Self-Provisioned Hybrid Clouds

Linton Abraham, Michael A. Murphy, Michael Fenn, and Sebastien Goasguen
School of Computing
Clemson University
Clemson, SC 29634-0974 USA
{labraha, mamurph, mfenn, sebgoa}@clemson.edu

Abstract

Virtual Organizations are dynamic entities that consist of individuals and/or institutions established around a set of resource-sharing rules and conditions. The VO may require the use of on-site (local) and off-site (public) compute resources that can be leased or autonomically provisioned, based on workload and site policies. Virtual Organization Clusters provide the necessary computing infrastructure by building upon existing physical grid sites without disrupting the existing infrastructure or requiring any engagement from end users. VOCs also separate the physical and virtual administrative domains and thus encourage more sites to participate in the resource sharing and hosting. The VO can relinquish the compute resources based on job completion or other operational parameters such as cost. This paper expands on previous work with the Virtual Organization Cluster Model by demonstrating its scalability across multiple grid sites with the use of a structured peer-to-peer overlay networking system. A novel approach by which the model can extend to lease-based systems, such as the Amazon Elastic Compute Cloud (EC2), is introduced.

1. Introduction

The availability of large-scale computing resources for research is an absolute requirement for domain scientists and users. For example, as telescopes grow bigger and extend further into outer space, astronomers and physicists are forced to confront the overwhelming vastness of the universe. Consequently, data and research that simulate various facets of celestial bodies require enormous computing power. Rising costs that are associated with the manpower, energy and hardware required to accommodate these large distributed applications, adversely affect deployment. Virtual Organization Clusters (VOCs)[1] and Cloud Computing enable researchers to minimize these costs. VOCs enable Virtual Organizations to leverage the resources of multiple participating grid sites, allowing greater utilization of existing hardware and increased resource availability for end users. Figure 1 shows the primary design and architecture of a VOC with a single participating grid site. This architecture was explained in detail in [2].

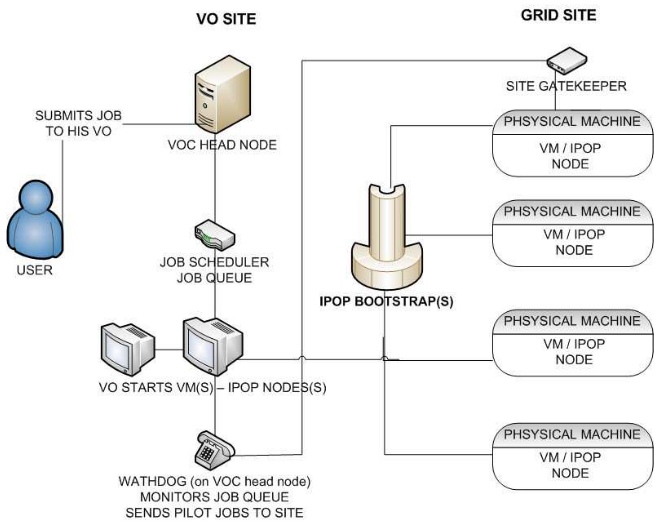

Figure 1. A Single Grid Site VOC

Cloud Computing allows scientists and other users to run their applications on leased systems without incurring the direct costs of acquiring and maintaining hardware. Cloud Architectures [3] address major issues that arise from large-scale data processing. In traditional data processing environments, it is difficult not only to get as many machines as an application requests but also to get a resource allocation in a timely manner. It is also difficult to distribute and co-ordinate a large-scale job on different machines and provision an appropriate infrastructure to recover from failures. Traditional architectures also are inefficient when adapting nodes to dynamic workloads, especially when systems must be re-imaged to change software stacks to support different users. Applications built from Cloud Architectures are hosted in the networks of systems, where the physical location of the infrastructure is determined by the cloud provider. Most Cloud providers offer simple APIs to Internet-accessible services that scale on demand, hiding the complex reliability and scalability logic behind services. Cloud computing provides high utilization and cost optimization for hosted applications.

The primary purpose of this paper is to establish a foundation by which the VOC model can be extended to leverage generic cloud resources. In the remainder of this paper, related work is discussed in Section 2. The VOC Model is described in Section 3. Section 4 discusses the overlay network that links widely distributed resources into a dedicated private cluster. A procedure for conducting operational testing of a VOC and the design and architecture of a prototype cloud system is discussed in Section 5. Experimental results are presented in Section 6. Some initial conclusions are presented Section 7.

