2010 IEEE Asia-Pacific Services Computing Conference

A Synthetical Performance Evaluation of OpenVZ, Xen and KVM

Jianhua Che
State Grid Electric Power Research Insititute
Nanjing, China
chejianhua@zju.edu.cn

Yong Yu
State Grid Electric Power Research Insititute
Nanjing, China
shicongcong@sgepri.sgcc.com.cn

Congcong Shi
State Grid Electric Power Research Insititute
Nanjing, China
shicongcong@sgepri.sgcc.com.cn

Weimin Lin
State Grid Electric Power Research Insititute
Nanjing, China
linweimin@sgepri.sgcc.com.cn

***Abstract*—Although virtualization holds numerous merits, it meanwhile incurs some performance loss. As the pivotal component of a virtualization system, the efficiency of virtual machine monitor(VMM) will largely impact the performance of the whole system. Therefore, it's indispensable to evaluate the performance of virtual machine monitors with different virtualization technologies. In this paper, we measure and analyze the performance of three open source virtual machine monitors-OpenVZ, Xen and KVM, which adopt the container-based virtualization, para-virtualization and full-virtualization respectively. We first measure them as a black box about their macro-performance on the virtualization of processor, memory, disk, network, server applications(including web, database and Java) and their micro-performance on the virtualization of system operation and context switch with several canonical benchmarks, and then analyze these testing results by examining their design and implementation as a white box. The experimental data not only show some valuable information for designers, but also provide a comprehensive performance understanding for users.**

***Index Terms*—virtualization; virtual machine; virtual machine monitor; performance; evaluation**

I. Introduction

Virtualization has gained widespread uses in cloud computing [8], server consolidation [10] and information security [6] for its multitudinous benefits such as *flexibility*, *isolation*, *high resource utilizing rate*, *easy IT infrastructure management*, *power saving* and so on [13]. In virtualization systems, resource virtualization of underlying hardware and concurrent execution of virtual machines are in the charge of a software called virtual machine monitor(VMM) or hypervisor [9]. By creating the same view of underlying hardware and platform APIs from different vendors, virtual machine monitor enables virtual machines to run on any available computer. This not only eases the numerous application of desktop computers, but also reduces the hardware cost of distributed environments. However, these benefits are not always for free. The existing of virtual machine monitor level debases the performance of some specific operations. As one of the core components, virtual machine monitor will affect the performance of virtualization systems to a great extent, so it's important to measure and analyze the performance of virtual machine monitors.

Our motivations to evaluate the performance of virtual machine monitors are as follows. Firstly, there are a lot of virtual machine monitors so far. Facing so many virtual machine monitors, users sometimes do not know which the best one for their applications is. This situation mainly arises from the shortage of a synthetical evaluation to these virtual machine monitors. To alleviate this quandary, measuring and analyzing the performance of virtual machine monitors is expecting. Secondly, it's well known that virtualization introduces some overhead and the performance of virtual machine monitors adopting different virtualization technologies will be diverse. But how about the performance degradation of virtual machine against physical machine? How much difference between different virtualization technologies? what factors lead to the performance loss of virtualization systems? To answer these questions, measuring and analyzing the multiple-faceted performance(e.g.,macro-performance and micro-performance) of virtual machine monitors is necessary.

In this paper, the effectiveness of three open source virtual machine monitors available for the x86 platform-OpenVZ [17], Xen [11] and KVM [1] have been evaluated. Specifically, our contributions lie in two points:

First, we select them as the delegates of *operating system-level virtualization*, *para-virtualization* and *full-virtualization* respectively to measure their performance of processor subsystem, file subsystem, network subsystem, and provide a quantitative and qualitative comparison of three virtual machine monitors. The testing data offer some helpful information for users to pick the most suitable virtual machine monitor for their applications.

Second, we measure them as a black box to obtain their macro-performance and micro-performance data, and analyze them as a white box to observe their performance characteristics. By investigating their design principle and pros and cons, some potential performance bottlenecks are discovered. The drawn conclusions present valuable details for designers to improve their design of virtual machine monitors.

The rest of this paper is organized as follows: We begin in Section 2 with related works. Then, we introduce some details

978-0-7695-4305-5/10 $25.00 © 2010 IEEE
DOI 10.1109/APSCC.2010.83

IEEE computer society

about three virtual machine monitors-OpenVZ, Xen and KVM in Section 3 and describe the experimental setup and result analysis of measuring them in Section 4. Finally, we conclude with discussion in Section 5.

II. Related Work

Performance evaluation on virtual machine monitors has been extensively studied [10], [2], [4], [11], [3], [5]. Barham et al. [11] gave a full-scale introduction to Xen and measured its performance against VMWare, User-Mode Linux and Base Linux with SPECcpu2000, OSDB, dbench and SPECweb99. Clark et al. [4] reproduced the results from [11] with almost identical hardware, and compared Xen with native Linux on a less powerful PC and evaluated the ability of Xen as a platform for virtual web hosting. Padala et al. [12] enriched the evaluation by including OpenVZ for server consolidation. Compared with them, we do the same test of LMbench, and measure the network performance with NetIO(they used *ttcp*). Although they have done more tests with several extra benchmarks that need to purchase the license, we evaluate more virtual machine monitors.

