A Study of a KVM-based Cluster for Grid Computing

Michael Fenn, Michael A. Murphy[*], and Sebastien Goasguen[†]
School of Computing
Clemson University
100 McAdams Hall
Clemson, SC 29634-0974 US
{mfenn,mamurph,sebgoa}@cs.clemson.edu

ABSTRACT

We present a performance study of a virtualized cluster based on the virtualization system KVM. We show benchmark results from the High Performance Computing Challenge (HPCC) application suite including the High Performance Linpack (HPL) benchmark. We also present the mechanism by which this cluster is connected to the Open Science Grid (OSG). Our results show that jobs with low amounts of network communication will only suffer moderate overhead (≈10%) due to virtualization, while MPI applications will suffer from a considerable overhead in the 60% range. The KVM cluster under investigation does prove to be suitable for current High Throughput Computing (HTC) grid usage on OSG where the Condor middleware is used.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies Performance attributes*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Experimentation, Measurement, Performance

Keywords

KVM, cluster, benchmark, grid, Condor

1. INTRODUCTION

A Virtual Organization (VO) allows scientists to share resources on the grid-provided computing facilities without regard to physical location. Each VO can have dramatically different requirements and membership, with objectives and computing resource needs changing over time [6]. Given the diverse nature of VOs, and the challenges involved in providing suitable computing environments to each VO, virtual machines (VMs) present a promising abstraction mechanism for providing grid computing services [5].

[*]This material is based upon work supported under a National Science Foundation Graduate Research Fellowship
[†]This work is also supported by an IBM faculty award

We evaluate the performance of a High Performance Computing (HPC) virtualization model. This model defines a cluster that would not run jobs directly, but would instead host Virtual Machines (VMs). Such a cluster could be provided by a Virtualization Service Provider (VSP) – an organization very much like the Amazon EC2 system that provides compute services without having to maintain each end-user's application software. Virtual Organizations (VOs) can contract with VSPs to provide compute power without having to worry about hardware or software compatibility. A VO would be free to experiment with a virtual cluster locally, only submitting it to a VSP when the configuration was perfected, thus saving time and effort. A VO would be able to deploy to a VSP's hardware with reasonable assurances that the operating environment will be fully compatible. However, a virtualization model requires acceptable performance in order to make the advantages of virtualization worthwhile.

This paper briefly describes a model for developing Virtual Organization Clusters (VOCs) for the purpose of grid computing. A VOC is a cluster made of virtual machines configured to support a single VO and deployed by a VSP. A VSP and a VOC have been developed at the Cyberinfrastructure Research Laboratory at Clemson University, where they are in the testing and refinement stages as a resource on the Open Science Grid Integration Testbed. A physical cluster was created with the Kernel-based Virtual Machine (KVM) hypervisor along with the popular Xen hypervisor through a dual-boot setup. A test VOC was constructed with CentOS 5.2 Virtual Compute Nodes (VCNs), providing VO compute services to the Engage OSG VO. Initial benchmarking presented in this paper indicate that the VOC is suitable for vanilla-universe Condor jobs. Severe networking overhead was discovered in KVM, creating large penalties in jobs which heavily leverage the network, including those using the Message Passing Interface (MPI), such as a subset of the High Performance Computing Challenge (HPCC) benchmarking suite.

The remainder of this paper is organized as follows: Section 2 provides a short summary of related work. Section 3 presents the KVM-based cluster under investigation and discusses the implementation of the virtual clusters, while Section 4 details the physical hardware configuration on which the VOC was built. Measurements are presented in Section

5, followed by conclusions and future research directions in Section 6.

2. RELATED WORK

One promising solution to the problem of providing different VOs with different environments is virtualization. Figueiredo *et. al.* [5] proposed the use of virtualization technologies to allow different users to have administrative access to their own environments. When designed in such a way, grid resources could be transparently moved between physical clusters via migration techniques such as the one described in [8]. Virtualized systems can provide a greater degree of isolation between users (VOs) than traditional multiprogramming systems [5].

