PROCEEDINGS OF SCIENCE

Status and evolution of CRAB

Fabio Farina*
University and INFN Milano-Bicocca
E-mail: fabio.farina@cern.ch

S. Lacaprara
INFN Legnaro

W. Bacchi
University and INFN Bologna

M. Cinquilli
University and INFN Perugia

G. Codispoti
University and INFN Bologna

M. Corvo
CERN

A. Dorigo
INFN Padova

A. Fanfani
University and INFN Bologna

F. Fanzago
CERN

O. Gutsche
FNAL

C. Kavka
INFN Trieste

M. Merlo
University and INFN Milano-Bicocca

L. Servoli
University and INFN Perugia

D. Spiga
University and INFN Perugia

PoS(ACAT)020

PROCEEDINGS OF SCIENCE

Starting from 2007 the CMS experiment will produce several Pbytes of data each year, to be distributed over many computing centers located in many different countries. The CMS computing model defines how the data are to be distributed such that CMS physicists can access them in an efficient manner in order to perform their physics analysis. CRAB (CMS Remote Analysis Builder) is a specific tool, designed and developed by the CMS collaboration, that facilitates access to the distributed data in a very transparent way. The tool's main feature is the possibility of distributing and parallelizing the local CMS batch data analysis processes over different Grid environments without any specific knowledge of the underlying computational infrastructures. More specifically CRAB allows the transparent usage of WLCG, gLite and OSG middleware. CRAB interacts with both the local user environment, with CMS Data Management services and with the Grid middleware. CRAB has been in production and in routine use by end-users since Spring 2004. It has been extensively used during studies to prepare the Physics Technical Design Report (PTDR) and in the analysis of reconstructed event samples generated during the Computing Software and Analysis Challenge (CSA06). This involved generating thousands of jobs per day at peak rates. In this contribution we discuss the current implementation of CRAB, experience with using it in production and plans for improvements in the immediate future.

ACAT 2007 - XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research
April 23-27, 2007
Amsterdam, Netherlands

*Speaker.

PoS(ACAT)020

1. Introduction

CMS [1] is one of the four experiments that will collect data at LHC [2]. The CMS detector has 15 millions of channels; through them data will be taken at a rate of some TB/s, of which just some will be selected to be written on disk. To catch those data there is an on-line selection system (trigger) that will reduce the frequency of data taken from 40 MHz (LHC frequency) to 100 Hz (writing data frequency), that means 100 MB/s and 1 PB data per year. The use of the grid instruments chosen by LCG [3, 4] and OSG [5] projects allows to solve a complicated problem: the access to the data and the distributed resources by CMS users. The Computing Model of CMS defines an hierarchic architecture of the grid sites with one Tier-0 site (CERN) that is linked with the CMS Data Acquisition System, some Tier-1 sites and many Tier-2 centers. There are also Tier-3 sites mainly department resources. The data will pass first through the Tier-0 site and will be processed and transferred to the lower level sites. For every transfer there is a manipulation and selection of the data in order to reduce the dimension and to get just the interesting part of those. A representation of the CMS multi-tier computing model can be seen in Figure 1.

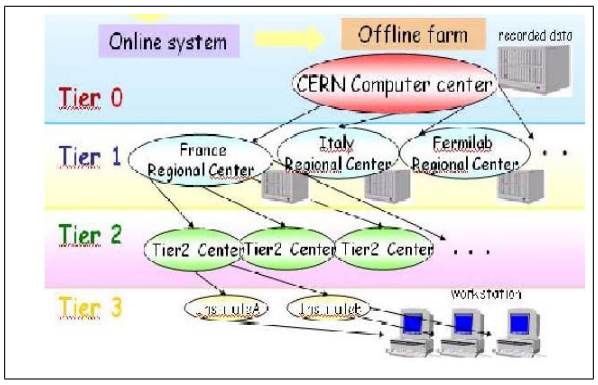

Figure 1: CMS Computational Grid organized in hierarchical-geographical Tiers.

