Improving Virtualization Performance and Scalability with Advanced Hardware Accelerations

Yaozu Dong*, Xudong Zheng*, Xiantao Zhang*, Jinquan Dai*, Jianhui Li*, Xin Li*, Gang Zhai*，Haibing Guan#

*Intel China Software Center, Shanghai, China
{eddie.dong, xudong.zheng, xiantao.zhang, jason.dai, jian.hui.li, xin.li, edwin.zhai}@intel.com

#Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, China
hbguan@sjtu.edu.cn

Abstract* — Many advanced hardware accelerations for virtualization, such as Pause Loop Exit (PLE), Extended Page Table (EPT), and Single Root I/O Virtualization (SR-IOV), have been introduced recently to improve the virtualization performance and scalability. In this paper, we share our experience with the performance and scalability issues of virtualization, especially those brought by the modern, multi-core and/or overcommitted systems. We then describe our work on the implementation and optimizations of the advanced hardware acceleration support in the latest version of Xen. Finally, we present performance evaluations and characterizations of these hardware accelerations, using both micro-benchmarks and a server consolidation benchmark (vConsolidate). The experimental results demonstrate an up to 77% improvement with these hardware accelerations, 49% of which is due to EPT and another 28% due to SR-IOV.

Keywords: Virtualization, Virtual Machine; PLE, EPT, SR-IOV

I. INTRODUCTION

Virtualization is the key enabling technology for server consolidation and elastic resource management in the cloud environment [1]. However, virtualization not only brings additional performance overheads (in CPU and memory, as well as I/O), but also introduces additional scalability issues for multi-core and/or overcommitted systems.

Full virtualization (with binary translation) [25] and paravirtualization [6][9][28] were proposed when there were not sufficient hardware supports. Full virtualization supports unmodified guest OSes, but requires the OS binary to be translated at run time. Paravirtualization improves virtualization performance with moderate modifications to OS codes, which, however, is difficult to apply to proprietary OSes. Basic hardware virtualization technologies, such as Intel Virtualization technology [23], are introduced to eliminate the need for binary translation in full virtualizations and to improve the virtualization performance.

Unfortunately, these traditional approaches are inefficient in or incapable of addressing the virtualization performance and scalability issues, especially those brought by the modern, multi-core and/or overcommitted systems. To further improve the performance and scalability of virtualization, a lot of advanced hardware accelerations are introduced, such as Pause Loop Exit (PLE) [12], Extended Page Table (EPT) [12], and Single Root I/O Virtualization (SR-IOV) [18].

In this paper, we present our work of supporting these advanced hardware accelerations in the latest version of Xen [6], sharing our experience on how these accelerations are used and what additional software optimizations are required to improve the virtualization efficiency. The major contributions of our work are as follows.

- Summary of our experience on the performance and scalability issues of virtualization, especially those brought by the modern, multi-core and/or overcommitted systems.
- Presentation of our work on the implementation and optimizations of advanced hardware acceleration support in the latest version of Xen, including hybrid interrupt vectoring, improving page table locality with large memory pages and confined Guest Page Table, and empirical PLE configuration.
- Performance evaluations and characterizations using not only micro-benchmarks, but also a server consolidation benchmark (vConsolidate) [3].

The results show an up to 77% performance improvement (49% of which due to EPT and another 28% due to SR-IOV) in the server consolidation benchmark, and an up to 14% improvement, due to PLE in the micro-benchmarks.

The rest of the paper is organized as follows. Section II provides an overview of the virtualization challenges (brought by the modern, multi-core and/or overcommitted systems). Section III describes the advanced hardware accelerations for virtualization and how they are supported in Xen. Section IV describes the required software optimizations when using the hardware accelerations. Section V presents the characterizations and improvements with the advanced hardware accelerations for vitalization, using both micro-benchmarks and a server consolidation benchmark. Section VI discusses the related work and finally section VII concludes the paper.

II. Challenges of Virtualization Performance and Scalability

In this section, we first provide an overview of the virtualization architecture of Xen, and then describe the challenges of virtualization performance and scalability (especially those brought by the modern, multi-core and/or overcommitted systems).

A. *Xen Architecture*

Figure 1 shows the virtualization architecture implemented in Xen, based on Xen's Hardware Virtual Machine (HVM) implementation with basic hardware supports such as Intel Virtualization Technology [23].

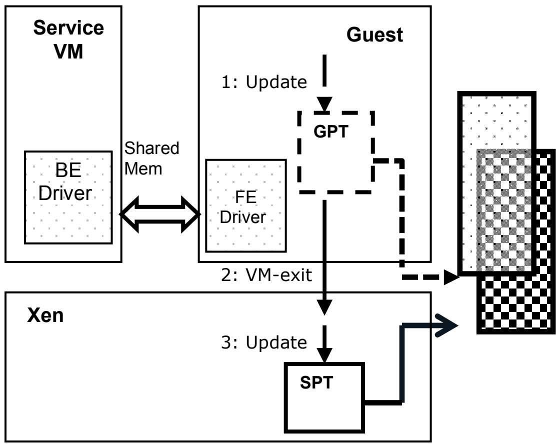

Figure 1. Xen HVM architecture

Shadow Page Table (SPT) is adopted in Xen HVM to virtualize memory. SPT is fully controlled by Xen hypervisor to manage the translation from guest linear memory address to host physical memory address. The guest VM only updates the Guest Page Table (GPT), which is write-protected so that the update of the GPT entry by the guest can be trapped-and-emulated (through the VM-exit [23] event handler in the hypervisor). Consequently, it can be guaranteed that the SPT is always in synchronization with the GPT.

Since full I/O virtualization incurs extremely high overheads due to excessive hypervisor interventions, I/O paravirtualization (or PV I/O) [9] is adopted in Xen HVM. However, in PV I/O the hypervisor still needs to intervene in an I/O request from the frontend in the guest VM and activate the backend driver in the service VM (that is domain 0 in Xen), which then responds to the request (such as receiving a packet from device and copying the packet to the guest buffer).

B. *Virtualization challenges*

Unfortunately, both SPT and PV I/O bring additional costs for virtualization. In particular, SPT suffers from excessive VM-exit (caused by the write-protection on GPT). The performance of PV I/O is limited by the interaction between the frontend and backend drivers and the packet copy in the service VM.

