2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing

Performance Measuring and Comparing of Virtual Machine Monitors[*]

Jianhua Che, Qinming He, Qinghua Gao, Dawei Huang
College of Computer Science, Zhejiang University, Hangzhou 310027, China.
{chejianhua,hqm,qhgao,Davidhuang}@zju.edu.cn

Abstract

With its growth and wide applications, virtualization has come through a revival in computer system community. Virtualization offers a lot of benefits including flexibility, security, ease to configuration and management, reduction of cost and so forth, but at the same time it also brings a certain degree of performance overhead. Furthermore, Virtual Machine Monitor(VMM) is the core component of virtual machine(VM) system and its effectiveness greatly impacts the performance of whole system. In this paper, we measure and analyze the performance of two open source virtual machine monitors-Xen and KVM using LINPACK, LMbench and IOzone, and provide a quantitative and qualitative comparison of both virtual machine monitors.

1. Introduction

Recently virtualization has gained more and more interest among computer system researchers because it not only offers logical decoupling between operating systems(OSes) and underlying hardware, but also wins popular uses in several scenarios such as server consolidation and share hosting[2, 13]. In virtualization systems, resource virtualization of underlying hardware and concurrent execution of virtual machines(VMs) are in charge of a software called Virtual Machine Monitor(VMM) or hypervisor. By creating the illusion of physical resources and platform APIs, virtual machine monitor enables simultaneously running of multiple virtual machines inside it. Virtual machine monitor can be achieved based on virtualization technologies such as full-virtualization[16, 9], para-virtualization[16, 14] and hardware-assisted virtualization[16, 15, 4], etc. Full-virtualization presents a faithful emulation of hardware interfaces to guest OSes which don't realize themselves living in virtual machines and requires no change to these guest OSes, for example, VMware[16], KVM[1], QEMU[6]. Para-virtualization just provides a virtual machine abstraction that is similar but not identical to underlying hardware, and hence guest OSes need some modifications for the sake of porting into virtual machines, like as Xen[14], Denali[3], User Mode Linux(UML)[8]. Hardware-assisted virtualization remedies the non-virtualization's limitation of traditional x86 platform by extending legacy processor's architecture, for instance, Intel's Vanderpool Technology[15] and AMD's Pacifica[4].

Virtualization serves flexibility, security, ease to configuration and management, reduction of Total Cost of Ownership(TCO) for many business systems[13, 5]. However, it also meanwhile incurs some overhead of performance. As one of the core components, virtual machine monitor will impact the performance of virtualization systems to a great extent. And further, the virtualization of CPU, memory and I/O operations are three main aspects of designing virtual machine monitors, so it's critical for exposing and improving the performance defects of virtualization systems to measure and compare their processing efficiency in various virtual machine monitors.

This paper provides a series of performance experiments on testing two virtual machine monitors available for the x86 platform-Xen3.1 and KVM-60 to validate their readiness for virtualization.The experimental results are discussed and reveal Xen3.1 holds a better performance than KVM-60. Specifically, the contributions of this paper consist in: 1) As for the virtualization of CPU and memory in Xen and KVM, we have measured the overhead of CPU virtualization using LINPACK and memory virtualization with LMbench, and compared the performance disparity of virtualized against native environment. 2) Concerning the virtualization of I/O operations in Xen and KVM, we have tested the different cases such as using single or dual CPUs and with or without Intel's VT, and proved the impact of hardware assistance to software virtualization. 3) We as well analyzed the underpinning capacity of Xen and KVM for Windows and Linux as a guest OS respectively.

The rest of this paper is organized as follows. We begin in Section 2 with related works. Then, we introduce the details for two different virtual machine monitors-Xen

[*]Research was supported by the State Key Development Program for Basic Research of China("973 project", No.2007CB310900).

and KVM in Section 3 and three benchmarks in Section 4. In Section 5, we describe the experimental setup and results evaluation of benchmarking Xen and KVM. Finally, we conclude with discussion in Section 6.