2. Distributed analysis with CRAB

The data analysis on a distributed environment is a complex computing task because the used data have a dimension of hundred MegaBytes, that makes the data transfer not convenient for just some small analysis. Also the other heterogeneous instruments are not local, but distributed all around the world. CRAB is the official CMS analysis software that easily interfaces the user with the grid environment hiding the system complexities. Following the analysis model it allows an easy access to the data distributed over the grid in a very transparent way and the user is not required to have any deep knowledge about the grid. Actually, CRAB simplify the process of CMS analysis allowing to process officially published data and hiding as much as possible the grid complexity to the final user, so that remote data can be accessed with the same facility of local data: CMS user sends through this tool his analysis code to the site where the selected data are. CRAB needs to be installed on the User Interface (UI) -which is the user access point to the grid-, it supports any CMSSW (the CMS software framework) based executable, with any modules/libraries, including

the user provided ones, and finally it deals with the output produced by the executable. From a user point of view, the main steps of the workflow of this software are:

- *Job Creation*: interaction with data discovery services (DBS and DLS) [6] and with the user environment; the task is splitted into smaller jobs and the input sandbox is prepared.
- *Job Submission*: interaction with Resource Broker,
  Workload Management System and proxy services. The main activity are the job submission to sites that matches the user requirements.
- *Job Status*: check the status of the jobs using BOSS mechanism [7].
- *Job Output*: retrieval of the job output from the grid (output sandbox); also the output can be transferred to a nearby Storage Element or to another one specified by the user.

Other functionalities provided by the tool are the possibility to get information about the causes that made jobs to be aborted (considering both the CMS framework and the grid middlewares), to kill them and to resubmit the failed jobs.

A schematic representation for the standard CRAB workflow is reported in figure 2.

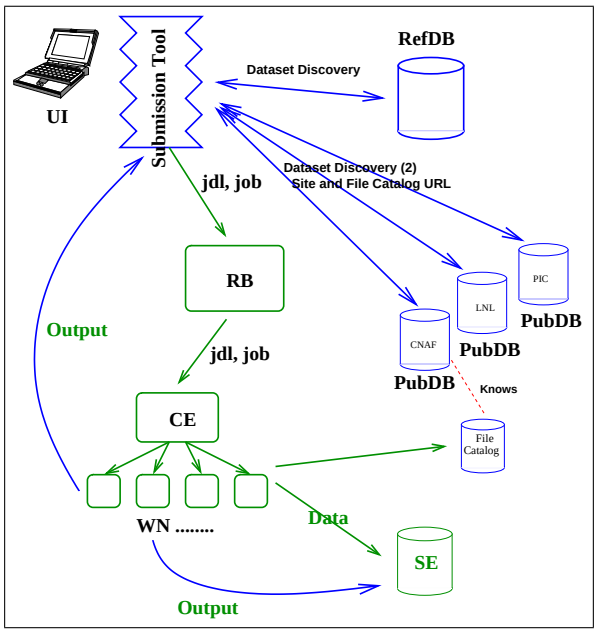

Figure 2: CRAB Workflow w.r.t. middleware workload management services and CMS data management tools.

The user interacts with CRAB via a simple configuration file called crab.cfg. Along with user specific parameters, like the dataset/owner pair that the user wants to access, the specific CMS software version he wants to use, the executable developed to perform a given analysis and the total number of events to access, the configuration file contains values to instruct CRAB on how to handle the specific CMSSW job type and how to interact with Grid middlewares and its submission layer, that is BOSS. The user can also specifies a different wrapper for his executable, in case he wants specific controls which are not foreseen on the standard one. Once CRAB has

read the configuration file, it starts querying CMS specific services which are designed to return the list of sites where data are physically located. These aforementioned services are DBS (Data Bookkeeping Service) and DLS (Data Location Service). DBS is responsible to tell CRAB which data are available, along with specific data parameters describing the particular set of events, while DLS is responsible to localize logical data names w.r.t. the host and the local protocol suitable to access them. The final logical-to-physical filename mapping will be performed on the processing site using logical file catalog technologies.

