Selective Hardware/Software Memory Virtualization

Xiaolin Wang
Dept. of Computer Science and Technology, Peking University, Beijing, China, 100871
wxl@pku.edu.cn

Jiarui Zang
Dept. of Computer Science and Technology, Peking University, Beijing, China, 100871
zjr@pku.edu.cn

Zhenlin Wang
Dept. of Computer Science, Michigan Technological University, Houghton, MI 49931, USA
zlwang@mtu.edu

Yingwei Luo
Dept. of Computer Science and Technology, Peking University, Beijing, China, 100871
lyw@pku.edu.cn

Xiaoming Li
Dept. of Computer Science and Technology, Peking University, Beijing, China, 100871
lxm@pku.edu.cn

Abstract

As virtualization becomes a key technique for supporting cloud computing, much effort has been made to reduce virtualization overhead, so a virtualized system can match its native performance. One major overhead is due to memory or page table virtualization. Conventional virtual machines rely on a shadow mechanism to manage page tables, where a shadow page table maintained by the VMM (Virtual Machine Monitor) maps virtual addresses to machine addresses while a guest maintains its own virtual to physical page table. This shadow mechanism will result in expensive VM exits whenever there is a page fault that requires synchronization between the two page tables. To avoid this cost, both Intel and AMD provide hardware assists, EPT (extended page table) and NPT (nested page table), to facilitate address translation. With the hardware assists, the MMU (Memory Management Unit) maintains an ordinary guest page table that translates virtual addresses to guest physical addresses. In addition, the extended page table as provided by EPT translates from guest physical addresses to host physical or machine addresses. NPT works in a similar style. With EPT or NPT, a guest page fault can be handled by the guest itself without triggering VM exits. However, the hardware assists do have their disadvantage compared to the conventional shadow mechanism – the page walk yields more memory accesses and thus longer latency. Our experimental results show that neither hardware-assisted paging (HAP) nor shadow paging (SP) can be a definite winner. Despite the fact that in over half of the cases, there is no noticeable gap between the two mechanisms, an up to 34% performance gap exists for a few benchmarks. We propose a dynamic switching mechanism that monitors TLB misses and guest page faults on the fly, and dynamically switches between the two paging modes. Our experiments show that this new mechanism can match and, sometimes, even beat the better performance of HAP and SP.

Categories and Subject Descriptors D.4.2 [**Operating Systems**]: Storage Management – main memory, virtual memory.

General Terms Algorithms, Management, Measurement, Performance, Design, Experimentation, Verification.

Keywords virtual machine; hardware-assisted virtualization; shadow paging; dynamic switching; hardware assisted paging

VEE'11 March 9–11, 2011, Newport Beach, California, USA.

1. Introduction

System virtualization has regained its popularity in the recent decade and has become an indispensable technique for supporting cloud computing. Virtualization provides server consolidation and creates an illusion of a real machine for an end user. To make a virtual machine (VM) acceptable for the end user, it is critical for it to match the performance of a native system with the same resource subscription. However, virtualization brings an additional layer of abstraction and causes some unavoidable overhead. The performance of a VM is often much inferior to the underlying native machine performance. Much effort has been made recently to reduce virtualization overhead in both software and hardware sides [1, 4, 11, 13]. This paper focuses on one major overhead caused by memory or page table virtualization.

Most operating systems (OSes) support virtual memory so an application can bear a view of the whole address space. The OS maintains a group of page tables, which map virtual memory addresses to physical memory addresses for each process. The hardware memory management unit (MMU) translates virtual memory addresses to physical memory addresses according to these page tables. With virtualization, the physical memory is virtualized and the virtual machine monitor (VMM) needs to support physical to machine address translation.

In a system with paging enabled, the VMM can realize memory virtualization on a per-page basis and enforce isolation among multiple VMs. There exist three address spaces in a virtualized system: 1) machine address, the address which appears on the system bus; 2) guest physical address, the pseudo-physical address as seen from VMs; and 3) guest virtual address, the conventional linear address that the guest OS presents to its applications. As illustrated in Figure 1, we denote the mapping from guest physical address to machine address as *p2m*, and the mapping from guest virtual address to guest physical address as *v2p*.

Figure 1. Machine, physical and virtual address

Since there is an additional layer of address translation in a virtualized system, a common scheme to accelerate this two-layer address translation is to generate a composition of *v2p* and *p2m*, denoted as *v2m*, and then load it directly into the hardware Memory Management Unit (MMU). The VMM controls mapping *p2m* and it can retrieve mapping *v2p* by querying the guest page tables. As illustrated in Figure 2, all three existing memory virtualization techniques, para-virtualization, shadow paging-based full virtuali-

zation and hardware-assisted full virtualization, take this approach. They differ in hardware support and/or the way the VMM synchronizes *v2m* and *v2p*. Note that full virtualization does not require modification to the guest OS while para-virtualization does. This paper focuses on a combination of the two full virtualization techniques.