2. Related Work

Virtualization was first introduced in the late 1960s. It separates a virtual operating system known as the "guest" system from the physical machine known as the "host" system. The primary motivation behind virtualization has always been to decouple software applications from the underlying physical hardware implementation. [4] Virtualization of grid systems was first proposed in [5] and provides much required abstraction and usability to grid users and system administrators. In a virtualized environment, access can be granted as per the administrators needs. This reduces the potential harm that can be done to the hardware. Hardware multiplexing is also possible; the same hardware can be used to host different kinds of Virtual Machines. [5] Higher-level system components may also be virtualized; examples include virtual networking systems such as ViNe [6], VNET [7], VDE [8], and IPOP[9]. Virtualization at the application layer has been realized in systems such as In-VIGO [10].

Shirako [11], is a lease-oriented model in which the resource allocation is explicitly handled by automated brokers. Cluster-On-Demand (COD) [12] is an automated system that performs rapid re-installation of physical machines or Xen virtual machines [13] and provides the back-end clustering needed by Shirako. The Grid Resource Oversight Coordinator (GROC) permits physical resources to be leased by Virtual Organizations for the purpose of deploying completely virtualized grids. The physical grid site host a virtual cluster belonging to the same VO, but does not span individual clusters across multiple grid sites. [14]

Globus Virtual Workspaces allocates a Virtual Workspace given a description of the hardware and software requirements of an application. This allows the workspace to be instantiated and deployed on a per-application basis. The Workspace must then be explicitly deployed on a host site and then access is derived to run computational jobs. [15] By utilizing a leasing model, Virtual Workspaces can provide high-level, fine-grained resource management to deliver specific Quality of Service guarantees to different applications using a combination of pre-arranged and best-effort allocations [16]. By leasing multiple resources simultaneously, Virtual Workspaces can also be aggregated into clusters [17].

Local clusters often need to be extended within the same LAN. Autonomic virtualization overlay systems like VioCluster have been developed for that sole purpose. The concept of dividing each cluster into a physical and a virtual domain was first introduced with the VioCluster. Virtual domains are transparently and autonomically re-sized by means of a broker application, which trades machine allocations between groups by following explicitly configured policies. This brokering process is transparent to end-users, permitting the virtualized cluster to be used as if it were a regular physical cluster. [18] By utilizing the Violin overlay network, virtual domains on several different physical clusters may be combined into a single execution environment with a private network space [19]. Dynamic Virtual Clustering allows clusters of virtual machines to be instantiated on a per-job basis for the purpose of providing temporary, uniform execution environments across clusters co-located on a single research campus. These clusters comprise a Campus Area Grid (CAG), which is defined as "a group of clusters in a small geographic area … connected by a private, high-speed network" [20]. Latency and bandwidth properties of the private network are considered to be favorable, thereby allowing a combination of spanned clusters to function as a single high-performance cluster for job execution. However, the software configurations of the different component clusters may differ, as the component clusters may belong to different entities with different management. DVC permits Xen virtual machines with homogeneous software to be run on federated clusters on the same CAG whenever the target cluster is not in use by its owner, thereby allowing research groups to increase the sizes of their clusters temporarily [20].

Virtual Organization Clusters (discussed in detail in Section 3)[1] differ from explicit leasing systems such as Globus Nimbus and Shirako, in that virtual clusters are leased autonomically through the use of pilot jobs, which provide for dynamic provisioning in response to increasing workloads for the associated VO [2]. VOCs remain transparent to end users and to non-participating entities, while enabling participating VOs to use overlay networks, such as IPOP [9], to create virtual environments that span multiple physical domains. Unlike Campus Area Grids and VioCluster environments, however, these physical domains are grid sites connected via low-bandwidth, high-latency networks. These unfavorable connections, coupled with overheads introduced by the addition of virtualization systems, make VOCs better suited to high-throughput, compute-bound applications than to high-performance applications with latency-sensitive communications requirements [21].

OpenVPN (discussed in Section 4)is an open source virtual private network (VPN) tool that facilitates the creation of point-to-point or server-to-multi client encrypted tunnels between host computers. It does satisfy three of the core requirements of the VOC model, but requires users to handle load balancing between multiple servers [22]. OpenVPN cannot handle the autonomic addition or removal of servers and clients without making significant modifications to the routing tables of a few (or all) participating nodes in the system.

IPOP or Internet Protocol over Peer-to-Peer [9], is a tunneling software that uses a peer-to-peer architecture rather than a client-server architecture. IPOP Brunet [23], a software library for P2P networking written in C# and developed using Mono. It's key features include:

1) Network transports are abstracted as Edge objects. Thus enabling TCP, UDP and TLS/SSL transports,
2) Completely distributed UDP NAT traversal,
3) Implementation of Chord/Kleinberg/Symphony [24] type ring topology for routing,
4) A complete DHT implementation and
5) Distributed tunneling system to maintain topology in the presence of some routing difficulties (untraversable NATs, BGP outages, firewalls, etc.).

IPOP [9] is a self-configuring IP-over-P2P virtual network overlay which provides the capability for nodes behind NATs and some firewalls to all appear to be in the same subnet. Besides being transparent to applications, IPOP was deployed in the VOC [1] model for the reasons explained in Section 4.

