Adapting to the Unknown With a few Simple Rules: The glideinWMS Experience

Igor Sfiligoi, Benjamin Hass, Frank Würthwein
University of California San Diego
La Jolla, CA 92093, USA
email: {isfiligoi,bhass,fkw}@ucsd.edu

Burt Holzman
Fermi National Accelerator Laboratory
Batavia, IL 60510, USA
email: burt@fnal.gov

***Abstract*—The High Energy Physics community requires a large amount of compute resources and had to adopt the Grid computing paradigm to satisfy its needs. Scheduling of jobs in the Grid environment is however very challenging, due to the high autonomy enjoyed by the participating resource providers, requiring the user community to constantly adapt to the ever-changing conditions. The CMS experiment addressed this problem by developing the glideinWMS system, which addresses this problem by using an overlay compute pool and a few simple rules for provisioning the needed resources. This approach has been very successful and is now being used by several other communities in the Open Science Grid. This paper provides the description of the glideinWMS system, the algorithms used as well as an analysis of the experience CMS has had using the system.**

Keywords-layered scheduling; adaptation; glideinWMS.

I. INTRODUCTION

Over the past decade, the high throughput computing in science has been moving from dedicated compute clusters to a widely distributed, shared Grid infrastructure in an effort to distribute the system maintenance effort, increase the average equipment utilization and gather additional compute resources in times of need. This paper explores the challenges of doing Grid-wide user job scheduling in this environment.

One of the core principles of the Grid paradigm that makes it so appealing for the scientific resource providers is the high autonomy enjoyed by each participating compute cluster, allowing them to participate in the system without sacrificing neither the quantity nor the quality of compute resources given to the local users. As a consequence, while the external, opportunistic users can request to use compute resources, they have no guarantee if and when those compute resources will become available.

Another core principle that makes the Grid paradigm appealing to the user community is, instead, the freedom users retain to schedule the compute resources the way they deem more fit; the users are free to choose the product they like, or for example, only submit to local resources due to the perceived ease of use. As a consequence, there cannot be a single Grid-wide job scheduler instance, and there is also no guarantee that the various instances will exchange information with one another.

A Grid-wide scheduler is thus unlikely to ever obtain an accurate state of the whole system, much less be able to predict what the state of the system will be in the near future. Being able to adapt to the ever-changing situation is thus essential.

This paper describes the approach taken by the glideinWMS system [1,2], a scheduling solution developed by the Compact Muon Solenoid (CMS) experiment [3] and extensively used in the Open Science Grid (OSG) [4,5]. The key component of this system is the conceptual simplicity of the approach; the user scheduling is solved by the use of an overlay compute pool and the resource provisioning is handled by just a few simple rules.

A schematic description of the system architecture is provided in Section II, while Section III provides a detailed description of the resource provisioning rules. Section IV provides an analysis of the experience CMS has had using the system. Finally, Section V provides a comparison against other Grid-wide scheduling systems.

II. THE GLIDEINWMS ARCHITECTURE

The glideinWMS approach to Grid-wide user job scheduling is based on the pilot paradigm. In this paradigm, the scheduling system does not even try to directly schedule the user jobs on Grid resources, but instead creates a dynamic overlay pool of compute resources on top of the Grid resources, by submitting so-called **pilot jobs**, and then schedules user jobs inside this pool. A schematic view of a pilot system is shown in Fig. 1. More details can be found in [1].

The pilot jobs are effectively resource provisioners; once one of them starts on a Grid resource, it takes ownership of that resource for the allocated lease time, and gives it in exclusive use of the pilot system, by joining the overlay pool. The pilot system scheduler thus has complete control over this overlay pool, and can make scheduling decisions based on trustworthy information obtained from the provisioned resources, just as in a truly dedicated compute cluster.

