Abstract

The heterogeneity of resources in computational grids, such as the Canadian GridX1, makes application deployment a difficult task. Virtual machine environments promise to simplify this task by homogenizing the execution environment across the grid. One such environment, Xen, has been demonstrated to be a highly performing virtual machine monitor. In this work, we evaluate the applicability of Xen to scientific computational grids. We verify the functionality and performance of Xen, focusing on the execution of software relevant to the LHC community. A variety of production deployment strategies are developed and tested. In particular, we compare the execution of job-specific and generic VM images on grids of conventional Linux clusters as well as virtual clusters.

1 Report Specification

1.1 Audience

This paper is intended for my supervisors, current and future co-op students working in the High Energy Physics Group, and for the attendees of the Computing and High Energy Physics 2006 conference in Mumbai, India, in a slightly modified format.

1.2 Prerequisites

Readers of this paper should have a basic understanding of virtual machines, as well as a good background in general computing knowledge. A knowledge of high energy physics and the application of high throughput computing to this field is useful, but not essential.

1.3 Purpose

This paper summarizes and reports on my work during the Fall 2005 term in the Department of Physics and Astronomy at the University of Victoria. It contains an in-depth investigation of my major project during the term, which involved benchmarking the Xen virtual machine environment for use in high energy physics computing projects.

2 Introduction

With the increasingly wide deployment of particle physics applications on grids around the world, heterogeneity of computing environments is becoming a problem. Many applications, such as the ATLAS simulation code, are designed to run on a specific operating system (OS), and often a specific version of that OS. This prevents them from harnessing the computing power at grid sites that run different OSs. We have encountered this problem on the Canadian GridX1 (see Appendix A), where all sites do not run the same OS. This means that jobs cannot take full advantage of the resources, but instead must be sent to sites with a compatible OS. A solution to this problem is to use virtual machine (VM) technology to create a homogeneous environment across the grid for individual applications.

VM technology allows a machine running a base OS to divide up the physical system resources (i.e. memory, disk, CPU cycles) among itself and one or more ”virtual” machines. The base OS controls the VM’s access to the physical hardware and prevents them from interfering with each other. The VMs each have their own “slice” of memory and “disk”, and are allowed to use the CPU in a shared manner, just as any other multitasking system. The flavour of the base OS does not matter to software running in the VMs. Each VM appears to be a real machine to users.

This paper discusses results using one specific VM environment, an open source software (OSS) project called Xen [3], within a Condor-C grid. Most VM technologies cause a dramatic decrease in performance [4] over the physical machine due to the extra layer of software running in the base OS to enable the virtualization. We will show that Xen avoids this problem in a series of comprehensive benchmarks.

Other groups have also put considerable effort into benchmarking Xen’s performance. In ”The Art of Virtualization”, Barham et al. [6] compared native Linux, Xen, VMWare, and User Mode Linux. And in ”Xen and the Art of Repeated Research”, Clark et al. [5] repeated the first group’s tests and confirmed their findings to a large degree. However, neither of these experiments have specifically addressed Xen’s performance for HEP applications.

In Section 2, we will discuss HEP-relevant Xen benchmarks and the implications they have for continued use of Xen in this environment. In Section 3, we will lay out three different methods of integrating Xen into a Grid environment, and show which one is the best. Finally, in Section 4 we will review our findings and conclude.

3 Xen for HEP Applications

Xen uses a technique called “paravirtualization” to maximize performance. Unlike other virtual machine monitors such as VMWare, which does full virtualization, Xen only intercepts “priviledged” instructions. This means that many of the operations run at full native speed, thereby allowing Xen to perform better than other VMs. However, the guest OSs must be modified.

