Virtual Cluster Workspaces for Grid Applications

Xuehai Zhang,[1] Katarzyna Keahey,[1,2] Ian Foster,[1,2] Timothy Freeman[1]

[1]*University of Chicago*
[2]*Argonne National Laboratory*

Abstract

Virtual machines provide a promising platform for computational Grids. By their very nature -- virtualization of underlying hardware -- they enable instantiation of a new, independently configured guest environment on a host resource. In addition, they offer the benefits of isolation and fine-grain enforcement and, given the ability to serialize their state and migrate, offer increased flexibility to environments in the Grid. To take advantage of this new technology in Grid computing, we introduced the concept of virtual workspaces which can be configured, managed and deployed in a Grid environment. Since clusters underlie most significant Grid deployments today, in this paper we extended the notion of virtual workspaces to include virtual clusters. We describe changes to Grid architecture and evaluate virtual cluster creation and management, the impact of executing in virtual clusters on applications as well as the possibility of running several virtual clusters on one physical cluster.

1. Introduction

Most significant Grid deployments today, such as Grid3 [1] or Open Science Grid (OSG) [2], rely on clusters that provide a powerful computation platform for their user communities. However, sharing such clusters between different virtual organizations (VOs) [3] is not always easy. Problems arise, for example, when VOs have requirements for execution environment configuration that are not compatible with the cluster's installed libraries and toolkits and are potentially also incompatible across VOs. While this incompatibility can be remedied by partitioning a cluster and automatically installing a certain set of libraries and toolkits, as in [4], other sharing problems such as isolation, controlled sharing, and fine-grained usage enforcement persist.

The recent resurgence of interest in virtual machines (VMs) [5] resulted in the development of cost-effective and promising solutions such as VMware [6] and Xen [7]. Since VMs offer the ability to instantiate a new, independently configured guest environment on a host resource, and since they also provide outstanding isolation and enforcement properties, combining such virtual machines with Grid technology (as suggested in [8, 9]) may provide an answer to many of the problems in Grids today. In addition, the ability to serialize the state of a VM and migrate it opens new opportunities for better load balancing and improved reliability that are not possible with traditional resources.

To study these advantages in the context of Grid user communities, in this paper we extend our earlier effort in combining Grid and VM technology [10] and explore the application of the virtual machine technology to clusters. We define a *virtual cluster* to be a set of virtual machines configured to behave as a cluster and intended to be scheduled on a physical resource at the same time. Such a virtual cluster can be configured with software required by a specific Grid community; for example, a virtual Grid3 cluster is a virtual cluster configured to operate as a cluster within the Grid3. We adopted this example as a test case because it is nontrivial and

provides access to interesting application workloads. Moreover, success with this case will provide a convincing demonstration to a major application community of the feasibility of virtual cluster technology.

Specifically, in this paper we extend the definition of a virtual workspace [10] to encompass the notion of a cluster. We describe extensions needed for workspace definition, architecture extensions, and changes to Grid services supporting workspace definition and deployment. In this context, we describe how such technology can be used to build Grid3 clusters. We use the BLAST application [11] from the Grid3 GADU project [12] to evaluate the impact of running in a virtual cluster on Grid3 workloads. To gain further insight into the trade-offs associated with the use of virtual clusters, we compare the cost of using a virtual cluster with the cost of deployment and management of a virtual cluster and consider scenarios in which multiple virtual clusters, owned by different virtual organizations, could be run on the same physical cluster.

The rest of this paper is organized as follows. Section 2 provides background on relevant virtualization efforts. Sections 3 and 4 describe conceptual and architectural extensions to the virtual workspace needed to accommodate virtual clusters; Section 4 also describes the virtual cluster implementation used in this project. Section 5 describes experimental evaluation of virtual cluster deployment and management, running application workloads, and running multiple virtual clusters on one real cluster. We conclude in Section 6 with comments on future work.

2. Virtualization and Grid Computing

A virtual machine [5] is an emulation of lower layers of a computer abstraction on behalf of higher layers. A VM representation contains a full image of RAM, disk, and other devices. A virtual machine monitor (VMM) is a software process that manages the hardware resources of the real machine among instances of VMs, thus allowing multiple instances of VMs to run simultaneously on the same hardware. Recent, widespread interest in virtualization led to the development of new and efficient virtualization projects such as VMware [6] and Xen [7].

