An Evaluation of KVM for Use in Cloud Computing

Michael Fenn, Michael A. Murphy[*], Jim Martin, and Sebastien Goasguen
School of Computing
Clemson University
Clemson, South Carolina 29634-0974 USA
{mfenn,mamurph,jim.martin,sebgoa}@cs.clemson.edu

Abstract

In this paper we describe a virtual cluster based on the Kernel-based Virtual Machine (KVM) as an alternative to VMWare and Xen. Specifically we show how the virtual cluster is built and tailored to fit virtual organizations. The technique presented in this paper, known as the Virtual Organization Cluster Model, shows great potential for cloud computing. In our implementation, we used a minimalist installation of Slackware Linux 12 on 14 compute nodes, ensuring minimal host overhead. Our prototype Virtual Organization Cluster is composed of 28 virtual computes nodes, each running CentOS 5.1 with Condor, and MPI. Results of testing the prototype were encouraging, with the exception of network performance. Single node virtualization penalty was measured at 8.4% while 28 nodes High Performance Linpack benchmark runs incurred a 75-78% penalty.

1. Introduction

A current problem in scientific computing is that the expertise needed to deploy and maintain a cluster remains scarce despite recent declines in hardware costs. Smaller research groups may not be able to afford to have their own clusters, and purchasing time on an existing cluster raises concerns such as vendor lock-in and data security.

In this paper, we present a cloud computing model which defines a minimum specification for a compute cluster. This cluster's sole purpose is to host other compute clusters through virtualization. In this way, a Virtualization Service Provider (VSP) can sell compute power without having to directly maintain each end-user's particular application.

Similarly, Virtual Organizations (VO's) can purchase compute power from VSP's without having to worry about hardware or software compatibility. A VO is free to develop a model cluster locally, perhaps even on a personal workstation, test it, and then deploy it to a VSP's hardware with reasonable assurances that the operating environment will be fully compatible.

We will first provide a brief overview of virtualization technologies, followed by a description of our model virtual cluster. Then, we will define the infrastructure for which a VSP would be responsible. Finally, we will present some results of a model cluster constructed at the Cyberinfrastructure Research Laboratory at Clemson University.

2. Virtual Machine Model

In essence, machine virtualization is making one computer appear to be multiple computers [3]. Machine virtualization is accomplished with a program called a *hypervisor,* while systems running under a hypervisor are known as *virtual machines* (VMs). There are two basic types of hypervisor [2]:

- *Type 1* hypervisors directly interface with the system hardware. All operating systems run inside a virtual machine. There is usually a special, privileged virtual machine that can manage the others. Xen is an example of this type of hypervisor.
- *Type 2* hypervisors run as a normal program inside a normal operating system. This OS is known as the *host*. Each *guest* OS runs as a process in the host OS. These processes can be manipulated just like any other process. VMWare and KVM are examples of this type of hypervisor.

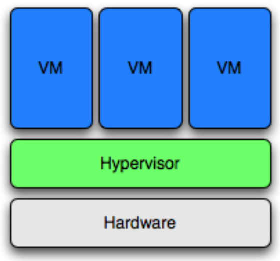
See Figure 1 for a comparison of Type 1 and 2 hypervisors.

A strict Type 2 hypervisor requires that all I/O devices be emulated completely in software, resulting in added overhead for I/O calls. *Paravirtualization* allows the virtual machine to make calls directly to the hypervisor, resulting in potentially increased efficiency. Paravirtualization requires modifications to the guest kernel [2].

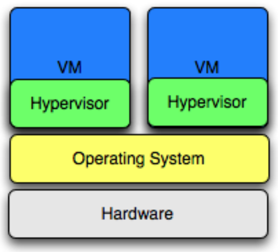
See Table 1 for a comparison of KVM and Xen.

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

Figure 1. Hypervisors



(a) Type 1



(b) Type 2

Table 1. KVM vs. Xen

KVM	Xen
Type 1 Hypervisor	Type 2 Hypervisor
Host is a privileged guest	Host directly on hardware
Unprivileged guests	Guests have user privileges
x86 ring abstraction	UNIX process abstraction
Paravirtualized guests	Unmodified guests

2.1. Kernel-based Virtual Machine (KVM)

The Kernel-based Virtual Machine (KVM) is a Type 2 hypervisor maintained by Qumranet, Inc [1][5]. KVM is based on the QEMU emulator and derives all its management tools from QEMU. The main focus of KVM development is to use the x86 VT extensions, which allow virtual machines to make system calls [7]. KVM versions newer than KVM-62 have support for paravirtualized Linux guests, but we did not utilize this capability in our initial prototype.

KVM uses a set of Linux kernel modules to provide VT support. KVM can run on a stock Linux kernel that is: (a) new enough and (b) has had the KVM modules built for it. In contrast, Xen requires a heavily patched Linux kernel, on which development lags behind the mainline kernel.