3. Virtual-Organization Clusters

Virtual Organization Clusters (VOCs), described more thoroughly in [1], enable the creation of virtual cluster environments that are compatible across sites, deploy-able without per-node replication, transparent to end users, implementable in a phased and non-disruptive manner, optionally customizable by Virtual Organizations, and designed according to a specification that permits formal analysis. Since VOCs are constructed from virtual machines (VMs), and multiple VM instances can be spawned from a single image, VOC environments are nominally homogeneous (and therefore software compatible) across grid sites. If each grid site utilizes a central distributed file system store such as PVFS [25], VOC nodes can be booted directly from the shared file system, without staging the multi-gigabyte image files to the physical compute nodes. Once operational, VOCs remain completely transparent to the end user, since the virtual environments are autonomically managed without explicit resource reservation requests. VOCs also remain transparent to Virtual Organizations and other entities that choose not to deploy them, allowing VOC implementations to be added to existing production grids without disrupting the operational infrastructure. Different technologies can be utilized to implement VOCs, since a VOC is simply an implementation of a system that conforms to the specifications presented in the VOC Model.

A key specification of the VOC Model is the explicit separation of administrative domains. Each physical site on the grid is a unique Physical Administrative Domain (PAD), which is managed by local administrators. Components of each PAD include the physical computing resources, networking interconnections, and all associated infrastructure, including power distribution and cooling. Each hosted VOC is a separate Virtual Administrative Domain (VAD) that is managed by the owning VO or an agent thereof. This explicit administrative access permits each VAD to have a customized software environment. Moreover, the separation of administrative domains implies a separation of policy authority. Local site owners and VOC owners may implement their own resource allocation, scheduling, and management policies. These policy decisions are all independent: no coordination is required between the VOC administrators and the site administrators. VOCs are explicitly a "best-effort" system and will execute on any physical site willing to provide VM hosting services.

Jobs can be submitted directly to a VOC through a dedicated, grid-facing head node. This cluster head node contains the necessary grid interconnection software (for example, Globus [26]) and appears as a compute element on the grid. As the size of the job queue on the private head node increases, pilot jobs are submitted to grid sites to obtain physical resources. In turn, these pilot jobs start virtual machines, which are dynamically configured, or "contextualized" [27], at boot time to start the overlay network and join the scheduler pool on the private head node. User jobs are then executed on the virtual cluster as provided by the scheduling policies implemented in the VOC. Figure 2 shows the design of a VOC that spans multiple grid sites. Implementation of this design is detailed in Section 6.

4. Peer-to-Peer Overlay Networks

The VOC model does not require the use of any specific networking model. A potential client-server architecture and peer-to-peer architecture are discussed in this section.

Virtual networking systems that enable VOCs to span multiple resources must be able to:

1) Support an ad-hoc network, i. e. support the dynamic addition and removal of nodes based on workload,
2) Load balance between multiple servers efficiently and without user involvement,
3) Assign IP addresses dynamically,
4) Route traffic efficiently between machines in the pool,
5) Have the capability to traverse NAT and firewalls.

