Web Application for Detailed Real-time Database Transaction Monitoring for CMS Condition Data

Michele de Gruttola[*], Salvatore Di Guida[†], Vincenzo Innocente[†] and Antonio Pierro[**]

[*]*CERN, CH-1211 Genève 23, Switzerland, and INFN Sezione di Napoli e Università degli Studi Di Napoli Federico II, Complesso Universitario Monte Sant'Angelo, Edificio 6 - Via Cinthia, I-80126 Napoli, Italy*
[†]*CERN, CH-1211 Genève 23, Switzerland*
[**]*INFN-Bari - Bari University, Via Orabona 4, Bari 70126, Italy*

Abstract. In the upcoming LHC era, database have become an essential part for the experiments collecting data from LHC, in order to safely store, and consistently retrieve, a wide amount of data, which are produced by different sources. In the CMS experiment at CERN, all this information is stored in ORACLE databases, allocated in several servers, both inside and outside the CERN network. In this scenario, the task of monitoring different databases is a crucial database administration issue, since different information may be required depending on different users' tasks such as data transfer, inspection, planning and security issues. We present here a web application based on Python web framework and Python modules for data mining purposes. To customize the GUI we record traces of user interactions that are used to build use case models.

In addition the application detects errors in database transactions (for example identify any mistake made by user, application failure, unexpected network shutdown or Structured Query Language (SQL) statement error) and provides warning messages from the different users' perspectives.

Finally, in order to fullfill the requirements of the CMS experiment community, and to meet the new development in many Web client tools, our application was further developed, and new features were deployed.

Keywords: CMS, CMSSW, ORACLE, POOL, CORAL, Python, PopCon, Database, Transaction Monitoring, Web Application
PACS: 89.20.Hh

INTRODUCTION

In the CMS experiment[1][2], heterogeneous resources and data are put together in different Oracle-based databases, and made available to users for a variety of different applications, such as the calibration of the various subdetector components and the reconstruction of all physical quantities.

In this complex environment, it is absolutely necessary to monitor Database Resources and every application which performs database transactions, in order to detect faulty situations, contract violations and user-defined events.

PopCon monitoring (Populator of Condition Objects monitoring) is an open source web based service implemented in Python, and designed for a heterogeneous database server, that performs data transfers to provide both fabric and application monitoring. It promotes the adoption of the Standard web technologies, service interfaces, protocols and data models.

POPCON TOOL

A proper reconstruction of physical quantities needs data which do not come from collision events of the *CMS experiment*: these "non event" data, therefore, are stored in ORACLE Databases.

PopCon[3] (Populator of Condition Objects tool) is an application package fully integrated in the overall CMS framework[4][5] intended to store, transfer and retrieve data using Oracle Database.

Even without LHC[7] beams, expected for the autumn of this year, this mechanism was intensively and successfully used during 2008 and 2009 tests with cosmic rays and now it is under further development. Up to now, 0.5 TB of data per year have been stored into the *CMS Condition Databases*.

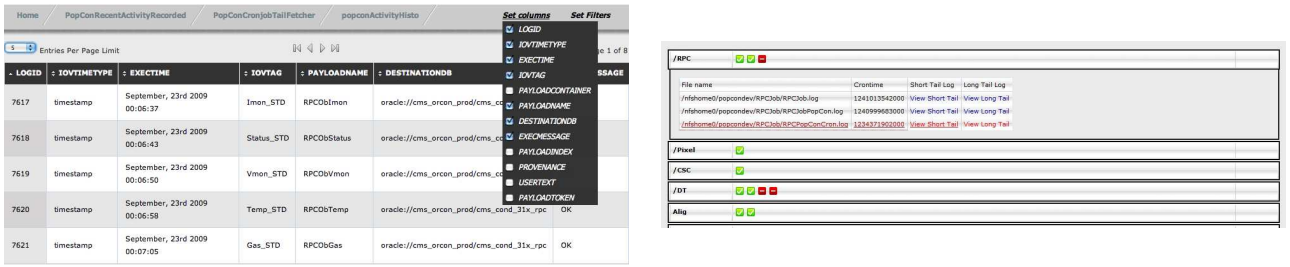

FIGURE 1. The PopCon web interface represents information about database transactions in different types: both charts and tables. A user can easily add or remove columns by clicking the checkbox and also columns can be sorted. Information could be grouped according to different filters. In the error GUI, different colours help to identify, quickly, the seriousness of the problem.

