Controlled overflowing of data-intensive jobs from oversubscribed sites

I Sfiligoi1, F Wuerthwein1, B Bockelman2, D C Bradley3, M Tadel1, K Bloom2, J Letts1 and A Mrak Tadel1

1University of California San Diego, La Jolla, CA 92093, USA
2University of Nebraska – Lincoln, Lincoln, NE 68588, USA
3University of Wisconsin – Madison, Madison, WI 53706, USA

E-mail: isfiligoi@ucsd.edu

Abstract. The CMS analysis computing model was always relying on jobs running near the data, with data allocation between CMS compute centers organized at management level, based on expected needs of the CMS community. While this model provided high CPU utilization during job run times, there were times when a large fraction of CPUs at certain sites were sitting idle due to lack of demand, all while Terabytes of data were never accessed. To improve the utilization of both CPU and disks, CMS is moving toward controlled overflowing of jobs from sites that have data but are oversubscribed to others with spare CPU and network capacity, with those jobs accessing the data through real time Xrootd streaming over WAN. The major limiting factor for remote data access is the ability of the source storage system to serve such data, so the number of jobs accessing it must be carefully controlled. The CMS approach to this is to implement the overflowing by means of glideinWMS, a Condor based pilot system, and by providing the WMS with the known storage limits and let it schedule jobs within those limits. This paper presents the detailed architecture of the overflow-enabled glideinWMS system, together with operational experience of the past 6 months.

1. Introduction

The CMS[1] experiment at LHC is recording a large number of experimental physics events which result in several Petabytes of data each year. The data is processed once by a central operations team with the result being several partially-overlapping physics streams of events. These streams are then distributed to about 50 sites worldwide, with each file being available at two or more locations. A similar workflow produces an equivalent amount of simulated, or Monte Carlo (MC) events that are needed to validate the physics analysis.

The CMS analysis computing model for analyzing this data by the physics community was always relying on jobs running near the data. This has the advantage of avoiding the use of Wide Area Networking (WAN), thus being limited by the throughput of the site-local storage subsystem only, and resulting in high CPU utilization. However, it also restricts those same jobs to only CPUs close to the data. As a result, the proper placement of data at various sites is of paramount importance, and in CMS it was organized at management level, based on the expected needs of the CMS community[2].
However, community needs change in time, resulting in periods of time when a large fraction of CPUs at certain sites were sitting idle due to lack of demand, while a large number of users' jobs were waiting for CPU at sites with popular physics streams. As a side effect, Terabytes of disk space hosting unpopular physics streams were never accessed in the same period of time.

To improve the utilization of both CPU and disks, CMS is moving toward controlled overflowing of jobs from sites that have data but are oversubscribed to others with spare CPU and network capacity, with those jobs accessing the data over WAN, with the remote-I/O capabilities delivered by the Xrootd software[3]. The major limiting factor for remote data access is the ability of the source storage system to serve such data, so the number of jobs accessing it must be carefully controlled. The CMS approach to this is to implement the overflowing by means of glideinWMS[4], a Condor[5] based pilot system, and by providing the WMS with the known storage limits and let it schedule jobs within those limits.

It should be also noted that the glideinWMS software is composed of three components, a Glidein Factory, a VO Frontend and Condor proper, but in this paper we will not distinguish the three, and the term “glideinWMS” may apply to any combination of them. We believe this makes the paper much more readable, and the interested reader should consult the referenced paper for more details.

2. Underutilized resources

A major driver for this activity has been the realization that CMS was using its compute and storage resources inefficiently. On one side, we noticed that a large fraction of CMS datasets were rarely used, while on the other, many sites often did not have any CMS jobs queued.

