FERMILAB-CONF-09-373-CD

The pilot way to Grid resources using glideinWMS

Igor Sfiligoi [a 1], Daniel C. Bradley [b 2], Burt Holzman [a 3], Parag Mhashilkar [a 4],
Sanjay Padhi [c 5], Frank Würthwein [c 6]
[a] Fermilab, Batavia, IL, USA, [b] University of Wisconsin, Madison, WI, USA
[c] University of California at San Diego, California
email: [1] sfiligoi@fnal.gov, [2] dan@hep.wisc.edu, [3] burt@fnal.gov, [4] parag@fnal.gov,
[5] Sanjay.Padhi@cern.ch, [6] fkw@ucsd.edu

Abstract

Grid computing has become very popular in big and widespread scientific communities with high computing demands, like high energy physics. Computing resources are being distributed over many independent sites with only a thin layer of Grid middleware shared between them. This deployment model has proven to be very convenient for computing resource providers, but has introduced several problems for the users of the system, the three major being the complexity of job scheduling, the non-uniformity of compute resources, and the lack of good job monitoring.

Pilot jobs address all the above problems by creating a virtual private computing pool on top of Grid resources. This paper presents both the general pilot concept, as well as a concrete implementation, called glideinWMS, deployed in the Open Science Grid.

1. Introduction

Grid computing is being widely deployed by big and widespread scientific communities with high computing demands, like high energy physics. Examples of such grids are the Open Science Grid (OSG)[1] and the European Grid for E-SciEnce (EGEE)[2]. While Grids ease the deployment and operation of computing resources, they also bring new challenges for the users of those resources. Section 2 provides an architectural overview of the Grid paradigm, together with a schematic overview of its strengths and weaknesses.

User groups have approached the Grids using different paradigms. Section 3 presents the pilot paradigm, arguably one of the most user friendly ones, presenting its strengths and weaknesses, and comparing it to other established paradigms.

Section 4 presents the glideinWMS, a concrete implementation of the pilot paradigm. Finally, section 5 describes how far the glideinWMS is known to scale.

2. Grid overview

The Grid paradigm is based on the distributed computing paradigm, but spanning many administrative domains. A Grid deployment, like the OSG, is composed of several independent Grid sites, each maintaining a locally managed distributed computing system, with just a thin middleware layer shared between the sites that handles inter-domain authentication.

The Grid middleware is typically composed of a portal to the site resources, also known as "*the gatekeeper*" or the "*Compute Element*", and a set of command line tools on the compute resources themselves, also known as "*worker nodes*", to be used by the user jobs to talk to other Grid services. Everything else, including the local workload management system (WMS) and the operating system, are left to the site administrator's discretion. For a graphical overview, see Figure 1.

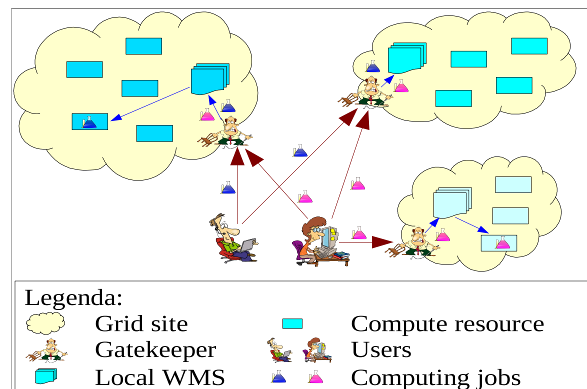

Figure 1. Grid overview

The Grid paradigm is very convenient for the resource providers. They can continue to operate the local distributed resources according to local preference and expertise, integrating them easily with other, non-Grid resources. Moreover, the decentralized nature of the Grid allows for easy scalability as one just needs to split a site into multiple logical pieces if scalability becomes an issue.

For users, however, this represents a step backwards:

- Instead of having a single system to deal with, they have many; workload management issues become a serious problem.
- Monitoring of the submitted jobs is also a problem; the Grid middleware abstracts the monitoring of the actual local WMSes running on the Grid sites, thus providing only a subset of the monitoring data.
- The final problem the users face is the variety of configuration setups between compute resources; users need to provide complex setup wrappers to successfully run their jobs.

