Effective HTCondor-based monitoring system for CMS

J Balcas¹, B P Bockelman², J M Da Silva³, J Hernandez⁴, F A Khan⁵, J Letts⁶, M Mascheroni⁷, D A Mason⁷, A Perez-Calero Yzquierdo⁴,⁸, J-R Vlimant¹ for the CMS Collaboration

¹California Institute of Technology
²University of Nebraska-Lincoln
³Universidade Estadual Paulista
⁴Centro de Investigaciones Energeticas, Medioambientales y Tecnologicas
⁵National Center for Physics
⁶University Of California, San Diego
⁷Fermi National Accelerator Laboratory
⁸Port d’Informacio Cientifica

E-mail: justas.balcas@cern.ch

Abstract. The CMS experiment at the LHC relies on HTCondor and glideinWMS as its primary batch and pilot-based Grid provisioning systems, respectively. Given the scale of the global queue in CMS, the operators found it increasingly difficult to monitor the pool to find problems and fix them. The operators had to rely on several different web pages, with several different levels of information, and sift tirelessly through log files in order to monitor the pool completely. Therefore, coming up with a suitable monitoring system was one of the crucial items before the beginning of the LHC Run 2 in order to ensure early detection of issues and to give a good overview of the whole pool. Our new monitoring page utilizes the HTCondor ClassAd information to provide a complete picture of the whole submission infrastructure in CMS. The monitoring page includes useful information from HTCondor schedulers, central managers, the glideinWMS frontend, and factories. It also incorporates information about users and tasks making it easy for operators to provide support and debug issues.

1. Introduction
The Compact Muon Solenoid (CMS) [1] is a generic purpose detector at the Large Hadron Collider (LHC) [2], which delivers collisions of protons and heavy ions at unprecedented energies. The data of this experiment is collected at 1 KHz and is processed in real time at the Tier0 (T0) [4] center using calibration constants provided within 48 hours of data taking. Upon delivery of improved software and calibration, the data is fully or partially reprocessed over the LHC grid [3], composed of tiered computing centers (Tier1, Tier2, Tier3) and opportunistic resources [5]. The analysis of this large dataset requires simulated collisions in a ratio of about 1.3 per collision events. A large amount of heterogeneous computing resources are required for both data reprocessing and simulation production and the management of this activity is a complex task [6]. These resources are shared with users that require computing facilities to further process the CMS data and simulation. The grid jobs for both central production and analysis are eventually handled by HTCondor [7] under the glide-in-wms scheme [8], where pilot jobs are ran in the local batch queues and subsequently from a global pool.
In this paper we present a monitoring system which provides an overview of HTCondor for jobs at given production sites, for production workflows, and other key HTCondor components. First the technical details of implementation are introduced, then the different views are described together with how these resources are used.

2. Implementation
The virtual machine which runs the data aggregation and serves monitoring is hosted in the CERN Openstack and has the following characteristics: 1) 4CPUs, 2) 8GB RAM, 3) one 200GB High IO disk (500 IOPS), 4) five 100GB High IO disks (500 IOPS). The following applications are utilized:

- HTCondor [7] - is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor that places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion. We use the HTCondor Python bindings, which provides access to the client-side APIs for HTCondor and the ClassAd language. These Python bindings use the HTCondor shared libraries. Their use is more efficient than parsing the output of the HTCondor client tools.

- Apache [9] - is a widely used web server software, developed and maintained by the Apache Software Foundation under an open source software license. The Apache mod_wsgi package implements a simple way to use Apache module which can host any Python web application which supports the Python WSGI specification.

- Genshi [10] - is a Python library that provides an integrated set of components for parsing, generating, and processing HTML, XML or other textual content for output generation on the web.

- RRDtool [12] is an open source, industry-standard, high performance data logging and graphing system for time series data. We use the python-rrdtool [11] which delivers the same API as original binding.

- Single sign-on (SSO) [13] - is a session and user authentication service that permits a user to use one set of login credentials (e.g. name and password) to access multiple applications. The service authenticates the end user for all the applications the user has been given rights to and eliminates further prompts when the user switches applications during the same session. On the back end, SSO is helpful for logging user activities as well as monitoring user accounts.

Each view has an independent process which runs every 3 minutes and produces specific output (See more details in Monitoring Elements and section 3 in Figure 1) for REST calls. All aggregated information is saved in an RRD database, on a separate High IO disk. The HTML, CSS and Javascript files are on a separate High IO disk, which is used for graphical interface. Any communication with monitoring has to have a valid SSO cookie from the CERN authorization service and the user has to be authorized to access this website (Currently configured to allow only CERN registered and active users).

