# Experience in CMS with the common analysis framework project

**Marco Mascheroni[1], Daniele Spiga[2], Tommaso Boccali[3], Daniele Bonacorsi[4], Mattia Cinquilli[2], Domenico Giordano[2], Federica Fanzago[5], Ian Fisk[6], Maria Girone[2], Jose Hernandez[7], Preslav Konstantinov[8], Niccolò Magini[2], Valentina Mancinelli[2], Hassen Riahi[9], Lola Saiz Santos[10], Eric W Vaandering[6]**

[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. This distributed computing infrastructure was based on elements of PanDA and prior CMS workflow tools. After an extensive feasibility study and development of a proof-of-concept prototype, the project now has a basic infrastructure that supports the analysis use cases of both experiments via common services. In this paper we will discuss the state of the current solution and give an overview of all the components of the system.

## 1. Introduction

ATLAS (A Toroidal LHC Apparatus) and CMS (Compact Muon Solenoid) are the two largest LHC multi-purpose detector experiments at CERN. To meet their computing needs, they make use of the resources and the infrastructure provided by the Worldwide LHC Computing Grid (WLCG) [1]. Traditionally, the physics and middleware software needed for data analysis have been developed separately. However, since the available development effort is steadily decreasing, sustainability is becoming a concern. A two-year shutdown of the LHC 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[2].

Even though experiments have similar analysis workflows, common solutions among high level submission tools have never been developed; each experiment has its own workflow manager. ATLAS uses a number of clients that connect to PanDA (ATLAS Production and Distributed

Analysis [3]), while CMS uses CRAB (CMS Remote Analysis Builder [4]) in conjunction with GlideInWMS [5]. In March 2012, ATLAS, CMS, and the experiment support group at CERN started a feasibility study to explore the development of a common analysis framework, based on PanDA, to submit jobs to the WLCG computing resources.

The results of the feasability study showed there are many similarities between the workflow of the two experiments. The separate solutions both rely on: an experiment-specific frontend (for perform task management), data discovery, job splitting and submission. Further, there is a job manager that handles individual jobs, their scheduling, killing and resubmission. The common proposal targeted the latter, as it has more commonalities 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 into a more consolidated version installed on 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[6] (Setcion 2.1), and the common components which come from the PanDA environment (Section 2.2). Setcion 3 is about the testbed deployed by CMS in order to test the whole system. Finally, the CMS experiences gained during this project are presented in Section 4.

## 2. Overall Architecture

The overall architecture of the system is shown in Figure 2. There are experiment-specific components and other shared components. The experiment-specific components deal with user data analysis request. In such a request, a user specifies some input parameters (for example, dataset to be analyzed, how to split the dataset), and the code he or she wants to run. For each request a *task* is created in a central database. Each task is then broken down into multiple parts, (*jobs*), for independent execution. Jobs belonging to the same task execute the same user code but on different portions of the input dataset. The experiment-specific components do not store information about single jobs, but once job specifications have been created, they are sent to one of the shared components which stores them in another database. The shared components are a *common job manager* between both ATLAS and CMS.

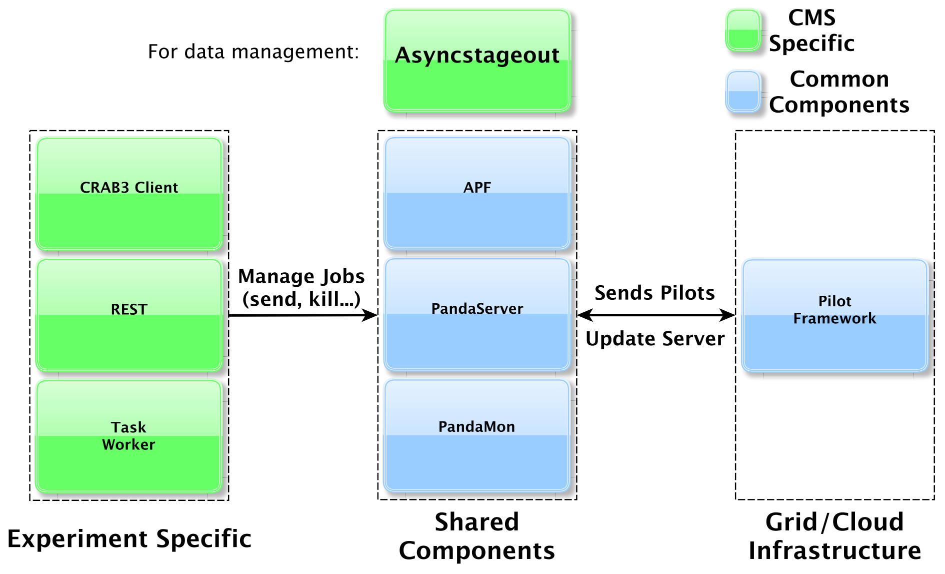

**Figure 1.** The overall architecture of the system

There are three major experiment-specific: a lightweight client (in this implementation, a python-based command line interface) for user interaction, a REpresentational State Transfer interface (REST)[7] which validates and inserts the client task request into an Oracle database, and a task worker which takes the requests from the REST, generates the job specifications and submits them to the job manager.