The script generator engine collects all the information and starts creating jobs, splitting the payload according to the number of events requested by the user and the total number of events available for the given dataset. It also generates the JDL files specifying the list of sites hosting data retrieved from DBS and DLS. Before passing the control for job submission to BOSS, CRAB checks that at least one site fulfills the requirements put into the JDL files. This avoids submitting jobs to sites which, for some reasons, don't have the required CMS software version or have been removed from the Information System due to scheduled downtime or Grid middleware problems. The control is then passed to BOSS, which performs job submission to the Grid. Currently CRAB is able to submit jobs, via BOSS, both to WLCG and OSG. After jobs submission the user has to check only the status to tell CRAB to retrieve the output once they finish. The user can decide, via the crab.cfg file, how to manage output files. He can copy its output directly on the Storage Element close to the Worker Node which run the job, store them on a chosen Storage Element or have it back on the User Interface from which he runs CRAB. CRAB provides also a postmortem functionality to trace back the Grid history of all jobs in order to collect information useful for debugging purposes.

3. CRAB usage statistics

CRAB has been used with success for more than two years by CMS physicists, to perform data analysis. The first intensive usage of the tool by a large number of users from different places, was during the Spring 2006, for the Physics TDR [8] preparation. Moreover, CRAB has been used to access data during the CMS data challenges. The last one was CSA06, when millions of simulated events were analyzed, reaching peaks of 100k submitted jobs per month. The submission daily rate for the CSA06 challenge is reported in figure 3. In this plots each color represents a different site where the jobs run. It is important to notice that CRAB reached submission peaks of more than 40k jobs/day. During the Magnet Test Cosmic Challenge (MTCC), CRAB was also successfully used to access the real data distributed among several sites, the first real data for CMS. There are about 300 CRAB users distributed around the world, which means a daily rate of submitted jobs which reaches peaks of 10k jobs. The first part of 2007 highlights an even wider use of CRAB, with the top five users that have submitted more than 200k jobs. For the same period the daily rate is reported in figure 3.

4. CRAB evolution: toward a Grid-enabled server

The present workflow of CRAB is based on a direct submission from the UI, where the user is working, to LCG and OSG via RB. This standalone model has the advantage of simplicity, but

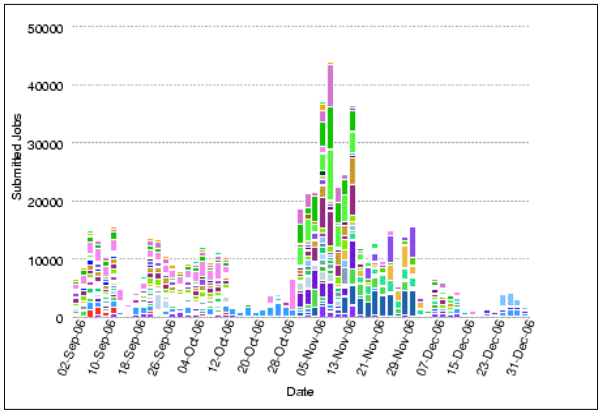

Figure 3: CRAB usage statistics during the CSA06 challenge.

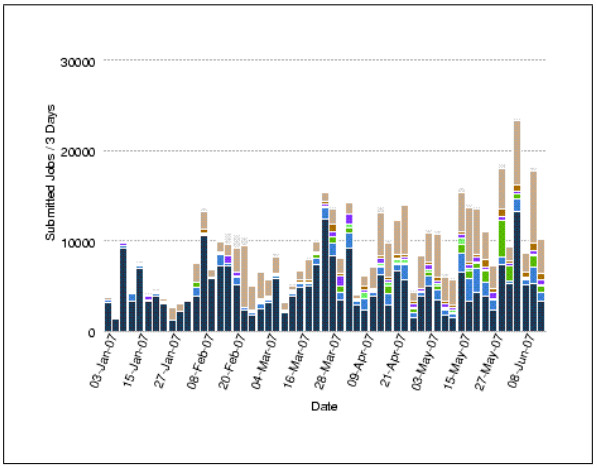

Figure 4: CRAB submission rate during the first half of 2007.