With superior isolation properties, fine-grained resource management, and the ability to instantiate independently configured guest environments on a host resource, virtual machines provide a good platform for Grid computing [8, 9]. The In-Vigo project [13, 14] and the associated Virtuoso project [15] explored some of the issues involved in combining Grid and virtual machine technology especially as relates to networking and deployment. Our approach differs in that it focuses on virtual workspaces, first-class entities that need to be managed independently of their deployment, treating virtual machines as one of their implementations (infrastructure based on dynamic accounts [16] provides another one).

Driven by community requirements, we also focus on clusters as a primary Grid platform. Such a focus has been recognized by other groups. The Cluster on Demand infrastructure [4] first introduced the notion of a virtual cluster (albeit in its first iteration not using virtual machines). We recognize this effort as complementary to our work; our long-term focus is on describing and managing clusters in the Grid rather than developing tools of local control. Another relevant effort is an exploratory project [17] evaluating virtual machines for Grid computing in clusters: although the authors do not propose a specific architecture, many of the questions they pose are similar to ours.

3. Virtual Clusters

As described in [10], a virtual workspace is composed of workspace metadata (represented as an XML Schema) and implementation-specific information such as a pointer to the image of a VM implementing a given workspace. The intent of the metadata is to capture workspace requirements in terms of virtual resource, software configuration, and other salient characteristics. In this section,

we describe the extensions to the workspace metadata and implementation necessary to represent a new type of workspace: a virtual cluster. We introduce the term *atomic workspace* to describe a workspace consisting of a single execution environment and *cluster workspace* to describe a virtual cluster.

