# CRAB: an Application for Distributed Scientific Analysis in Grid Projects

D. Spiga\(^{1-3-6}\), S. Lacaprara\(^2\), M. Cinquilli\(^3\), G. Codispoti\(^4\), M. Corvo\(^5\), A. Fanfani\(^4\), A. Fanzago\(^5-6\), F. Farina\(^6-7\), C. Kavka\(^8\), V. Miccio\(^6\), and E. Vaandering\(^9\)

\(^1\)University of Perugia, Perugia, Italy  
\(^2\)INFN Legnaro, Padova, Italy  
\(^3\)INFN Perugia, Perugia, Italy  
\(^4\)INFN and University of Bologna, Bologna, Italy  
\(^5\)CNAF, Bologna, Italy  
\(^6\)CERN, Geneva, Switzerland  
\(^7\)INFN Milano-Bicocca, Milan, Italy  
\(^8\)INFN Trieste, Trieste, Italy  
\(^9\)FNAL, Batavia, Illinois, USA
Abstract - Starting from 2008 the CMS experiment will produce several Petabytes of data each year, to be distributed over many computing centers located in many different countries. The CMS computing model defines how the data has to be distributed in a way that CMS physicists can efficiently run their analysis over the data. CRAB (CMS Remote Analysis Builder) is the tool, designed and developed by the CMS collaboration, that facilitates the access to the distributed data in a totally transparent way. The tool's main feature is the possibility to distribute and to parallelize the local CMS batch data analysis processes over different Grid environments without any specific knowledge of the underlying computational infrastructures. More specifically CRAB interacts with both the local user environment, with CMS Data Management services and with the Grid middleware.

Keywords: Grid Computing, Distributed Computing, Grid Application, High Energy Physics Computing.

Introduction

The Compact Muon Solenoid (CMS)[1] is one of two large general-purpose particle physics detectors integrated in to the proton-proton Large Hadron Collider (LHC)[2] at CERN (European Organization for Nuclear Research) in Switzerland. The CMS detector has 15 millions of channels; through them data will be taken at a rate of TB/s and then selected by 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 2 PB data per year. This challenging experiment is a collaboration constituted by about 2600 physicist from 180 scientific institutes all over the world. The big quantity of data to analyze (but also to simulate) needs a plenty of resources to satisfy the computational experiment requirements, as the large disk space needed to store the data and many cpus where to run physicist's algorithms. It is also needed a way to make all data and shared resources accessible from all the people in the collaboration. This environment has encouraged distributed resources from a facility infrastructure point of view and to satisfy these problematics, CMS has defined an ad-hoc computing model. This leans on the use of Grid computing resources, services and toolkits as basic building blocks, making realistic requirements on grid services. In particular CMS decided to use a combination of generic Grid tools, provided by the LCG (LHC Computing Grid)[3] and OSG (Open Science Grid)[4] projects, as well as specialized CMS tools are used together. In this environment, the computing system has been arranged in tiers (Figure 1) where the majority of computing resources are located away from the host lab. The system is geographically distributed, consistent with the

![Figure 1. Multi-tier architecture based on distributed resources and Grid services.](image-url)
nature of the CMS collaboration itself:

- The Tier-0 computing centre is located at CERN and is directly connected to the experiment for the initial processing, reconstruction and data archiving.
- A set of large Tier-1 centers where the Tier-0 distributes data (processed and not); these centers provide also considerable services for different kind of data reprocessing.
- A typical Tier-1 site distributes the processed data to smaller Tier-2 centers which have powerful cpu resources to run analysis tasks and Monte Carlo simulations.

**Distributed analysis model in CMS**

For the CMS experiment in the Grid computing environment there are many problematic points due to the wide amount of dislocated resources (intended as data and hardware). The CMS Workflow Management System (CMS WM) manages the large scale data processing and reduction process which is the principal focus of experimental HEP computing. Just for this the CMS WM has to be considered the main flow to manage and access the data, interacting with the Grid middleware and giving to the user an unique interface that allows to interact with the generic Grid services and with the specific experiment services as an unique common environment. The Grid services are mainly constituted by the Grid WMS which takes jobs into account, performs match-making operations and distributes the jobs toward the Computing Element (CE); the CE manages local queues that point to a set of resources located to a specific site as the Worker Nodes where the jobs run; finally there are Storage Elements (SE) that are logic entities that warranty an uniform access to a storage area where the data is stored. As part of its computing model, CMS has chosen a baseline in which the bulk experiment-wide data is pre-located at sites and thus the Workload Management (WM) system submit the jobs to the correct CE. The Dataset Bookkeeping System (DBS) allows to discover, access and transfer various forms of event data in a distributed computing environment. The analysis model[5] is batch-like and consists of main steps: the user runs interactively on small samples of the data somewhere in order to develop his code and test it; once the code is the desired one, the user selects a large data and submit the very same code to analyze the chosen data. The results is made available to the user to be analyzed interactively. The analysis can be done in step, saving the intermediate ones and iterating over the latest ones. The distributed analysis workflow over the Grid backs on the Workflow Management System, which is not directly user oriented. In fact the analysis flow in the above specified distributed environment is a more complex computing task because it assume to know which data are available, where data are stored and how to access them, which resources are available and are able to comply with analysis requirements, also at the above specified Grid and CMS infrastructure details.
The CMS Remote Analysis Builder

