Dynamic Provisioning of Virtual Organization Clusters

Michael A. Murphy, Brandon Kagey, Michael Fenn, and Sebastien Goasguen
School of Computing
Clemson University
Clemson, South Carolina 29634-0974 USA
{mamurph, bkagey, mfenn, sebgoa}@cs.clemson.edu

Abstract

Virtual Organization Clusters are systems comprised of virtual machines that provide dedicated computing clusters for each individual Virtual Organization. The design of these clusters allows individual virtual machines to be independent of the underlying physical hardware, potentially allowing virtual clusters to span multiple grid sites. A major challenge in using Virtual Organization Clusters as a grid computing abstraction arises from the need to schedule and provision physical resources to run the virtual machines.

This paper describes a virtual cluster scheduler implementation based on the Condor High Throughput Computing system. By means of real-time monitoring of the Condor job queue, virtual machines that belong to individual Virtual Organizations are provisioned and booted. Jobs belonging to each Virtual Organization are then run on the organization-specific virtual machines, which form a cluster dedicated to the specific organization. Once the queued jobs have executed, the virtual machines are terminated, thereby allowing the physical resources to be re-claimed. Tests of this system were conducted using synthetic workloads, demonstrating that dynamic provisioning of virtual machines preserves system throughput for all but the shortest-running of grid jobs, without undue increase in scheduling latency.

1. Introduction

Grid communities, such as the Open Science Grid, are frequently organized around Virtual Organizations (VOs), or groups of entities and individuals with a shared scientific mission [1]. A key challenge in providing resources to these VOs has been the issue of software compatibility across disparate grid sites. Virtual Machines (VMs) have been identified as a mechanism for constructing computing clusters with homogeneous software environments. [2]–[5] By allowing each VO to provide its own VMs, it is possible to provide each VO with a dedicated cluster of machines that execute its jobs exclusively. Clusters constructed in this way are termed *Virtual Organization Clusters* or VOCs. [6]

Since VOCs exist entirely within virtual machine space, entire clusters may be started, stopped, or resized dynamically and automatically. Although mechanisms have been described for the dynamic per-user re-allocation of physical hardware [7], the rapid creation of clusters of virtual machines [8], [9], and the provisioning of workspace environments to meet specific resource constraints [3], [10], dynamic instantiation and re-sizing of virtual clusters has remained a challenging issue. Moreover, matching jobs owned by a particular VO to VO-specific virtual clusters running on private networks has not been addressed in the context of virtual clusters. This paper describes an approach to solving both of these problems by dynamically instantiating, re-sizing, and removing Virtual Organization Clusters, while ensuring that grid computing jobs submitted by users affiliated with a specific VO execute only on the VOC owned by that specific VO.

The remainder of this paper is organized as follows. related work is discussed in section 2 while section 3 reviews the model of, and motivations behind, creating virtual compute clusters on a per-VO basis. This model is extended in section 4, where the design of the dynamic provisioning system is described. Results of testing the provisioning system are presented in section 5, after which conclusions are presented in section 6.

2. Related Work

Constructing clusters from virtual machines was proposed in [2] and realized by degrees in In-VIGO [11], VMPlants [12], and Virtual Clusters on the Fly [9]. In-VIGO focused on the end-to-end design of a Web service that could employ VMs as part of a cluster computing system, while VMPlants and Virtual Clusters on the Fly focused on rapid construction of virtual clusters. All three systems were particularly concerned with the issue of specifying and adapting to requirements and constraints imposed by the user. Unlike the VOC Model, which specifies how to design and execute a virtual cluster from existing VMs, these projects were more concerned with constructing the VMs themselves. In addition, VMPlants and Virtual Clusters on the Fly had explicit one-to-one mappings between VM instances and VM disk images.

Dynamic Virtual Clustering (DVC) [5] implemented the scheduling of VMs on existing physical cluster nodes within a campus setting. The motivation for this work was to improve the usage of disparate cluster computing systems located within a single entity. Only local jobs originating from the campus were run inside virtual containers: the system was not connected to a computational grid. VMs were directly attached to existing physical clusters, extending individual homogeneous clusters across heterogeneous systems. Like VMPlants, DVC utilized one virtual disk image per VM instance, and these disk images were transferred between physical hosts.

Virtual resources have previously been attached to computational grids by systems such as Globus Virtual Workspaces [3], [4], [10]. These workspaces were created from job-specific specifications, allowing for the dynamic instantiation of execution containers for grid jobs. While these containers did provide isolation and customization, they were still transient environments that were generated from a specification and instantiated for the life of the job to be executed within.