2.1. Database
RRDtool (round-robin database tool) aims to handle time series data such as network bandwidth, temperatures or CPU load and is therefore well adapted to our monitoring use-case. The data is stored in a circular buffer based database, thus the system storage footprint remains roughly constant over time. RRDtool assumes time-variable data in intervals of a certain length. This
Figure 1. Detailed overview of \texttt{cms-gwmsmon} backend implementation. 1. User or Application; 2. Communication with Monitoring, HTTPS and SSO based; 3. Apache config through wsgi to read/provide information from the disks; 4. Each independent view cronjob; 5. Minimized example of CMS Global pool.

interval, usually named \textit{step}, is specified upon creation of an RRD file and cannot be changed afterwards. Because data may not always be available at just the right time, RRDtool will automatically interpolate any submitted data to fit its internal time-steps.

The value for a specific step, that has been interpolated, is named a primary data point (PDP). Multiple PDPs may be consolidated according to a consolidation function (CF) to form a consolidated data point (CDP). Typical consolidation functions are average, minimum, maximum and all of these are stored for real time monitoring. After the data has been consolidated, the resulting CDP is stored in a round-robin archive (RRA). A round-robin archive stores a fixed number of CDPs and specifies how many PDPs should be consolidated into one CDP and which CF to use.

After one hour the archive will “wrap around”: the next insertion will overwrite the oldest entry. This behavior in this context is referred to as “round-robin” and is the reason for the program’s name. However this is different from the common computer science definition, which is a method of distributing resources among multiple consumers or processes.

RRD also includes tools to extract round-robin data in a graphical format, for which it was originally intended. To save disk space, \texttt{cms-gwmsmon} compacts the older data using a Consolidation Function (CF), which performs computation on many data points to combine it into a single point over a longer period (1h, 1d, 1w, 1m, 1y). Currently each new value is stored every 3 minutes; in that case, we would only need 175200 data points to store the data for an entire year, but using consolidation, we store only first 480 values for the last day (every 3 minutes interval), for 1d - every hour (24 values), 1w - every hour (168 values), 1m - every hour (5208 values), while for 1y we keep only daily values, which is in total 365 values in same RRD database file. In total, using consolidation for a 1 year each RRD database file will keep 6245 points at maximum, which is 28 times lower than if we keep it at every 3 minutes interval.

3. Data Collection and Monitoring Elements
Each independent view process uses the same function for querying collector or schedulers, but it interprets the data differently. All information which is stored on disk is accessible through the HTTP interface and REST calls. The various available views are described in the following sections.
3.1. Production View
The information is retrieved from all WMAgent [14] schedulers (see Tables 1, 2). All these values are used to fill the RRD database, which stores information about each workflow, subtask, priority status of running and idle, site on which job is running or idle and the requirements for the job. Monitoring the priority profile of what is running and pending over time is a crucial view for production and is shown in Figure 2.

3.2. Analysis View
The information is retrieved from all CRAB3 [15] schedulers and the classAds from each job as reported in Table 1 and 3. Enough information is collected to allow a deeper inspection of users’ tasks as shown in Figure 3, including computing requirements and status. Due to different implementations of WMAgent and CRAB3, some detailed job information (such as upcoming jobs) is not published by WMAgent and limits the scope of the monitoring.

3.3. Total View
Information is queried about all schedulers (see Table 4), including the status they are in. As can be seen in Figure 4, the monitoring is crucial in order to understand at a glimpse if components are running into errors. Several json files containing information from other views are used to prepare the integral of site usage. In addition, information about all available pilots in the
The global pool is collected to determine what each pilot is doing. The pilot’s information provided in Table 5 can be retrieved through REST API.

3.4. Factory View
We are getting the information from the collector and three different factories (UCSD, GOC, CERN) about running pilots at each site. The classads available are reported in Table 6. As Factories are multipurpose and can submit pilots for different projects, we only collect information from CMS frontends: frontend_CMSG-v1_0_cmmspilot, frontend_CMS_T0-Frontend_cmmspilot.

4. Discussion
This paper presented the technical implementation of a powerful monitoring element of the CMS production system and the description of the information that it provides. The system described herein provides useful information for monitoring a large part of the CRAB3 infrastructure and the HTCondor scheduler at a quick glance. In particular, knowing the total number of running analysis jobs helps to determine the general load on the system and the utilization of the grid resources. Furthermore, this system helps debug occasional problems that are hard to reproduce in development environments.

