

PAPER • OPEN ACCESS

## Efficient monitoring of CRAB jobs at CMS

To cite this article: J M D Silva *et al* 2017 *J. Phys.: Conf. Ser.* **898** 092036

View the [article online](#) for updates and enhancements.

### Related content

- [Distributed analysis in ATLAS](#)  
A. Dewhurst and F. Legger
- [Using ssh as portal – The CMS CRAB over glideinWMS experience](#)  
S Belforte, I Sfiligoi, J Letts et al.
- [Recent Standard Model results from CMS](#)  
Simon de Visscher and CMS collaboration

# Efficient monitoring of CRAB jobs at CMS

J M D Silva<sup>1</sup>, J Balcas<sup>2</sup>, S Belforte<sup>3</sup>, D Ciangottini<sup>4</sup>, M Mascheroni<sup>5</sup>,  
E A Rupeika<sup>6</sup>, T T Ivanov<sup>7</sup>, J M Hernandez<sup>8</sup>, E Vaandering<sup>5</sup>

<sup>1</sup>Universidade Estadual Paulista, Sao Paulo, Brazil

<sup>2</sup>California Institute of Technology, Pasadena, CA, USA

<sup>3</sup>INFN Sezione di Trieste, 34127 Trieste, Italy

<sup>4</sup>INFN Sezione di Perugia, 06123 Perugia, Italy

<sup>5</sup>Fermi National Accelerator Laboratory, Batavia, IL, USA

<sup>6</sup>Vilnius University, Lithuania

<sup>7</sup>University of Sofia "St. Kliment Ohridski", Bulgaria

<sup>8</sup>CIEMAT, Madrid, Spain

E-mail: marco.mascheroni@cern.ch

**Abstract.** CRAB is a tool used for distributed analysis of CMS data. Users can submit sets of jobs with similar requirements (tasks) with a single request. CRAB uses a client-server architecture, where a lightweight client, a server, and ancillary services work together and are maintained by CMS operators at CERN.

As with most complex software, good monitoring tools are crucial for efficient use and long-term maintainability. This work gives an overview of the monitoring tools developed to ensure the CRAB server and infrastructure are functional, help operators debug user problems, and minimize overhead and operating cost. This work also illustrates the design choices and gives a report on our experience with the tools we developed and the external ones we used.

## 1. Introduction

CRAB [1] is a tool used by more than 1,500 users worldwide for distributed analysis of CMS data in the high-throughput infrastructure of the Worldwide LHC Computing Grid (WLCG [2]). At any given time CRAB has processing requests from about 500 different users that submit sets of Grid jobs with similar requirements (tasks) with a single user request. The typical number of active tasks in CRAB ranges from a few hundred to a few thousand tasks. The number of jobs in each task can vary between a few dozen and a few thousand, being the most common values in the 100-2000 jobs range.

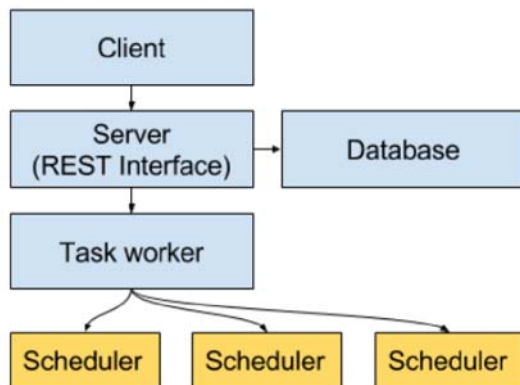
CRAB uses a client-server architecture, where a lightweight client, a server, and ancillary services work together and are maintained by CMS operators at CERN.

### 1.1. CRAB Architecture

The CRAB Server provides a REST Interface for the communication with the other components of the CMS Computing System. Users interact with it via a light client for submission, monitoring and control of their analysis tasks. The TaskWorker is a queue system responsible for processing user's tasks, expanding user-provided configurations into lists of jobs that are submitted to HTCondor [3] schedulers for execution on the Grid. Information about tasks is



stored in an Oracle-based database. The High level view of the CRAB architecture is shown in Figure 1.