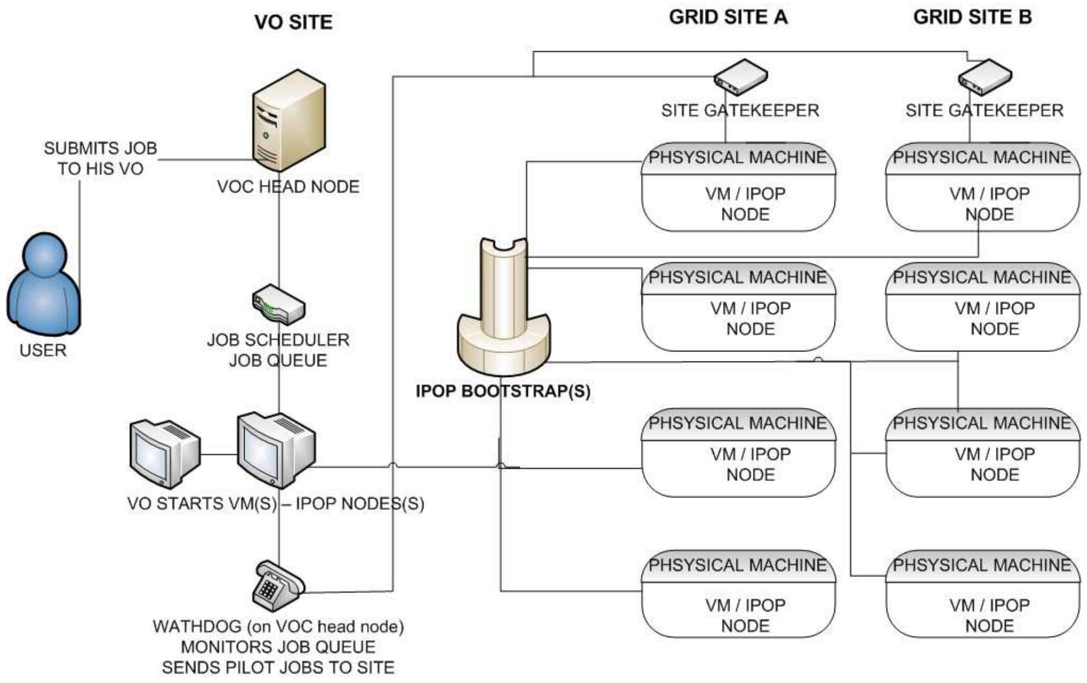

Figure 2. A VOC across multiple grid sites

4.1. Client-Server Architectures

Client-server architectures rely heavily on centralized servers for connectivity and routing. In many VPN systems such as OpenVPN [22], the server is used not only to assign addresses to the clients but also function as a STUN (Simple Traversal of User Datagram Protocol [UDP] Through Network Address Translators [NATs]) server. This often leads to a bottleneck in the server. While OpenVPN supports load balancing among multiple servers, users have to set up routing tables to manage the clients. Such systems also require heavy modifications as servers enter or leave the system. Therefore, scalability of client-server based systems has always been a major cause of concern.

4.2. Peer-to-Peer Architecture

Peer-to-Peer systems have an architecture where all the participating nodes act as peers. There is no centralized server and hence no single point of failure in a such system. Peer-to-peer systems often implement an application layer overlay network on top of the native or physical network topology. Such overlays are used for indexing and peer discovery. Content is typically exchanged directly over the underlying Internet Protocol (IP) network. Anonymous peer-to-peer systems are an exception, and implement extra routing layers to obscure the identity of the source or destination of queries. Peer-to-peer networks are typically formed dynamically by ad-hoc additions of nodes. In an ’ad-hoc’ network, the removal of nodes has no significant impact on the network. The distributed architecture of an application in a peer-to-peer system provides enhanced scalability and service robustness. [28]

As mentioned in Section 2, IPOP or IP-over-P2P can be used for the networking in the VOC Model. Details the IPOP implementation used can be found in Section 6. IPOP is a structured P2P based system. Structured P2P systems employ specific algorithms that help to establish connections within the overlay through the use of Distributed Hash Tables (DHTs) to store node-specific information. DHTs are a class of decentralized distributed systems that provide a lookup service similar to a hash table. Key-value pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures [29].

Unstructured P2Ps have no algorithm for organization or optimization of network connections. Such systems are similar to client-server systems in the fact that a central server is often used for index functions and to bootstrap the entire system. Napster [30]is an example of an unstructured P2P system.

5. Test Procedure

Two types of tests are performed: the first is a long-term test of a transparent VOC, and the second is a test of a VOC that spans multiple grid sites via an overlay network. These tests could not be run concurrently as the implementation of an overlay network would disrupt the operational testing.

5.1. Operational Testing

This section discusses a set of operational tests conducted on a cluster of 16 nodes. In these experiments, Virtual Organization Clusters were transparently provided to specific Virtual Organizations on the Open Science Grid, and jobs from those VOs were executed in 32-bit CentOS virtual machines. On June 1, 2009, a short operational test was started, in which a single VOC was placed into service on behalf of the Engage VO, using the delayed response watchdog provisioning algorithm with a minimum of two VMs and a maximum of sixteen VMs. After approximately 44 hours of testing, the VOC was removed from service on June 3, 2009.

Following the short operational test, a long operational deployment – approximately two months in length – was effected using the same watchdog algorithm. Two VOCs were attached to the Open Science Grid, with one VOC dedicated to the Engage VO and the other VOC dedicated to the NanoHub VO. Both VOCs were set to a minimum size of two VMs and a maximum size of sixteen VM, which resulted in utilization of all 32 physical CPU cores whenever both VOCs were at maximum size. The long-running operational experiment commenced on June 4, 2009 and completed on August 17, 2009.

5.2. Overlay Network Testing

This section discusses a prototype implementation of a system for dynamically provisioning nodes across multiple grid sites. As illustrated in Figure 2, the user submits jobs to the VO's Grid Compute Element (CE). The user does not need to be concerned about how the resources he requires are obtained and allocated. The dynamic provisioning (resizing of the VOC) is performed by a watchdog that is set up on the VOC head node. In the current implementation, the watchdog uses a simple greedy algorithm. The watchdog monitors incoming jobs on the gatekeeper. When a job enters the local queue, the watchdog determines whether or not it should start a Virtual Machine (VM) on one of the physical hosts. If started, the virtual machine then joins the IPOP pool. The watchdog continues to monitor the job queue and starts VMs depending on the workload. The local scheduler (e.g. Condor [31], PBS [32], LSF [33]) schedules the jobs on the virtual machines. Once jobs are executed and there are more VMs than jobs, the watchdog terminates idle VMs.

