FERMILAB-PUB-07-468-CD

glideinWMS - A generic pilot-based Workload Management System

Igor Sfiligoi[1]

[1]Fermi National Laboratory, Batavia, IL 60510 , USA

E-mail: sfiligoi@fnal.gov

Abstract. The Grid resources are distributed among hundreds of independent Grid sites, requiring a higher level Workload Management System (WMS) to be used efficiently. Pilot jobs have been used for this purpose by many communities, bringing increased reliability, global fair share and just in time resource matching. glideinWMS is a WMS based on the Condor glidein concept, i.e. a regular Condor pool, with the Condor daemons (startds) being started by pilot jobs, and real jobs being vanilla, standard or MPI universe jobs. The glideinWMS is composed of a set of Glidein Factories, handling the submission of pilot jobs to a set of Grid sites, and a set of VO Frontends, requesting pilot submission based on the status of user jobs. This paper contains the structural overview of glideinWMS as well as a detailed description of the current implementation and the current scalability limits.

1. Introduction

With millions of jobs coming from thousands of users, and tens of thousands resources deployed to run them, a higher level Workload Management System (WMS) is essential for the Grid[1] be used efficiently. WMSes are nothing new in the computing world, having been used on local computing clusters for decades. Such WMSes are usually referred to as batch system.

The Grid resources are however clustered into hundreds of independent pools, each pool having its own local WMS and its own set of policies, with only a thin common interoperability layer. Such a hierarchical and heterogeneous system introduces new levels of complexity and is generally not supported by mainstream batch system developers, requiring major software development effort to obtain a Grid-wide WMS.

An alternative approach to the problem is to create a homogeneous virtual private pool of compute resources, and use a standard batch system to manage them. In order to gather resources, batch system components are packaged as Grid jobs and sent to all the available Grid pools. Such jobs are also known as pilot jobs.

The glideinWMS is based on the Condor[2] batch system. Condor was chosen because of its naturally distributed architecture. Indeed, Condor itself provides a rudimentary pilot submitter, called glidein. The glideinWMS provides a mean to automatically submit pilot jobs to the Grid, trying to maximize the amount of completed jobs, while minimizing the wasted resources. The standard Condor daemons then handle the scheduling of user jobs to the available resources.

2. Condor glideins

Condor is a widely used batch system deployed on thousands of local pools worldwide. Its architecture is based on a set of daemons, collaborating by exchanging messages over the network.

A Condor pool typically has three classes of machines:

- *Submit machines.* Each of these machines run a **schedd** daemon that holds a job queue. Several users can submit to the same queue and Condor will still maintain a far use of resources.
- *Execute machines.* Each of these machines run a **startd** daemon that handles the resource, advertising several of its properties (like OS, available memory, etc.). Any job started on this resource will be handled by this daemon.
- *A central manager machine.* This machine runs a **collector** and a **negotiator**. The collector holds the list of all the other Condor daemons in the system. The negotiator matches jobs to resources.

See figure 1 and figure 2 for a schematic overview.

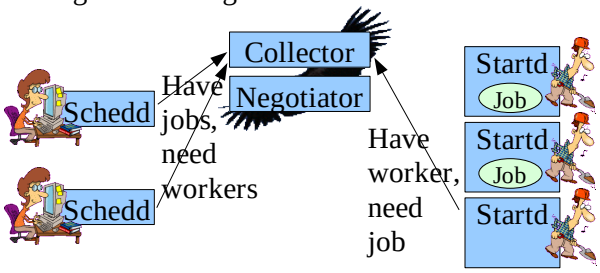

Figure 1. Condor daemons

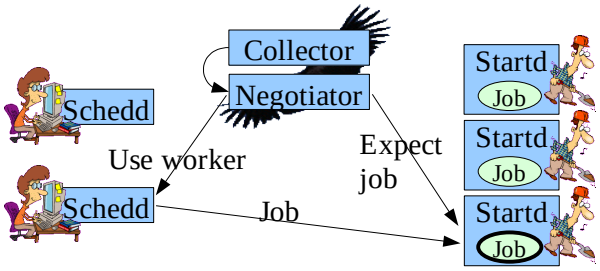

Figure 2. Condor match

A Condor glidein is simply a startd started as a Grid job. Once the startd registers with the collector, a schedd can send a job as it would do in a local pool, as shown in figure 3.