Monitoring is used to automatize the central production operations [6]. The profile of the pilots available at grid sites is used to pre-match workload to their production destination and allows to determine how much resources can be applied to over-demanding workload. The overall availability of sites is used as a dynamic indicator of how much work can be dispatched to them at any given time. This improves load balancing of resources for workflows that cannot run at all sites. The number of pending and running jobs per workflow/subtask is used to determine if more resources should be applied to specific workflows through a dynamic overflow mechanism. Runtime and memory usage is used to amend the job requirements and improve their accuracy based on the large scale observations over the LHC grid.

In addition to the automated procedures, the monitoring system is used on a daily basis to spot developing issues in the production schedd, computing site, etc. This has helped greatly.
the complex task of understanding the behavior of components in a distributed computing environment and has fueled the development of automated procedures.

References
[1] S. Chatrchyan et al. CMS Collaboration 2008 The CMS experiment at the CERN LHC J. Inst. 3 S08004
[2] Evans L and Bryant P 2008 LHC Machine J. Inst. 3 S08001
[3] Knobloch J et al. 2005 LHC Computing Grid Technical Design Report CERN-LHCC-2005-024
[4] D Hufnagel - The CMS TierO goes Cloud and Grid for LHC Run 2 - J. Phys.: Conf. Ser. 664 (2015) 032014
[5] S Fuess et al. - The HEP Cloud Facility: elastic computing for High Energy Physics. This Conference
[6] J-R Vlimant et al - Software and Experience with Managing Workflows for the Computing Operation of the CMS Experiment. This conference
[7] Douglas Thain, Todd Tannenbaum, and Miron Livny, “Distributed Computing in Practice: The Condor Experience” Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005
[8] J Letts et al for the CMS collaboration Using the glideinWMS System as a Common Resource Provisioning Layer in CMS 2015 J. Phys.: Conf. Ser. 664 062031 doi:10.1088/1742-6596/664/6/062031
[9] Apache, https://www.apache.org/
[10] Genshi, https://genshi.edgewall.org/
[11] RRDtool - rrdpython, https://oss.oetiker.ch/rrdtool/prog/rrdpython.en.html
[12] RRDtool, https://oss.oetiker.ch/rrdtool/index.en.html
[13] Single Sign On and Account Management Services, http://information-technology.web.cern.ch/services/SSO-Service
[14] E Fajardo et al for the CMS collaboration A new era for central processing and production in CMS 2012 J.Phys.: Conf. Ser. 396 042018 doi:10.1088/1742-6596/396/4/042018
[15] M Mascheroni et al for the CMS collaboration CMS distributed data analysis with CRAB3 2015 J. Phys.: Conf. Ser. 664 062038 doi:10.1088/1742-6596/664/6/062038

Appendix: Tables

| ClassAd | Information |
|---------|-------------|
| DESIRED_Sites | List of CMS Site Names on which job can be considered for job matching |
| MATCH.Glidein_CMSSite | CMS Site Name on which job is running |
| RequestMemory | The required amount of memory in MiB that this job needs to avoid excessive swapping. |
| MaxWallTimeMins | Absolute max time limit in minutes for job runtime |
| RequestCpus | A requested number of CPUs (cores). |
| JobStatus | HTCondor job status |
| JobPrio | Priority of HTCondor job |
| QDate | Time at which the job was submitted to the job queue |

Table 1. Specific ClassAds for building production and analysis views.

| ClassAd | Information |
|---------|-------------|
| WMAgent_RequestName | WMAgent Task Request Name |
| WMAgent_SubTaskName | WMAgent Sub-task name |

Table 2. Additional WMagent ClassAds collected to build the production view.
| ClassAd            | Information                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| CRAB_UserHN        | Grid Username                                                               |
| CRAB_ReqName       | CRAB Task name                                                              |
| ClusterId          | ClusterId identifies an HTCondor cluster                                    |
| TaskType           | TaskType, one of: ROOT, Job                                                 |
| CRAB_UserWebDir    | Users task web directory URL                                                 |
| DAG_NodesReady     | The number of DAG nodes that are ready to run, but which have not yet started running |
| DAG_NodesUnready   | The number of DAG nodes that are not ready to run. This is a node in which one or more of the parent nodes has not yet finished |
| DAG_NodesTotal     | The total number of nodes in the DAG, including the FINAL node, if there is a FINAL node |
| DAG_NodesFailed    | The number of DAG nodes that have failed. This value includes all retries, if there are any |
| DAG_NodesDone      | The number of DAG nodes that have finished successfully. This means that the entire node has finished, not only an actual HTCondor job or jobs |

