Early experience on using glideinWMS in the cloud

Early experience on using glideinWMS in the cloud

W Andrews[3], B Bockelman[4], D Bradley[5], J Dost[3], D Evans[1], I Fisk[1], J Frey[5], B Holzman[1], M Livny[5], T Martin[3], A McCrea[3], A Melo[6], S Metson[2], H Pi[3], I Sfiligoi[3], P Sheldon[6], T Tannenbaum[5], A Tiradani[1], F Würthwein[3] and D Weitzel[4]

[1] Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510, USA
[2] University of Bristol, Tyndall Avenue, Bristol BS8 1TL, UK
[3] University of California, San Diego, 9500 Gilman Dr, San Diego, CA 92093, USA
[4] University of Nebraska-Lincoln, 118 Schorr Center, Lincoln, NE 68588, USA
[5] University of Wisconsin, Madison, WI, USA
[6] Vanderbilt University, 6301 Stevenson Center, Nashville, TN 37235, USA

E-mail: burt@fnal.gov

Abstract. Cloud computing is steadily gaining traction both in commercial and research worlds, and there seems to be significant potential to the HEP community as well. However, most of the tools used in the HEP community are tailored to the current computing model, which is based on grid computing. One such tool is glideinWMS, a pilot-based workload management system. In this paper we present both what code changes were needed to make it work in the cloud world, as well as what architectural problems we encountered and how we solved them. Benchmarks comparing grid, Magellan, and Amazon EC2 resources are also included.

1. Introduction

In the last ten years, the High Energy Physics community has steadily adopted the grid computing paradigm in order to process and analyze data on a global scale. The implementation for the Large Hadron Collider (LHC), known as the Worldwide LHC Grid (WLCG), is built upon two international cyberinfrastuctures – the Open Science Grid (OSG) and the European Grid Infrastructure (EGI) [1-3]. The WLCG relies on leveraging resources (computing, storage) that have been offered for use to the community, but are owned, operated, and maintained by the resource owners. The resources are widely distributed across the globe, and represent a very large scale in the aggregate.

GlideinWMS is a pilot-based workload management system designed for the grid. Principally developed by the CMS experiment, it is currently in use by a large number of HEP and non-HEP virtual organizations [4]. In essence, GlideinWMS sends a "pilot job" which dynamically provisions resources from the grid and synthesizes them into a virtual Condor batch system. In addition to presenting the user with a simplified interface in which to do work, the pilot approach has many other advantages, including the ability to perform late-binding validation of acquired resources and custom application-level monitoring in near real-time.

Recently, cloud computing has come into vogue as a solution for both commercial and scientific computing. This term generically refers to a set of virtualized resources that are constructed and torn down dynamically in response to demand. The current industry standard is considered to be Amazon Web Services (AWS). AWS provides a computing resource – the Elastic Compute Cloud (EC2), and a storage solution – Simple Storage Service (S3) [5,6]. Recently, the Department of Energy has begun operating the Magellan project, a research and development effort to provide a cloud-based distributed computing and data analysis testbed [7]. The current Magellan implementation has been built with the open source Eucalyptus software package [8] and has an EC2-compatible API.

In this work, we discuss the challenges and solutions in adapting glideinWMS to the cloud, focusing on EC2-compliant implementations. We present benchmarks using CMS software on the cloud via glideinWMS, and future directions for the project.

2. GlideinWMS – an overview on the grid

GlideinWMS leverages software from the Condor project [9] whenever possible, and is comprised of several logical components as shown in figure 1 [10].