3.1. Virtual Cluster Description

Following conventions common in Grid3 and OSG, we distinguish two kinds of nodes in a virtual cluster: a head-node and worker nodes. The purpose and configuration of a head-node are typically different from those of worker nodes, especially in software and operational setup. Although worker node configurations are similar, they may be assigned different names or their status may be different (for example, some nodes may not be operational). For these reasons, we represent each node of a cluster by a separate atomic workspace, each with its own metadata and image handle, as described in [10]. A set of atomic workspaces representing the nodes of the cluster is then wrapped by an XML section containing the information about the cluster as a whole such as its type (cluster/atomic), name, number of nodes, or time it was instantiated. All other information about a workspace is derived from the metadata of the atomic workspaces describing those nodes.

```
<xs:simpleType name="vwType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="cluster"/>
      <xs:enumeration value="atomicvm"/>
   </xs:restriction>
</xs:simpleType>

<xs:element name="virtualWorkspace">
   <xs:complexType>
      <xs:sequence>
            <!-- "generic" section -->
            <xs:element name="type" type="vwType" default="cluster"/>
            <xs:element name="nodeNumber" type="xs:integer"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="creationTime" type="xs:dateTime"
                        minOccurs="0"/>

            <!-- a list of head node and worker nodes -->
            <xs:element ref="hn:headnode"/>
            <xs:element ref="wn:workernode" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

3.2. Virtual Cluster Implementation

A virtual cluster workspace is implemented in terms of multiple virtual machine images. However, since it would be wasteful to stage several copies of potentially identical worker node images, we preserve that appearance while using various optimization strategies and image reconstruction. The simplest optimization strategy is image cloning: we transfer only one image for all the worker nodes and one image for a head node, and then clone the worker node images at staging or deployment time. This will work for a set of shutdown images but not necessarily for paused images (i.e., images including serialized RAM with execution in progress). We can further

leverage the understanding of image structure to put together the disk content of a VM on the fly: for example, Xen represents the disk associated with an image as a set of partitions each represented by a separate file; these partitions can be mounted on deployment. Some of them may be available locally reducing staging time of VMs. In cases where differences between images are less well articulated, we can experiment with different techniques such as described in [10].

3.3. Configuring a Grid3 Virtual Cluster

Grid3 [1] and Open Science Grid [2] support production-quality petascale Grid infrastructure for large-scale scientific applications. Membership in these Grids imposes configuration constraints on the participating sites. These requirements include specific versions of operating system, NFS running across the head node and worker nodes, a scheduler (such as PBS [18]), and potentially other software. We used the guidelines available in [19] to prepare the configuration of our virtual cluster; similar guidelines could be used to prepare clusters for other communities and organizations. After a Grid3 cluster workspace configuration is created in the VW Repository, the virtual cluster can be deployed using the services described below.

4. Interacting with Virtual Clusters

As described in [10], our architecture is based on two sets of services: VW Repository which allows authorized Grid clients to configure, manage, and inspect workspaces; and VW Manager, which orchestrates workspace deployment. Configuration and management include actions such as adjusting the software configuration of a workspace or extending its lifetime. While the current creation process relies on pre-created images, we are working on incorporating various increasingly interesting configuration options relying on technologies such as Pacman [20] or SmartFrog [21]. Workspace deployment typically involves communication with a VMM running on a physical host.

In this section we describe the changes to the architecture needed to support virtual cluster workspaces.

4.1. Virtual Workspace Manager

The VW Manager is implemented as two Web Service Resource Framework (WSRF) [22] services: VW Manager Factory and VW Manager Service.