Recently, Deshane [14] presented an independent work about performance comparison of Xen and KVM at Xen Summit, which measured their overall performance, isolation and scalability with a CPU-intensive test, a kernel compile, an IOzone write test, and an IOzone read test. In this paper, we extend the work in [7] to evaluate synthetically the performance of OpenVZ, Xen and KVM, involving macro-performance and micro-performance. Similarly, we conduct more tests with SPECCPU2006, RAMSPEED, Bonnie++, NetIO, WebBench, SysBench, SPECJBB2005 and get some different conclusions about the performance of Xen and KVM.

In addition to open source virtual machine monitors' comparison, VMware and XenSource made their respective experiments to compare the performance of their commercial virtual machine monitors [15], [18]. They used SPECCPU2000, PassMark, Netperf, SPECJBB2005 and building SPECCPU2000 INT package as workloads to test VMware ESX Server 3.0.1 and commercial XenEnterprise 3.2 based on Xen 3.0.4. Different from them, we choose a higher version of Xen and an open-source KVM as the delegate of full-virtualization.

III. Virtual Machine Monitor

Virtual machine monitor itself is a software that draws the identical abstraction of a physical machine for multiple virtual machines [16]. At present, the popular virtual machine monitors involve VMware, Hyper-V, OpenVZ, Xen and KVM, etc. As OpenVZ, Xen and KVM provide the open source version, so we select them as the delegates of operating system-level virtualization, para-virtualization and full-virtualization to evaluate their performance.

A. OpenVZ

OpenVZ [17] is an operating system container developed by SWsoft, which can create multiple isolated domains, i.e. Virtual Private Servers(VPSs) or Virtual Environments(VEs) on a single server. Due to adopting operating system-level virtualization, OpenVZ requires the kernel of guest operating system are the same as host operating system.

OpenVZ includes a custom kernel and several user-level tools. The custom kernel is a modified Linux kernel with the function of virtualization, isolation, checkpointing and resource management. The resource manager comprises three components: fair CPU scheduler, user beancounters and two-level disk quota. OpenVZ implements a two-level CPU scheduler: the OpenVZ scheduler determines the CPU time slice's assignment based on each container's cpu unit value, and the Linux scheduler decides the process to execute in this container with the standard Linux process priority. Real CPU time is distributed proportionally. OpenVZ controls container operations and system resources such as memory, internal kernel objects(i.e. IPC shared memory segments) and network buffers with about 20 parameters in user beancounters. These resources(including each container's CPU unit value) can be changed without container rebooting. In OpenVZ, each container is assigned an I/O priority as the basis of allotting available I/O bandwidth by the I/O scheduler. Thus, no single container can saturate an I/O channel.

B. Xen

Xen [11] is a para-virtualization hypervisor originally developed at the University of Cambridge, which needs to modify the kernels of host and guest operating system. Xen requires no change to application binary interface(ABI) and thus existing applications can run without any modification. Later, Xen has implemented full-virtualization with the advent of virtualizable hardware.

By running itself in ring 0 and migrating guest operating systems to ring 1, Xen decouples them from underlying hardware and holds full control on system resource. When a guest operating system tries to execute a sensitive privileged instruction(e.g. installing a new page table), the physical processor will trap it into Xen. Guest operating system can update hardware page table with the validation of Xen and access machine memory address in a discontinuous way, because Xen monopolizes the top 64MB section of every address space to escape a TLB flush when entering and leaving the hypervisor. As for memory fault, Xen betakes an extended stack frame to record the fault address. Regarding software exceptions such as system call, Xen allows guest operating system to register a fast handler directly executed by the physical processor, but this handler's installation should be verified by Xen.

From its 2.0 version, Xen hosts most unmodified Linux device drivers into an initial domain called Domain0, which plays the role of a driver domain and controls other guest domains, CPU scheduler parameters and resource allocation policies, etc. To achieve I/O's virtualization, Xen designs a shared memory and asynchronous buffer-descriptor ring model based on device channels. In this model, two factors should be taken into consideration: I/O message and I/O data. Xen provides two communication mechanisms between hypervisor and guest domain: synchronous call using hypercall to send

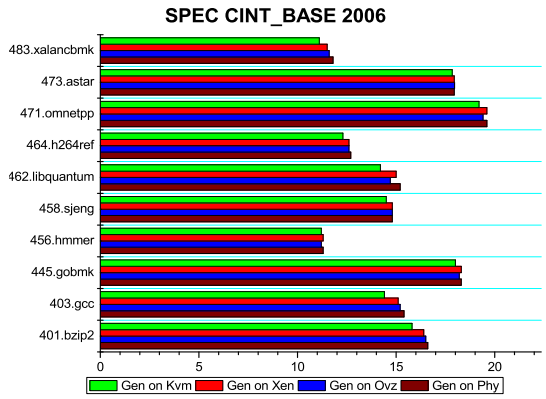

Fig. 1. The result of CINT2006 in four environments

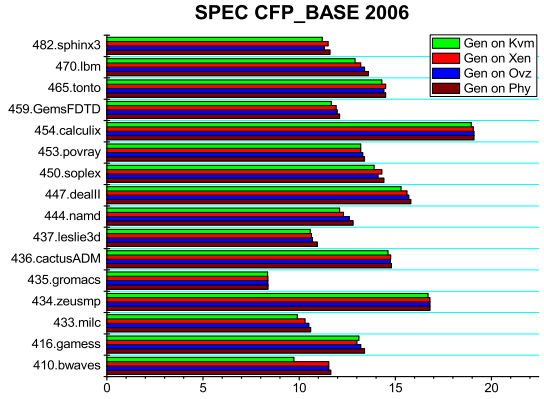

Fig. 2. The result of CFP2006 in four environments

message from guest domain to Xen and asynchronous event using virtual interrupt to send notification from Xen to guest domain.

