Use of glide-ins in CMS for production and analysis

Journal of Physics: Conference Series

Daniel C Bradley, Oliver Gutsche, Kristian Hahn, Burt Holzman, Sanjay Padhi, Haifeng Pi, Igor Sfiligoi, Eric Vandering, Frank Würthwein (Any one else ??)

UWM, FNAL, MIT, UCSD

E-mail: cms-dct-wms@fnal.gov

Abstract. With the evolution of various grid federations, the Condor glide-ins represent a key feature in providing a homogeneous pool of resources using late-binding technology. The CMS collaboration uses the glide-in based Workload Management System, glideinWMS, for production (ProdAgent) and distributed analysis (CRAB) of the data. The Condor glide-in daemons traverse to the worker nodes, submitted via Condor-G. Once activated, they preserve the Master-Worker relationships, with the worker first validating the execution environment on the worker node before pulling the jobs sequentially until the expiry of their lifetimes. The combination of late-binding and validation significantly reduces the overall failure rate visible to CMS physicists. We discuss the extensive use of the glideinWMS since the computing challenge, CCRC08, in order to prepare for the forthcoming LHC data-taking period. The key features essential to the success of large-scale production and analysis at CMS resources across major grid federations, including EGEE, OSG and NorduGrid are outlined. Use of glide-ins via the CRAB server mechanism and ProdAgent as well as first hand experience of using the next generation CREAM computing element within the CMS framework is also discussed.

1. Introduction

The CMS collaboration has adopted Grid computing as its base computing model to simplify the deployment and management of the tens of thousand CPUs needed to accomplish its mission. While the Grid computing paradigm has proven to be a boon for resource providers, allowing them to keep their administrative autonomy over the resources they manage, the added abstraction layer has introduced several problems for the users, ranging from higher complexity to decreased reliability.

One solution that has proven to significantly reduce user problems is the late-binding, or pilot technology. A late-binding Workload Management System (WMS) hides the complexity of the Grid environment by dynamically creating a virtual private pool of compute resources, thus giving users an environment similar to a dedicated batch cluster. This paper describes the experience of the CMS collaboration with one late-binding WMS implementation called glideinWMS.

2. Late binding based Workload Management System - GlideinWMS

The Grid paradigm calls for the compute resources to be partitioned into multiple independent pools, called Grid sites, with only a thin common layer to provide interoperability, as shown

in Fig. X. Without some additional tools, this approach makes the life of a Grid user quite unpleasant.

The four major problems the users experience are:

- A user must partition his/her jobs between the resource pools. Finding the optimal partition is far from an easy task, as explained below.
- At any Grid site, the common layer provides only very limited information about the status and policies of the batch system that handles the local resource pool. This is a necessary evil that allows the common layer to present the information from all the different batch system implementations in a uniform way.
- The common layer provides only very limited information about the progress of a job, once it is accepted at a Grid site. Again, this is a necessary evil that allows the common layer to monitor jobs submitted to different batch systems.
- Each Grid site is allowed to configure the worker nodes the way it likes, within very permissive limits. Users are expected to write their compute jobs in such a way to automatically adapt to any condition they encounter.

The approach taken by the late binding Workload Management Systems (WMS) to ease the user burden is to create a dynamic virtual private pool of compute resources by submitting pilot jobs to the Grid sites. Once a pilot job starts, it joins the virtual private pool and starts a user job from the late binding WMS job queue, as shown in Fig X.

This insulates the users from the Grid layer, giving users the impression of running on a single, local, dedicated pool of compute resources and thus elimitating, or at least reducing the magnitude of the above mentioned major problems:

- By having a single resource pool, no job partitioning is needed.
- Detailed information about the status and policies of the virtual private pool can be made available, since users can use the tools provided by the specific implementation of the late binding WMS instance used.
- Detailed information about the progress of a job can be made available, since users can use the tools provided by the specific implementation of the used late binding WMS instance.
- The late binding WMS can reduce the hetereogeneity of the compute resources, by either only gathering properly configured worker nodes, or by providing wrapper scripts that provide common tools and libraries.

The late binding WMS used by CMS is called **glideinWMS**. GlideinWMS is based on the Condor batch system, with the addition of a thin layer responsible for the submission of the pilot jobs.

A glideinWMS virtual private pool is just a regular Condor pool, where worker node Condor daemons, i.e. *condor_startd* and *condor_starter*, have been downloaded, configured and started by a glideinWMS pilot job; such pilot jobs are known as *glideins*. Since the different Condor daemons are dispersed around the world and use wide area networking to communicate to each other, the daemons are configured to use strong, x509 based authentication for authorization and message integrity purposes. The worker node Condor daemons are also configured to have a limited lifetime, in order to fit within the wallclock limits of the batch system of the Grid site they are running on. For all other practical purposes, the resulting Condor pool is indistinguishable from a dedicated Condor pool without a shared file system.

The submission of glideins is regulated by two types of daemon processes; one or more **glidein factories** and one or more **VO frontends**. The two types of processes communicate by means of ClassAds using a dedicated *condor_collector* daemon:

(i) A glidein factory advertizes what Grid sites it knows about, and can submit to, together with the characteristics of the site (for example: what versions of CMS software are installed there).

(ii) The VO frontend queries the user queues, i.e. the *condor_schedds*, matches the found jobs with the factory ClassAds, and then advertizes how fast should the factory submit new glideins.

(iii) Finally, the factory reads the VO frontend ClassAds and starts submitting the glideins at the specified rate.

All together make the system work as illustrated in Fig. X.

2.1. Interoperability between EGEE, OSG, and NorduGrid

In principle, the problem of interoperability between different grids is reduced to having a condor_G client for submission of the glideins for the particular grid. All other incompatibilities can in principle be resolved via appropriate configuration of the glideins. In practice, CMS deals with many of the differences between EGEE and OSG inside the CRAB or ProdAgent layers of the software stack, thus requiring little special configuration in the glideins. As part of our work for CCRC08, we worked with the Condor team on the details of submitting glideins via condor_G to NorduGrid. This resulted in the first ever use of NorduGrid resources for data analysis with CRAB.

2.2. Scalability of the system

3. Use of glideinWMS for Monte Carlo Production and Data Reprocessing

4. Use of glideinWMS for data analysis

4.1. CMS User Analysis and CCRC-08

4.2. User analysis using Crabserver

4.3. User level MC production using Crabserver

4.4. Recent results and JobRobots

5. Coherent monitoring interface for the system

6. Experience using next generation CREAM CE