Dynamically provisioned Virtual Organization Clusters are not completely orthogonal to these related projects. Rather, VOCs provide an abstraction for site-independent computational clusters, in which the VMs are owned and managed by the VO instead of a middleware system, and the virtual disk images are not considered transient but are instead permanent. Dynamic provisioning simply adds a mechanism for instantiating VMs to create clusters, without imposing any constraints on how the VM images get created or what site policies will apply to the systems. Thus, dynamic VOCs could be used in conjunction with systems that programmatically generate the initial disk image from which the VM instances are spawned.

3. VOC Model

The Virtual Organization Cluster Model, described in detail in [6], provides a means by which each VO can have its own dedicated cluster and associated administrative domain. Virtual Organization Clusters using this model are formed from VMs that are either created at the physical site where they are to be used, created by grid middleware (for example, In-VIGO [8]), or manually transmitted to the physical site by a VO administrator. Unlike systems that utilize one virtual disk image per VM instance [3], [5], [9], VOCs are explicitly designed to work with either one or two disk image files: a single image representing the configuration of a compute node, and an optional second image that contains a cluster head node. All compute node VMs are spawned from the single compute node image using a copy-on-write mechanism that allows the original image file to be read-only. The result of this design is that both disk space and network bandwidth requirements are reduced, while management of the VOC is simplified.

A major benefit of the VOC Model design is that few constraints are placed on the VM. The VO has great flexibility in selecting the operating system and software environment best suited to the requirements of its users. Of the few constraints that do exist, the primary ones are as follows:

- **Image Compatibility.** The VM image must be in a format usable by the Virtual Machine Monitor (VMM) or hypervisor software in use at the physical site(s) where the VOC will be executed.
- **Architecture Compatibility.** The operating system running in the VM must be compatible with the system architecture exposed by the VMM or hypervisor.
- **Dynamic Reconfigurability.** The guest system inside the VM must be able to have certain properties, such as its MAC address, IP address, and hostname, set at boot time.
- **Scheduler Compatibility.** When only a single image file is used with a shared scheduler provided by the physical site, the scheduler interface on the VM must be compatible with the shared scheduler.

For the purpose of discussing dynamic provisioning and scheduling of jobs on VOCs, the single-image case is simpler, since there is only one job queue to be maintained at any location on the system. The implementation described in this paper thus assumes that each VO provides only a compute node image, which has the required scheduler client installed and operable. However, we note that multiple VOCs could be totally isolated from one another by deploying separate head node images.

4. Dynamic VOC Architecture

In order to provision Virtual Organization Clusters dynamically, grid-enabled middleware was developed. This middleware enables physical systems to host VOCs that are connected to the Open Science Grid. Grid jobs arrive via Globus [1], [13] and are deposited in a job queue on a gatekeeper system, where the Condor job scheduler [14] runs. A watchdog process periodically samples the Condor queue and starts virtual machines that belong to the VO with which each job is associated. As the number of jobs in the queue for a particular VO increases, the watchdog will attempt to start additional VMs to increase the size of the respective VOC, subject to the limitations imposed by the hardware and by site policy. When the watchdog observes fewer jobs in the queue than there are executing VOC nodes for a particular VO, VMs belonging to the VOC are terminated. This process ensures that provisioning of physical resources dynamically adapts to the job loads of the VOs supported by the grid site, without any manual intervention or need for external middleware.

It should be emphasized that the design of dynamic VOCs is strongly biased toward High Throughput Com-

puting (HTC), as opposed to High Performance Computing (HPC). Thus, there is no provision for enforcing any type of scheduling deadlines. Jobs execute as VOC node resources are available. As shown in prior work [6], there is little motivation for enhancing the dynamic model to support HPC, given the current state of the technology. Virtualization performance penalties of over 60% have been observed with HPC jobs, while performance of latency-tolerant HTC applications was affected by under 10%. Such low HTC overheads are believed to be acceptable for HTC applications.

4.1. Job Tagging

Upon the arrival of a grid job, a corresponding Condor job is created by Globus. The dynamic VOC system modifies the Condor ClassAd to add a requirement that the target system match the VO with which the grid job is associated. Within the ClassAd, this additional requirement takes a simple name equals value form, in which the name of the VO is prefixed to form the name, while the boolean condition of truth is used as the value.