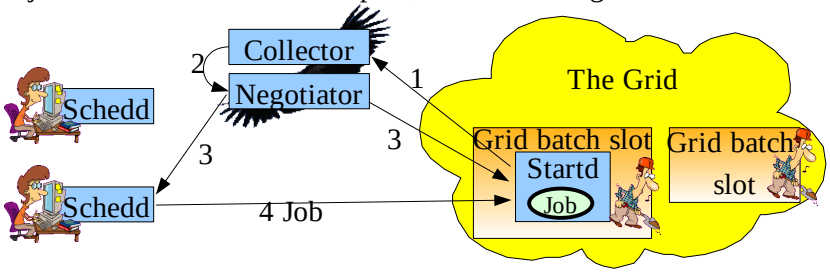

Figure 3. Condor glidein

3. The glideinWMS

3.1. Overview

The glideinWMS is a WMS based on a virtual private Condor pool. The system is composed of several elements:

- *A Condor central manager machine.* This machine runs a collector and a negotiator and holding the state of the virtual private pool.
- *A set of submit machines.* These machines run the schedds and keep the job queues.
- *A set of Glidein Factories.* Each of these machines run a **glidein factory daemon** that will submit the pilot jobs to a set of Grid pools.
- *A set of VO Frontends*. Each of these machines run a **VO frontend daemon** that monitors the schedd queues, matches them to the glidein factories, and decides which glidein factory should submit the pilot jobs, and how many of them.
- *A WMS collector machine.* This machine runs a collector that is used for communication between the glidein factory daemons and the VO frontend daemons.

See figure 4 and figure 5 for a schematic overview.