In addition, multi-processor (MP) guest OS and overcommitted systems bring new challenges to the virtualization performance and scalability. For instance, MP kernel usually has very heavy usage of locks to ensure atomic accesses to shared data structure. In a system where virtualized CPUs are overcommitted, the impacts of lock contentions are even worse because the VCPU that currently holds a lock may happen to be preempted by the hypervisor [24][29].

MP guest also brings additional synchronization overheads due to lock contentions on Shadow Page Table accesses. The hypervisor needs to track the page table for each VCPU. Because MP guest shares the page table entries for the kernel space among VCPUs, the corresponding SPT entries are also shared among these VCPUs to avoid the synchronization effort, and consequently the hypervisor needs to first acquire a lock before it updates the SPT.

Lastly, PV I/O has serious scalability issue in an overcommitted system [14][16][20][22]. As the guest VM count increases, multiple frontend drivers will request services from a single backend driver concurrently, which becomes the scalability bottleneck (such as saturating the VCPU of the service VM due to the packet copy).

III. Advanced Hardware Accelerations

In this section, we describe the advanced hardware accelerations and how we extended the Xen architecture to support them, addressing the performance and scalability challenges of virtualization. Figure 2 shows the architecture of the latest version of Xen that supports these hardware accelerations.

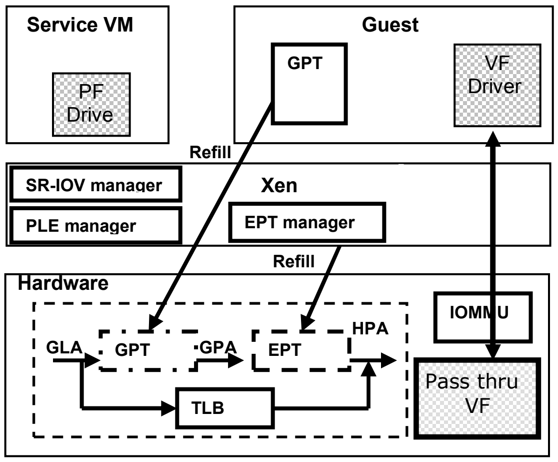

Figure 2. Extended Xen architecture using advanced hardware accelerations

A. *Pause Loop Exit*

Pause Loop Exit (PLE) technology provides hardware assistance to detect guest spin-lock. Modern CPU employs PAUSE instruction as spin loop hint to the processor. However, the PAUSE instruction is also widely used in the OS kernel for other purposes (such as in a delay loop). PLE technology enables VMM to detect spin-lock by monitoring the execution of PAUSE instructions through the PLE_Gap and PLE_Window values. PLE_Gap is an upper bound on the cycle count between two successive executions of PAUSE in a loop. The CPU detects a spin-lock if it identifies a series of PAUSE instruction executions and the latency between two successive executions is within the PLE_Gap value. PLE_Window is an upper bound on the amount of time a VCPU is allowed to execute in a spin-loop. If a spin loop execution time surpasses the PLE_Window value, the CPU will invoke a VM-exit to notify the hypervisor.

The PLE manager in Xen yields its CPU quantum when receiving a VM-exit indicating a long-waiting spin loop. In addition, the PLE manager is responsible for setting the PLE_Window and PLE_Gap values.

B. *Extended Page Table*

The Extended Page Table (EPT), which implements a two-dimensional page table [4], is proposed to eliminate the excessive VM-exit and lock contentions in Shadow Page Table. With the EPT support, the Guest Page Table (GPT) contains the mapping from the guest linear memory address (GLA) to guest physical address (GPA), while the EPT contains the mapping from guest physical address to host physical address (HPA), as illustrated in Figure 3.

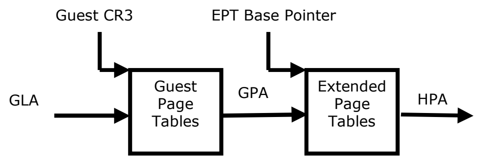

Figure 3. Extended Page Table (EPT)

The EPT manager in XEN maintains an EPT table per guest, which works with the EPT hardware to manage the translation from GPA to HPA, as shown in Figure 2. The guest in EPT gets back the ownership of its page tables, and consequently EPT can completely eliminate the need for the write-protection on Guest Page Table accesses and avoids altogether the excessive VM-exit and lock contentions.

Address translation in the EPT is accelerated by paging-structure caches and Translation Look-aside Buffer (TLB). With the EPT, hardware walks both the guest page table and the EPT, fills the paging-structure caches and forms a merged TLB, translating from the GLA to the HPA directly, as shown in Figure 2.

C. *SR-IOV*

SR-IOV proposes a set of hardware enhancements to the PCIe device [18]. An SR-IOV-capable device has single or multiple Physical Functions (PFs). Each PF can create multiple "light-weight" instances of PCI function entities, known as Virtual Functions (VFs), which are configured and managed by the PF. Each VF can be assigned to a guest for direct access, but still shares major device resources, to achieve both resource sharing and high performance (offloading memory protection and address translation to IOMMU).

We implemented a generic virtualization framework for SR-IOV devices, as shown in Figure 2. The PF driver running in the service VM is responsible for configuring and managing the VFs. On the other hand, the VF driver running in the guest has direct access to the VF runtime resource. The SR-IOV manager receives an interruption from the I/O device, and redirects it to the VF driver in the guest, which can then access the associated data directly from the VF resources. In particular, our implementation with the SR-IOV-capable NIC eliminates the well-known packet classification, address translation and copy overhead, which has demonstrated a 2.63X (from 3.6Gbps to 9.48Gbps) performance improvement over PV NIC [8].

IV. SOFTWARE OPTIMIZATIONS

In this section, we share our experience with these advanced hardware accelerations, and describe what additional software optimizations are required to improve the virtualization efficiency when using the hardware accelerations.

A. *Hybrid Interrupt Vectoring*

The scalability of SR-IOV suffers from the shortage of interrupt vectors. Each VF interrupt source in Xen SR-IOV needs to own a dedicated vector to avoid interrupt sharing. However, the number of interrupt vectors in the x86 architecture is limited to 256, and some of them are reserved or pre-allocated by system. Consequently, the number of interrupt vectors available for VFs is only around 200. However, each VF in an SR-IOV-capable device (such as Intel 82599 NIC [13]) may consume 3 interrupt sources. Conseqeuntly, Xen can only accommodate about 66 VFs before the vectors are exhausted.