As an example, the Requirements field for a job associated with the Engage VO, targeting a 32-bit Linux system, might have the form:

```
Requirements = (Arch == "INTEL") &&
(OpSys == "LINUX") && (VO_ENGAGE ==
TRUE)
```

One requirement imposed upon each VOC compute node VM is that its Condor interface expose a corresponding ClassAd field to match the VO. Thus, a VOC node that is owned by the Engage VO should include the following field in its ClassAd:

```
VO_ENGAGE = TRUE
```

With this mechanism, the Condor scheduler will handle the matching process of a grid job to a corresponding VM belonging to an associated VO. The dynamic provisioning middleware, however, is still responsible for starting the VM. Only after a VM has booted and joined the pool does Condor proceed to run the job.

4.2. VM Management

Management of the VMs comprising the individual VOCs is performed entirely by the watchdog, using the Condor queue as its data source. As the watchdog observes an increasing number of jobs associated with a particular VO, it attempts to start additional VMs in order to increase the size of the corresponding VOC. The maximum number of available slots in which to start VMs is known to the watchdog, and it will not exceed the number of slots pre-specified by the system administrator. Furthermore, the total number of slots available across the physical fabric at a single site may be further subdivided among the VOs supported by that site. Thus, administrators can enforce flexible site policies to balance resource usage among different VOs.

Whenever the watchdog observes that a single VOC has more running VMs than there are jobs belonging to the corresponding VO in the queue, the watchdog will reduce the size of the VOC by terminating VMs that are not claimed by Condor. Since the VOC model [6] specifies the use of a single read-only disk image to spawn all VMs in a VOC, termination of VMs is accomplished instantly by means of killing the virtual machine monitor process. Once a VM has been terminated, its slot is re-claimed by the watchdog and added to the pool of unclaimed physical Condor slots.

5. Test Results

A prototype of the dynamic Virtual Organization Cluster scheduler was implemented, and tests of the system were conducted using synthetic workloads. For analytical simplicity, the system only supported a single VO, with a single virtual machine slot per physical host, for a total limit of 16 slots. This simplifying design assumption permitted the use of a minimalistic physical system policy, so that the performance and behavior of the unrestricted mechanism could be observed. As a further simplification, the VMs used for the VOC in this test were all spawned from a single 20 GB image on a shared filesystem. Figure 1 depicts the architecture of the test system hardware.

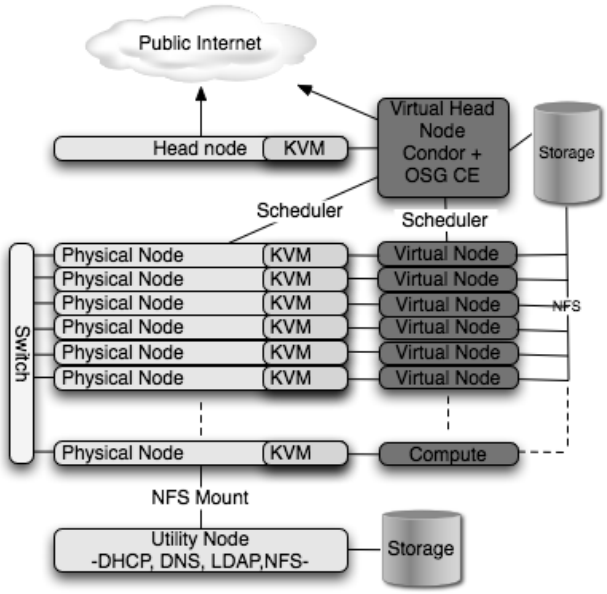

Figure 1. Test System Architecture

Several sets of tests were conducted using the prototype system, the first of which was an analysis of the approximate time required to boot each VOC such that it joined the Condor pool. Since all VMs started from an identical state

— the result of using a single virtual disk image to spawn all VMs — the boot times were assumed to be constant for all members of the same VOC. Two test suites were employed to observe job scheduling behavior. In the first suite, jobs were submitted locally: that is, directly to the Condor queue, without any use of Globus. Globus was used as the vehicle for job submission in the second suite, allowing its effects to be observed.

Each test suite consisted of five tests, in which the periodicity of job submission, size of each job group submission, and run length of each job were varied. These tests were arranged as follows:

- Two submission groups of 50 jobs each were submitted with sufficient temporal separation so as to execute the vast majority of jobs from the first submission group, prior to execution of the second submission group. This test was designed to simulate submitting large groups of jobs in which the results of the first group were retrieved before submitting the second group.
- Periodic sets of 10 jobs, each with a 10-second execution time, were submitted 90 seconds apart.
- Similar periodic sets of 10 jobs, each with a 10-second execution time, were submitted 30 seconds apart.
- Periodic sets of 10 short jobs, each with a 1-second execution time, were submitted 30 seconds apart.