C. KVM

KVM(Kernel-based Virtual Machine) [1] is a virtual machine monitor using full-virtualization initially developed and sponsored by Qumranet(formerly Comanet) in Israel. As a kernel module added into Linux, KVM enjoys all advantages of standard Linux kernel and hardware-assisted virtualization such as Intel VT or AMD-V. Nevertheless, the para-virtualization implementation of KVM is in progress presently.

KVM implements virtualization by augmenting the traditional *kernel* and *user* mode of Linux with a new mode named *guest*, which has its own *kernel* and *user* mode and answers for code execution of guest operating systems. Guest mode has no privilege to access the I/O devices, otherwise will be changed into user mode. KVM consists of two components: *kernel module* and *user-space*. Kernel module sees to the virtualization of hardware resources through */dev/kvm* and *kill* command. With */dev/kvm*, guest operating system may have its own address space allocated by the Linux scheduler. The physical memory mapped for each guest operating system is actually the virtual memory of its corresponding process. A set of shadow page tables is maintained to support the translation from guest physical address to host physical address. User-space takes charge of the I/O's virtualization by employing a lightly modified QEMU to simulate the behavior of I/O or sometimes necessarily triggering the real I/O. KVM also provides a mechanism for user-space to inject interrupts into guest operating system. Any I/O requests of guest operating system are trapped into user-space and simulated by QEMU.

IV. Performance Measure and Analysis

In order to evaluate the performance of OpenVZ, Xen and KVM, a series of experiments have been conducted by testing native environment and virtual machine on OpenVZ, Xen and KVM with SPECCPU v2006, LINPACK v10.0.2, RAMSPEED v3.4.4, LMbench v3.0, IOzone v3.287, Bonnie++ v1.03, NetIO v126, WebBench v1.5, SysBench v0.4.8 and SPECJBB v2005. All experiments have been run on a Dell OPTIPLEX 755 business machine with a 2.33GHz E6550 Intel Core2 DUO processor, 2GB DDRII 667 RAM, Seagate 250GB 7200 RPM SATA II disk. We adopted Gentoo Linux 2008.0 AMD64 with kernel 2.6.18 as both host operating system(i.e. native Gentoo) and guest operating system(i.e. virtualized Gentoo), and OpenVZ in patched kernel 2.6.18 028stab056.1, Xen 3.3 and KVM-75 as virtual machine monitor. The memory allocated for each virtual machine was set to 1.5GB.

A. Macro-Performance Measure

The Macro-Performance mainly means the overall performance of a subsystem or whole system. Specifically, we will measure the virtualization overhead of processor subsystem, file subsystem, network subsystem, web server, database server and Java server in three virtual machine monitors.

1) Processor Virtualization: To observe the virtualization of processor, we have conducted three experiments to measure the performance of virtualized processor in OpenVZ, Xen, KVM and native Gentoo. The first experiment has tested their efficiencies for computing-intensive workload with SPEC CPU2006 as Figure 1 and Figure 2. KVM displays a little low score regardless of CINT2006 or CFP2006, while OpenVZ and Xen are very close to native Gentoo. Particularly, KVM obtains a very low (almost about 84.31% to native Gentoo) score on 410.bwaves benchmark of CFP2006. 410.bwaves numerically simulates blast waves in three dimensional transonic transient laminar viscous flow, which needs an amount of data transfer. Therefore, KVM spends more time to switch among *guest*, *kernel* and *user* mode than OpenVZ and Xen when updating the matrix of the nonlinear damped periodic system.

The second experiment has tested their float-point computing power with LINPACK as Figure 3. All four environments run on an UP system(marked by $-1u$ in the figure) and a SMP system(marked by $-2u$ in the figure) with Intel VT-x opened. KVM shows an obvious degressive float-point computing peak value, while OpenVZ and Xen are almost accordant to native Gentoo regardless of running on the UP or SMP system. The maximal GFLOPS of OpenVZ, Xen and KVM running on the