2. Related Works

With measurement and evaluation of virtual machine monitors, there has existed several works[2, 5, 14, 17, 18]. Barham et al.[**?**] gave a comprehensive introduction and several performance evaluations to Xen. The authors described virtual machine interfaces(VMIs) to CPU, memory, device I/O and the design and implementation of control and data transfer mechanism in Xen. By benchmarking Xen against VMware Workstation, ESX Server, UML and so on, they compared the system throughput with executing multiple applications on a single native OS against running every application in virtual machines, and then evaluated the performance isolation between guest OSes in Xen and the total overhead of running large number of OSes on the same hardware. Clark et al.[5] reproduced the results from [14] with almost identical hardware. Moreover, they compared Xen with native Linux on a less powerful PC and evaluated the ability of Xen as a platform for virtual web hosting. Menon et al.[2] described a system-wide statistical profiling toolkit-Xenoprof and its performance measuring and debugging on Xen with network applications.

It's mostly noted that VMware and XenSource made their respective experiments to test the performance of their products with the same benchmarks[17, 18]. In [17], VMware used PassMark, Netperf, SPECcpu2000, SPECjbb2005 and building SPECcpu2000 INT package as workloads to test VMware ESX Server3.0.1 and open source version 3.0.3 of Xen and indicated that ESX Server3.0.1 delivered the superior, production-ready performance and scalability needed for an efficient and responsive datacenter, but Xen3.0.3 could not do that. As a response, XenSource compared XenEnterprise3.2 based on Xen3.0.4 and enhancements packaged for Windows guests with ESX Server3.0.1. XenSource made clear that XenEnterprise3.2 performs just as well as ESX Server3.0.1 and in many cases better, although in a few tests with less performance being highlighted for improvement in later releases[18].

As with KVM, Larabel[12] made a comparison between KVM Linux 2.6.20-rc3 and Xen3.0.3 using Gzip compression, LAME compilation, LAME encoding, RAMspeed and stated that KVM obtained one-up behavior in Gzip compression but lower performance in other cases than Xen.

3. Virtual Machine Monitors

This section describes the details of two different virtual machine monitors-Xen and KVM. By introducing the accomplishment of their CPU, memory and I/O operation's virtualization, the details needed for analyzing the experimental results in Section 5 are depicted.

3.1. Xen

Xen is an open source hypervisor based on the x86 platform originally developed at the University of Cambridge[2, 14]. Xen uses para-virtualization that host OS needs to be explicitly ported to its virtual machine interfaces in order to allow many tens of virtual machines running on a single physical machine with less performance loss. Xen does not require change to application binary interface(ABI) and thus existing applications can run without any modification. Afterward, Xen has implemented full-virtualization with the advent of virtualizable hardware.

To avoid the damage from their misbehavior, Xen usually runs in higher privilege level than the kernels of guest OSes. It's guaranteed by running Xen in ring 0 and migrating guest OSes to ring 1. When a guest OS tries to execute a sensitive privilege instruction(e.g. installing a new page table), the processor will stop and trap it into Xen. In Xen, guest OSes are responsible for allocating the hardware page table, but they only have the privilege of direct read and updating the hardware page table must be validated by Xen. Additionally, guest OSes can access hardware memory with only non-continuous way because Xen occupies the top 64MB section of every address space to avoid a TLB flush when entering and leaving the hypervisor. As for the page fault, Xen betakes an extended stack frame to record the faulting address which should be normally read from the privileged processor register(CR2). Regarding the exception such as system calls, Xen allows each guest OS to register a fast exception handler which can be accessed directly by the processor without passing via ring 0. However, this handler should be verified before it's installed in the hardware exception table.

Xen hosts most unmodified Linux device drivers into an initial domain called Domain0, which plays the role of driver domain. Domain0 is created at boot time and responsible for the control of creating, pausing, migrating and terminating other domains(guest domains), CPU scheduling parameters and resource allocation policies, etc. To achieve I/O operation's virtualization, Xen proposes a shared memory and asynchronous buffer descriptor ring model based on device channels. In this model, two aspect of factors must be taken into consideration: transferring I/O message and I/O data. Xen provides two communication mechanisms between guest domains or Xen and guest domains: synchronous call using hypercalls to send messages from guest domains to Xen and asynchronous event using virtual interrupts to send notifications from Xen to guest domains. When the data requested by a guest domain is moved into physical memory, Domain0 will send a virtual interrupt to the corresponding guest domain and exchange the memory page containing the data with a vacant memory page presented by the guest domain.

3.2. KVM

KVM(Kernel-based Virtual Machine)[1] is another open source virtual machine monitor besides VMware using full-virtualization based on the x86 platform that must contain the hardware upholding of virtualization such as Intel's Virtualization Technology(VT) and AMD's Secure Virtual Machine(SVM/AMD-V). It is developed and sponsored by Qumranet(formerly Comanet) in Israel. As a kernel driver added into Linux, KVM enjoys all advantages of the standard Linux kernel and hardware-assisted virtualization. So far, KVM is making its step towards para-virtualization.

