Grid Protocol and its negative impact on Factory Operations

J. Dost and I. Sfiligoi, UCSD

1. Introduction

The main purpose of the gatekeeper is to abstract away the details of the underlying batch system at a site. The gatekeeper does this by providing a uniform grid protocol layer that grid submission tools like HTCondor-G and globus-job-run can communicate with. When users submit to a site using these types of tools, they don't have to understand the batch system-specific commands needed to submit to and query the status of their jobs. This gives site admins the freedom to pick from a variety of popular batch systems: HTCondor, PBS, LSF, SLURM, etc., without affecting how the end user submits to their site. It is important to note however that the grid was supposed to have a single protocol, but in reality we have many: Globus, CREAM, Nordugrid, and most recently HTCondorCE.

In a joint collaboration between OSG and CERN, we operate and maintain the OSG Glidien Factory, a resource provisioning service. Our factory has been in production for over 3 years now and currently allows VOs to submit to 126 sites worldwide. While we have acquired in depth knowledge on how to submit to the various grid protocols, we have also learned about and struggled with the limitations of the traditional gatekeeper + batch system model that has become the standard for administering a grid site. We have come to find that the gatekeeper service itself is at fault for many of the problems we face in our day to day operations of the Glidein Factory. Ultimately, we propose that for provisioning systems such as GlideinWMS, it would be beneficial to allow the system to directly submit to the underlying batch system at a site, bypassing the gatekeeper service completely. This would greatly simplify the management of the provisioning service by avoiding the issues described in detail in the following sections.

2. GlideinWMS Overview

Before examining the deficiencies of gatekeepers and showing how they negatively affect the GlideinWMS provisioning system, we must first explain some details of how the GlideinWMS works, and define a few terms that will be used throughout this document. GlideinWMS is a pilot based provisioning system. The idea of a pilot system is to submit generic grid jobs to sites rather than user jobs. These generic jobs are known as “pilots” or “glideins” as we call them in GlideinWMS. Pilots reserve slots at a site for a temporary amount of time and make them available for user jobs to run on. The advantage of maintaining a pilot based system rather than direct grid submission is outside the scope of this paper. GlideinWMS manages the resources temporarily claimed by pilots by creating a virtual HTCondor pool. There are two parts of the system that work together to maintain the virtual pool: the “factory” and the “frontend.”

The factory is the component responsible for submitting glideins to sites. Under the hood it uses HTCondor-G for the submission. The factory configuration file contains all the details needed to submit a pilot using HTCondor-G to a particular site queue. It is important to note that pilots only temporarily claim resources at a site. For each site queue, the factory configuration file also defines “GLIDEIN_Max_Walltime,” which tells the glidein when to terminate. We usually configure the factory to set GLIDEIN_Max_Walltime to just under the max allowed wall time at the site queue (It is beneficial for glideins to have the longest possible wall time, but we don't want the site to hard kill the glideins before they have a chance to clean up).

The other component needed to maintain the virtual HTCondor pool, the frontend, monitors user jobs waiting to run in the virtual pool, and requests the factory to maintain a number of idle glideins at the site queues. This number is known as “Requested Idle.” The Requested Idle number for each site queue is proportional to user demand for that particular queue. The factory then takes this number and

compares it to the actual number of idle glideins in the HTCondor-G queues. If there are less idle glideins than what the frontend requests, the factory will submit more, until the actual number idle matches Requested Idle. Note that the factory and frontend services are usually run on different nodes, and are administered by different people from different institutions. The OSG factory for example is hosted redundantly at SDSC and IU, but each VO that uses the OSG factory runs its own frontend service to serve its user base and manage their own dedicated virtual HTCondor pool.

