INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 9, Number 2, Pages 351–370



# PERFORMANCE COMBINATIVE EVALUATION FROM SINGLE VIRTUAL MACHINE TO MULTIPLE VIRTUAL MACHINE SYSTEMS

KEJIANG YE, JIANHUA CHE, QINMING HE, DAWEI HUANG, AND XIAOHONG JIANG

**Abstract.** Virtualization technology is widely used in server consolidation, high performance computing, and cloud data center due to its benefits on high resource utilization, flexible manageability, and dynamically scalability. However, it also introduces additional performance overheads. It's worthy to evaluate the overheads and to find the bottleneck of virtualization in different scenarios. In this paper, we propose a combinative evaluation method to analyze the performance from single virtual machine to multiple virtual machine systems that measures and analyzes both the macro-performance and micro-performance. By correlating the analysis results of two-granularity performance data, some potential performance bottlenecks come out.

**Key words.** Virtualization, Performance, VMM, HPC, Virtual cluster.

## 1. Introduction

Virtualization technology has recently become increasing popular in wide areas, such as server consolidation [1, 2], high performance computing (HPC) [3, 4], and modern cloud data center [5, 6]. Single virtual machine system virtualized on single physical machine is a basic form. While multiple virtual machine system virtualized on single or multiple physical machines is a more common scenario (see Figure 1). It is obvious that virtualization brings many benefits such as flexible resource management, high reliability, performance isolation, and OS customization. In a typical virtualization system, resource virtualization of underlying hardware and concurrent execution of virtual machines are in the charge of virtual machine monitor (VMM) or hypervisor [7]. 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 physical machines. However, virtual machine monitor also complicates the implementation of traditional computer architectures and depresses the performance of some specific operations, so it's significant to assess the performance of virtualization in both single virtual machine and multiple virtual machine system.

Currently, there are several mature virtualization solutions (typical virtual machine monitors) for x86 system using different implementation methods, such VMware [8] and KVM [9] for full virtualization, Xen [10] for both para-virtualization and full virtualization, and OpenVZ [11] for operating system level virtualization. Different virtualization solutions have different strengths and weaknesses. For example, full virtualization supports both Linux and Windows virtual machines but the performance is relatively poor, while para-virtualization cannot support Windows virtual machine but the performance is better. It is necessary to choose a most appropriate virtualization method for particular purposes. Further, in the multiple virtual machine system, such as HPC virtualization system, different forms of communication

Received by the editors October 12, 2009 and, in revised form, April 28, 2010.
This research was supported by the National 973 Basic Research Program of China under grant NO.2007CB310900 and National Natural Science Foundation of China under grant NO. 60970125.

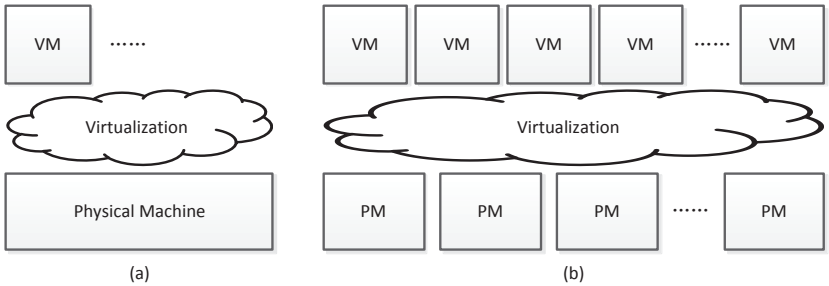


FIGURE 1. Typical virtualization scenarios: (a) the single virtual machine system on single physical machine; (b) the multiple virtual machine system on single or multiple physical machines.

between virtual machines is a major performance bottleneck, such as MPI (Message Passing Interface) communication, network communication, etc. How can the virtualization affect the communication efficiency is an important issue. Based on the above analysis, it is essential to analyze the performance overheads of single virtual machine system (such as processor virtualization, memory virtualization, disk I/O virtualization, network virtualization, etc.), compare the virtualization efficiency of different typical virtual machine monitors, and investigate the various communication overheads in multiple virtual machine system.

The performance of applications running in virtual machines is different from that in the native environment due to the existing of virtual machine monitor. A lot of work has been done in the performance evaluation on single virtual machine system focusing on the performance of CPU, memory, disk I/O, and network [10, 12–15], and the performance issues of server consolidation [1, 2, 16]. However, to our knowledge, there is still relatively few work focus on the performance evaluation of multiple virtual machine system, especially the HPC virtualization system. Some of the researchers have investigated into virtualization performance of HPC applications [4, 17–19]. However they didn't perform a deep analysis on the performance overheads and bottleneck of the network I/O processing mechanism in multiple virtual machine system. What's more, they only focused on the macro performance evaluation and didn't refer to micro performance analysis such as profiling analysis from the hardware architecture perspective.

In this paper, we firstly study the component virtualization overheads of single virtual machine system (such as the virtualization efficiency of processor, memory, disk I/O, network I/O, etc) by comparing the performance of different virtualization technologies, i.e. para-virtualization, full virtualization, and operating system level virtualization. Then we create two 16-node virtual clusters, and do a comprehensive performance evaluation of multiple virtual machine system (i.e. HPC virtualization system) to investigate the virtualization efficiency for HPC applications, including floating point computing performance, memory bandwidth performance, data transfer rate, network bandwidth and latency. Especially, we present a detailed assessment of various communication overheads with HPC applications running in virtual cluster. Besides, we also investigate the micro performance for both single virtual machine and multiple virtual machine systems. For single virtual machine system, we investigate some specific operations such as *system call* and *context switch*. While for multiple virtual machine system, we analyze the profiling data

collected from the hardware events when running HPC applications in multiple virtual machine environment using Oprofile/Xenoprof toolkit [20]. We focus on three hardware events CPU_CLK_UNHALTED, INST_RETIRED, and LLC_MISSES on Intel platform to gather CPU cycles, instruction number, and L2 cache misses respectively which are helpful for explaining the performance bottleneck.

Experimental results show that: 1) disk I/O is a performance bottleneck and the latencies of *process create* and *context switch* are two main factors that baffle the performance of single virtual machine system; 2) the optimized network I/O processing mechanism in Xen's para-virtualized cluster can achieve better efficiency compared to full virtualized cluster since the Front-End/Back-End network I/O mechanism of para-virtualization can cause fewer traps than emulated I/O mechanism of full virtualization and performs better performance in inter-domain communication; 3) the MPI and network communication overheads in multiple virtual machine system are the main bottleneck for full virtualized cluster, which cause huge L2 cache miss rate.

The rest of the paper is structured as follows. In Section 2, we introduce the background of typical virtual machine monitors, network I/O virtualization in Xen, and the background of HPC virtualization. In Section 3, we present our experimental setup, macro benchmarks and micro performance analysis and profiling tools for both single and multiple virtual machine system. In Section 4 and 5, we perform a detailed evaluation and analysis on the virtualization overheads and bottleneck of single virtual machine system and multiple virtual machine system respectively. Section 6 presents the related work. Finally we give our conclusion and future work in Section 7.

## 2. Background & motivation

**2.1. Typical virtual machine monitors.** KVM [9] (Kernel-based Virtual Machine) is a virtual machine monitor based on full-virtualization and hardware-assisted virtualization technology such as Intel VT or AMD-SVM. KVM implements virtualization by augmenting traditional *kernel* and *user* mode of Linux with a new process mode named *guest*. Guest mode has no privilege to access the I/O devices. KVM consists of two components: *kernel module* and *user-space*. Kernel module manages the virtualization of memory and other hardware through a character device */dev/kvm* and *kill* command. With */dev/kvm*, each virtual machine may have its own address space. A set of shadow page tables are maintained for translating guest address to host address. User-space takes charge of the I/O's virtualization with a lightly modified QEMU.