Ideally, a virtual organization could be hosted entirely inside virtual machines, allowing VOs to be mobile and manageable across wide area networks. There are already proposed models for virtualization of clusters [1, 10, 13, 14]. Several different hypervisors have been studied previously, but we focus on the Kernel-based Virtual Machine (KVM) [15]. Utilizing a hypervisor-equipped physical cluster to virtualize another entire cluster would, in effect, enable each VO to have its own dedicated computational cluster.

One challenge when constructing virtual clusters is providing networking support. Several networking libraries have been developed that could permit each VOCs network to be isolated from both the physical network and other virtual networks. Low-level virtualized networks provided by both Virtuoso [16] and Virtual Distributed Ethernet (VDE) [4] can be used to interconnect individual VCNs to make a single VOC.

In this paper we present extensive performance results that validate the potential use of VMs in grid computing.

3. VIRTUAL CLUSTER MODEL

An essential component of any virtual machine model is the choice of hypervisor. We compare two, KVM and Xen, in Sections 3.1 and 3.2 respectively. The configuration of the virtual compute nodes is disussed in Section 3.3, and their deployment to OSG is described in Section 3.4.

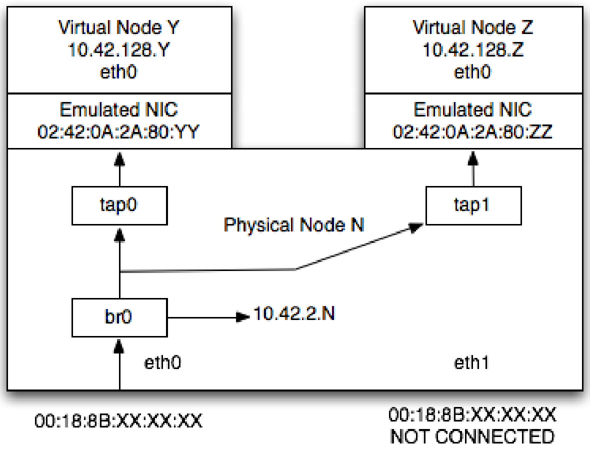
3.1 Kernel-based Virtual Machine (KVM)

We chose to use Qumranet Inc.'s Kernel-based Virtual Machine (KVM) as our hypervisor [7]. KVM is an extension of the well-known QEMU emulator with support for the x86 VT extensions. These extensions allow virtual machines to make system calls without unnecessarily invoking the host kernel, thus potentially saving two context switches [18]. KVM requires a recent Linux kernel with the KVM modules enabled. This virtualization system differs from other virtualization technologies that require heavily modified kernels, whose development is not often current with the mainline kernel. We originally used KVM-57, but issues with the emulated network interface card (NIC) compelled us to upgrade to KVM-77.

KVM inherits all QEMU tools, thus supporting the QEMU Copy-on-write (QCOW) disk image format. QCOW supports a *snapshot* mode for disk I/O where all disk writes are directed to a temporary file and are *not* persisted to the original image. This mode allows multiple VMs to be run from a single master disk image mounted from a network location. Such an arrangement mitigates the storage requirements associated with running a cluster of VMs [9]. *Snapshot* mode also makes destroying a running virtual cluster as simple as sending SIGKILL to the hypervisor on each physical node.

Also inherited from QEMU is the ability to use the TUN/TAP Ethernet bridge available in the Linux kernel. The bridge essentially emulates a switch, allowing each VM to have individual networking resources, separate from both the host and other VMs. Each TAP device acts as a virtual Ethernet endpoint, each connected to a software bridge, along with the hardware Ethernet endpoint as shown in Figure 1.