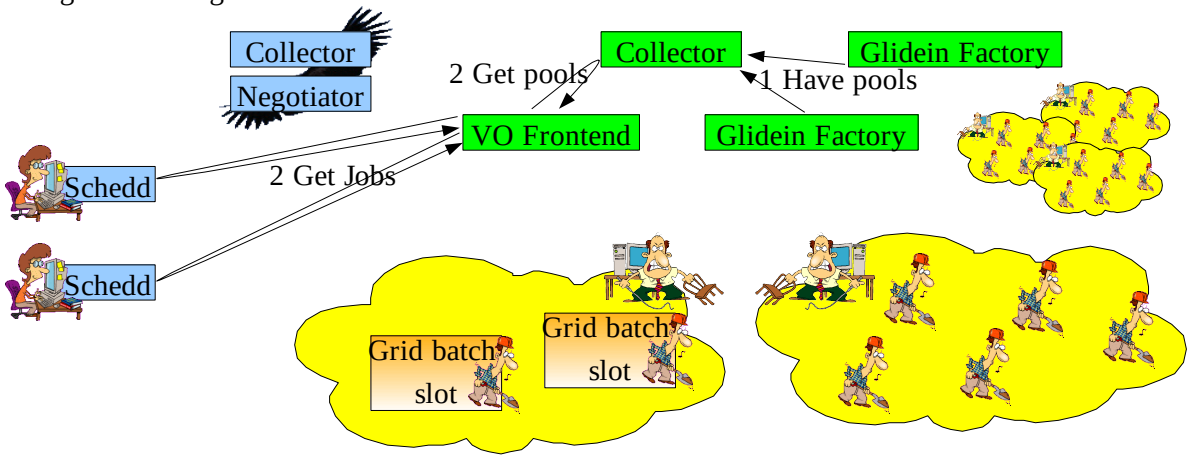

Figure 4. glideinWMS daemons

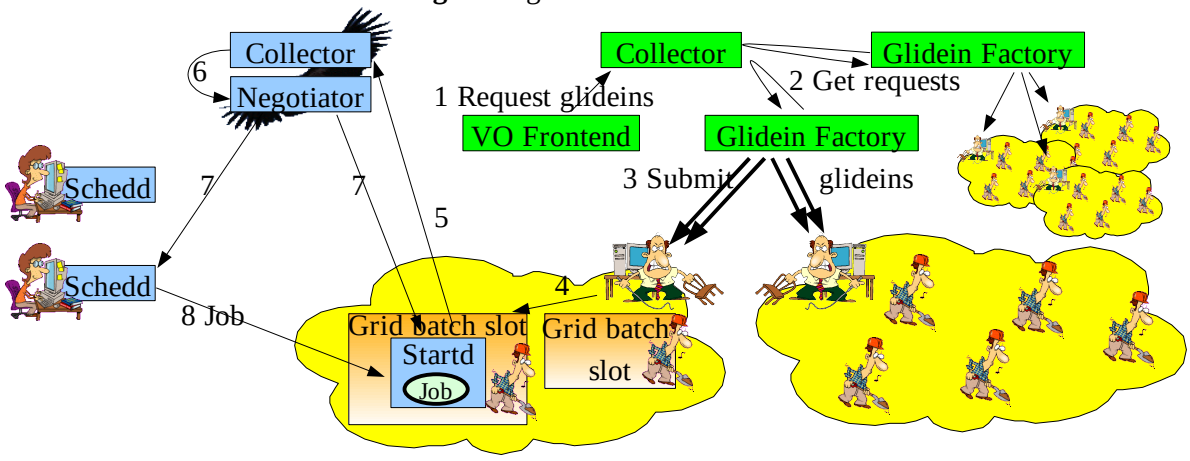

Figure 5. glideinWMS resource matching

The system has been designed to be highly scalable. There are only two pieces that cannot be replicated; the Condor central manager and the WMS collector, and both are relatively lightly loaded. Any other daemon, if overloaded, can be split into two or more daemons of the same kind, and its load distributed among them.

3.2. Operation logic

The glideinWMS tries to maximize the amount of user jobs, while minimizing the amount of wasted resources. It does this by keeping a steady pressure on the Grid pools; as long as there are jobs in the schedd queues that could potentially run on a Grid pool, a fixed number of pilot jobs is being kept in that pool queue. However, as soon as there are no more suitable jobs waiting in any of the schedds, no more pilot jobs are submitted. If any glideins start after all the suitable user jobs have started, the glidein itself will exit within a few minutes.

The pilot jobs are being submitted by the Glidein Factories, but it is VO Frontends' job to decide how many pilot jobs to keep in each and every Grid pool. This number is calculated by matching the attributes of the user jobs, provided by the schedds, to the attributes of the Grid pools, provided by the Glidein Factories. If the number of matches is higher than the desired pressure, the Glidein Factories are told to keep the pressure. Else, the pressure is reduced as appropriate.

Once a pilot job starts on a Grid resource, it first validates it. This includes looking for the appropriate software libraries, ensuring sufficient disk space is available, and so on. Once these tests successfully complete, it configures the Condor daemons (i.e. the startd and supporting code) and starts them. All control is delegated to them, and the pilot job wrapper just waits for their termination to do the final cleanup.

Once the Condor daemons start, they behave like in a dedicated Condor pool. The startd registers back with the Condor central manager and waits to be matched. Once a suitable job is found, the schedd holding the job will contact the startd and the job starts running; see Figure 5 for a schematic view. The startd can run multiple consecutive user job, to keep the pilot wrapper overhead low for very short jobs. However, the pilot needs to end within the Grid back slot lease time, so the startd will stop accepting new jobs if they cannot complete in time.

One important observation is that the negotiator on the Condor central manager machine decides which job(s) will be run on the resource provided by the pilot job. The decision is taken the moment the resource becomes available, so the job started on a given glidein is most probably not the one for which the glidein was submitted. This is especially true when several jobs are run by the same glidein.

3.3. Glidein Factories and VO Frontends

The only two pieces that are glideinWMS specific are the glidein factory and the VO frontend daemons. Everything else in the system uses standard Condor binaries.

The glidein factory daemon is configured to manage one or more Grid pools. The information about the Grid pools, like the CPU type and installed OS, the installed software libraries and the associated disk storage are given to the daemon as configuration parameter, and can be gathered in a variety of ways; the daemon itself does not actively look for updates. The daemon advertises this information to the WMS collector, using one ClassAd per Grid pool, and waits for orders.

