The Compact Muon Solenoid Experiment

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

Conference Report

13 November 2013

The Common Analysis Framework Project

Marco Mascheroni for the ATLAS and CMS collaborations

Abstract

ATLAS, CERN-IT, and CMS embarked on a project to develop a common system for analysis workflow management, resource provisioning and job scheduling in the distributed computing infrastructure based on elements of PanDA. After an extensive feasibility study and development of a proof-of-concept prototype, the project has now a basic infrastructure that can be used to support the analysis use case of both experiments with common services. In this paper we will discuss the state of the art of the current solution, giving an overview of all the components of the system.

Presented at CHEP2013 Computing in High Energy Physics 2013
Experience in CMS with the common analysis framework project

Marco Mascheroni¹, Daniele Spiga², Tommaso Boccali³, Daniele Bonacorsi⁴, Mattia Cinquilli², Domenico Giordano², Federica Fanzago⁵, Ian Fisk⁶, Maria Girone², Jose Hernandez⁷, Preslav Konstantinov⁸, Niccolò Magini², Valentina Mancinelli², Hassen Riahi⁹, Lola Saiz Santos¹⁰, Eric W Vaandering⁶

¹ Universita e INFN, Sezione di Milano-Bicocca
² CERN, European Organization for Nuclear Research, Switzerland
³ INFN Sezione di Pisa
⁴ Universita e INFN, Sezione di Bologna
⁵ Universita e INFN Sezione di Padova
⁶ Fermi National Accelerator Laboratory
⁷ CIEMAT, Madrid, Spain
⁸ INRNE, Sofia, Bulgaria
⁹ Universita e INFN, Sezione di Perugia
¹⁰ University of California San Diego

E-mail: marco.mascheroni@cern.ch, daniele.spiga@cern.ch

Abstract. ATLAS, CERN-IT, and CMS embarked on a project to develop a common system for analysis workflow management, resource provisioning and job scheduling in the distributed computing infrastructure based on elements of PanDA. After an extensive feasibility study and development of a proof-of-concept prototype, the project has now a basic infrastructure that can be used to support the analysis use case of both experiments with common services. In this paper we will discuss the state of the art of the current solution, giving an overview of all the components of the system.

1. Introduction

ATLAS (A Toroidal LHC Apparatus) and CMS (Compact Muon Solenoid) are the two biggest LHC multipurpose detector experiments at CERN. For their computing needs they make use of the resources and the infrastructure provided by the Worldwide LHC Computing Grid (WLCG) [1]. Traditionally, all the technical software needed for the data-analysis has been developed individually by the two experiments. However, since the effort put in development is steadily decreasing, sustainability is becoming a concern. The long shutdown represents an opportunity to rethink about some aspects of the computing model. Common solutions have proved to be a successful way to optimize the development effort, and reduce the support cost of a tool: several “multi-experiment” tools have been proposed and are widely used by the LHC community[?].

Even though experiments have similar analysis workflows, common solutions among high level submission tools have never been developed, and each experiment has its own workflow manager. ATLAS uses a different number of clients which connect to PanDA (ATLAS Production and
Distributed Analysis [2]), CMS uses CRAB (CMS Remote Analysis Builder [3]) in conjunction with GlideInWMS [4]. In March 2012, ATLAS, CMS, and the experiment support group at CERN started a feasibility study to explore the possibility to develop a common system based on PanDA to submit jobs to the WLCG computing resources.

The results of the feasibility study showed there were many common points between the workflow of the two experiments, and both solutions rely on a more experiment specific part, which does the task management, the data discovery, the splitting and the submission; and on a more general part (a job manager) which handles single jobs, their resubmission and killing, and the scheduling. The common proposal targeted the latter, as it is more general, has more common points, and less interactions with experiment specific services. The work on this project culminated in the realization of a proof-of-concept prototype in December 2012, which evolved in a more consolidated version that was installed in a CMS testbed. The testbed has been open to volunteers for testing.

In this paper we will discuss both the experiment specific components, which have been partly taken from CRAB3 [5] (Section 2.1), and the common components which come from the PanDA environment (Section 2.2). Section 3 is about the testbed CMS deployed in order to test the whole system. Comments about the CMS experience in this project will be given in the conclusions (Section 4).