**Figure 1.** CRAB Server Architecture.



**Figure 2.** Number of Tasks submitted to CRAB every week, from initial rollout to fall 2016.

### 1.2. Monitoring goals

The scope of this paper is to describe the monitoring tools used by developers and operators of the CRAB service to make sure that the system is performing as expected and to troubleshoot possible problems. Users have an alternative set of monitoring tools to check their submissions, but with  $O(1k)$  user tasks being processed inside CRAB at any given time, operators need more aggregate views which still let them drill down to details at the single job level if needed. One of CRAB operator's day-to-day responsibilities is debugging user tasks (upon user request). More than 8000 tasks are currently submitted every week as shown in Figure 2.

The main goals in developing operator monitoring tools were simplicity, functionality and the connection to more detailed, already existing, monitoring.

## 2. Monitoring tools

Two complementary tools have been developed for CRAB server operation monitoring, a simple Kibana [4] dashboard (CRAB dashboard) to visualize the status of the server and rates of work in and out of the pipelines, and a task-oriented page (CRABMon) to allow quick access to details of any given task.

### 2.1. CRABMon

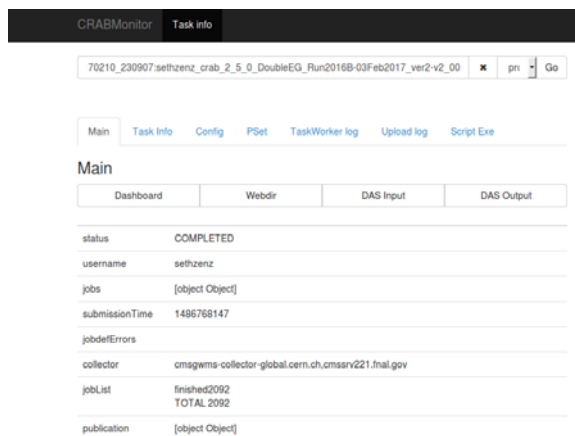
CRABMon is a dedicated javascript-based monitoring page, developed within CMS collaboration and dedicated to monitoring system operation. It gathers the results from multiple CRAB APIs exposed via a REST interface. It is used by CRAB operators to monitor the details of submitted jobs, task status, configuration and parameters, user code, and log files. Links to relevant data are provided when a problem must be investigated. The software is largely javascript developed in-house to maximize flexibility and maintainability, although jQuery is also utilized.

CRABMon is available to every CMS user via the central CMS web portal (CMSWeb) which provides secure authentication based on grid certificates. The CMSWeb portal also provides us with http proxy capability allowing CRABMon to offer to experts direct access to relevant log files locally stored on the TaskWorker and/or the schedulers.

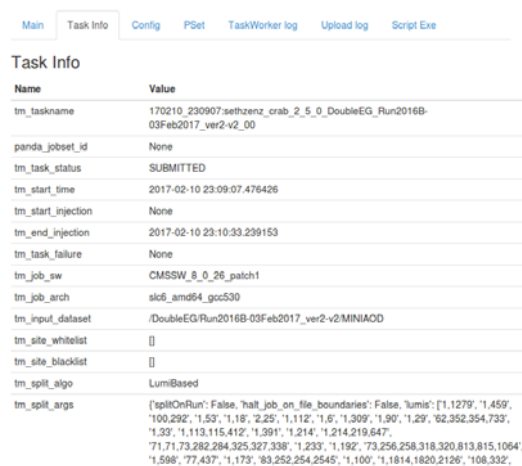
We intentionally kept the CRABMon features limited in order to have a quick and agile tool. The list of what can be accessed with one mouse click is:

- a summary and a full breakdown of the task information in the task database, avoiding the need to write SQL queries by hand,
- quick links to the task configuration used in the submission and the log from the client side,
- a detailed view of all task jobs in the CMS Job Dashboard [6], [7] which gives access also to job log and details of the FTS transfer used to move job output to the user home site,
- links to the view of the input and output datasets in the CMS dataset bookkeeping database.