The users do not want to deal with the previously described issues and they want to analyze data in a simple way. The CMS Remote Analysis Builder (CRAB)[6] is the application designed and deployed by the CMS collaboration that, following the CMS WM, allows a transparent access to distributed resources over the Grid to end physicists. CRAB perform three main operations:

- Interaction with the CMS analysis framework (CMSSW) used by the users to develop their applications that runs over the data;
- The Data Discovery step, interacting with the CMS data management infrastructure, when required data is found and located;
- The Grid specific steps: from submission to output retrieval are fully handled.

The typical workflow (Figure 2) involves the concept of task and job. The task corresponds to the high-level objective of an user (run an analysis over a defined data). 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 by many jobs. A typical analysis workflow in this contest consists of:

- the data discovery to determine the Storage Elements (SE) of sites storing data, interacting with DBS;
- the preparation of the input sand-box: a package containing the user application, with the relative libraries and data files;
- the job preparation, which creates a wrapper over the user executable; it prepares the environment where the user application has to run (WN level) and at the end of this handles the output produced;
- the job splitting which takes in to account the specific data information, the data distribution and the coarseness requested by the user;
- the Grid job configuration that consists of a file filled using the Job Description Language (JDL) which is interpreted by the WMS and which contains the job requirements;
- the task (jobs) submission to the Grid;
- the monitoring of the submitted jobs which involves the WMS and that allows to check the status of a job and its progress;
• when a job is finished from the Grid point of view, the last operation is the output retrieval which allows to handle the job output (which can also include the copy of the job output to a Storage Element) through the output-sandbox.

CRAB is used by the user on the User Interface (UI), which is the access point to the Grid and where is available the client of the middleware. The user interacts with CRAB via a simple configuration file divided into main sections and then by CLI. In the configuration file there are all the specific parameters of the task and the jobs; after the user has developed and tested locally his own analysis code, he specifies which application he wishes to run, the dataset to analyze and the general requirements on the input dataset as the job splitting parameter, the information related to how threat the output that can be retrieved back to the UI or can be directly copied to an existing Storage Element. There are also post-output retrieval operations that can be executed by the users, which include the data publication that allows to register user data into a local DBS instance, to consent an easy access on the user-registered data.

**CRAB Architecture and Implementation**

The programming language used to develop CRAB is Python[7]. It allows a reduced development time and an improved maintenance, included the fact that does not need to be compiled. CRAB can perform three main kinds of job submission where each one is totally transparent to the user:

• the direct submission to the Grid interacting directly with the middleware;
• the direct submission to local resources and relative queues, using the batch system, in a LFS (Load Sharing Facilities) environment;
• the submission to the Grid using the CRAB server as a layer where to delegate the interaction with the middleware and the task/job management.

Actually the major effort of the development activity is devoted to the client-server implementation. The client is on the User Interface, while the server could be located somewhere over the Grid. The CRAB client is directly used by the users and it enables to perform the operations involved in the task/job preparation and creation, as: the data discovery, the input-sandbox preparation, the job preparation (included the job splitting) and the requirement definition. Then the client makes a request, completely transparent to the user, to the CRAB server. This one fulfills each request, handling the task and performing the related flow, in any kind of Grid interaction:

• job submission to the Grid;
• automatic job monitoring;
• automatic job output retrieval;
• re-submission of failed jobs, following particular rules for different kinds of job failures;
• every specific command requested by the user as the job killing;
• notify the user by e-mail when the task reach a specified level and when it is fully ended (the output results of each jobs are ready).

This operation partitioning between the client and the server allows to automatize as much as possible the interaction with the Grid reducing the unnecessary human load, having all possible actions into the server side (at minimum those on client side), centralizing the Grid interaction and then allowing to handle every kind of trouble on a unique place. This also permits to improve the scalability of the whole system.