3. The pilot paradigm

The pilot paradigm, also known as the "*just-in-time*" model, tries to simplify users' life by creating a virtual private pool of compute resources on top of the Grid.

In this scenario a pilot WMS, composed of a job queue and a pilot factory, is set up for a group of users. When a user submits one or more compute jobs to the pilot queue, the pilot factory sends pilots to all the suitable Grid sites. Once a pilot job starts running, it fetches a compute job from the pilot queue and the compute job starts to run. For a graphical overview, see Figure 2.

With the introduction of a pilot WMS, users have a single reference point, making the job submission much simpler; users don't need to keep an updated list of Grid sites anymore, the pilot WMS keeps it for them. Monitoring is also much improved; the pilot jobs running on the worker nodes can gather and report any information the pilot WMS deems important for the users.

The pilot WMS can also partially hide the heterogeneity of the compute resources. Since the pilot jobs start before user jobs, they can prepare the environment for the fetched user job. Changes like installing a set of software libraries or defining certain environment variables are easy to do.

Obviously, a pilot job cannot do miracles, like changing an instruction set of the CPU. In those cases, the pilot job will simply refuse to fetch any user job and will just terminate. Such instances should be rare, and mostly due to misconfiguration. And while the resource providers may complain about the wasted resources, the users' jobs, at least, will be protected from incompatible environments.

3.1. Identity management with pilots

The pilot paradigm does not fit well within the original Grid authentication and authorization model. As can be seen in Figure 2, the real user identity is never communicated to the Grid site, because the user job never traverses the gatekeeper.

On Grids that rely on OS user identity protections, like is the case for OSG and EGEE, the pilot operation model also has security implications, since the pilot job is not a privileged process and thus cannot change the OS user identity after fetching a user job. This is problematic for two reasons. First, the pilot job and the user job run under the same OS identity envelope, allowing a malicious user job to compromise the pilot job infrastructure. Additionally, when several jobs are running on the same worker node, jobs from different users will run under the same OS identity envelope, allowing a malicious user to compromise the jobs of other users.

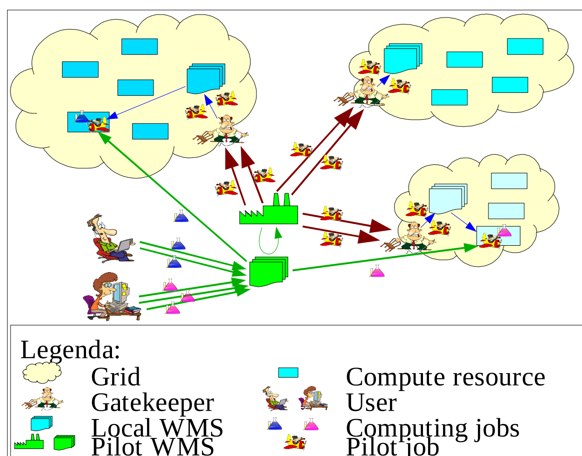

Figure 2. Pilot paradigm overview

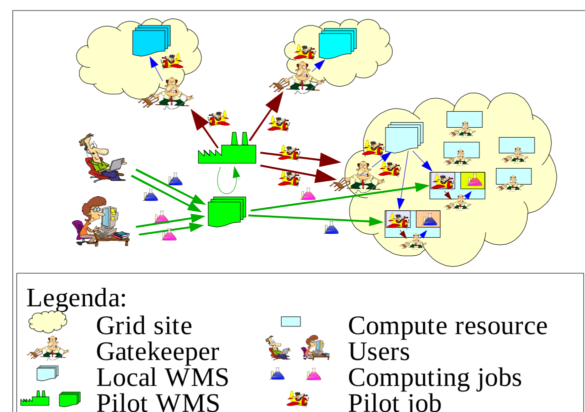

Figure 3. Identity switching with mini-gatekeepers

To address these problems, several Grids have started to deploy mini-gatekeepers on the worker nodes themselves. These mini-gatekeepers will identify the user fetched by the pilot and execute the user job under appropriate OS user identity. OSG and EGEE use gLExec[3] for this purpose. For a graphical overview, see Figure 3.