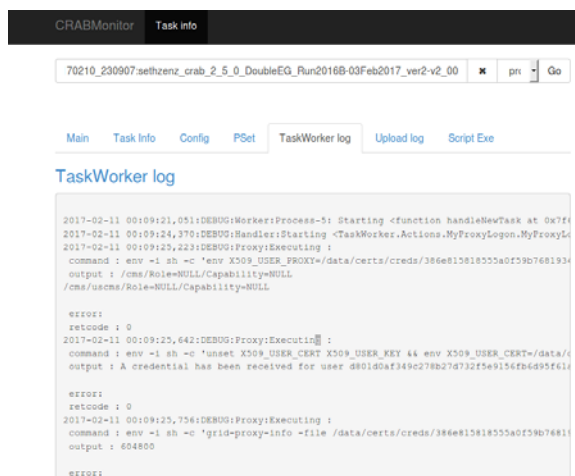
Figures 3, 4, 5 and 6 provide examples of typical CRABMon views used in daily work.



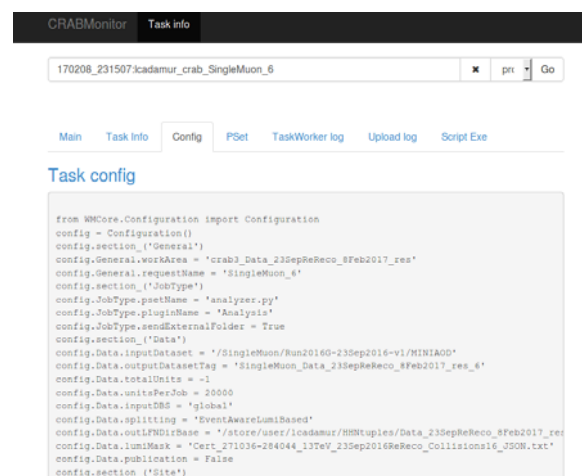
**Figure 3.** CRABMon main view providing some general information about a task (such as task and job statuses) and links to external services to complement available information.



**Figure 4.** CRABMon provides an easy way to access the information stored in the task database (helps avoid writing SQL queries by hand).



**Figure 5.** Task worker logs for a specific task. These logs allow debugging problems that happen during task submission.



**Figure 6.** Access to user configuration files (client configuration, analysis framework configuration, ancillary files).

Over time, CRABMon has proved itself to be extremely effective in enabling operators to quickly troubleshoot user problems. The current CRAB client prints on the user screen the URL for the CRABMon view so that they can use it in reporting issues.

## 2.2. CRAB dashboard

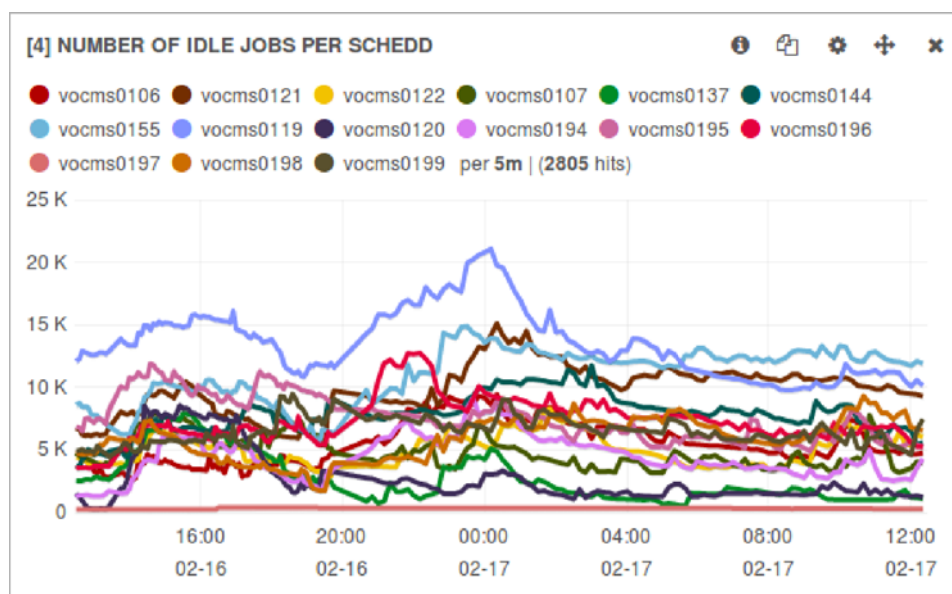
We collected in one web page a set of Kibana views deployed on the CERN Kibana instance that helps CRAB operators monitor the core components of the infrastructure (Task worker, schedulers, etc.) in real time. A simple crontab runs on the TaskWorker every 5min collecting relevant metrics via a few queries to the Task database (via CMSWeb and CRABServer API) and a few HTCondor *condor\_q* commands. Those metrics are formatted in a JSON file which is pushed via *curl* to the CERN monitoring infrastructure where they can be used as targets of Kibana queries. We used Kibana version 3 in this work.