The primary VW Manager Factory operation is **create**: it creates an "active workspace" resource and starts the VM associated with the workspace reference provided as input. To allow better control of potentially heavyweight actions, we exposed two additional operations in this interface: **load**, which loads the VM image corresponding to a virtual machine into the VMM, and **stage**, which stages data necessary to workspace deployment (i.e., the VM image) to the resource where it is to be deployed. If the stage operation has not been called before load, the load operation will call it, and if load has not been called before create, it will be called by the create operation. In general however, the staging operation need not be associated the VW Manager. In the future, we plan to experiment with the Replica Location Service (RLS) [23] via the VW to keep track of and manage copies of VM images associated with specific workspaces.

The VW Manager Service implements the following operations: **pause/unpause**, which pauses/unpauses the VM associated with an identified workspace; **stop**, which shuts down the VM associated with a workspace; and **unstage,** which releases the local hold on workspace data. After the unstage operation, the client may no longer assume that workspace data is available locally.

4.2. VW Manager for a Virtual Cluster: Modus Operandi

The interaction with the VW Manager takes place as shown in Figure 1. A workspace is staged to the VW Manager running on the head node of the physical cluster with a workspace as argument. The head node's VW Manager first establishes whether the workspace is a virtual cluster using the type element described in Section 3.1. For a cluster workspace, it orchestrates the networking (see Section 4.3) and other administrative information (modifying workspace implementation to reflect this configuration if needed) for each worker node workspace and stages the worker node workspaces as well as the image reconstruction instructions to the VW Managers running on the nodes hosting the worker nodes.

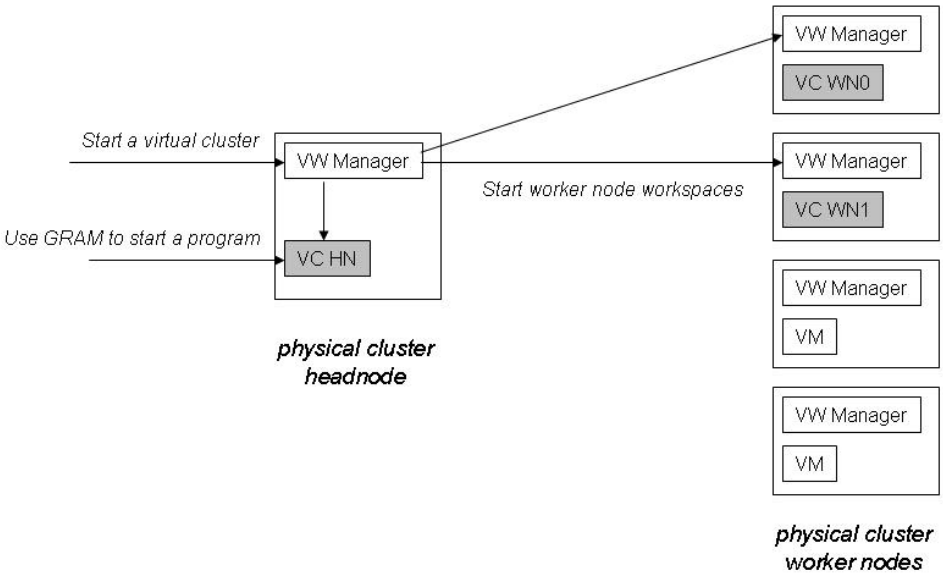


Figure 1: Physical and virtual clusters: the gray areas denote a virtual cluster.

The load and start operations are simply concurrently repeated to all the VW Managers on the cluster, as are the pause, stop, and unstage operations. When a virtual cluster is deployed on a physical cluster, a GRAM service comes up as part of the head node startup operation and advertises its EPR. A client can then use GRAM to submit jobs to the virtual cluster.

4.3. Implementation Details

We make the following assumptions about the physical cluster that will host virtual clusters. Each machine on the cluster (worker nodes as well as the head node) must run the Xen 2.0 VMM, GT4 and the VW Manager. We also assume GridFTP [24] installation used to transfer images from head node to worker nodes.

Networking has been handled as follows. All nodes in the virtual cluster are assigned addresses from a private network sharing the same subnet as the physical cluster and are thus able to communicate among themselves. The VW Manager keeps track of the IP addresses of physical nodes as well as available segments of unused IP addresses and assigns segments at the time of staging.

The virtual head node is equipped with two virtual network cards: one with a public IP address (allocated from a reserved pool) and one with a private IP. The VW Manager configures its Xen configuration file by adding relevant virtual network card information. Since only one address can

be configured by modifying the Xen configuration file, it then mounts the virtual head node image and configures the /etc/network/interfaces file to add information about the second virtual network card. A client can submit jobs by using the external IP address but does not communicate with nodes inside the cluster.

5. Experimental Results

To assess the practicality and trade-offs involved in using a virtual cluster, we ran experiments evaluating its startup time and management time, the impact of running on a virtual cluster for applications, and the feasibility of sharing a resource between more than one virtual cluster.

We ran our experiments on a testbed constructed on top of the Chiba City cluster at Argonne National Laboratory (ANL) [25]. We divided the testbed into two sections: Xen-enabled and pure Linux (running no Xen software). Each section includes 8 nodes on a 100 Mbps LAN. Each node is equipped with two 500 MHz Intel PIII CPUs (with a 512 KB cache per CPU), 512 MB main memory, and 9GB of local disk. All the nodes run Linux kernel 2.4.29.