Xen [10] is a para-virtualized hypervisor that needs kernel modifications of host and guest operating systems, but no changes to application binary interface(ABI) so that existing applications can run without any modification. Guest domains can manage hardware page table with read-only privilege, but update operation should be validated by Xen. Regarding system calls, Xen allows guest domains to register fast handlers directly accessed by physical processor bypassing ring 0. From its 2.0 version, Domain0 hosts most unmodified Linux device drivers and controls resource allocation policies and guest domain management. To achieve I/O virtualization, Xen designs a shared memory and asynchronous buffer-descriptor ring model and two communication mechanisms: synchronous call and asynchronous event. Now, Xen also supports full virtualization by the hardware assistant of virtualization.

OpenVZ [11] is an operating system container that requires the same kernels of host and guest operating systems. It includes a custom kernel and several user-level

tools. The custom kernel is a modified Linux kernel with virtualization, isolation, checkpoint 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 that the OpenVZ scheduler see to 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. OpenVZ controls container operations and system resources with about 20 parameters in user beancounters.

**2.2. Network I/O virtualization in Xen.** Xen implements the network virtualization by using two I/O rings to perform the data transfer between DomU (virtual machine) and Dom0 (driver domain). The first ring is responsible for processing outgoing packets while the second ring is responsible for processing incoming packets. The process of network packet processing is described as follows: (1) The NIC receives a packet request and generates an interrupt; (2) VMM forwards the interrupt to the Dom0 (driver domain); (3) Dom0 DMAs the packet into the reception I/O ring; (4) Dom0 copies data directly from the back-end driver to the front-end driver in DomU via shared memory; (5) The back-end driver in DomU requests the VMM to send a virtual interrupt to notify the DomU of the new packet; (6) The packet is processed in the DomU. This Front-End/Back-End virtualized network I/O mechanism with shared memory can efficiently improve the network I/O performance.

Figure 2 shows the Front-End/Back-End virtualized network I/O mechanism in Xen. The Dom0 runs a modified version of OS that uses native driver to manage hardware devices. Other VMs (DomU) communicate with Dom0 to transmit and receive packets through shared memory I/O channels.

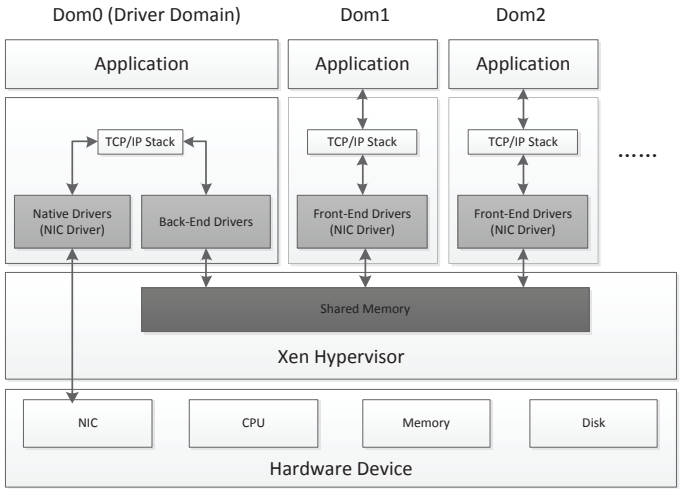


FIGURE 2. The Front-End/Back-End virtualized network I/O in Xen.

**2.3. HPC virtualization.** Applying virtualization technology into the HPC area can greatly improve the efficiency, such as efficient virtual machine management, high fault tolerance, etc. However, there are several requirements should be satisfied to ensure the virtualization efficiency in HPC system: (1) The virtualization performance overheads should not seriously affect the application performance, especially

the performance-critical HPC applications; (2) The virtualization technology should effectively improve the management efficiency such as fast creating and shutdown VM, fast clone and migrate VM, and flexible distribution of virtualized resources; (3) The virtualization technology should ensure the reliability and security of HPC system, such as rapid recovery from crashes, fast reconfiguration VM, and good isolate between VMs.

## 3. Experimental methodology

**3.1. Experimental setup.** The single virtual machine experiments are performed on Dell OPTIPLEX 755 with Intel Core2 Quad CPU at 2.4GHz and 2GB physical memory. Each virtual machine is allocated with 1VCPU and 1GB virtual memory. While the multiple virtual machine experiments are performed on Dell 2900 PowerEdge server, with 2 Quad-core 64-bit Xeon processors at 1.86GHz and 6GB RAM which is more powerful to support multiple virtual machines. We use Ubuntu Linux 8.10 AMD64 with kernel 2.7.12 as host and guest operating system, and OpenVZ in patched kernel 2.7.12, Xen 3.3 and KVM-72 as virtual machine monitor. The virtual machines used for multiple virtual machine system are allocated with 2VCPUs and 256MB memory. The MPI environment used is MPICH 2.1.0.8.

In order to ensure the data precision, each of the showed experiment results was obtained via running benchmark five times on the same configuration, the highest and lowest values for each test were discarded, and the remaining three values were averaged.

**3.2. Macro benchmarks.** We choose a number of benchmarks for the macro performance evaluation of single virtual machine performance, SPECCPU 2006 for the CPU virtualization evaluation, RAMSPEED 3.4.4 for the memory virtualization evaluation, Bonnie++ 1.03 for the Disk I/O virtualization evaluation, NetIO126 for the network I/O virtualization evaluation, and SPECJBB 2005 for the Java server virtualization evaluation.

We choose the HPC Challenge Benchmark suite (HPCC) [21] for the macro performance evaluation of multiple virtual machine system, i.e. HPC virtulization. The HPCC suite is a comprehensive set of synthetic benchmarks designed to profile the performance of several aspects of a cluster. The testing applications used in our study are listed as follows: *HPL* measures the floating point rate of execution for solving a linear system of equations; *DGEMM* measures the floating point rate of execution of double precision real matrix-matrix multiplication; *FFT* measures the floating point rate of execution of double precision complex one-dimensional DDF (Discrete Fourier Transform); *FTRANS* measures the rate of transferring for large arrays of data from multiprocessor's memory; *STREAM* measures the memory bandwidth and the corresponding computation rate for simple vector kernel; *RandomAccess* measures the rate of integer random updates of memory; and *Network latency* measures the communication latency and *Bandwidth* measures the communication bandwidth.

There are three running modes in HPCC: **single** means that a single processor runs the benchmark, **star** means all the processors run separate independent copies of the benchmark with no communication, **mpi** means all processing elements run the benchmark in parallel using explicit data communications. In our experiments, three problem sizes were evaluated, they are 1000MB, 2000MB, 3000MB. The block and grid sizes used are common-block : 80, 100, 120; grid: 2*2, 1*4, 4*1.

**3.3. Micro performance analysis and profiling tools.** LMbench [22] is a suite of simple, portable, ANSI/C micro benchmarks for UNIX/POSIX. In general, it measures two key features: latency and bandwidth. LMbench is intended to give system developers insight into basic costs of key operations. We use this performance analysis tool to analyze the performance bottleneck of single virtual machine system.

OProfile is a system profiling tool for Linux. It is capable of profiling all parts of a running system, from kernel to shared libraries and binaries. It is an ideal tool for profiling entire systems to determine bottlenecks in real-world systems. Xenoprof [13, 20] is an extended OProfile tool that is a system-wide statistical profiling toolkit implemented for Xen virtualization environments. It allows coordinated profiling of multiple VMs in a system to obtain the distribution of hardware events such as clock cycles, instruction number, and cache misses. This is useful for finding the performance bottleneck and can help optimize the performance of Xen. In this paper, we use the above tools to gather the events like CPU cycles, instruction number and L2 cache misses. Based on the profiling data, we can find the bottlenecks of running HPC applications in Xen virtualization environment. We use Oprofile 0.9.3 with Xen patch as our data gather tool. We set CPU counter frequency with 100000. This means that Oprofile will generate a sample for every 100000 occurrences of a specific event such as LLC miss.