For monitoring the overall performance of the CRAB Server we decided to display two kind of metrics:

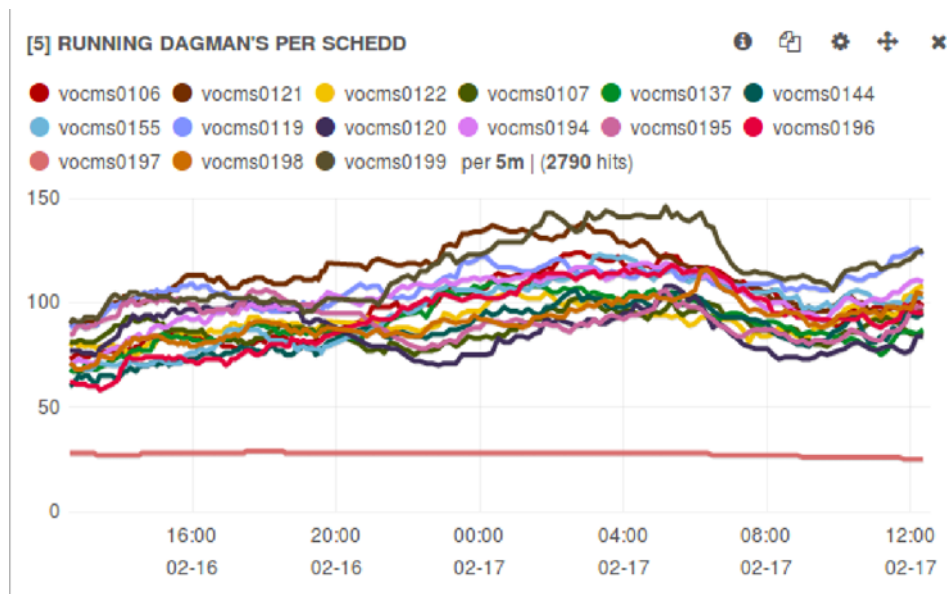
- "how many" metrics: time series of snapshots (like in Figures 7, 8 and 10) which indicate how much load the system is managing at any given time and how large is the queue of pending work.
- "how fast" metrics: evolving picture of the rate of change of the load inside TaskWorker, i.e. the speed at which new requests come in and get out in one of the possible final states (basically SUBMITTED or FAILED with FAILED broken in various details). One such view is shown in Figure 9.

We found it particularly useful to have the plot of tasks inside the TaskWorker in the so called non-final states (see Figure 10). That allows to quickly spot cases where some small fraction of tasks gets stuck in the internal processing, while simply looking at reported errors in the final output is not useful.