Nodes of the Xen-enabled section were configured with Xen 2.0 distribution (domain 0 runs port of Linux 2.4.28 and the user domain runs port of Linux 2.6.10) and rebooted with XenoLinux. Domain 0 was booted with 128 MB memory, while the user domain (unless specified otherwise) was booted with 360 MB. Nodes of the straight Linux section were running Linux 2.4.29 without SMP.

5.1. Creating and Interacting with a Virtual Cluster

In this group of experiments we look in detail into the actions required for creation and management of a virtual cluster. In the first experiment, we evaluate virtual cluster staging, and in the second the time spent on significant virtual cluster operations described in Section 4.1.

To evaluate staging, we assume that all the relevant workspace data as described in Section 3.2 has already been staged in to the local disk of the head node of the physical cluster. This data contains one head node image and one worker node image. The objective of the test is to evaluate how long it takes to stage virtual worker node images configured to requirements described in Section 3.3 to physical nodes so that they are ready for deployment.

The staging time will clearly depend on the size of VM nodes. Table 1 shows the information about the VM images for virtual cluster head node and worker node. Normally, a Xen image contains a configuration file, disk image, and optionally a representation of RAM if is a paused image. Since we start with “cold” images (shutdown), the primary constituent of the image is VM’s disk. Besides the Debian Sarge OS, the GT 3.9.4 package occupies the largest storage size in the virtual cluster head node image. The remaining constituents, including OpenPBS (Torque), NFS kernel server, MPICH 1.2 and support infrastructures, altogether take slightly more than 20 MB. The worker node is only half as big; and in addition to the operating system, OpenPBS (Torque), and MPICH, it includes the NCBI BLAST software and one nucleotide sequence database. The latter takes a very large share of the whole image storage. Note that this arrangement stages data as part of the workspace configuration. Alternatively, data could be staged at run-time after the workspaces have been deployed.

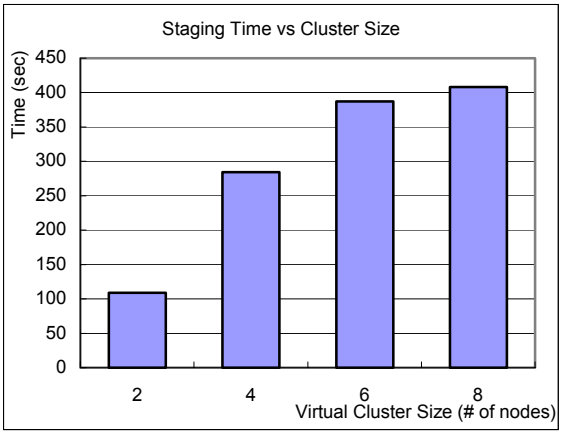
Table 1: VM image sizes

Node Type	VM Image(s) Size	Primary VM Image Constituents
Head node	1.1 GB file system image (140 MB free space) and 200 MB swap image (generated by head node VW Manager on the fly)	Debian Sarge 3.1 OS (~320 MB) GT 3.9.4 with GRAM (~312 MB) JDK 1.4.2 (~60 MB) Apache Ant 1.6.2 (~4 MB) PostgreSQL 7.4 (~10 MB) Torque (OpenPBS) 1.2.0 (<1 MB) NFS kernel server (~200 KB) MPICH 1.2 (~6 MB)
Worker node	550 MB file system image (50 MB free space) and 200 MB swap image (generated by worker node VW Manager on the fly)	Debian Sarge 3.1 OS (~320 MB) MPICH 1.2 (~6 MB) Torque (OpenPBS) 1.2.0 (<1 MB) BLAST month.nt database (~130MB)

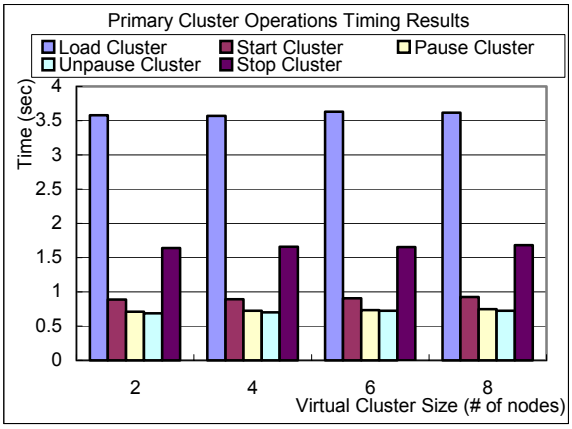
Staging is implemented as follows: a copy of the worker node image is transferred via GridFTP [24] to each physical node on which the cluster will be deployed. Then the VW Manager running on each physical node modifies the image to integrate it into the virtual cluster (modifying the NFS and PBS configurations). Similar modification has to be made to the virtual head node image (which stays on the head node of the physical cluster).

Figure 2(a) shows the staging times for different sizes of requested VW cluster. As expected, the times are dominated by transfer time and increase as the number of nodes of the virtual cluster increases. Although the transfers are started concurrently, they share the same physical link between the head node to the LAN switch, which can support a transfer rate no greater than 10 MB/sec. As the cluster size increases, the transfer rate also increases slightly because GridFTP is able to leverage several concurrent connections.

Once the virtual worker nodes are staged, cluster startup is relatively fast. Here, we measure the timings of several VW cluster operations as described in Section 4.1; the results are shown in Figure 2(b). Each of these operations simply broadcasts the operation message to physical worker nodes. The load operation cost is dominated by bringing the VM image from disk to memory (~2.83 sec loading cost, which includes ~1 sec image reconstruction cost compared to ~0.787 sec



(a) VW cluster staging time



(b) VW cluster: primary operations

Figure 2: Deploying and managing a virtual cluster.

communication cost). Compared with the other operations, the cost is ~0.453 sec for start, ~0.27 sec for pause, and ~1.01 sec for stop.