The pilot system, of course, now needs to schedule the pilots themselves across all the Grid resources, and do this with only partial information. Due, however, to the nature of pilot jobs, this task is much simpler than direct Grid-wise user job scheduling. Unlike user jobs, all pilot job payloads are the same, and thus the order in which they start is irrelevant. Moreover, each and every pilot can run jobs from any user of the community, relinquishing the need of inter-

 ISBN: 978-1-61208-156-4

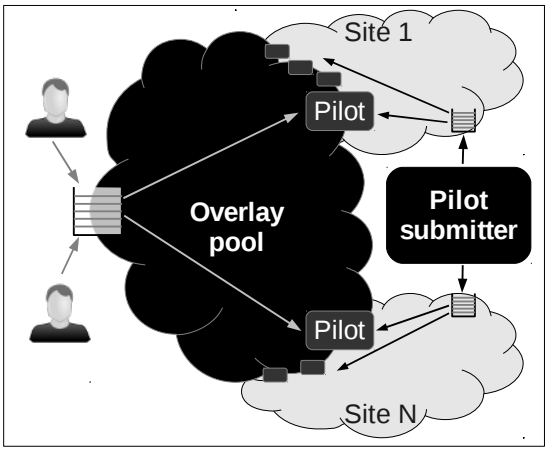

Figure 1. A pilot system

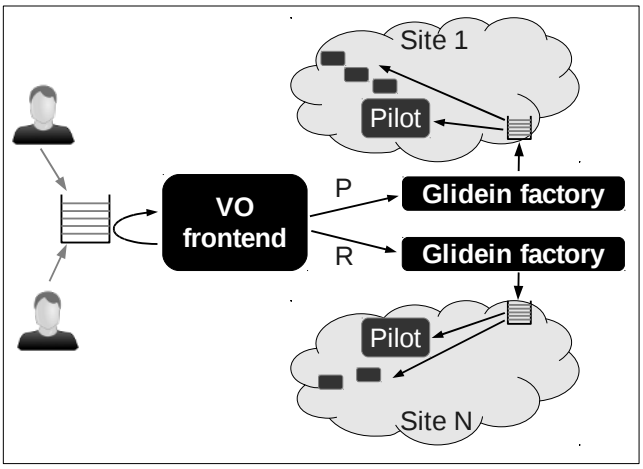

Figure 2. The glideinWMS pilot submission

pilot priorities. The glideinWMS scheduling algorithms exploit these properties and do not track the single pilot jobs themselves, but only monitor and regulate the cardinality of pilot jobs in the queues.

Somebody, however, still has to actually submit and track the single pilot jobs as they are submitted to the Grid sites. In the glideinWMS system this is delegated to a set of processes called **glidein factories**, one per logical Grid site, which are essentially just slaves to the actual scheduling process, called the **VO frontend**, as shown in Fig. 2.

The communication between the frontend and a factory is based on the concept of **constant pressure**; the frontend asks the factory to keep a certain number of pending, or idle pilot jobs in the Grid queue, and to continue to submit new ones to replace the ones that start running, until the frontend issues a new request changing that number, possibly to zero. In Fig. 2, the pressure numbers are represented by letters P and R.

The adaptability of the system thus lies in the calculating, at any given point in time, the appropriate pressure point for each and every Grid site. If the pressure is too low, there may not be enough idle pilots in a Grid site's queue when compute resources at the site become available, resulting in a smaller overlay pool and thus lower user job throughput. If instead the pressure is too high, the Grid resources added to the overlay pool may not be needed anymore by any user job, resulting in wasted CPU cycles.

III. Regulating the Pilot Pressure

The pilot jobs are being submitted to Grid sites because there is an expectation that when they do start up and provide compute resources to the overlay pool, there will be user jobs that can make use of them. However, in order to accurately forecast the available jobs at pilot startup, the system would need to know the current state of the users' job queue, the Grid site scheduling policies, the behavior of all other Grid-wide schedulers, the behavior of the local users and the run times of the users' jobs. Only the first one is available to the glideinWMS VO frontend.