3. Glidein Submission Starvation

Glidein submission starvation is a result of the fact that gatekeepers are not accurately reporting the number of glideins running at the batch system. While we have observed inaccurate status reporting with all grid protocols (excluding HTCondorCE since we don't have a production site running it yet), it will be shown in section 4 that Globus gt5 is the worst offender by orders of magnitude. We are able verify gatekeeper status inaccuracy in two ways. First, the frontend provides the factory with a view of the number of glideins running at a site that is completely independent from communication with the gatekeeper. It is able to do this by periodically polling the virtualized HTCondor pool using condor_status and counting the registered glideins. It sends this information periodically to the factory along with its Requested Idle value updates. If we compare the number of registered glidiens as reported by the frontend to the number of running pilots as reported by the gatekeeper, we see that these numbers do not match. The second way to verify gatekeeper accounting inaccuracies is by comparing the glidein runtime as reported by HTCondor-G, vs. the GLIDEIN_Max_Walltime limits we have set in the factory for the specific site. In doing so we discover that quite often when querying the factory HTCondor-G queues, glideins will be reported as running O(10) days, whereas in general GLIDEIN_Max_Walltime is set to 48 hours, +/- 24 depending on the site's own walltime limits. It is not possible for these glideins to be running much greater than GLIDEIN_Max_Walltime, since the glideins are strict about self-terminating at this limit. Thus, this discrepancy also reveals that gatekeepers are not reporting accurate accounting of the real picture in the batch system.

Given that gatekeepers do not accurately report the state of the batch system, let's take a look at how this causes starvation in glidien submission. In addition to providing the factory with the desired Requested Idle glideins to maintain on a site queue, the frontend also provides a dynamically generated value known as “Max Requested.” This maximum value is also dynamically generated based on user demand (proportional to user jobs idle + user jobs running at the site queue). It provides the factory with a limit to the total number of glideins allowed to be maintained on a queue at any given time. Max Requested is put in as a safety limit to ensure the factory doesn't DOS a site with glidein submissions in case there are any site problems that may cause the glideins to never successfully register back to the virtual HTCondor pool. Unfortunately, if the gatekeeper is not capable of providing accurate accounting, the queues will appear to exceed the Max Requested value, and the factory will refuse to submit new glideins, even though the actual number of glideins in the batch system has not actually exceeded Max Requested.

An example of starvation can be seen in figure 1. The green area, “Running” is the number running as reported by the gatekeeper to the HTCondor-G queue. The purple line, “Claimed” corresponds to the number of glideins actually still running, as reported by the frontend. As long as the red line corresponding to Max Requested stays below Running, the factory will refuse to submit more glideins, whether or not there is user demand for them. Note in this example starvation prevented any new glideins from being submitted to the site on the order of days (Oct. 20th– Oct. 22nd).

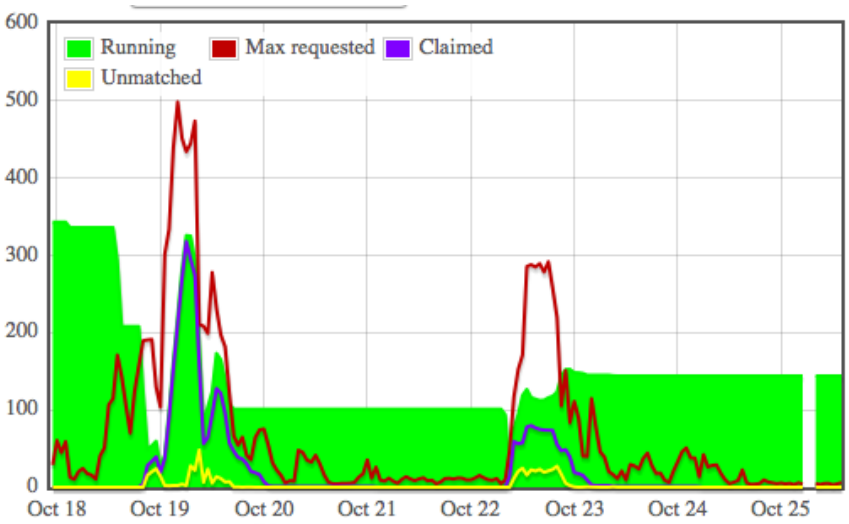

Figure 1. Glidein submission starvation

Typically when we discover glideins that are "running" indefinitely, they will never go away on their own. They may have terminated normally in the site batch system, and be long gone from the site, but the gatekeeper never lets our factory queue know that they have completed. Glidein submission will only resume if either the VO finally has enough user demand that Max Requested rises above the artificial number of glideins running (See figure 1, Oct. 22nd) or we can fix the situation by manually removing the stale glideins on our factory queues using condor_rm.