Figure 2 also shows that each VO has one watchdog. Once all the local compute resources are exhausted, the watchdog contacts the gatekeeper on another participating grid site. It then starts up VMs on that site and those machines join the VOs pool as well.

Experiments were conducted with a lease based system called the Amazon Elastic Compute Cloud (EC2) [34]. The local (on-site) compute resource consisted of 16 VMs dedicated to the Engage VO [35], which consisted of 16 dual core machines with each core dedicated to a VO. The implemented watchdog is able to handle both machines in the local cluster and on Amazon EC2. The local cluster specifications and the original implementation of the watchdog are explained in detail in [2]. The watchdog for this experiment was designed to handle the complexities of multiple grid sites, the local cluster and a set of EC2 instances in this case.

The basic watchdog algorithm is:

1) Invoke PROBE_STATE, a module to determine the number of jobs in the schedule's queue,
2) Examine the jobs to VMs ratio and enter a the appropriate state, described below,
3) Take an action based on the current state,
4) Wait for a given interval (10 seconds in the prototype).

While the VOC Model supports any batch scheduler, the prototype implementation uses the Condor job manager [31]. The PROBE_STATE module is used to determine the state of the Condor queue when invoked. One of four possible is then entered:

1) Number of Jobs in the queue is GREATER than the number of VMs reporting AND limit on number of local VMs HAS NOT been reached.
2) Number of Jobs in the queue is GREATER than the number of VMs reporting AND limit on number of local VMs HAS been reached.
3) Number of Jobs in the queue is LESSER than the number of VMs reporting AND MORE VMs than the local cluster's limit are reporting.
4) Number of Jobs in the queue is LESSER than the number of VMs reporting AND LESSER VMs than the local cluster's limit are reporting.

Separate modules handle each state. State 1 invokes a START_LOCAL module that only starts machines on the local cluster. The module communicates with the gatekeeper using Globus. The gatekeeper contacts the head node to start up local VMs.

State 2 invokes a START_EC2 module used to start EC2 instances, once local resources have been exhausted. The module uses the standard EC2 API command ec2-run-instances to boot an instance. Both the local and EC2 nodes are configured to join the IPOP pool and are part of a single VO. Each Amazon instance is pre-configured with the software packages that are required by the VO. The nodes

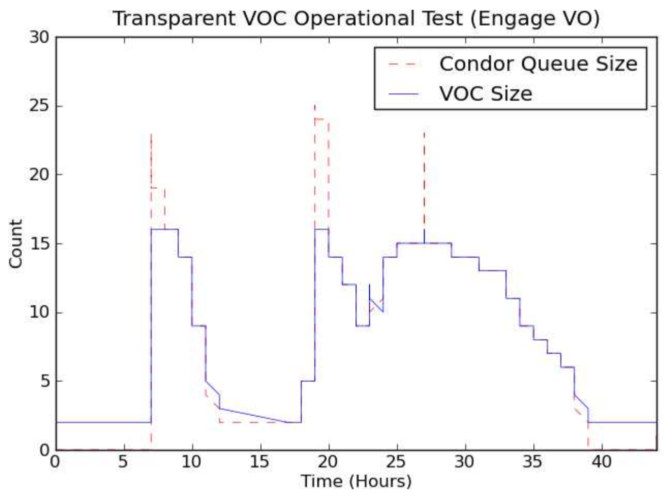

Figure 3. Short operational test (44 hours) with the physical cluster configured to support a 16-node Virtual Organization Cluster dedicated to the Engage Virtual Organization.

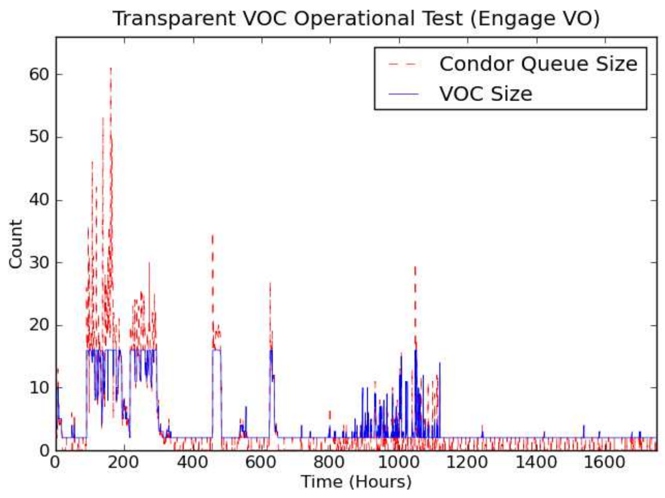

Figure 4. Long operational test: Engage VO. A second operational VOC, dedicated to the NanoHub VO, was sharing the same hardware.

are also configured to join the IPOP overlay once booted. Since Amazon bills a partial hour of usage as a full hour, a limit of a 100 instances is imposed in order to control costs. Thus an EC2_instance_counter is maintained to ensure that the limit isn't exceeded.