The VO frontend is configured to look for jobs in a list of schedds and match them to the information published by the glidein factories, as described in the previous subsection. Once the

pressure numbers have been calculated, it publishes this information back, one ClassAd per Grid pool, to the WMS collector. The VO frontend daemon is given the list of schedds to monitor, the matching rule and the maximum pressure to apply to any site, and from there on it operates autonomously.

The glidein factories have dedicated processes that poll the WMS collector for ClassAds requiring pressure for a specific Grid pool and act on it. Each of these processes is autonomous; it will keep a steady pressure on a side until it is told to change the pressure. This means that the pressure will be maintained without continuous interventions from VO frontend. The WMS collector will however drop a ClassAd if not refreshed in a reasonable amount of time, preventing an infinite amount of glideins to be submitted if the VO frontend dies.

See figure 6 for a schematic overview.

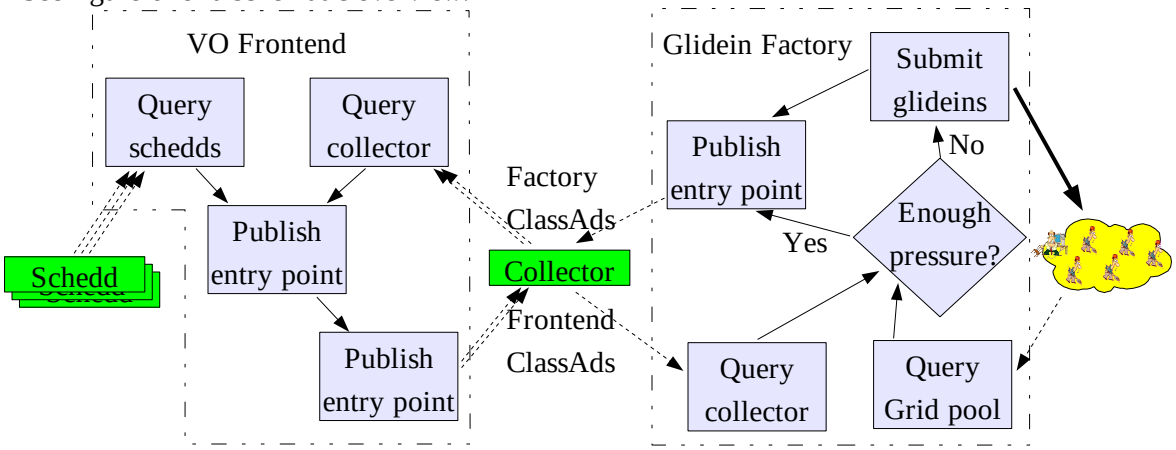

Figure 6. Glidein Factories and VO Frontends

3.4. The pilot jobs

The pilot job wrapper is a very lightweight script; it's only task is to fetch external files, and execute a subset of them. HTTP is used for file delivery, allowing for easy caching. Transferring files in clear text does however introduce severe security risks, especially since some of this files are executable. The glideinWMS pilot addresses this problem by using SHA1 checksums on all the files.

The SHA1 checksum checking is hierarchical. A file containing the checksums of all the other files to be downloaded is created its checksum calculated. This later checksum is then passed as an argument to the pilot job wrapper, so that it can verify the integrity of this summary file. Once the checksum files has been verified, all the other files can be verified by the information contained in it.

The files downloaded by the pilot wrapper include configuration files, validation scripts, configuration scripts and Condor binaries. All executable files are executed in the order they are downloaded, and if any of them fails, the pilot wrapper script exits. The condor startup script is the latest one in the list and starts the Condor daemons. This guarantees that Condor daemons only start after all the validation tests have been passed.

4. Condor and the Grid

Condor has been developed for and is mainly deployed on local area networks with daemons installed by local system administrators. When moving to a widely distributed, opportunistic Grid model,

several core assumptions are invalidated, and workarounds are needed to make the system work. This section explains the most pressing ones.

4.1. Communication security

Most Condor installations around the world use unauthenticated network traffic and rely solely on IP based restrictions for securing their pools. While this is acceptable in tightly controlled environments, it is obviously not an option for a world-wide virtual pool like the only glideinWMS provides.