The purpose of the latter three tests was to observe the behavior of the system under regular periodic loads, with variations in the period, job size, and per-job execution time. Execution times were varied in order to determine the sensitivity of the system to the boot time latency, while period and size variations were performed to test the responsiveness of the watchdog.

5.1. VM Boot Times

Tests were performed using local job submission (directly to the Condor queue) to measure the boot times for the VMs comprising the VOC. Boot times were measured from the time of initiation of the virtual machine to the time at which the VM joined the Condor pool. Actions performed as part of the boot procedure included allocation of memory to the VM, initialization of the VM kernel, acquisition of a DHCP lease and corresponding hostname by the VM, and the starting of run-time services. Since all VMs were spawned from a single image, it was not possible to configure the network settings statically, requiring dynamic configuration on a per-instance basis.

Submissions were performed in groups of 10 one-second jobs, with a period of 30 seconds between groups. The boot process for a VM was considered to be complete once it joined the Condor pool, as observed by the watchdog. As shown in figure 2, the first VM booted in response to incoming jobs joined the pool approximately 60 seconds after the first job was submitted, or about 55 seconds after the watchdog observed the first job and started the VM.

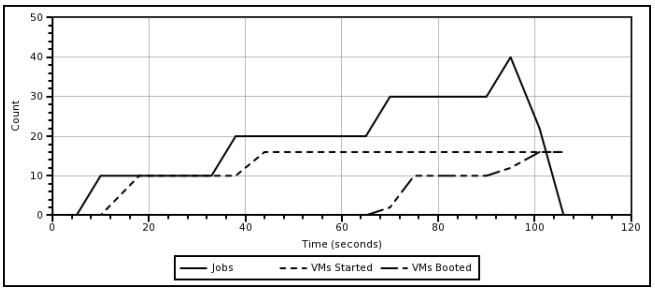

Figure 2. Virtual Machine boot delays relative to job submission time. VMs are started (VMs Started) in response to jobs arriving in queue (Jobs). Since each VM is a virtual Linux machine, there is a boot delay between VM start time and the time at which the VM joins the Condor pool and is ready to accept jobs. The total count of these booted machines (VMs Booted) corresponds to the size of the Condor pool for the VOC.

Since the watchdog required approximately 6 seconds to start all 10 initial VMs, a corresponding delay of approximately 7 seconds was observed between the time at which the first VM joined the Condor pool and the time at which the tenth VM joined the Condor pool. At a test wall time of approximately 38 seconds, the watchdog responded to the second batch of submitted jobs and began to increase the size of the VOC, continuing until the 44 second mark, at which point the 16 VM slots were exhausted. The additional 6 VMs joined the Condor pool between wall clock times of 92 and 101 seconds, corresponding to boot times in the range of 54 to 57 seconds. No additional VMs could be started once the slots were exhausted at 101 seconds, after which point the 16 running VMs were able to complete the remaining 1-second jobs quickly.

Variations in VM boot time were expected, owing to the dynamic configuration processes that must occur during VM boot. Based on the test results, a conservative upper bound of 60 seconds was attributed to the VM boot process.

5.2. Jobs Submitted Locally

To effect completion of the first test suite, batches of jobs were submitted locally. The first test utilized two groups of 50 jobs each, with a sufficiently long delay between submission to allow all but two of the first batch to complete before submitting the second batch. As shown in figure 3, the watchdog started the maximum number of VOC nodes by 20 seconds into the test. The majority of the first set of jobs completed rapidly between 162 and 198 seconds wall clock time, or about 178 seconds after submission. Given the 60-second boot time assumption, the total run

time for 48 10-second jobs on 16 VMs was approximately 118 seconds. Between sets, the watchdog was observed to reduce the size of the VOC from 16 VMs to 2 VMs. The second set of jobs completed in approximately half the time as the first, with a clear "step-down" pattern observed in the Condor queue between 280 and 310 seconds. While this pattern can be partly attributed to the need to reboot the VMs that were stopped during VOC contraction, it is important to remember that Condor only schedules jobs on a periodic basis, as it is designed for high job throughput, not high performance. Therefore, some of the "step-down" behavior could be attributed to Condor simultaneously starting fewer jobs than were slots available.

As shown in figures 4 through 6, the watchdog continued to exhibit predictable behavior similar to that observed in the first test. Whenever the number of jobs waiting in the queue dropped below the number of running VMs in the VOC, the VOC was contracted by terminating VMs. Conversely, whenever more jobs were waiting than VMs were running, as long as VM slots remained available, the watchdog expanded the VOC by adding VMs. A noticeable delay between queue size and VM count was observed in both cases, which was the exact behavior expected from the periodic sampling done by the watchdog. Longer delays were observed between initial job submission and job completion, owing to the time required to boot the VMs.