To address this issue, we have implemented per-PCPU vectoring in the latest version of Xen. In per-PCPU vectoring, an interrupt is represented using both a PCPU ID and a vector number. When the interrupt is received, hypervisor remaps the physical interrupt to guest interrupt with a per-PCPU remap table before it fires the guest interrupt to the mapped VCPU. Consequently, total number of interrupt vectors that can be accommodated in the system can be increased by multiple-fold, which greatly improves the SR-IOV scalability.

However, in per-PCPU vectoring, I/O APIC (Advanced Programmable Interrupt Controller) may generate redundant messages for level-triggered interrupts. This is because after delivering a level-triggered interrupt message, the I/O APIC will suppress further delivering of the same interrupt until an End of Interrupt (EOI) message is issued by the CPU. However, the EOI message is not tagged with the PCPU ID (due to the current hardware implementations). Consequently, when an EOI message is received, the I/O APIC will stop the suppression of all the interrupts with the matching vector number, even though the EOI message may be issued by a completely different PCPU. So, the I/O APIC may generate redundant interrupt messages for level-triggered interrupts.

Our hybrid vectoring mechanism resolves this issue by using global vectoring for level-triggered interrupts, which are the minority of interrupt sources due to the limitation of I/O APIC pin #, and per-CPU vectoring for edge triggered interrupts, which are the majority of interrupt sources in modern platform (Message Signaled Interrupts fall into this category).

B. *Improving Page Table Locality*

With EPT, hardware needs to walk both the Guest Page Table and EPT to fetch both table entries, to form a merged TLB, translating from the guest linear memory address to the host physical address directly. Unfortunately, this introduces additional cycles to walk the EPT table to refill a TLB entry. In particular, a single 4-level Guest Page Table walk invokes 5 rounds of the 4-level EPT walk, one for each guest page entry and an additional one for the final translation of the guest physical address of the datum itself [4].

The additional EPT walk may cost non-trivial CPU cycles if cache miss happens frequently during the page table walk, and therefore, it is critical to improve the locality of the TLB entries, EPT paging-structure caches, and memory caches. We have implemented large memory pages in Xen by allocating the physical memory for each guest in 2MB chunks. Consequently, the last level EPT page table entry (EPTE) can be completely by-passed in EPT walk. In addition, large memory pages help reduce TLB pressures because each TLB entry can now cover more address space.

Furthermore, guest OSes could be paravirtualized to allocate the memory pages for the Guest Page Table from one or more contiguous 2MB physical memory spaces (that is, confining Guest Page Table in one or more 2MB working sets for each guest process), which can further improve the cache locality of the EPT walk (likely hitting the same EPT entries for all Guest Page Table walk).

C. *Empirical PLE Configurations*

The configuration of PLE_Gap and PLE_Window in PLE_manager is critical for identifying guest spin-lock and generating VM-exit events. The VM-exit event helps the hypervisor to perform better scheduling. However, overly frequent VM-exit can also introduce additional virtualization overheads.

We have used an empirical approach to determine these two values. Figure 4 shows the lock acquisition time in Kernel Build with a 16 VCPUs HVM guest running on top of 16 PCPUs. There are only 0.5% of all spin-locks that may wait more than 2048 cycles. Therefore, we choose to set PLE_Window (the maximum cycles a VCPU is allowed to execute a spin-lock loop continuously) to 2048 cycles. PLE_Gap is not sensitive to performance as if it is bigger than a certain threshold as shown in Figure 5. We set PLE_Gap to a moderate value (128 cycles), that is, if two successive executions of the PAUSE instruction occur within 128 cycles, they are considered to be from the same spin-lock loop.

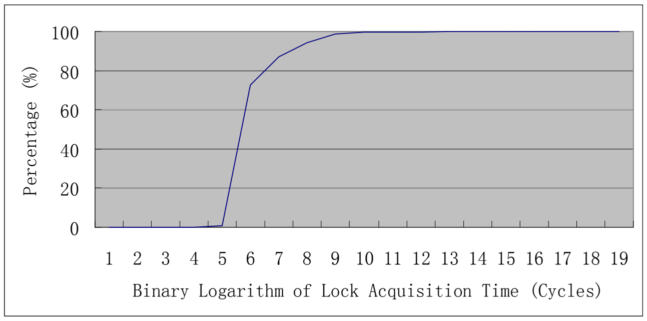

Figure 4. Lock acquisition time in Kernel Build

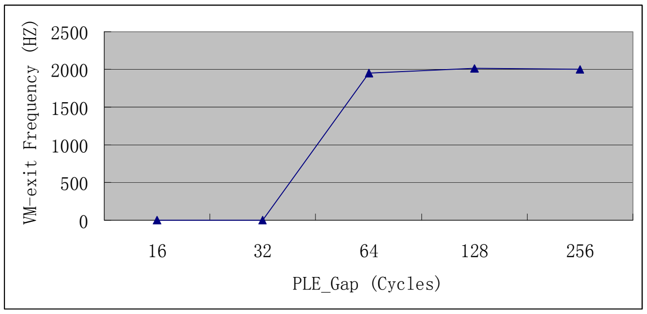

Figure 5. VM-exit frequency of PAUSE instruction in Kernel Build

V. EVALUATIONS

In this section, we present the characterizations and improvements of virtualization and scalability, for the advanced hardware accelerations and corresponding software optimizations. Our experiment runs a server consolidation benchmark (vConsolidate), as well as micro-benchmarks on an Intel Xeon server system. The detailed configurations of the system used in our experiment are shown in Table I.

TABLE I. SYSTEM CONFIGURATIONS

Host Processor	Dual-socket quad-core Intel Xeon processor (SMT enabled)
Host Memory	48GB ECC DRAM
Host Storage	Storage Area Network (SAN)
Host Network	Intel 82599 10GbE NIC (with SR-IOV support)
Hypervisor	Xen Upstream (64bit, change set 19590)
HVM Guest	RHEL 5U1 (kernel updated to 2.6.30)

The micro-benchmark results show that PLE brings up to 14% improvements. EPT brings about 24% performance improvements and achieves 3.9X speedup over Shadow Page Table in the UP guest and 16-VCPU MP guest respectively. Large memory pages and confined Guest Page Table bring ~2% and ~3% additional performance improvements, respectively. In the server consolidation benchmark, EPT achieves an up to 49% performance advantage over Shadow Page Table. SR-IOV brings an additional 28% improvement.

In total, EPT+SR-IOV can reach about 1.77x the performance of that in the baseline Shadow Page Table test.