To illustrate the dataset utilization problem, we here present the access statistics of the CMS datasets in use during summer 2011. As can be seen from Table 1, 14% of all datasets were never used, while 2% of the datasets accounted for the vast majority of all activity.

| Number of accesses | 0 | 1 | 10 | 100 | 1k | 10k | 100k | 1M |
|-------------------|---|---|----|-----|----|-----|------|----|
| Number of datasets| 272 | 161 | 511 | 629 | 175 | 114 | 34 | 2 |
| Fraction of datasets| 14% | 8% | 27% | 33% | 9.2% | 6.0% | 1.8% | 0.1% |

We also present the information collected in the same period at the glideinWMS based server hosted at UCSD. While only representing a fraction of CMS analysis activity, it still provides an indication of the kind of problems faced by CMS. In Fig. 1:

- the blue line represents the number of gatekeepers, or Compute Elements (CEs) with jobs running on them,
- the red area represents the number of CEs with at least one job waiting to get the needed resources,
- the black line represents the number of CEs with more than 1k jobs waiting, thus representing the CEs with strong demand, and
- the green line represents the number of CEs supported by the UCSD setup.

As can be seen, the blue line has some dips, indicating that not all CEs are used all the time. Moreover, the red area is very spiky and the black line is consistently low, indicating that only a few CEs are really in strong demand. Furthermore, while not all sites have the same amount of CEs, it can be assumed that the same conclusions can be applied to the sites as well.

The two problems go hand in hand in CMS, since datasets are distributed among sites in a relatively homogeneous fashion. Sites with popular datasets thus tend to be always fully utilized, with long wait times for the users, while sites hosting rarely used datasets may serve very few CMS jobs.
3. Remote access to data

To solve the above mentioned problems, CMS decided to allow remote access to data through remote I/O streaming using the Xrootd protocol over the WAN. This approach is based on two observations: it is preferable to sacrifice a bit of CPU efficiency to get better overall throughput, and most sites have spare storage bandwidth, both in terms of bytes per second and transactions per second, even when all their CPUs are running CMS jobs.

Reading data over WAN is obviously less efficient than reading it from a site-local storage system. However, if there are no CMS jobs waiting for resources at one site, a slightly inefficient job is still orders of magnitude better than no job at all. Said that, CMS has invested significant R&D effort into minimizing the inefficiency of WAN access, and recent CMS software can access remote data with only about a 10% efficiency hit[6].

A potentially bigger problem is getting enough bandwidth out of the data hosting site. CMS sites have historically sized their storage systems based on the size of their compute cluster, which means that the amount of bandwidth left for remote access is limited. If too many external processes were to start reading the hosted data, it would likely drastically degrade the performance of the storage subsystem, reducing the efficiency of both the remote and the local processes. Nevertheless, most large CMS sites do have some spare bandwidth; it just must be carefully managed.

4. The adopted solution

The solution adopted by CMS to control the remote access to data is based on glideinWMS, a Condor based pilot system, which was already in use for several years by the CMS analysis activities[7]. However, we had to significantly change its configuration in order to implement the overflow use case.

The CMS analysis uses the CRAB2 Server software to submit Condor jobs on behalf of the users. The user provides the list of datasets needed; CRAB2 looks up the list of sites that host those datasets and specifies the list of sites in the job description forwarded to Condor in the job ClassAd[8,9]. This list is intended to be a whitelist of sites where the job is allowed to run. In years past, it was indeed used as such by glideinWMS to decide where to provision new glideins and by the glideins to decide whether to start eligible jobs. See Fig. 2 for a schematic view.

![Figure 1. Number of CEs used during summer 2011 by the glideinWMS based CMS CRAB2 server hosted at UCSD](image-url)
To implement the remote access, we changed the logic to instead interpret the CRAB2-provided list as a whitelist of sites from where the data will be streamed from, and use it essentially just for regulating the number of glideins requested for each data source. The rules for the selection of sites the jobs can run on are thus very relaxed, i.e. essentially any site is a good one, and any restriction is done purely for efficiency reasons. For example, we could select only sites on the same continent. Of course, we want to continue to use the “regular logic” for most of the glideins, as running jobs close to the data is still the most efficient computing model. The remote access should be used only when the sites hosting the datasets are oversubscribed, i.e. we want the “overflow logic”. A schematic view can be found in Fig. 3.