3.2. Comparison with other paradigms

Another approach is the so-called push, or oracle model. This model deploys a WMS that gathers the user jobs and forwards them to the Grid site that has the highest probability of running them sooner.

From the user point of view, the submission and monitoring experience is very similar to the pilot paradigm; they see a single job queue. Moreover, the push WMS can also wrap the user jobs in appropriate wrappers, hiding most of the compute resource heterogeneity.

However, at the technical level, the push model has several problems. First, deciding where to send the user jobs is far from being a trivial task. The WMS has to make an educated guess, since the available monitoring data do not provide enough information for a precise measurement; this is what earns it the "*oracle*" name. Next, the priorities between users are handled independently by each Grid site; while the push WMS can leverage that information (if known) to send user jobs to the sites with better priorities, it has no way to influence them. A pilot WMS suffers neither of these problems, as it uses only the pilot identity in communication with Grid sites, and handles priorities between user jobs internally.

The push WMSes also fare worse than pilot WMSes in monitoring, as they can only rely on information provided by the Grid middleware.

The main disadvantage of pilot-WMSes over push-WMSes is the increased resource utilization of a WMS. A push WMS only needs to make a site-selection decision and hand the user job to the remote Grid site. A pilot-WMS, instead, must handle pilot jobs, user jobs and the handling and monitoring of the resources provided by the pilot jobs.

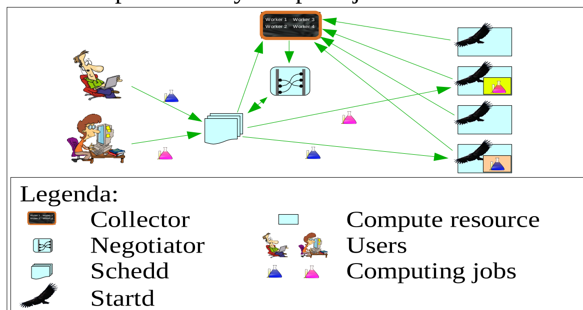

Figure 4. Condor overview

4. glideinWMS overview

Several pilot WMSes have been implemented, but this paper describes the pilot based WMS called glideinWMS. It implements the pilot factory code and uses Condor as a user job workload management system.

4.1. Condor overview

Condor[4] is a mature yet still actively improved workload management system. It started as a batch system for harnessing idle cycles on personal workstations[5], but has since become a major player in the dedicated compute resource area.

A Condor pool is composed of several logical entities:

- The central manager gathers and keeps information about the other nodes. A process called "*collector*" acts as a dashboard.
- Execute nodes provide compute resources. On each execute node a process called "*startd*" manages the compute resource. The startd is also responsible for publishing the characteristics of the managed resource, including the current status as well as CPU type, amount of memory, system load, etc.
- Submit nodes handle the user jobs. On each submit node a process called "*schedd*" maintains the job queue.
- The matchmaker matches compute resources to user jobs. A process called "*negotiator*" is responsible for this task.

The communication flow in Condor is fully asynchronous. Each startd and each schedd advertise to the collector using their own schedule. Similarly, the negotiator starts a matchmaking cycle using its own timing.

The resource allocation is completely driven by the negotiator[6]; in each cycle, it retrieves the characteristics of user jobs from the schedds and matches them against the idle resources. All the matches are then ordered based on user priority and communicated back to the schedds, who in turn transfer the matched user jobs to the affected startds for execution. To fairly distribute the resources among users, the negotiator tracks resource consumption by the users and calculates user priorities based on that.

The above description is necessarily just an approximation; the interested reader should read the Condor documentation for further details. For a graphical overview, see Figure 4.

As reader might notice, Condor assumes bi-directional networking between all the nodes. However, most networks today employ some kind of firewall, making this deployment unfeasible

anywhere but in tightly controlled LAN environments.