A. *Micro-Benchmarks*

We used several micro-benchmarks (including Kernel Build, SPECjbb, WebBench, and Netperf) in our experiment. When running WebBench and Netperf, the testing system runs as the "server" and is directly connected to the "client" that runs in native. The service VM (that is domain 0 in Xen) employs 8 VCPUs. The other detailed configurations of the VM guests for these micro-benchmarks are shown in Table II.

TABLE II. MICRO-BENCHMARK GUEST CONFIGURATIONS

	Guest OS	VCPUs	Guest Memory
Kernel Build	32bit Linux	1-64	512MB
SPECjbb	64bit Linux	2	4GB
WebBench	32bit Linux	2	1.5GB
Netperf	32bit Linux	1	512MB

- **Kernel Build**

Figure 6 shows the overheads of Shadow Page Table (SPT) in Kernel Build. In the uniprocessor (UP) guest, about half of the VM-exit events are due to the write-protection on Guest Page Table (GPT), and about 12.5% of the total CPU cycles are spent on processing these VM-exit. On the other hand, in the multi-processor guest (4 VCPUs in our experiment), though the VM-exit events caused by the write-protection on GPT are fewer than those in the UP guest, about 18.7% of the total CPU cycles are spent on processing these VM-exit. This is because the lock contentions on SPT accesses in the hypervisor brings extra synchronization overheads.

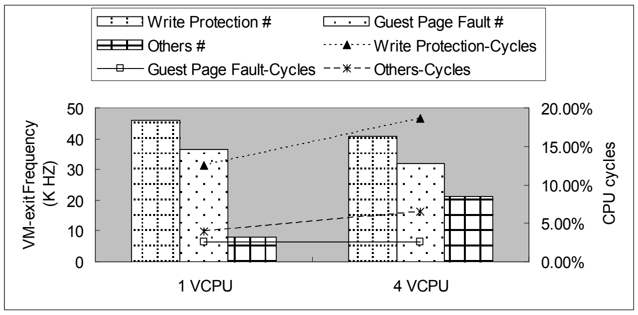

Figure 6: Overheads of Shadow Page Table in Kernel Build

Figures 6 and 7 show the performance and scalability improvements brought by the Extended Page Table (EPT) and large memory pages. In the UP guest (as shown in Figure 7), we run Kernel Build in each guest and measure the average time spent per build. EPT brings about 24% performance improvements over Shadow Page Table, and large memory pages bring about 2% additional performance improvements. In the MP guest (as shown in Figure 8), the performance of Shadow Page Table does not scale beyond 4 VCPUs (due to the lock contentions in the hypervisor), while EPT scales well, even with 16 VCPUs (about 8.17X speedup compared to the 1-VCPU case). EPT achieves 3.9X speedup compared to the SPT in 16-VCPU case.

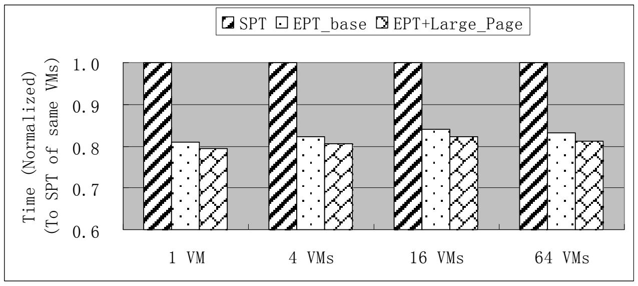

Figure 7. Performance and scalability of Kernel Build in multiple UP guests

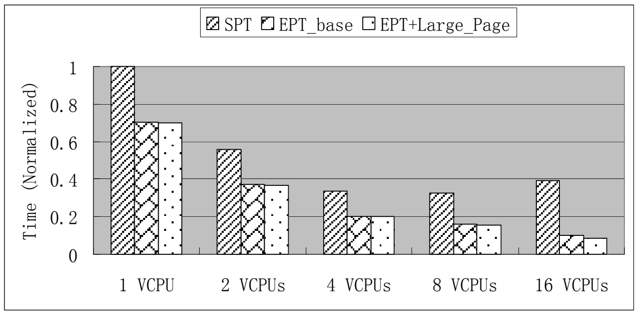

Figure 8. Performance and scalability of Kernel Build in an MP guest

Figure 9 shows the performance and scalability improvements brought by Pause Loop Exit (PLE) on top of EPT with large memory pages when running Kernel Build in the MP guest with 4 VCPUs. PLE brings about 14% improvements when the virtualized CPUs are overcommitted (that is 16-VM case with a total of 64 VCPUs), but it doesn't help if the CPU is not overcommitted (that is 1-VM case with a total of 4 VCPUs).

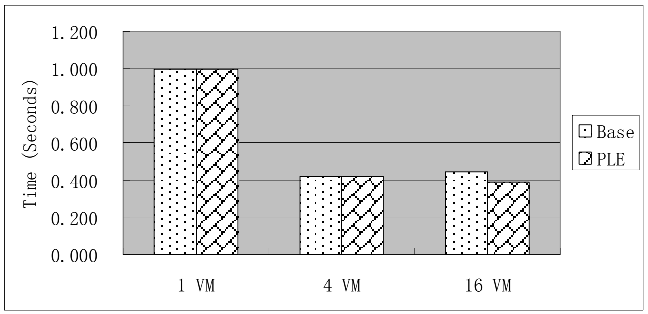

Figure 9. Performance and scalability of Kernel Build in a 4-VCPU guest

- **SPECjbb/WebBench**

EPT with large memory pages achieves 8% and 14% performance advantage over base line EPT in SPECjbb and WebBench, respectively. WebBench is more sensitive to large memory pages than SPECjbb, because WebBench frequently switches its processes (and hence page tables) per client requests, stressing the TLB entries, EPT paging-structure caches, and memory caches. The results are shown in Figure 10.

The benefits of confining Guest Page Table to one or more contiguous 2MB physical memory space depend on the memory allocation patterns of the applications. If the application allocates memory pages in a relatively static fashion, the memory pages of the Guest Page Table typically come from a contiguous memory block (such as in Linux). However, when the application frequently allocates and frees memory pages, the memory pages of the Guest Page Table may scatter in different locations across the entire memory space. Consequently, Guest Page Table confinement will bring bigger performance gains.

Figure 10 also shows that confining Guest Page Table to a 2MB contiguous physical memory space achieves ~3% additional performance advantage on top of EPT with large memory pages in WebBench, which frequently create and destroy pages per client requests. However, there are almost no performance gains for SPECjbb, because it allocates memory in a relatively static fashion.