## 4. Performance study of single virtual machine system

In this section, we firstly measure the overall performance of single virtual machine system from macro perspective. Then we analyze the micro performance using a micro performance analysis tool - LMbench. After that, we give a correlation analysis based on the experimental discoveries.

**4.1. Macro performance measurement.** The macro performance mainly means the overall performance of a subsystem or whole system. We measure the virtualization overheads of processor subsystem, file subsystem, network subsystem, and java server in three virtual machine monitors which represent three typical virtualization technologies, i.e. operating system-level virtualization, para-virtualization, and full-virtualization respectively.

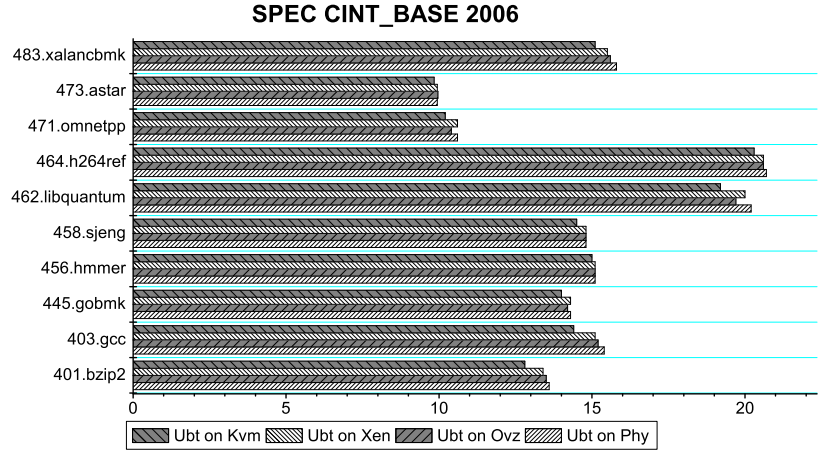


FIGURE 3. The result of CINT2006 in four environments.

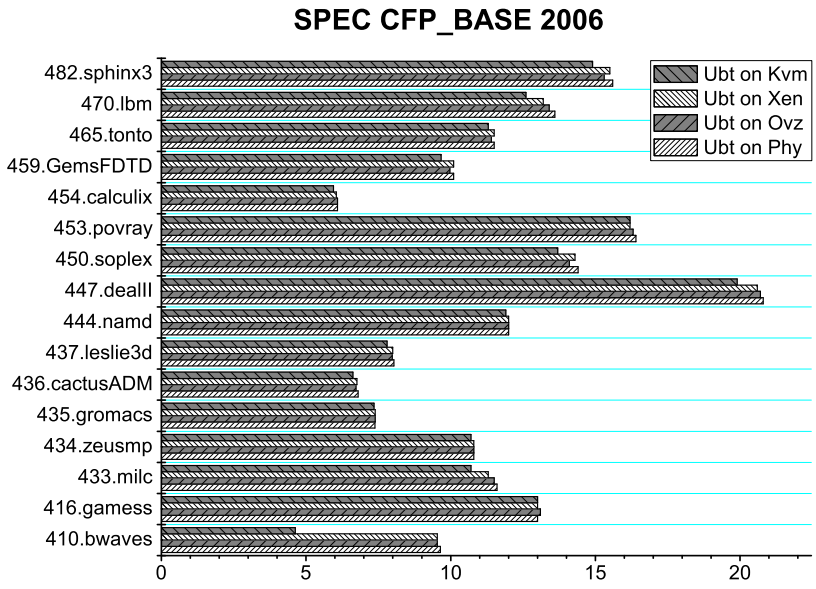


FIGURE 4. The result of CFP2006 in four environments.

**4.1.1. Processor virtualization.** As shown in Figure 3 and Figure 4, KVM displays a low score regardless of CINT2006 or CFP2006 especially on 410.bwaves benchmark (about 50% degradation), while OpenVZ and Xen are almost accordant to native environment. 410.bwaves numerically simulates blast waves in three dimensions, which presumes amount of data exchange. Therefore, KVM spends many time to switch among *guest*, *kernel*, and *user* when updating the nonlinear damped periodic system's matrix data. Although KVM still need to improve its efficiency for computing-intensive workloads, processor virtualization is not already the performance bottleneck of virtualization systems.

**4.1.2. Memory virtualization.** The speed in four environments (KVM, Xen, OpenVZ, and Physical machine) discloses a basic harmony on four memory access operations regardless of integer or float point number as shown in Figure 5. The result of four integer operations is almost the same as four float operations, but the speeds of *Copy* and *Scale* are lower than *Add* and *Triad*, which mainly because *copy* and *scale* involve more data moving. Generally, the presence of virtualization layer doesn't cumber the performance of memory access operations, especially their memory access speed or bandwidth.

**4.1.3. Disk I/O virtualization.** The bandwidth of three disk I/O operations (put_block, rewrite, get_block) in KVM is lowest as shown in Figure 6, because KVM implements its virtualized disk I/O based on QEMU. Xen batches guest domains' I/O requests and accesses the physical device with relative driver. After being moved into physical memory with DMA, the I/O data will be transferred to guest domains by memory page flipping. OpenVZ's second level scheduler, i.e. the Jens Axboe's CFQ I/O scheduler allots all available I/O bandwidth according to each container's I/O priority, so it holds the best block *read* and *write* performance when only hosting one container.

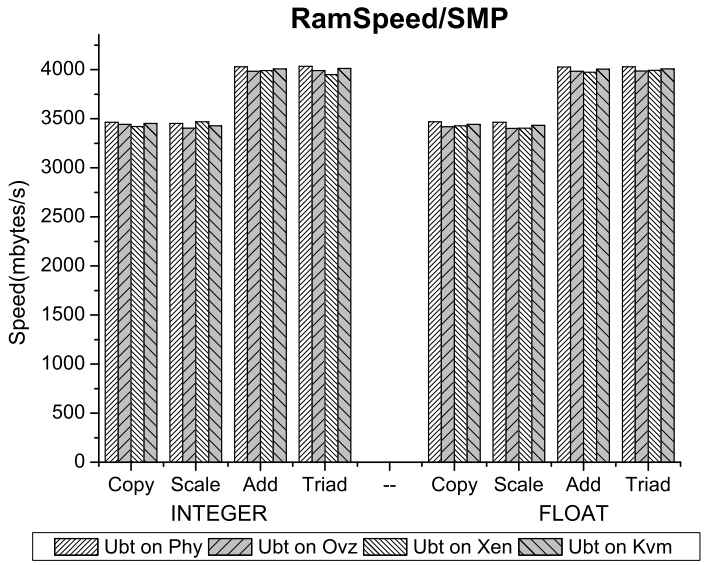


FIGURE 5. The speed of memory access in RAMSMP.