The effect of extremely short jobs, or exceptionally long periods related to job execution length, were evident in the results, as illustrated in figures 4 and 6. Since the watchdog employed a simplistic VM scheduling policy, it terminated VMs as soon as VOC sizes were found to exceed queue sizes. This aggressive VM termination had a negative effect on throughput, as illustrated by comparing figure 4 to figure 5 and figure 5 to figure 6. In the case of relatively long periodicity relative to execution time, it was necessary to re-expand the previously contracted VOC, thereby incurring VM boot delays. For exceptionally short-running jobs submitted regularly, the initial cluster of submissions completed quickly, allowing the entire VOC to be removed from operation. Two interesting, but mutually disadvantageous, phenomena were observed after the next group of jobs arrived in the queue (figure 6 at approximately 130 seconds wall time). Due to caching of the virtual machine monitor process, and its initial read-only VM data, on the physical host, the VOC nodes were able to boot somewhat faster during VOC re-start. However, the rapid job execution and simplistic scheduling algorithm in the watchdog combined to limit the total size of the VOC to 10 nodes following the re-start. The result of this combination of properties was exceptionally low throughput following the restart.

5.3. Jobs Submitted Through Globus

A second suite of identical tests was executed using the Globus job manager as the submission vehicle. In order to avoid submission errors, it was necessary to introduce a small delay of two seconds between the submissions of individual jobs. As a result, the process of submission of a batch of jobs through the Globus system was longer than the process of local submission, resulting in the slower queue size growth visible in figures 7 through 10. In addition, some non-constant delays were observed between submission of jobs to Globus and the time at which the jobs were delivered to Condor. This delay, especially evident in the queue size jitter shown in figure 8, was not particularly alarming, once again due to the emphasis taken by both systems on high throughput, as opposed to high performance.

Although net throughput was slightly reduced by the late arrival of the last set of jobs, the addition of Globus as a "buffer" reduced the extremity of the VOC contraction. While the size of the VOC was reduced, necessitating the rebooting of some nodes, the VOC was never completely removed from operation as it was in the local test. This buffering effect was not observed with extremely short jobs, as depicted in figure 10, where the VOC was briefly completely terminated.

6. Conclusions

Virtual Organization Clusters provide a mechanism by which each Virtual Organization may have its own dedicated computational clusters on grid sites. VOCs can be dynamically instantiated, expanded, contracted, or terminated in response to the size and number of jobs submitted by users affiliated with each individual VO. With dynamic scheduling of VOCs, it is possible to specify policies that limit the number of concurrent virtual machine instances running simultaneously for a single VOC, thereby effectively constraining the maximum size of individual VOCs. Physical grid sites can thus share limited hardware resources among different VOs, while providing each VO its own dedicated cluster execution environment.

Early performance tests of a prototype VOC scheduler were encouraging. The VOC nodes were dynamically added and removed from operation, without any intervention by the system administrator or other grid middleware. While the performance of the system was negatively impacted by over-responsiveness of the watchdog in removing VOCs from operation, it is likely that parameterized tuning of VM start-stop behavior would be able to mitigate some of the performance degradation. Future research will be conducted to analyze optimal policies for starting and stopping VMs in response to grid loads.

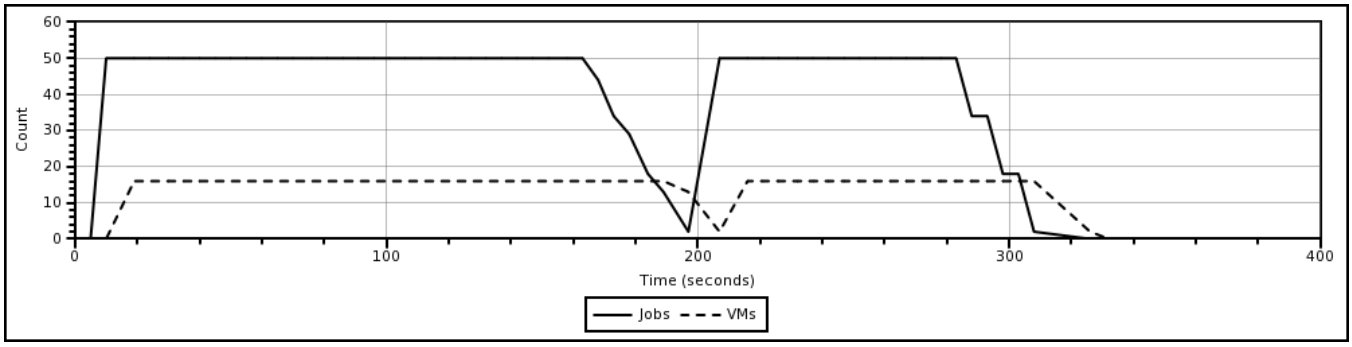

Figure 3. Two submissions of 50 jobs, 10-second execution time, submitted locally