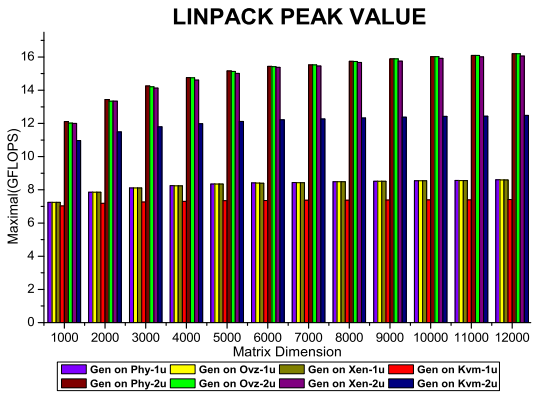

Fig. 3. The result of LINPACK in four environments

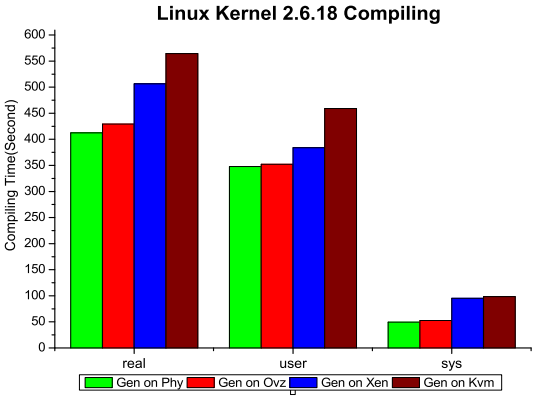

Fig. 4. The result of Linux kernel(2.6.18) compiling

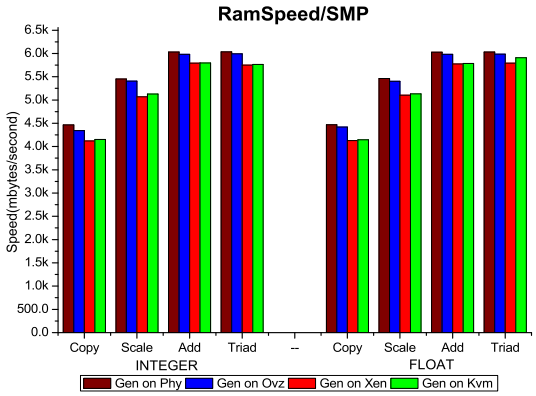

Fig. 5. The speed of memory access in RAMSPEED

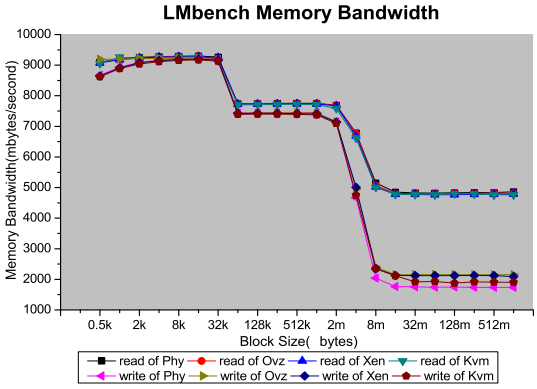

Fig. 6. The bandwidth of memory read and write in LMbench

UP system is about 99.94%, 99.27% and 78.35% to native Gentoo detachedly. These data indicate that the processing efficiency of KVM on float-point computing is not so good as OpenVZ and Xen, similarly for KVM needs a few of mode switches when updating the matrix of LINPACK. The maximal GFLOPS of three virtualized Gentoos running on the SMP system lay out the same rule as running on the UP system. It's necessary to note that, the maximal GFLOPS of three virtualized Gentoos running on the SMP system are not twice as much as running on the UP system due to inter-processor communication.

The third experiment has measured the time of compiling the Linux kernel 2.6.18 in four environments as Figure 4. As the memory required to compile a single file is far less than the total memory of virtualized Gentoo, so the kernel compiling time can reflect the performance of processor virtualization to a great extent. In kernel compiling, OpenVZ holds the closest time to native Gentoo, Xen follows OpenVZ with a slender lag and yet KVM spends the most time.

According to our results, OpenVZ and Xen own the preferable performance in three kinds of tasks by virtue of their superior virtualization mechanism. However, there is still room for KVM to improve its implementation efficiency on float-point computing and kernel compiling. In general, the overhead of processor virtualization in relatively mature virtual machine monitors may be negligible and processor virtualization has not been the performance bottleneck of a virtualization system.

2) Memory Virtualization: To study the virtualization of memory, we have run the SMP version of RAMSPEED and LMbench to measure their access bandwidth of virtualized memory in OpenVZ, Xen, KVM and native Gentoo as Figure 5 and Figure 6.

The results of RAMSPEED and LMbench in four environments disclose a basic harmony on all memory access operations. The speed of four INTEGER operations are almost the same as four FLOAT operations, but the speed of *copy* and *scale* are lower than *Add* and *Triad*, mainly because *copy* and *scale* involve more data movements. The memory *read* and *write* bandwidth of LMbench in four environments show the performance of a typical memory architecture: level I cache(16K in our machine), level II cache(4M in our machine) and main memory. Generally, the results of RAMSPEED and LMbench suggest that the presence of virtualization layer doesn't impact the performance of memory access operations, especially memory access bandwidth.