The staging time is currently the major component of virtual cluster startup time and takes minutes (rather than seconds as for the other operations). It could be decreased by judiciously decreasing the image size of the worker node, for example, by "caching" frequently used image components (such as the operating system) or even whole images on the worker nodes of the physical cluster. In addition, it may be advantageous to move the staging of application data to the job staging phase where it could be overlapped with other actions. We are also investigating using multicast techniques or cascading image transfers (where other nodes receive the images from other nodes) to increase the effective bandwidth of the cluster stage operation.

5.2. Running Jobs in Virtual Cluster

Once a virtual cluster has been deployed, it can be used as a virtual partition and mirror a real Grid3 cluster setup, or it can be used as an execution environment for a sequential or parallel application. In this section we look into the efficiency of running a Grid3 submission engine as well as parallel MPI-based applications on a virtual partition. In our experiments we use the BLAST application [11] from the GADU project [12] as a current Grid3 workload.

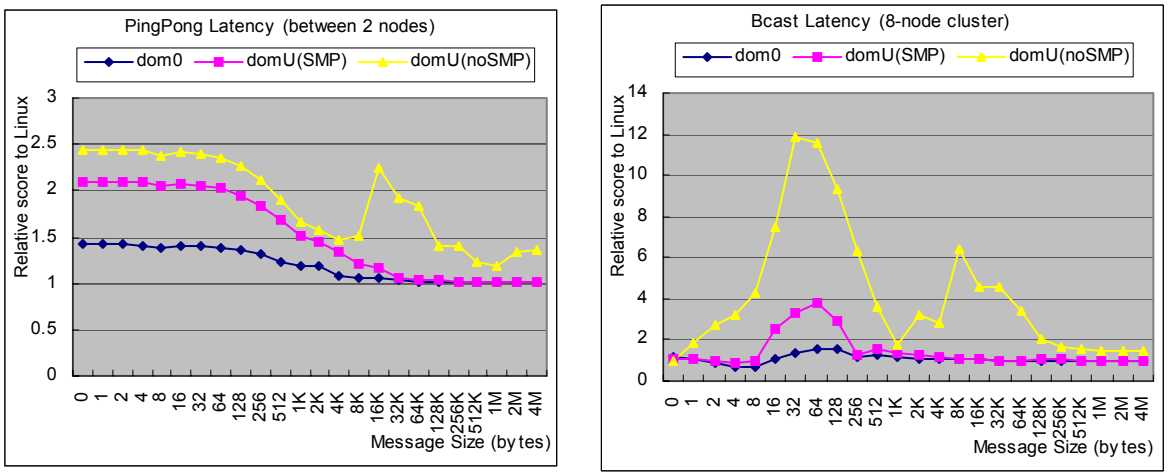
5.2.1. Batch Submissions in a Virtual Cluster

We first timed GT4 job submission to a virtual cluster and found little difference between real and virtual cluster. For simple job submission, the submission times to virtual cluster and real cluster were ~4.839 sec and ~4.733 sec, respectively (median of 10 tries with 0.103 and 0.986 standard deviation respectively), for jobs requiring 2 MB data stage-in, the times were ~27.528 sec and ~27.572 sec, respectively (median of 10 tries with 0.563 and 0.538 standard deviation, respectively).

We then compared the job execution time of a batch of 100 sequential BLAST jobs submitted to an 8-node virtual and a real cluster. The jobs were scheduled by the local PBS scheduler interfaced through GT4 GRAM to run on each cluster node, and all the data required for execution has already been prestaged. The results showed that the virtual cluster adds little overhead to regular cluster submissions: the median time per job for VW cluster was 24.493 sec as compared to 24.470 sec for the real cluster with 0.789 and 0.641 standard deviation respectively. Although the 360 MB memory of the virtual node is smaller than the memory of the real node (512 MB), this difference had no effect on the execution time because at ~140 MB the BLAST database is much smaller than the memory sizes of both the virtual and the real nodes.

5.2.2. Running Parallel Applications in a Virtual Cluster

To understand the effect that using Xen will have on communication between virtual machines and therefore on execution of a data-parallel program, we ran a series of microbenchmarks, the Pallas MPI Benchmarks (PMB) [26]. The full results are available on the Web [27]. Figure 3 shows representative examples.



(a) PingPong benchmark's transfer latency.

(b) Bcast's transfer latency (8 nodes)

Figure 3: Communication cost in virtual cluster. The figures show communication costs for Xen domain 0, Xen user domain (virtual cluster) with SMP turned on, and Xen user domain without SMP. The data has been normalized to communication cost on the real cluster. The x-axis shows message size.

Figure 3(a) shows the difference in communication cost between the real and virtual cluster. Across all of our tests the overhead of domain 0 has been relatively stable, at most 50%, which, if present, typically decreases with message size. Some benchmarks (e.g., SendRecv) showed surprisingly "better than real" performance. The increase in communication cost of domain U over domain 0 is explained by an additional latency factor added when messages from domain U are transferred via the virtual network interface (what we will refer to as the "Xen latency").. In addition, messages waiting in the queue associated with this interface may be delayed because, in order to save on CPU cycles, Xen does not send the notification each time it receives a packet; this strategy probably contributes to occasional spikes as that visible in Figure 3(b). Overall, while we found the "Xen latency" increase to be generally reasonable, it is prone to occasional spikes that make communication expensive. Context switching between domain 0 and domain U in the noSMP case can contribute significantly to latency increase, generally exacerbates any spikes (see Figure 3(b)), and, in our experience, introduces a significant nondeterministic factor to communication. Although the "Xen latency" can be significantly improved and controlled when working with smart networking technology such as Myrinet or InfiniBand [28], the problems associated with context switching is likely to remain.