Figure 1: VCN Bridging



3.2 Xen

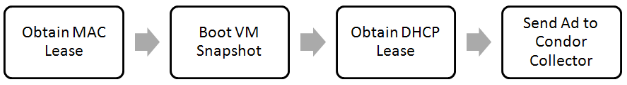
Xen is a hypervisor and is similar to KVM in the sense that it allows multiple guest operating systems, known in Xen parlance as domains, to run concurrently on the same hardware [2]. Xen, however differs from KVM in that it does not rely on any type of CPU instructions for virtualization support, instead it uses a technique known as *paravirtualization* [19]. In this technique, the guest operating system (OS) is modified in such a way that all supervisor instructions (those that would, in KVM, be handled by the VT extensions) are replaced by *hypercalls* into the Xen hypervisor. This allows for much greater performance than pure emulation (QEMU) and competitive performance with VT solutions such as KVM.

3.3 Virtual Compute Nodes

Each VOC is composed of VMs known as Virtual Compute Nodes (VCNs). Every VO that wishes to use a VSP's computational power must submit a VCN image (or set of images) to the VSP, along with some configuration parameters. The VO must carefully construct the VCN image, since each image could be used to start multiple VMs. To this end, a VCN image must not make any assumptions about the network hardware, hostname, or other system-specific settings. Dynamic configuration must be used instead.

As illustrated in Figure 2, when booting a VCN, the hypervisor must first obtain a MAC address, then boot the VM in snapshot mode. One the VCN OS begins to boot, it will receive a Dynamic Host Configuration Protocol (DHCP) lease for its IP address and then join a scheduling pool, such as Condor.

Figure 2: VCN Boot Process



3.4 Grid Integration

Our test VOC was built from two CentOS 5.2 VM images: the VCN image and an OSG gatekeeper image that could be shared among multiple VOC's. CentOS was chosen due to its extensive support for cluster and scientific computing software. The virtual head node was configured to run an OSG gatekeeper through the Virtual Data Toolkit (VDT), Condor central manager and submit daemons, the Ganglia monitoring daemon, and the Ganglia metadata daemon. The virtual compute node was configured with the Condor starter daemon, MPICH2, ATLAS, and the Ganglia monitoring daemon. Our model VOC was designed to be compatible with the OSG, and was successfully deployed to OSG Integration Testbed as the Clemson-Birdnest site. All VCNs were Condor execute nodes and formed a pool managed by the virtual head node running Globus GRAM.

4. PHYSICAL SUPPORT MODEL

Our physical cluster consisted of seventeen Dell PowerEdge 860 1U rack-mount systems and one Dell PowerEdge 2970 2U rack-mount server. Each PowerEdge 860 machine used in this test was configured with a 2.66 GHz dual-core Intel Xeon CPU, 4 GiB of Double Data Rate 2 (DDR2) memory, and an 80 GB hard disk drive. The 2U PowerEdge 2970 server was configured with three 250 GB hard disk drives in a Redundant Array of Independent Disks, level 5 (RAID-5) configuration that was employed to host installation images, user home directories, network services, and a shared VM image store exported via a Network File System server. One of the 1U nodes was used both as a physical head node and as a host for the shared virtual head node. The other sixteen 1U nodes hosted two VCN compute images each. For an overview of the VOC layout, see Figure 3.

4.1 Host Operating System Configuration

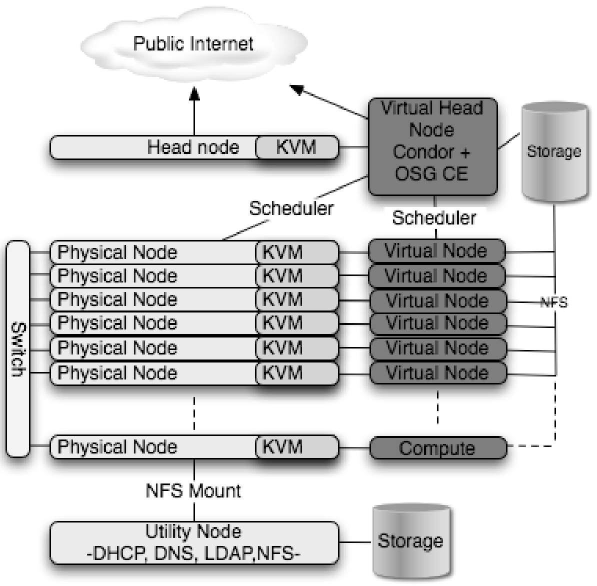
CentOS 5.2 was chosen for the host Operating System (OS) for many of the same reasons as it was chosen for the VCN OS. Sharing of software packages was an additional benefit of having a homogeneous OS environment. While CentOS does ship with KVM support, we chose to build our own KVM packages from the most recent sources. This allowed us to leverage the rapid pace of KVM development.