POPCON MONITORING ARCHITECTURE AND FEATURES

PopCon monitoring is structured in five main components:

- the **PopCon API DB Interface** retrieves the entities monitored by *PopCon tool*;
- the **PopCon user Interaction Recorder** is a collection that retains an interaction history by each user.
- the **PopCon data-mining** extracts patterns from data, entities monitored by *PopCon tool* and the history of recorded user interactions, hence transforming them into information such as warnings, errors or alarms according to use case models.
- the **PopCon info collector** aggregates the information produced by the different database transactions and the history of recorded user interactions, and encodes them in JSON[1] format.
- the **PopCon Web Interface** displays the information about the database transactions from the different user perspectives, organizing data in tables and/or charts (see Figure 1, where data are collected in a table).

The *PopCon API DB Interface* is a Python script that gives access to the PopCon account on the Oracle Database. This component uses the `cx_Oracle`[2] python module to connect to Oracle DBs and call various PL/SQL package methods.

PopCon user Interaction Recorder creates and makes accessible the records of activities made by each user. Collected records are used to implement and improve a web interface, which can be designed for information browsing for different users in different ways. This component interacts with, and receives information from the *PopCon Web Interface*.

Through the use of sophisticated algorithms, *PopCon data-mining* can extract information from logs of database transactions (operator, data source, date and time, metadata) and the *PopCon User Interaction Recorder* (sequence of actions to get to the right contents, average time on each page to compute the attention applied by the visitor) finding existing patterns in data.

This algorithm iterates two main steps.

The first step, called *harvesting user interaction statistics*, records the following list of measurements subdivided into two categories:

- tracks of the browsed page, like most requested pages, least requested pages, most accessed directory, average time on the page, average time on web site, ordered sequence of visited pages, new versus returning visitors (by means of cookies), number of views per each page;
- tracks of user activity at the page level:
  - Changing attributes of graphical elements (e.g. changing charts representation from line chart to pie chart or histogram chart, sorting and filtering data in a table);
  - Removing/adding object elements (e.g. remove/add columns to the table).

[1] JSON (JavaScript Object Notation) is a lightweight data-interchange format

[2] `cx_Oracle` is a Python extension module that allows access to Oracle databases. `http://cx-oracle.sourceforge.net/`.

The second step, called *grouping attributes of user interaction with significant correlation*, gathers in different subgroups the *tracking user activity* and *tracking browsed page* that have similar attributes, like most accessed directory and common graphics elements, in order to create mutually exclusive collections of user interactions sharing similar attributes.

To reach this goal, we use an algorithm handling mathematical and statistical calculations, such as probability and standard deviation, to uncover trends and correlations among the attributes of the user interaction.

For example, after scanning the history of recorded user interactions, an association rule "*the user that visits page one also visits page two and chooses to see histogram reports (90%)*" states that nine out of ten users that visit the page one also visit the page two and prefer to see the bar chart. We can buid use case models, based on these statistics, in order to reflect the requirements and the needs of each user. As a result, the user, classified under this use case, will take advantage to see a web interface based on his perspective, helping him to find and manage the information he needs more quickly.

PopCon is integrated within the `CMSSW`[8] framework, which depends on different tools like `POOL`[3][9][10][11] and `CORAL`[4] and on database software like `ORACLE` and `SQLite`. This application can be used in two different ways:

- since it is integrated in the framework, users can write python scripts which are executed by the framework executable `cmsRun`.
- the framework itself provides an application which, using *PopCon* libraries, allows the exportation of data into the offline database.

These applications are responsible for maintaining and handling operations which are related to database transactions. In this scenario, it is very difficult to catch all error messages coming from different heterogeneous resources. Therefore, we follow this strategy: every application provides an error output consisting of three components: the name of application, the error code, that is unique for each tool, and the description of the error itself. So, PopCon developers can clearly understand what is wrong with their tool, while the end-user is able to check if the data exportation (database transaction) they want to perform was successful or not.

This error metric, for each tool, is provided by the framework developers in `XML` format, in order to make it independent from the message sent to stdout and/or stderr.

Besides describing what the error is and how it occurred, most error messages provide advice about how to correct the error.

To help both users and developers to classify correctly the observed damage, the error messages are defined by the level of issue with a different colour. These levels are:

- Fatal. The program cannot continue (red colour).
- Major (Error). The program has suffered a loss of functionality, but it continues to run (orange colour).
- Minor (Warn). There is a malfunction that is a nuisance, but it does not interfere with the program's operation (deep green colour).
- Informational. Not an error, this is related information that may be useful for troubleshooting (green colour).

