

## Automation of user analysis workflow in CMS

D Spiga<sup>1</sup>, M Cinquilli<sup>2</sup>, G Codispoti<sup>3</sup>, A Fanfani<sup>3</sup>, F Fanzago<sup>4</sup>,  
F Farina<sup>5</sup>, S Lacaprara<sup>6</sup>, E Miccio<sup>7</sup>, H Riahi<sup>2</sup>, E Vaandering<sup>8</sup>

<sup>1</sup>CERN

<sup>2</sup>Università and INFN Perugia

<sup>3</sup>Università and INFN Bologna

<sup>4</sup>INFN-Padova

<sup>5</sup>INFN Milano Bicocca

<sup>6</sup>INFN-LNL

<sup>7</sup>INFN-CNAF

<sup>8</sup>FNAL

E-mail: Daniele.Spiga@cern.ch, Mattia.Cinquilli@cern.ch,  
Giuseppe.Codispoti@bo.infn.it, Alessandra.Fanfani@bo.infn.it,  
Federica.Fanzago@cern.ch, Fabio.Farina@cern.ch, Stefano.Lacaprara@pd.infn.it,  
vmiccio@mail.cern.ch, Hassen.Riahi@pg.infn.it, ewv@fnal.gov

**Abstract.** CMS has a distributed computing model, based on a hierarchy of tiered regional computing centres. However, the end physicist is not interested in the details of the computing model nor the complexity of the underlying infrastructure, but only to access and use efficiently and easily the remote services. The CMS Remote Analysis Builder (CRAB) is the official CMS tool that allows the access to the distributed data in a transparent way. We present the current development direction, which is focused on improving the interface presented to the user and adding intelligence to CRAB such that it can be used to automate more and more the work done on behalf of user. We also present the status of deployment of the CRAB system and the lessons learnt in deploying this tool to the CMS collaboration.

### 1. Introduction

The Compact Muon Solenoid (CMS) experiment [1] is one of the two general purpose physics experiments at the European Laboratory for Particle Physics (CERN) [2] starting operation in 2009. The scientific analysis of data taken by the detector and MonteCarlo events simulation requires a large amount of well organized computing resources. To guarantee more than 2000 CMS collaborators located in 40 countries around the world to be able to carry out their physics analysis with minimal geographical and processing constraints the CMS experiment has had a worldwide distributed computing model [3] from the beginning. The CMS distributed model implements the Grid middleware to manage three main levels, or tiers, of computing.

Tier 0 (T0) is located at CERN where the accelerator and experiment are located and includes 20% of the total required computing resources of CMS. The next level is represented by Tier 1 (T1) regional centers which represent 40% followed by the Tier 2 (T2) centers which represent another 40% of the total required computing resources of CMS. Each Tier level has its well defined responsibilities mainly differentiated by their resources dedication.

Also a set of specialized tools have been developed on top of WorldWide LHC Computing Grid (WLCG) [4] and Open Science Grid (OSG) [5] to manage the distributed resources. Phedex [6]

distributes data across the destination sites; Dataset Bookkeeping System (DBS) [8] is the data discovery service allowing to track datasets both for real data and simulated events. The ProductionSystem [7] manages Monte Carlo data production and the CMS Remote Analysis Builder (CRAB), the interface proposed to the physicist to perform the Grid distributed analysis in a transparent way. Section 2 gives an overview of the analysis workflow in the CMS experiment. Section 3 discusses how the automation has been implemented and near future development. Section 4 discusses the status of deployment, while Sections 5 and 6 report on results and conclusions respectively.

## 2. Overview

The CMS analysis model is data location driven. The user analysis runs where data are located, and foresees that all CMS users must use the Grid in order to perform their own analysis. Within the computing model the Tier-2 level represents the primary analysis facilities for CMS, where more flexible, user driven activities can occur.

To optimize the distributed resources usage an association of users to Tier-2 centers either based on geography or regions of interest has been defined. In addition the Tier-2 centers are also associated with one or more CMS defined Physics Groups.

The typical analysis workflow can be summarized as follows:

- User runs interactively on small data sample developing the analysis code.
- Users analysis code is shipped to the site where sample is located.
- Results are made available to the user for the final plot production.

