CRAB: Distributed analysis tool for CMS

Leonardo Sala on behalf of the CMS Collaboration

leonardo.sala@cern.ch

Abstract. CMS has a distributed computing model, based on a hierarchy of tiered regional computing centers and adopts a data driven model for the end user analysis. This model foresees that jobs are submitted to the analysis resources where data are hosted. The increasing complexity of the whole computing infrastructure makes the simple analysis work flow more and more complicated for the end user. CMS has developed and deployed a dedicated tool named CRAB (CMS Remote Analysis Builder) in order to guarantee the physicists an efficient access to the distributed data whilst hiding the underlying complexity.

This tool is used by CMS to enable the running of physics analysis jobs in a transparent manner over data distributed across sites. It factorizes out the interaction with the underlying batch farms, grid infrastructure and CMS data management tools, allowing the user to deal only with a simple and intuitive interface.

We present the CRAB architecture, as well as the current status and lessons learnt in deploying this tool for use by the CMS collaboration. We also present the future development of the CRAB system.

Keywords: CMS, CRAB, distributed analysis, Grid
PACS: 07.05.-t, 07.05.Kf

1. INTRODUCTION

The CMS experiment [1] is one of the four LHC experiments at CERN. Its large scientific programme leads to an equal amount of data and Monte Carlo simulations to be analyzed by a community composed by more than 2000 physicists, spread out all over the world. This leads to a worldwide computing model [2] which implements Grid middleware and is composed by a three-tiered architecture:

- The Tier 0, located at CERN and including 20% of the total computing resources. It takes care of data reconstruction;
- The Tier 1s, regional centers including 40% of the total computing resources. They are in charge for data reconstruction and reprocessing;
- the Tier 2s, including the remaining 40% of the resources. they are devoted to the support of the analysis activities.

Dedicated tools have been developed on top of WorldWide Computing Grid (WLCG) [3] and Open Science Grid (OSG) [4], in order to:

- Transfer data among the various Tiers: PhEDEx [5];
- Track data (both real and simulated): Data Bookkeeping System (DBS) [6];
- Manage the Monte Carlo production: ProductionSystem [7];
- Let physicists perform analysis on the Grid: CMS Remote Analysis Builder (CRAB) [8].

Section 1.1 will give an overview of the analysis work flow; section 2 will describe CRAB architecture; section 3 will summarize the results of CRAB usage in a production system; and 4 will outline the conclusions and future plans.

1.1. The Analysis Work flow

As data is organized in large datasets, analysis activities are data location driven to minimize network occupancy. The usage of resources is optimized with the association of Tier 2s based on geography or Physics Groups of interest.

The typical analysis work flow performed by users is:

- An analysis program is developed and tested locally on a small amount of events;
- The user decides on which datasets (or part of) to run his code;

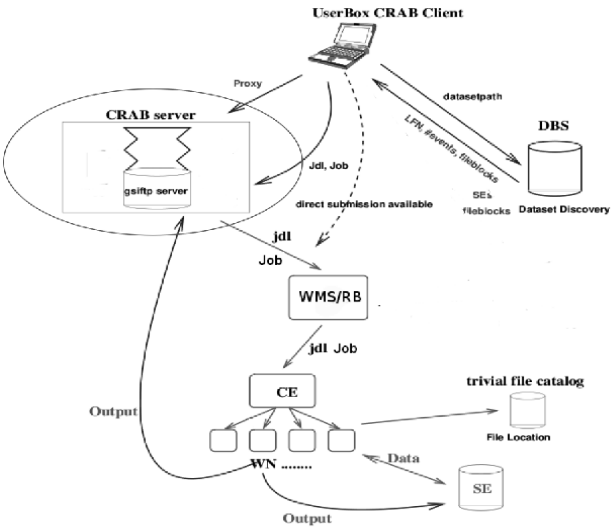

FIGURE 1: CRAB work flow.

- User's code is *shipped* (not compiled) to the destination site, using the data location paradigm described above;
- Results are made available to user, that can use them for further processing.