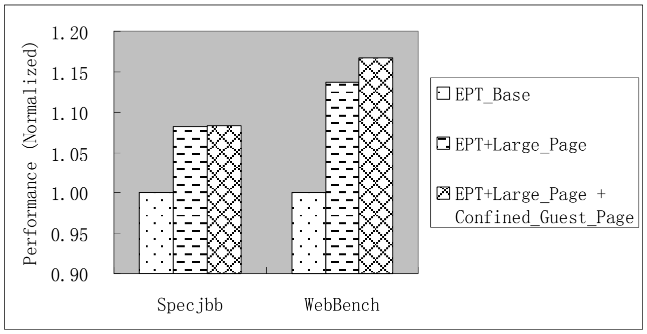

Figure 10. Performance of SPECjbb/WebBench with Confined Guest Page Table

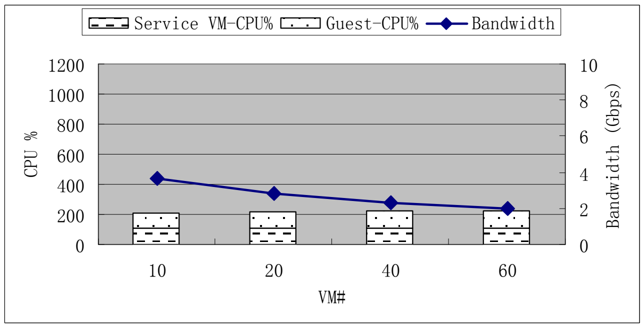

Figure 11: Performance of single-threaded PV NIC (100% CPU utilizations means a full thread cycles)

- **Netperf**

The scalability of PV NIC suffers from excessive CPU utilizations due to packets copy in service VM. The existing Xen PV NIC driver uses only a single thread in the backend for packet copy, which can easily saturate at 100% CPU utilization (although the tested system and the service VM has 16 threads in total, that is 1600% CPU resource). Consequently, it does not scale well with additional CPU cores and can achieve only about 3.6Gbps peak bandwidth in our experiment (using 10 VM guests), as shown in Figure 11.

We have enhanced the Xen PV NIC driver to run in multi-thread mode for backend service, to improve its scalability. Consequently, the enhanced PV NIC can achieve the line rate (that is 9.48 Gbps) with 10 VM guests at the cost of more than 700% CPU overheads, as shown in Figure 12.

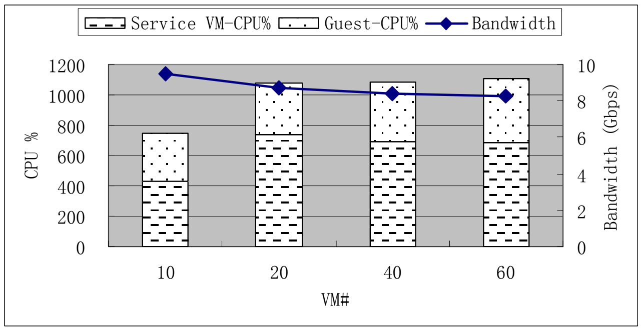

Figure 12: Scalability of multi-threaded PV NIC (100% CPU utilizations means a full thread cycles)

In SR-IOV test, the service VM runs the PF driver and a dedicated VF is assigned to each guest. As shown in Figure 13, SR-IOV demonstrates perfect scalability, achieving line rate (that is 9.48Gbps) with 60 VM guests at the cost of only 0.88X additional CPU thread overhead over 10 VM guests.

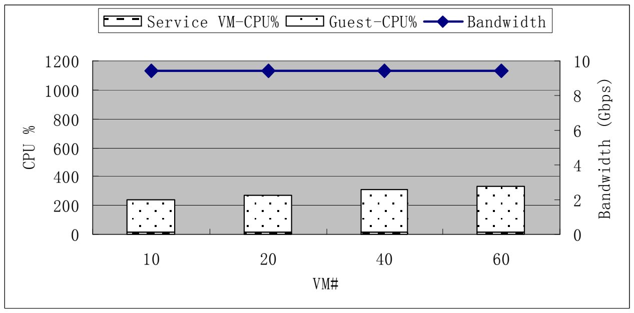

Figure 13. Scalability of SR-IOV NIC (100% CPU means a full thread cycles)

B. Server Consolidation Benchmark

vConsolidate [3] is a VMM agnostic benchmark for virtualization consolidation, which includes a compute-intensive workload, a Web server, an email server and a database server. Each workload runs in a separate VM. The compute-intensive workload runs SPECjbb, which is modified to consume roughly 50% of the CPU, by inserting random sleep sessions every few milliseconds to represent the real life scenario. The Web server runs WebBench using Apache Webserver. The email server runs LoadSim, representing a Microsoft Exchange workload that runs transactions on Outlook with 500 users logged-in simultaneously. The database server runs SysBench, representing an OLTP workload running transactions against an MySQL database. An idle VM is added to mimic the real life scenario because datacenters are not fully used at all times. Those 5 VMs running different workloads comprise a Consolidated Stack Unit (CSU).

The throughput of each CSU is measured as a geometric mean of all 4 workloads (Web, Mail, DB, and Java) and the total system throughput is a sum of all the CSUs the system runs.

TABLE III. VCONSOLIDATE CONFIGURATIONS

	VCPUs	Guest Memory	Guest OS	Application
Web	2	1.5GB	32-bit Linux	Apache
Mail	1	1.5GB	32-bit Windows	Exchange
DB	2	1.5GB	64-bit Linux	MySQL
Java	2	2.0GB	64-bit Linux	BEA JVM
Idle	1	0.4GB	32-bit Windows	

In our experiment, we run vConsolidate using the configuration shown in Table III. The service VM employs 4 VCPUs with 512MB memory and all guests use PV disk and PV NIC drivers, except that in the SR-IOV test, each guest runs a VF driver with a dedicated VF. In the server, or system under test (SUT), each CSU uses 2 Intel X25E 64GB SSDs as its storage, and talks to a mail client, a WebBench client, and a Web controller (as shown in Figure 14). Multiple CSUs are used in our experiment, and the server and clients are connected through a Gigabit Ethernet switch.

