Scalability of network facing services used in the Open Science Grid

I Sfiligoi, H Pi, F Würthwein, C Theissen and J M Dost
UC San Diego, 9500 Gilman Dr., La Jolla, CA 92093, USA
E-mail: isfiligoi@ucsd.edu

Abstract. The Open Science Grid relies on several network facing services to deliver resources to its users. The major services are the Compute Elements, Storage Elements, Workload Management Systems and Information Systems. All of these services are exposed to traffic coming from all over the world in an unmanaged way, so it is very important to know how they behave at different levels of load. In this paper we present the methodology and the results of scalability and reliability tests performed by OSG on some of the above services. The major services being tested are the Condor batch system, the GT2, GRAM5 and CREAM CEs, and the BeStMan SRM SE.

1. Introduction
Grid computing deployments, such as Open Science Grid (OSG)[1] and Europen Grid for E-SciencE (EGEE) [2], rely on several network facing services to deliver resources to its users. Example services are Compute Elements (CE), Storage Elements (SE), Workload Management Systems (WMS) and Information Systems (InfoSys). These services are mission critical, so extensive downtimes are unacceptable.

In order to avoid unexpected structural service problems, OSG has established a working group dedicated to testing the scalability and reliability of the services it uses. This team tests both existing and proposed software packages up to (and possibly slightly beyond) scales expected to be reached in the foreseeable future, evaluating and documenting both the response times and the failure modes.

Different services have different testing needs: Section 2 describes the methodology and tools used for testing SEs and InfoSys, Section 3 describes the methodology and tools used for WMS, and finally, Section 4 describes the methodology and tools used for CEs. Finally, Section 5 provides a discussion of how this activity was useful to OSG.

2. Testing the Storage Elements and Information Systems
Storage Elements (SEs) are the most heavily used services in the Grid environment, with potentially every worker node on the planet occasionally talking to them. The traffic pattern is thus highly unpredictable, with frequent bursts of activity. The wide geographical distribution of the clients further adds to the stress, since different clients will have significantly different network connectivity, in terms of bandwidth, latency and packet drop.
To effectively test an SE, the test environment must mimic the distributed nature of the real world, both in size and geographic distribution. Only grid resources can provide this type of structure, therefore all our tests have been run on top of them.

In this section we present the methodology used to perform these tests. The actual results of the storage tests can be found in [4] and [10].

2.1. Initial testing strategy
For the first round of tests we submitted our test jobs to the production Grid resources, collected the resulting data and performed a time-correlation analysis. For easier analysis, we typically presented the results of such tests as a scatter plot.

We also developed and used a multi-threaded client to artificially increase the number of clients of the SE. We could do that without any ill effect on the production resources since each client connection is very lightweight, and it takes hundreds of threads to use a full CPU.

Using the above method, we managed to measure the performance of one SE technology with concurrency levels in the range of few to about 3k concurrent clients. However, while the obtained results were very useful, the procedure used was less than ideal for the following reasons:

- The tests had to be re-run many times to obtain data points at high enough concurrency points. Since we used production resources, we were often limited in the number of batch slots available to us.
- The tests were not reproducible. Every test run gave us a scatter plot with different points.
- The data analysis was time consuming. We had to manually correlate test job logs and execution times, possibly across multiple time zones.
- And using production resources for testing purposes can be considered wasteful.

So we decided to invest effort in automating the procedure before moving to the next round of tests.

2.2. Automating the test procedure
The above problems were addressed using two complementary approaches. To address the first three problems, we developed a tool, called the GlideTester[3], to automate the test procedure. The last problem was addressed by engaging with several Grid sites to provide dedicated test resources.

GlideTester is a tool that fully automates a set of test runs. The operator provides the necessary inputs and the tool then carries out the tests autonomously, without any further human intervention. The needed inputs are:

- a test script, such as a simpler wrapper around lcg_ls,
- the desired length of the test, and
- the list of concurrency points.

The result of each invocation are a set of data points, one per each desired concurrency level, that can be easily converted into a scatter plot or any other reporting tool. The results are also easily reproducible; each invocation is guaranteed to provide the data points at exactly the same concurrency points.

The test resources at select Grid sites were provided by deploying a shadow pool at each. A shadow pool is a set of batch system resources deployed alongside an existing production pool. While the worker nodes are the same for both pools, the submission point and the control services are separate. In order not to interfere excessively with the production pool, only lightweight jobs, with small CPU, input and output requirements are allowed.

