Evaluation of Virtual Machines for HEP Grids

A. Agarwal, R. Desmarais, I. Gable, A. Norton, R. Sobie, D. Vanderster
Department of Physics, University of Victoria
Victoria, BC, Canada

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.

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 OS's. We have encountered this problem on the Canadian GridX1, 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 VM's 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 VM's. Each VM appears to be a real machine to users.

This paper discusses results obtained using a specific VM environment, an open source software project called Xen [1], within a Condor [2] grid. Most VM technologies cause a dramatic decrease in performance over the physical machine due to the extra layer of software running on the base OS to enable the virtualization. We show that Xen avoids this problem in a series of comprehensive benchmarks. Additionally, we discuss strategies for deploying Xen in a production grid environment such as GridX1.

Other groups have also put considerable effort into benchmarking Xen's performance. In "The Art of Virtualization", Barham et al. [1] compared native Linux [3], Xen, VMWare [4], and User Mode Linux [5]. In "Xen and the Art of Repeated Research", Clark et al. [6] 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.

The remainder of this paper is organized as follows. In the next section, we discuss HEP-relevant Xen benchmarks and the implications they have for continued use of Xen in this environment. In the following section, we evaluate different methods of integrating Xen into a Grid environment, and make some recommendations for future investigation. Finally, we review our findings and conclude.

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 "privileged" instructions. The result of this technique is that many of the operations run at full native speed, thereby allowing Xen to perform better than other VM's. However, the performance improvement comes at an administrative cost – guest OS's are required to be modified to execute in the Xen environment.

Xen runs as a process in the base OS, called "domain 0" (or "dom0"). It requires a Linux kernel patched with the Xen source code to be installed. The guest OS also needs to run a modified kernel, and is called a "domain U" (or "domU"). DomU's are controlled via an administration tool in dom0, and can be configured to have virtual network interfaces to allow them to communicate over Ethernet.

Benchmark Suites

We ran each of six benchmark test suites 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], focusing on the performance of specific operations within the system. They include bonnie++ [7], which tests read and write performance to memory

[1] A synthetic (or micro-) benchmark tests a component of a system, whereas an application benchmark tests many components of the system to evaluate the overall system performance.

Table 1: System setup for Xen benchmarking

CPU:	Athlon XP 2500+ (1826.005 MHz)
RAM:	ATLAS: 850MB, other: 256MB
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

and disk; UnixBench [8], which tests process creation, file system performance, and system call throughput; and Lmbench [9], which tests interprocess communication (IPC), system call throughput, and file system, I/O, and network performance. The final three benchmarks were chosen to provide an overall view of system performance. They include Ab [10], which provided a realistic view of the performance of an application that was heavy on process creation and context switching; a Linux Kernel build [3], which tests moderately CPU-intensive application performance; and finally, the ATLAS KitValidation suite (version 10.0.0) [11], to test the performance of a standard HEP application.

The results obtained for Bonnie++, Ab, Linux build, and ATLAS are the average of three executions. UnixBench was executed twice and showed little variation in its results. Lmbench was executed a single instance. All tests were performed using the setup shown in Table 1.

Results

In order to best illustrate our findings, we have included a selection of the benchmark results in Figure 1. All results are normalized to the native Linux performance. The first two results show disk block output and input as measured by Bonnie++. As we can see, SL3-Xen's performance in writing to the disk is quite close to native SL3; when reading from the disk, Xen is only 25% slower. I/O in general is quite efficient under Xen. This is especially apparent when working with memory, in which case Xen is actually faster than native Linux (resulting from I/O buffering in the virtualization layer).

The next two results in Figure 1, generated from Lmbench results, demonstrate an observed weakness in Xen. All integer and floating point operations performed nearly identically under SL3-Xen and native SL3. However, system calls that required the OS to switch into privileged mode caused a performance hit of up to 50%. Looking at "Lmbench syscall" in Figure 1, we see that a simple system call (one that did not require the OS to switch kernel modes) is approximately the same in both systems. In "Lmbench Read", however, a read system call (which causes the OS to switch) took nearly twice as long. This is due to the extra instructions needed to perform the virtualization. This effect is apparent in all operations that require the OS to switch to kernel mode: namely the 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" 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.

The overall efficiency of Xen is demonstrated by the application benchmarks. Starting with "Ab Walltime" in Figure 1, we see that Ab took 1.5 times as long under SL3-Xen. Ab is however an atypical application in that it relies on rapid process creation, which results in a large number of context switches. An example of source code compilation is illustrated in "Linux Build Time". When compiling the Linux kernel, SL3-Xen requires less than 25% more time than native SL3. Finally, "ATLAS Runtime" demonstrates that ATLAS performs quite well under Xen. Because the majority of the operations it performs are arithmetic, the ATLAS application suffers a negligible performance hit.