Condor does provide strong authentication methods, and supports both integrity checks and encryption. glideinWMS supports the GSI authentication, also known as x509 authentication, being the current Grid authentication mechanism. However, the added security increases quite a bit the resource consumption of the Condor daemons, and consequentially limits is scalability. Until very recently, the Condor team did not spend any significant time trying to optimize for the fully authenticated scenario, since the vast majority of its clients were not using it. This has recently changed, and there are good signs that optimizing for fully secured pools could become a priority.

4.2. Working over NATs and firewalls

Condor has been developed in a network friendly environment, bidirectional network traffic was not a problem. In the present Grid world, a significant fraction of computing resources are behind a NAT or have some sort of firewall in place. As shown in figure 3 and figure 5, the schedd needs to talk to the startd in order for a job to start, so a NAT or a strict firewall at the Grid pool border will not allow such communication to occur.

To work in such environment, Condor recently introduced the Generic Connection Broker (GCB). In this scenario, the startd first establishes a long lived TCP connection with a GCB sitting on a public network node, and GCB associates it with a dedicated TCP port. The startd then communicates to the central manager the obtained GCB port instead of the local TCP port. When anyone needs to talk to the startd, they will contact the GCB server that will relay the network traffic to the startd. Since the long lived TCP connection was initially originated from inside the NAT and/or firewall, all successive traffic, no matter the direction, is permitted from then on.

See figure 7 for a schematic overview.

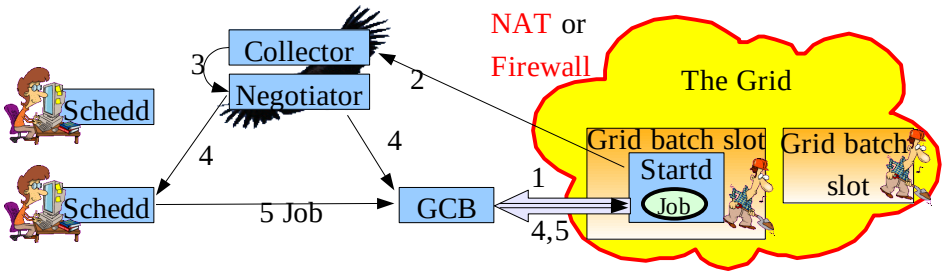

Figure 7. Using GCB to bridge firewalls

4.3. Running the startd as a non-privileged user

Condor startd is usually installed and run as a privileged user. This way, once a user job is received, the user job is run under a different user identity, protecting the startd from the user job. In the Grid

world, however, the startd will be started as a regular user, and will not be allowed to change the identity of the user job.

This has significant security consequences. Since all the Condor traffic is secured by using only the pilot's credential, a malicious user could steal that credential and mimic the startd. This will allow him to further user jobs, potentially stealing sensitive information and corrupting other user's data. So only trusted users should be run in this scenario.

One way to address this problem is to use a system provided tool like gLExec. gLExec is a Apache suExec derivative that, given a GSI proxy and an executable, switches the user identity and runs the given executable under that identity. See figure 8 and figure 9 for a schematic overview.

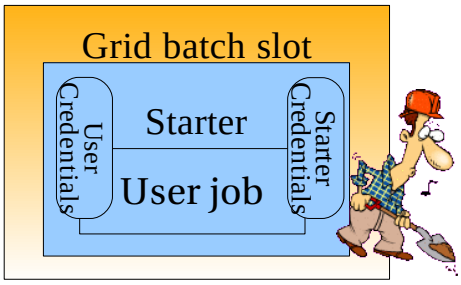

Figure 8. Startd running as a regular user

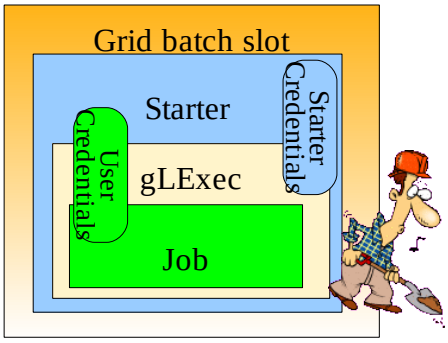

Figure 9. Using gLExec for identity switching

gLExec is currently deployed only at the Fermi National Laboratory, but is scheduled to be deployed on most sites on the US Grid, called the Open Science Grid (OSG), over the next couple of months. It is currently being tested by the OSG integration testbed.