4. Automatic Stale Glidein Removal

In an attempt to improve the starvation problem described in section 2, on October 31st, we set up scripts to periodically remove glideins that are detected to be stale. We define stale glideins as any that remain on our factory queues such that they have been in the Running status longer than 1.5 times the GLIDEIN_Max_Walltime limit set for the queue. The motivation for the factor of 1.5 is just a safety precaution, in case there are any other conditions we are not aware of that may cause glideins to run longer than the max walltime but still be valid. So far no such conditions have been observed. The periodic removal occurs every 12 hours. As figure 2 shows, with auto removal in place, the starvation does eventually clear up. Unfortunately, when glideins terminate much less than GLIDEIN_Max_Walltime at the batch system, we still have to wait for 1.5 x GLIDEIN_Max_Walltime to pass before new glideins are able to submit again, and this can still be on the order of days.

In addition to putting the automatic removal in place, we also have put in place scripts to count the number of stale glideins removed per site, per day. This has given us some illuminating results. The first major observation is that the worst offender sites are all running Globus gt5 gatekeepers. We also observe CREAM and Nordugrid gatekeepers occasionally lose track of glideins, but the difference is 2 orders of magnitude worse for gt5 sites. For example, USCMS-FNAL-WC1, the worst offender gt5 site last week had an average rate of 360 glideins lost per day, whereas IN2P3-CC-T2, the worst offender CREAM site had an average rate of 6 glideins lost per day.

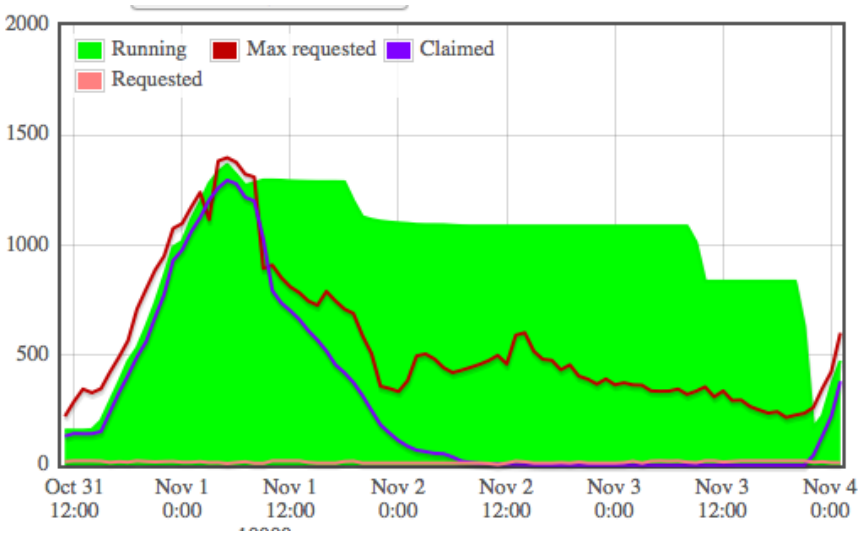

Figure 2. Starvation recovery

Another disturbing result is that the factory-wide average number of stale glidiens removed daily last week is 1670. However, as Figure 3 shows, looking at our total number of reportedly running glideins (green area) vs actually running glideins (purple line), it can be seen that the difference is about 10k. This is roughly same value of difference we observed before we put the auto removal into place. Thus our auto removal has little effect on the total starvation the factory experiences as a whole. The rate of auto removal is at least an order of magnitude slower than the rate at which gatekeepers lose track of glideins. We can conclude then that in Globus gt5 gatekeepers failing to accurately send status updates back to the factory is not a rare occurrence. It is happening constantly every day.