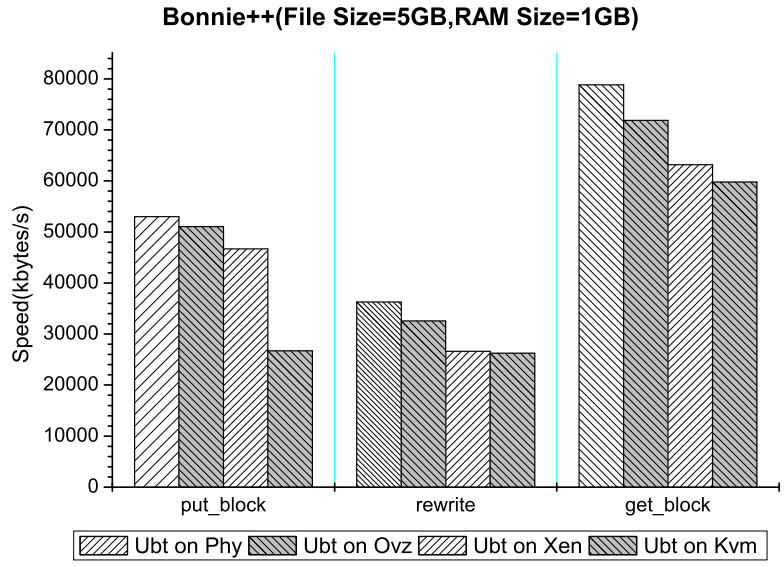


FIGURE 6. The bandwidth of three disk I/O operations in Bonnie++.

**4.1.4. Network I/O virtualization.** The speeds of transmitting and receiving data packets in four environments display an orderly sequence as shown in Figure 7 and Figure 8. OpenVZ has the best performance of **TCP** transmission due to its efficient virtual network device (*venet*). Xen can have a very close transmitting speed to native environment regardless of the packet size because the packet payload is transmitted with a scatter-gather DMA. Different from OpenVZ and Xen, KVM transmits data packets through a network interface simulated by QEMU.

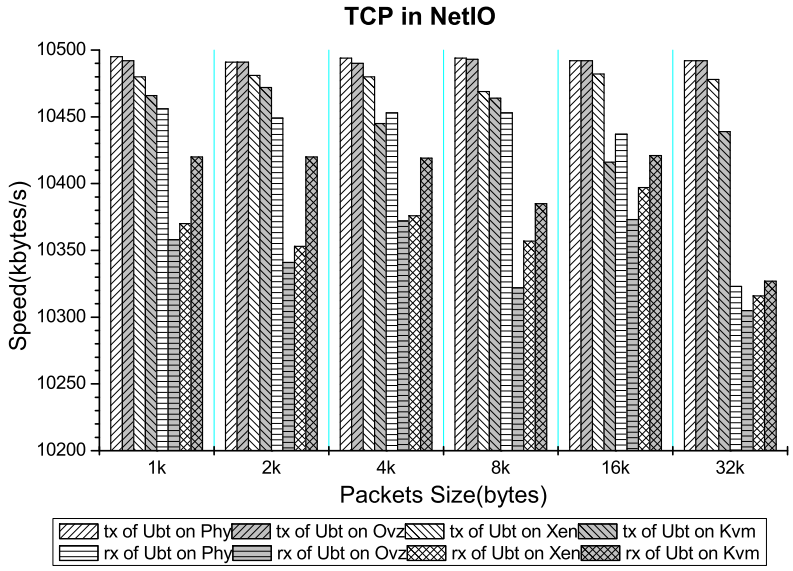


FIGURE 7. The TCP result of NetIO, tx is transmit, and rx is receive.

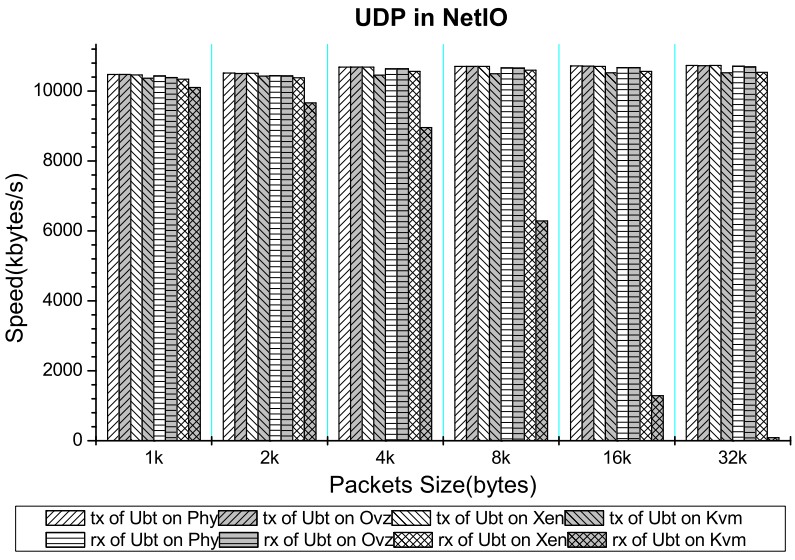


FIGURE 8. The UDP result of NetIO, tx is transmit, and rx is receive.

Therefore, transferring a data packet in KVM is expensive. When receiving a data packet, OpenVZ will cost some time in moving data from the network device to the container. Xen implements the packet transferring in main memory. Consequently, Xen obtains a fast receiving speed.

The case of **UDP** transmission is something different. OpenVZ and Xen have the almost same performance as native environment no matter transmitting or receiving a data packet. KVM gives a declining 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, which implies its inferior efficiency of UDP transmission.

**4.1.5. Java server virtualization.** As for their capabilities of sustaining a virtual machine to act as java server, Xen wins the highest score when the number of warehouses is under 4 as shown in Figure 9. It is because Xen deals with various java transactions using para-virtualization. But OpenVZ keeps a higher score than Xen when the number of warehouse exceeds 3, perhaps thanks to OpenVZ's nicer ability to support more concurrent processes. KVM exposes the worst score owing to its emulating manner to deal with java transactions. However, all their performance falls behind native environment, clarifying the need of further improvement on processing efficiency of these java transactions.

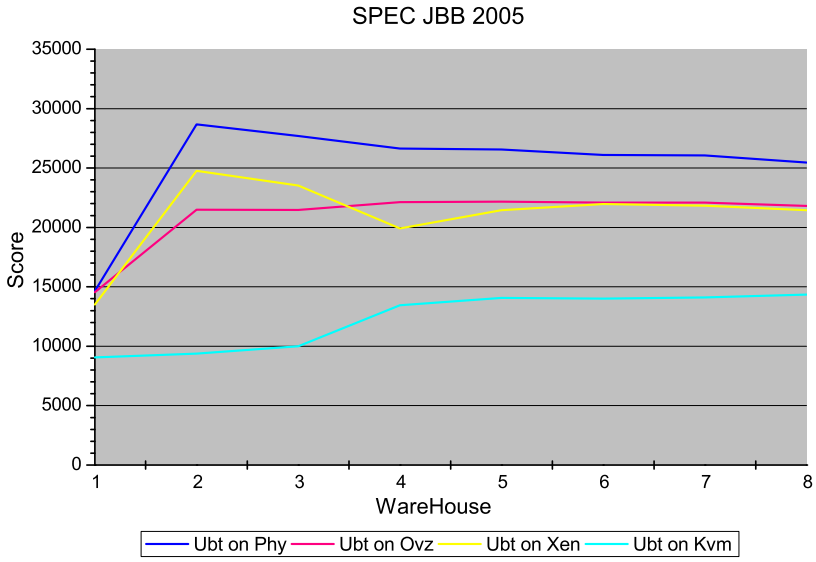


FIGURE 9. The result of SPECjbb2005 test.

**4.2. Micro performance analysis.** The micro-performance mainly means the performance of some specific operations or instructions. We measure the virtualization overhead of some operations like *system call* and *context switch* in three virtual machine monitors. Table 1 shows the results.

**4.2.1. System operations virtualization.** As implementing common system operations with the host's APIs, OpenVZ holds a similar latency as native environment except for process fork. KVM shows a latency close to native environment and better than OpenVZ and Xen in several system calls and signal install, but longer delay than OpenVZ and Xen in process create. As process fork involves memory allocation and hardware page table update, KVM has to wait for available address space and shadow page table update. It is a long trip to remap guest address to host address with shadow page table. Xen discards shadow page table and allows guest domain itself to register a local page table and some common signal or exception handlers with MMU. This enables guest domains to batch hardware page table updates with a single hypercall and less TLB flushing times.