5. glideinWMS monitoring

In order to be useful, a WMS must be fully monitorable. Being based on Condor, the glideinWMS inherits all the Condor monitoring tools needed by the users; condor_q for user job monitoring, condor_status for resource monitoring and CondorView for Web based job and resource monitoring.

The glideinWMS can also use the standard Condor monitoring tools to monitor both the glidein factories and the VO frontends; both the factory and the frontend daemons publish in their status using the ClassAd machanism.

Additionally, the glideinWMS maintains a historical memory of the daemon states, using RRDTool Round Robin Database files. This information is then converted into graphics images and published on a Web page.

6. Scalability considerations

The glideinWMS has been benchmarked in a test environment to find scalability issues:

- Central Manager Machine – No scalability limits have been found up to 5k running glideins.
- Submit Machine – No scalability limits have been found up to 90k queued jobs and 5k running jobs using a single schedd on the machine. The memory consumption scales linearly with the number of running jobs, at approximately 1.5M per running job.
- Glidein Factory Machine – A single machine can handle up to approximately 50 Grid pools. The main limit is CPU usage imposed by the glidein factory monitoring.

- VO Frontend Machine – No limit has been found up to 120k queued user jobs, serving 6 schedds and handling 50 Grid pools.
- GCB – A single GCB machine can handle around 700 glideins. It is currently not known why it does not scale further, as the system load stays low on the GCB machine even with 1k glideins, and no internal GCB limits seem to be reached.

None of the above results limit the scalability of a glideinWMS instance, but multiple nodes are needed to scale to 5k glideins distributed over 100 Gird pools, and beyond.

7. Advantages of glideinWMS over other Grid-wide WMSes

There has been several Grid-wide Workload Management Systems developed in the past. The most well known general purpose ones are the OSG ReSS and the gLite WMS. Several other groups have developed group specific WMSes, some examples being CDF GlideCAF, LHCb DIRAC and ATLAS PANDA. It is interesting to notice that most group-specific WMSes are pilot based, while most general purpose ones are not.

The reason why pilot based systems are preferred by most final users lies in the advantages provided by the pilot infrastructure:

- *Late binding.* Pilots are sent to all suitable Grid sites. Only once pilots start are real jobs selected for that resources. No forecasting is needed.
- *Reliability*. A broken Grid site will either kill pilot jobs or pilots will detect the problem at startup. Real jobs only start on well-behaved resources.
- *Grid-wide fair share.* The relative priorities between jobs of the same VO are set inside the WMS. Grid sites only manage priorities between different VOs.

The glideinWMS inherits all the advantages of the pilot infrastructure, but is also a general purpose WMS. Moreover, being based on a mature yet active batch system like Condor, it benefits from the experience and manpower...

8. Summary

The glideinWMS is a general purpose pilot based Grid wide WMS. It is based on the Condor batch system with only a thin layer on top of it.

The glideinWMS has shown to scale well and has already been used inside CMS by a small group of people. In the near future, CMS is planning to use it for both organized data reconstruction and Monte Carlo production, as well as for user analysis. Although glideinWMS will not be the only WMS used inside CMS, it is expected to handle a signification fraction of CMS's jobs.

CDF is also planning to integrate it inside its GlideCAF infrastructure.

Acknowledgements

The glideinWMS development has been funded by USCMS, and was performed at the Fermi National Laboratory.

The original Condor glidein based WMS concepts were first developed in the context of the CDF GlideCAF[3], with considerable help from the Condor team. The constant pressure concept comes from the ATLAS PANDA project.

References

[1] Foster I and Kesselman C 1998 *The Grid: Blueprint for a New Computing Infrastructure*. (San Francisco, CA: Morgan Kaufmann Publishers)

[2] Thain D, Tannenbaum T and Livny M 2005 Distributed Computing in Practice: The Condor Experience *Concurrency - Practice and Experience* **17 2-4** 323-56

[3] Belforte S, Norman M, Sarkar S, Sfiligoi I, Thain D and Wuerthwein F 2006 Using Condor Glide-Ins and Parrot to Move from Dedicated Resources to the Grid *Lect. Notes in Info.* **81** 285-92