All compute nodes were install via a Red Hat-style kickstart script, with some custom additions. Since kickstart installations can only utilize packages from a single repository, we wrote a custom addition to the kickstart system that allowed packages from multiple repositories to be installed automatically.

4.2 Physical Support Services

A Lightweight Directory Access Protocol (LDAP) server was used as a central repository of configuration information for both the physical nodes and VCNs, including hostnames, Internet Protocol (IP) addresses, and Media Access Control (MAC) addresses. VCN MAC addresses were generated as

Figure 3: VOC Organization



locally-administered addresses to avoid any potential conflicts on the local subnet. To aid in administration of the physical nodes, an LDAP-aware, batch, remote administration tool was written[1].

The PowerEdge 2970 utility system provided a Domain Name System (DNS) server (dnsmasq) and a Dynamic Host Configuration Protocol (DHCP) server (ISC DHCPD). To maintain a single configuration source, utilities were written to generate DNS and DHCP configuration files dynamically from LDAP. All VCN images were hosted on a Network File System (NFS) export from the utility system.

5. RESULTS

Four different performance tests were executed to evaluate the VOC: the standard supercomputing HPL [3] benchmark, HPL block size tuning where sensitivity of the overall performance is measured with respect to various block sizes, the High Performance Computing Challenge (HPCC) benchmark that complements HPL with measures of bandwidth and additional floating point operations, and VCN boot time measurements.

5.1 High Performance Linpack (HPL)

Our 16 node physical cluster had a theoretical peak of 341 GFLOPS, calculated using 2 CPU cores per node clocked at 2.66 GHz, with 4 FLOPS per cycle per core. HPL tests were performed to compare the actual performance to the theoretical peak. The following HPL parameters were optimized for our cluster and remained constant throughout these tests: block size (NB), process mapping (PMAP), threshold, panel factorization (PFACT), recursive stopping criterium (NBMIN), panels in recursion (NDIV's), recursive panel factorizations (RFACT's), broadcast (BCAST), lookahead depth (DEPTH), SWAP, swapping threshold, L1 form, U form, Equilibration, and memory alignment. The problem

[1] http://cirg.cs.clemson.edu/software/stoker/

size (N) was derived with the formula:

$$N = \sqrt{nDU} \tag{1}$$

where n was the number of nodes tested, D was the number of double-precision floating-point numbers that can fit into a single node's memory (bytes of node memory / 8), and U was the ratio of memory available for user processes to total memory (we used U=0.8 to leave space for OS processes). All tests were run with ATLAS 3.8.1 (tuned separately for physical nodes and VCNs) and MPICH2 1.0.5p4.

For our physical cluster, the optimal value of N was computed to be 83,000. With this value of the HPL problem size, the performance of the physical hardware was measured at 190 GFLOPS, or 56% of theoretical peak, a reasonable value for a cluster utilizing standard Gigabit Ethernet networking.

5.2 Boot Times

In order to determine the practicality of scheduling VMs as processes, boot times were measured and compared to the physical hardware. An XML-RPC boot timing server was deployed to monitor the virtual systems. Boot times for the physical nodes were measured by hand with a chronograph. Table 1 summarizes boot time test results.