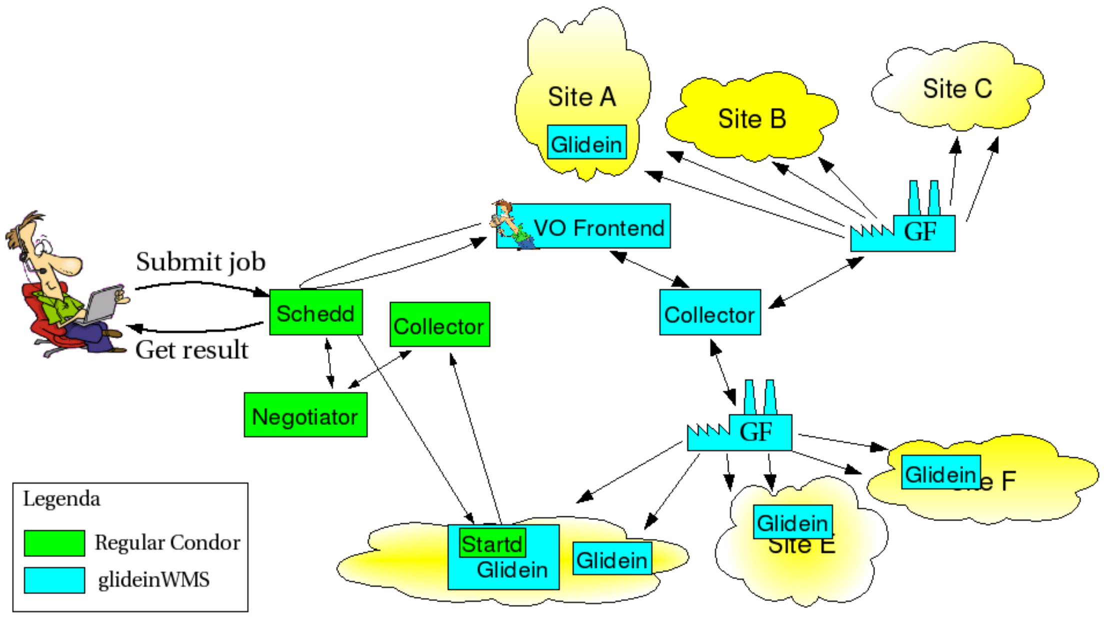

Figure 1. Overview of the glideinWMS system.

Briefly, the user submits work to a local scheduler ("schedd") on a submit machine. A frontend polls a set of submit machines, and sends a request for glideins to the glidein factory ("GF"). The glidein factory interfaces with the grid gatekeepers and submits glideins for execution on remote sites. The glideins begin executing, pull down factory and frontend-specific customizations if supplied, and communicate back to a collector, creating a virtual Condor batch system. The workloads from the submit machines are then matched to the glideins and executed via Condor communication protocols, bypassing the gatekeepers.

3. Challenges

In the course of migrating the glideinWMS architecture from the grid to the cloud, we discovered several differences between these distributing computing approaches. We categorized these differences as either related to the different job environment in the cloud (the virtual machine challenge) or the different credential management model in the cloud (the authentication challenge).

3.1. The Virtual Machine Challenge

One challenge in transitioning from the grid to the cloud is how one manages the virtual machines. In the grid, the maintenance of the operating system, system libraries, and other core services are the responsibility of the site administrator; and application software is installed as needed in persistent storage. On the cloud, these responsibilities shift to the user. For the benchmark results presented here, images were manually create and uploaded; this method will not scale for future use.

3.1.1. Contextualization
There are two types of virtual machine contextualization that are often discussed. The first, "image contextualization", refers to modifying a stock virtual machine to perform specific functions; for example, the creation of a web server VM image. "Instance contextualization", on the other hand, refers to the ability to perform customizations of the virtual machine at run-time. Amazon EC2 and EC2 API compliant clouds require image contextualization in order to perform instance contextualization. The challenge then is to make use of the instance contextualization facilities provided by the cloud implementations. Amazon EC2 and EC2 API-compliant clouds provide a "user data" facility; when a request is submitted for a VM, up to 16KB can be forwarded to an internal web server. The VM, if so configured via image contextualization, can access the user data and appropriately customize the instance during the VM's boot cycle.

In order for a glideinWMS to make use of a cloud, the VM instance must be made "glideinWMS aware". In the grid environment, the glidein startup script, glidein_startup.sh, is submitted as an executable to a remote gatekeeper to be run on the worker node. To replicate this behaviour in the cloud, we created a custom image containing a bootstrap service. This service accesses the EC2 user data, which contains a configuration file from the glideinWMS factory and an X.509 proxy used for authenticating for the factory. The bootstrap service then downloads glidein_startup.sh from the glideinWMS factory. This is executed as on a grid batch system, and the image spawns the required components in order to form the virtual Condor node.