The workflow involves the concept of task and job. The job is the traditional queue system concept, corresponding to a single instance of an application started on a worker node with a specific configuration and output. A task is generally composed of many jobs.

From a technical point of view, each one of the previously listed steps corresponds to one or more interactions with a set of different services. Job preparation requires both to query the Data Bookkeeping and Location System (DBS) and to interact with the user local environment setup. The job submission, tracking, output retrieval and all the other batch interactions require to interact with the middleware, and in particular to configure the Grid job in the case of WLCG consists of a file filled using the Job Description Language (JDL) interpreted by the Workload Management System (WMS) [9] (a similar approach is required for OSG and many batch systems). Also, specific T2s configuration parameters, (e.g. namespaces, storage endpoint) are made available querying the CMS Sites information (SiteDB) [10] service.

All this complexity can represent a hard task for the end user who is mainly interested in the end-results of the job. To hide as much as possible the computing infrastructure, but always preserving the flexibility required by end user analysis activity, CMS designed and deployed a dedicated tool, called the CMS Remote Analysis Builder (CRAB).

## 3. Automation: Client Server Architecture

The first implementation of CRAB was based on a standalone tool. The interaction with the Grid was only direct, leaving to the user all the tasks such as submission, status check and output retrieval. After a long experience achieved with this CRAB setup, CMS planned and designed its natural evolution to a more scalable, automated and powerful architecture, based on a Client-Server approach. The guidelines of the evolution were mostly aimed to keep the interface proposed to the user invariable w.r.t. the previous standalone implementation, and to design a server as similar as possible to the Production and T0 System. The programming language used to develop CRAB is Python [11].

```
[CRAB]
jobtype          = cmsww
scheduler        = glite
server_name      = Bari

[CMSSW]
datasetpath      = /HiggsGammaGammaM120/CMSSW_2_0_0_1202115095/RECO
pset             = myHiggsAnalyzer.py
total_number_of_events = -1
events_per_job   = 1000

[USER]
copy_data        = 1
storage_name     = T2_IT_Legnano
remote_dir       = myHiggsAnalysis
```

**Figure 1.** Minimal CRAB configuration file organized in three sections containing different configuration parameters

### 3.1. The Client

The client is a command line application used by the user on the User Interface(UI). It takes care of the local environment interaction, packing private user library and code in order to replicate remotely the very same local configuration; it queries the DBS performing the data discovery. Finally it implements the communication with the server, based on web services technology, using SOAP [12]. The client uses an SQLite [13] database for logging purpose. The interaction with the database is performed using the BossLite [14] API.

To interact with CRAB, the end user just needs to use a simple configuration file organized into sections containing key-value pairs (Fig. 1), and uses it relying on the friendly CLI proposed.

### 3.2. The Server

The internal CRAB server architecture (Fig. 2) is based on components implemented as independent agents communicating through an asynchronous and persistent message service (as a publish and subscribe model) based on a MySQL [15] database. Each agent takes charge of specific operations and the modular design allows new features to be added to the service in a transparent way. Most of the components implement a multithreading approach, using safe connection to the database. This feature allows to manage many tasks at the same time shortening the delay time for a single operation that has to be accomplished on many tasks. This is a key point for a service with the role to provide the user with data to analyse in order to produce the final physics results.

As shown in Figure 2 the CRAB Server architecture relies on a dedicated Grid Storage Element to store the input/output-sandboxes transferred by/from the Client/Worker Nodes. The server uses a specific interface made up by a set of API and a core with hierarchical classes which implement different protocols, allowing to interact transparently with the associated storage area. By design the storage area is not required to be close to the service itself; such feature plays a key role in the Grid Scheduler interaction. The latter is performed through BossLite API designed to guarantees the complete interoperability between different flavours of both the Grid and local batch systems.