KVM introduces virtualization capability by augmenting the traditional *kernel* and *user* modes of Linux with a new process mode named *guest*, which has its own kernel and user modes and answers for code execution of guest OSes. KVM comprises two components: *kernel module* and *user-space*. Kernel module(namely *kvm.ko*) is a device driver that presents the ability to manage virtual hardware and see to the virtualization of memory through a character device */dev/kvm*. With /dev/kvm, every virtual machine can have its own address space allocated by the Linux scheduler when being instantiated. The memory mapped for a virtual machine is actually virtual memory mapped into the corresponding process. A set of shadow page tables is maintained to support the translation of memroy address from guest to host. KVM can easily manage guest OSes with *kill* command and */dev/kvm*. User-space takes charge of I/O operation's virtualization. It employs a device model to simulate the behavior of I/O operation, and sometimes necessarily triggers real I/O operations such as transmitting a packet on an Ethernet interface. KVM also provides a mechanism for user-space to inject interrupts into guest OSes. In essence, user-space is a lightly modified QEMU, which exposes a platform virtualization solution to an entire PC environment including disks, graphic adapters and network devices. Any I/O requests of guest OSes are intercepted and routed into user mode to be emulated by QEMU.

4. Testing Benchmarks

This section describes some basic knowledge about three benchmarks used in the following experiments.

4.1. LINPACK

LINPACK(LINear system PACKage)[10] contains a number of Fortran subroutines based on BLAS(Basic Linear Algebra Subprograms) library to work out different linear equations and linear least-squares problems written by Jack Dongarra, Jim Bunch, Cleve Moler and Pete Stewart. LINPACK was designed for applying to supercomputers in the 1970s and early 1980s and now acts as one of the most authoritative benchmarks in high performance computers. The TOP500 computers in the world are sorted by the LINPACK's result. To follow the development of computer architectures, LINPACK evolves into EISPACK and LAPACK. EISPACK mainly dedicates to numerical computation of the eigenvalues and eigenvectors of matrices. LAPACK(Linear Algebra PACKage) is a free, portable and modern library of Fortran 77 routines to figure out the most common problems in numerical linear algebra and has largely superseded LINPACK.

LINPACK measures the actual peak value of float-point computing power indicated in giga of float-point operations per second(GFLOP/s or GFLOPS). LINPACK contains three level of problem sizes: LINPACK100, LINPACK1000 and HPL. LINPACK100 is the earliest version limited by memory size and forbids any change except for some compilation optimizations to the source program. LINPACK1000 permits to modify or replace the algorithms and softwares so as to solve linear systems with 1000 order matrix and make full use of hardware advantages to obtain as high as possible performance, but all optimizations should accord with the same relative accuracy as standard techniques such as Gaussian elimination with partial pivoting. For large-scale distributed memory systems, HPL(High-Performance LINPACK) runs with random matrix sizes N in double precision(64bits) arithmetic searching for the size N_{max} corresponding to the maximal performance P_{max}. HPL can also report the problem size $N_{max/2}$ where half of the performance $P_{max/2}$ is achieved.

4.2. LMbench

LMbench[11] is an open micro-benchmark suite to test physical hardware and basic processing units of operating systems and common applications developed by Silicon Graphics's Larry McVoy and Hewlett-Packard's Carl Staelin from 1993 to 1995. LMbench embraces multiple lightweight benchmarks and attempts to determine and separate all potential performance defects existed in various common applications. LMbench mainly focuses on the bandwidth and latency of data movement between processor, cache, memory, disk and network. Specifically, LMbench measures not only the bandwidth of cached file read, memory read, memory write, memory copy(bcopy and hand-unrolled loop), pipe and TCP, but also the latency of context switching, networking(connection establishment, pipe, TCP, UDP and RPC hot potato), file system creating and deleting, process creation(fork, fork+exec and system), signal handling, system call, etc. Besides the above, LMbench may also test the processor clock rate calculation. LMbench does not take any advantage of multiprocessor features to get the performance data. The source program of LMbench usually comprises five directories, i.e. *src*, *bin*, *doc*, *scripts* and *results*. The src directory contains all source files of LMbench. After the source programs are compiled, the executable files will be saved in the bin directory. The doc directory holds the presentations made by LMbench authors, the references to published articles and the rebuttal of an article claiming LMbench is flawed. The scripts directory provides some template scripts used to generate and post-process LMbench's output. Finally, the results directory preserves all output files of every test.