To put these experiments in context, we ran a parallel BLAST (mpiBLAST [29]) in a virtual cluster. mpiBLAST segments a BLAST database among worker nodes and uses a master-worker communication style (using send/receive and broadcast operations) to assign and collect work. We ran mpiBLAST on both 8-node virtual cluster in SMP mode and real cluster. The 130 MB nucleotide sequence database is segmented into 8 fragment databases, and each is assigned to a worker node. We use the same query (less than 1 KB) used in our batch BLAST experiment. mpiBLAST greatly speeds the BLAST search, but the communication overhead does not significantly affect virtual cluster operation: the average query time at 3.836 sec for the 8-node virtual cluster is approximately the same as the 3.815 sec for the 8-node real cluster.

5.3. Multiple Virtual Clusters

Our final experiment examines what happens if two virtual organizations share the same CPU and memory resources on a cluster, that is, if more than one virtual cluster is deployed on the same nodes of one physical cluster.

To examine the issue, we deployed two 8-node virtual clusters on 8 physical cluster nodes so that each physical node is running one VM from each virtual cluster. In order to ensure that the two VMs "fit" equally well into the physical memory of the node, we configured the VMs with 180 MB of memory (unlike in the previous experiments where they were configured with 360MB memory). We then repeated the experiment from Section 4.2.1 on the cluster.

We first ran this experiment in noSMP mode (test 1); this deployment configuration forces two VMs (one from each virtual cluster) to compete for one CPU resource (leaving the other CPU on the node unused). For comparison, we reran this experiment in SMP mode (test 2) such that we pinned one VM from each virtual cluster to one of the two CPUs on every node.

Table 2 shows the average job execution time results for both testing scenarios. We observe that the time is more than twice as long when two virtual nodes compete for the same CPU: since BLAST is CPU intensive, this overhead is due to context switching. Further, BLAST performance in test 2 (~44.289 sec) is worse than BLAST performance in Section 4.3.2 (~24.493 sec). We hypothesized that this result was due to configuring the VMs with less memory -- BLAST is memory intensive, and when the database sequence is too big to store in memory, the performance deteriorates -- and we confirmed this hypothesis by rerunning the experiment in Section 5.2.1 with VM configured in the same way.

Table 2: BLAST execution on competing and noncompeting virtual clusters

Metrics	Test 1	Test 2
Median time per job	109.39 sec	44.289 sec
Stand Deviation	5.09	1.064

These results point to the fact that running more than one virtual cluster on the same physical cluster partition can be costly. While we can map different VMs to different available CPUs on the node, reducing the available memory can significantly reduce the performance of memory intensive applications.

6. Conclusions and Future Work

We have described how clusters of virtual machines can be used to provide on-demand customized execution environments in the Grid for nontrivial environments. A cluster configuration, such as the Grid3 environment described here, can be defined and configured ahead of time and then easily deployed based on need as a ready-to-go "computation capsule." A comparison of running both parallel and batch applications on such clusters shows that it can be done with relatively small overhead.

Barring staging time, such deployment is independent of cluster size and is not a costly operation -- a cluster can be set up in a matter of a few seconds. The price of the on-demand availability of arbitrary configurations, including the operating system, translates primarily into the time of staging arbitrary VM images to worker nodes. This time can be flexibly reduced by making the VM images less arbitrary (i.e., caching image parts on worker nodes as discussed in Section 3.2) or hidden by overlapping staging with execution, as is commonly done today for job execution.

Still, the cost of virtual cluster deployment and management probably justifies the expectation that they may be used on a VO level for large groups of short jobs (therefore requiring job management infrastructure inside a cluster) as well as for single long-running jobs. As expected, the cost of running batch jobs in a virtual cluster, such as is common for a Grid3 workload, was very acceptable. Fully understanding its effect on data-parallel applications however, will require more investigation.