In summary, an application's performance under Xen depends on its reliance on system calls and process creation versus disk or memory I/O and arithmetic operations. Applications in the former category, such as Ab, will demonstrate an slowdown of up to 50%. Simulation applications such as ATLAS run at near native speed. In the context of HEP computing, where single processes are dominant, it is clear that Xen may be utilized without a performance penalty.

Practical Xen Issues

During the course of preparing this study, we encountered a number of practical issues that may be of interest to others in the community.

First, since the HEP community commonly uses Redhat Enterprise Linux 3 (RHEL3), or its derivatives, we felt it would be useful to comment on how to install Xen on such a system. Scientific Linux version 3.0.4 works smoothly as a domU with little modification. However, in order to use SL3 as a dom0, we found it necessary to upgrade to SL3.0.5 due to Python versioning issues. Additionally, we compiled Xen from source after having problems with the precompiled packages. Users requiring a rapid evaluation of Xen are recommended to deploy Fedora Core 4. This version of the OS provides precompiled and functional Xen kernels and utilities.

Second, a review of the domU security brought up important issues. Although domU's are protected from each other, there is not currently a mechanism in dom0 to prevent Xen users from accessing, modifying, and shutting down arbitrary domU's. Any user that has access to the Xen administration tools has essentially physical access to all of the virtual machines. A stopgap measure to ensure some level of security in dom0 can be instituted by restrict-

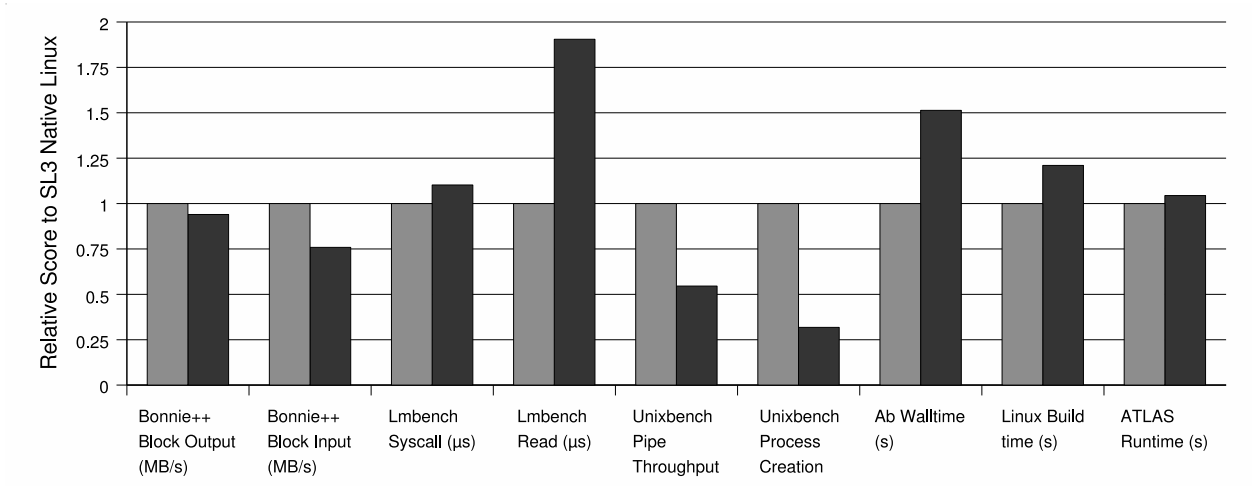

Figure 1: Relative Performance of native Linux (grey) and Xen Linux (black)

ing access to the Xen administration tools to a trusted group of users. However, a savvy user may gain unauthorized access to the domU's by compiling a new set of Xen administration tools. Clearly, the security implications of this issue must be understood before deploying Xen in a production environment. Since Xen is still beta software, we hope that this will be addressed in future releases.

XEN IN A GRID ENVIRONMENT

Virtual machine environments such as Xen provide a practical solution to the problems caused by the heterogeneity of typical grid deployments. Many applications are limited by strict requirements regarding the platform, operating system, and configuration of the execution hosts. In this study, we investigated Xen as a solution to this problem as it is occurs on GridX1, a Canadian computational grid of Linux clusters.

GridX1: A Canadian Computational Grid

GridX1 [12] 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 [13] distributed in the Virtual Data Toolkit [14]. 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.

Xen Deployment Strategies

System designers have many options for deploying Xen in a production grid environment, such as GridX1. A common requirement of all deployment strategies is the installation and availability of Xen on the grid execution hosts. As described earlier, we have verified that it is possible to install Xen on Linux distributions the are popular in the HEP community, including Scientific Linux 3 and Fedora Core 4. The task of installing and testing Xen will soon disappear as it is expected to become a standard feature of most Linux distributions.