KVM supports the QEMU Copy-on-write (QCOW) disk image format, allowing it to support a "snapshot" mode for its disk I/O operations. In "snapshot" mode, all disk writes are directed to a temporary file, and changes are *not* persisted to the original disk image file. Multiple VM's can be run from one disk image, somewhat mitigating the huge storage requirements associated with hosting a grid of VM's[4]. Destroying a virtual cluster is as simple as sending SIGKILL to each hypervisor and deleting the image from disk.

KVM supports the standard Linux TUN/TAP model for Ethernet bridging. By using this model, each VM gets its own networking resources, making it indistinguishable from a physical machine.

2.2. Virtual Compute Nodes

Central to the Virtual Organization Cluster Model (VOCM) is the Virtual Organization Cluster (VOC), which is composed of Virtual Compute Nodes (VCN). Each Virtual Organization (VO) that wishes to utilize the compute facilities provided by a Virtualization Service Provider (VSP) must provide a VM image or set of VM images, along with some general configuration parameters. Since each image will potentially be used to spawn multiple VM's, the configuration of each image must not make any assumptions about the type of networking (hardware interface), hostname, or system-specific configuration settings. Instead, dynamic networking configuration should be used. Once a hostname has been obtained, dynamic configuration based upon the hostname is allowed.

Our model VOC was built from two VCN's, each with CentOS 5.1 installed. CentOS provides substantial "out-of-the-box" support for cluster computing applications and, along with its cousin, Red Hat Enterprise Linux, is widely supported in the scientific and high-performance computing communities. The two VCN's were:

1. A virtual head node, which was configured with the Condor central manager and submit daemons (condor_collector, condor_negotiator, condor_schedd), Ganglia monitoring daemon (gmond), and Ganglia meta daemon (gmetad).

2. A virtual compute element, which was configured with the Condor job starter (condor_startd), MPICH2, ATLAS (tuned for the virtual CPU), and Ganglia monitoring daemon (gmond).

Our model VOC was designed as an Open Science Grid (OSG) compute element (http://www.opensciencegrid.org). The virtual head node used Condor to distribute incoming OSG jobs to the virtual compute elements.

3. Support Model

Preparing the physical (as opposed to virtual) cluster for VOC support required configuring the host OS, setting up support services, configuring networking services, and configuring storage services. In our prototype implementation, support services included a Lightweight Directory Access Protocol (LDAP) server for centralized administration of hosts and physical user accounts, a Dynamic Host Configuration Protocol (DHCP) server for assigning IPv4 addresses to nodes, and a Domain Name Server (DNS) for host resolution.

3.1. Host OS configuration

When providing virtualization services, the host OS should be minimalist in order to reserve as many resources as possible for the VCNs. To this end, Slackware Linux 12 was chosen as the host OS. A custom kernel was compiled to enable support for KVM and additional network metrics. All unnecessary hardware drivers and other features were left out of the kernel to minimize its memory footprint. KVM driver modules also were built for the custom kernel.

For maintainability reasons, all the Slackware nodes were installed via PXE boot and a custom automated install script. This allowed the whole cluster to be re-created quickly in case of added nodes, hardware failure, or administrator error. All additional software was maintained in the form of Slackware packages to allow for rapid deployment across the entire cluster.

3.2. Physical Support Services

Configuration information for each VCN was stored in an LDAP database to provide a centralized administration mechanism. Each VCN was represented as an LDAP entry with the hostname, IP address, and MAC address fields. The MAC address was generated as a locally-administered address as to avoid conflicts with any other MAC's on the network. An LDAP-aware, batch, remote administration tool was also written to aid the systems administrator. To assist in the dynamic configuration of VCN's, the physical cluster provided DHCP and DNS services to the guests. In order to maintain a single configuration source, DHCP and DNS configuration files were generated from the LDAP database by a custom utility. These utilities were required to be run whenever the host information in LDAP is updated. All VCN images were maintained on an NFS export that was mounted on each physical compute node. These images were accessed in a read-only mode since KVM's snapshot mode was employed to start several VM's from the same image file. Any writes to the virtual disk were made to a temporary file on each physical compute node. When KVM exited, these files were deleted.

4. Results

Two different tests were performed: the standard supercomputing High Performance Linpack benchmark and measuring the VCN boot times.

4.1. High Performance Linpack (HPL)

The following HPL parameters were optimized for our cluster and remained constant throughout these tests: block size (NB), process mapping (PMAP), threshold, panel fact (PFACT), recursive stopping criterium (NBMIN), panels in recursion (NDIV's), recursive panel fact. (RFACT's), broadcast (BCAST), lookahead depth (DEPTH), SWAP, swapping threshold, L1 form , U form, Equilibration, and memory alignment. The problem size (N) was derived with the formula

$$N = \sqrt{nDU}$$

where n was the number of nodes tested, D was the number of doubles 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 (80% is a good rule of thumb). All tests were run with ATLAS 3.8.1 (tuned separately for physical nodes and VCN's) and MPICH2 1.0.5p4.