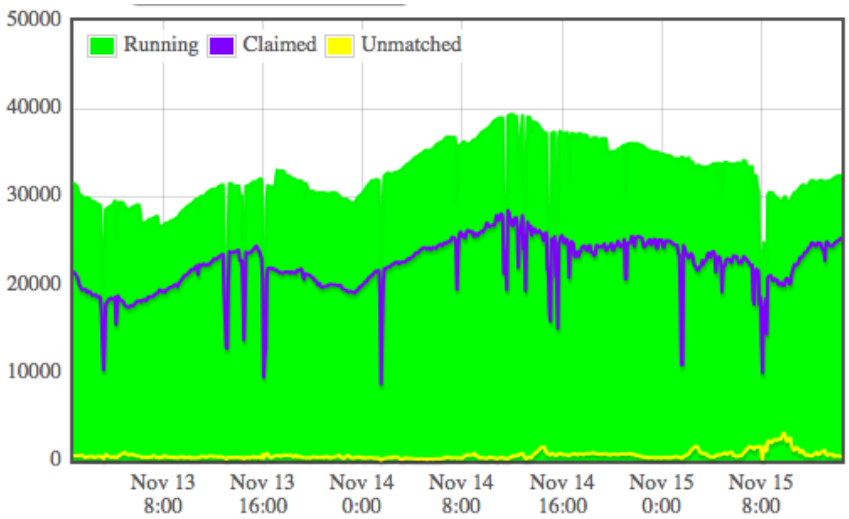

Figure 3. Factory total stale glideins

In the next section we will show other common problems we encounter with gatekeepers which are less gt5 specific.

5. Insufficient and Obtuse Debugging Information

Often times the gatekeeper does not provide sufficient information to debug a site problem, and in other cases, the the information it does provide is too ambiguous or cryptic to extract anything meaningful. When either happens, we can't solve the problem without requiring assistance from the site admins to dig for the root cause on their end.

The main case where we see a complete lack of debugging information from the gatekeeper is when a glidein lands on a black hole worker node. When this happens, the gatekeeper does not complain at all. The HTCondor-G submit log shows that the glidein ran and terminated successfully. However the output logs will be returned empty. Even though we get no help from the gatekeeper, we are able to infer that it is likely a glidein that ran on a black hole worker node if the runtime is significantly less than 20 minutes. This is because under normal circumstances, even if the glidein finds that the environment is not suitable to run user jobs on the worker node, the glidein will sleep for 20 minutes before terminating. The best we can do is supply the site admin with approximate times of the short running glideins, along with the grid job id and pilot proxy DN. In our experience, only the most expert site admins know how to use this data to determine if there are black hole worker nodes. In this case, we are stuck waiting for digging to occur at the site since we do not have direct access to the batch system; we have no way to proactively help the site admins understand what is going on.

The cases where the gatekeeper does provide information, but where that information is obtuse, is usually when a problem occurs at a site after the glidein has been authenticated at the gatekeeper, but before it can execute normally. In this case, the gatekeeper will report an error message and the glidein will go into “hold” status on the factory submit node. Unfortunately the error messages that are reported back from the various grid protocols typically do not give us a clear picture what is really going on. Here are a couple of examples that we observe often in factory operations.

In the first example, at a Globus site, if the user scratch area on the gatekeeper node is not writeable, for example if the disk is full or if an NFS mount is down, we might get error messages like the following:

018 (1182822.000.000) 10/14 08:53:05 Globus job submission failed!
Reason: 10 an end-of-file was reached
globus_xio: An end of file occurred

018 (1190567.006.000) 10/16 15:33:17 Globus job submission failed!
Reason: 47 the gatekeeper failed to run the job manager

Note the vagueness of these messages. A wide range of site problems could be described with these error messages besides a disk being full. Because we don't have direct access to the submit node of the site batch system, there is no way to directly verify that the disk is full, without contacting the site admin to ask what is going on.