Using this new setup, we were able to perform tests at well defined concurrency points in a repeatable manner, with minimal human intervention. In particular, we tested two other SE technologies in this manner, and each test run required only a day wallclock time and few hours of
human time. Compare this to $O(1 \text{ month})$ wallclock and $O(1 \text{ week})$ human time of the previous method.

The added efficiency allowed us to perform many more test runs, with concurrency levels in the same range of few to about 3k concurrent clients. The net result was a continuous test-report-fix-test cycle between us and the software developers, which resulted in significantly better understanding of the software bottlenecks.

2.3. Information system testing
Information Systems have a very similar access pattern as SEs, although the access rate is typically an order of magnitude lower.

We thus used the same methodology, relying on GlideTester and test Grid resources to perform the measurement of two OSG information systems, namely BDII and RESS. The test results were communicated to the operators of the OSG information systems, but were not deemed interesting enough to earn a dedicated publication, not having discovered any significant weaknesses, so they are not referenced here.

3. Testing the workload management systems
Workload management systems (WMS) are subject to different load patterns. The main scalability factor is the number of user jobs that need to be handled, and not the number of clients talking to a WMS. Moreover, there is currently no standard API that can be used across all the WMS, so we decided to concentrate our effort with just one, namely Condor.

Condor in the major WMS in OSG. It is being used for
- handling dedicated clusters,
- as a submission mechanism to the Grid, also known as Condor-G, and
- as a pilot infrastructure for effectively utilizing the Grid resources, as the foundation of glideinWMS.

The most demanding scalability requirements among these use cases are on glideinWMS, since it may potentially have to handle all the user jobs of a VO; one such VO, CMS, estimates that it may in the near future need to handle up to 300k user jobs, with 40k running concurrently across both OSG and EGEE resources.

As with the later SE testing, we used a fully automated procedure to perform our tests. The testing framework for generating the load on the glideinWMS has been developed by our team the year before, and is called loadtest_condor[5]. Similarly to GlideTester, it is fully automated and configuration driven. It creates the test jobs to be run, submits them to the Condor queue with a pre-defined frequency and lets them run for a configurable amount of time. The aim is to show that the system is stable at a certain level of job pressure, or record the misbehavior.

Another aspect of the scalability testing is the monitoring of the resource usage during the test runs. In particular, we were interested in classifying how many resources are needed by different processes used by the glideinWMS. To the best of our knowledge, there was no tool available to perform this task, so we developed a tool ourselves, and called it procs_monitor[6].

Taken together, the system proved to be very effective, with each run requiring only $O(\text{one day})$ wallclock time and few hours of human time. It allowed us to easily push the limits of Condor while documenting both its resource consumption and breaking point. The knowledge has been continuously fed back to the Condor development team, who then provided updated code to be tested again. This test-report-fix-test cycle resulted in a significant improvements in the quality of the Condor code on a relatively short time scale.

More details about the results of our testing and recent Condor improvements can be found in [7] and [11].
4. Testing the compute elements

Compute elements (CEs), also known as Gatekeepers, have scalability and reliability properties similar to both SE and WMS; they are accessed by a wide variety of nodes, and they have to handle many user jobs. Ideally, thus, one would need to combine the above described tools to properly test them.

However, testing a CE requires keeping a job queue on the client for long periods of time, and this is not something we could do on our test Grid resources. So we settled for testing the CEs from one node only. Testing from a single node, while suboptimal, is not completely unrealistic. Many user communities rely on centralized services to submit all of their jobs, for example though gliteWMS or a glideinWMS factory.

In OSG, the de-facto client for submitting to all CEs is Condor, so we reused loadtest_condor, introduced in the previous section, for the CE tests as well. The major OSG CE software, the pre-WS GRAM-based CE, also known as GT2, has not been actively developed for several year now, so additional testing was not deemed useful. Instead, we concentrated on two new CE solutions, namely the new Globus 5 CE, also known as GRAM5, and the INFN CREAM CE, that have been selected as possible candidates to replace it. The OSG scalability team has thus worked with the software providers to deploy a OSG-compliant test CE instance of each, test them with the OSG client tools, and report back any bugs or missing functions, repeating the loop as long as deemed necessary.

