Virtual Organization Clusters: Self-Provisioned Clouds on the Grid

Michael A. Murphy[a,1,2,**], Sebastien Goasguen[a,2,*]

[a] *School of Computing, Clemson University, 120 McAdams Hall, Clemson, SC 29634-0974 USA*

Abstract

Virtual Organization Clusters (VOCs) are a novel mechanism for overlaying dedicated private cluster systems on existing grid infrastructures. VOCs provide customized, homogeneous execution environments on a per-Virtual Organization basis, without the cost of physical cluster construction or the overhead of per-job containers. Administrative access and overlay network capabilities are granted to Virtual Organizations (VOs) that choose to implement VOC technology, while the system remains completely transparent to end users and non-participating VOs. Unlike existing systems that require explicit leases, VOCs are autonomically self-provisioned and self-managed according to configurable usage policies.

The work presented here contains two parts: a technology-agnostic formal model that describes the properties of VOCs and a prototype implementation of a physical cluster with hosted VOCs, based on the Kernel-based Virtual Machine (KVM) hypervisor. Test results demonstrate the feasibility of VOCs for use with high-throughput grid computing jobs. With the addition of a "watchdog" daemon for monitoring scheduler queues and adjusting VOC size, the results also demonstrate that cloud computing environments can be autonomically self-provisioned in response to changing workload conditions.

Keywords: grid computing, cloud computing, self-provisioning, autonomic resource management

*Corresponding author

**Principal corresponding author

Email addresses: mamurph@cs.clemson.edu (Michael A. Murphy), sebgoa@clemson.edu (Sebastien Goasguen)

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

[2] The authors would like to thank Linton Abraham, Michael Fenn, and Brandon Kagey for their assistance in system administration and data collection.

1. Introduction

Grid systems, such as the Open Science Grid (OSG) [33], enable different entities to share computational resources using a flexible and secure framework. These resources enable users to run computational jobs that exceed the capabilities of the systems available at any single location. To mitigate compatibility issues that result from resource heterogeneity, computational grids could be designed as Service Oriented Architectures, in which grid-enabled applications do not interact with low-level systems and resources directly. Instead, these applications communicate with abstract service libraries, which are built on top of standard network communications protocols such as TCP and IP. The services layer acts as a high-level operating system for the underlying physical resources, providing both abstraction and arbitration functionality. Since co-scheduling physical resources across sites is a complex task, such grid services are well-adapted for High Throughput Computing (HTC) applications, which tend to be compute-bound and do not require completion deadlines. [15]

Virtualization of grid systems has been proposed as a mechanism for providing custom environments to users on grid systems that expose low-level computational resources as opposed to services [14], thereby enabling private clouds to be constructed using grid computing resources as a hosting utility [5]. Virtualized grid systems to date have taken the approach that new middleware should be developed for the leasing of physical resources on which to run virtual containers. The most widely published systems – Virtual Workspaces [25], In-VIGO [1], and Shirako [24] – all require the addition of system-specific middleware at both the execution and submission endpoints. In some cases, these systems require outright replacement of entire middleware stacks. With such requirements imposed on both the hosting site and the user, these lease-oriented systems are not transparent and cannot be easily deployed in a non-disruptive fashion. In contrast, a completely autonomic system would adapt to changing workloads and resource availability, without requiring manual intervention by either the user or system administrators [45].

The system presented here is a clustering overlay for individual grid resources, which permits Virtual Organizations of federated grid users to create custom computational clouds with private scheduling and resource control policies. This system is designed according to a specification known as the Virtual Organization Cluster Model, which stipulates the high-level properties and constraints of Virtual Organization Clusters. This model is technology-agnostic, permitting the use of different virtualization technologies, networking systems, and computational grids. A prototype cluster designed according to the model demonstrates that a viable implementation of the VOC specification is possible.

The motivation for the Virtual Organization Cluster Model and Virtual Organization Clusters (VOCs) built according to the model is to create a virtual cluster environment that is homogeneous across sites, autonomically self-provisioned, 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. Transparency

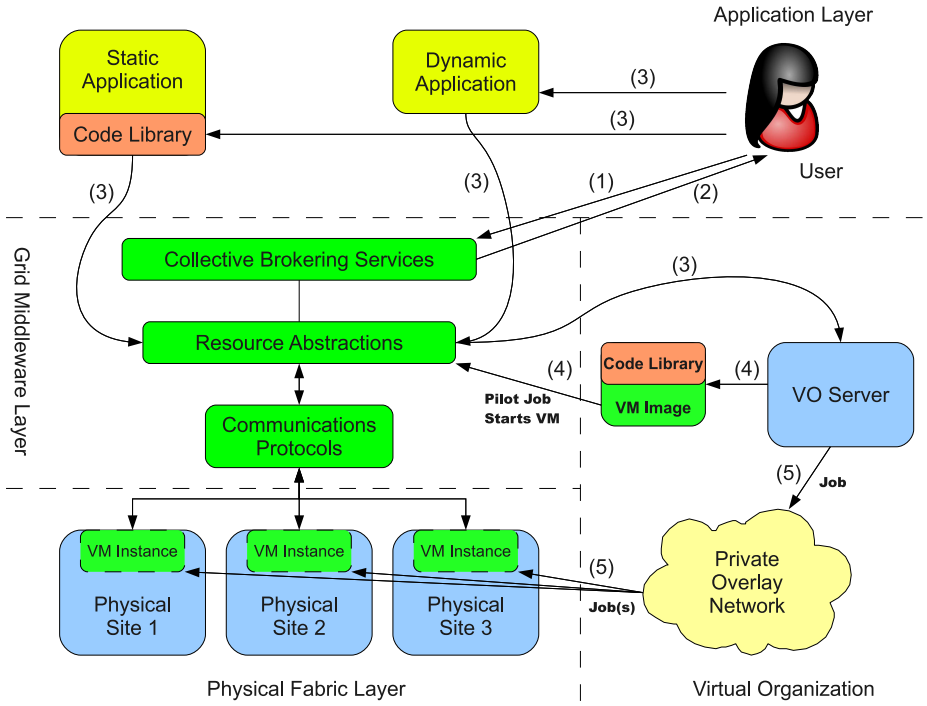

Figure 1: A use case for Virtual Organization Clusters, in which pilot jobs are used to reserve physical resources to host virtual machines. User jobs are then privately scheduled to run in the virtual machines, using an overlay network to create a virtual cluster. It should be noted that the user submits her job(s) to a resource abstraction of the VO Server, which autonomically handles the creation and destruction of virtual environments. The user is unaware of the details of this environment management.

is achieved by designing the system so that no submission endpoint middleware is needed, and VOC technologies can be added to execution endpoints with minimal disruption of existing middleware. Moreover, grid sites that choose to provide VOCs may provide them transparently, so that neither the user nor the VO is aware that execution is actually occurring on a VM. Conversely, a site and a VO may choose to permit the VO to supply the VOC image, thereby allowing the VO to have optional administrative access for software stack and policy customization.