To obviate this limitation, Condor provides a proxying service, called "*GCB*"[4,7]. With the use of GCBs, only the GCBs themselves need to accept incoming connection, all other communication can be routed through them.

Detailed description of GCB operation is beyond the scope of this document and the interested reader should refer to the GCB documentation.

4.2. Condor glideins

Having started as a scavenger batch system, Condor is an excellent choice for a pilot WMS; all a pilot job needs to do is configure and start a startd.

Condor itself provides a basic pilot job infrastructure, in the form of a command line tool, called "*condor_glidein*"[8]. This tool creates a pilot job, a "*glidein*", and submits it to the Grid. Thus, with some effort, a power user can create a pilot WMS.

4.3. glideinWMS overview

A real pilot WMS needs an autonomous pilot factory, so that users are fully insulated from the Grid complexities. glideinWMS[9] provides a pilot factory for a Condor based WMS.

The glideinWMS pilot factory is composed of three logical entities:

- A dashboard used for message exchange. A standard Condor collector is used for this task.
- Glidein factories create and submit pilot jobs to the Grid. On each glidein factory a process called "gfactory" is responsible for this, and uses Condor-G[8] as a Grid submission tool.
- VO frontends monitor the status of the Condor WMS and regulate the number of pilot jobs sent by the glidein factories. A process called "*frontend*" does the matchmaking.

The above description is necessarily just an approximation; the interested reader should read the glideinWMS documentation for further details. For a graphical overview, see Figure 5.

The separation of tasks allow a glideinWMS pilot factory to be shared by several Condor WMSes. Each frontend can be configured to monitor one Condor WMS, while the gfactories can stay the same; a gfactory is responsible for managing the Grid resources independently of who asks for them.

The separation of tasks also allows for better scalability; the only element that cannot be replicated is the dashboard. If either the frontend or the gfactory become overloaded, they can easily be split and distributed over two or more nodes.

5. glideinWMS scalability

The main selling point of the Grids is their distributed nature, which naturally avoids global scalability issues. Pilot solutions, and glideinWMS in particular, use a more centralized approach and thus are intrinsically less scalable.

As explained in the previous section, the glideinWMS is composed of several loosely coupled processes, so a scalability limit in a component can be solved by deploying many instances. The only element that cannot be replicated is the collector.

However, from a practical point of view, one is interested to see how far the system can scale with a single schedd, as well as how far can it scale with a single glideinWMS pilot factory instance, since this is the simplest possible installation.

The short-term design goal of glideinWMS was to handle O(10k) batch slots from O(100) Grid sites distributed all over the planet, and with O(100k) jobs in the queue with an average startup rate of O(1 Hz), so a set of tests aimed at reaching these goals was performed.

5.1. Collector scalability

To test the collector scalability, the central manager node was installed in Europe, with pilots being run at a few test Grid sites in the US. This setup was chosen to mimic the worst case scenario, network wise. Condor version 7.1.3 was used.

The major scalability limit we found was related to initial security handshake paired with high network latencies. Under these conditions, a single collector cannot cope with more than approximately 6 pilots starting per minute.

To obviate this, a tree of collectors was installed, with a master collector and 20 slave collectors. With

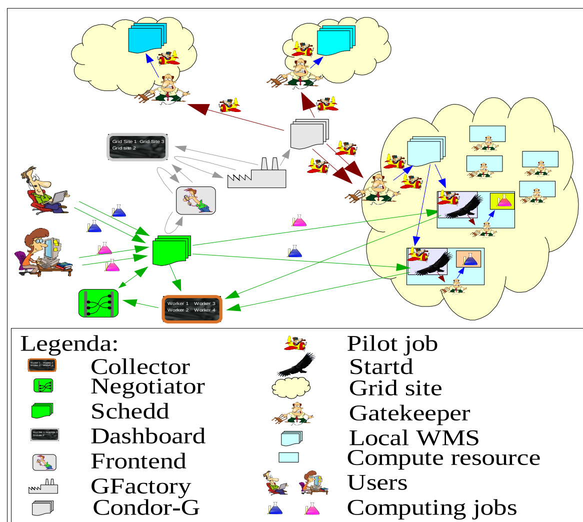

Figure 5: glideinWMS overview

all the collectors on the same physical node, no authentication is needed between them.