Xen runs as a process in the base OS, called “domain 0” or “dom0”. It requires a kernel patched with the Xen

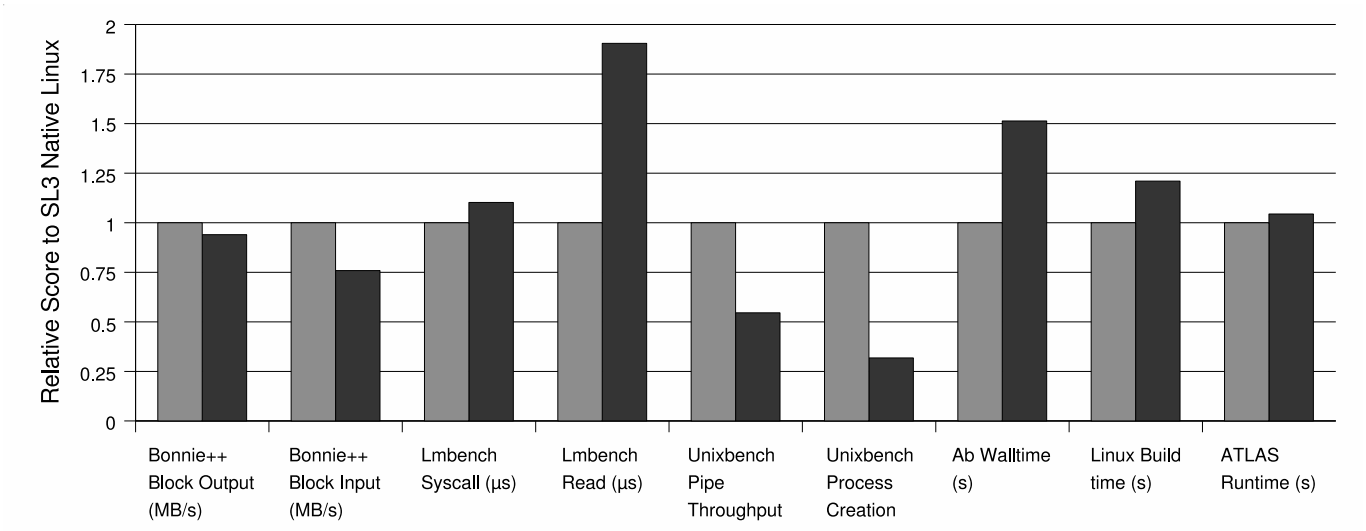

Figure 1: Relative Performance of native Linux (grey) and Xen Linux (black)

source code to be installed. The guest OS also needs to run a modified kernel, and is called a "domain U" or "domU". DomUs are controlled via an administration tool in dom0, and can be set up to have virtual network interfaces to allow them to communicate over Ethernet.

3.1 Benchmark Suites

We ran each of six benchmark testsuites on both a native Scientific Linux 3 install, and the same install running as a guest image (domU) under Xen. The first three are synthetic tests [1], focussing on the performance of specific operations within the system. They are bonnie++, which tests I/O to memory and disk; UnixBench, which tests process creation, filesystem performance, and system call throughput; and Lmbench, which tests interprocess communication (IPC), system call throughput, and filesystem, I/O, and network performance. The last three were chosen to provide a realistic view of performance that was relevant to the HEP community. They include Ab, which provided a realistic view of the performance of an application that was heavy on process creation and context switching; a Linux Kernel build, which tests moderately CPU-intensive application performance; and finally, the ATLAS KitValidation (version 10.0.0)suite, to test the performance of a standard HEP application.

Bonnie++, Ab, the kernel build, and ATLAS were run three times on each system to generate the results. UnixBench was run twice and showed little variation in results; Lmbench once. All tests were performed on the following setup:

[1] A synthetic benchmark tests a component of the system, whereas application benchmarks test the overall system's performance.

CPU:	Athlon XP 2500+ (1826.005 MHz)
RAM:	Limited to 256MB[2]
Disk:	Maxtor 6B200P0, ATA DISK drive
Motherboard:	ASUS A7VBX-MX SE
Network:	Tested only loopback interface.
Domain-0 OS:	Fedora Core 4
Domain-U OS:	Scientific Linux 3.0.4
Xen version:	2.0.7

3.2 Results

We have included selected graphs (see Figure 1) to illustrate our findings. The first two show output and input, respectively, from Bonnie++. As one can see, SL3-Xen's performance in writing to the disk is quite close to native SL3, but even reading from the disk, Xen is only 25% slower. I/O in general is quite good under Xen, especially when to and from memory, since the Xen virtualization layer does some buffering which causes it to appear that Xen is actually faster in that case.