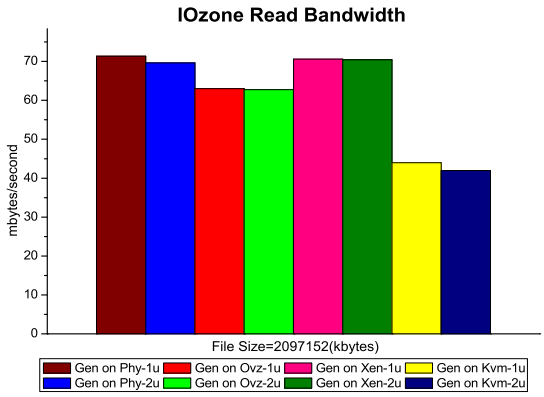

Fig. 7. The bandwidth of file *read* in IOzone

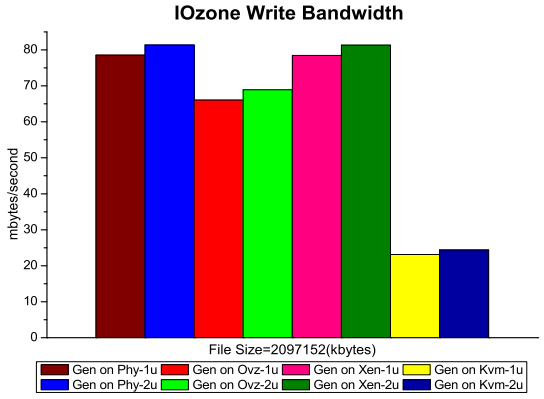

Fig. 8. The bandwidth of file *write* in IOzone

3) Disk Virtualization: To investigate the virtualization of disk, we have run IOzone and Bonnie++ in four environments. As shown in Figure 7 and Figure 8, the file *read* and *write* bandwidth of KVM running on the UP system is the lowest in three virtual machine monitors. It is well-known that KVM implements its virtual disk I/O with the modified QEMU process, the disk I/O requests of guest operating system in KVM are trapped into the kernel mode and the kernel schedules QEMU to simulate the disk I/O operation with a huge performance loss. OpenVZ has a two-level disk I/O scheduler with the Jens Axboe's CFQ scheduler lying at its second level, and assigns the available I/O bandwidth according to each container's I/O priority specified by its CPU priority. This strategy effectively guarantees the fair share of host I/O channel among containers, but at the cost of utilizing incompletely them. Therefore, OpenVZ only holds a partial *read* and *write* bandwidth compared with native Gentoo. Instead of emulating hardware devices, Domain0 pends the I/O request of guest operating system and activates the relevant driver to access the hardware device. After the I/O data are moved into the physical memory page with DMA, Domain0 will notify the guest operating system to exchange the physical memory page with its own empty page. This "page-flipping"

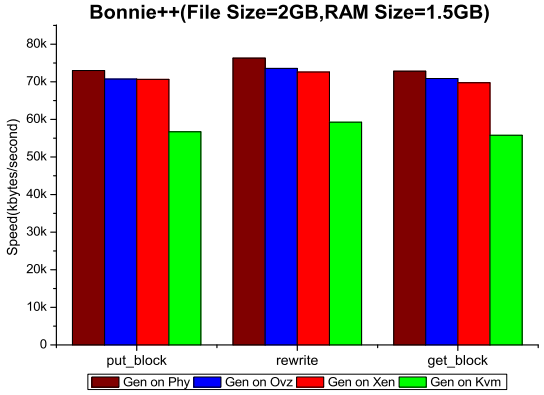

Fig. 9. The bandwidth of three disk I/O operations in Bonnie++

technology saves a lot of time required by data copy through the whole virtualization architecture. Similarly, when receiving a disk write request, Domain0 will look up the virtual block device(VBD) identifier and offset in a translation table to create the logical disk and corresponding sector address. The zero-copy data storage replaces using DMA between physical disk and pinned memory pages of the requested guest operating system, so Xen possess a better file *read* and *write* bandwidth.

OpenVZ, Xen and KVM running on the SMP system exhibit a lower *read* bandwidth than running on the UP system, while the case of *write* bandwidth is exactly reverse. This is mainly because the consistency constraint between dual processors. On the contrary, two processors may write the buffer data concurrently without any synchronous requirement, thus the *write* bandwidths of three virtualized environments running on the SMP system become a little higher than that of three virtualized environments running on the UP system.

From the result of Bonnie++ running on the UP system as Figure9, Xen shows up the best bandwidth of three disk I/O operation for its optimistic implementation of virtual disk I/O, and OpenVZ takes up the second place with an obvious surpassing to the disk I/O emulation method of KVM.

According to our results, disk I/O takes up a large performance overhead in both environments especially in virtualized environment, and disk I/O is a performance bottleneck of virtual machine monitors. Therefore, the optimization to disk I/O virtualization will significantly improve the performance of virtualization systems.

4) Network Virtualization: The speed of transmitting and receiving data packet in four environments displays an orderly sequence as Figure 10 and Figure 11. OpenVZ implements network transmission with its virtual network device(*venet*), which exchanges data packet based on the IP header like a P2P connection between container and host. Hence, OpenVZ has the best performance of TCP transmission. If a guest operating system in Xen wants to transmit a data packet, it simply enqueues a buffer descriptor into the transmitting ring. Xen transmits the packet payload with a scatter-gather DMA and