Most of the logic is thus based on the current state of the users' job queue. At each iteration, the VO frontend collects the information about all the idle user jobs and matches them against the information available about the Grid sites, similarly to how the matchmaker in the overlay pool would do it. The details of what information is available and how is the matchmaking performed is beyond the scope of this document, and the interested reader should refer to the glideinWMS manual [6] instead.

Simply calculating the number of idle user jobs for every Grid site is however not enough. If a user job is capable of running on more than one Grid site, simply counting each job against each matching site will result in double counting for some. The VO frontend thus keeps track of how many Grid sites each job matches against, and counts each job as only the appropriate fraction against each matched site. As an example, if a job matches against sites A, B and C, it will be counted as 1/3 against each of them.

Once the weighted count of matched idle user jobs is computed, the VO frontend calculates the pressure point for each Grid site as a function of the number of matched idle user jobs, as in

$$P_s(t)=f(I_s(t)). \quad (1)$$

As stated above, knowing the current state of the users' job queue is not enough to obtain the optimal value. However, given that there is no easy way to obtain the vast majority of data needed for a reliable forecast, the VO frontend does not even try. Instead, the VO frontend uses a simple heuristic to achieve the desired result.

In our multi-year experience of using the Open Science Grid, we noticed that Grid jobs tend to start and terminate with a relatively flat frequency. Most Grid sites will start and terminate O(10) jobs every few minutes, and it is very rare to have O(100) Grid jobs terminate in the same period. As such, having a pressure point of O(10) is sufficient to get access to the vast majority of available Grid resources.

With the maximum pressure point capped at O(10), the remaining range is small enough to not require fine tuning. The VO frontend thus simply divides the number of currently idle user jobs by 3, resulting in the following formula:

$$f(I_s(t)) = \lceil min(I_s(t)/3, C_s) \rceil. \quad (2)$$

 ISBN: 978-1-61208-156-4

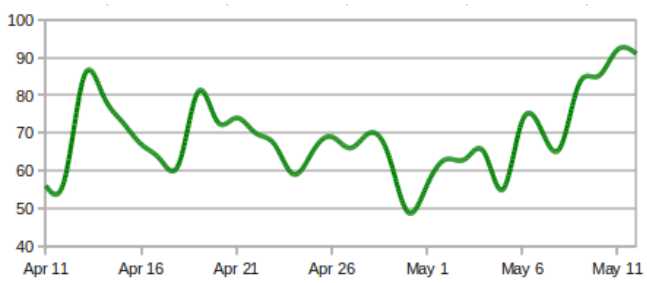

Figure 3. Number of Grid sites used by the CMS glideinWMS system

The factor 3 was chosen pretty arbitrarily, but following the simple logic that the pressure point should be lower than the number of idle jobs, in order to not over-provision, and that the fraction should still be high enough to obtain the desired amount of Grid resources at an acceptable rate.

IV. Operational experience

The CMS experiment has been using the glideinWMS system for over two years and has been generally very satisfied with the experience. The glideinWMS instance at UCSD is serving a user community of about 4k users and scheduling their jobs on Grid resources distributed across about 100 Grid sites located in the Americas, Europe and Asia. The actual numbers of Grid sites used by user jobs in a recent month can be seen in Fig. 3.

As simple as the algorithms described above are, the system proved to be very effective and efficient. Fig. 4 contains the status of the users' job queue in a recent month, both idle and running. As can be seen, the CMS users have been using up to about 16k CPUs in that period, with steep ramp-ups and ramp-downs based on user jobs demand. This resulted in short wait times for user jobs; as shown in Fig. 5, most jobs started within an hour.

Furthermore, as an indicator of the overall system efficiency, the amount of over-provisioned resources, labeled as *Unmatched* in both Fig. 4 and Fig. 6, has been consistently very low and typically represented less than 5% of all the provisioned resources.

The glideinWMS systems has also been recently adopted by several other OSG communities [7], with similar results. It is worth noting that the addition of several other independent glideinWMS frontend instances has had no impact on the performance of the CMS frontend.