**4.2.2. Context switch.** OpenVZ switches a container's processor context by resorting to the host kernel, so it owns a proximal latency. Xen execute a hypercall at least to change the page table base, which incurs some extra overhead. As KVM needs to remap the shadow page table for context switch, it cost the most time.

TABLE 1. System operations time and context switch latency in $\mu s$

| | Ubt | Ovz | Xen | KVM | C. Swit. | Ubt | Ovz | Xen | KVM |
|---|---|---|---|---|---|---|---|---|---|
| syscall | 0.1164 | 0.1165 | 0.5699 | 0.1179 | 2p0k | 1.01 | 1.75 | 3.91 | 4.87 |
| read | 0.2237 | 0.2848 | 0.9219 | 0.2268 | 2p16k | 1.67 | 2.29 | 4.7 | 6.39 |
| write | 0.2151 | 0.2722 | 0.8714 | 0.2162 | 2p64k | 1.41 | 1.49 | 4.37 | 5.46 |
| stat | 0.9223 | 1.2904 | 1.835 | 0.9461 | 4p0k | 1.29 | 1.78 | 4.43 | 5.55 |
| fstat | 0.2275 | 0.2304 | 0.8257 | 0.2314 | 4p16k | 1.98 | 2.44 | 5.16 | 6.44 |
| open/close | 1.839 | 2.3595 | 4.1215 | 1.8672 | 4p64k | 1.57 | 1.97 | 4.76 | 6.04 |
| sigl insl | 0.2404 | 0.2434 | 0.789 | 0.2487 | 8p0k | 3.81 | 2.04 | 5.41 | 5.87 |
| sigl hndl | 1.5389 | 1.6534 | 3.6856 | 6.872 | 8p16k | 2.28 | 2.65 | 6.17 | 6.77 |
| pipe | 4.7132 | 4.7898 | 14.959 | 12.522 | 8p64k | 1.89 | 2.17 | 5.76 | 6.97 |
| fork+exit | 134.318 | 188.212 | 473.82 | 849 | 16p16k | 2.38 | 2.83 | 6.4 | 7.95 |
| fork+exec | 490.727 | 481.364 | 1279 | 2203.3 | 16p64k | 1.93 | 2.29 | 5.88 | 7.59 |
| fork+sh | 1065 | 1770.3 | 2731 | 4877 | 64p64k | 4.47 | 8.26 | 8.58 | 15.11 |

**4.3. Correlation Analysis.** Performance bottleneck usually means all key factors that block multiple facets of computer systems. Therefore, one needs to correlate all analysis results to identify the performance lagging factors. From the macro-performance data, we can find that the overhead of processor virtualization in relatively mature virtual machine monitors is minimal, but disk I/O appears a large performance loss. Hence, disk I/O should be a performance bottleneck of virtualization systems. From the micro-performance data, we can find that the latencies of process create and context switch in virtualized environment fall behind native environment with a huge degree, which implies two main factors that baffle the performance of virtualization systems. Therefore, we may preliminarily determine hardware page table update, interrupt request and I/O are three main performance bottlenecks for common virtualization systems. As most high-cost operations involve them, it's critical for researcher and developer to optimize the handle mechanism of hardware page table update, interrupt request and I/O.

## 5. Performance study of multiple virtual machine system

In the above section, we have evaluated the performance overheads of single virtual machine system from the virtualization efficiency of processor, memory, disk I/O, network I/O, and java server, and analyzed the performance bottleneck from micro operations or instructions level analysis. While in this section, we evaluate the performance characterization of multiple virtual machine system (which is more typical in real scenarios) from the virtual cluster efficiency of floating point computation, memory bandwidth, data transfer rate, and network communication. We create two 16-node virtual clusters with MPI environment for the running of HPC applications. Combined with the micro profiling data analysis, some potential performance characterization comes out.

**5.1. Macro performance measurement.** To simulate the multiple virtual machine environment (HPC environment), we create two 16-node virtual clusters based on Xen virtual machine monitor, one is for para-virtualization, the other is for full virtualization. Figure 10-17 show the detailed macro results using HPCC benchmark from different aspects with various running processor number (i.e. thread number). We classify the results into computational performance, memory performance, data transfer rate, and communication performance.

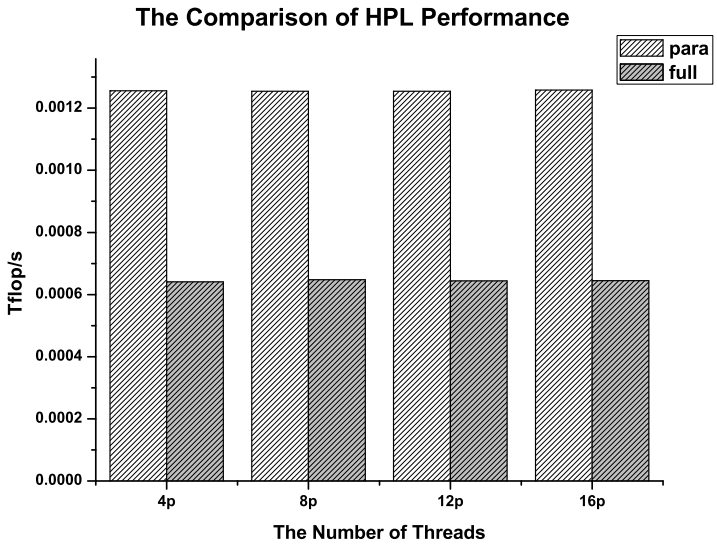


FIGURE 10. The performance comparison of HPL in a 16-node para-virtualized and full virtualized cluster

**5.1.1. Computational Performance.** Figure 10-12 illustrate the floating-point performance. From the figures, we found that the performance of full virtualized cluster is worse than para-virtualized cluster at a degradation of 48.98% in HPL testing, 19.07% in DGEMM testing, and 24.38% in FFT testing. It indicates that the computational applications are sensitive to the virtualization in different degrees. What's more, from the DGEMM testing, we found that the **star** performance decreases as the processor number increases while **single** performance keeps stable. It is because in **star** mode all the processors run separate independent copies of the benchmark which means more system resources will be consumed as the processor number increases that affecting the performance. For FFT benchmark, it is obvious that when running in **mpi** mode, full virtualization gets worse performance than para-virtualization due to the huge overheads of MPI communication. The MPI communication overheads are the major factors affecting the floating-point performance in full virtualized cluster.

**5.1.2. Memory Performance.** Figure 13 and 14 show the memory performance of two virtual clusters. The RandomAccess presents a similar phenomenon with the FFT performance that running in the **mpi** mode, full virtualization obtains very poor performance due to the huge MPI communication overheads. In order to save space, we only present the STREAM performance running in **single** mode. We

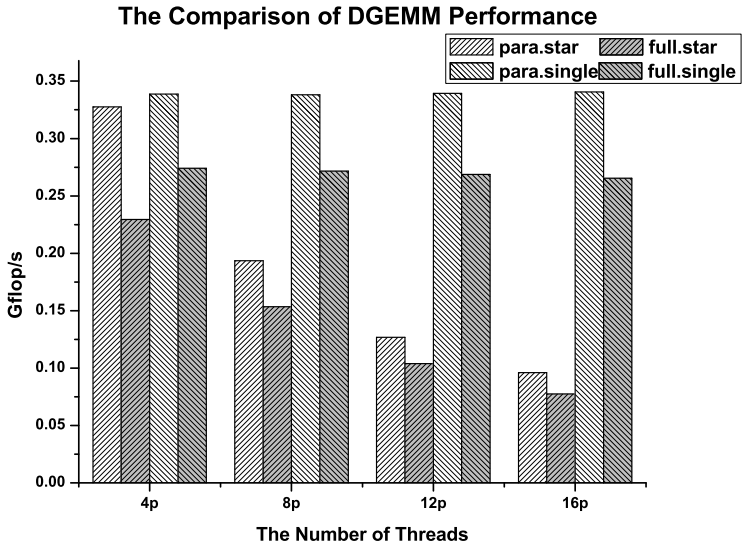


FIGURE 11. The performance comparison of DGEMM in a 16-node para-virtualized and full virtualized cluster