In Figure 1 you can see a snapshot of the error interface.

As further example, we describe another kind of error not depending on the particular application, but on Hardware/Software/Network problems. To discover this kind of error, we perform a time series analysis on database transactions associated with the discovery and use of patterns such as periodicity. Since dates and times of the database transactions are recorded along with the user's information, the data can be easily aggregated into various forms equally spaced in time. For example, for a specific account the granularity of database transactions could be hourly and for other account could be daily. This information allows to discover two main kinds of alarm:

- Scanning the entities monitored by *PopCon* (logs of database transactions), the association rule "*during a long period, a specific user performs a database transaction at regular time intervals*" states that, probably, if these regular intervals suddenly change without a monitored interaction by an administrator, and, for particular cases, by the user, there can be network connectivity problems, or machine failures on the network. In details, if the system finds an exception to this pattern in data, it triggers an action to inform a user about possible problems

[3] `POOL` is the persistency framework for object storage common to the LHC experiments.
[4] `CORAL` is a vendor independent API for relational database access, data and schema handling.

by email. Besides, the web user interface provides red/orange/green alarms, according to the seriousness of the problem, so that this exception is immediately visible by the user.

- Taking the size of data together with the periodicity of database data transactions we can forecast the rate at which disk capacity is being filled in order to prevent a disk becoming full, alerting the database manager and the administrators of the machines dedicated to the data exportation some days in advance.

The errors produced by several applications of *PopCon tool* inside the CMS collaboration, stored both in log files and in tables of a dedicated database account, were the starting point for the study of feasibility of an automatic error detection in a heterogeneous software environment[12].

The *PopCon Info Collector* retrieves data from the *PopCon API DB* and the *PopCon User Interaction Recorder*. This component interacts with *PopCon Data-mining* to find existing patterns in data previously taken, and, finally, encodes them in JSON format, providing the result to the *PopCon Web Interface*.

The system has a front-end Apache server and back-end application servers. The *PopCon Web Interface* is an application created with a Python-based framework using `Cheetah`[5] template engine to structure the web site. The *PopCon Web Interface* is built on the `CherryPy`[6] framework application server, which runs behind Apache providing security module to automatically show a role-optimized view of the system and its controls. A set of reusable components, known as "widgets", are being made available. These are usually built using the (`jQuery`) libraries and are written in CSS and JavaScript. Where possible, these are reused in order to provide identical functionality across direct components, so that a user feels comfortable with a standard style sheet for all web tools. The services run on a fairly standard configuration: a pair of Apache servers working as a load balanced proxy in front of many application servers. The front end servers are accessible to the outside world, while the back end machines are firewalled off from remote access[13][14]. With this infrastructure, we can minimize problems related with security issues: in particular, each user is unable to handle database objects. Thanks to `AJAJ`[7], we can provide real-time feedback to our users exploiting server-side validation scripts, and eliminate the need for redundant page reload that is necessary when the pages change. In fact, this component allows to send requests asynchronously and load data from the server. The *PopCon Web Interface* uses a programming model with display and events. These events are user actions: they call functions associated to elements of the web page and then actions are recorded by the *PopCon user Interaction Recorder*. The contents of pages coming from different parts of the application are extracted from JSON files provided by the *PopCon Info Collector*.

POPCON MONITORING FROM THE DIFFERENT USERS' PERSPECTIVES

The design of the presentation of the data collected by *PopCon monitoring* is based on the requirements given by different types of users, each of them having to do with a different abstraction level of a Database administration issue: the ORACLE Database Administrator level, the central *CMS detector* level, the *CMS sub-detector* level and the End-User level.

- The ORACLE Database Administrator may wish to face up to databases security issues for which he is responsible. Typical example that can be detected: people on the inside (using *PopCon tool*) and outside (using *PopCon Web Interface*) network who can access and what these users do; programs accessing a database concurrently, in order to avoid further multiple accesses to the same account; if all such processing leave the database or data store in a consistent state; illegal entries by hackers; malicious activities such as stealing content of databases; data corruption resulting from power loss or surge; physical damage to equipment.
- The central CMS detector manager and the *PopCon tool* developer may require the possibility of analysing the behaviour of their applications for each CMS sub-detector.
- The sub-detector CMS manager may require the possibility to analyse the behaviour of his transactions on his own sub-detector database account.
- The End-User may require the possibility to analyse the behaviour of his own personal transaction such as size and rate/duration of the transactions, or detect fault situations related to insufficient password strength or inappropriate

[5] `Cheetah` - The Python-Powered Template Engine. `http://www.cheetahtemplate.org/`.
[6] `CherryPy`: a pythonic, object-oriented HTTP framework. `http://www.cherrypy.org/`.
[7] `AJAJ`: Asynchronous Javascript and JSON

access to critical data such as metadata.