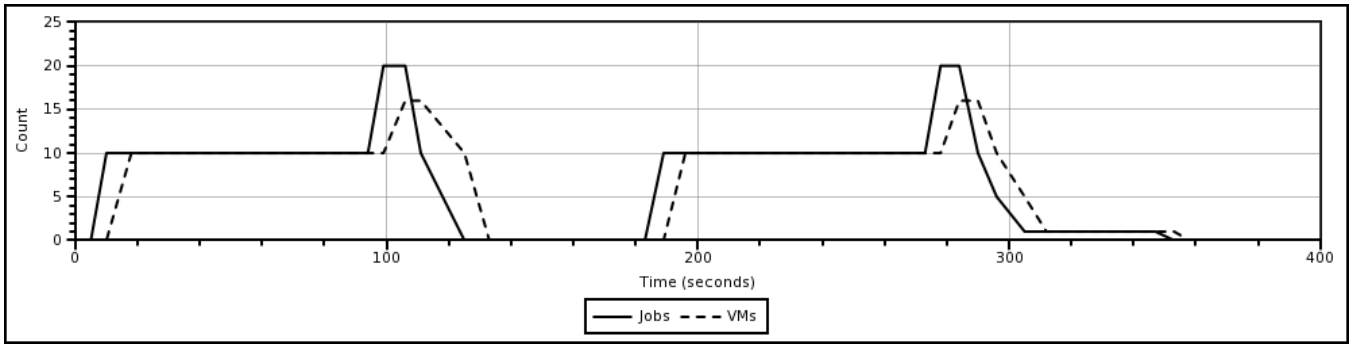

Figure 4. Submitted 10 jobs every 90 seconds, 10-second execution time, submitted locally

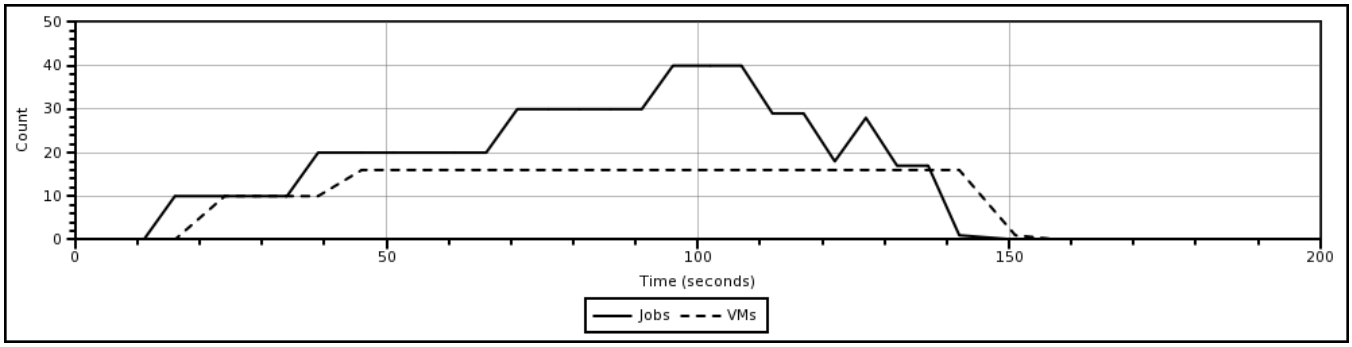

Figure 5. Submitted 10 jobs every 30 seconds, 10-second execution time, submitted locally