Table 2 is a comparison of low-overhead Slackware Linux on the physical nodes to CentOS. These tests were run with 56 GiB of memory over 14 nodes.

Table 3 is a comparison of HPL run on the physical nodes to HPL run on the VOC. These tests were run with 28 GiB of memory over 14 physical nodes and 28 VCNs. Since each physical node is dual-core, two VCNs were run on each.

Table 2. Slackware vs. CentOS 5.1

Process Grid (PxQ)	14x2
Problem Size	77000
CentOS GFLOPS	115.6
Slackware GFLOPS	129.6
Slackware Advantage	12.11%

Table 3. Bare-Metal vs. VOC

Process Grid (PxQ)	1x1	14x2	7x4
Problem Size	10300	54800	54800
Physical GFLOPS	6.821	111.6	138.2
VOC GFLOPS	6.248	24.41	33.60
Virtualization Penalty	8.401%	78.13%	75.69%

4.2. Boot Times

Booting the physical hardware was accomplished in under three minutes for each node. The physical head node required 79 seconds from power-on to completion of the boot process. Of this time, the first 43 seconds were occupied by the Power-On Self Test (POST) procedure and menu time-outs present for human interaction. Each physical compute node booted in a range of time from 160 seconds to 163 seconds. Of this time, up to 117 seconds were occupied by the POST procedure, menu time-outs, and a PXE boot time-out, leaving approximately 45 seconds for the actual Linux boot procedure. Some variation in recorded times was likely due to human error in timing the boot procedure, which was done by means of a stopwatch. After the machines were booted, approximately 75 processes were found to be running on the physical head node, with approximately 65 processes running on each physical compute node.

The 29 CentOS virtual machines (28 compute nodes and 1 head node) booted in an average of 58 seconds from initiation of the KVM process. A minimum boot time of 52 seconds, and a maximum boot time of 63 seconds, were observed. Approximately 5 seconds can be attributed to a boot-loader time-out in the virtual machine configuration, giving an average Linux boot procedure time of 47 seconds. Following the boot procedure, approximately 100 processes were found to be running on the virtual head node, with approximately 75 processes running on each VCN.

5. Conclusions

Our choice of a low-overhead host OS seemed to be well-supported by the 12.11% (Table 2) speed advantage of Slackware Linux 12 over CentOS 5.1 on *identical* hardware. Since the Slackware system had such a small memory footprint, larger problem sizes could be used while still avoiding swapping.

Boot times for virtual machines were comparable to the actual post-boot-loader-time-out portion of the physical machine boot processes, even though more processes were started in the VM's at boot time. Since there were no hardware time-outs present in the VM boot procedure, the effective wall time for booting a VCN was substantially less than the time required for booting a physical compute node. Furthermore, the similar times for the actual Linux boot processes indicates that KVM disk I/O overhead is negligable in the context of basic system operations. Thus, performing maintenance tasks that require rebooting the system should result in less downtime for a Virtual Organization Cluster when compared to an equivalent physical cluster.

As shown in Table 3, KVM was efficient when network I/O is not a factor. Unfortunately, poor network performance for supercomputing applications can be expected, as shown by the large (75-78%) overheads present in HPL runs. Additional testing will be needed to determine the cause of this performance loss and resolve the underlying issue.

Poor network performance could be attributed to several factors, which will require more testing to differentiate:

- Since the Spanning Tree Protocol (STP) was not enabled on our TUN/TAP bridges, routing loops might have been present, introducing a large amount of network latency
- Each VCN on a physical node had a 1Gbps link to the bridge, but each physical node only has 1Gbps to the switch, collisions might have occurred. Such collisions could have caused binary-exponential back-off, thus crippling latency and bandwidth under heavy load.
- The code for the emulated Gigabit Ethernet NIC in QEMU was a recent addition at the time of this experiment, and it could have been incomplete.

The loss in performance of network-sensitive jobs represents only a small fraction of the total number of jobs on the Open Science Grid. This makes our model particularly well suited to Condor jobs in the "vanilla" and "standard" universes. These jobs are computationally expensive, but only rely on network I/O to get their initial state and return their results. So, even though Condor has MPI capability, the low performance of our actual prototype is not really a limiting factor because most sites only provide support for the vanilla and standard universes [6].

The Virtual Organization Cluster Model provides great potential for cloud computing, since it defines a minimum specification for VOC support. A VO is not tied to a particular site for its computing resources but is instead free to move between sites that support the VOCM. This migration could be as simple as bringing down the VOC, copying two disk images and a small configuration file, and booting the VOC on a completely different site.

References

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

[2] IBM. Ibm systems virtualization, December 2005.

[3] M. T. Jones. Virtual linux. Technical report, IBM, December 2006.

[4] K. Keahy, 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.

[5] Qumranet. Kvm - kernel-based virtualiztion machine white paper, 2006.

[6] D. Thain, T. Tannenbaum, and M. Livny. *Condor and the Grid*, chapter 11, pages 299–350. Wiley, 2003.

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