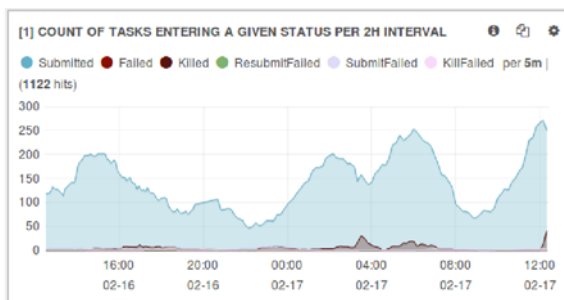
That simple page was found to be perfectly adequate to the need, and over the course of the last twelve months the only changes we made were in the list of HTCondor schedd's to monitor.



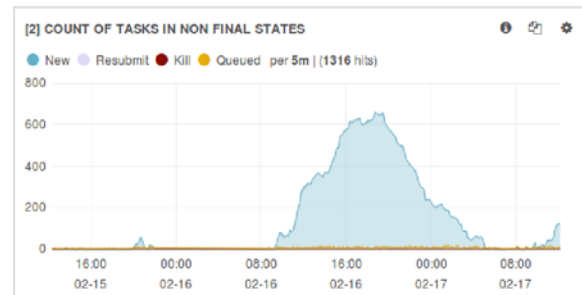
**Figure 7.** Idle jobs in the HTCondor schedd's used by CRAB server. The system was lightly loaded at this time. One schedd is set aside to drain for maintenance



**Figure 8.** Number of active DAGs in each HTCondor schedd. Each task is managed by one such DAG. This picture is important to us to verify proper load sharing across the schedd's since there is a limit to how many DAGs can be allowed on each machine (currently 175).



**Figure 9.** Rates of tasks (running average in last 2h) out of TaskWorker.



**Figure 10.** Count of tasks known in the Task database and currently in a non-final state in TaskWorker state machine.

On the implementation side, the use of Kibana framework turned out to be very cumbersome for our intended use. Changing the page interactively is inconvenient, and we had to resort to a script which produces the JSON view of the dashboard, script that over the time became unmaintainable. While some improvement could be obtained by differently naming the metrics to make it easier to find them in the queries, we also feel that the refresh time of the page on the browser is inefficient when you select multiple days. On the implementation side, the use of Kibana framework turned out to be very cumbersome for our intended use. Changing the page interactively is inconvenient, and we had to resort to a script which produces the JSON view of the dashboard, script that over the time became unmaintainable. While some improvement could be obtained by differently naming the metrics to make it easier to find them in the queries, we also feel that the refresh time of the page on the browser is inefficient when you select multiple days.

For the above reasons we are now starting to move this dashboard to Grafana [5]. The outcome of that work would be the subject of a future report.

### 3. Conclusion

The CRAB server for CMS is a service which manages large and critical user analysis workflows. Timely analysis of CMS data depends on it. We have found that the approach of focusing on overall pipeline performance metrics (rate in, rate out, length of internal queues) is simple to implement, yet effective in prompt detection of problems. The tools described in this work are used in daily operations basis and there is no plan to significantly change them. Only a few metrics will be added or removed as the architecture of CRAB evolves and we try to use friendlier graphic generation packages.

### References

- [1] Cinquilli M et al., 2015 CRAB3: Establishing a new generation of services for distributed analysis at CMS J. Phys. Conf. Ser. 396 032026
- [2] Worldwide LHC Computing Grid, <http://wlcg.web.cern.ch/>
- [3] <http://research.cs.wisc.edu/htcondor/>
- [4] <https://www.elastic.co/products/kibana>
- [5] <http://grafana.org/>
- [6] Karavakis E et al, CMS Dashboard Task Monitoring: a user-centric monitoring view,2010, J. Phys.: Conf. Ser. 219 072038 doi:10.1088/1742-6596/219/7/072038
- [7] Karavakis E et al, User-centric monitoring of the analysis and production activities within the ATLAS and CMS Virtual Organisations using the Experiment Dashboard system, in proceedings of EGI Community Forum 2012 / EMI Second Technical Conference, 2012, PoS(EGICF12-EMITC2)110