In the course of this work, we extended the functionality of glidein_startup.sh to make it sufficiently general to execute in both grid and cloud environments; the same script can instantiate a pilot on an EC2 instance or a batch system.

3.1.2. Virtual Organization (VO) Supplied Software
Some VOs have software that is relatively simple to install and configure. These VOs can include their software as part of the setup for their user jobs or as the part of the image contextualization. However, some VOs, like CMS, have large, frequently updated, complex software packages. On the grid, these VOs typically pre-install and update their software at each supported site through special installation jobs. The software is then shared via a networked file system and mounted on each worker node in the cluster. This model is not a good fit for the cloud, as no vendor-provided shared filesystem exists.

A solution that we investigated is CVMFS [11]. This provides a read-only file system mounted on the virtual machine. The file system is a FUSE mount served over HTTP. We configured the virtual machines to point directly to a server located at CERN (deemed acceptable as we were running only small scale tests). CVMFS can also specify a number of HTTP proxy servers for load-balancing. One advantage to this approach is, unlike a networked file system installation, only the files opened by the user job are retrieved and cached on the virtual machine. It is not necessary to install all of CMS software (50 gigabytes or more) on a foreign site. It also eliminates the need to load CMS software into Amazon's S3, which decreases the associated cost with using the cloud.

3.1.3. Image Creation
The question of VM image authorship is a non-trivial one. GlideinWMS deployments generally service multiple VOs – who retains the responsibility for maintaining the image? If the VO provides the image, it has complete control over customization and testing; on the other hand, it still must follow the process as outlined in section 3.1.1, possibly resulting in significant duplicated efforts. If the glidein factory provides a base image, the work is performed once by the factory owner, and should better ensure the required glideinWMS services are properly installed. However, removing the OS customization flexibility negates one of the most powerful advantages of cloud computing. A solution currently under investigation allows the factory to provide a glideinWMS-compatible base image, and allow the VO to perform image contextualization. This enables VOs to use glideinWMS regardless of whether they need contextualization. There are some drawbacks – it may make it more difficult for VOs to validate the dynamically-produced image, and could have a negative impact on

performance, but we believe the advantages of providing a highly-customizable cloud solution to the end user at little cost outweighs the disadvantages.

3.2. The Authentication Challenge

All grids currently used by the HEP community use X.509 proxies [12] as the underlying authentication mechanism. A proxy is used for the initial job submission, and it is also forwarded to the worker node to allow the job to access remote resources while it is executing. GlideinWMS was thus designed around it, with the frontend delegating a X.509 proxy to the factory for both glidein submission and for the glidein to secure the communication with the VO services. Unfortunately, cloud resources are not based around this security paradigm. This introduced three problems for the glideinWMS: the lack of a credential delegation mechanism, the need for two sets of credentials, and credential lifetime concerns.

The X.509 proxy delegation is the foundation of glideinWMS security. The proxy being delegated to the worker node is used by the glidein to secure the communication to the VO-run collector; without it, the glidein would not be trusted by the VO. Since most clouds do not support any credential delegation, we decided to continue to use X.509 proxies, and find ways to deliver them to the worker nodes via secure channels. We currently have a solution for the AWS: we ship the proxy as one of the job arguments, which is sent using SOAP via secure HTTP and has privacy guarantees within the cloud [13]. We are investigating how best to translate this solution to other cloud architectures.

The use of X.509 proxies in the glideins in the cloud implies that the glideinWMS now needs to handle two sets of credentials: a non-X.509 credential for glidein submission, and an X.509 credential for glidein authentication to the VO services. This was not envisioned in the initial glideinWMS design so the protocol between the frontend and the factory was extended. We added this functionality in a backwards-compatible manner by leaving the X.509 proxy delegation unchanged, and allowing for an additional, optional credential forwarding step. If the optional credential is not present, the factory will default to using the proxy for glidein submission.