To summarize, *PopCon monitoring* automatically detects the cookies installed in each user's browser and this information is used to match the user with a role (Oracle Database Administrator, *PopCon tool* developer, CMS sub-detector manager, End-User), in order to provide a customized report that allows each user to have a customized printout of information depending on his needs.

The use of data mining techniques to extract patterns from logs of database transactions (operator, date and time) and the history of recorded user interactions has some general advantages. The storage of these patterns will help the user to read and understand quickly the current situation without going through several pages and use the search fields.

CONCLUSIONS

Although the number of samples analysed here is limited, the applied approach demonstrates that our application is dynamic since it can work and parse the different types of data for which date is a primary key.

Date can be written in many different ways because of flexible Python functions which work with date and parses it.

Another important feature of this application is that the *PopCon User Interaction Recorder* could be used in combination with *PopCon data-mining* to provide almost the same functionality in general for any application. It's indeed a flexible part which helps to collect and interpret information about user activities, and the actions made while he handles the application. This information can also be used to provide new and comfortable features for users, as we are using it to adapt the *PopCon Web Interface* to the user's needs.

REFERENCES

1. The CMS Collaboration, CMS Physics TDR, Volume I: Detector Performance and Software. *Technical Report CERN-LHCC-2006-001; CMSTDR-008-1*, CERN, Geneva, 2006.
2. The CMS Collaboration, CMS Physics Technical Design Report, Volume II: Physics Performance. *J. Phys. G*, 34(6):995–1579, 2007.
3. de Gruttola, M., Di Guida, S. and Innocente, V., PopCon (Populator of Condition Objects): First Experience in Operating the Population of the "Condition Database" for the CMS Experiment. *International Conference on Computing in High-Energy Physics (CHEP 2009), Distributed Processing and Analysis track*, 23–27 March 2009, Prague, Czech Republic.
4. The CMS Collaboration, CMS Computing TDR, *CERN-LHCC-2005-023, 20 June 2005*; `http://cdsweb.cern.ch/record/838359`.
5. Jones, C.D. et al., Analysis Environments for CMS, *J. Phys.: Conf. 2008 Ser.* 119 032027.
6. Brun, R. and Rademakers, F., ROOT - An Object Oriented Data Analysis Framework, *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A* 389 (1997) 81–86. See also `http://root.cern.ch/`.
7. The LHC Project, LHC Design Report, Volume I: the LHC Main Ring. *Technical Report CERN-2004-003-V-1*, CERN, Geneva, 2004.
8. Jones, C.D. et al., The New CMS Event Data Model and Framework, *Proceedings of the International Conference on Computing in High-Energy Physics (CHEP 2006)*, 13–17 February 2006, T.I.F.R. Mumbai, India.
9. Duellmann D. et al., THE LCG POOL Project - General Overview and Project Structure, *Proceedings of the International Conference on Computing in High-Energy Physics (CHEP 2003), MOKT007*, 24–28 March 2003, La Jolla, California.
10. Xie, Z. et al., POOL Persistency Framework for the LHC: New Developments and CMS Applications, *Proceedings of "Frontier Science 2005: New Frontiers in Sub nuclear Physics"*, 12–17 September 2005, Milan, Italy.
11. Govi, G. et al., CMS Offline Conditions Framework and Services, *International Conference on Computing in High-Energy Physics Conference (CHEP 2009), Distributed Processing and Analysis track*, 23–27 March 2009, Prague, Czech Republic.
12. Di Guida, S., Innocente, V., Kondrat, N., Kuzborskij, I., and Pierro, A., Probing Methods for Automatic Error Resolution in a Heterogeneous Software Environment, *Seventh International Conference of Computational Methods in Science and Engineering (ICCMSE 2009)*, 29 September - 4 October 2009, Rhodes, Greece.
13. Metson, S. et al., CMS Offline Web Tools, *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2007)*, 2–7 September 2007, Victoria, British Columbia, Canada.
14. Dziedziniewicz, K. et al., CMS Condition Database Web Application Service. *International Conference on Computing in High-Energy Physics (CHEP 2009), Distributed Processing and Analysis track*, 23–27 March 2009, Prague, Czech Republic.