The new architecture offers many opportunities for improving the automation by reducing the number of operations to be performed by the physicist. The user prepares the configuration and responsibility for task execution is delegated to the server. Most of the operations, which do not need direct user actions, can be addressed by the server, a 24x7 service designed to care care of managing user's tasks (using exactly the user credential). After the request submission the user should just wait for the server notification about the output availability. The server is the place where to implement the intelligence needed to detect the jobs success/failure and takes action for eventual job resubmission. The aim of the server is to improve scalability of the whole system providing to CMS specific functionality. Finally the service represents also a natural place where more advanced and automated workflows can be integrated. An example of a real use case will be supported is the new submission logic that uses the trigger system. The

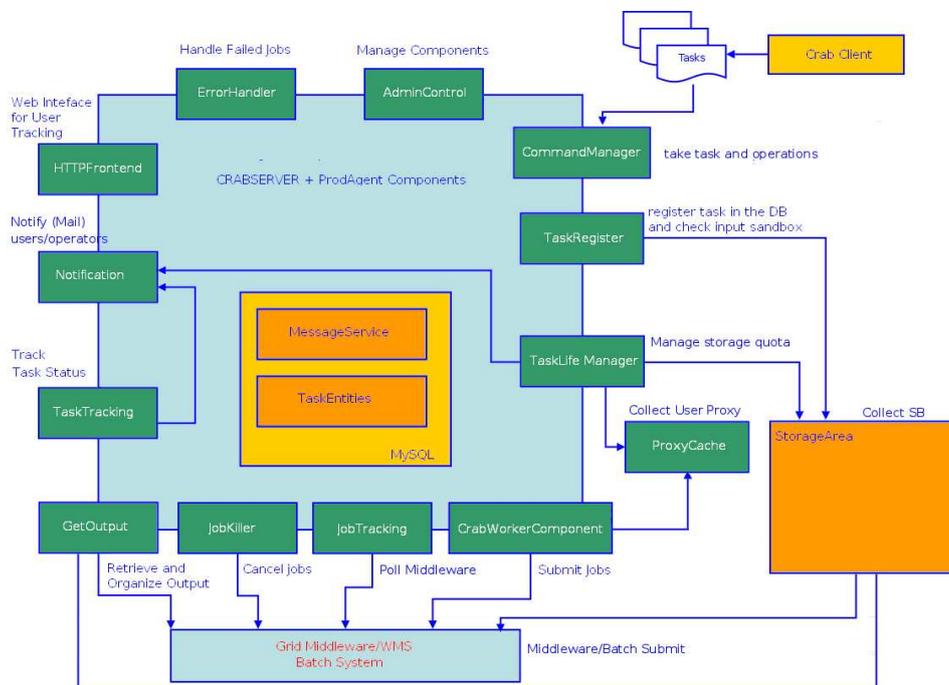
flow is summarized in the following:

- User submits jobs to the server asking to run on a defined dataset.
- Server queues submitted jobs and polls data bockkeeping system.
- Once requested data becomes available the server automatically releases jobs.

#### 4. CRAB Server Deployment

At the end of 2008 the CRAB server deployment activity started and at the time of writing 4 instances have been installed around the world. Two instances have been deployed in Italy, one in France, one in Germany plus one at CERN. All of them use the gLite WMS to submit jobs. One more instance is under deployment in USA which is actually submitting jobs through glide-in WMS [16]. One of the lessons learnt thanks to this activity, is the importance to have an easy access to the information of all the services and daemons running within the server. For this purpose the HTTPFrontend component, which provides a web interface to the server, has been implemented providing intuitive interfaces to the most relevant information for both users and administrators. For each component/daemon/service running within the server the web interface shows the related status, the CPU and Memory usage. In addition it gives information on the number of users submitting jobs, the number of jobs and tasks, and related status.

These features are used by the administrator of the service in order to promptly detect problems. Figure 3 illustrates a snapshot of a relevant summary page presented by the HTTPFrontend component. The Server is by definition a centralized system collecting users jobs, which could in principle represents a single point of failures. Having many instances running at different sites will of course avoid this eventual issue.



**Figure 2.** Schema of the CRAB Server architecture and its internal inter-connections. Shown are the various components including the mysql core (in yellow) and storage elements (in orange).