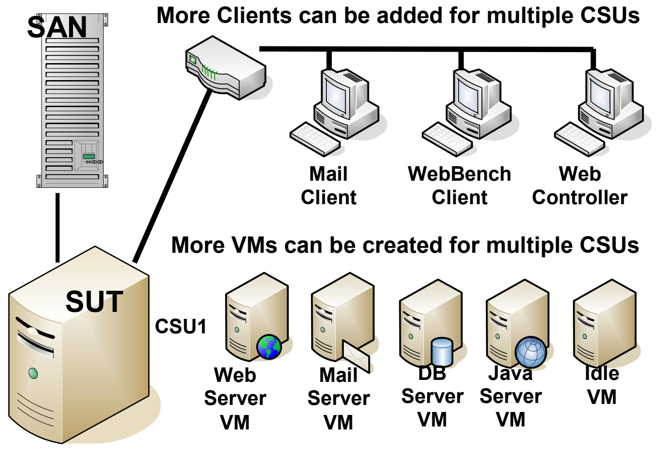

Figure14. The vConsolidate experiment environment

- **Performance**

Figure 15 shows the vConsolidate performance in the 1-CSU configuration. EPT+Large_Page, or EPT_LP configuration supports both hardware EPT support and large memory pages optimization, achieves about 29% performance improvement over Shadow Page Table (SPT) with lower CPU utilizations, and large memory pages in EPT bring 3% performance gain over EPT_base, which only uses hardware EPT support. Using SR-IOV further reduces the CPU utilizations from 28% to 24%. An interesting observation is that PV I/O has a 1.5% performance advantage over SR-IOV. This is because the CPU is not overcommitted in the 1-CSU configuration, and packets copied by the spare CPUs in service VM may improve cache locality.

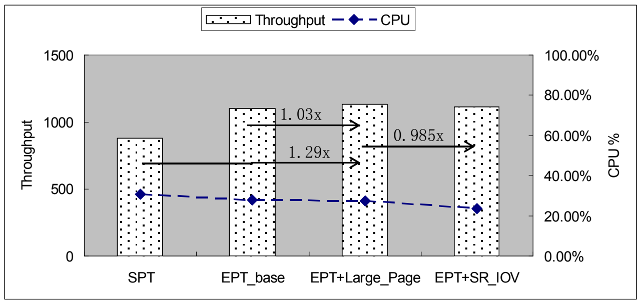

Figure15. vConsolidate performance in the 1-CSU configuration

Further breakdown of vConsolidate performance is shown in Figure 16. WebBench performance in EPT_LP is about 215% of that in SPT, which dominates the contribution to the overall vConsolidate performance. SPECjbb, SysBench, and LoadSim demonstrate moderate performance improvement (up to 14%).

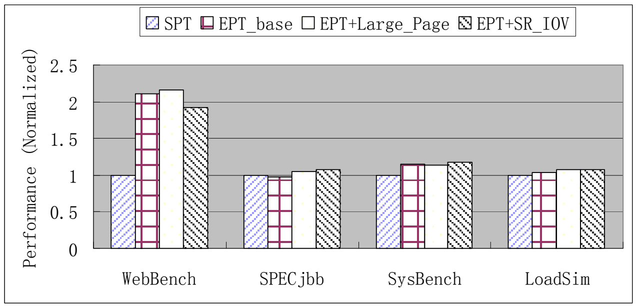

Figure 16. Breakdown of vConsolidate performance

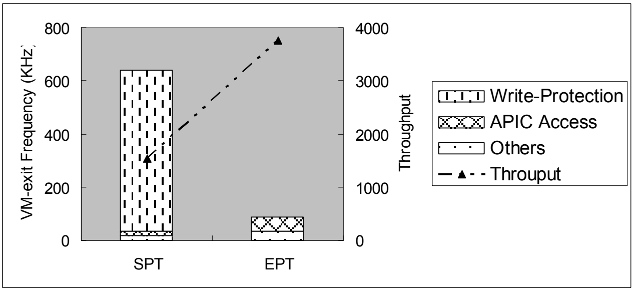

Figure17. VM-exit Frequency in WebBench

The high performance boost of WebBench in EPT_LP is due to the elimination of excessive VM-exit caused by the write-protection on Guest Page Table. As shown in Figure 17, these VM-exit events account for about 95% of the total VM-exit events in WebBench, and about 38% of the virtualization overhead of total CPU cycles. Most of these VM-exit events actually come from the execution of fork function, which reflects the fact that WebBench uses fork frequently to create a new process to handle each client request.

- **Scalability**

The performance of vConsolidate does not scale well in Shadow Page Table when more Consolidation Stack Units (CSUs) are deployed, as shown in Figure 18. Shadow Page

Table (SPT) achieves about 58% performance improvement from 1 to 2 CSUs, and tops its performance at the 3-CSUs configuration with an additional 23% improvement, using 95% CPU cycles. The SPT test fails to run vConsolidate with the 4-CSU configuration due to mail server failure.

The performance of EPT_LP scales much better than that of Shadow Page Table. The EPT_LP performance improves by 75% from 1 to 2 CSUs, and by 24% improvement from 2 to 3 CSUs (with 94% CPU utilizations). EPT_LP tops its performance in the 4-CSU configuration with 99% CPU utilizations, with an additional 3% performance gain over the 3-CSU configuration. The peak performance of the EPT_LP in the 4-CSU configuration is 2.25x of that in the 1-CSU configuration, and about 1.49x of the peak performance of Shadow Page Table (in the 3-CSU configuration).

Using SR-IOV together with EPT_LP achieves even better scalability, as shown in Figure 18. It brings about 3%, 8%, and 19% performance advantages over EPT_LP only, in the 2-, 3- and 4-CSU configurations, respectively. Its peak performance in the 4-CSU configuration is 2.73x of that in the 1-CSU configuration, and about 1.77x of the peak performance of SPT (an additional 28% improvement compared to EPT_LP only). This is due to the elimination of the service VM intervention in the guest network packet processing, because the intervention introduces additional network latency, service VM overhead, and cache pollution.