One useful property of the X.509 proxy security model is the ability to derive short-lived credentials from long-lived ones; this significantly reduces the security risk of delegating work to another party. With respect to glideinWMS, this allows the VO to reduce the risk associated to the delegation of its credential to a factory operated by a different party. We have not yet found a good solution to this issue in the cloud, and are currently delegating long-lived credentials. Amazon's Identity and Access Management system [14] appears promising, but has not been fully evaluated.

4. Benchmarks

We selected a few fundamental measurements to characterize the performance of glideinWMS on the cloud. We identified three metrics of interest: how long it takes to provision resources; how efficiently the acquired resources perform a domain-specific computing task; and how quickly the resources can perform the same task.

4.1. Performance

We ran a CMS benchmark to get a performance comparison between the Amazon EC2 virtual machine and a physical machine. The CMS benchmark is a Monte Carlo simulation of top and anti-top quark production. This simulation was specifically chosen because it generates many particles, each of which is tracked through the CMS detector. Simulating and tracking multiple trajectories is a CPU-intensive process. Since little storage is required to run the simulation, most of the resulting I/O will be memory I/O.

The benchmark is first run as a single process to obtain baseline data. It then simultaneously executes one job per core. The benchmark outputs the wall time and CPU time; the CPU efficiency is calculated as the ratio of CPU time to the wall time. The benchmark was run on a typical worker node in the USCMS Tier-1 facility at Fermi National Accelerator Laboratory and on an "m1.large" EC2 instance. The worker node is an AMD Opteron 2915 node with 8 physical cores, 24 GB memory, and runs Scientific Linux 5.4. The EC2 m1.large instance has 2 virtual cores (with 2 EC2 Compute Units each), 7.5 GB memory, and runs Scientific Linux 5.5.

Table 1. Benchmark results for a CMS Monte Carlo simulation.

System	Number of cores	Avg. CPU Time per core (s); smaller is better	Avg. Wall Time per core (s)	Avg. CPU Efficiency
Amazon EC2	1	76.98	103.7	74.22%
	2	74.91	142.5	52.71%
AMD Opteron 2915	1	68.81	96.22	71.51%
	8	69.04	96.19	71.79%

The physical machine reported just over 96 seconds per event for the single process. When the number of processes was scaled to the number of cores, the time per event remains relatively unchanged with only sub-second fluctuations. With a single process the EC2 instance was able to process at a rate of 103.7 seconds per event. When both cores are used the time per event is increased to 142.5 seconds.

Comparing the single-process results, one can see that there is a 7% performance penalty incurred when running on EC2. This is likely due to the virtualization layers. There is also a significant performance decrease on EC2 when going from one to two cores. This merits further investigation, but suggests we are running into a memory bottleneck within EC2's virtualization.

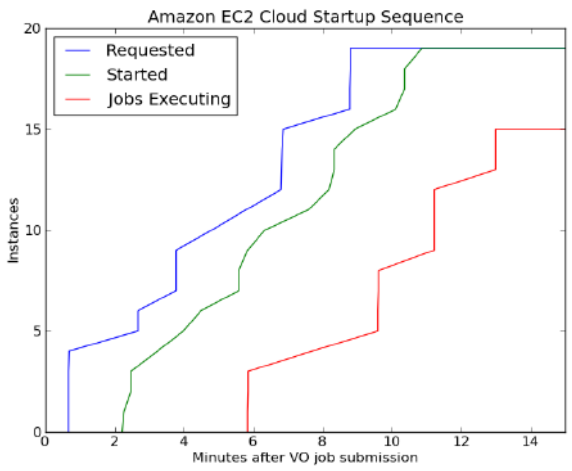

Figure 2. Response time for provisioning glideinWMS execute nodes in Amazon EC2.