The remainder of this paper is organized as follows. Related work is presented in section 2, after which the high-level Virtual Organization Cluster Model is described in section 3. Section 4 describes a prototype implementation of a Virtual Organization Cluster and associated physical testbed, with test results following in section 5. Finally, section 6 presents conclusions and describes future work.

2. Related Work

Virtualization at the Operating System (OS) level, originally developed for IBM mainframe systems in the late 1960s, permits an operating system installed directly on the computational metal, known as the "host" system, to run a second "guest" operating system or "appliance" inside a virtual container. Virtualization systems allow architecture-compatible software systems to be decoupled from the underlying physical hardware implementation, thereby allowing computation to be location independent. [13] Virtualization of grid systems, first proposed in [14], offers substantial benefits to grid users and system administrators. Users of virtualized systems can be granted administrative access rights to their virtual machines, thereby allowing end-user customization of the software environment to support current and legacy applications. Since the hardware administrators retain control of the Virtual Machine Monitors (VMMs) or hypervisors, coarse-grained resource controls can be implemented on a per-VM basis, allowing hardware resources to be shared among different VMs. [14] Higher-level system components may also be virtualized; examples include dynamic service overlays [2], cloud storage frameworks [22], and virtual networking systems such as ViNe [44], VNET [42], VDE [8], and IPOP[20]. Virtualization at the application layer has been realized in systems such as In-VIGO [1].

Globus Virtual Workspaces, implemented as part of the Globus Nimbus toolkit, provide a lease-oriented mechanism for sharing resources on grid systems with fine-grained control over resource allocation. A Virtual Workspace is allocated from a description of the hardware and software requirements of an application, allowing the workspace to be instantiated and deployed on a per-application basis. Following the construction of the environment, the Workspace must be explicitly deployed on a host site, after which it can be directly accessed to run computational jobs. [25] 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 [40]. By leasing multiple resources simultaneously, Virtual Workspaces may be aggregated into clusters [16].

Selection and final customization of VM appliances to match the requirements of individual jobs are accomplished via "contextualization," a process by which job requirements are used to make minor configuration changes, such as network and DNS settings, to an existing appliance [4, 27]. Contextualization is done once per VM invocation, just prior to boot time, and involves injecting configuration data into the VM image prior to initialization of the instance. A centralized "context broker" provides an XML description of the configuration parameters to be set on a per-appliance basis. [26] This contextualization process may occur during workspace scheduling [18].

For performance reasons, workspace jobs should have sufficient run length so as to make the overhead of leasing and starting a workspace relatively small. Given the non-trivial data copy overheads involved with constructing a workspace, it has been proposed that overhead be treated as a property of the job execu-

tion site, instead of charging the overhead to the user as part of the resource allocation [39]. Providing a combination of pre-arranged and best-effort leases has been shown to have better aggregate performance than a traditional best-effort scheduler without task preemption, while approaching the performance of a traditional best-effort scheduler with task preemption [41][40].

Another lease-oriented mechanism for grid system virtualization is Shirako [24], which allows resource providers and consumers to negotiate resource allocations through automated brokers designed around a self-recharging currency model [23]. Back-end cluster provisioning is provided by Cluster-On-Demand (COD) [7], an automated system that performs rapid reinstallation of physical machines or Xen virtual machines [21]. An application of Shirako, called the Grid Resource Oversight Coordinator (GROC), permits physical resources to be leased by individual Virtual Organizations for the purpose of deploying completely virtualized grids, which are accessed by end users through existing Globus middleware. Each physical grid site hosts a self-contained virtual cluster belonging to the same VO, without spanning of individual clusters across grid sites. [34]

Autonomic virtualization overlay systems have been devised for the purpose of extending local clusters to utilize idle resources available on other clusters within the same local area. VioCluster is based on the concept of dividing each cluster into a physical and a virtual domain, where the virtual domain may be borrowed and administered by another group within the same entity. Virtual domains are transparently and autonomically resized 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. [37] 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 [36].

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" [10]. 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. [10]

Another approach to providing virtualization services at various levels in a grid system is to modify the scheduling system, the application itself, or both. The In-VIGO system dynamically utilizes virtual machines, virtual networks,

and virtual interfaces to support specially modified applications [1]. By modifying parallel applications to become self-scheduling at the user level, adaptive workload balancing can be used to improve parallel application performance without the introduction of system-level virtualization [28]. Conversely, the CARE Resource Broker enables applications to be scheduled in dynamically allocated virtual machine containers through a custom scheduling interface that is resource-aware [38].

Virtual Organization Clusters [31] differ from lease-oriented systems such as Globus Nimbus and Shirako, in that the virtual clusters are created autonomically and are dynamically provisioned in response to increasing workloads for the associated VO [32]. VOCs remain transparent to end users and to non-participating entities, while enabling participating VOs to use overlay networks, such as IPOP [20], to create virtual environments that span multiple grid sites. Unlike Campus Area Grids and VioCluster environments, however, these disparate grid sites are 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 [12]. Unlike systems that require application modification or dedicated submission portals, VOCs can interoperate with existing schedulers and job management middleware.

3. Virtual Organization Cluster Model

The Virtual Organization Cluster Model describes the high-level properties and constraints of Virtual Organization Clusters. VOCs may be implemented using a wide variety of technologies and specific techniques. The VOC Model does not require any particular choice of virtualization system, overlay network, or middleware layer, nor does the model constrain implementations to a single grid or fixed set of Virtual Organizations. Instead, the model presents a design philosophy and formal architectural specification for VOCs. *Any* virtualized environment system whose design is compatible with the VOC Model can be called a Virtual Organization Cluster.

In order to give a precise definition of a Virtual Organization Cluster (section 3.3), it is first necessary to describe the separation of administrative domains (section 3.1) and define a few key terms (section 3.2).

3.1. Separation of Administrative Domains

The Virtual Organization Cluster Model specifies the high-level properties of systems that support the assignment of computational jobs to virtual clusters owned by single VOs. Central to this model is a fundamental division of responsibility between the administration – and associated policies – of the physical computing resources and the virtual machine(s) implementing each VOC, permitting a division of labor between physical and virtual system administrators [19]. For clarity, the responsibilities and policy decisions of the hardware owners