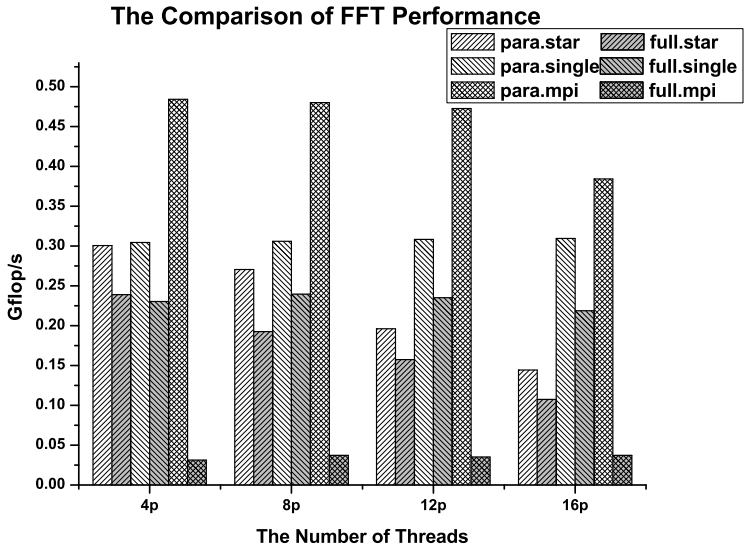


FIGURE 12. The performance comparison of FFT in a 16-node para-virtualized and full virtualized cluster

find the performance of full virtualization is very close to the para-virtualization which means the memory virtualization efficiency is not the bottleneck affecting the performance of HPC applications.

**5.1.3. Data Transfer Rate.** In the PTRANS testing as shown in Figure 15, full virtualization pursues poor performance obviously. It is because, the PTRANS

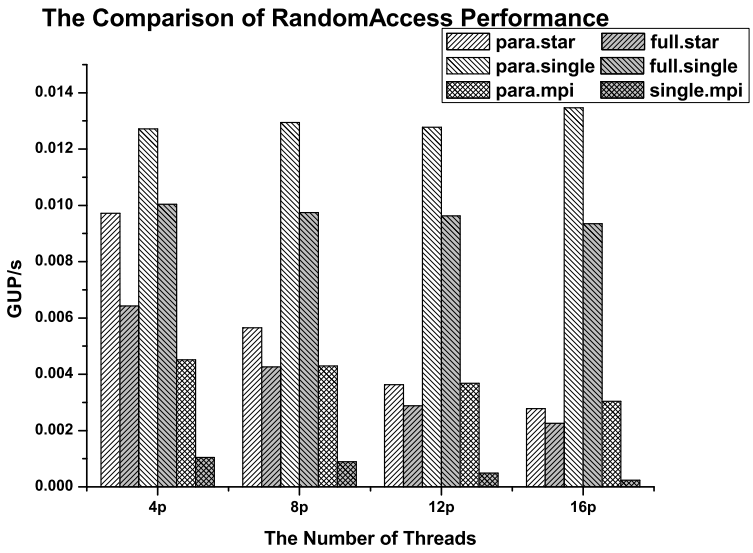


FIGURE 13. The performance comparison of RandomAccess in a 16-node para-virtualized and full virtualized cluster

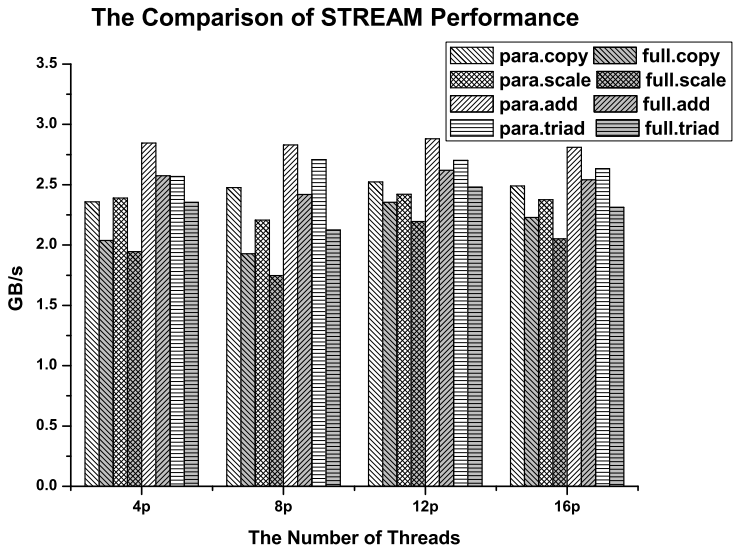


FIGURE 14. The performance comparison of STREAM in a 16-node para-virtualized and full virtualized cluster

exercises the communications where processor pairs communicate with each other simultaneously and leads to significant overheads. It again indicates the communication overheads including the data transferring is a bottleneck in full virtualized cluster.

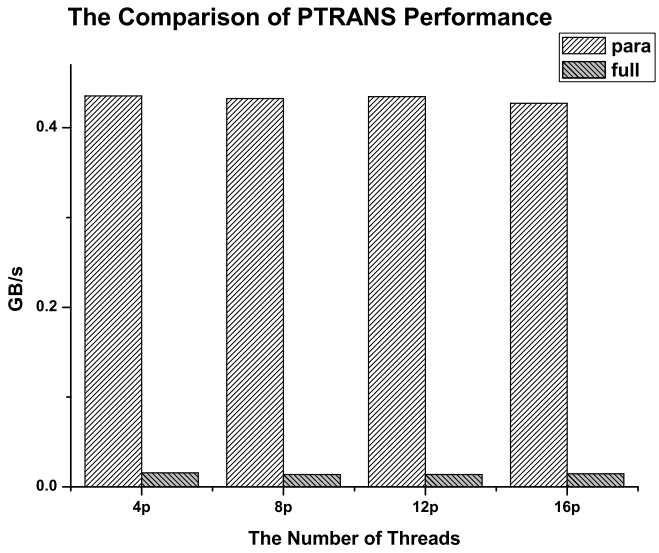


FIGURE 15. The performance comparison of PTRANS in a 16-node para-virtualized and full virtualized cluster

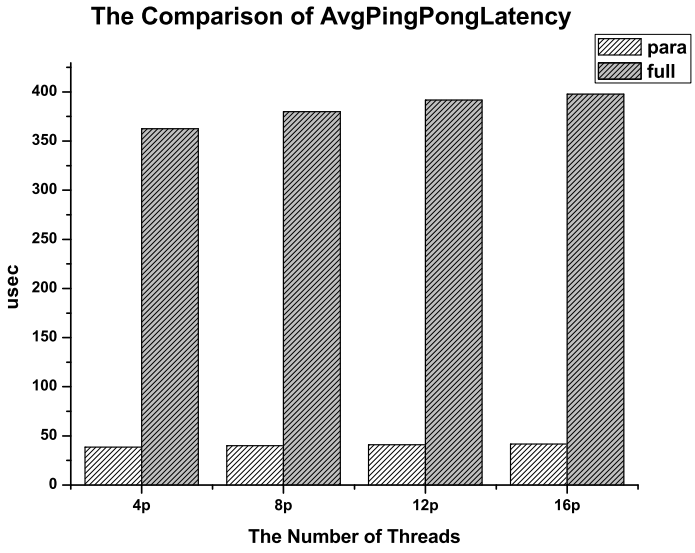


FIGURE 16. The performance comparison of average network latency in a 16-node para-virtualized and full virtualized cluster