Figure 2. A schematic view of the traditional CMS glidein-based analysis workflow

In order to have the regular and overflow logic to coexist in the same system, we first had to move all the matchmaking logic into the glideins themselves. The original CRAB2 integration with glideinWMS used the standard Condor practice of expressing the job requirements as a boolean expression that was passed to Condor during job submission, with the glideins providing the needed attributes and essentially accepting any job; an example can be found in Fig. 4. While this has historically worked fine as long as the matchmaking logic was homogeneous, it made it impossible to mix the regular and overflow logics in the same system. We thus decided to move the requirements entirely into the glideins, with the jobs just publishing appropriate attributes and having no explicit restrictions on where they can run, as illustrated in the example in Fig. 5. This of course required a change in both the glideinWMS configuration and CRAB2 software, as the job description is generated by the latter.

Figure 3. A schematic view of the overflow-based workflow
The glideinWMS changes were implemented by instantiating two glideinWMS instances; one for the regular glideins and one for the overflow glideins, both running on the same hardware. The regular glideinWMS instance was essentially left unaltered, modulo the glidein configuration changes described above. The overflow glideinWMS instance instead differs from the regular one in three critical aspects; it has a much more relaxed resource provisioning logic, only provisions glideins when the data source site is oversubscribed and implements the per-site limits. This deployment scenario was chosen for ease of maintenance, since we need multiple overflow groups, as described below, and glideinWMS does not support nested groups.

With remote access, any compute resource can run any job, so the only necessary condition for the overflow glideinWMS instance logic is not to run on the data source site itself. Additionally, for efficiency reasons, we also require that the target site is located on the same continent as the data source site.

As remote access is less efficient than local access, we want to avoid requesting overflow glideins unless the source data site is oversubscribed. We estimate this by looking at the age of user jobs in the Condor queues, where the existence of old jobs indicates that the site is not providing enough CPU resources. The current threshold has been set to 6 hours. Furthermore, when the overflow glideins are requested at other sites, overflow glideins should not “steal” CPU resources from regular glideins. Since the two instances do not talk to each other, a lower priority credential is used for overflow provisioning.

To implement the per-site limits, we had to implement one group per source site, as the glideinWMS software only allows to set limits on the number of glideins being provisioned by means of groups in the configuration file. This makes the configuration file quite big, but still quite manageable. More problematic is the fact that the actual site limits are hard coded in the instance configuration, and can only be modified by the instance administrator. This means that these numbers cannot be made dynamic, and any change in the estimated storage throughput by the data source site owners must propagate through a human readable channel, such as email, and thus changes can only be done sparingly.

5. Monitoring the system

While the glideinWMS system described above handles autonomously the task it was given, it relies heavily on the appropriate limits put into its configuration. Setting those limits is however not (yet) a precise science, since we have to estimate the maximum number of remote readers based on the expected storage bandwidth availability, an estimate in itself. A sophisticated infrastructure has thus
been put in place to monitor the behavior of the storage systems, so we can detect a storage overload in its early stage, and react appropriately.

The main monitoring infrastructure[10] is based on MonALISA[11]. The monitored parameters include, among others, server load, incoming and outgoing traffic and number of connected clients. MonALISA provides both visualization tools and triggers for reporting of various failure modes and excessive usage. The details of individual user sessions and data-transfers are also monitored, and can be used when investigating excessive or abusive usage of the system.

The user jobs themselves are also monitored. A daily email is generated containing the average CPU efficiency and error rates of both regular and overflow jobs. The needed information is all available in the ClassAds of the completed jobs, a standard Condor monitoring tool. Furthermore, CMS jobs have well defined exit codes, making it easy to separate file access errors from other modes of failure. By comparing the two types of jobs one can infer if the problem is due to remote data access, or if the user jobs themselves were misbehaving. For example, in the example email shown in Fig 6, we can see the user *uscms1832* is failing both for regular and overflow jobs (i.e. exit code 84), so his problems are likely not due to the use of overflows themselves.