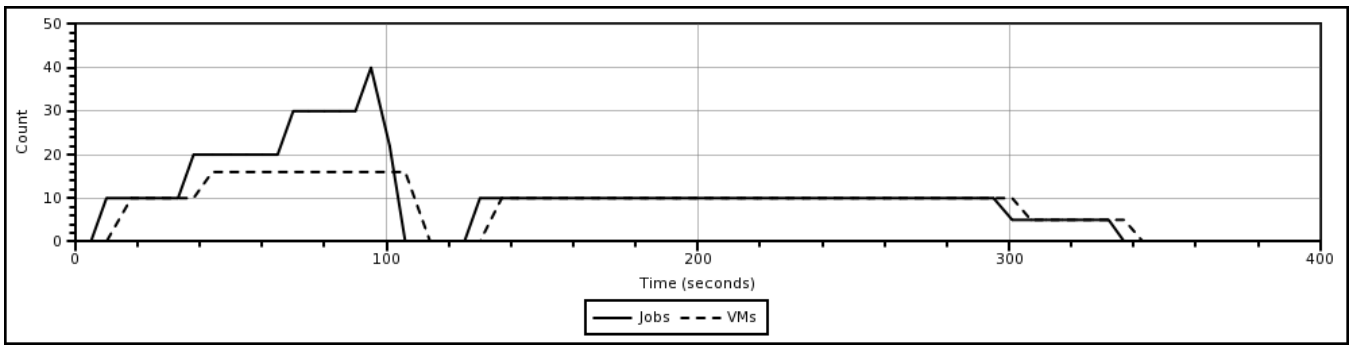

Figure 6. Submitted 10 jobs every 30 seconds, 1-second execution time, submitted locally

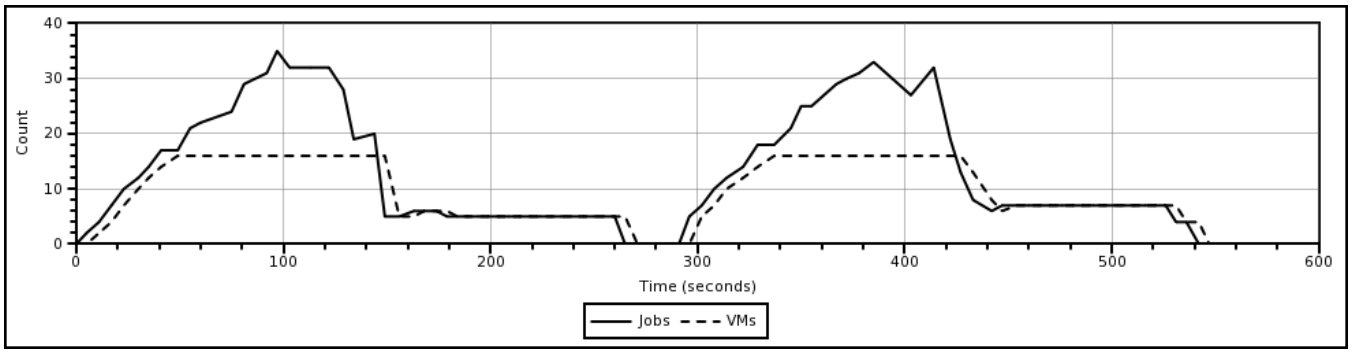

Figure 7. Two submissions of 50 jobs, 10-second execution time, submitted through Globus

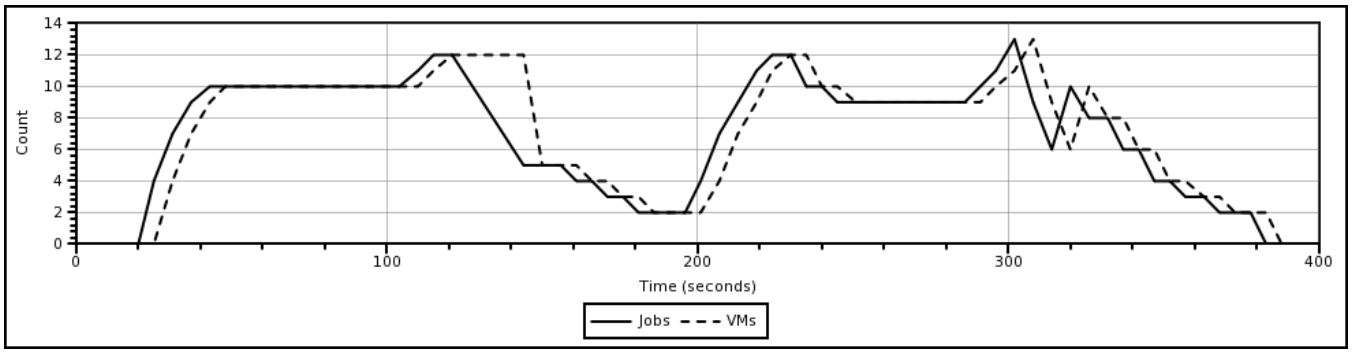

Figure 8. Submitted 10 jobs every 90 seconds, 10-second execution time, submitted through Globus