The second example is observed at CREAM sites in particular. These errors occur when the batch system reaches capacity in its queues and it refuses to allow more jobs come in from the gatekeeper. When this happens we may see a reasonably descriptive hold reason:

CREAM error: BLAH error: submission command failed (exit code = 1) (stdout:)
(stderr:qsub: Maximum number of jobs already in queue for user MSG=total number of
current user's jobs exceeds the queue limit: user prdcms59@creamce02.ciemat.es,
queue medium-) N/A (jobId = CREAM008428794)

But other times we will see a completely misleading error message that we would never have guessed the root cause was simply hitting batch system limits:

CREAM error: BLAH error: submission command failed (exit code = 1) (stdout:)
(stderr:mkdir: cannot create directory `/var/jwgen//crm02_844243283.debug': File
exists-[ERROR] Globus::GRAM::Error::JOB_UNSUBMITTED-Invalid job
description-) N/A (jobId = CREAM844243283)

We have only come to understand how to interpret the hold reasons in the examples above and a few others only after many communication exchanges between the site admins, some even requiring the help from Globus and CREAM developers and experts. The number of hold reasons we don't

understand how to interpret or debug is much greater than the few that we do know how to handle. In both cases, because we don't have direct access to the submit node of the site batch system, we have no way to use the tools and logs provided by the batch system directly. Instead, we have a barrier in front convoluting the actual problem, and we must depend on site admins and other experts to look behind the barrier for us to figure out what is going on.

6. Why Use Gatekeepers at All?

Returning to the original motivation for using gatekeepers in the first place, recall that the supposed benefit is that the gatekeeper abstracts away the details of the batch system from the end user. However, today in the OSG Glidein Factory, we deal with multiple grid protocols. We have to understand how to submit to and debug glideins at sites running Globus (gt5), CREAM, Nordugrid, and soon HTCondorCE gatekeepers. The number of grid protocols available are approximately equal to the number of batch systems out there: HTCondor, PBS, LSF, SGE, SLURM. In an ideal world, the advantage of abstracting the batch system would only make sense if every site would use the same gatekeeper protocol in front. From our point of view it is no easier to learn to submit to 4 different grid protocols than it is to learn to submit to 4 different batch systems. Given this fact, compounded with the fact that gatekeeper implementations have been shown in the previous sections to be unreliable and force us to probe a black box rather than directly diagnose site problems, we believe there is no real advantage to using gatekeepers over direct submission to site batch systems.

One viable replacement for the gatekeeper that would benefit GlideinWMS by giving direct access to batch systems could be to give the factory GSI SSH access to the submit nodes at the sites. Sites would only require SSH logins at most for the factory and possibly an account per VO frontend. This is about 12 users total, and would be similar to the SSH setup most admins already have in place to allow for local user submission to their clusters. Authentication would still be provided as it is now, through GSI. For admins who require job wrapper customizations, this can still be implemented without a gatekeeper. Instead the batch system job submission commands themselves can be wrapped to perform the same customizations, as long as from the factory submission point of view they work in the expected manner. For those worried this solution will allow for the execution of arbitrary code on the submit nodes, really this is not any less secure than what we have now, with the ability to execute arbitrary code submitted as fork jobs. Also, to be clear, end users will not have direct access to the submit nodes. Only the required GlideinWMS infrastructure accounts mentioned previously for the factory and VO frontends will need GSI SSH access.

7. Conclusion

Grid resource provisioning systems such as GlideinWMS must submit to a wide variety of sites running different batch systems. Because the number of grid protocols are approaching the number of widely used batch systems, the benefit to having the grid protocol layer becomes non-existent. However, replacing the unnecessary and unreliable grid protocol layer with a direct batch system submission implementation would greatly reduce the cost of human effort for both the factory operators and the site admins. This will be achieved both by giving the GlideinWMS system more accurate accounting of what is really going on at the site, and by providing the factory operators with direct access to batch system internals needed to help understand and fix difficult site problems that are only masked by the gatekeeper in front.