**5.1.4. Communication Performance.** Figure 16 and 17 show the network communication performance for both para-virtualized cluster and full virtualized cluster. The latency of full virtualized cluster is 9.38 times to para-virtualized cluster, and the bandwidth of full virtualized cluster is only 3.22% of the para-virtualized

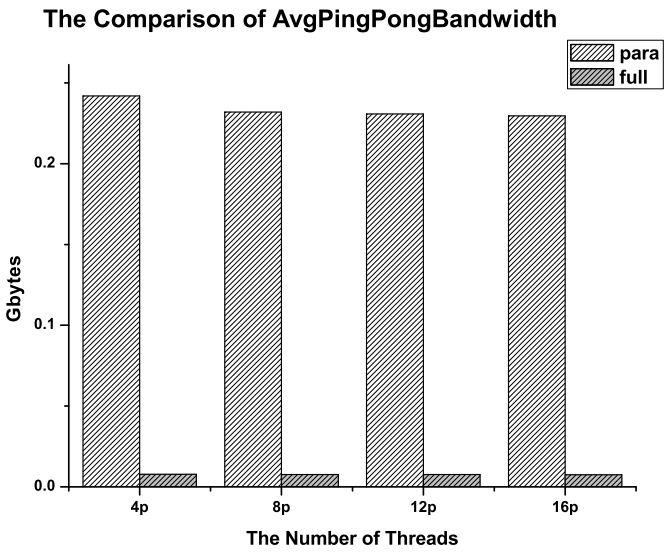


FIGURE 17. The performance comparison of average network bandwidth in a 16-node para-virtualized and full virtualized cluster

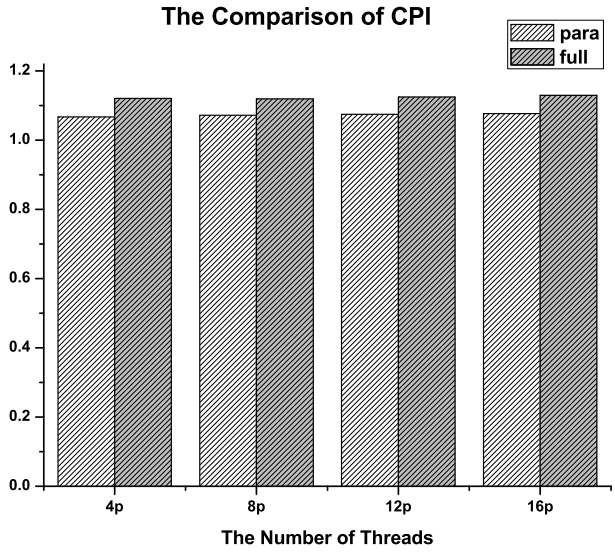


FIGURE 18. The comparison of CPI in a 16-node para-virtualized and full virtualized cluster

cluster since the network communication overheads of full virtualized cluster is too significant.

**5.2. Micro performance analysis.** In order to analyze the performance bottleneck from micro perspective, we collect the profiling data from several hardware events using Oprofile/Xenoprof toolkit. Figure 18 and 19 show the CPI (Cycles

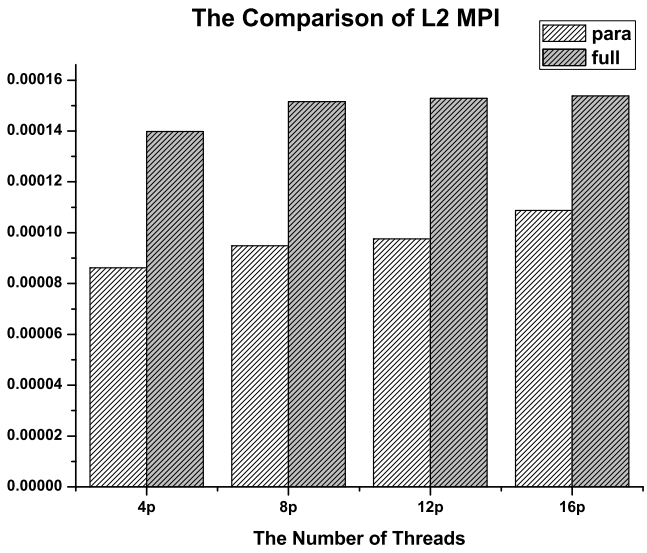


FIGURE 19. The comparison of L2 MPI in a 16-node para-virtualized and full virtualized cluster

per Instruction) and L2 MPI (L2 cache Misses per Instruction). It is obvious that the L2 cache misses is the main cause affecting the virtualization efficiency between para-virtualized and full virtualized cluster. What's more, the increase in the number of threads will lead to the increase of L2 cache miss rate due to the competition of shared resources.

**5.3. Correlation Analysis.** From the macro benchmark evaluation, we have already found that the multiple virtual machine system loses a part of performance when running HPC applications. It is because the communication operations is very common in HPC applications, such MPI communication, data transmission, and network communication, which will result in huge communication overheads and affect the performance of multiple virtual machine system. What's more, the para-virtualized cluster achieves better communication performance than full virtualized cluster. It is because, in the para-virtulized network I/O mechanism (Front-End/Back-End mechanism), the network interfaces can DMA the packet into a buffer, and read the packet header and decide where to send the data. With the shared memory, the network interfaces can directly copy the data or remap the page, either of which is particularly cheap. This mechanism makes the inter-domain communication with low overheads without via the network interface and achieves better performance that no Domain0 interactions are required beyond the initial setup. However, in the emulated network I/O mechanism of full virtualized cluster, there are frequent traps into the VMM when the packet requests arrive and sacrifice a lot of performance. Different from the macro analysis, we have also investigate the CPI (Cycles per Instruction) and L2 MPI (L2 cache Misses per Instruction) from the micro perspective to help explain the performance penalty of multiple virtual machine system. The obvious increase of L2 cache miss rate is the main cause affecting the virtualization efficiency between para-virtualized and full virtualized cluster.

## 6. Related work

Many research has analyzed the performance overheads of virtualization in single virtual machine [10, 12, 13] using traditional benchmarks focusing on CPU, memory, I/O, and network. Barham et al. [10] gave a full-scale introduction to Xen and measures its performance against VMware, User-Mode Linux and Base Linux with SPECCPU2000, OSDB, DBench and SPECWeb99. Clark et al. [12] reproduced the results from [10] 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. Che et al. [23] studied an initial comparison of Xen and KVM. Differ from the above work, we present a combinative evaluation of the performance issues of single virtual machine system that investigates both macro and micro performance to explore the overheads and bottleneck.

In multiple virtual machine system, some researchers evaluated the performance of server consolidate scenario [1, 2, 16]. Ye et al. [24] analyzed the performance of virtual cluster and presented a performance model. The work described in [18, 25] evaluate the HPC performance impact when running MPI codes in Xen para-virtualized environment. Work in [4] investigated the performance and management overheads of VM-based HPC framework using VMM bypass I/O scheme and InfiniBand. Work in [26, 27] compared performance impacts of different virtualization technologies on HPC applications, including para-virtualization, full virtualization and OS level virtualization. However they only evaluate the macro performance using benchmarks and not refer to a deep analysis into architecture characterization from the micro perspective.

Menon et al. [13] firstly used Xenoprof to diagnose the performance overheads in Xen. They focused on the network performance when running applications in the VM, and used the information extracted by Xenoprof to uncover bugs and optimize the performance of Xen. Tikotekar et al. [28] used a real scientific application to evaluate the virtualization performance and also used the Oprofile tool to better understand the overheads of virtualization. Ye et al. [29] developed a automatic and configurable benchmarking tool - vTestkit for the performance evaluation in virtualization environment. Recently, Kundu et al. [30] modeled the virtualization performance by collecting CPU, memory and I/O parameters to train the artificial neural network model.