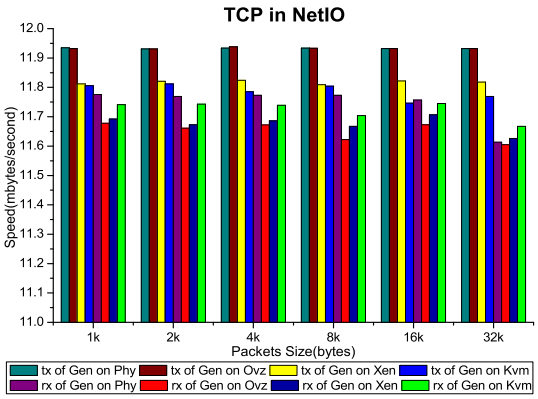

Fig. 10. The TCP result of NetIO, tx is transmit, and rx is receive

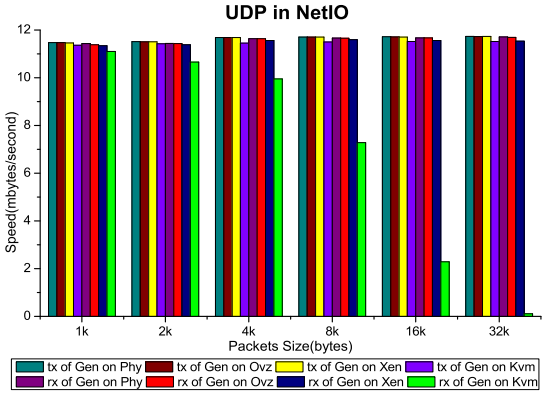

Fig. 11. The UDP result of NetIO, tx is transmit, and rx is receive

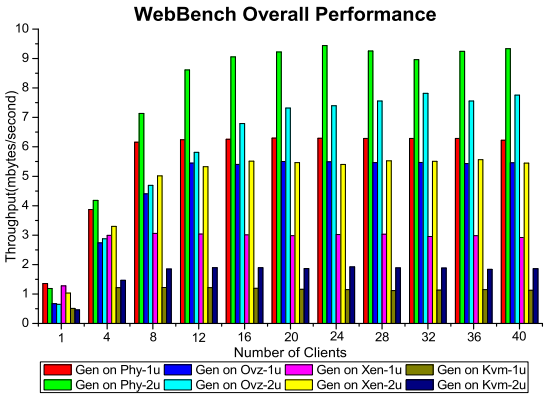

Fig. 12. The throughput result of WebBench

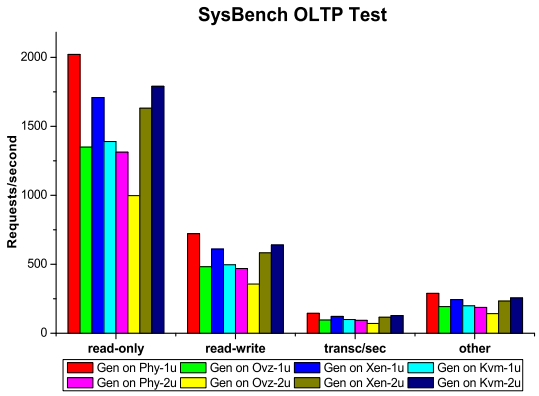

Fig. 13. The OLTP test of SysBench

only copies the descriptor and packet header to match the filter rules, so Xen can have a very close transmitting speed to native Gentoo. KVM implements its network transmission with the same way as QEMU. The guest operating system that sends a packet will be interrupted into user-space by the Linux kernel, and then user-space transmits the packet through a network interface simulated by QEMU. Therefore, transferring a data packet in KVM is expensive.

One costly work in receiving a data packet is payload copy between virtual machine monitor and guest operating system. When receiving a data packet, OpenVZ will firstly scan the IP address table and notify the corresponding container to access the network interface. Moving data from the network device to the container costs some times. To efficiently receive a packet, Xen will immediately check the receiving rule set to determine the destination VIF, and exchanges the packet buffer with a page on the relevant receiving ring. Although this requires the page-aligned receiving buffers to queue at the network interface, it implements the packet transferring in main memory. Consequently, Xen obtains a fast receiving speed. Strangely, the speed of receiving data packets in KVM is higher than OpenVZ and Xen with unknown reason.

The case of UDP transmission is something different. OpenVZ and Xen have the almost same performance as native Gentoo no matter transmitting or receiving a data packet. KVM gives a degressive receiving speed when packet size is increased from 1 to 32 kilobytes, mainly because of its serious packets losing. In fact, the packet losing ratio of KVM arises from about 53% to 99% along with the packet size increasing.

5) Web Server: To explore the capability of OpenVZ, Xen and KVM to support a virtual machine to act as a Web server, we have run WebBench in four environments. According to the result of WebBench shown in Figure 12, the Web throughput of four environments gradually reaches their upper limit till the client number is bigger than 8. Furthermore, when the client number is between 1 and 8, the Web throughput of four environments running on the SMP system show better performance than running on the UP system. Since one client produces just one working thread in WebBench, it not only fails to exert fully the capability of dual processors, but also gets even worse due to the inter-chip(or core) communication. With multiple threads created by multiple clients, the SMP system can process them in parallel, so the Web throughput of four environments running on the SMP system is higher than running on the UP system.