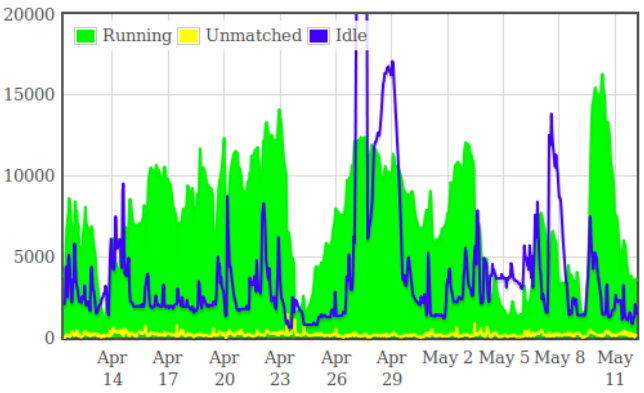

Figure 4. Snapshot of the CMS glideinWMS system

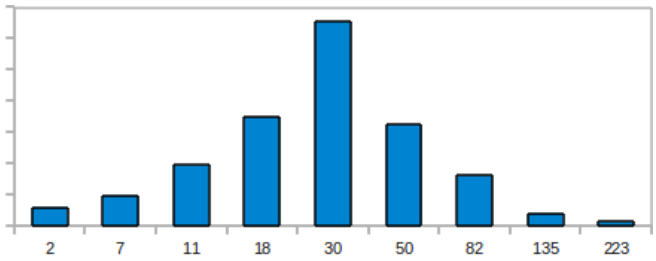

Figure 5. CMS user jobs wait times distribution, in minutes

V. Related work

There are several other products that provide Grid-wide scheduling. They can be categorized as being either direct-submission or pilot systems.

Two major direct-submission systems are the gLite Workload Management System (gLiteWMS) [8] and the Resource Selection Service (ReSS) [9] based OSG Matchmaker (OSGMM) [10]. Compared to glideinWMS, both require a more complex setup by requiring continuous information flow from each and every Grid site. This approach also is less flexible and more brittle, since the information source is controlled by the site, and thus cannot be influenced or verified by the scheduling system; this is not a problem for pilot-based systems like glideinWMS, because the needed information is collected directly by the pilots themselves.

Major pilot-based systems are PanDA [11], DIRAC [12] and MyCluster [13].

The PanDA approach to scheduling of pilots to Grid sites is even simpler than the glideinWMS approach; the system continuously submits pilots to all available Grid sites. If there are no suitable user jobs available at pilot startup, it quickly exists, wasting very little wallclock time. The major drawback of this approach is the high load it imposes on the batch system of every Grid site. By contrast, the glideinWMS system only sends out pilots that are expected to be needed by user jobs, although it too will quickly terminate them if the submission logic was faulty and there are no suitable user jobs available.

DIRAC and MyCluster rely instead on separate services running at each Grid site. This allows them to gather detailed information about the Grid sites, and thus making a more informed decision. The major drawback of this approach is

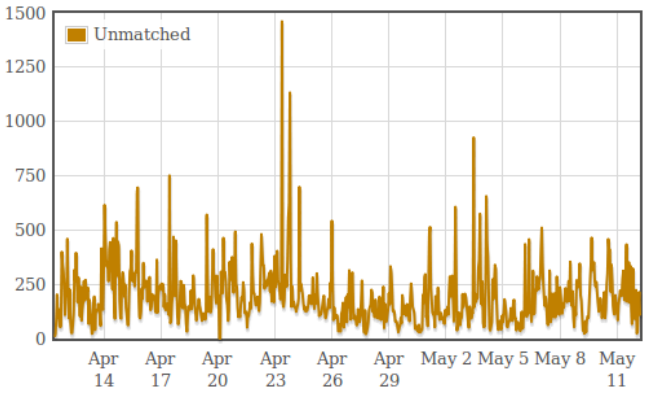

Figure 6. Over-provisioned resources in the CMS glideinWMS system

 ISBN: 978-1-61208-156-4

the assumption that Grid sites will allow the Grid schedulers to install long-lived services at the sites; while this is possible at some sites, many others do not allow this option, thus severely limiting the amount of resources that can be gathered using those systems.