4.3. IOzone

IOzone[7] is a free filesystem benchmark that proposed by Oracle's William D. Norcott and improved by Hewlett-Packard's Don Capps and Tom McNeal. IOzone takes various file operations as basic workloads to test the I/O performance of filesystems and allows conner to adopt diverse testing modes from automated to partial. The file operations include *read*, *write*, *re-read*, *re-write*, *read backwards*, *read strided*, *fread*, *fwrite*, *random read/write*, *pread/pwrite variants*, *aio_read*, *aio_write* and *mmap*.

IOzone owns numerous characteristics. Its 64-bit's compatible source program is written in ANSII C and able to measure the performance of POSIX async I/O, normal and mmap file I/O, single and multiple stream, POSIX pthreads, multi-process and size-configurable processor cache, I/O latency data for plots, stonewalling in throughput tests to eliminate straggler effects and so on. Moreover, IOzone also supports different size file's testing and provides the options to measure fsync, O-SYNC and NFS. Additional important note is IOzone offers Excel importable output for graph generation. The *makefile* in IOzone shows a list of supported platforms and the source program should be made into the suited one by typing make target in the command line. To obtain the accurate results, conner needs to make sure that the maximum size of testing file is bigger than the buffer cache. If not knowing the size of buffer cache or using a dynamic buffer cache, then just set the maximum size to be larger than the total physical memory of the platform.

5. Experiments and Performance Study

In order to observe and compare the performance of Xen and KVM, a series of experiments have been conducted by benchmarking native Linux and virtual machines in Xen and KVM with LINPACK10.0.2, LMbench2.5 and IOzone3.287. All experiments were based on a Dell OPTIPLEX 755 business machine with 2.33GHz E6550 Intel Core2 DUO processor, 2GB DDR II 667 RAM and Seagate 160GB 7200 RPM SATA II disk. Moreover, we adopted native Fedora core 8 based on Linux 2.6.24.4(Fedora8) as host OS, Xen3.1 and KVM-60 as virtual machine monitor. The memory allocated for virtual machines(Fedora8 and Windows XP SP2) in Xen and KVM is set to 1.5GB.

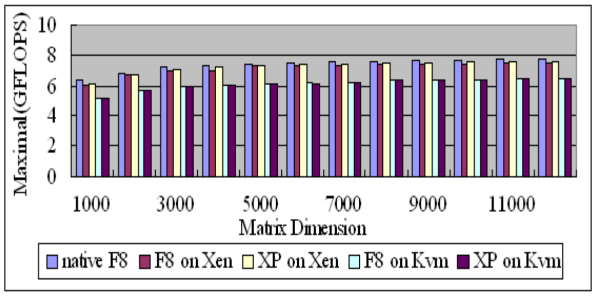

Figure 1. LINPACK results of virtulized Fedora8 and XP in Xen,KVM and native Fedora8

In the first experiment, we have run LINPACK just with one CPU and thread to test and compare the float-point computing power of virtualized Fedora8 in Xen and KVM against native Fedora8, virtualized Fedora8 against virtualized Windows XP in Xen and KVM as Figure 1. The LINPACK results exhibit different peak values of float-point computing in three environments, and the maximal GFLOPS of virtualized Fedora8 in KVM is about 83.46% equal to native Fedora8 and approximately 97.28% With regard to Xen. It suggests that the processing efficiency of KVM on float-point computing is not as good as Xen mainly for KVM needs to check whether every executing instruction is an interrupt, page fault, I/O or common instruction to decide exiting from or staying in guest mode, this judgment wastes a lot of time. Reversely, Xen allows guest OS to execute the float-point computing instructions on physical CPU to save the solving time. Obviously, the overhead of CPU virtualization in a relatively mature virtual machine monitor like Xen may be negligible.

In addition, the LINPACK behavior of virtualized Windows XP comes up prior to virtualized Fedora8 based on the same Xen, and yet KVM doesn't look like this way. An interpretation for this is as a virtual machine monitor embedded in Linux kernel, KVM makes a favorable backup to Fedora8. On the contrary, Xen owns a few enhancement packages for Windows as guest OS and so virtualized Windows XP has a better performance than virtualized Fedora8.