## 5. Results

The huge CRAB user community, which during 2008 saw more than 1000 physicists submitting jobs, generates a lot of feedback. This is a really important aspect in view of the LHC start-up since feedback contributed to define development and operation priority in order to be prepared for the first collisions. Figure 4 shows a mean value of  $\sim 90$  distinct users per day using the distributed computing infrastructure and the described machinery to run over simulated dataset and cosmic ray samples. The large user base impacts strongly on the effort required to develop, support and deploy tools that support analysis on the grid. The overall efficiency, described in term of job success rate is reported in Figure 5. The resulting fraction of succeeded jobs is approaching 60%. From a detailed analysis (Fig. 6) aimed to understand root problems of the  $\sim 40\%$  of failure rate, it appears that 12% of problems come from the Grid infrastructure issues (e.g. site specific, gLite WMS etc.) while 25% can be decomposed mainly as:

- User configurations errors.
- Remote stage out issues.
- Few % of failures reading data at site.

These are the results obtained considering the statistics achieved from July 2008 to March 2009 and the related data have been achieved by the CMS monitoring project (DashBoard) [17]

These results are of course an important indication that something must be done in order to improve the overall analysis job efficiency. A part of the irreducible fraction of failures due to problems within the user own code, CRAB probably could help on this. Some possible ideas to reduce the actual  $\sim 28\%$  of failures can be:

- To provide some interactive checks to help in preventing user misconfigurations.
- To allow to run on its workstation the very same crab jobExecutor in order to debug problems.
- To check the remote stage out configuration at creation time.

Note that about the stage out issues strategies for more stable solutions have being discussed within the Collaboration.

### CrabServer Components and Services Monitoring

Display the status of components and active service in this CrabServer:

Status of all components and external services of this CrabServer

Allow to access components logs through web:

Show logs for

Watch message

Show message

or whatch mes

(Work in progress)

Display components CPU usage:

Show CPU plot for  for last

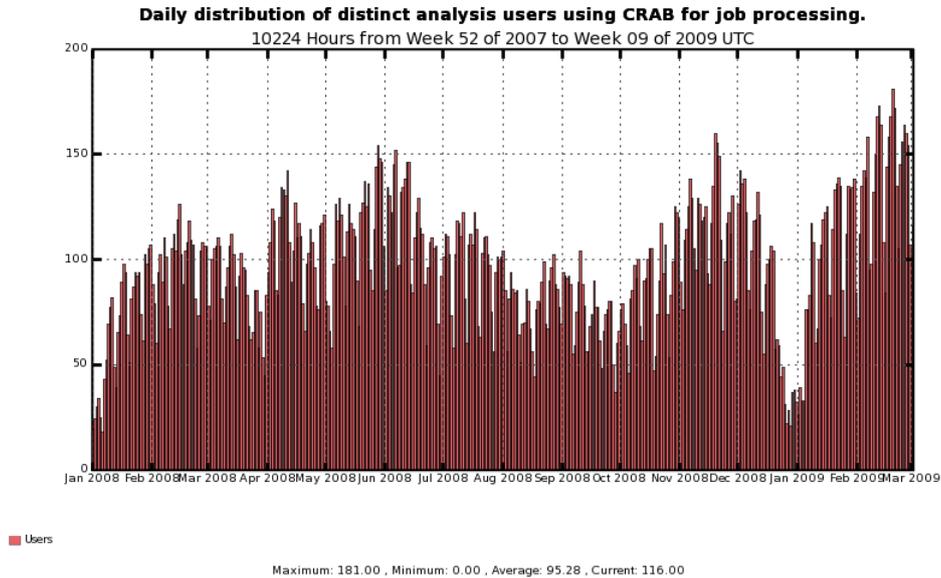
Display services CPU usage:

Show CPU plot for  for last

Display resources usage:

Show plot for  for last

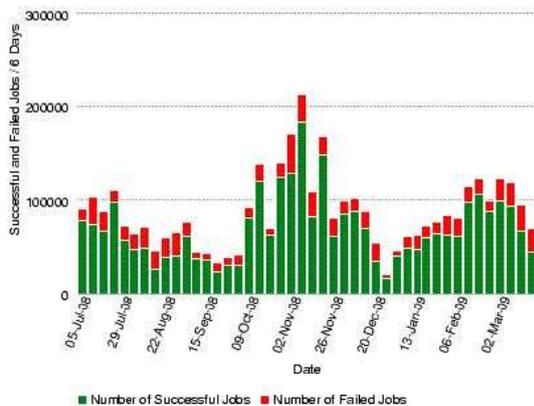
**Figure 3.** Snapshot of one of the web interface implemented by the server. In particular this show the relevant informations for administrator purpose.