There are also several pilot-based scheduling systems that require the users to explicitly request pilots at various Grid sites. These manually-provisioned systems thus lack end-to-end automation, and a comparison to the glideinWMS would not be meaningful.

VI. Conclusions

The Grid computing environment has many advantages for compute resource providers, but does introduce significant challenges for effective user job scheduling. The main issue is the lack of complete information, which requires continuous adaptation of a Grid-wide scheduling system of each and every user community.

The glideinWMS system addresses this problem by reducing the scheduling complexity through the adoption of the pilot paradigm, where only the pilot jobs need to be scheduled across Grid sites. The user jobs are instead handled within the resulting well-behaved overlay compute pool.

Unlike user jobs, all pilot jobs carry the same payload. Together with the fact that there is only one pilot user, this allows the glideinWMS system to only consider the cardinality of the pilot jobs, drastically reducing the scheduling complexity.

Moreover, operational experience tells us that the Grid job startup frequency is very low, allowing for the capping of the pilot job pressure at an equally low number without significant loss in effectiveness. This makes scheduling of pilot jobs effectively trivial. And the years of operational experience CMS has with the glideinWMS system confirm that this approach works very well.

Using the pilot paradigm and looking only at the cardinality of the pilot jobs thus allow us to reduce the hard Grid-wide scheduling problem into a mostly trivial endeavor.

Acknowledgment

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

References

[1] I. Sfiligoi et al., "The pilot way to grid resources using glideinWMS," CSIE, WRI World Cong. on, vol. 2, pp. 428-432, 2009, doi: 10.1109/CSIE.2009.950.

[2] "glideinWMS," http://tinyurl.com/glideinWMS, Accessed June 2011.

[3] The CMS Collaboration et al. "The CMS experiment at the CERN LHC," J. Inst, vol. 3, S08004, pp. 1-334, 2008, doi: 10.1088/1748-0221/3/08/S08004.

[4] R. Pordes et al., "The open science grid," J. Phys.: Conf. Ser., vol. 78, 012057, pp. 1-15, 2007, doi: 10.1088/1742-6596/78/1/012057.

[5] "Open Science Grid Home page," http://www.opensciencegrid.org/, Accessed June 2011.

[6] "Glidein Frontend documentation," http://tinyurl.com/glideinWMS/doc.prd/frontend/configuration.html, Accessed June 2011.

[7] I. Sfiligoi et al., "Operating a production pilot factory serving several scientific domains," J. Phys.: Conf. Ser., in press.

[8] P. Andreetto et al., "The gLite workload management system," J. Phys.: Conf. Ser., vol. 119, 062007, pp. 1-10, 2008, doi: 10.1088/1742-6596/119/6/062007

[9] P. Mhashilkar, G. Garzoglio, T. Levshina, and S. Timm, "ReSS: Resource Selection Service for National and Campus Grid Infrastructure," J. Phys.: Conf. Ser., vol. 219, 062059, pp. 1-8, 2010, doi: 10.1088/1742-6596/219/6/062059

[10] "OSG MM - The Open Science Grid Match Maker," http://osgmm.sourceforge.net/, Accessed June 2011.

[11] T. Maeno, "PanDA: distributed production and distributed analysis system for ATLAS ," J. Phys.: Conf. Ser., vol. 119, 062036, pp. 1-4, 2008, doi: 10.1088/1742-6596/119/6/062036

[12] A. Tsaregorodtsev et al., "DIRAC: a community grid solution," J. Phys.: Conf. Ser., vol. 119, 062048, pp. 1-12, 2008, doi: 10.1088/1742-6596/119/6/062048

[13] E. Walker, J.P. Gardner, V. Litvin, and E.L. Turner, "Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment," CLADE, 2006 IEEE, pp. 95-103, 2006, doi: 10.1109/CLADE.2006.1652061

 ISBN: 978-1-61208-156-4