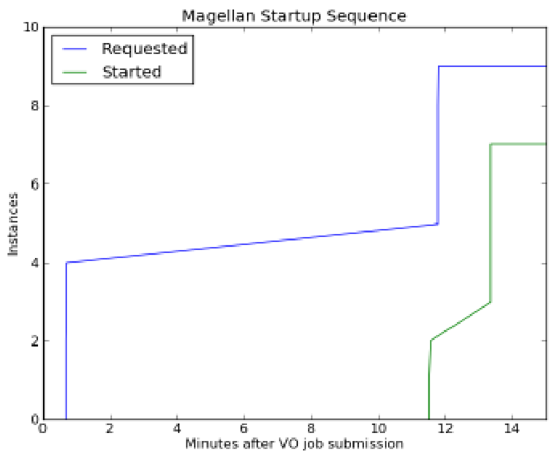

Figure 3. Response time for provisioning glideinWMS execute nodes in Magellan.

4.2. Startup Time

Measurements of time to provision new resources and begin utilizing them were conducted in the EC2 and Magellan clouds, as depicted in figures 2 and 3. In EC2, new worker nodes appeared in the Condor pool about 1.5 minutes after they were requested. Startup of jobs on those nodes then happened with a scheduling lag within Condor of about three minutes. In Magellan, worker nodes appeared in the Condor pool about 5 minutes after they were requested. Startup of jobs was not successful due to problems that are under investigation.

4.3. Network

Network bandwidth was measured using the iperf tool between worker nodes in EC2 and computers at the University of Nebraska-Lincoln. The observed transfer rate was 500 Mbps. The round trip time was 35 ms. This connection benefits from the peering arrangements between Amazon and Internet2, and may not be indicative of performance at other research computing facilities.

5. Summary and Outlook

Interest in cloud computing continues to increase from both the academic and commercial spheres. With this work, we have established first steps towards utilizing the widely available glideinWMS framework in order to utilize cloud resources. A key result was the ability to seamlessly combine cloud- and grid-based resources without presenting a different end-user experience.

We are working on making available more dynamic contextualization services to allow VOs to provide significant portions of the image environment (or none, if the stock environment provided by the glideinWMS factory meets their needs). We are also evaluating new cloud credential management tools such as Amazon's Identity and Access Management.

Acknowledgments

This work was partially funded by the US National Science Foundation under Grants No. PHY-0533280 (DISUN), PHY-0612805 (CMS M&O), and OCI-0943725 (STCI); the US Department of Energy under Grant No. DEFC02-06ER41436 subcontract No. 647F290 (OSG); and several US CMS related grants. The research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357, funded through The American Recovery and Reinvestment Act of 2009.

References

[1] Pordes R et al. 2007 The Open Science Grid *Journal of Physics: Conference Series* **78** 012057.
[2] Shiers J 2007 The Worldwide LHC Computing Grid (worldwide LCG) *Comp. Phys. Comm.* **177** 219-223.
[3] The European Grid Infrastructure (EGI), http://www.egi.eu
[4] Sfiligoi I et al. 2010 Operating a Production Pilot Factory Serving Several Scientific Domains, these proceedings
[5] Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2
[6] Amazon Simple Storage Service (S3), http://aws.amazon.com/s3
[7] The Magellan Project, http://magellan.alcf.anl.gov/, http://magellan.nersc.gov/
[8] Nurmi D et al. 2009 Eucalyptus: an Open-source Cloud Computing Infrastructure *J. Phys.: Conf. Ser.* **180** 012051
[9] Thain D, Tannenbaum T and Livny M 2005 Distributed Computing in Practice: The Condor Experience *Concurrency and Computation: Practice and Experience* **17** 323-356
[10] Sfiligoi I et al. 2009 The Pilot Way to Grid Resources Using GlideinWMS *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering* **2** 428-432
[11] Buncic P et al. 2010 CermVM – a Virtual Software Appliance for LHC Applications *J. Phys.: Conf. Ser.* **219** 042003
[12] RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, http://www-rfc-editor.org/rfc/rfc3280.txt
[13] Amazon EC2 User Guide, http://tinyurl.com/ec2-userguide
[14] Amazon Identity and Access Management (IAM), http://aws.amazon.com/iam