While our work gives some answers to the trade-offs involved in evaluating the use of virtual clusters, we realize that more work will have to be done to examine these trade-offs in detail, especially in view of the constantly changing technology. In addition to issues of efficiency trade-offs, issues of migrating virtual clusters or their use in load balancing and improving response are still unaddressed.

Acknowledgments

We acknowledge the help of Mike Wilde and Doug Scheftner for helping us configure a Grid3 cluster and for helping us select and install Grid3 applications. We also thank Rick Bradshaw for assistance with the Chiba City cluster testbed at Argonne. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

1. Foster, I. and others. *The Grid2003 Production Grid: Principles and Practice*. in *IEEE International Symposium on High Performance Distributed Computing*. 2004: IEEE Computer Science Press.
2. *Open Science Grid (OSG)*: http://www.opensciencegrid.org/.
3. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of Supercomputer Applications, 2001. **15**(3): p. 200-222.
4. Chase, J., L. Grit, D. Irwin, J. Moore, and S. Sprenkle, *Dynamic Virtual Clusters in a Grid Site Manager.* accepted to the 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
5. Meyer, R.A. and L.H. Seawright, *A Virtual Machine Time Sharing System.* IBM System Journal, 1970. **9**(3): p. 199-218.
6. Sugerman, J., G. Venkitachalan, and B.H. Lim. *Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor*. in *USENIX Annual Technical Conference*. 2001.
7. Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. *Xen and the Art of Virtualization*. in *ACM Symposium on Operating Systems Principles (SOSP)*.
8. Figueiredo, R., P. Dinda, and J. Fortes. *A Case for Grid Computing on Virtual Machines*. in *23rd International Conference on Distributed Computing Systems*. 2003.
9. Keahey, K., K. Doering, and I. Foster. *From Sandbox to Playground: Dynamic Virtual Environments in the Grid*. in *5th International Workshop in Grid Computing*. 2004.
10. Keahey, K., I. Foster, T. Freeman, X. Zhang, and D. Galron, *Virtual Workspaces in the Grid.* ANL/MCS-P1231-0205, 2005.
11. *BLAST*: http://www.ncbi.nlm.nih.gov/BLAST/.

12. Rodriguez, A., D. Sulakhe, E. Marland, V. Nefedova, N. Maltsev, M. Wilde, and I. Foster. *A Grid-Enables Service for High-Throughput Genome Analysis*. in *Workshop on Case Studies on Grid Applications*. 2004. Berlin, Germany.
13. Adabala, S., 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, 2004.
14. Krsul, I., A. Ganguly, J. Zhang, J. Fortes, and R. Figueiredo. *VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing*. in *SC04*. 2004. Pittsburgh, PA.
15. Sundararaj, A. and P. Dinda. *Towards Virtual Networks for Virtual Machine Grid Computing*. in *3rd USENIX Conference on Virtual Machine Technology*. 2004.
16. *Workspace Management Service*: http://www-unix.mcs.anl.gov/workspace/tech_preview_2/docs/index.html.
17. Havard, K., F. Bjerke, and R.J. Andresen, *Virtualization in Clusters*: http://www.idi.ntnu.no/~havarbj/clust_virt.pdf.
18. *Portable Batch System*: http://www.openpbs.org.
19. *Virtual Data Toolkit (VDT) 1.3.0 Requirements*: http://www.cs.wisc.edu/vdt/releases/1.3.0/requirements.html.
20. Youssef, S., *Pacman: A Package Manager*. 2004: http://physics.bu.edu/~youssef/pacman/.
21. *Smart Framework for Object Groups (SmartFrog)*: http://www.hpl.hp.com/research/smartfrog/.
22. Foster, I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, *Modeling Stateful Resources with Web Services*. 2004: www.globus.org/wsrf.
23. Chervenak, A.L., N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, *Performance and Scalability of a Replica Location Service*, in *SC2004 High Performance Computing, Networking, and Storage Conference*. 2004.
24. Allcock, W., *GridFTP: Protocol Extensions to FTP for the Grid*. 2003, Global Grid Forum.
25. *Chiba City, the Argonne Scalable Testbed Cluster*: http://www-unix.mcs.anl.gov/chiba/.
26. *Pallas MPI Benchmarks*: http://www.pallas.com/e/products/pmb/.
27. Zhang, X. and K. Keahey, *Evaluation of a Virtual Xen Cluster Using the Pallas MPI Benchmarks Suite*. 2005: <http://people.cs.uchicago.edu/~hai/vm1/vcluster/PMB>.
28. Fraser, K., S. Hand, R. Neugebar, I. Pratt, A. Warfield, and M. Williamson, *Safe Hardware Access with the Xen Virtual Machine Monitor.* 2004.
29. Darling, A.E., L. Carey, and W. Feng. *The design, implementation and evaluation of mpiBLAST*. in *ClusterWorld Conference & Expo and the 4th International Conference on Linux Clusters*. 2003.