**Table 3.** Additional CRAB3 classAds collected to build the analysis view.

| ClassAd            | Information                                                                 |
|--------------------|-----------------------------------------------------------------------------|
| CMSGWMS_Type       | Scheduler type, one of: prodschedd, crabschedd                              |
| Machine            | A string with the machine’s fully qualified host name.                      |
| TotalRunningJobs   | The total number of jobs from this condor_schedd daemon that are currently running, not including local or scheduler universe jobs. |
| TotalIdleJobs      | The total number of jobs from this condor_schedd daemon that are currently idle, not including local or scheduler universe jobs. |
| TotalHeldJobs      | The total number of jobs from this condor_schedd daemon that are currently on hold. |
| MaxJobsRunning     | An existing HTCondor configuration parameter that controls the number of jobs that will run at one time on a specified scheduler |
| IsOK||isWarning||isCritical | Custom CMS metric, to check the health status of Scheduler                        |
| TotalSchedulerJobsRunning | The total number of scheduler universe jobs from this condor_schedd daemon that are currently running |
| TotalSchedulerJobsIdle | The total number of scheduler universe jobs from this condor_schedd daemon that are currently idle. |
| StartSchedulerUniverse | boolean value as set in the configuration variable START_SCHEDULER_UNIVERSE expression |
| MyAddress          | String with the IP and port address of the condor_startd daemon which is publishing this machine ClassAd |
| ScheddIpAddr       | String with the IP and port address of the condor_sched daemon which is publishing this Scheduler ClassAd |

**Table 4.** Classads from schedulers for building CMS Global pool HTCondor schedulers overview.
| ClassAd                             | Information                                                                 |
|------------------------------------|-----------------------------------------------------------------------------|
| Name                               | The name of this resource; typically the same value as the Machine attribute, but could be customized by the site administrator |
| PartitionableSlot                  | A boolean value identifying that this slot may be partitioned                |
| GLIDEIN_CMSSite                    | CMS Site name on which pilot is running                                     |
| TotalSlotCpus                      | The number of CPUs (cores) in this slot. For static slots, this value will be the same as in Cpus |
| Cpus                               | The number of CPUs (cores) in this slot. It is 1 for a single CPU slot, 2 for a dual CPU slot, etc. For a partitionable slot, it is the remaining number of CPUs in the partitionable slot. |
| SlotType                           | The partitionable slot will have this attribute set to “Partitionable”, and all dynamic slots will have this attribute set to “Dynamic”. Other - “Static” |
| State                              | String which publishes the machine’s HTCondor state                         |
| Memory                             | The amount of RAM in MiB in this slot. For static slots, this value will be the same as in TotalSlotMemory. For a partitionable slot, this value will be the quantity remaining in the partitionable slot. |
| TotalMemory                        | The quantity of RAM in MiB available across the machine (not the slot). For partitionable slots, where there is one partitionable slot per machine, this value will be the same as machine ClassAd attribute TotalSlotMemory. |
| ChildMemory                        | A ClassAd list containing the values of the Memory attribute for each dynamic slot of the partitionable slot |
| ChildCpus                          | A ClassAd list containing the values of the Cpus attribute for each dynamic slot of the partitionable slot |
| ChildAccountingGroup              | A ClassAd list containing the values of the AccountingGroup attribute for each dynamic slot of the partitionable slot |
| AccountingGroup                   | A String of the AccountingGroup attribute for a Static slot. Only if it is not partitionable slot |
| GLIDEIN_ToRetire                   | Timestamp which defines how long should the condor_startd be running before entering into the Retiring state (thus not accepting any new jobs). |
| GLIDEIN_ToDie                      | Timestamp which defines when the condor_startd will enter ENDED State and will be removed from the Site queue |

Table 5. ClassAds from pilot statistics.

| ClassAd                   | Information                                                                 |
|---------------------------|-----------------------------------------------------------------------------|
| GLIDEIN_CMSSite           | CMS Site Name                                                               |
| GLIDEIN_MaxMemMBs         | Maximum memory available per pilot                                          |
| GLIDEIN_Max_Walltime      | Maximum walltime in minutes for a pilot runtime                             |
| GLIDEIN_CPUS              | Number of CPUs(cores) available in pilot                                   |
| DefaultPerFrontendMaxHeld | Maximum Held pilots per frontend in the factory. After this limits new pilots are not sent to the site queue |
| DefaultPerFrontendMaxIdle | Maximum Idle pilots per frontend in the factory. After this limit new pilots are not sent to the site queue |
| StatusHeld                | Currently Held pilots in the factory for this entry                         |
| StatusIdle                | Currently Idle pilots in the factory for this entry                         |
| StatusRunning             | Currently Running pilots in the factory for this entry                       |

Table 6. Parameters taken from all Factories XML configurations.