The last two points need the interaction with different services, e.g. DBS, or also need some knowledge of the Grid infrastructure, first of all the Job Description Language (JDL) [9] or site specific configuration details like the storage endpoint, which in CMS are stored in SiteDB [10].

2. THE CMS REMOTE ANALYSIS BUILDER (CRAB)

CRAB has been designed to hide the complex interactions necessary to perform physics analysis in a Grid environment, namely: location of data samples, performed through a query to DBS; job splitting following user wishes and data location at sites; preparation of proper JDL scripts; interaction with the Grid flavors and local batch system (WLCG, OSG, lsf..) for job execution. CRAB furthermore can perform job monitoring and output retrieval, both on a local area or on a Storage Element through standard Grid tools.

CRAB architecture has been developed using a client-server architecture, which allows scalability, central control and debugging, more efficient error handling. Fig. 1 shows the typical work flow which follows a job submission. The language chosen to code CRAB is Python.

2.1. The Client

The client is a Command Line Interface (CLI) which allows a easy user interaction and furthermore the possibility to be integrated into users scripts. When invocated with the `-create` option it takes care of the local environment interaction; packages private user library and code; perform data discovery through DBS; communicate with the server based on web services technology, using SOAP [11]. The client uses an SQLite database for logging purpose. The interaction with the database is performed using the BossLite [12] API. To interact with CRAB, a simple configuration file is needed for the user: this file organized into sections which contains key-value pairs , and uses it relying on the friendly CLI proposed.

2.2. The Server

The CRAB server implements a modular architecture, based on independent agents communicating asynchronously through a persistent message system, based on a MySQL database (Fig. 2).

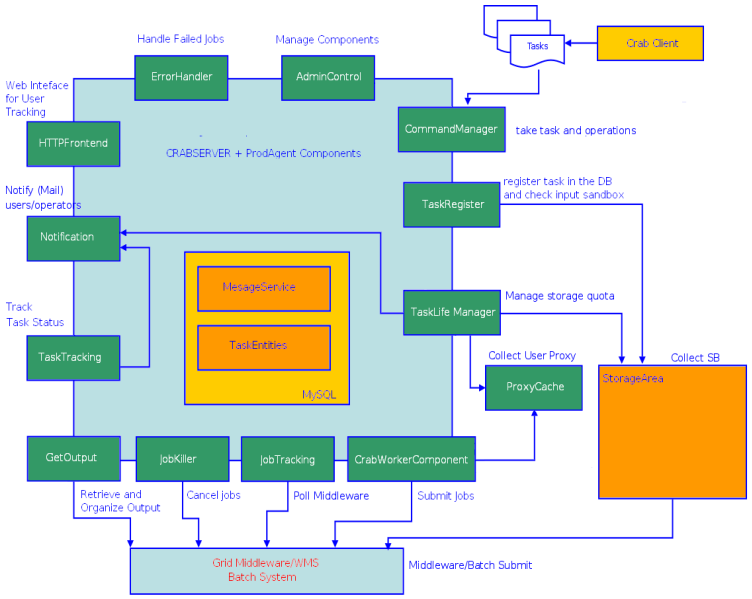

FIGURE 2: Scheme of the CRAB Server architecture and its internal inter-connections; in the schema both the mysql core (in yellow) and storage elements (in orange) have been included.

In order to manage many tasks at the same time many of the independent components implement a multithreading approach, using safe connection to the database. Users sandboxes are stored in a dedicated Storage Element; the server interacts with it using a dedicated set of API and a core with hierarchical classes which implement different protocols, allowing to interact transparently with the associated storage area.

The aim of CRAB server is to provide advanced optimization needed and heavily used by the CMS experiment but actually absent in the Grid MiddleWare. It also provides many opportunities for improving the automation, transferring some sets of operations from the user to a 24x7 service, using the very same user credentials; examples are periodic polling for job status and the sending of an email to the user when the execution reaches a certain threshold of completeness. The server is also the place where to implement the intelligence needed to detect the jobs success/failure and takes action for eventual job resubmission.