Table 1: Boot Times (seconds)

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

The physical boot process was divided into three phases: Pre-eXecution Environment (PXE) timeout, a GRand Unified Bootloader (GRUB) timeout, and the actual kernel boot time. The Actual Boot column does not include either the PXE or the GRUB timeouts. The VM boot time measures the amount of time from KVM initiation on the host until all daemons had been started on the guest. Approximately 10 more processes were found to be running on the VCN than on the physical host. On average, the VCNs required an extra 21 seconds to boot.

5.3 High Performance Computing Challenge (HPCC) Benchmark

Tables 2, 3, and 4 are comparisons of the VOC to the physical cluster by means of the HPCC benchmark. Tests were run on a single physical node (single process) versus a single VCN. Two tests were then run on the full cluster, the first utilizing one dual-CPU VCN per physical node and the second with two single-CPU VCN per physical node. All parameters except problem size (N) and block size (NB) were maintained from the previous HPL tests. Problem sizes were scaled to fit into available VOC memory (1 GiB per CPU per node, see Equation 2)

$$Mem_{VOC} = 1GiB \cdot N_{CPU} \cdot N_{VCN} \tag{2}$$

according to Equation 1, and block sizes were increased in order to compensate for latency. The overhead due to virtualization was calculated with the formula:

$$Overhead = \frac{Physical - VOC}{Physical} \cdot 100\% \tag{3}$$

In cases when larger values indicate worse performance (i.e. latency) the result of Equation 3 was multiplied by -1. Otherwise, negative values indicate increased performance of the VOC over the physical cluster.

Under KVM, single-node virtualization overhead ranged from under 10% to around 16% with G-FFTE being an outlier at 42% (Table 2). Xen fared similarly, except its outlier was G-Random Access at 35%. However, the full cluster overhead for MPI applications under KVM (Tables 3 and 4) is quite high at 52% for G-HPL with single-CPU VMs and 85% with dual-CPU VMs. Xen performed much better with penalties of 23% for G-HPL with single-CPU VMs and 30% with dual-CPU VMs. RandomRing latency was approximately three times worse with KVM than with the physical hardware. Associated HPL performance of the VOC was thus quite poor. Also note that Xen's RandomRing latency was about 30% lower than KVM's latency in the same benchmark. We believe this to be a contributing factor to KVM's poor HPL performance due to excessive context switching between user and kernel mode in its network code.

We believe that the large difference in best-case RandomRing latencies (Table 4) between KVM ($228\mu s$) and Xen ($74\mu s$) can be attributed to Xen's paravirtualization of guest network devices. Xen's network device drivers can make a call directly into the Xen hypervisor, avoiding any context switches. KVM's network device drivers must make a system call, which which is the trapped by the VTx instructions and then passed to the user-mode KVM process. This unnecessary context switch would cause additional latency, but it is also possible that Xen's network code is simply more mature than KVM's code. Nevertheless, we believe that context switches play some role in Xen's improved network latencies, and therefore, its improved HPL performance.

5.4 Block Size (NB) Tuning

To test the hypothesis that increased latency significantly decreases performance for MPI jobs, a test was run with varying block sizes. Figure 4 summarizes the results on both the physical cluster and the VOC. The total number of blocks that need to be distributed is the problem size divided by the block size, so greater block sizes should reduce the total number of transfers, thus reducing the effects of latency. HPL performance increased with increasing block size, reaching an optimal point at a block size of 400. Above this threshold, performance began to decrease as load-balancing became inefficient.

Table 2: Physical vs. Virtualized, Single Process

Process Grid	1x1 Physical	1x1 Xen VOC	Xen Overhead	1x1 KVM VOC	KVM Overhead
Problem Size	10300	10300	0%	10300	0%
G-HPL (GFLOPS)	7.913	7.393	6.566%	7.218	8.771%
G-PTRANS (GB/s)	0.729	0.588	19.415%	0.635	12.946%
G-Random Access (GUP/s)	0.002	0.001	35.519%	0.002	15.818%
G-FFTE (GFLOPS)	0.799	0.658	17.733%	0.461	42.370%
EP-STREAM Sys (GB/s)	3.866	3.375	12.704%	3.808	1.491%
EP-STREAM Triad (GB/s)	3.866	3.375	12.704%	3.808	1.491%
EP-DGEMM (GFLOPS)	8.348	7.689	7.892%	7.682	7.977%
RandomRing Bandwidth (GB/s)	N/A	N/A	N/A	N/A	N/A
RandomRing Latency (μs)	N/A	N/A	N/A	N/A	N/A