With this setup, the main collector was able to handle 12k startds, with a peak pilot joining rate of 2Hz. The system was kept above 10k for more than 48 hours and there were no signs that it could not handle more startds.

5.2. Schedd scalability

The same setup used for collector testing was used for schedd scalability, with the addition of GCBs to bridge NATed worker nodes. The schedd was installed on the same node as the 20+1 collectors, while the GCBs had dedicated nodes.

The schedd scaled to approx. 11k running jobs, with approx. 100k jobs in the queue before running out of system resources. The main resource bottleneck was memory; by the time Condor was handling 11k running jobs, all 16GB of memory were being used and the system started to swap heavily.

We left the system running at approx. 10k level for 43 hours. In that period of time, the system ran 220k jobs, giving us 1.4Hz startup rate. All jobs finished successfully, although approx. 2% were started more than once to recover from job-handling failures.

Since we reached the target design goal, we did not try to scale it further. However, it is our belief that by doubling the memory we should be able to double the number of handled jobs.

5.2. glideinWMS pilot factory scalability

The glideinWMS pilot factory scalability test used a different setup. All of the glideinWMS components were installed in the US, and the glideins were being sent to O(100) production Grid sites in the US and in Europe. The load on the schedd and the collector was much smaller than in the other tests, with an average of 1k running and 10k queued jobs, with peaks of 3k running and 50k queued. The system was being used for more than a month.

The gfactory, Condor-G and the glideinWMS dashboard were hosted on one node, the frontend shared a node with a GCB, the schedd was installed on a dedicated node, and a tree of collectors was using another dedicated node.

The dashboard and the frontend never reached a scalability limit. Instead, the gfactory did.

The gfactory ran into IO transaction limits when serving more than approx. 50 Grid sites. The load comes mostly from the administrative monitoring subsystem of the gfactory. By disabling part of the monitoring, an option available since glideinWMS v1_4, the gfactory was able to handle the available 120 Grid sites, but the load on the system was still uncomfortably high. Further improvements to the glideinWMS code are envisioned to improve this.

6. Summary

Grid computing can help in the deployment of large amounts of compute resources, but presents severe drawbacks for the final users. The pilot reduces the users' problems by creating a virtual private pool on top of Grid resources, thus preserving the relative ease of management and monitoring typical of local compute resource pools.

glideinWMS is one implementation of the pilot paradigm. The base WMS is handled by the Condor WMS, while the pilot factory is provided by the glideinWMS software. The system has been tested to the design goal of handling O(10k) batch slots from O(100) Grid sites distributed all over the planet, with O(100k) jobs in the queue with an average startup rate of O(1 Hz).

7. Acknowledgements

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

The paper was partial sponsored by the National Science Foundation under Grant No. PHY-0533280 (DISUN) and PHY-0427113 (RACE).

8. References

[1] R. Pordes, et. al., “The open science grid”, Journal of Physics: Conference Series 78, IoP Publishing, 2007 (15pp)
[2] “EGEE Home”, http://www.eu-egee.org/
[3] I. Sfiligoi et. al., “Addressing the pilot security problem with gLExec”, Journal of Physics: Conference Series 119, IoP Publishing, 2008 (6pp)
[4] “Condor home page”, http://www.cs.wisc.edu/condor/
[5] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations", *Proc. of the 8th Int. Conf. of Dist. Comp. Sys.*, June, 1988, pp 104-111.
[6] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", *Proc. of the 7th IEEE Int. Symp. on High Perf. Dist. Comp.*, Chicago, IL, July 28-31, 1998
[7] B. Beckles, S. Son, and J. Kewley, "Current methods for negotiating firewalls for the Condor system", *Proc. of the 4th UK e-Science All Hands Meeting 2005*, Nottingham, UK, September 19-22, 2005.
[8] J. Frey et. al., "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Journal of Cluster Computing*, 2002, vol 5, pp. 237-246.
[9] “glideinWMS home page”, http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/