## 2. Overall Architecture

The overall architecture of the system is shown in Figure 2. There is an experiment specific client which connects to a common job manager, which in turn sends pilots to the distributed infrastructure. The experiment specific components are essentially three: a lightweight client which is a python command line interface for the user, a REpresentational State Transfer interface (REST) [6] which validates the requests of the user and insert them in an Oracle database, and a task worker which takes the requests from the REST, generate the jobs and submit them to the job manager. The common job manager is made of a server which takes the jobs specifications from the task worker (the PanDA server), and puts them into an Oracle DB. Then the auto pilot factory (APF) sends pilots to the Grid based on the information it gets from the server. Pilots download the jobs and update the PanDA server with the information regarding the job state. Finally, a data management component (asyncstageout) moves the output files from the local storage to the remote one [7].

In the following two subsections we will first discuss the CMS experiment specific components, and then the common components.

### 2.1. CMS experiment specific client

The cms experiment specific software is the responsible of task management. It adopts a client-server model that has been demonstrated to improve the management of analysis workflows. In fact, the usage of a server simplifies the user’s life by handling most of the load on its side, and by automating most operations that do not need direct user interaction. Another advantage of the model is the optimization of the the resource usage, in particular exploiting at its best the Grid middleware from few central points (the servers) to improve the scalability of the whole system. The server handles the interactions with other services without really impacting the client side, but still providing an improved quality of service and of workflow execution, thanks to the fact that it is possible to significantly reduce the number of requests to external services by properly using HTTP protocol headers and caching mechanisms.

Figure 2 shows in a more detailed view the interactions between the experiment specific components in the system.

**REST:** The REST interface exposes three resources through HTTP. The first resource (workflow) is used by the client to manipulate the task, the second resource (filemetadada)
is used by the PanDA server to insert job metadata when a job finishes, and the third resource (\texttt{workflowdb}) is used by the TaskWorker component to update the status of the manipulated tasks. There are many advantages in using RESTful approaches. First, there is a clear separation between the data and the interface: currently we are using an Oracle database to store the tasks, but changing it behind the scenes without modifying the other component is easy with this approach. Secondly, from a security point of view users authentication is handled with the HTTPS protocol using X509 certificates. Then, a REST interface is also useful because it’s really easy to script against.

From an architectural perspective this component has been designed to support multiple job manager. Currently the only supported one is PanDA, but others as GlideInWMS can be added in the future.
**CRABClient:** The main commands of the CRABClient are `submit`, `resubmit`, `status`, `getout`, `getlog`, `report`. The `submit` command loads the user configuration, and sends it to the REST interface (packaged in a HTTPS PUT request). The other commands are similar in the sense that all the commands trigger actions by modifying the REST Task database. The client is a really light tool (just `pycurl` and `python2.6` are needed), which is an operational advantage as users will need to reinstall new versions of the client less frequently. A modular and pluggable architecture has been designed: adding new commands to the client or a new type of job is really simple, the developer just need to subclass the right python class.

**TaskWorker:** The TaskWorker component is responsible of fetching work from the REST interface, where all the task of the user are stored in the Oracle database, and perform works like `submission`, `kill`, `resubmit` to the PanDA server. Many instances of the TaskWorker can run in parallel on different machines, and each one can run multiple processes (Figure 3). Each process is responsible to execute one work on a task by instantiating the set of actions associated to it. For example, Figure 4 shows the actions associated to the work which makes a submission.

After executing the work, a call to the REST interface is made to update the task status. Like the REST interface, the TaskWorker component has also been designed to support multiple job managers.

### 2.2. Shared components

All the components described so far handle user’s tasks. As tasks are split into jobs by the TaskManager, they are submitted to the PanDA system. The Production and Distributed Analysis (PanDA) system has been developed to meet ATLAS production and analysis requirements for a data-driven workload management system capable of operating at LHC data processing scale. Its main component are shown in Figure 5.

**PanDA server:** The PanDA server is the main component of the system and manages all job information centrally thanks to an Oracle database. The server main activities are: to handle queues of jobs, and to handle job priority and scheduling.