The primary common components are: the PanDA server, which gets the job specifications from the task worker and puts them into an Oracle DB, and the auto pilot factory (APF), which sends jobs to the WLCG according to the queue information in the PanDA server. For optimization reasons, single jobs are not sent to the distributed infrustructure, but instead *pilot* jobs are sent. A pilot job keeps running on a worker node, and executes several payload jobs. In each “loop, a pilot downloads the user code associated to a job, executes the code, and then updates the PanDA server with the resulting job state. It then starts a new loop, asking the PanDA server for another job.

Finally, a data management component (*asyncstageout*) moves the output files from the local storage to the remote one [8].

In the following two subsections, we will discuss the components specific to CMS experiment, then the common components, and the integration effort required by CMS to bring them into the existing computing environment.

## *2.1. CMS experiment specific components*

The software specific to the CMS experiment for task management is based on a client-server model. This model simplifies the user interaction by handling most of the logic on server side and by automating operations that do not need direct user interaction. Another advantage of the client-server model is optimized resource usage; in particular, interacting with the Grid middleware from only a few central points. For example, we can improve the scalability of the whole system by significantly reducing the number of requests to external services via properly use of HTTP protocol headers and caching mechanisms.

Figure 2 presents a detailed view of the interactions between the experiment specific components in the system.

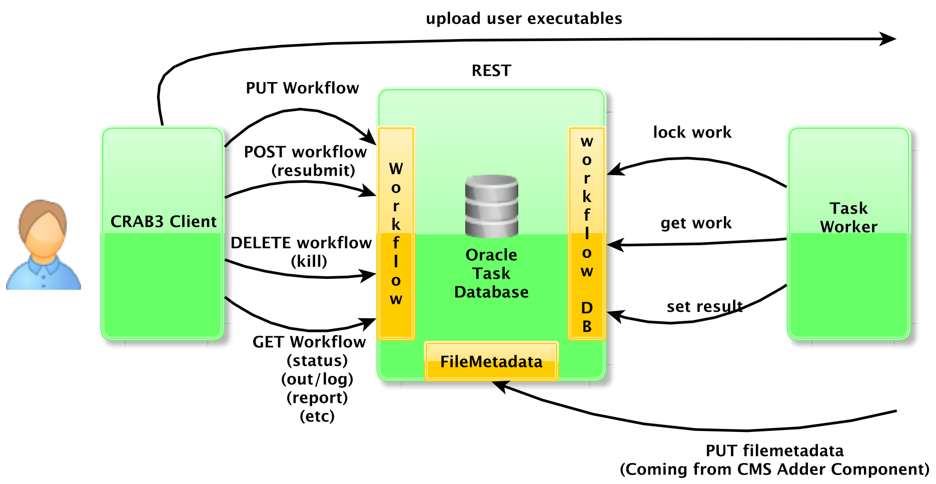

**Figure 2.** The architecture of the CMS components

**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 (`filemetadata`)

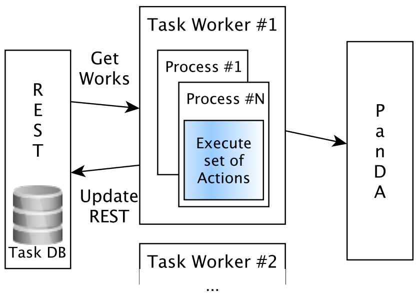

**Figure 3.** Structure of a TaskWorker: multi-process and distributed.

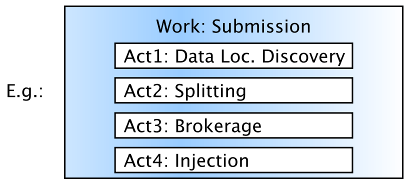

**Figure 4.** Execution of the set of actions associated to the work.

is used by the PanDA server to insert job metadata when a job finishes; and the third resource (`workflowdb`) is used by the TaskWorker component to update the status of the manipulated tasks. There are many advantages in using REST-ful approaches. This provides is a clear separation between the data and the interface: currently we are using an Oracle database to store the tasks, but we could change databases without modifying the client. From a security point-of-view, user authentication is handled with the HTTPS protocol using X509 certificates; no custom authentication layer is needed. Furthermore, REST, as a popular design paradigm, is well-integrated with many scripting languages.