For probing into memory virtualization, we have used LMbench to benchmark virtualized Fedora8 in Xen and KVM against native Fedora8. In all metrics of LMbench,

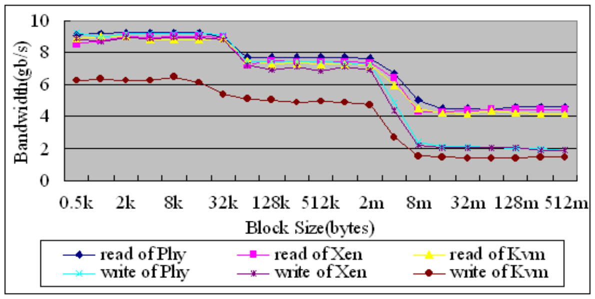

Figure 2. Memory bandwidth of read and write in LMbench

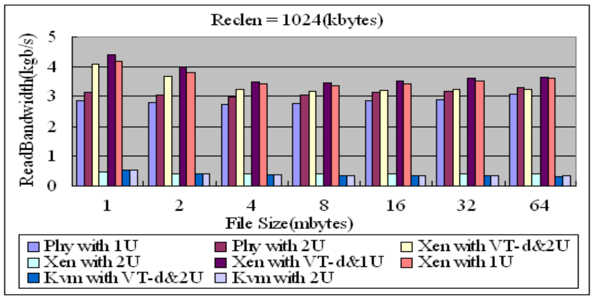

Figure 3. Read bandwidth of different environments and cases in IOzone

we focused on the latency of system operations and context switch and the bandwidth of memory reading and writing.

Table 1. System operations-times in *ms*

	native F8	F8 on Xen	F8 on KVM
syscall	0.356	0.363	0.377
stat	2.810	2.335	2.466
fstat	0.640	0.622	0.648
open/close	3.814	3.320	3.534
fork	88.73	1511.5	2200
exec	315.8	4043	5508
sh	1584	12302	16693
sig hndl	1.529	1.335	1.624
pipe	6.25	44.71	59.11

Table 2. Context switch-times in *ms*

	native F8	F8 on Xen	F8 on KVM
2p0k	0.82	3.05	4.09
2p16k	1.31	4.94	5.46
2p64k	2.66	6.65	8.17
8p16k	1.53	4.90	3.61
8p64k	2.83	6.51	7.45
16p16k	1.53	4.90	3.67
16p64k	2.87	6.69	7.77

Xen and KVM display slower *fork*, *exec*, *sh* and *pipe* performance than native Fedora8 and KVM shows a worse mark than Xen. Due to the process creating and executing need updating the hardware page table, these operations will always trap into and validated by Xen or rebuild the shadow

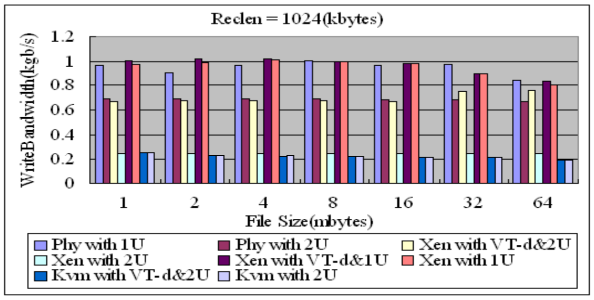

Figure 4. Write bandwidth of different environments and cases in IOzone

page tables and map guest virtual to host physical memory in KVM to cause awful performance of both. Table2 shows context switch times between different numbers of processes with different working set sizes. As a whole, Xen manifests a faster context switch than KVM except when working set size is 16k with more than two processes. It mostly relates to cache effects and context switch overhead for larger working set sizes is slight against cache effects.

The bandwidth of memory reading in three environments denotes a basic harmony, but KVM exposes a big reduction in the bandwidth of memory writing as Figure 2. Since KVM-60 lacks the support to extended page tables(EPT) and optimizations to MMU virtualization and employs a method of memory virtualization resembling full-virtualization, which needs to pre-scan the code that will be executed to replace those sensitive privilege instructions, thus account for the low-grade performance of KVM. Reversely, Xen uses para-virtualization with a cooperative fashion that guest OSes realize themselves being virtualized and batch update requests to amortize the overhead of entering the hypervisor, so Xen can obtain the almost same performance to native Fedora8.

To investigate the virtualization of file system's I/O, we have separately run IOzone on native Fedora8, virtualized Fedora8 in Xen and KVM with or without Intel's VT-d using single or dual CPUs as Figure 3 and Figure 4.