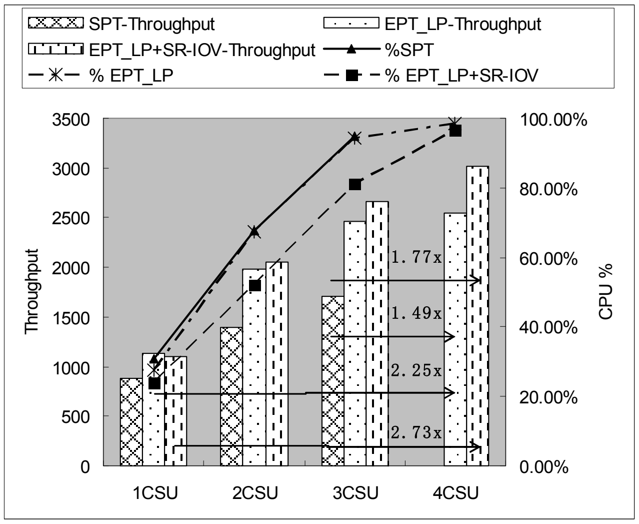

Figure 18. vConsolidate scalability

Further breakdown of scalability of each workload in SPT, EPT_LP and EPT_LP+SR-IOV are shown in Figures 19 – 21. With both EPT_LP and SR-IOV, all workloads except SysBench scale very well from 1 to 4 CSUs, before the CPUs are saturated. SysBench have very heavy disk I/O and reaches its peak performance in the 3-CSU configuration.

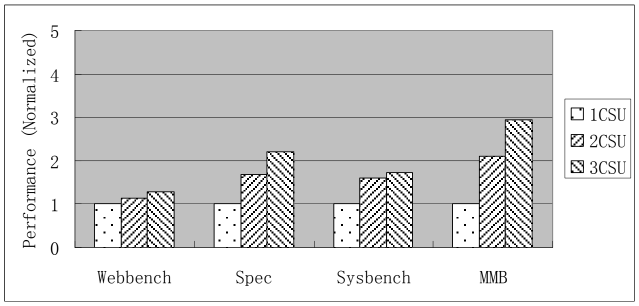

Figure 19. Breakdown of vConsolidate Scalability in Shadow Page Table

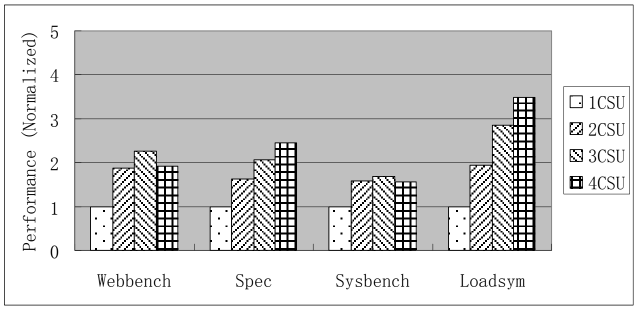

Figure 20. Breakdown of vConsolidate Scalability in EPT_LP

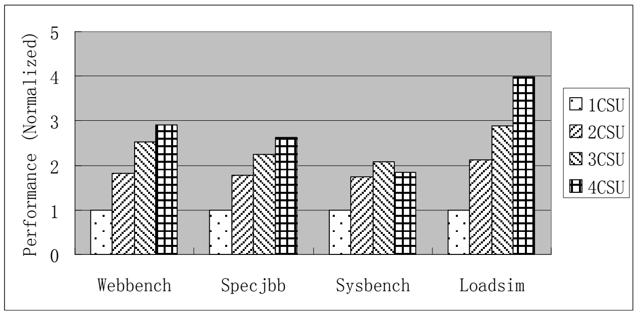

Figure 21. Breakdown of vConsolidate Scalability in EPT_LP+SR-IOV

The EPT_LP test has similar scalability, but it has higher CPU utilizations than when SR-IOV is also used. WebBench in EPT_LP configuration reaches its peak performance in the 3-CSU configuration, because it has very high network I/O and PV NIC brings very high overheads in the 4-CSU configuration. On the other hand, Shadow Page Table cannot scale beyond 3 CSUs because of the excessive VM-exit and lock contentions.

VI. Related Work

Full virtualization (with binary translation) [25], proposed when there were not sufficient hardware supports, supports unmodified guest OSes, but requires the OS code to be translated to binary when needed. Unfortunately, this incurs very high virtualization overheads. Basic hardware virtualization technologies, such as Intel Virtualization technology [23], are introduced to eliminate the need for binary translation and to improve the virtualization performance. In addition, paravirtualization [6][9] improves virtualization performance with moderate modifications to OS codes, which however is difficult to apply to proprietary

OSes.

Impacts of spin-lock to MP guest performance and scalability have been explored before. For instance, the approaches to donate the wasted spinning time to the lock holder or to avoid preempting lock-holders are studied in [24]. Heuristic based detection of lock-holder, when the number of unique stores executed within N committed instructions is less than some pre-defined threshold, is explored in [29]. Neither of these approaches reported the results on real-world VMM or hypervisor products.

Hypervisor scheduling algorithm is also explored. Communication-aware CPU scheduling algorithm for co-located multi-tier applications is explored in [10]. Hybrid scheduling framework for virtual machine systems is explored when the virtual machine is used to execute the concurrent applications [30]. They do not address the virtualization efficiency issue of spin-lock in MP guest.

Memory virtualization technologies have been widely studied in the literature. Barham improves memory virtualization performance with paravirtualization by modifying the guest OS to batch Guest Page Table access operations [6]. Adams analyzed architecture level events of page table updates, context switches, and performance of Shadow Page Table [2]. However, they do not fully take advantage of modern hardware technologies to maximize resource utilization. Waldspurger employed ballooning technique, content-based page sharing and hot I/O page remapping to share memory among guests [25]. Gupta leverages sub-page level sharing (through page patching) and in-core memory compression [11]. However, their impacts on system level scalability are unclear.

Software optimizations to PV I/O virtualization have also been widely studied, such as page remapping and batch packet transferring [9], Xenloop [26], scheduler [14][17], virtual switch enhancement, and transmit queue length optimization [14]. However, none of them can completely eliminate the hypervisor intervention to performance packet movement, and therefore still have the high overheads due to I/O packet copy, latency, and cache pollution.

Hardware-assisted solutions are proposed to PV I/O to achieve high-performance, such as VMDq [22], and self-virtualized devices [20]. In these solutions, each VM is assigned a portion of resources, or a pair of queues, to transmit and receive packets with reduced VMM intervention. SR-IOV [18][8] uses IOMMU for memory protection and address translation. VMM-Bypass I/O is explored in [15], extending OS-bypass design of InfiniBand software stack to bypass VMM for performance. Neither of these approaches analyzed the impacts on system level scalability.

VII. Conclusion

Traditional virtualization technologies incur very high virtualization overheads, especially on the modern, multi-core and/or overcommitted systems. For instance, Shadow Page Table suffers from excessive VM-exit due to the write protection on Guest Page Table, and high synchronization overheads due to lock contentions on SPT accesses for the MP guest.