| All sites                                      | (continuation from the left)                                                                                                                                 |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Overflow: 904 (6.5% wall 26.5%) Normal: 13800  |                                                                                                                                                               |
| Exit 0: 95.0% (vs 91.7%) wall 97.1%            | 2 uscms1832                                                                                                                                                 |
| Exit 84: 0.77% (vs 0.88%) wall 0.053%          | 1 uscms2514                                                                                                                                                 |
| Efficiency: 77.2% (vs 74.9%)                  | ...                                                                                                                                                         |
| Eff >80%: 50.1% (vs 45.3%)                    | Overflow users with exit not 84                                                                                                                                 |
| Only UCSD+Nebraska+Wisconsin+Purdue           | 5 uscms1525                                                                                                                                                 |
| Overflow: 829 (15.7% wall 90.0%) Normal: 5264 | 71 uscms1832                                                                                                                                                 |
| Exit 0: 95.4% (vs 93.4%) wall 97.3%           | ...                                                                                                                                                         |
| Exit 84: 0.72% (vs 0%) wall 0.053%            | Normal users with exit not 84                                                                                                                                 |
| Efficiency: 77.3% (vs 74.2%)                  | 96 uscms1525                                                                                                                                                 |
| Eff >80%: 49.2% (vs 33.8%)                    | 2 uscms1832                                                                                                                                                 |
| (continues on the right)                      | ...                                                                                                                                                         |

**Figure 6.** Example job monitoring email

We have recently extended the above report by including data from the Xrootd redirector logs. Unfortunately, Xrootd and Condor log data lack a common identifier that we can use to directly join the two datasets, so we have to do an approximate match based on the start and end date of the job, and the reported hostname. Using this method, we can report the data source for many of the jobs which fail due to file-open failure.

6. Operational experience

CMS is still in the process of validating the above setup and has thus enabled overflow from only a few selected sites to a few other selected sites. The four sites with overflow enabled from them are University of Nebraska – Lincoln (UNL), University of California San Diego (UCSD), University of Wisconsin – Madison, and Purdue University, and we overflow into the same four sites plus California Institute of Technology (Caltech).

The overflow setup has been put in production in Fall 2011, after about a month of testing by volunteer users. Since then, it has accounted for about 7% of all the jobs that ran through the glideinWMS-based CRAB2 instance. The operational experience has been for the most part very positive, with just a few significant problems experienced in over six months of operation.
The first problem was due to the specific needs of CMS software to use remote files. CMS jobs expect a site-local configuration file to point them to the global Xrootd redirector; the overflow glideins must thus only run on worker nodes with this setup put in place. We did configure the overflow glideinWMS to only use sites that claimed to have the appropriate setup in place, but discovered that for various reasons only a subset of worker nodes had. This resulted in a significant fraction of overflow jobs failing. Luckily, the solution was simple; we added a check for the appropriate setup at glidein startup. This prevents misconfigured worker nodes from being considered for matching overflow jobs, insulating the users from the problem and delegating the troubleshooting of the bad nodes to the glideinWMS operators[12]. However, a significant number of user jobs did fail before we properly diagnosed the problem, showing the importance of proper monitoring and alarms.

A similar problem exists regarding misbehaving site-specific services. In the traditional setup, if a site-specific service misbehaves, only jobs for that site will fail; jobs meant for other sites will not be affected. In the overflow setup, any job can run at any site, making the impact of a misbehaving site service much more problematic. While not a new problem, the increased scale of the problem prompted us to spend significantly more time writing validation tests for the glideinWMS system, thus drastically reducing the related error rates for user jobs running on both traditional and overflow setup.

We have also discovered one unexpected flaw in the implementation of the federated Xrootd setup that we used in Fall 2011. When a user contacted the Xrootd redirector with the request of a file, the redirector would query the known Xrootd data-hosting servers, and then point the user to the server which claimed to have the file and be the less loaded. However, if the chosen Xrootd server did not deliver the file, e.g. because the user could not be authenticated or authorized, the user job would fail. If such a misbehaving Xrootd server were to claim to also be very lightly loaded, a large fraction of the overflow jobs could fail due to most requests being redirected to this server. We have seen up to 70% error rates for short periods of time in the first months of overflow operations.

This problem have been solved since CMS software version 5, released in early 2012, and jobs using this newer version now have a fall-back mechanism in place to address it; if a Xrootd server refuses to serve a file, the client will ask the redirector for a different server, if one exists. Error rates due to Xrootd server problems for these jobs have been consistently low, and have been due mostly to corrupted files. The same problem exists also for non-overflow jobs, but the load leveling nature of the Xrootd federation can propagate the problem to many more jobs.

Furthermore, debugging Xrootd related problems can be much harder for jobs using older versions of CMS software, since they only log the access to the Xrootd redirector, and not the actual Xrootd server. This, too, has been fixed in version 5.