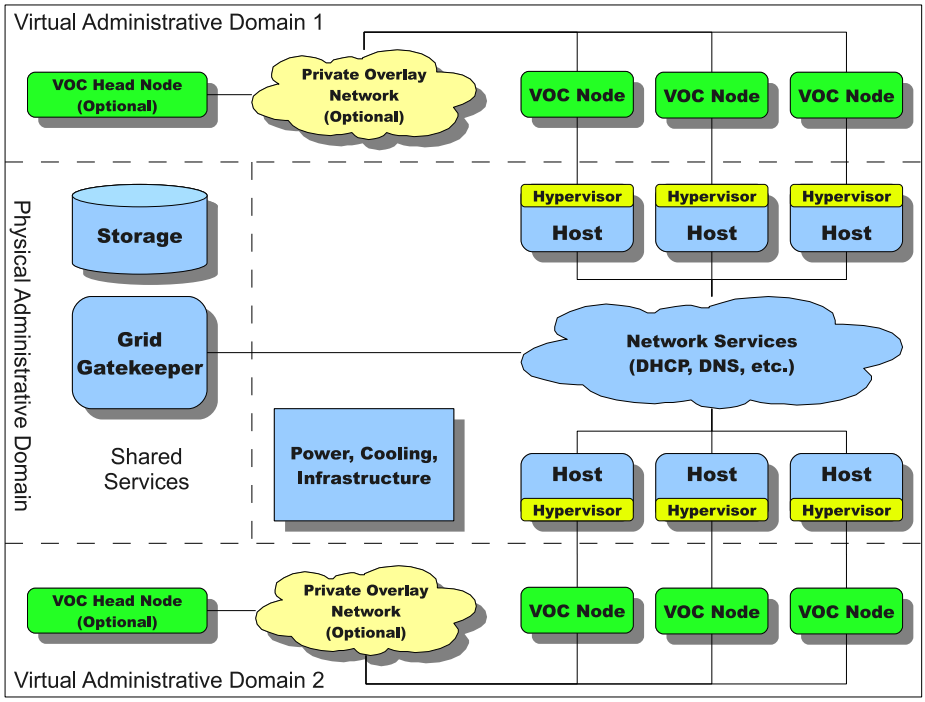

Figure 2: Implementations of Virtual Organization Clusters divide the grid into administrative spaces: each grid site has Physical Administrative Domain (PAD) that includes the hardware and software services needed to execute virtual machines. Each VOC is an isolated Virtual Administrative Domain, into which scientific software applications, computational libraries, and supporting programs are installed. Different administrative domains in this model may have different administrative staff and independent policies.

are said to belong to the Physical Administrative Domain (PAD). Responsibilities and policy decisions delegated to the VOC owners are part of the Virtual Administrative Domain (VAD) of the associated VOC. Each physical cluster has exactly one PAD and zero or more associated VADs. Figure 2 illustrates this division of responsibility.

Physical Administrative Domain

VOCs are executed atop physical computing fabric made available by different organizations over a standard grid computing platform such as the Open Science Grid [33]. Each of physical site on the grid is an isolated Physical Administrative Domain (PAD), managed independently from the VOCs it hosts. The PAD contains the physical computer hardware (see figure 2), which comprises the host computers themselves, the physical network interconnecting those hosts, local and distributed storage for virtual machine images, power distribution systems, cooling, and all other infrastructure required to construct a cluster from hardware. Also within this domain are the host operating systems and central physical-level management systems and servers, as well as the system policies applied to both the hardware and software. Fundamentally, the hardware cluster provides the hypervisors needed to host the VOC system images as guests.

Virtual Administrative Domain

Each Virtual Administrative Domain (VAD) consists of a set of homogeneous virtual machine instances spawned from a common image and dedicated to a single Virtual Organization (VO). A VAD may interact with its underlying Physical Administrative Domain in one of three ways. In the simplest case, a single virtual machine image is used to spawn all the VOC nodes in the VAD, each of which connects to a shared scheduler provided by the PAD. This case permits a physical site to use a VOC to sandbox all jobs from a single VO in a manner transparent to the VO itself. Alternatively, the VO may supply a second VM image to be used as a virtual head node and dedicated grid gatekeeper on each physical site. Finally, the VO may provide a dedicated head node and grid gatekeeper on a remote site, utilizing a private overlay network (depicted in figure 2) to schedule jobs on the VOC nodes.

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. A proposed solution to the image format problem is presented in the Open Virtualization Format specification [9].

- **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 or immediately prior to boot.

- **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.

3.2. Terminology

In order to give a precise definition to the concept of Virtual Organization Clusters, and to provide a framework for evaluating the behavior and performance of VOCs, some preliminary terminology is necessary. These terms are defined in abstract grid computing contexts to avoid circular definitions.

Definition 1. A grid technology is *transparent* to an entity if the entity can utilize the technology through existing grid middleware services that are installed as part of a standard grid interconnection system, without the addition of any extra middleware. Furthermore, the grid technology must also be transparent according to the definition presented in [11]: access to the technology must be accomplished via services and not by direct connection to specific systems.

Transparency is a key issue in the design and prototype implementation of the VOC Model. Most virtualized grid systems require the addition of specific middleware, and perhaps replacement of existing middleware, at both the execution endpoint and the submission endpoint in order to utilize virtualization capabilities through the acquisition of leases [1, 24, 25]. As a result, implementing these systems requires a substantial investment of resources for both cluster administrators and cluster users. Moreover, requiring users to access leased environments directly violates the original principle of transparency required by Enslow [11]. By extending the concept of transparency in definition 1 to include middleware, systems that claim this property must be usable by an end-user equipped only with the standard grid access tools in existence and installed before the addition of the new technology. Ideally, the user would also be unaware of the existence of the new system. If this property can be satisfied, then administrative action is only required at the computational sites where the system is to be deployed, minimizing disruption to the existing grid infrastructure.

Definition 2. A job is *compatible* with a cluster if the job is executable using the hardware, operating system, and software libraries installed on the cluster. The cluster in this case may be physical or virtual.

For a given cluster system, the *compatible fraction* of jobs belonging to an entity is the ratio of compatible jobs belonging to that entity to the total number of jobs on the grid belonging to that entity. An entity is *compatible* with a

cluster if the compatible fraction of jobs belonging to that entity is non-zero on the cluster.

Compatibility refers to the ability of a cluster system, whether physical or virtualized, to run a job with the available hardware (or virtual hardware) and installed software. Several different measures of compatibility may be considered when evaluating virtualized clusters, including the breadth of compatible VOs and the total proportion of compatible jobs across all VOs. Implementation of a new cluster technology might enable the cluster to support a larger number of different VOs; however, such an implementation might simultaneously reduce the previously compatible fraction of jobs for VOs already supported, reducing the total proportion of grid jobs that are compatible with the cluster. A trade-off may arise between these two measures for certain design decisions.

It is important to note that compatibility does not necessarily imply that a cluster is willing to run jobs from all compatible VOs. Local policies may restrict cluster usage to a specific subset of VOs, even though the cluster is compatible with a larger set of VOs. This distinction between capability and policy is formalized by the following definition:

Definition 3. An entity is *authorized* on a specific cluster if local cluster policies permit the entity to run jobs on the cluster. An *unauthorized* entity is denied use of a cluster system only by policy and not by an insufficiency of mechanism.

The main purpose of definition 3 is to separate mechanism from policy when defining and evaluating technologies such as VOCs. It would be incorrect to treat a VO as incompatible if VO jobs were rejected by the cluster simply because local policy did not provide execution services to that VO. Technical compatibility, or the ability to run jobs within particular environments, is a separate issue from that of accounting for actual resource usage or determining the set of VOs to be given access to a particular resource.

Definition 4. An entity is termed to be *participating* in a grid-enabled technology if the entity chooses to utilize the specific capabilities of the technology, including, but not limited to, the use of specific middleware or specific configuration settings. Entities that choose not to deploy the technology under discourse are termed *non-participating*.