it lacks some features, which can be provided by a more advanced architecture like a Grid-server. In this kind of architectural choice a server is placed between the user and the Grid. In this way the server provides higher level of automation and latency hiding during the analysis process. The main goals of the client-server architecture is to face as much as possible the whole analysis work flow problems and to improve the scalability and the throughput of the analysis process itself. The aim of the project is also to create a tool which is easy to use for physicists and easy to maintain for administrators. Following the CRAB philosophy the client-server implementation is transparent to the end users: the interface, the installation, the configuration procedure and the usage steps are exactly the same as for the standalone version. From the global architectural point of view the adoption of the Grid-enabled server makes the previously described analysis workflow change as reported in figure 4. In the new scenario the role of the client is to interact with DBL/DLS for the data discovery, to prepare the jobs reading the local environment and finally to send the user proxy and the prepared task to the server. It is the server that actually manages the project: it interacts with

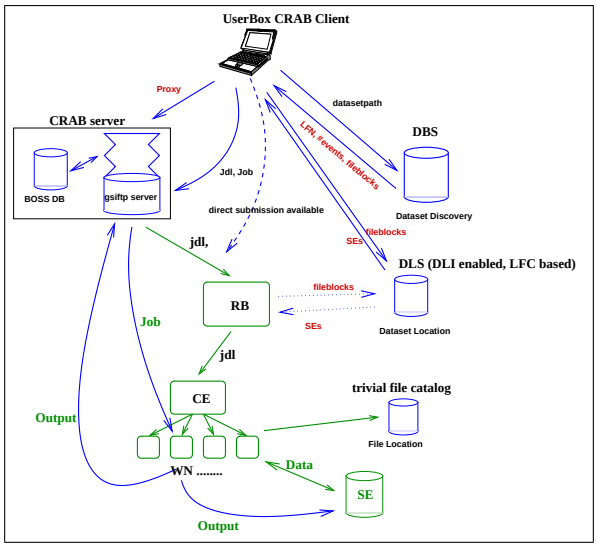

Figure 5: CRAB workflow after the adoption of the CRAB server.

the Grid from the job submission up to the output retrieval. Each interaction must be performed using the user credentials to preserve the CMS Virtual Organization security constraints.

From an architectural point of view, the server is based on components implemented as independent agents. This multi agent system communicates through an environment than provides only an asynchronous and persistent blackboard message service. This messaging service is used to realize the publish and subscribe model that the server agents adopt for interprocess communications. The core of the messaging service is built on the top of MySQL database [9]. The chosen multi agent approach has been chosen as similar as possible to the CMS production system, see [10]. The common architectural model allows an easier maintenance of the Work Load Management tools. The other external softwares adopted by the server are the following: a gridftp server and a WS-Delegation compliant service. The gridftp server is used as the channel for all the communications from client to server. The WS-Delegation service performs the user proxy delegation and it is deployed as a CGI-gSoap [11] standalone server.

The current CRAB server implementation includes the following components:

- **DropBoxGuardian**: this component is the entry point for all the tasks and the directives coming from the clients. Its main role is to dispatch incoming work to the proper agents. It monitors the gridftp and the WS-Delegation repositories in a spool server fashion. This check is performed periodically and the items arriving are signaled to the ProxyTarAssociatior or to the CommandManager components. Actually DropBoxGuardian is the only synchronous component in the submission workflow for the CRAB server.

- **ProxyTarAssociator**: this agent is triggered when a new user task arrives. It creates the data structures and updates the proper configuration files to let the tasks be managed properly by the server environment. Finally, it associates a proxy to a proper task crosschecking the file ownerships.

- **CrabWorker**: performs the submission process to the grid using the same submitter modules of CRAB (i.e., EDG, gLite, gLite-bulk and condor-g). From the architectural point of view the CrabWorker is a multi-component system. Its role as agent is used to perform interprocess communications with the other components however, internally it is organized as a multi-threaded system with producer-consumer synchronization mechanisms. The messages coming from ProxyTarAssociator are intercepted by a front-end thread that enqueue the submission requests. The requests are elaborated from a pool of submission threads that interact with the CRAB code. The submission results are enqueued again. The result queue is managed by a notification thread which publish messages either to retry the submission process (introducing a feedback in the component) or to inform the monitoring components to take care of the submitted analysis.

