to later changes. Along with the table structure one needs to think about the indexing and its impact on the overall query performance of the database. More information is given in a dedicated section about performance later in this note.

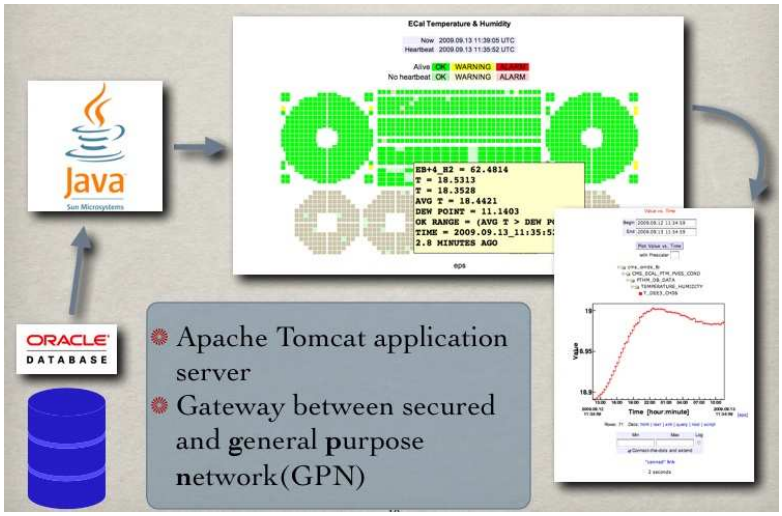

FIGURE 2. Schema of the Web Based Monitoring (WBM) architecture.

Web Based Monitoring

The browsing of the database content can be achieved by using a standard web browser like Mozilla Firefox™. The interface between the web server and the database is provided via JDBC calls by Java servlets. The data returned to the user is displayed on the website in a textual and graphical way. According to the data type, plots can be produced as graphs, histograms or 2-D maps. The Web Based Monitoring (WBM) is based on the Apache Tomcat application server [6], developed by the Apache Software Foundation, that executes Java servlets and renders Java Server Page pages. These servlets interact with the database via Java Database Connectivity (JDBC) calls and render web pages on the client side (see Figure 2). For ECAL, WBM allows users to search for runs with a simple form. Runs can be searched by means of their number or by date. Users can specify the run type or the detector components to be included in the DAQ in the resulting run list. Selecting a run, a summary page shows all the information stored in the monitoring task tables, as well as in the configuration tables, in a suitable way. Software is organized in such a way that the main page is common to all views.

The data needed to build the right plot are retrieved from the database using always the same tool that obtains the necessary information from their description, avoiding duplication of software and significantly reducing time needed for maintenance. In fact, the addition of a new task or the introduction of new page layouts does not change the way in which plots are produced. Plots are produced on the fly, but a caching mechanism avoids unnecessary queries if data have already been extracted within the last few hours. Besides each plot a link is provided with the data used to produce it, in text form as well as a ROOT [7] Tree. The WBM also provides a tool to make history plots of any value in the data tables. Also shown is the last obtained environmental data, which is displayed as color coded maps, so that shifters can easily identify channels with a bad behavior looking for red spots in the map or in pale color. A pale color indicates that the corresponding value is not current. Details about each channel are always available as properly formatted text as well.

PERFORMANCE

During the design of the schema, performance and scale tests were done to ensure that the application would satisfy the long-term requirements of the ECAL. For each type of data, growth rates of the data are constant and well-known, enabling us to simulate the growth of the database. It was assumed that the database would have to store data for one year's worth of CMS running before older data would be archived. Knowing the limits to the database size allowed us to focus our performance testing. Several different schemas were tested to learn how read and write performance would scale as a function of number of channels in the database. These schemas differed in how the primary key value timestamps were stored and indexed, and how the payload data would be stored. Options for primary key

organization included inline with the data, or in a separate table with an integer ID (the winning method). Options for payload storage included in table columns with channels- in-rows, all channels together in CLOB or BLOB columns, and channels-in-columns format. Two access patterns were tested: selecting all channels for a type of data given a timestamp t, and selecting a single channel at time t. The matrix of possibilities above was exercised and guided the design of the current schema. These performance benchmarks were absolutely crucial to developing a schema that would work for our application. As an example, the first iteration of our schema for storing the pedestals was found to take 20 seconds to query all ECAL barrel channels (61200) at time t with only 10 days worth of ECAL running stored. This result scales linearly with time, leading to the conclusion that if we had one year of data the query would take 12.2 minutes to execute. It may seem surprising that such a schema could even be considered, but at the time we were only working with small portions of the detector (3600 channels) and small databases, so the scaling difficulty was not immediately apparent. With the results of the performance testing, we arrived at a schema which could do the same query 55 times faster when executed on a database with 1,000 primary key values, and more importantly the scaling was constant with more values. Calculations based on the scaling factors obtained in testing showed that the same query on a one-year database could be done in 0.9 seconds, which is nearly 800 times faster than what we started with.

CURRENT EXPERIENCE

The current implementation of services for database access is very satisfactory, both for users and developers. Due to the pattern employed in the definition of the tables developers can develop generic applications that automatically retrieve all the information needed to provide the user data in the proper format. The services were exercised during 2008 and are currently under test while the CMS experiment has been undergoing commissioning. The frequency of updates, readings and the size of the transactions are almost independent on the type of activity, so we can reasonably extrapolate the behavior of the system at the time when there will be p-p collisions from LHC. From summer to autumn 2008 the database size grew by 3.3 GB/month. Taking into account that the duty cycle during this time was about 1/3 of the one foreseen during LHC running, we can estimate that the database size will grow by 10 GB/month, i.e. 66 GB/year, assuming 200 days of operation of LHC per year.

CONCLUSION

The database services for the control and monitoring of the electromagnetic calorimeter of the CMS experiment are already well established and ready for p-p collisions. Users interact with the database by means of three classes of applications. The estimated rate of growth of the database, obtained measuring the size of the database after real data taking during commissioning, is comfortable, being less than 100 GB/year. The query performance was improved by a factor of nearly 800, compared to the beginning, by optimizing the database schema.

ACKNOWLEDGMENTS

The author would like to thank the CMS ECAL Collaboration for its help.

REFERENCES

1. S. Chatrchyan, The CMS experiment at the CERN LHC, Tech. rep., JINST 3 S08004 (2008).
2. G. Bayatian, CMS ECAL Technical Design Report, Tech. rep., CERN/LHCC 97-33 (2008).
3. G. Organtini, The CMS ECAL Database Services for Detector Control and Monitoring , Tech. rep., CERN (2009).
4. `http://itcobe.web.cern.ch/itcobe/Services/Pvss` (2009).
5. `http://www.qtsoftware.com/products` (2009).
6. `http://tomcat.apache.org` (2009).
7. `http://root.cern.ch` (2009).