The next two results in Figure 1 are generated from Lmbench results, and this is where we found the weakness in Xen. All integer and floating point operations performed exactly the same under SL3-Xen as under native SL3, but system calls that required the OS to switch into priviledged mode caused a performance hit of up to 50%. Looking at "Lmbench syscall" in Figure 1, one can see that a simple system call (one that did not require the OS to switch kernel modes) is approzimately the same in both systems. In "Lmbench Read", however, a simple read system call (that did cause the OS to switch) took nearly twice as long. This is due to the extra code needed to perform the virtualization. This effect is apparent in all operations that require the OS

to switch to kernel mode: system calls including read, write, stat, fstat, open/close, pagefaults, process creation/forking, and creating and using pipes.

We confirmed this effect using UnixBench, which contributed the next two results in Figure 1. "Unixbench Pipe Throughput" in Figure 1 confirms the 50% decrease in performance under SL3-Xen for kernel-mode tasks. "Unixbench Process Creation" shows an especially bad performance hit, with SL3-Xen turning in roughly 25% of native SL3's throughput.

Moving on to the application-level benchmarks, starting with "Ab Walltime" in Figure 1, we can see that Ab took 1.5 times as long under SL3-Xen. However, one must keep in mind that Ab requires lots of context switches in the course of operations such as process creation which cause a larger performance decrease under Xen. This is clearly illustrated in Figure 1, "Linux Build Time". Sl3-Xen, compiling the same kernel, didn't even take 1.25 times as long. And finally, in Figure 1, "ATLAS Runtime", one can see that ATLAS performs quite well under Xen. This is because the majority of the operations it performs are arithmetic and have no performance hit under Xen.

Application performance under Xen depends on the profile of the application. One that creates many processes [3], such as Ab, or uses lots of pipes, will demonstrate an slowdown of up to 50%. However, one that performs mainly arithmetic operations, such as ATLAS, will run at near native speed.

3.3 Practical Xen Issues

We felt it necessary to include a couple of practical issues that arose in the course of testing Xen as they may be of interest to others in the community attempting to replicate this work.

First, since the HEP community frequently uses Redhat Enterprise Linux 3 (RHEL3) or derivatives, we felt it would be useful to include information on how to install Xen on such a system. Although Scientific Linux 3.0.4 worked as a domU, we found it necessary to install SL3.0.5 as a dom0 instead of earlier versions due to Python versioning issues. We compiled Xen from source after having problems with pre-compiled RPMs.

Second, the security of the domUs may prove to be an issue. Although they are protected from each other, there does not currently exist any security in dom0 to prevent unauthorized users from access, modifying, and shutting down the domUs. Any user that has access to the Xen administration tools has essentially "physical" access to the virtual machines. Since Xen is still beta software, we hope that this will be addressed in future releases.

[3] defined to be a substantial proportion of the total operations the program performs.

A stopgap measure to ensure some level of security in dom0 can be instituted as follows: if the Xen administration tools are installed in dom0 world-executable, then anyone who has access to that machine can start, stop, and access the domUs running on that machine. This can be prevented one of two ways: first, user accounts can be restricted to only trusted users. Second, user accounts to dom0 can be given out freely. By creating a 'xen' group, chowning all xen administration tools to that group, and then placing only trusted users into this group, one can make reasonably secure the Xen domUs.

However, a savvy user could still compile her own Xen admin tools and use those to "break in" to the domUs. Clearly, this security issue needs to be addressed before deploying Xen in a production grid environment.

4 Xen in a Grid Environment

We will now examine three methods for integrating Xen into a grid environment, starting with a description of our testbed setup.

1. The user prepares a complete guest OS image containing the application and input files, uploads the image to a job repository, and submits a script to fetch and start a Xen domU.