State 3 invokes a STOP_EC2 module when the number of jobs is less than the number of VMs, terminate priority is given to the EC2 instances. The STOP_EC2 module invokes another module called CHECK_IDLE. This module ensures that only the IP addresses of idle VMs are returned, guaranteeing that VMs that are scheduled and claimed will not be terminated. The system design ensures that all EC2 instances are terminated before the termination of the local VMs, ensuring that local reasources are more heavily utilized than resources provided by off-site grid locations. Once all the instances are terminated, the watchdog shuts down VMs in the local cluster after ensuring that the number of jobs is less than the number of VMs

State 4 invokes a STOP_LOCAL module that shuts down local VMs. As in the STOP_EC2 module, the CHECK_IDLE module returns idle VMs. Using this list, the STOP_LOCAL module terminates VMs. In both the STOP_LOCAL and STOP_EC2 modules, the systems are terminated by sending the 'poweroff' signal over secure shell.

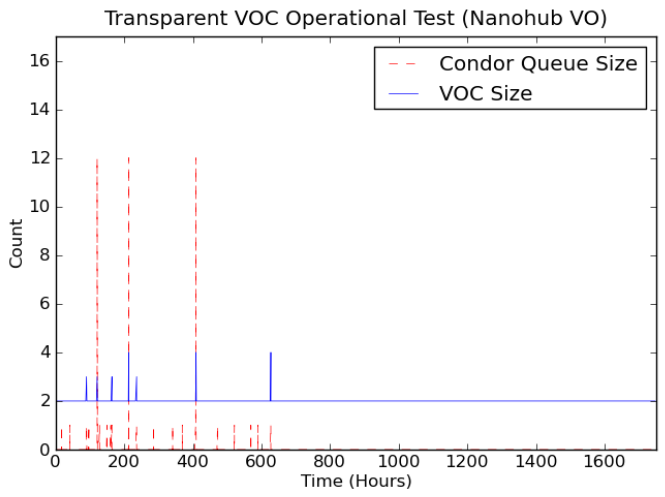

Figure 5. Operational test: NanoHub VO. A second operational VOC, dedicated to the Engage VO, was sharing the same hardware.

6. Results

6.1. Operational Results

As visualized in Figure 3, several bursts of jobs arrived during the short test period, and these bursts were accommodated by increasing the size of the VOC to the maximum specified level (16 nodes). All virtual nodes were hosted directly by the prototype system in order to contain costs, and no leased EC2 instances were employed.

As illustrated in Figure 4, jobs associated with the Engage VO continued to arrive in bursts for the first two thirds of the long test period, resulting in temporary increases in VOC size. Bursts of jobs associated with the NanoHub VO (Figure 5) were less frequent, significantly smaller in size, and limited to the first third of the long test period. Subsequent analysis determined that most NanoHub jobs, and an increasingly larger number of Engage jobs, required 64-bit operating environments. Since the prototype VOCs provided 32-bit environments, fewer jobs were sent to the prototype system as August approached.

After completion of the operational tests, the prototype system became obsolete for research purposes. The Intel Xeon processors installed in the physical compute nodes were equipped with the first generation of virtualization extensions, which did not include extended page tables or virtualized Input/Output devices. Rather than immediately discarding the hardware, a single Virtual Organization Cluster was deployed for the STAR VO [36], using a custom virtual machine image provided by the VO administrators. In this deployment, the VOC was still provided transparently on OSG by the system, although the VO provided the software stack in a sort of "semi-transparent" arrangement. Thus, the prototype implementation was converted into a production system, and it was still in service as of October 2009.

6.2. Overlay Network Results

Figure 6 shows the short jobs that were run. Five hundred, 10 second sleep jobs submitted in one batch. As seen in the figure, since the jobs are short and embarrassingly parallel, the EC2 instances and local VMs are still running for a period of time after all the jobs are complete. In Figure 7, three hundred 10 second sleep jobs were submitted in one batch. As these were longer jobs, the instances were terminated before the all the jobs were completed because of the use of the local VMs. Therefore, the system is better suited to longer jobs due to the cost factor inherent in using the leased resources. The use of compute resources located at multiple grid sites ensures that adequate compute resources are always available to the VO and hence the end user. The overlay network allows nodes in any geographic location to be a part of the VO. To the user, it all appears as a large number of machines in the same pool.