In order to facilitate transparency and enable partial deployment of new grid-enabled technologies such as VOCs, it is necessary to accommodate the different schedules upon which different entities may choose to deploy the technology. Moreover, some entities may choose not to deploy the technology due to technical or policy constraints. Achieving interoperability of VOCs with existing grid systems requires that the grid remain able to support both participating and non-participating entities.

3.3. A Formal Definition of Virtual Organization Clusters

Utilizing the previous definitions, a Virtual Organization Cluster may be formally defined as a set of *homogeneous* computational resources that:

- Consists entirely of virtual machines that are scheduled and hosted by commodity physical resources on the grid;
- Autonomically changes size in response to changing workload demands;
- Is owned by, or dedicated to, exactly one Virtual Organization;
- Is transparent to end users;
- Is transparent to non-participating virtual organizations;
- Provides a Virtual Administrative Domain to participating VOs; and
- Optionally permits participating VOs to utilize a private overlay network to span resources across physical sites, receive jobs from a remote scheduler, and access private resources.

A VOC is effectively a set of *compute nodes* in the traditional cluster computing sense. The scheduling of jobs on these compute nodes may be performed by a shared local scheduler on the same physical site as the VOC, by a dedicated virtual head node on the same physical site as the VOC, or by a central scheduler accessible via an overlay network. The flexibility afforded by this definition permits a wide range of VOC deployments, from a transparent system provided on behalf of a Virtual Organization without any involvement of the VO itself, to a fully overlaid environment with a private scheduling and policy domain directly manageable by the VO.

Figure 3 depicts two VOCs dedicated to two separate VOs. In this case, VO_1 is non-participating and thus chooses not to deploy VOCs or any related middleware. Nevertheless, Site 2 provides VO_1 with a transparent VOC, so that all VO_1 user jobs on Site 2 run within virtual machines. Simultaneously, VO_2 chooses to utilize a VOC with a private overlay network. End user jobs from users affiliated with VO_2 are submitted directly to the private head node owned by VO_2, and VOC nodes are autonomically started on both sites in response to the size of the scheduler queue. VO_2 user jobs are then privately scheduled and routed by means of an overlay network.

4. Prototype Implementation

In order to evaluate the viability of the Virtual Organization Cluster Model, a small prototype system was constructed (figure 4). The physical hardware consisted of 16 dual-core compute nodes, a private network using switched gigabit Ethernet, 9 dual-drive storage nodes using the PVFS [6] distributed filesystem, and a private head node. Slackware Linux 12 was installed on these systems initially, but this installation was later replaced CentOS 5.2. Up to 32 single-core, or 16 dual-core, virtual machines could be started to provide VOC nodes, using the Kernel-based Virtual Machine (KVM) hypervisor for virtualization [35]. A shared head node with a Globus [17] interface to the Open Science Grid [33] was created as a virtual machine. This shared head node provided a

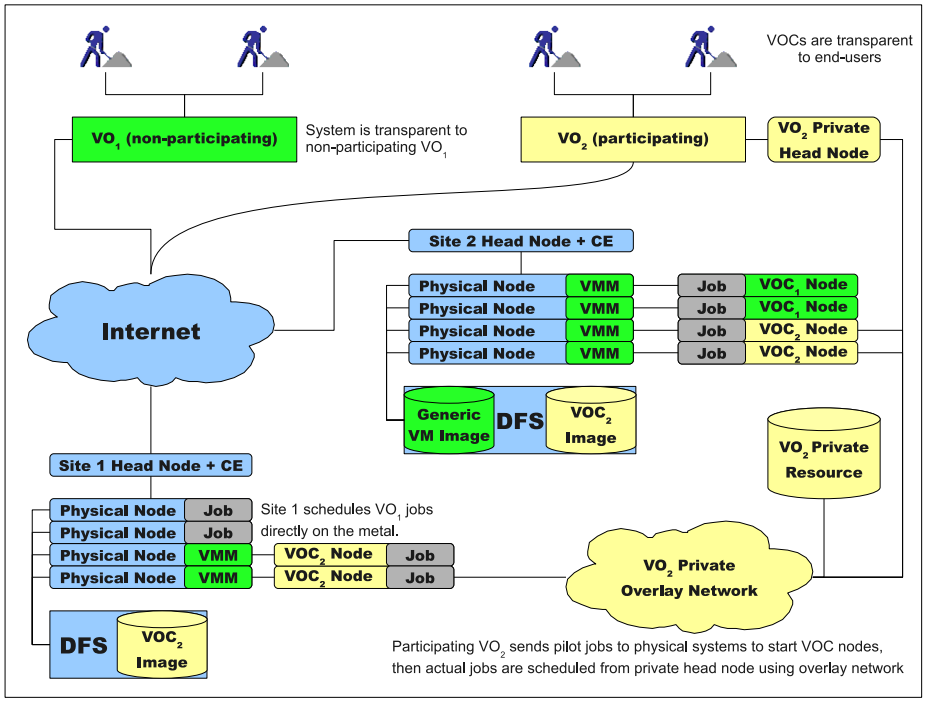

Figure 3: Virtual Organization Clusters on a grid system. In this example, Site 2 transparently provides a VOC to VO_1, which does not deploy its own VOCs. VO_2 chooses to deploy VOCs and has a customized environment with private scheduling and resource control.

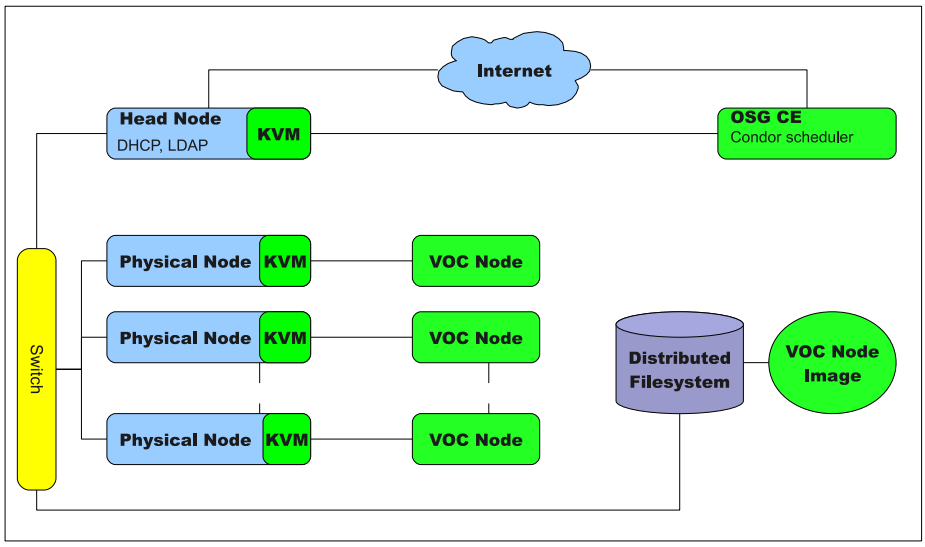

Figure 4: Initial prototype test system. The shared grid gatekeeper (Open Science Grid Compute Element, or OSG CE) and Condor scheduler were located in a virtual machine for convenience. These components conceptually belong to the Physical Administrative Domain.