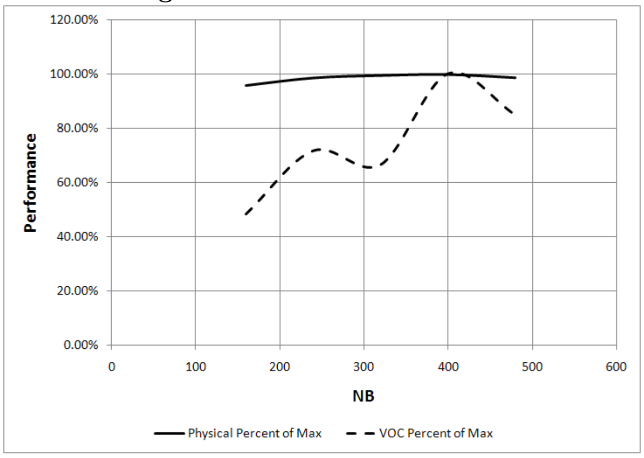
Table 3: Physical vs. VOC, One 2-CPU VM per Physical Node (32 processes)

Process Grid	7x4 Physical	7x4 Xen VOC	Xen Overhead	7x4 KVM VOC	KVM Overhead
Problem Size	58600	58600	0%	58600	0%
G-HPL (GFLOPS)	169.807	118.067	30.470%	25.178	85.173%
G-PTRANS (GB/s)	0.867	0.496	42.818%	0.069	91.985%
G-Random Access (GUP/s)	0.014	0.009	35.910%	0.004	73.082%
G-FFTE (GFLOPS)	2.287	1.717	24.899%	0.399	82.556%
EP-STREAM Sys (GB/s)	59.046	62.678	-6.151%	82.599	-39.889%
EP-STREAM Triad (GB/s)	1.845	1.959	-6.151%	2.581	-39.889%
EP-DGEMM (GFLOPS)	8.271	7.669	7.269%	6.901	16.559%
RandomRing Bandwidth (GB/s)	0.023	0.017	23.425%	0.007	67.419%
RandomRing Latency (μs)	74.444	150.831	102.611%	290.463	290.179%

Table 4: Physical vs. VOC, Two VMs per Physical Node (32 processes)

Process Grid	7x4 Physical	7x4 Xen VOC	Xen Overhead	7x4 KVM VOC	KVM Overhead
Problem Size	58600	58600	0%	58600	0%
G-HPL (GFLOPS)	169.807	130.862	22.935%	81.401	52.063%
G-PTRANS (GB/s)	0.867	0.830	4.302%	0.447	44.968%
G-Random Access (GUP/s)	0.014	0.011	22.941%	0.004	70.643%
G-FFTE (GFLOPS)	2.287	0.746	67.380%	1.751	23.449%
EP-STREAM Sys (GB/s)	59.046	62.382	-5.650%	73.110	-23.818%
EP-STREAM Triad (GB/s)	1.845	1.949	-5.650%	2.285	-23.818%
EP-DGEMM (GFLOPS)	8.271	7.726	6.588%	7.114	13.979%
RandomRing Bandwidth (GB/s)	0.023	0.007	68.779%	0.027	-17.148%
RandomRing Latency (μs)	74.444	125.258	67.259%	228.383	206.787%

Figure 4: NB vs. Performance



6. CONCLUSIONS

Based on the preliminary results of our study, we can conclude that KVM is generally efficient when network I/O latency is not a factor, as demonstrated by the low single node overhead in HPL. We encountered some unusual results for STREAM on the full cluster, which exhibits low temporal locality and high spatial locality according to Luszczek *et. al.* [11]. We are still investigating why virtualization would improve performance in these situations.