2. The user does not prepare a custom image, just a job script, which is uploaded to a job repository. The execute machine then starts a generic Xen image, which in turn fetches the job and runs it.

3. Each physical machine in the cluster runs one or more persistent Xen domUs. Each domU would have its own grid certificate, and appear to the world as a permanent machine. In this way, a cluster running one OS could masquerade as another "virtual" cluster running a different OS.

5 Conclusions

5.1 Technical

Xen represents a promising solution to the problem of heterogeneity among compiled applications, especially since it is slated to be included in an upcoming release of the main Linux kernel. Our benchmarks have shown that HEP applications can run satisfactorily under Xen, and, furthermore, that Xen will run on an operating system commonly used in HEP computing environments.

5.2 Comments on Work Term

I found this workterm to be an excellent learning experience, both technically and otherwise. Virtual machines were not something I was familiar with when starting this job, but I was able to gain a good background understanding while working by reading papers and documentation, and gained specific expertise with Xen through my work with it. I also had an opportunity to refresh my Plone content management skills by doing small maintenance tasks to the group website, which is hosted in Zope/Plone.

The only suggestion I have is that there were times when I felt that I didn't have enough work; sometimes it is helpful to have two or three tasks to switch back and forth between when one comes to a roadblock. Also, the workload toward the end of the work term seemed much heavier than at the beginning or middle of the term. But these are relatively minor, and other students may not have the same work habits.

I enjoyed working in a research environment, and discovered how it is different compared to a corporate or government position. In conclusion, this was a thoroughly rewarding co-op work term, and I would definitely recommend this job to other students.

A GridX1

GridX1 is a pan-Canadian grid of computational resources provided by shared facilities at a number of Canadian institutions. GridX1 resources include Linux-based PBS and Condor clusters at the Centre for Subatomic Research at the University of Alberta, the Research Computing Centre at the University of Victoria, the WestGrid cluster at the University of British Columbia, the Research Computing Support Group at the National Research Centre in Ottawa, and the BigMac cluster at the University of Toronto High Energy Physics Group. While the combined resources amount to approximately 2500 processors and over 100 TB of storage, GridX1 users are given access to only a fraction of these resources. All sites have gigabit network connectivity to the national research network provided by CANARIE. GridX1 has been deployed using version 2 of the Globus Toolkit [1] distributed in the Virtual Data Toolkit [2]. This version of the middleware is popular with production grid deployments due to its maturity, which has resulted in a relatively stable deployment platform. In addition to the basic middleware, a number of GridX1-specific tools have been developed to manage users and monitor tasks and resources. GridX1 utilizes the Condor-G resource management system to advertise resources and schedule jobs. The Condor-G scheduler has proven to be quite effective in its stability and speed. Through an interface to the LHC Compute Grid, GridX1 has successfully executed over 20 000 jobs for the LHC project.

References

[1] The Globus Tookit, see http://www.globus.org/toolkit/

[2] VDT, The Virtual Data Toolkit, see http://vdt.cs.wisc.edu/index.html

[3] The Xen virtual machine monitor, see http://www.cl.cam.ac.uk/Research/SRG/netos/xen/

[4] The Xen virtual machine monitor: Performance, see http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html

[5] Xen and the Art of Repeated Research, see http://www.clarkson.edu/class/cs644/xen/files/repeatedxen-usenix04.pdf

[6]

[7] Bonnie++, see http://www.coker.com.au/bonnie++/

[8] UnixBench, see http://www.tux.org/pub/tux/benchmarks/System/uni

[9] LMbench - Tools for Performance Analysis, see www.bitmover.com/Lmbench/

[10] Apache HTTP Server Project, see http://httpd.apache.org/

[11] Linux Kernel Archives, see http://www.kernel.org

[12] Condor: High Throughput Computing, see http://www.cs.wisc.edu/condor/

[13] Condor v. 6.7 Manual: Condor-C, see http://www.cs.wisc.edu/condor/manual/v6.7/5_3Grid_Universe.html