Furthermore, having a single gateway for user jobs can help in highlighting single points of failure and eventual issues. The HTTPFrontend component has been designed to provide a web interface to the server and provides intuitive interfaces to the most relevant information for both users and administrators. For each component/demon/service running within the server the web interface show 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. This allows the server administrator to promptly highlight issues and to take effective measures.

3. CRAB USAGE IN THE CMS COLLABORATION

CRAB is used every day by the CMS Collaboration to submit jobs to the Grid infrastructure. Fig. 3 shows that CRAB is used each day by a mean of more than 100 distinct users, with four CRAB servers available among Europe and the US. This heavy usage allows the generation of a lot of feedback; this is very important to drive the development towards the very-next LHC data taking.

Analyzing a statistics of 20 million submitted analysis jobs between 2008 and 2009, it can be shown that the total success rate is 64.1%; an $\approx$9.4% of failures can be addressed to Grid related issues as specific site or WMS failures; canceled jobs are the 3.2% of the total. The biggest part of failures (23.3%) is to be addressed to application specific issues. A further analysis can show that these are mainly composed by:

- User errors (configuration files, code related failures);
- Stage-out on remote Storage Elements;
- A few % of failures in reading data at sites.

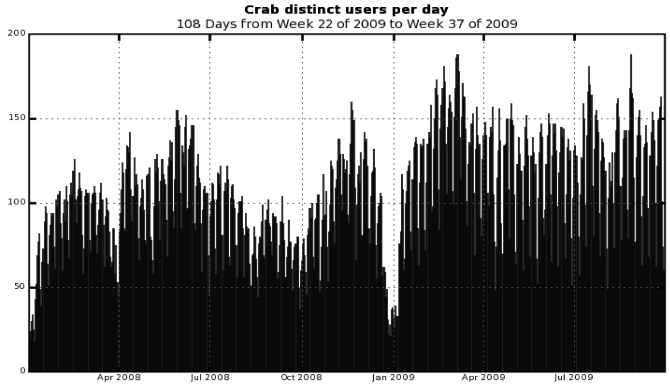

FIGURE 3: CRAB distinct users per day, as retrieved from the CMS Dashboard [13] during the period from June 2009 to September 2009

The actual job performance can be improved with e.g. the introduction of some interactive checks to prevent user misconfigurations, or sanity checks on the remote stageout. Note that about the stage out issues strategies for more stable solutions have being discussed within the Collaboration.

4. CONCLUSIONS

CRAB demonstrated to have met its principal goals, providing a friendly interface for the physics distributed analysis. The server architecture allowed to reach scalability and stability during normal analysis operations: this is further demonstrated by the $\approx$100 users which use CRAB every day.

The development will be aimed in the short term to optimize the actual user interface, focused on the physical domain needs. Further improvements are foreseen also for the error reporting and the internal tracking services. In the middle/long term, the development team is going to migrate to use the CMS Workload Management common core [14], which is under development at the time of writing.

ACKNOWLEDGMENTS

Special thanks to Daniele Spiga, Giovanni Codispoti, Federica Fanzago, Julia Andreeva and all the CRAB development team for the material and suggestions provided.

REFERENCES

1. The CMS Collaboration, JINST 0803:S08004, 2008.
2. The CMS Collaboration, CERN-LHCC-2005-023
3. LHC Computing Grid (LCG), Web Page, http://lcg.web.cern.ch/LCG/ and LCG-TDR-001 CERN/LHCC 2005-024, (2005).
4. OSG Web Page, http:/opensciencegrid.org.
5. A. Delgado Peris et al., Nucl.Phys.Proc.Suppl.177-178:279-280,(2008).
6. A. Afaq et al., 2008 J. Phys.:Conf. Ser. 119 072001.
7. S. Wakefield et al., PoS(ACAT08) 032
8. D. Spiga et al., Nucl.Phys.Proc.Suppl.177-178:267-268 (2008)
9. P. Andreetto et al., Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2007.
10. S. Metson et al., Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2009.
11. SOAP Messaging Framework, http://www.w3.org/TR/soap/
12. G. Codispoti et al., Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2009.
13. Julia Andreeva et al., Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2007.
14. S. Wakefield et al., Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2009.