This component has been designed to support multiple job managers; while PandDA is currently the only one supported, another system (such as HTCondor) could be added in the future.

**CRABClient:** The main commands of the CRABClient are `submit`, `resubmit`, `status`, `getoutput`, `getlog`, and `report`. The `submit` command loads the user configuration and submits it to the REST interface (a HTTPS PUT request with JSON serialization). The other commands are similar in the sense that all the commands trigger actions by updating the database via REST. The client is a lightweight python package with minimal external dependencies (just `pycurl` and `python2.6` are needed), which allows for simple client (re-)deployment. A modular and pluggable architecture has been designed: adding new commands to the client or a new type of job is a matter of implementing a new subclass.

**TaskWorker:** The TaskWorker component is responsible for fetching work from the REST interface and implementing `submission`, `kill`, and `resubmit` commands 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 for executing one unit of work on a task. Figure 4 shows the actions associated with `submission`.

After executing the work unit, 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 users tasks. After a set of job specifications are generated from a task, they are stored into 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 components are shown in Figure 5.

**PanDA server:** The PanDA server is the main component of the system and manages all

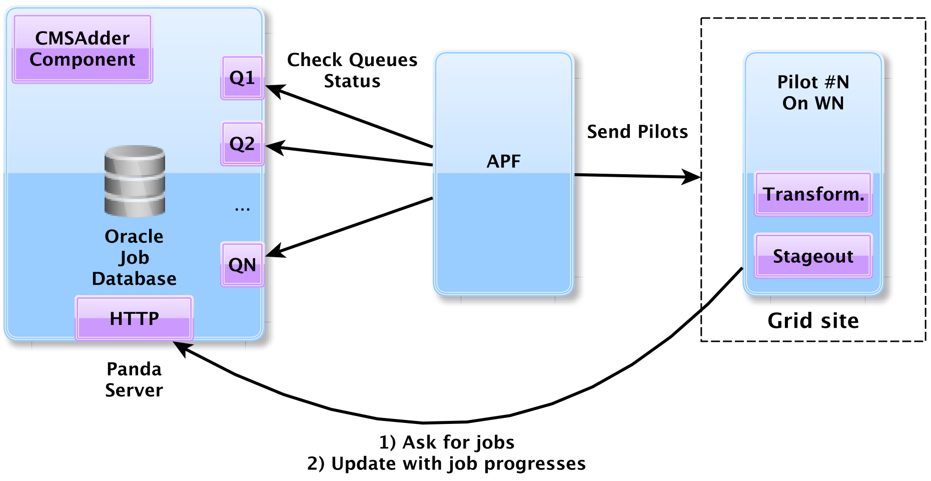

**Figure 5.** The architecture of the shared components

job information centrally in an Oracle database. Its main activities are: managing queues of jobs, and to handle job priority / scheduling.

The CMS and ATLAS installations share the same PanDA server code (albeit with a different configuration) and the database schema. The deployment of the PanDA server required some effort and development effort to remove ATLAS-specific elements from the default installation scripts and configuration files.

Experiment-specific data management components have been used: CMS developed a custom version of the `AdderComponent`, a component which executes a callback when a job finishes. The CMS plugin informs the CMS-specific AsyncStageout component that a job is ready to start transferring output.

**Auto Pilot Factory:** The auto pilot factory component [9] queries all the queues of jobs in the PanDA server, and, based on the queue backlog, it keeps an appropriate number of pilot jobs running on the respective grid sites. An increased number of payload jobs queued for a site in PanDA will cause the APF component to send additional pilots jobs to the site.

The APF component has no CMS-specific aspects and, from the CMS integration perspective, it proved to be easy to manage and deploy. Only one new feature was required - CMS needed pilots to be submitted from a pool of user certificates.

**Pilot framework:** The pilots running on the worker nodes queries the PanDA server for new payload jobs. If a payload job is returned, the pilot will download and execute the user code. Updates about the job status are sent to the PanDA server, which holds all the information about the jobs, and uses the updates to populate the PanDA monitor system.

The pilot framework has undergone some modifications to allow the execution of experiment-specific code. The code has been reorganized to result in a more pluggable architecture, and two CMS plugins have been developed to allow the execution of CMS workflows: the stageout plugin, and the transformation plugin (which is responsible for handling the user's code execution). More information about the PanDA pilots can be found in [10].