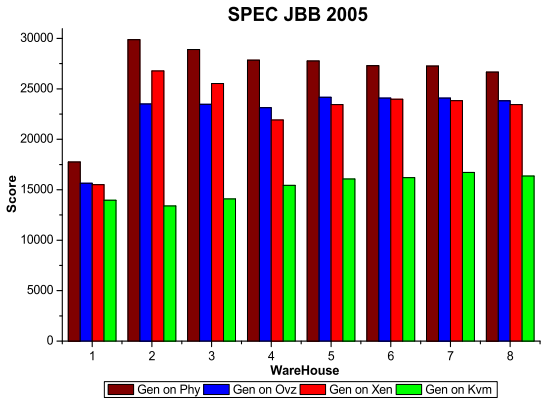

Fig. 14. The result of SPECjbb2005 test

6) Database Server: To examine the ability of OpenVZ, Xen and KVM to support a virtual machine to act as a database server, we have measured their throughput of *query*, *update*, transaction submitting and other operations in OLTP with SysBench. The database server running in three virtualized Gentoos is MySQL. According to the result of SysBench shown in Figure 13, Xen keeps the highest throughput of four database operations among three virtualized Gentoos when running on the UP system. It's mainly because Xen uses the para-virtualization method to deal with the requests of four database operations, just like the way in the experiments of WebBench and NetIO. As OpenVZ implements a strict isolation in network communication between guest and host operating system, it slows down this time. KVM has an equal to or even higher throughput than OpenVZ.

The case of running on the SMP system is quite different. The throughput of native Gentoo, OpenVZ and Xen cut down with different degrees individually in contrast to running on the UP system, because all of them involve a problem of synchronous lock between dual processors. But strangely, we don't know why KVM running on the SMP system performs with an ascending trend in contrast to running on the UP system.

7) Java Server: To inspect the capability of OpenVZ, Xen and KVM to support a virtual machine to act as a Java server, we have tested their throughput under different warehouse numbers with SPEC JBB2005. According to the result of SPEC JBB2005 shown in Figure14, Xen wins the highest score when the warehouses number is smaller than 4. It's mainly because Xen uses the para-virtualization method to deal with various Java transactions, e.g., *new_order*, *payment*, *order_status*, *delivery*, *stock_level* and *cust_report*. But OpenVZ keeps a better performance than Xen when the warehouse number exceeds 3, perhaps because OpenVZ has a nicer capability than Xen to sustain more concurrent processes with the host's assistance. KVM exposes the worst score for the inferior efficiency of full-virtualization to deal with various Java transactions. Although Xen and OpenVZ hold a superior performance to KVM, however, all of their performance fall behind native Gentoo. This clarifies the need of further improvement on the processing efficiency of these Java transactions for three virtual machine monitors.

B. Micro-Performance Measure

The Micro-Performance mainly means the fine performance of some specific operations or instructions. In this section, we have measured the virtualization overhead of common system operations and context switch in four environments with LMbench. Specially, we focus on their latencies shown in table I and table II.

TABLE I
SYSTEM OPERATIONS-TIMES IN μs

	Phy	Ovz	Xen	KVM
syscall	0.1083	0.1105	0.5699	0.1229
read	0.1746	0.1759	0.7752	0.1893
write	0.1742	0.1754	0.7515	0.183
stat	0.9829	0.9978	2.5575	1.0149
fstat	0.1896	0.1965	0.7603	0.2187
open/close	1.4639	1.4765	4.0108	1.7463
fork+exit	115.71	117.24	401.75	880.67
fork+exec	349.87	357.38	1074.2	2193
fork+sh	1222.2	1401.5	3414.5	5728
sigl insl	0.2289	0.2308	0.7155	0.2514
sigl hndl	1.6909	1.6977	3.5129	5.9628
pipe	2.4568	2.5545	8.9256	10.034

1) System Operation Virtualization: As OpenVZ implements common system operations such as system call(e.g. *read*, *write*, *stat*, *fstat*, and *open/close*), process create(i.e. *fork*, *fork+exec*, *fork+/bin/sh -c*), signal install and handle, pipe build with the host's APIs, it's not curious that OpenVZ holds a similar latency to native Gentoo in all operations except for *fork+exec* and *fork+/bin/sh -c*, which execute a new or existed program respectively after creating a new process. Interestingly, KVM shows a latency close to native Gentoo and better than OpenVZ and Xen in several system calls and signal install, but has longer delay than OpenVZ and Xen in process create. As the process create often involves memory allocation and hardware page table updating, KVM has to submit the request to the Linux kernel and wait for available address space. It is a long trip to remap guest physical address to host physical address with the shadow page table. Differently, Xen discards the shadow page table and allows guest operating system itself to register a local page table and common signal/exception handlers with MMU. This enables guest operating system to batch the hardware page table updatings with a single hypercall, and reduces the TLB flush frequency. Therefore, Xen owns a better performance than KVM in process create, signal handle and pipe build.

2) Context Switch Virtualization: Context switch is a sensitive privileged operation to store and restore the processor state(i.e. context) such that multiple processes can share a single processor resource, which should be hooked by most virtual machine monitors. OpenVZ switches the processor context of a guest operating system by resorting to the host kernel, accounting for its identical latency of context switch to native

Gentoo. Xen has to execute a hypercall at least once to change the page table base and complete the context switch, which incurs some extra overhead. KVM also turns to the Linux kernel to switch the processor context, but involves remapping the shadow page table, so it needs more time of context switch. Furthermore, the time of context switch for bigger size processor state data sets(perhaps more representative for real applications) becomes longer with eliminating the cache effect.