7. Conclusions

By dynamically provisioning compute resources, VOCs provide high utilization and resource availability as shown in the results. The use of multiple grid sites and the subsequent availability of resources is achieved without any involvement from the end user. The inclusion of the IPOP overlay network and the pilot jobs (used by the watchdog to send requests to multiple grid sites) enables Virtual Organizations to allocate resources and schedule jobs privately. With the use of more IPOP bootstrap nodes, the system is highly

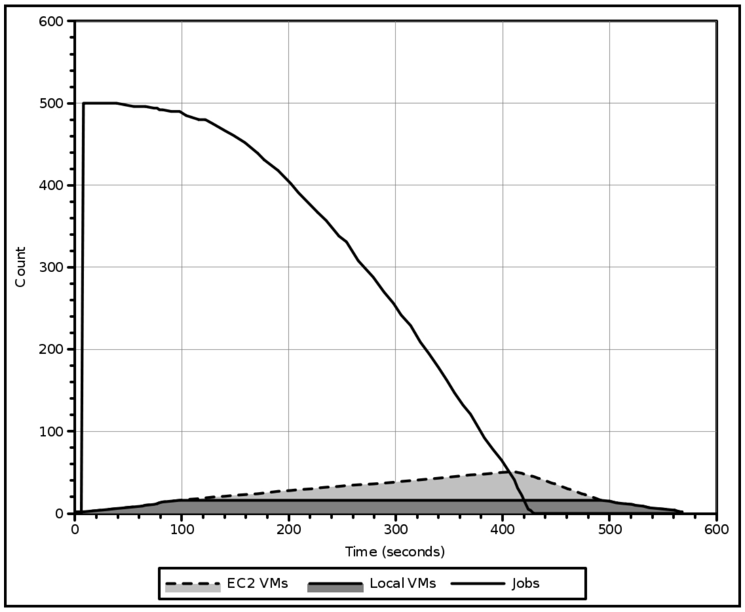

Figure 6. Short Jobs - 500 x 10 second jobs

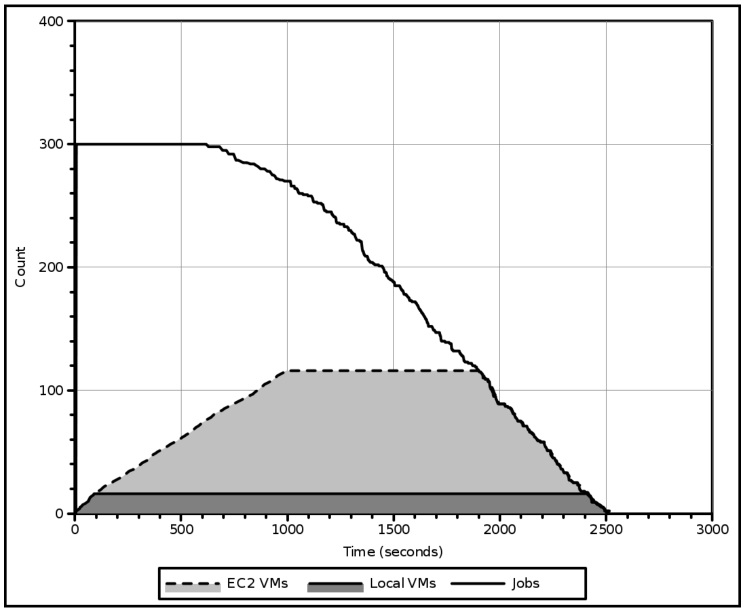

Figure 7. Long Jobs - 300 x 10 minute jobs

scalable and capable of spanning multiple grid sites and enables cross-domain management of the virtual computing nodes. The overhead introduced by IPOP [9] [2] is within acceptable limits for compute-bound tasks.

It has also been shown that the VOC model can be extended efficiently to leasing models such as Amazon EC2 [34], Shirako [37] and DVC [38]. The current watchdog uses a greedy implementation and might increase the cost of the system. Work is being done to implement an autonomous watchdog that monitors the states of the machines and provisions based on the running time and other requirements of jobs coming in. VOCs require no particular networking or operational environment to provide end users with a highly stable computing infrastructure. The experiments and results discussed in this paper illustrate one of several ways this model can adapt to requirements; showcasing the versatility of the VOC Model

References

[1] 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.

[2] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, “Dynamic provisioning of Virtual Organization Clusters,” in *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid ’09)*, Shanghai, China, May 2009.

[3] A. Weiss, “Computing in the clouds,” *netWorker*, vol. 11, no. 4, pp. 16–25, 2007.

[4] R. Figueiredo, P. A. Dinda, and J. Fortes, “Resource virtualization renaissance,” *Computer*, vol. 38, no. 5, pp. 28–31, May 2005.

[5] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, “A case for grid computing on virtual machines,” in *23rd International Conference on Distributed Computing Systems*, 2003.