The CMS installation of the PanDA server uses the exact same code as the one of ATLAS, and no branching has been done. The job state machine is the very same, and the Oracle database schema has been cloned from the ATLAS production one. Data management components have been plugged: in particular CMS developed its version of the AdderComponent, a component which executes a callback when a job finishes and communicate to the Asyncstageout component that it can start the transfers.

**Auto Pilot Factory:** The auto pilot factory component [8] queries all the queues of jobs
in the PanDA server, and, based on the information found there, it keeps a number of pilots running on the Grid sites. The more jobs need to run in a certain moment, the more pilots the APF component will send to the distributed infrastructure.

No particular effort has been put in the installation of the APF: just a small CMS patch to allow the usage of a pool of certificates for the pilot is needed. The configuration of the pilot factory has been done through AGIS\cite{9} which has been manually filled with CMS site information.

**Pilot framework:** The pilots running on the worker nodes ask for jobs the PanDA server. Once they get the jobs, they download the user code and they execute it. Updates about the status of the jobs are sent to the PanDA server, which it has all the information about the jobs, and it uses them to populate the PanDA monitor system.

The pilot framework required some development: the original pilot was modified to allow the execution of experiment specific code. The code is now pluggable, and two CMS plugins have been developed to allow the execution of the CMS workflows: the stageout plugin, and the transformation plugin (which is responsible of the user’s code execution). More information about the PanDA pilots can be found in \cite{10}.

### 3. The CMS testbed

A CMS testbed to verify that everything was working as expected has been installed. The testbed is smaller than the real ATLAS production environment since it has been designed to scale with the activity of dozens beta users, and it is composed of about ten machines.

The configuration of the computing elements in the pilot factory has been automatically generated from the AGIS\cite{9} information system. AGIS has been populated by hand, and the needed information have been taken from SiteDB\cite{11} and from the GlideInWMS XML configuration file. This was used as a temporary solution since site administrators in CMS do not use AGIS; a production environment will need a different workflow. 47 CMS sites have been added to the testbed.

PanDA Server has been installed following the instructions provided by ATLAS developers. Some customizations to the schedconfig cache scripts and to ATLAS Quattor configuration had to be done. Few configuration parameters not explicitly mentioned in the instructions need to be adapted. The PanDA database schemas for the testbed have been deployed on CMS integration
database (int9r). Both old and new PanDA monitor instances have been deployed. Resources and jobs are easily accessible through an HTTP interface and users can access the output of pilot jobs to investigate issues at each site.

4. Conclusions
Driven by the need of a more sustainable computing model, CMS, ATLAS, and CERN-IT started evaluating the possibility of building a common solution for the analysis use case based on PanDA. After doing a feasibility study, and after building a proof-of-concept prototype, CMS deployed a feature complete version of the software in a testbed in August. This testbed showed that a common solution where ATLAS and CMS share the same job submission engine is a viable solution.

However, the use of PanDA in a production environment still needs some joint work between the two experiments. Some open issues need to be addressed first. For example, the installation process of the product should use a package managing system, like the Red Hat package manager (RPM). Then, the database schema naming convention has to be reviewed since it contains ATLAS specific elements. After, improvements on the scheduling system could be done since currently there is no clear separation between the scheduling algorithm and the source code, and scheduling policies cannot be given as external configuration files. Finally, the integration of glExec to facilitate site-level user traceability is going on; this will improve the current situation where the user’s payload is executed with the pilot credentials, and the only option to identify who is running a job is to analyze the pilot logs.

Directions for future work regarding the CMS testbed include the demonstration of the viability of the design concept of multiple workflow by integrating the testbed task manager components with the current CMS production workflow system. A stress test to demonstrate whether the system is able to manage without disruptions the CMS load in terms of job is an essential task which needs to be done before a production release. The integration of an automated and distributed analysis testing system like Hammercloud is also under consideration.

Acknowledgments
The authors would like to thank the ATLAS PanDA team for the suggestions and for the contributions received during the development, and during deployment of the CMS common analysis framework testbed

References
[6] Richardson L and Ruby S 2007 RESTful Web Services - Web services for the real world (O’Reilly Media)

1 Address of the PanDAMon instance: http://pandamon-cms-dev.cern.ch