We have implemented and optimized the support of advanced hardware accelerations in the latest version of Xen, including Pause Loop Exit (PLE), Extended Page Table (EPT), and Single Root I/O Virtualization (SR-IOV). The experimental results demonstrate very good performance and scalability on the multi-core and overcommitted system, for both micro-benchmark and a server consolidation benchmark. The results show an up to 77% improvement in the server consolidation benchmark (49% of which due to EPT and another 28% due to SR-IOV), and an up to 14% improvements in the micro-benchmarks due to PLE.

Acknowledgment

The other members contributing to this study include Xiaowei Yang, Xiaohui Xin, Yu Zhao and Intel Linux OS VMM team. We also would like to thank the helpful academic works sponsored by the National Natural Science Foundation of China (Grant No. 60773093, 60970107, 60970108), the Science and Technology Commission of Shanghai Municipality (09510701600, 10511500102, 10DZ1500200).

References

[1] Amazon Elastic Compute Cloud User Guide, http://aws.amazon.com/

[2] Keith Adams, Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization, ASPLOS-12, San Jose, CA, 2006, pp. 2-13

[3] Padma Apparao, Ravi Iyer, Xiaomin Zhang, Don Newell, Tom Adelmeyer, Characterization & analysis of a server consolidation benchmark, Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Seattle, WA, 2008, pp. 21-30

[4] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini and Srilatha Manne, Accelerating Two-Dimensional PageWalks for Virtualized Systems, ASPLOS-13, Seattle, WA, 2008, pp. 26-35

[5] A. J. Bernstein, Program Analysis for Parallel Processing, IEEE Trans. on Electronic Computers". EC-15, pp. 757–62, 1966

[6] 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, In proceedings of the 19th ACM symposium on Operating Systems Principles, Bolton Landing, NY, 2003, 164-177

[7] J. P. Casazza, M. Greenfield, K. Shi, Redefining Server Performance Characterization for Virtualization Benchmarking, Intel technology Journal, Volume 10, Issue 03

[8] Yaozu Dong, Xiaowei Yang, Xiaoyong Li, Jianhui Li, Kun Tian, Haibing Guan, High Performance Network Virtualization with SR-IOV, HPCA-16, Bangalore, India, 2010.

[9] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams, Safe hardware access with the Xen virtual machine monitor, In 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure, Boston, MA, 2004.

[10] Sriram Govindan, Arjun R. Nath, Amitayu Das, Bhuvan Urgaonkar, Anand Sivasubramaniam, Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms, Proceedings of the 3rd ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, San Diego, CA, 2007, pp. 126-136

[11] D..Gupta, etc, Difference Engine: Harnessing Memory Redundancy in Virtual Machines, In Proc. of the 8th OSDI, 2008, Sandiego, CA.

[12] Intel Corporation, Intel® 64 and IA-32 Architectures Software Developers' Manual, http://www.intel.com/products/processor/manuals/index.htm.

[13] Intel Corporation, Intel® 82599 10 Gigabit Ethernet Controller Datasheet. http://www.intel.com

[14] G. Liao, D. Guo, L. Bhuyan, S. R King, Software techniques to improve virtualized I/O performance on multi-core systems. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, 2008, 161-170

[15] J. Liu, et al. High Performance VMM-Bypass I/O in Virtual Machines. Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2006, 3-3

[16] A. Menon, A. L. Cox, W. Zwaenepoel, Optimizing Network Virtualization in Xen. Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2006, 15-28.

[17] D. Ongaro, A. L. Cox and S. Rixner, Scheduling I/O in Virtual Machine Monitors. Proceedings of 4th ACM/USENIX International Conference on Virtual Execution Environments, Seattle, WA, 2008, 1-10.

[18] PCI Special Interest Group, http://www.pcisig.com/home

[19] Gerald J. Popek, Robert P. Goldberg, Formal requirements for virtualizable third generation architectures, Communications of the ACM, v.17 n.7, pp. 412-421, July 1974

[20] H. Raj, K. Schwan, High performance and scalable I/O virtualization via self-virtualized devices. Proceedings of the 16th international symposium on high performance distributed computing, Monterrey, CA, 2007

[21] John Scott Robin and Cynthia E. Irvine, Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, Proc. 9th USENIX Security Symposium, Denver, CO, 2000

[22] J. R. Santos, Y. Turner, G. Janakiraman and I. Pratt, Bridging the Gap between Software and Hardware Techniques for I/O Virtualization, Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2008. 29-42

[23] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, L. Smith, Intel Virtualization Technology, 38(5), Los Alamitos, CA, IEEE Computer Society Press , 2005, 48–56

[24] V. Uhlig, J. Levasseur, E. Skoglund, U. Dannowski, Towards scalable multiprocessor virtual machines, In Virtual Machine Research and Technology Symposium (2004), USENIX, pp. 43–56.

[25] C. A. Waldspurger, Memory resource management in VMware ESX server, ACM SIGOPS Operating Systems Review, v.36

[26] J. Wang, K. Wright, and K. Gopalan, XenLoop: A Transparent High Performance Inter-VM Network Loopback, in Proceedings of the 17th international symposium on High performance distributed computing, Boston, MA, 2008, 109-118

[27] Philip M. Wells, Koushik Chakraborty, Gurindar S. Sohi, Dynamic heterogeneity and the need for multicore virtualization, ACM SIGOPS Operating Systems Review, v.43 n.2, April 2009

[28] A. Whitaker, M. Shaw, and S. D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical Report 02-02-01, University of Washington, 2002.

[29] P. M. Wells, K. Chakraborty, G. S. Sohi, Hardware support for spin management in overcommitted virtual machines, In Proceedings of the 15th International Conference on Parallel Architecture and Compilation Techniques (PACT'06) (Seattle, Washington, USA, Sept. 2006), ACM, pp. 124–133.

[30] Chuliang Weng, Zhigang Wang, Minglu Li, and Xinda Lu, Hybrid Scheduling Framework for Virtual Machine Systems, Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Washington, DC, 2009, pp. 111-120.

[31] K. K. Yue, D. J. Lilja. An Effective Processor Allocation Strategy for Multiprogrammed Shared-Memory Multiprocessors. TPDS. IEEE Transactions on Parallel and Distributed Systems (TPDS), VOL. 8, NO. 12, 1997.