We now examine three methods for integrating Xen into a grid environment. The first, and least intrusive, method treats Xen-based jobs no differently than regular jobs on the grid. In this case, a user prepares a complete guest OS image containing the application and input files. The image is configured to start the application at boot time using the inittab. To execute the job, the user must submit a bootstrap script to the grid. The bootstrap script handles the tasks of retrieving the OS image from a user specified location, starting the Xen guest OS, and performing any cleanup after the job completes. While this method may be useful for one-off jobs, the overhead required in building and transferring a unique image for each job negates the usefulness of this strategy for large scale applications.

It is possible to eliminate the large data transfer overhead

seen in the first strategy by developing a set of generic application images. In this case, the user does not prepare a custom OS image, but instead relies on a precreated image that provides the basic application but lacks run-specific input data. As in the previous strategy, each job is initiated by a bootstrap script which retrieves and starts the guest OS image. Because the application specific images are predistributed across the grid, the data transfer overhead is minimized. Upon booting the OS image, a startup process retrieves a job script which is executed within the guest OS. The job script handles the tasks of a specific job, including the retrieval of input data, execution of the job, and exporting of the output data. The method presents a useful strategy for utilizing Xen in a grid. Aside from the predistribution of the application specific OS image, the method requires little overhead.

The third and final strategy is to deploy persistent virtual clusters. Using this technique a cluster running a nonstandard or incompatible OS could masquerade as another "virtual" cluster running a different OS. In this method, each of the physical worker-nodes in a cluster runs one or more persistent Xen domU's. The batch system running on the headnode of the cluster is configured with a separate queue which has access to the virtual worker-nodes. When a task is submitted to the grid, an OS requirement expression in the job description file ensures that the task is matched to an appropriate cluster queue. In the case of arrival at a virtual cluster queue, the task executes within a domU worker node using the same scripts that are used on native nodes. This strategy presents an attractive approach because it requires neither user knowledge of Xen nor custom task or application specific Xen images. However, because the domU's are persistent, this strategy demands dedicated processor and memory resources on each of the worker nodes.

Variations on the final two strategies may be used to execute LCG jobs on GridX1 resources that do not have the required OS or application software installed. Using the first strategy, LCG jobs submitted to the GridX1 interface may be stored in a job script repository. The interface then submits a bootstrap script which retrieves an appropriate image, retrieves the LCG job script, and executes the job. In the second strategy, the LCG jobs can be passed as-is to a site that is persistently running an appropriate domU.

CONCLUSIONS

Xen represents a promising solution to the problem of heterogeneity on computation grids. In a selection of synthetic and application benchmarks, we identified the characteristics of applications that perform well and poorly under Xen. Most relevant to the HEP community is the demonstration that the ATLAS KitValidation performs satisfactorily under Xen. We have also demonstrated that Xen is usable today on Linux distributions common in HEP computing environments; however, it is clear that widespread adoption will follow the inclusion of Xen in the standard distributions. Finally, we have introduced a set of Xen-grid deployment strategies. We have found that creating and delivering one-off task images is not practical. Application-specific images may avoid the image delivery overhead, and the deployment of persistent virtual clusters is also appealing. However, further practical experience is required to comment on the viability of the latter strategies.

REFERENCES

[1] 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*. Proceedings of the nineteenth ACM symposium on Operating systems principles, pp 164-177, Bolton Landing, NY, USA, 2003.

[2] M. Litzkow, M. Limey, and M. Mutka. *Condor - a hunter of idle workstations*. In Proc. 8th Intl Conf. on Distributed Computing Systems, pages 104-1 11,1988.

[3] Linux Kernel. http://www.kernel.org

[4] VMware - Virtualization Software. http://www.vmware.com/

[5] J. Dike. *A user-mode port on the Linux kernel*. Proceedings of the 2000 Linux Showcase and Conference, October 2000.

[6] B. Clark, T. Deshane, E. Dow, S Evanchik, M. Finlayson, J. Herne, and J.N. Matthews. *Xen and the art of repeated research*. In Proceedings of the Usenix annual technical conference, Freenix track, July 2004.

[7] R. Coker. The Bonnie++ benchmark. http://www.coker.com.au/bonnie++/.

[8] The UnixBench benchmark. http://www.tux.org/pub/tux/benchmarks/System/unixbench/

[9] L. McVoy and C. Staelin. *lmbench: Portable tools for performance analysis*. In Proceedings of the USENIX Annual Technical Conference, pages 279-294, Berkeley, Jan. 1996. Usenix Association.

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

[11] The ATLAS Experiment. http://atlas.web.cern.ch/Atlas/index.html

[12] GridX1 website. http://www.gridx1.ca/

[13] I. Foster and C. Kesselman. *Globus: A metacomputing infrastructure toolkit*. Intl. Journal of Supercomputing Applications, 11(2):115-128, 1997.

[14] The Virtual Data Toolkit. http://vdt.cs.wisc.edu/