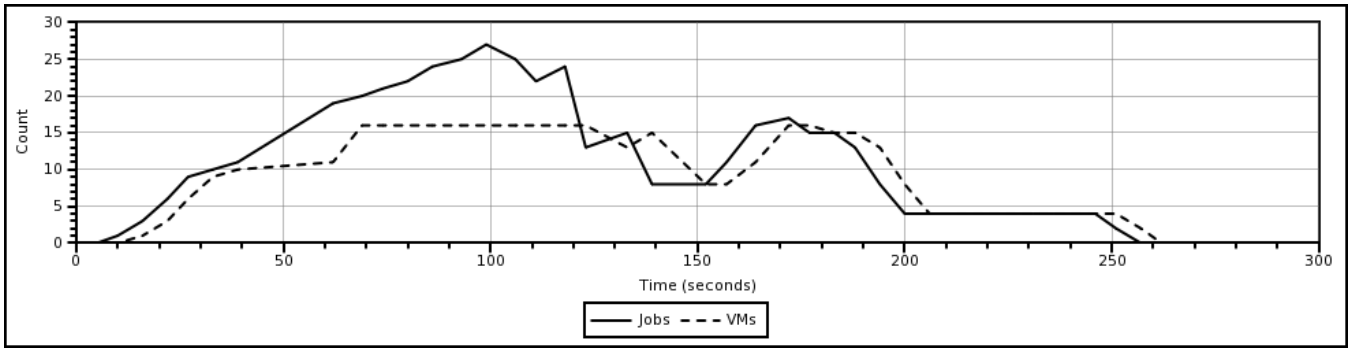

Figure 9. Submitted 10 jobs every 30 seconds, 10-second execution time, submitted through Globus

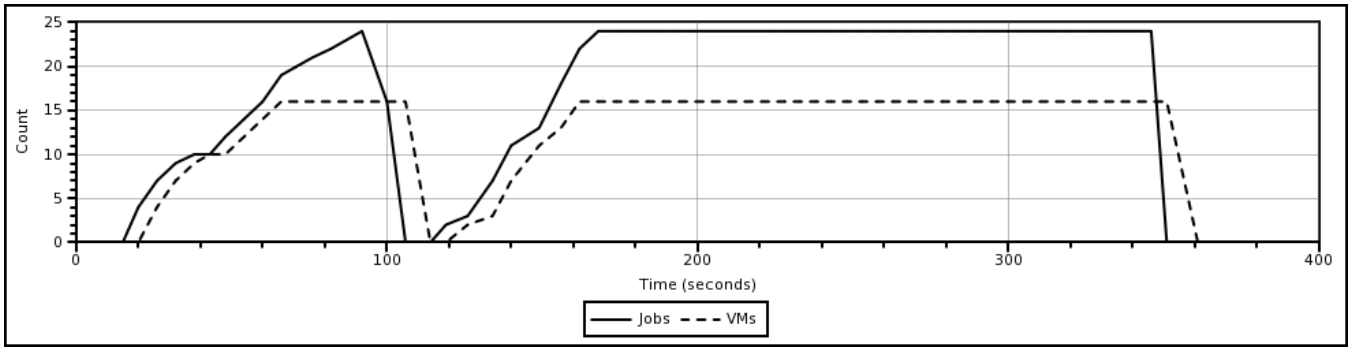

Figure 10. Submitted 10 jobs every 30 seconds, 1-second execution time, submitted through Globus

Acknowledgment

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *International Journal of Supercomputing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[2] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[3] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the Grid," in *11th International Euro-Par Conference*, Lisbon, Portugal, September 2005.

[4] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang, "Virtual clusters for grid communities," in *CCGrid 2006*, Singapore, May 2006.

[5] W. Emeneker and D. Stanzione, "Dynamic virtual clustering," in *IEEE Cluster 2007*, Austin, TX, September 2007.

[6] M. A. Murphy, M. Fenn, and S. Goasguen, "Virtual organization clusters," in *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, February 2009.

[7] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, June 2003.

[8] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing grids: the In-VIGO system," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896–909, June 2005.

[9] H. Nishimura, N. Maruyama, and S. Matsuoka, "Virtual clusters on the fly - fast, scalable, and flexible installation," in *CCGRID 2007: Seventh IEEE International Symposium on Cluster Computing and the Grid*, May 2007.

[10] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High Performance Distributed Computing (HPDC 2008)*, 2008.

[11] A. M. Matsunaga, M. O. Tsugawa, S. Adabala, R. J. Figueiredo, H. Lam, and J. A. B. Fortes, "Science gateways made easy: the In-VIGO approach," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 905–919, April 2007.

[12] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and managing virtual machine execution environments for grid computing," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004.

[13] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputing Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[14] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press, October 2001.