Condor [43] scheduler pool, which was originally used to receive end-user jobs for a transparent VOC application. A watchdog daemon monitored the Condor queue, starting and stopping virtual machines by means of a deterministic local placement algorithm. The prototype was later extended to test VOCs from participating VOs by adding support for the Internet Protocol Over Peer-to-peer (IPOP) [20] overlay network. With this addition, a separate Condor pool outside the prototype system was used for scheduling end-user jobs, while the original Condor scheduler on the shared head node was used to receive pilot jobs submitted externally.

5. Test Results

Tests were conducted on the prototype implementation to evaluate the viability of Virtual Organization Clusters and the associated autonomic self-provisioning system. Benchmark testing was used to measure the overheads introduced by the addition of virtualization technology (section 5.1). The behavior of the autonomic self-provisioning system was then observed through the use of synthetic workloads (section 5.2). Finally, the fault tolerance of the system was analyzed (section 5.3).

5.1. Virtual Cluster Performance

As an initial measure of virtual cluster performance, boot times were measured during the initial Slackware 12 installation of the prototype system. Boot

Table 1: Boot Times (seconds) for both the physical systems (Slackware 12) and virtual machines (CentOS 5.1). The physical system boot process involved several timeouts, including delays for the Power-On Self Test, disk controller management interface, Preboot eXecution Environment (PXE) for network booting, and a bootloader management menu.

	Physical Node			VM
Statistic	PXE Timeout	Total Boot	Actual Boot	VM Boot
Minimum	105	160	43	61.2
Median	106	160.5	44	65.4
Maximum	107	163	46	70.2
Average	106.4	160.9	44.5	65.5
Std Deviation	0.63	1.03	1.09	2.54

times were measured manually for the physical boot procedure, while a simple boot timing server was constructed to measure VM booting time, which received boot starting notifications from the physical nodes and boot complete notifications from the associated virtual nodes. Timing of the boot process was performed at the server side, avoiding any clock skew potentially present between physical and virtual nodes, but possibly adding variable network latency. Boot times for the physical nodes were subject to even greater variation, as these were measured manually.

Results of the boot time tests are summarized in table 1. For the physical system, the boot process was divided into three phases: a PXE timeout, a GRUB timeout, and the actual kernel boot procedure. While total boot times ranged from 160 to 163 seconds, 105 to 107 seconds of that time were utilized by the PXE timeout, and 10 seconds were attributed to the GRUB timeout. Thus, the actual kernel boot time ranged from 43 to 46 seconds. In contrast, the virtual compute nodes required 61.2 to 70.2 seconds to boot. These virtual machines were configured with a different operating system (CentOS 5.1) and started approximately 10 additional processes at boot time, compared to the physical systems. As a result, not all the boot time discrepancy could be attributed to virtualization overhead. Nonetheless, the overhead was small enough that booting the VOC did not require an inordinate amount of time and was considered acceptable.

To measure the operational performance of the virtual cluster system, the High Performance Computing Challenge (HPCC) benchmark suite [29] was used, which included the High-Performance Linpack (HPL) benchmark, Random Access benchmark, Fast-Fourier Transform (FFTE) benchmark, Embarrassingly Parallel (EP-STREAM and EP-DGEMM) benchmarks, and Random-Ring network tests of bandwidth and latency. These tests were conducted after both the host and guest system software was changed to CentOS 5.2 and were repeated for three cases: a single virtual machine compared to a single host (1x1 process grid), 28 single-core virtual machines compared to 14 dual-core hosts (7x4 process grid), and 14 dual-core virtual machines compared to 14 dual-core hosts (7x4 process grid). The overheads of virtualization, as a percentage of

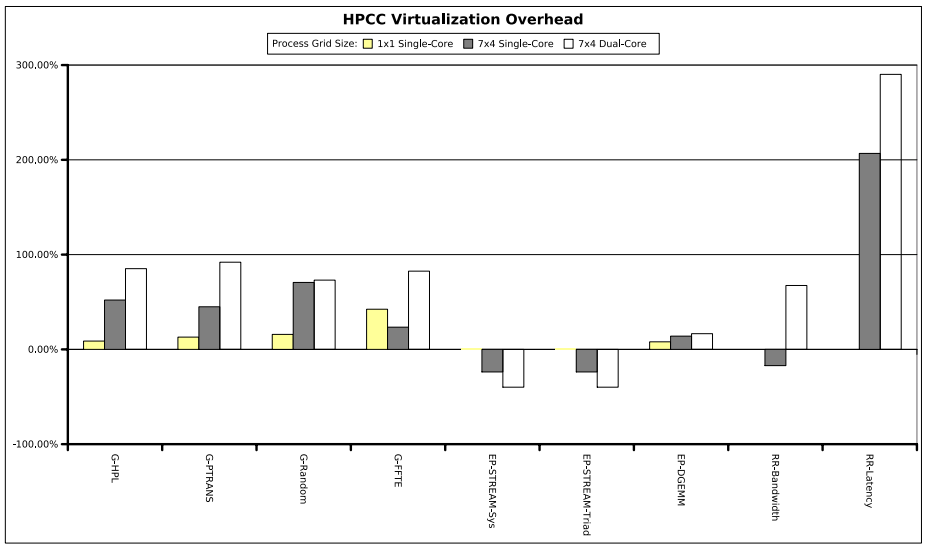

Figure 5: High Performance Computing Challenge (HPCC) benchmark overheads normalized relative to the performance of the host system. Three sets of tests were conducted: a single-core virtual machine compared to a single HPCC process (1x1 process grid), 28 single-core VMs compared to 14 physical hosts (7x4 process grid), and 14 dual-core VMs compared to 14 physical hosts (7x4 process grid).

physical performance, were computed for these data sets.

As illustrated in table 2 and figure 5, the HPL overhead for a compute-bound 1x1 process grid was observed to be 8.77%. For embarrassingly parallel jobs, the computational overhead (EP-DGEMM) was measured to be 7.98%. This type of application would be representative of a typical high-throughput grid job, or a "bag-of-tasks" application [3]. In contrast, MPI jobs that utilized inter-node communications (HPL, Random Access, Fast-Fourier Transform, and RandomRing) incurred substantial performance overheads on VMs for a 7x4 process grid. With HPL, these overheads were observed to be 52% for the single-core test and 85% for the dual-core test. Network latency was suspected for this observed overhead, as latency has been implicated as a cause of performance reduction in prior studies involving MPI [30, 29], and RandomRing latencies were over 100% higher than the metal for all tests.