Nevertheless, it takes time for all CMS users to migrate to the latest CMS software releases, and even when they do, it is improbable that the clients will ever be bug-free. We are thus heavily relying on monitoring to promptly catch such situations, and shut down the offending server(s), as well as identify other causes of problems and include bug-fixes in the yearly major CMS SW releases. The cumulative effect will be to further decrease the percentage of jobs failing due to this issue.

7. Future work

The most obvious evolution for the near future is expanding the number of sites included in the overflow setup, both as being the source as well as the destination for overflowing glideins. Our goal is to eventually cover all the major CMS sites.

On the development side, the major envisioned task is to make the limit setting more dynamic. We expect we will need help from the Condor daemons to achieve this task, and have started a discussion with their development team to identify a path to obtain the missing functionality.

Finally, we plan to continue improving our monitoring tools and to keep collaborating closely with the Xrootd team to improve the performance of the system.
8. Conclusions
The CMS analysis computing model for analysis jobs to only run near the data has proven to be suboptimal, resulting in both unused CPU time and sparsely accessed disk space. CMS has thus started looking at allowing remote access to the data in a controlled manner, and federated Xrootd data access with the overflow-based glideinWMS CRAB2 setup was the chosen solution.

All in all, the overflow setup has served CMS well, significantly shortening the wait times of users of the most sought after datasets. Over the past six months, the overflow glideins have provided CPU for about 7% of all the glideinWMS-based CRAB2 jobs. While we still have some rough edges to polish, we consider the experience so far to be a success.

Acknowledgements
This work is partially sponsored by the US National Science Foundation under Grants No. PHY-1104549, PHY-1104447, PHY-1104664 (AAA) and PHY-0612805 (CMS Maintenance & Operations), and the US Department of Energy under Grant No. DE-FC02-06ER41436 subcontract No. 647F290 (OSG).

References
[1] The CMS collaboration 2005 The CMS Computing Project Technical Design Report In proc. of CMS Technical design reports, pp.1-169. ISBN: 92-9083-252-5.
[2] Giordano D et al. 2012 Implementing data placement strategies for the CMS experiment based on a popularity mode To be published in these same conference proceedings.
[3] Dorigo A, Elmer P, Furano F and Hanushhevsky A 2005 XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment Proc. of the 4th WSEAS International Conference on Telecommunications and Informatics, pp.1-6. ISBN: 960-8457-11-4.
[4] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Würthwein F 2009 The Pilot Way to Grid Resources Using glideinWMS Computer Science and Information Engineering, 2009 WRI World Congress on, vol.2, pp.428-432. doi:10.1109/CSIE.2009.950
[5] Thain D, Tannenbaum T and Livny M 2005 Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience, vol.17, no.2-4, pp.323-356. doi:10.1002/cpe.938.
[6] Canal P, Bockelman B and Brun R 2011 ROOT I/O: The Fast and Furious J. Phys.: Conference Series 331 042005. doi:10.1088/1742-6596/331/4/042005.
[7] Bradley D et al. 2010 Use of glide-ins in CMS for production and analysis J. Phys.: Conference Series 219 072013. doi:10.1088/1742-6596/219/7/072013.
[8] Condor Team 2012 Condor Manual. University of Wisconsin Web site. http://research.cs.wisc.edu/condor/manual/v7.8/11_Appendix_A.html#sec:Job-ClassAd-Attributes
[9] Solomon M 2004 The ClassAd Language Reference Manual - Version 2.4. University of Wisconsin Web site. http://research.cs.wisc.edu/condor/classad/refman.pdf
[10] Tadel M et al. 2012 Xrootd Monitoring for the CMS experiment To be published in these same conference proceedings.
[11] Legrand I et al. 2009 MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. Computer Physics Communications, vol.180, no.12, pp.2472–2498. doi:10.1016/j.cpc.2009.08.003.
[12] Sfiligoi I, Würthwein F, Dost J M, MacNeill I, Holzman I and Mhashilkar P 2011 Reducing the Human Cost of Grid Computing With glideinWMS. Proc. of CLOUD COMPUTING 2011. pp.217-221. ISBN: 978-1-61208-153-3.