TABLE II
THE LATENCY OF CONTEXT SWITCH IN μs

	Phy	Ovz	Xen	KVM
2p0k	0.64	0.64	2.38	5.09
2p16k	1.24	1.28	2.92	5.86
2p64k	0.94	0.98	2.71	5.78
4p0k	0.88	0.88	3.06	5.19
4p16k	1.51	1.63	3.74	5.99
4p64k	1.18	1.19	3.43	5.49
8p0k	0.95	0.98	3	5.05
8p16k	1.46	1.55	3.51	5.95
8p64k	1.21	1.29	3.23	5.85
16p16k	1.49	1.53	3.46	6.01
16p64k	1.17	1.18	3.14	5.86
64p16k	1.52	1.56	3.45	6.22
64p64k	6.89	6.97	9.67	12.78

V. CONCLUSION AND FUTURE WORK

Performance evaluation on virtual machine monitors plays an important role in improving their design and implementation. Through measuring and analyzing OpenVZ, Xen and KVM with SPEC CPU2006, LINPACK, Kernel compiling, RAMSPEED, LMbench, IOzone, Bonnie++, NetIO, WebBench, SysBench and SPEC JBB2005, we found that OpenVZ has the best performance and Xen follows OpenVZ with a slight degradation in most experiments, while KVM has apparently lower performance than OpenVZ and Xen. Furthermore, this indicates that operating system-level virtualization and para-virtualization have some apparent advantage on data-intensive applications such as disk I/O, net I/O, web server, database server and Java server. However, operating system-level virtualization and para-virtualization have their own drawbacks: Operating system-level virtualization can only virtualize guest operating system that the kernel is the same as host operating system, and para-virtualization requires changes to the kernel of guest operating system. On the contrary, full-virtualization presents users the most convenient use without any modification to guest operating system. Hence, there is often a tradeoff between performance loss and easy use. In addition, according to our testing results, processor virtualization is not the performance bottleneck of a mature virtual machine monitor, whereas most virtual machine monitors lose a lot of performance on three aspects: hardware page table fault, interrupt request and I/O. Therefore, focuses should be fixed on optimizing these potential bottlenecks to improve the performance of virtual machine monitors.

In the future, we will measure and analyze the power consumption of OpenVZ, Xen and KVM with some authoritative power benchmarks, and study their scalability, isolation and live migration in depth.

REFERENCES

[1] A. Kivity, Y. Kamay, D. Laor, U. Lublin and A. Liguori, *kvm: the Linux Virtual Machine Monitor*. In *Linux Symposium*, pages 225-230, 2007.

[2] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman and W. Zwaenepoel, *Diagnosing performance overheads in the xen virtual machine environment*. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments(VEE'05)*, USA: ACM New York, pages 13-23, 2005.

[3] A. Whitaker, M. Shaw and S. Gribble, *Scale and Performance in the Denali Isolation Kernel*. In *Proceedings of Symposium on Operating Systems Design and Implementation(OSDI'02)*, pages 195-209, 2002.

[4] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne and J. Matthews, *Xen and the Art of Repeated Research*. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 135-144, 2004.

[5] C. A. L. Ongaro D. and R. S, *Scheduling I/O in Virtual Machine Monitor*. In *ACM/USENIX International Conference on Virtual Execution Environments(VEE'08)*, pages 1-10, 2008.

[6] D. Duchamp and G. De Angelis, *A hypervisor based security testbed*. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test*, USA:USENIX Association, 2006.

[7] J. H. Che, Q. M. He, Q. H. Gao and D. W. Huang, *Performance Measuring and Comparing of Virtual Machine Monitors*. In *International Workshop on End-User Virtualization on IEEE/IFIP International Conference on Embedded and Ubiquitous Computing(EUC2008)*, 2008.

[8] L.M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, *A break in the clouds: towards a cloud definition*. *ACM SIGCOMM Computer Communication Review*, USA:ACM Press New York, 39(1):50-55, 2006.

[9] M. Rosenblum and T. Garfinkel, *Virtual Machine Monitors: Current Technology and Future Trends*. *COMPUTER*, pages 39-47, 2005.

[10] P. Apparao, R. Iyer, X. Zhang, D. Newell and T. Adelmeyer, *Characterization & analysis of a server consolidation benchmark*. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments(VEE'08)*, USA: ACM New York, pages 21-30, 2008.

[11] 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*. *ACM SIGOPS Operating Systems Review*, 37(5):164-177, 2003.

[12] P. Padala, X. Zhu, Z. Wang, S. Singhal and K. Shin. *Performance Evaluation of Virtualization Technologies for Server Consolidation*. Technical Report HPL-2007-59, HP Labs, April 2007.

[13] R. Shiveley, *Enhanced Virtualization on Intel® Architecture-based Servers*. *Technology*, 2005.

[14] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah and B. Rao. *Quantitative Comparison of Xen and KVM*. *Xen Summit.*, June 23-24, 2008.

[15] V. Inc. *A Performance Comparison of Hypervisors*. Technical report, VMWare Inc., 2007.

[16] V. Inc. *Understanding Full Virtualization, Paravirtualization and Hardware Assist*. Technical report, VMWare Inc., 2007.

[17] Wiki. *Welcome to OpenVZ Wiki*. http://wiki.openvz.org/Main_Page.

[18] X. Inc. *A Performance Comparison of commercial hypervisors*. Technical report, XenSource Inc., 2007.