The Read bandwidth of virtualized Fedora8 in Xen with VT-d shows a surpassed performance than native Fedora8 and yet the same metric in Xen without VT-d and KVM arises a tremendous drop about 6 to 7 times compared with native Fedora8. The case of Write is similar to Read just with smaller numbers. As the VT-d embeded the remapping hardware of DMA and IRQ in the North Bridge chipset, guest OS can directly control I/O devices and avoid trapping into the hypervisor for each I/O request. With VT-d opened, KVM doesn't make any advance in I/O bandwidth because it achieves virtual I/O's processing with QEMU which simulates the I/O operations and ignores the hardware advantages. This fully proves the importance of hardware support to software virtualization. Furthermore, owing to synchronous lock of write operation, so it's common that the bandwidth of Write is lower than Read and the performance of single CPU is better than dual CPUs.

6. Conclusions

In this paper, we firstly described the basic knowledge about Xen and KVM and three benchmarks-LINPACK, LMbench and IOzone, and then conducted three experiments to measure the performance of virtualized Fedora8 and Windows XP in Xen and KVM and native Fedora8. By discussing the results of three experiments, we took a glance at the overhead of CPU, memory and I/O operation's virtualization, and revealed some interesting hints. Our experiments proved again that context switch, process creating and executing and I/O operation's virtualization puts a main source of the total virtualization overhead.

References

[1] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. *KVM:the Linux Virtual Machine Monitor.* Proc. of the Linux Symposium, Ottawa, Ontario, Canada: Springer-Verlag, 225−230, 2007.

[2] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, W. Zwaenepoel. *Diagnosing Performance Overheads in the Xen Virtual Machine Environment.* Proc. of the 1st ACM/USENIX International Conference on Virtual Execution Environments(VEE),13−23, 2005.

[3] A. Whitaker, M. Shaw and S.D. Gribble *Scale and Performance in the Denali Isolation Kernel.*Proc. of the 5th Symposium on Operating System Design and Implementation(OSDI), Boston, MA, 195−209, 2002.

[4] Advanced Micro Devices *Inc. AMD I/O-virtualization Technology(IOMMU) Specification.* 2006.

[5] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, J.N. Matthews. *Xen and the Art of Repeated Research.* Proc. of the 2004 USENIX Annual Technical Conference, Boston, MA, 135−144, 2004.

[6] F. Bellard. *QEMU:a Fast and Portable Dynamic Translator.* Proc. of the 2005 USENIX Annual Technical Conference, Anaheim, CA, 41−46, 2005.

[7] D. Capps and W.D. Norcott. *IOzone Filesystem Benchmark.* http://www.IOzone.org. 2004.

[8] J. Dike. *User Mode Linux.* Proc. of the 5th Annual Linux Showcase & Conference, Oakland, CA, 56−69, 2001.

[9] L.H. Seawright and R. A. MacKinnon. *VM/370−A Study of Multiplicity and Usefulness.* IBM Systems Journal. 18(1):4−17, 1979.

[10] J.J. Dongarra, P. Luszczek and A. Petitet. *The LINPACK Benchmark: past, present and future.* Concurrency and Computation Practice and Experience,15(9):803−820, 2003.

[11] L. McVoy and C. Staelin. *LMbench: Portable Tools for Performance Analysis.* Proc. of the 1996 Annual Conference on USENIX Annual Technical Conference, San Diego, CA, 279−294, 1996.

[12] Michael Larabel. *Linux KVM Virtualization Performance.* http://www.phoronix.com/scan.php?page=article&item=623&num=1.

[13] M. Rosenblum and T. Garfinkel. *Virtual Machine Monitors: Current Technology and Future Trends.* IEEE Computer, 38(5):39−47, 2005.

[14] 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.* Proc. of the nineteenth ACM Symposium on Operating Systems Principles(SOSP), 164−177, 2003.

[15] R. Uhlig, G. Neiger, D. Rodgers, et.al.*Intel Virtualization Technology.* IEEE Computer,38(5):48−56, 2005.

[16] VMware *Inc. Understanding Full Virtualization, Paravirtualization and Hardware Assist.* White paper. November 11, 2007.

[17] VMware *Inc. A Performance Comparison of Hypervisors.*January 31, 2007.

[18] XenSource *Inc. A performance comparison of commercial hypervisors.*March 05, 2007.