- **TaskTracking**: plays a vital role for the collection of the information about the advancements of the user tasks. It acts as a coordinator, using the messages emitted by the other components, to keep updated the status of every task. Moreover, it prepares the data structures needed to inform the user both when a task fails and it reaches a final state.

- **JobTracking**: this component is inherited by the CMS production systems. It polls periodically the grid to update the status of every single submitted job. With the collected data it keeps aligned the TaskTracking component and the failure recovery components. The JobTracking is somehow similar to the TaskTracking but the two agents operate at different level of granularity w.r.t. the user analysis. An additional feature provided by the component is the automatic job output file movement from the WN to the server storage.

- **ErrorHandler**: the ErrorHandler component interact tightly with the JobTracking to parse the job status and the job exit codes. The extracted information are then used to drive the single job resubmission (in case of failure) or to trigger the automatic output file retrieval from the worker nodes.

- **JobSubmitter**: the JobSubmitter is the component summoned by the ErrorHandler when a failed job is resubmitted. It performs a very basic, low performance resubmission w.r.t. the CrabWorker. Both the ErrorHandler and the JobSubmitter are inherited from the production tool.

- **CommandManager**: it is used as a front-end for all the infrequent features that the server exposes. The component philosophy is to act as a message router for the other agents. On the current implementation it treats the job interruption only. However, the architectural design grants an easy extension for new features.

- **JobKiller**: this component is driven by the CommandManager to kill whole tasks previously submitted to the grid. It is also able to stop single jobs. It is implemented as a plug-in system: in this way many different meta-scheduler and middleware can be supported (similarly to CRAB grid transparency).

- **Notification**: it receives directives from the TaskTracking and notifies the user by an e-mail when his tasks is ended and the output can be retrieved by the user. Moreover, it sends e-

mails in case of task submission failure (or partial successful submission) and in all the other situations when a direct user intervention is foreseen by the workflow.

- **RSSFeeder**: is the main tool for the CRAB server administrators to monitor the logging of the whole system. The feed are posted on a web page. In this way agents behavior and possible related problems can be easily tracked.

- **CrabWorkerWatchdog**: this optional component provides a very primitive transactional behavior to the whole submission chain. It registers to the messages published by the major players acting on the CRAB server: the CrabWorker and the TaskTracking. In case of catastrophic crash the component will try to recover all those submissions in a non-consistent status. This process is synchronized with the CrabWorker boot.

In addition to the described agents some new features will be introduced in the next server releases. In particular, a web interface to track job and task status and a new agent to monitor the disk server occupancy (with simple garbage collection) will be integrated.