The process took much longer than expected, since we discovered many problems both in the server and client, but it was well worth the effort as both new CEs have in the end proven to be much better than GT2. The results of the CE evaluation can be found in [8].

One interesting lesson learned during this exercise was the discovery of a significant difference in the integrated wallclock usage between CEs, that could not be explained by results provided by the testing framework alone. After careful, manual analysis of all the logs we had access to, it turned out that the test setup of CREAM we were using was overloading our test client. While in the end we did find the root cause, the effort needed was quite substantial; the testing framework should thus be improved.

5. Discussion

The tests performed by the OSG scalability and reliability group have been fundamental in guiding the OSG community in the past year.

The SE tests have been used to validate the scalability and robustness of the HDFS-backed BeStMan SE technology, leading to its acceptance in the OSG software stack, and subsequent deployment on several US CMS Grid sites [4].

The Information System tests have been used to validate the transition in OSG from BDII v4 to BDII v5.

The Condor tests have pinpointed several bottlenecks in the Condor code, which were subsequently addressed by the developers. As a result, a single scheduler node can now manage over 40k batch slots, up from 20k last year, and with a throughput rate of over 1M jobs/day, up from 200k last year. Similarly, the central manager can now handle up to 90k batch slots and effectively match them with O(100k) jobs, an order of magnitude improvement [7].

The CE tests have been used to evaluate the feasibility of two new CE technologies, namely CREAM and GRAM5, against the existing OSG CE solution, namely GRAM2, with the main goal being an objective comparisons between different implementations of the same functionality. The final result was the recommendation for acceptance of both new CE technologies in the OSG software stack [8].
6. Summary
The OSG scalability and reliability group has been testing several network facing services OSG is relying on, including several Compute Elements, several Storage Elements, a Workload Management System and an Information System, with the aim of identifying the limits of such services at scales expected to be needed in the near future. Any observed limits have been reported to the developers of such services, who can use it to improve their products. Several times we also collaborated with the developers to test pre-release software, thus speeding up the improvement cycle.

In order to perform our tasks, we had to develop several testing and monitoring frameworks. All of our code is open source, hosted in Sourceforge[9], and can be freely downloaded and used.

Acknowledgments
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 Grants No. PHY-0533280 (DISUN), PHY-0612805 (CMS Maintenance & Operations), and OCI-0943725 (STCI).

References
[1] The Open Science Grid Executive Board 2010 A Science Driven Production Cyberinfrastructure - the Open Science Grid OSG Document 976, http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=976
[2] Laure E and Jones B 2009 Enabling Grids for e-Science: The EGEE Project Grid computing: infrastructure, service, and applications 55-74, ISBN: 978-1420067668
[3] Sfiligoi I, Wuerthwein F and Theissen C 2010 Using Condor Glideins for Distributed Testing of Network Facing Services Proc. Int. Joint Conf. on Comp. Sci. and Opt. (CSO) 327-31, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5532083
[4] Pi H, Sfiligoi I, Wuertwein F and Rana A 2010 Measurement of BeStMan Scalability OSG Document 1004, http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=1004
[5] Sfiligoi I and Würthwein F 2010 Loadtest_condor – A workload generating framework for testing scalability and reliability of the Condor system OSG Document 1013, http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=1013
[6] Dost J M and Sfiligoi I 2010 Procs_monitor - A process-level resource monitor OSG Document 1012, http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=1012
[7] Bradley D et.al. 2010 An update on the scalability limits of the Condor batch system Int. Conf. on Comp. in High Energy and Nuclear Physics (CHEP 2010)
[8] Sfiligoi I and Padhi S 2010 Evaluation of new Compute Element software for the Open Science Grid: GRAM5 and CREAM OSG Document 1006, http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=1006
[9] Sfiligoi I 2010 OSG Scalability Area Tools Web site, http://sourceforge.net/projects/osgsscal/. Accessed Dec 2010.
[10] Sfiligoi I 2010 OSG Scalability, Reliability and Usability Twiki, http://hepuser.ucsd.edu/twiki2/bin/view/UCSDTier2/OSGScal, Accessed Dec 2010
[11] Sfiligoi I 2011 Condor Scalability tests Twiki, http://hepuser.ucsd.edu/twiki2/bin/view/UCSDTier2/CondorScal, Accessed Feb 2011