## 3. The CMS testbed

A CMS testbed to verify functionality has been deployed. The testbed showed that the CMS and ATLAS experiments can use the same submission infrastructure, and it showed that it is possible for the two experiments to share expertise and development effort in the future.

The configuration of the computing sites in the pilot factory has been automatically generated from the ATLAS Grid Information System (AGIS [11]). AGIS has been populated by hand; the necessary information have been taken from SiteDB [12] and from the GlideInWMS configuration. This is viewed as a temporary solution as site administrators in CMS do not use AGIS; a production environment will need a different workflow. There are 47 CMS sites in the testbed.

The PanDA server has been installed following the instructions provided by ATLAS developers. PanDA monitor instances have been deployed [1]. The PanDA monitor provides an easily-accesible interface for resources, jobs status, and accessing the outputs. Further, the monitor provides system administrators with a reasonable interface to investigate job issues at each site.

## 4. Conclusions

Driven by the need of a more sustainable computing model, CMS, ATLAS, and CERN-IT built and evaluated 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 testbed in August 2013. This testbed showed that a common solution, with ATLAS and CMS share the same job submission engine, is a viable solution.

The use of PanDA in a production environment still requires some joint future work. For example, the installation process should use a package managing system, like the Red Hat package manager (RPM). The database schema naming convention should be reviewed to generalize ATLAS-specific elements. Scheduling policies improvements are needed as currently there is no clear separation between the scheduling algorithm configuration and implementation in the source cod; scheduling policies cannot be given as external configuration files. Finally, the integration of glExec to facilitate site-level user payload traceability is ongoing; this will improve the current situation where the user's payload is executed with the pilot credentials. Right now, the only way to identify a payload jobs owner is to analyze the pilot logs.

Future work on the CMS testbed includes the demonstrating the viability of having multiple workflow systems; this could be done by integrating the testbed task manager components with the current CMS production workflow system. A stress-test demonstrating system scalability is needed before a production release. The adoption of an automated stress testing system like Hammercloud [13] is also under consideration.

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

## References

[1] Bird I *et al.* 2005 LHC computing grid. technical design report Tech. Rep. CERN-LHCC-2005-024
[2] Girone M *et al.* 2012 *Journal of Physics: Conference Series* **396** 032048 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032048`
[3] Maeno T, De K, Wenaus T, Nilsson P, Stewart G A, Walker R, Stradling A, Caballero J, Potekhin M, Smith D and Collaboration T A 2011 *Journal of Physics: Conference Series* **331** 072024 URL `http://stacks.iop.org/1742-6596/331/i=7/a=072024`
[4] Spiga D, Cinquilli M, Codispoti G, Fanfani A, Fanzago F, Farina F, Lacaprara S, Miccio E, Riahi H and Vaandering E 2010 *Journal of Physics: Conference Series* **219** 072019 URL `http://stacks.iop.org/1742-6596/219/i=7/a=072019`
[5] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Wurthwein F 2009 The pilot way to grid resources using glideinwms *Computer Science and Information Engineering, 2009 WRI World Congress on* vol 2 pp 428–432

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

[6] Cinquilli M, Spiga D, Grandi C, Hernndez J M, Konstantinov P, Mascheroni M, Riahi H and Vaandering E 2012 *Journal of Physics: Conference Series* **396** 032026 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032026`

[7] Richardson L and Ruby S 2007 *RESTful Web Services - Web services for the real world* (O'Reilly Media)

[8] Cinquilli M, Riahi H, Spiga D, Grandi C, Mancinelli V, Mascheroni M, Pepe F and Vaandering E 2012 *Journal of Physics: Conference Series* **396** 032025 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032025`

[9] Caballero J, Hover J, Love P and Stewart G A 2012 *Journal of Physics: Conference Series* **396** 032016 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032016`

[10] Zhao X, Hover J, Wlodek T, Wenaus T, Frey J, Tannenbaum T, Livny M and the ATLAS Collaboration 2011 *Journal of Physics: Conference Series* **331** 072069 URL `http://stacks.iop.org/1742-6596/331/i=7/a=072069`

[11] Anisenkov A, Klimentov A, Kuskov R and Wenaus T 2011 *Journal of Physics: Conference Series* **331** 072002 URL `http://stacks.iop.org/1742-6596/331/i=7/a=072002`

[12] Metson S, Bonacorsi D, Ferreira M D and Egeland R 2010 *Journal of Physics: Conference Series* **219** 072044 URL `http://stacks.iop.org/1742-6596/219/i=7/a=072044`

[13] Ster D C V D, Elmsheuser J, beda Garca M and Paladin M 2011 *Journal of Physics: Conference Series* **331** 072036 URL `http://stacks.iop.org/1742-6596/331/i=7/a=072036`