Improved sustainable memory transfers (EP-STREAM) were observed in virtualized cases, compared to the physical systems, in the 7x4 process grid cases. An EP-STREAM overhead of -23.82% was computed for the single-core case, while the overhead improved to -39.89% for a dual-core VM. These improvements were attributed to the emulated disk and memory subsystems present in KVM. Since the EP-STREAM tests show exceptionally high locality of spatial reference with low locality of temporal reference and high system memory usage [29], paging to disk was likely in the virtual memory subsystem. The emulated disk device present in KVM was physically a region of physical

Table 2: HPCC Performance

Process Grid	1x1		7x4		
	Physical	VOC	Physical	Single-Core	Dual-Core
Problem Size	10300	10300	58600	58600	58600
G-HPL (GFLOPS)	7.913	7.218	169.807	81.401	25.178
G-PTRANS (GB/s)	0.729	0.635	0.867	0.447	0.069
G-Random Access (GUP/s)	0.002	0.002	0.014	0.004	0.004
G-FFTE (GFLOPS)	0.799	0.461	2.287	1.751	0.399
EP-STREAM Sys (GB/s)	3.866	3.808	59.046	73.110	82.599
EP-STREAM Triad (GB/s)	3.866	3.808	1.845	2.285	2.581
EP-DGEMM (GFLOPS)	8.348	7.682	8.271	7.114	6.901
RandomRing Bandwidth (GB/s)	N/A	N/A	0.023	0.027	0.007
RandomRing Latency (μs)	N/A	N/A	74.444	228.383	290.463

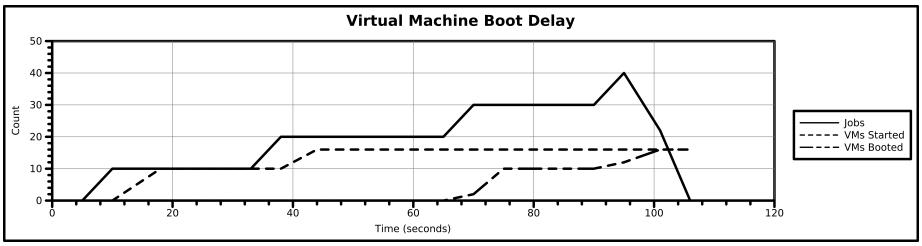

Figure 6: Virtual Machine boot delays relative to job submission time. As the watchdog daemon observed the Condor queue size increasing, VMs were started. These VMs were recorded as booted once they joined the Condor pool.

memory. Page faults in the guest could have been handled more efficiently than host page faults due to this emulation, resulting in a higher sustainable memory transfer rate in the guest.

5.2. Autonomic Self-Provisioning

5.2.1. Dynamic VM Boot Performance

A second set of tests were performed to measure the startup times (including the boot delay) for the VMs comprising the VOC. A synthetic workload comprised of groups of 10 one-second jobs was submitted directly to the local Condor pool, 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 6, 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.

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 dynamic processes that must occur during VM boot. These processes include the allocation of memory to the VM, initialization of the VM kernel, acquisition of a DHCP lease by the VM, and the starting of run-time services. Based on the test results, a conservative upper bound of 60 seconds was attributed to the VM boot process.

5.2.2. Autonomic Expansion and Shrinkage

A simple greedy algorithm was used as the autonomic Virtual Organization Cluster sizing policy. This algorithm, run once each watchdog interval, started new virtual machines whenever the size of the Condor queue exceeded the size of the VOC, provided that additional physical resources were available. As jobs completed and the size of the scheduler queue decreased, VOC nodes were terminated. As illustrated in figure 7, this algorithm proved to be over-responsive to the batches of short (10-second) jobs submitted directly to the scheduler. The VOC was rapidly expanded to use all 16 available processor cores at the first watchdog interval. A delay of approximately 60 seconds was observed while the VOC nodes booted, after which the short user jobs quickly ran to completion. Even though additional jobs arrived in the queue while the VOC nodes were booting, all jobs from the first two batches had completed within 140 seconds. At this time, the size of the VOC was shrunk to zero, causing the virtual machines to vacate the physical systems completely. When another batch of jobs arrived at 180 seconds into the test, all 16 virtual machines had to be restarted, resulting in another boot delay.

Submission of jobs through the Open Science Grid to the local Globus interface required several extra seconds for each job. As illustrated in figure 8, this extra latency spread out job arrival in the Condor queue, resulting in a slower start to the initial VOC. Since these jobs were short (10 seconds), the Condor queue size never exceeded 13 jobs, and the number of VOC nodes never reached the maximum size permitted by the physical cluster (16). In addition, the arrival spacing of the jobs resulted in a minimal size of 2 nodes during the VOC shrinkage period that occurred between batches. As a result, the VOC did not completely vacate the physical cluster, although the simple greedy policy was still over-responsive in terminating VMs.

To simulate the behavior of the watchdog daemon when a participating Virtual Organization utilized a non-transparent VOC, the Internet Protocol Over Peer-to-peer (IPOP) overlay network [20] was added to the system. A second Condor queue was run on an external system, along with a second watchdog-type daemon that generated pilot jobs for submission to the physical Condor

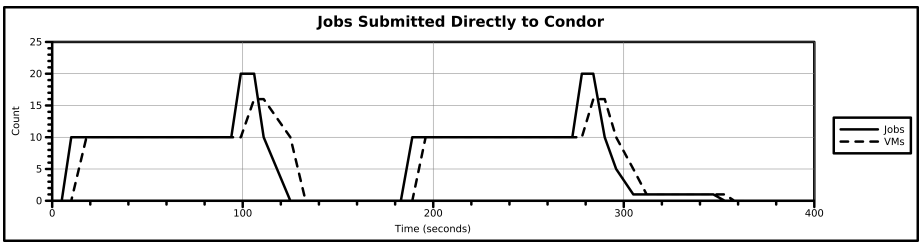

Figure 7: Batches of short jobs submitted directly to the Condor queue. As jobs completed, the VOC nodes were quickly terminated, causing the VOC to vacate the cluster completely. When additional jobs arrived in a later batch, the entire VOC had to be re-started.

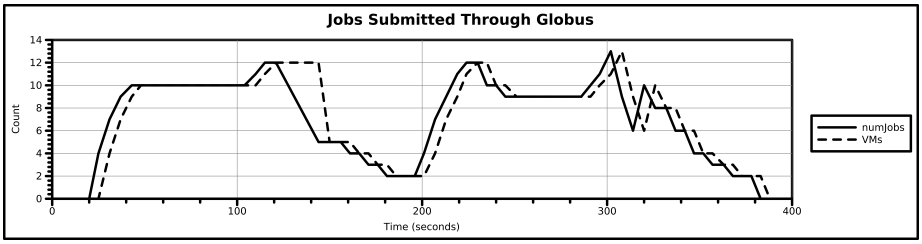

Figure 8: Jobs submitted through Globus. The additional delay of sending jobs across the grid and through the gatekeeper spread out job arrivals and prevented the VOC from being terminated completely.

queue via the grid and Globus. These pilot jobs were used to start VOC nodes. Each VOC node joined the external Condor pool, using the IPOP network, after completion of the boot process. Overlay scheduling was then performed to execute the original user jobs on the VOC. As depicted in figure 9, short jobs arriving in the external Condor queue resulted in the periodic submission of pilot jobs to the grid site. These pilot jobs arrived slowly relative to the batch of user jobs, due to the same latency introduced by the grid system and Globus interface. Once the jobs completed on the VOC, the external watchdog process terminated the pilot jobs, resulting in eventual termination of the VMs.