As shown in Tables 3 and 4, network latency is poor (three-fold increase), resulting in large virtualization overhead. Based on prior MPI studies and our own HPL testing, the high latency of the virtual network causes the poor HPL performance [12, 11]. However, HPL provides a good diagnostic tool because it distributes blocks of a size specified by the NB parameter. Thus, the nature of its network traffic can be controlled to some degree. As Figure 4 shows, HPL performs better under high-latency conditions when the block size is increased. This performance improvement is due to the fact that fewer, larger transfers will be less affected by latency than many small transfers. HPL performance eventually drops off due to poor load balancing with greater block sizes. Future work in the feasibility of using a KVM-based cluster for High Performance Computing on the grid will be focused on reducing the virtual network latency. We see latency reduction as a crucial need to make virtual clusters a mainstream HPC technique.

While the loss in performance of inter-node communication with MPI is disappointing, these types of jobs are not common on the Open Science Grid. Therefore, KVM does appear well-suited to Condor jobs in the vanilla and standard universes, which OSG sites primarily utilize [17]. We believe that 8.7% is an acceptable performance overhead in these situations, given the benefits gained in terms of VO compute environment customization. Additional evaluation using scientific applications is underway.

Finally, KVM is a promising hypervisor for grid computing. It was easily deployed (compared to Xen) and is simple to maintain while still providing good performance for many Condor jobs. While there are some issues with virtual networking, our results demonstrate the viability of a KVM-based Virtual Organization Cluster.

7. REFERENCES

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

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Symposium on Operating Systems Principles (SOSP '03)*, 2003.

[3] R. Bisseling and L. Loyens. Towards peak parallel linpack performance on 400. *Supercomputer*, 45:20–27, 1991.

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

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

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 15(3):200–222, 2001.

[7] I. Habib. Virtualization with KVM. *Linux Journal*, 2008(166):8, February 2008.

[8] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall. The efficacy of live virtual machine migrations over the Internet. In *Second International Workshop on Virtualization Technology in Distributed Computing*, Reno, NV, November 2007.

[9] K. Keahey, K. Doering, and I. Foster. From sandbox to playground: Dynamic virtual environments in the Grid. In *5th International Workshop on Grid Computing (Grid 2004)*, November 2004.

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

[11] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, and D. Takahashi. The HPC Challenge (HPCC) benchmark suite. In *Supercomputing '06*, 2006.

[12] M. Matsuda, T. Kudoh, and 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.

[13] A. Matsunaga, M. Tsugawa, M. Zhao, L. Zhu, V. Sanjeepan, S. Adabala, R. Figueiredo, H. Lam, and J. A. Fortes. On the use of virtualization and service technologies to enable grid-computing. In *11th International Euro-Par Conference*, August 2005.

[14] A. M. Matsunaga, M. O. Tsugawa, S. Adabala, R. J. Figueiredo, H. Lam, and J. A. B. Fortes. Science gateways made easy: the In-VIGO approach. *Concurrency and Computation: Practice and Experience*, 19(6):905–919, April 2007.

[15] B. Quetier, V. Neri, and F. Cappello. Selecting a virtualization system for Grid/P2P large scale emulation. In *Proceedings of the Workshop on Experimental Grid Testbeds for the Assessment of Large-scale Distributed Applications and Tools (EXPGRID'06)*, Paris, France, June 2006.

[16] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *Proceedings of the Third Virtual Machine Research and Technology Symposium*, San Jose, CA, May 2004.

[17] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. C. Fox, and A. J. G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 11, pages 299–350. Wiley, 2003.

[18] L. van Doorn. Hardware virtualization trends. In *Second International Conference on Virtual Execution Environments*, June 2006.

[19] A. Whitaker, M. Shaw, and S. D. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. Technical Report 02-02-01, University of Washington, 2002.