The communication between the client and the server is on SOAP[8]. Selecting it has some obvious reasons: it is a de facto standard in the Grid service development community, it uses HTTP protocol, provides interoperability across institutional and application language. The client has to assume nothing about the implementation details of the server and vice versa. In this case the SOAP based communication is developed by using gSOAP[9]. gSOAP provides a cross-platform development toolkit for developing server and client applications, allowing to not maintain any custom protocol. It does not require any pre-installed runtime environment, but using the WSDL (Web Services Description Language) it generates code in ANSI C/C++.

The internal CRAB server architecture (Figure 3) is based on components implemented as independent agents communicating through an asynchronous and persistent message service (as a publish and subscribe model)[9] based on a MySQL[10] database. Each agent takes charge of specific operations, allowing a modular approach from a logical point of view. The actual server implementation provides the following components:

• **CommandManager**: endpoint of SOAP service that handles commands sent by the client;
• **CrabWorker**: component that performs direct job submission to the Grid;
• **TaskTracking**: updates information about tasks under execution polling the database;
• **Notification**: notifies the user by e-mail when his task is ended and the output has been already retrieved; it also notify the server administrator for special warning situation;
• **TaskLifeManager**: manages the task life on the server, cleaning ended tasks;
• **JobTracking**: tracks the status of every job;
• **GetOutput**: retrieves the output of ended jobs;

![Figure 3. CRAB Server Architecture](image-url)
- **JobKiller**: when asked kills single or multiple jobs;
- **ErrorHandler**: performs a basic error handling that allows to resubmit jobs;
- **RSSFeeder**: provides RSS channels to forward information about the server status;
- **AdminComponent**: executes specific server maintenance operations.

Many of the above listed components are implemented following a multithreading approach, using safe connection to the database. This allows to manage many tasks at the same time shortening and often totally removing the delay time for an single operation that has to be accomplished on many tasks. The use of threaded components is very important when interacting with the Grid middleware, where some operation (e.g.: on a bulk of jobs at the same moment) requires a not unimportant time.

Two important entities in the CRAB server architecture are the WS-Delegation and a specific area on an existing Storage Element. The WS-Delegation is a compliant service for the user proxy delegation from the client to the server; this allows to the server to perform each Grid operation for a given task with the corresponding user proxy. The SE allows to transfer the input/output-sandboxes between the User Interface and the Grid, working as a sort of drop-box area. The server has 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 remote area, independently from the transfer protocol. The ability of the server to potentially interact with any associated storage server, independently from the protocol, allows to have a portable and scalable object, where the Storage Element that hosts the job sand-boxes is completely independent from the server. Then the CRAB server is then really adaptable to different environments and configurations. It is also complained to have a local disk area mounted on the local CRAB server instance, with a GridFTP server associated with, to be used for the sandbox transfers (instead of a remote Storage Element).

The interaction with the Grid is performed using the **BossLite** framework included in the server core. This framework can be considered a thin layer between the CRAB server and the Grid, used to interact with the middleware and to maintain specific information about tasks/jobs. BossLite is constituted by a set of API that works as an interface to the core. The core of BossLite maps the database objects (e.g.: task, job) and allows to execute specific Grid operations over database-loaded objects.

**Conclusions**

CRAB is in production since about three years and is the only computing tool in CMS used by generic physicist. As it shown in Figure 4 it is widely used inside the collaboration with more then 600 distinct users from February 2007 till January 2008 over about 50 distinct...
Tier-2 sites involved in the Grid analysis activities. When real data will be available the expected
daily rate of submitted jobs will be of about 100000 jobs per day. The CRAB tool is being
continuously evolving and the actual architecture allows to simply add new components to the
structure to follow the analysis requirements and to support new use cases that will come up.

References

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

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


4. The Open Science Grid project http://www.opensciencegrid.org


NOTES IN COMPUTER SCIENCE. High Performance Computing -HiPC 2007 14th International Conference.
Goa, India. 18-21 Dicembre 2007. vol. 4873, pp. 580-586

C.Kavka, M.Merlo, L.Servoli, A.Fanfani. (2007). "CRAB: the CMS distributed analysis tool development and
design". NUCLEAR PHYSICS B-PROCEEDINGS SUPPLEMENTS. Hadron Collider Physics Symposium 07. La
Biodola, Isola d'Elba, Italy. 20-26 Maggio 2007. vol.177-178C, pp. 267-268


9. SOAP Messaging Framework http://www.w3.org/TR/soap12-part1

10. The gSOAP Project http://www.cs.fsu.edu/~engelen/soap.html

11. F.Farina, S.Lacaprapa, W.Bacchi, M.Cinquilli, G.Codispoti, M.Corvo, A.Dorigo, A.Fanfani, F.Fanzago,
OF SCIENCE. XI International Workshopon Advanced Computing and Analysis Techniques in Physics Research