5.2.3. Multiple Virtual Organization Clusters

A simulated synthetic workload was tested with a simple greedy scheduling algorithm, to evaluate the sharing a single physical site among several Virtual Organization Clusters. This workload contained 8 jobs of varying length, divided into three jobs of 200 seconds in length for the Engage VO, three jobs of 400 seconds in length for the Engage VO, and two jobs of 600 seconds in length for the Nanohub VO.

As depicted in figure 10, both VOCs could be started and expanded to a sufficient size so as to provide execution capacity for all eight jobs in parallel, since the physical site had capacity for 16 virtual machines. When jobs completed, the sizes of each VOC could be reduced at the next watchdog interval,

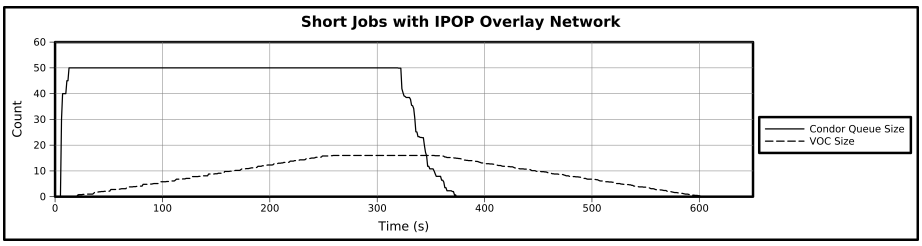

Figure 9: A single batch of short jobs was submitted to an external Condor queue, resulting in pilot jobs submitted through the grid to start VOC nodes. As jobs completed, the pilot jobs were slowly terminated, resulting in termination of the VOC.

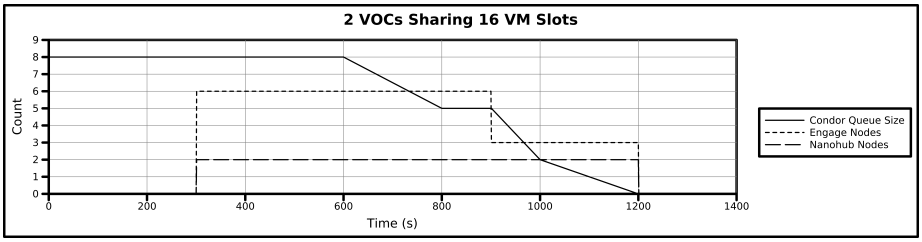

Figure 10: Simulation of two Virtual Organization Clusters sharing 16 Virtual Machine slots on a single physical site. Since there were only 8 total jobs (6 Engage and 2 Nanohub), both VOCs could expand freely.

and eventually all VOC nodes were removed when all jobs finished. The total execution time for the makespan was 1200 seconds.

When the number of virtual machine slots available to run the VOCs was reduced to four (figure 11), the total time for completion of the makespan was increased to 2100 seconds. Both the Engage and Nanohub VOCs were able to claim 2 slots at the first watchdog cycle, permitting the Nanohub VOC to execute both Nanohub jobs in parallel as soon as the corresponding VOC nodes booted. The Engage VOC was able to execute two jobs in parallel until the Nanohub VOC completely vacated the physical cluster, after which the Engage VOC started a third node. Parallelism was increased only briefly, as completion of the third job led to the removal the third node after only 300 seconds. As the remaining two Engage jobs completed, the Engage VOC nodes terminated, leading to a stair-step pattern in the graph.

5.3. Fault Tolerance

One issue was observed in the interaction of the VOC adaptation policy, the virtualization interface, and the Condor scheduler. When repeated batches of jobs were submitted to the virtual cluster (figure 12), the size of the VOC was autonomically expanded and then shrunk. However, the mechanism for terminating the virtual machines during the VOC shrinkage phase involved killing the associated Virtual Machine Monitor process associated with the VM instance.

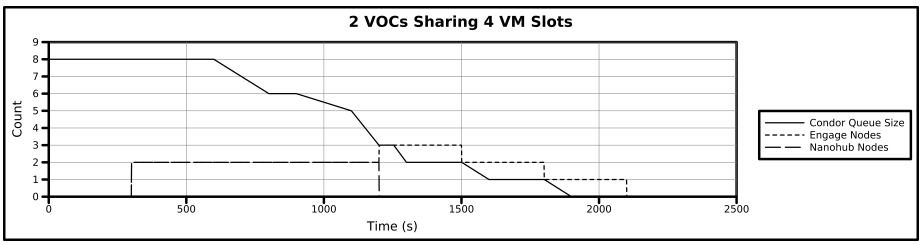

Figure 11: Simulation of two Virtual Organization Clusters sharing 4 Virtual Machine slots. Since there were 8 jobs (6 Engage and 2 Nanohub), neither VOC could be expanded to handle all jobs simultaneously. Until the Nanohub jobs completed, a steady state was observed with 2 VOC nodes per VO. Once the Nanohub jobs completed, the Engage VOC was able to expand to an extra node to complete its jobs somewhat more quickly.

Since the VM instances were read-only clones of a single, shared VM image, no data corruption resulted from this mechanism. However, the VOC nodes were not cleanly removed from the Condor pool during the shrinking process. As a result, the Condor scheduler occasionally matched jobs to non-existent machines, resulting in jobs being placed on hold indefinitely. In the experiment depicted in figure 12, 3 jobs from the final batch were matched to one of 8 VOC nodes that were no longer in operation but were still present in the Condor collector database. Due to this mismatch, the 3 VOC nodes that were still actually in operation at the time were idle. This experiment was terminated manually after 2000 seconds.

Since a Virtual Organization Cluster is hosted across a set of different Physical Administrative Domains, each of which may have different policies, the type of fault observed when a VM is uncleanly terminated may occur with relative frequency on an actual virtualized grid system. A physical grid site may, for example, impose a wall time limit on the pilot jobs used to start VOC nodes. The VOC middleware may not be aware of this limit, which could result in a pilot job – and its associated VM instance – being killed by the site scheduler. Similar faults could occur if a VOC exceeds a different resource limit or if higher-priority jobs on the physical site force immediate termination of the pilot job.