[6] M. Tsugawa and J. A. B. Fortes, “A virtual network (ViNe) architecture for grid computing,” in *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[7] A. I. Sundararaj and P. A. Dinda, “Towards virtual networks for virtual machine grid computing,” in *Third Virtual Machine Research and Technology Symposium*, San Jose, CA, May 2004.

[8] R. Davoli, “VDE: Virtual Distributed Ethernet,” in *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom 2005)*, Trento, Italy, February 2005.

[9] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, “IP over P2P: Enabling self-configuring virtual IP networks for grid computing,” in *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[10] 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.

[11] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. Yocum, “Sharing network resources with brokered leases,” in *USENIX Technical Conference*, Boston, MA, June 2006.

[12] 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.

[13] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, “Virtual machine hosting for networked clusters: Building the foundations for ‘autonomic’ orchestration,” in *First International Workshop on Virtualization Technology in Distributed Computing (VTDC ’06)*, Tampa, FL, November 2006.

[14] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase, “Toward a doctrine of containment: Grid hosting with adaptive resource control,” in *19th Annual Supercomputing Conference (SC ’06)*, Tampa, FL, November 2006.

[15] 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.

[16] B. Sotomayor, K. Keahey, and I. Foster, “Combining batch execution and leasing using virtual machines,” in *17th International Symposium on High Performance Distributed Computing (HPDC 2008)*, 2008.

[17] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang, “Virtual clusters for grid communities,” in *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, Singapore, May 2006.

[18] P. Ruth, P. McGachey, and D. Xu, “VioCluster: Virtualization for dynamic computational domains,” in *IEEE International Conference on Cluster Computing*, Boston, MA, September 2005.

[19] P. Ruth, X. Jiang, D. Xu, and S. Goasguen, “Virtual distributed environments in a shared infrastructure,” *Computer*, vol. 38, no. 5, pp. 63–69, 2005.

[20] W. Emeneker and D. Stanzione, “Dynamic virtual clustering,” in *2007 IEEE International Conference on Cluster Computing*, 2007.

[21] M. Fenn, M. A. Murphy, and S. Goasguen, “A study of a KVM-based cluster for grid computing,” in *47th ACM Southeast Conference (ACMSE ’09)*, Clemson, SC, March 2009.

[22] J. Liu, Y. Li, N. V. Vorst, S. Mann, and K. Hellman, “A real-time network simulation infrastructure based on OpenVPN,” *J. Syst. Softw.*, vol. 82, no. 3, pp. 473–485, 2009.

[23] P. O. Boykin, J. S. A. Bridgewater, J. S. Kong, K. M. Lozev, B. A. Rezaei, and V. P. Roychowdhury. (2007, September) A symphony conducted by Brunet. Online. [Online]. Available: http://arxiv.org/abs/0709.4048

[24] G. Manku, M. Bawa, and P. Raghavan, “Symphony: Distributed hashing in a small world,” in *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, 2003. [Online]. Available: http://citeseer.ist.psu.edu/manku03symphony.html

[25] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur, “PVFS: A parallel file system for Linux clusters,” in *ALS’00: Proceedings of the 4th annual Linux Showcase and Conference*, 2000.

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

[27] K. Keahey and T. Freeman, “Contextualization: Providing one-click virtual clusters,” in *4th IEEE International Conference on e-Science*, Indianapolis, IN, December 2008.

[28] R. Schollmeier and G. Schollmeier, “Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns,” in *Second International Conference on Peer-to-Peer Computing Proceedings (P2P 2002)*, September 2002, pp. 112–119.

[29] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Looking up data in P2P systems,” *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, February 2003.

[30] A. Oram, *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*, A. Oram, Ed. O’Reilly, March 2001.

[31] 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.

[32] B. Nitzberg, J. M. Schopf, and J. P. Jones, *PBS Pro: Grid computing and scheduling attributes*. Norwell, MA, USA: Kluwer Academic Publishers, 2004, ch. 13, pp. 183–190.

[33] I. Lumb and C. Smith, *Scheduling attributes and platform LSF*. Norwell, MA, USA: Kluwer Academic Publishers, 2004, ch. 12, pp. 171–182.

[34] Amazon Web Services, “Amazon elastic compute cloud (amazon EC2).” [Online]. Available: http://aws.amazon.com/ec2/

[35] Engage VO. [Online]. Available: https://twiki.grid.iu.edu/bin/view/Engagement/WebHome

[36] The STAR Experiment. [Online]. Available: http://drupal.star.bnl.gov/STAR/

[37] Duke Systems, “Shirako home page.” [Online]. Available: http://nicl.cod.cs.duke.edu/cereus/shirako.html

[38] W. Emeneker, “Dynamic virtual clustering,” Master’s thesis, Arizona State University, April 2007.