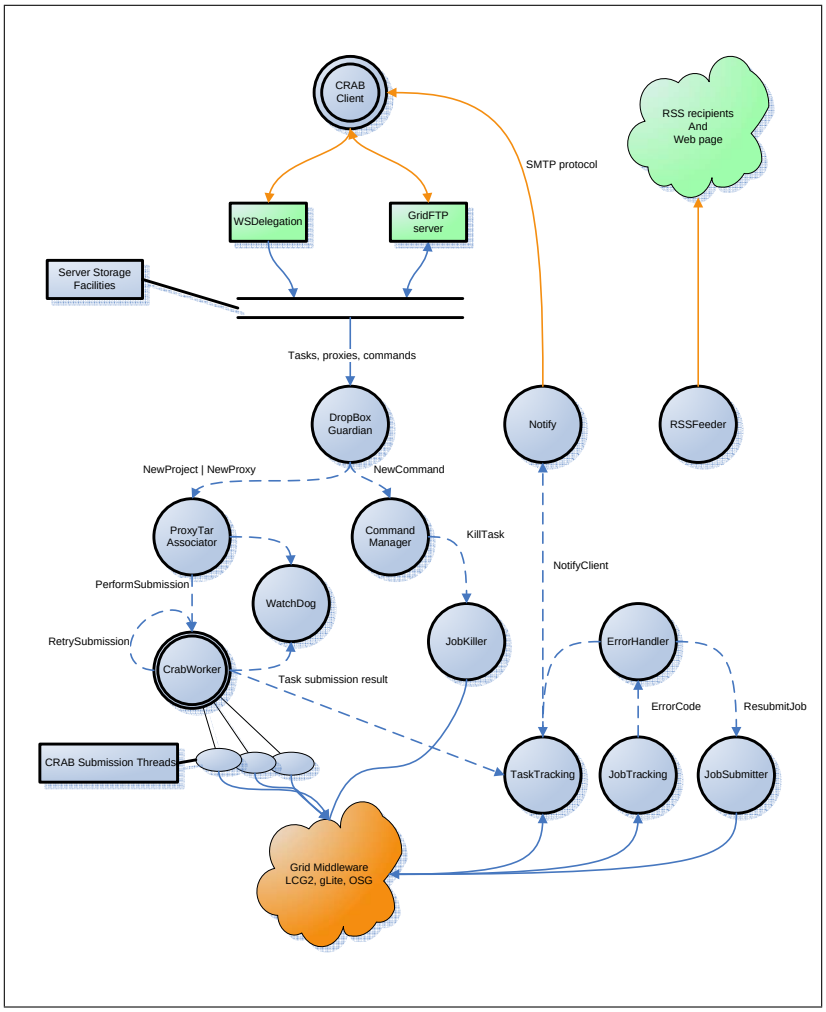

Figure 6: Dataflow diagram of the server agents

In figure 4 the whole server dataflow diagram is reported.
The solid lines describe direct/synchronous interactions and inclusion relations, the dashed lines describe the interactions mediated by the messaging system. The exchanged message topics are indicated also.

5. First server results, future work and conclusions

At the present time three instances of the CRAB server have been deployed. One for development activities and two dedicated to beta-testing community (including now about 5 people, but the number is increasing rapidly). The performances of the whole system are very promising both for the feedback given by the early users and for the evaluated stress test metrics. In particular, the server is able to receive up to 6k tasks in 2h and it is able to submit to the grid up to 7k jobs per day using EDG/condor_g brokers. With gLite a speedup of 3 has been observed, more tests have to be performed on the gLite bulk-submission speedup. The mean time to prepare a task for the CrabWorker component is 200ms approximately.
The measured success ratio for the submission workflow was 99.61% for the very first code release. This meant that about 4 tasks over 1000 failed for reasons connected to the server code. The current implementation fixed many related bugs and the success ratio has been further improved: the quantitative measures are scheduled in the immediate future. The current implementation is particularly light in terms of CPU and network consumption. The disks occupancy will be largely improved with the adoption of the under-development new components. Also the memory consumption is going to be improved in the next server releases.

The major efforts for CRAB and for CRAB server will be devoted to the preparation of the next CMS CSA07 challenge. In fact, this challenge will be the last one before the real data acquisition and to grant the nominal required throughput for the server will be the CRAB team major duty for the next months.

References

[1] The CMS experiment http://cmsdoc.cern.ch

[2] The Large Hadron Collider Conceptual Design Report CERN/AC/95-05

[3] LCG Project: http://lcg.web.cern.ch/LCG

[4] LCG Technical Design Report,CERN-TDR-01 CERN-LHCC-2005-024, June 2005

[5] OSG Project, http://www.opensciencegrid.org

[6] The Computing Project CERN/LHCC 2005-023 CMS TDR 7

[7] Batch Object Submission System, http://boss.bo.infn.it/

[8] Physics Technical Design Reports, CERN/LHCC/2006-021

[9] MySQL Community Database http://www.mysql.com

[10] Production Agent, https://twiki.cern.ch/twiki/bin/view/CMS/ProdAgent

[11] gSoap web service tool, http://gsoap2.sourceforge.net