## 7. Conclusion and future work

Our study was motivated by the interests in using virtualization technology in both single virtual machine system and multiple virtual machine system. However, the performance overheads of various virtualization scenarios are not yet clear enough due to the lack of micro analysis in traditional unitary evaluation method which measures the performance only by running macro benchmarks. In this paper, we firstly study the component virtualization overhead of single virtual machine system (such as the virtualization efficiency of processor, memory, disk I/O, network, etc) by comparing the performance of different virtualization technologies. Then we create two 16-node virtual clusters, and do a comprehensive performance evaluation of multiple virtual machine system to investigate the virtualization efficiency for HPC applications. Besides, we also investigate the micro performance for both single virtual machine and multiple virtual machine system. For single virtual machine system, we investigate some specific operations such as *system call* and *context switch*. While for multiple virtual machine system, we analyze the profiling

data from the hardware events when running HPC applications in virtualization environment using Oprofile/Xenoprof toolkit.

Experimental results show that: 1) Disk I/O is a performance bottleneck and the latencies of *process create* and *context switch* are two main factors that baffle the performance of single virtual machine system; 2) The optimized network I/O processing mechanism in Xen's para-virtualized cluster can achieve better efficiency compared to full virtualized cluster since the Front-End/Back-End network I/O mechanism of para-virtualization can cause fewer traps than emulated I/O mechanism of full virtualization which performs better performance in inter-domain communication; 3) Different forms of communication overheads (MPI communication, network communication, etc) in multiple virtual machine system are the main bottleneck for full virtualized cluster, which cause huge L2 cache miss rate.

Future work will include optimizing the virtualization performance for both single virtual machine and multiple virtual machine system (especially the disk I/O performance and communication performance), and analyzing the performance using modeling technology.

## Acknowledgments

The author thanks the anonymous reviewers for their comments and suggestions on the paper. This is an extension work of the conference paper published in the 2009 international conference on high performance computing and applications (HPCA'09).

## References

[1] M. Marty, M. Hill, Virtual hierarchies to support server consolidation, in: Proceedings of the 34th annual International Symposium on Computer Architecture, 2007, pp. 46–56.

[2] P. Apparao, R. Iyer, X. Zhang, D. Newell, T. Adelmeyer, Characterization & analysis of a server consolidation benchmark, in: Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2008, pp. 21–30.

[3] M. F. Mergen, V. Uhlig, O. Krieger, J. Xenidis, Virtualization for high-performance computing, SIGOPS Oper. Syst. Rev. 40 (2) (2006) 8–11.

[4] W. Huang, J. Liu, B. Abali, D. K. Panda, A case for high performance computing with virtual machines, in: Proceedings of the 20th annual International Conference on Supercomputing, 2006, pp. 125–134.

[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[6] K. Ye, D. Huang, X. Jiang, H. Chen, S. Wu, Virtual machine based energy-efficient data center architecture for cloud computing: a performance perspective, in: Proceedings of the IEEE/ACM International Conference on Green Computing and Communications, 2010, pp. 171–178.

[7] M. Rosenblum, T. Garfinkel, Virtual machine monitors: current technology and future trends, Computer 38 (5) (2005) 39–47.

[8] C. Waldspurger, Memory resource management in VMware ESX server, ACM SIGOPS Operating Systems Review 36 (SI) (2002) 181–194.

[9] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux virtual machine monitor, in: Proceedings of the Linux Symposium, Vol. 1, 2007, pp. 225–230.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM SIGOPS Operating Systems Review 37 (5) (2003) 164–177.

[11] OpenVZ: server virtualization open source project, http://openvz.org.

[12] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, J. Matthews, Xen and the art of repeated research, USENIX annual Technical Conference (2004) 135–144.

[13] A. Menon, J. Santos, Y. Turner, G. Janakiraman, W. Zwaenepoel, Diagnosing performance overheads in the Xen virtual machine environment, in: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, 2005, pp. 13–23.

[14] L. Cherkasova, R. Gardner, Measuring CPU overhead for I/O processing in the Xen virtual machine monitor, in: Proceedings of the annual Conference on USENIX Annual Technical Conference, 2005, pp. 24–24.
[15] P. Apparao, S. Makineni, D. Newell, Characterization of network processing overheads in Xen, in: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, 2006.
[16] K. Ye, X. Jiang, D. Ye, D. Huang, Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server, in: 12th IEEE International Conference on High Performance Computing and Communications, 2010, pp. 281–288.
[17] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, S. Scott, An analysis of HPC benchmarks in virtual machine environments, in: Proceedings of 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing, 2009, pp. 63–71.
[18] L. Youseff, R. Wolski, B. Gorda, C. Krintz, Evaluating the performance impact of Xen on MPI and process execution for HPC systems, in: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed computing, 2006, p. 1.
[19] A. Ranadive, M. Kesavan, A. Gavrilovska, K. Schwan, Performance implications of virtualizing multicore cluster machines, in: Proceedings of the 2nd Workshop on System-level Virtualization for High Performance Computing, 2008, pp. 1–8.
[20] Xenoprof: System-wide profiler for Xen VM, http://xenoprof.sourceforge.net/.
[21] HPC Challenge benchmark, http://icl.cs.utk.edu/hpcc.
[22] LMbench: tools for performance analysis, http://lmbench.sourceforge.net/.
[23] J. Che, Q. He, Q. Gao, D. Huang, Performance measuring and comparing of virtual machine monitors, in: Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Vol. 2, 2009, pp. 381–386.
[24] K. Ye, X. Jiang, S. Chen, D. Huang, B. Wang, Analyzing and modeling the performance in Xen-based virtual cluster environment, in: Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications, 2010, pp. 273–280.
[25] L. Youseff, R. Wolski, B. Gorda, C. Krintz, Paravirtualization for HPC systems, in: Proceedings of the ISPA Workshop on Xen in HPC Cluster and Grid Computing Environments, 2006, pp. 474–486.
[26] W. Emeneker, D. Stanzione, HPC cluster readiness of Xen and User Mode Linux, in: Proceedings of the IEEE International Conference on Cluster Computing, 2006, pp. 1–8.
[27] V. Chaudhary, M. Cha, J. Walters, S. Guercio, S. Gallo, A comparison of virtualization technologies for HPC, in: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications, 2008, pp. 861–868.
[28] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, S. L. Scott, A. M. Filippi, Effects of virtualization on a scientific application running a hyperspectral radiative transfer code on virtual machines, in: Proceedings of the 2nd Workshop on System-level Virtualization for High Performance Computing, 2008, pp. 16–23.
[29] K. Ye, J. Che, X. Jiang, J. Chen, X. Li, vTestkit: aperformance benchmarking framework for virtualization environments, in: Proceedings of the Fifth ChinaGrid Annual Conference, 2010, pp. 130–136.
[30] S. Kundu, R. Rangaswami, K. Dutta, M. Zhao, Application performance modeling in a virtualized environment, in: Proceedings of the 16th International Symposium on High Performance Computer Architecture, 2010, pp. 1–10.

College of Computer Science, Zhejiang University, Hangzhou 310027, China
*E-mail*: yekejiang@zju.edu.cn

Information & Network Security Laboratory of State Grid Corporation, State Grid Electric Power Research Institute, Nanjing 210003, China
*E-mail*: chejianhua@zju.edu.cn

College of Computer Science, Zhejiang University, Hangzhou 310027, China
*E-mail*: hqm@zju.edu.cn and tossboy.hdw@zju.edu.cn and jiangxh@zju.edu.cn