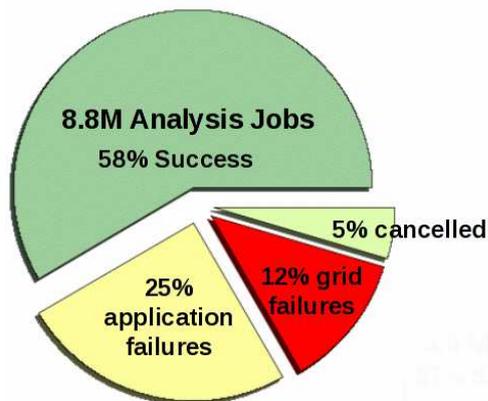
**Figure 4.** Number of CRAB daily distinct users in the 2008.

## 6. Conclusion and perspective

At the time of writing more than 1/3 of the whole CRAB user community migrated to use the server for jobs submission. The implemented client-server architecture demonstrates that automation of the analysis workflow can be supported. The stress test performed demonstrated that the CMS analysis jobs scale can be reached using about 4 CRAB server instances, even if the final deployment strategy is still something under discussion within the Collaboration. From the development point of view in the short term the effort will be spent on user interface optimization adding new functionality mostly focused on the physics domain like the implementation of minimal report per task to give to user the information he needs to create meaningful histograms out of his outputs. As middle/long term plan the goal of the development team is to migrate to use the CMS Workload Management common core [18], under development at the time of writing.



**Figure 5.** Success rate of jobs CMS jobs submitted with CRAB. Values computed referring to data achieved from July 2008 to March 2009.



**Figure 6.** Pie chart showing success rate of analysis jobs. Values computed referring to data achieved from July 2008 to March 2009.

## References

- [1] S. CHARTRCHYAN et al., The CMS Experiment at CERN LHC Journal of Instrumentation, vol 3, pp.s08004 (2008).
- [2] LHC Homepage, <http://cern.ch/lhc-new-homepage/>
- [3] THE CMS COLLABORATION CMS: The computing project. Technical design report., CERN-LHCC-2005-023, 166pp (2005).
- [4] LHC Computing Grid (LCG), Web Page, <http://lcg.web.cern.ch/LCG/> and LCG Computing Grid - Technical Design Report, LCG-TDR-001 CERN/LHCC 2005-024, (2005).
- [5] OSG Web Page, <http://opensciencegrid.org>
- [6] A. DELGADO PERIS et al. Data location, transfer. and bookkeeping in CMS Nucl.Phys.Proc.Suppl.177-178:279-280,(2008).
- [7] S. WAKEFIELD ET AL. Large Scale Job Management and Experience in Recent Data Challenges within the LHC CMS experiment.
- [8] A. AFAQ et al., The CMS dataset bookkeeping service, 2008 J. Phys.:Conf. Ser. 119 072001.
- [9] P. ANDREETTO et al., The gLite Workload Management System Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2007, Victoria, British Columbia (CA), Sep 2007.
- [10] S. METSON ET AL. SiteDB: Marshalling the people and resources available to CMS Proceedings of this conference.
- [11] Python Programming Language <http://www.python.org/>
- [12] SOAP Messaging Framework <http://www.w3.org/TR/soap/>
- [13] SQLite Home Page [www.sqlite.org](http://www.sqlite.org)
- [14] G. CODISPOTI ET AL. Use of the gLite-WMS in CMS for production and analysis Proceedings of this conference.
- [15] MySQL Open Source Database <http://www.mysql.com/>
- [16] S. PADHI ET AL. Use of glide-ins in CMS for production and analysis Proceedings of this conference.
- [17] JULIA ANDREEVA ET AL. Dashboard for the LHC experiments. Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2007, Victoria, British Columbia (CA), Sep 2007.
- [18] S. WAKEFIELD ET AL. Job Life Cycle Management libraries for CMS Workflow Management Projects Proceedings of this conference.