6. Conclusions

Virtual Organization Clusters provide a mechanism by which individual Virtual Organizations may operate and administer virtual cluster environments that support the computing needs of VO-affiliated users. As demonstrated by the prototype system, VOCs can execute High-Throughput Computing (HTC) grid jobs with reasonable overheads under 9%. Although High Performance Computing (HPC) applications that use latency-sensitive MPI library routines experience substantially greater overheads, VOCs still enable execution on physical sites that lack MPI support. This overhead represents a trade-off between performance and environment flexibility: whenever a job is directly compatible

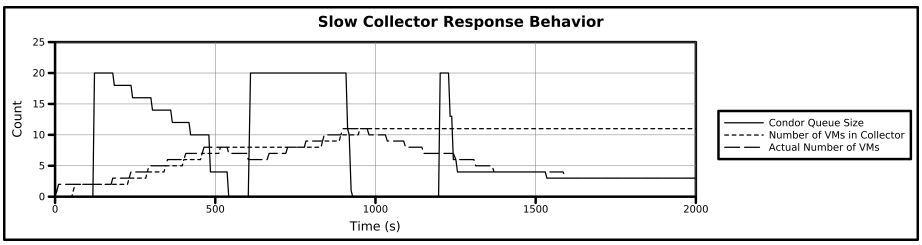

Figure 12: Slow Condor collector response. As VOC nodes were terminated by means of killing the associated hypervisor process, Condor was not notified when VOC nodes were removed from the pool. Thus, some jobs were held indefinitely after matching nodes that were no longer present in the pool.

with the physical system environment at a grid site, the addition of the VOC layer will reduce execution performance for that job. However, when the requirements of a job cannot be met by the environment present on the physical site, the addition of the VOC layer enables job execution when it would be otherwise impossible. The reduced execution performance in this case would be superior to the lack of execution that would otherwise result.

Early performance tests of a prototype dynamic VOC scheduler were encouraging. The VOC nodes were autonomically added and removed from operation in response to changing grid loads for the corresponding VOs, 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 simple greedy algorithm in removing VOCs from operation, different policies could be employed to moderate the responsiveness of the watchdog daemon. Future work on this system will include the development of formal equations to account for VOC overheads, investigation of lease-oriented systems as back-end resources, and improving the fault tolerance of the overlay scheduler.

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, X. Zhu, From virtualized resources to virtual computing grids: the In-VIGO system, Future Generation Computer Systems 21 (6) (2005) 896–909.

[2] O. Ardaiz, L. Navarro, Grid-based dynamic service overlays, Future Generation Computer Systems 24 (8) (2008) 813–823.

[3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, Y. Robert, Centralized versus distributed schedulers for bag-of-tasks applications, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 698–709.

[4] R. Bradshaw, N. Desai, T. Freeman, K. Keahey, A scalable approach to deploying and managing appliances, in: TeraGrid 2007, Madison, WI, 2007.

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering

computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.

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

[7] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, 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, 2003.

[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, 2005.

[9] Distributed Management Task Force, Inc., Open virtualization format specification (February 2009).
URL `http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf`

[10] W. Emeneker, D. Stanzione, Dynamic virtual clustering, in: 2007 IEEE International Conference on Cluster Computing, 2007.

[11] P. Enslow, What is a "distributed" data processing system?, Computer 11 (1) (1978) 13–21.

[12] M. Fenn, M. A. Murphy, S. Goasguen, A study of a KVM-based cluster for grid computing, in: 47th ACM Southeast Conference (ACMSE '09), Clemson, SC, 2009.

[13] R. Figueiredo, P. A. Dinda, J. Fortes, Resource virtualization renaissance, Computer 38 (5) (2005) 28–31.

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

[15] I. Foster, The grid: A new infrastructure for 21st century science, Physics Today 55 (2) (2002) 42–47.

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

[17] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, International Journal of Supercomputing Applications 11 (2) (1997) 115–128.

[18] T. Freeman, K. Keahey, Flying low: Simple leases with Workspace Pilot, in: Euro-Par 2008, Las Palmas de Gran Canaria, Spain, 2008.

[19] T. Freeman, K. Keahey, I. Foster, A. Rana, B. Sotomayor, F. Wuerthwein, Division of labor: Tools for growth and scalability of grids, in: 4th International Conference on Service Oriented Computing (ICSOC 2006), Chicago, IL, 2006.

[20] A. Ganguly, A. Agrawal, P. O. Boykin, 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.

[21] L. Grit, D. Irwin, A. Yumerefendi, 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, 2006.

[22] R. L. Grossman, Y. Gu, M. Sabala, W. Zhang, Compute and storage clouds using wide area high performance networks, Future Generation Computer Systems 25 (2) (2009) 179–183.

[23] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, Self-recharging virtual currency, in: Third Workshop on Economics of Peer-to-Peer Systems (P2PEcon), Philadelphia, PA, 2005.

[24] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, K. Yocum, Sharing network resources with brokered leases, in: USENIX Technical Conference, Boston, MA, 2006.

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

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

[27] K. Keahey, T. Freeman, J. Lauret, D. Olson, Virtual Workspaces for scientific applications, in: Scientific Discovery through Advanced Computing, Boston, MA, 2007.

[28] V. V. Korkhov, J. T. Moscick, V. V. Krzhizhanovskaya, Dynamic workload balancing of parallel applications with user-level scheduling on the grid, Future Generation Computer Systems 25 (1) (2009) 28–34.

[29] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, D. Takahashi, The HPC Challenge (HPCC) benchmark suite, in: Supercomputing '06, 2006.

[30] M. Matsuda, T. Kudoh, Y. Ishikawa, Evaluation of MPI implementations on grid-connected clusters using an emulated WAN environment, in: IEEE International Symposium on Cluster Computing and the Grid (CCGrid03), 2003.

[31] M. A. Murphy, M. Fenn, S. Goasguen, Virtual Organization Clusters, in: 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009), Weimar, Germany, 2009.

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

[33] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, The Open Science Grid: Status and architecture, in: International Conference on Computing in High Energy and Nuclear Physics (CHEP '07), 2007.

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

[35] Red Hat, Kernel-based Virtual Machine.
URL `http://www.linux-kvm.org`

[36] P. Ruth, X. Jiang, D. Xu, S. Goasguen, Virtual distributed environments in a shared infrastructure, Computer 38 (5) (2005) 63–69.

[37] P. Ruth, P. McGachey, D. Xu, VioCluster: Virtualization for dynamic computational domains, in: IEEE International Conference on Cluster Computing, Boston, MA, 2005.

[38] T. S. Somasundaram, B. R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajendar, R. Rajiv, G. Kannan, G. R. Britto, E. Mahendran, B. Madusudhanan, CARE Resource Broker: A framework for scheduling and supporting virtual resource management, Future Generation Computer Systems 26 (3) (2010) 337–347.

[39] B. Sotomayor, K. Keahey, I. Foster, Overhead matters: A model for virtual resource management, in: 2nd International Workshop on Virtualization Technologies in Distributed Computing (VTDC '06), Tampa, FL, 2006.

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

[41] B. Sotomayor, K. Keahey, I. Foster, T. Freeman, Enabling cost-effective resource leases with virtual machines, in: HPDC 2007 Hot Topics, Monterey Bay, CA, 2007.

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

[43] T. Tannenbaum, D. Wright, K. Miller, M. Livny, Condor – a distributed job scheduler, in: T. Sterling (ed.), Beowulf Cluster Computing with Linux, MIT Press, 2001.

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

[45] D. Xu, P. Ruth, J. Rhee, R. Kennell, S. Goasguen, Autonomic adaptation of virtual distributed environments in a multi-domain infrastructure, in: 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06), Paris, France, 2006.