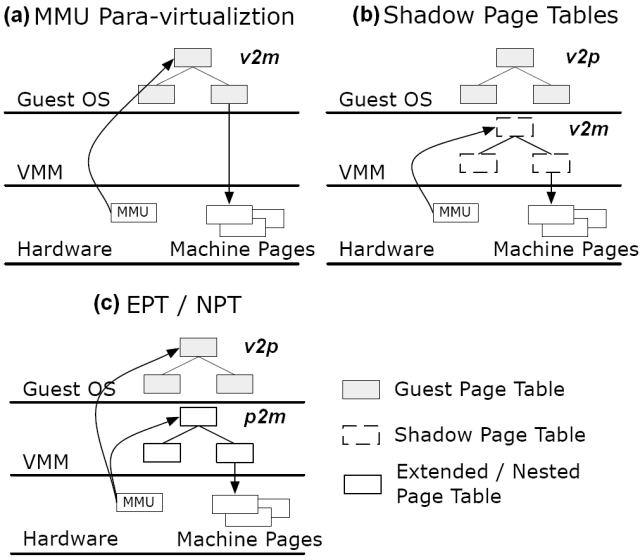

Figure 2. Comparison of memory virtualization architectures

At the cost of compatibility, para-virtualization can achieve better performance than full virtualization, as well as reduce the complexity of the VMM. As shown in Figure 2(a), the VMM simply replaces the mapping *v2p* stored in a guest page table with the composite mapping *v2m*. To ensure that the guest OS functions properly after the replacement, it requires some modification to the source code of the guest OS, which leads to the compatibility issue. For safety, the VMM needs to validate any updates to the page table by the guest OS. By taking back write permission on those memory pages used as the page table , the VMM prevents the guest OS from writing to any guest page table directly. The guest OS has to invoke hypercalls to the VMM to apply changes to its page table. XEN provides a representative hypervisor following this design [2].

A software solution to support full virtualization relies on a shadow paging (SP) mechanism for address translation, where a shadow page table maintained by the VMM maps virtual addresses directly to machine addresses while a guest maintains its own virtual to physical page table [5]. The VMM links the shadow page table to the MMU so most address translations can be done effectively. Figure 2(b) illustrates this implementation. The VMM has to ensure that the content in the shadow page table is consistent with what in the guest page table. Since the guest OS does not know the existence of the shadow page table and will change its guest page table independently, the VMM should perform all the synchronization work to make the shadow page table keep up with the guest page table. Any updates to the shadow page table need also to be reflected in the guest page table. All these synchronizations will result in expensive VM exits and context switches. Moreover, the source code structure for the SP mechanism is quite complex. VMWare Workstation,VMWare ESX Server, KVM, and XEN all implement shadow paging [2, 8, 12].

To avoid the synchronization cost in the SP mechanism, both Intel and AMD provide hardware assists, EPT (extended page table) and NPT (nested page table), to facilitate hardware-assisted address translation as illustrated by Figure 2(c) [4, 10]. We call the paging mechanism using EPT or NPT *hardware assisted paging* (HAP). With the hardware assists, the MMU maintains ordinary guest page tables that translate virtual addresses to guest physical addresses. In addition, the extended page table as provided by EPT translates from guest physical addresses to host physical or machine addresses. The function of NPT is similar to EPT. All evaluation in this paper is based on EPT on an Intel machine. However, we expect our design to work on an AMD machine with NPT support. With EPT or NPT, a guest page fault can be handled by the guest itself without triggering VM exits. However, the hardware assists do have their disadvantages compared to the conventional shadow mechanism. With HAP, an address translation from virtual to host physical needs to go through both the guest table and the extended table. This page walk yields more memory accesses and thus longer latency. The problem becomes more prominent in 64-bit systems compared to 32-bit systems since the page walk length is doubled in 64-bit systems.

Both HAP and SP have their own advantages and disadvantages. Our experimental results on SPEC CPU2006 [9] in Figures 3 and 4 show that neither HAP nor SP can be a definite winner. Both figures show the normalized execution time with respect to HAP. In eight of the twenty-nine benchmarks there is a 3% or more performance gap between the two mechanisms. Notably, SP is 34% slower than HAP for *gcc*. There is a 13%, 15%, and 22% performance gap, respectively, for *mcf*, *catusADM* and *tonto*.

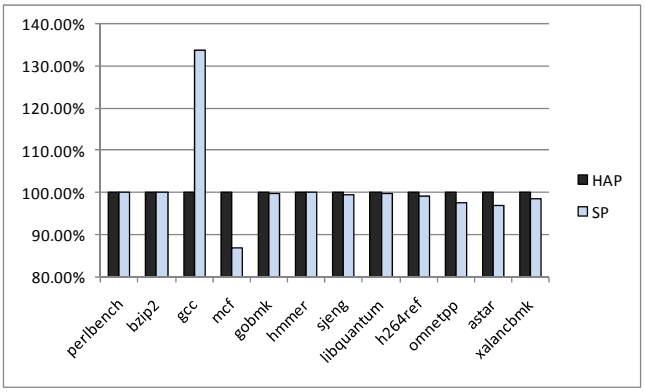

Figure 3. Normalized execution time (SPEC Int).

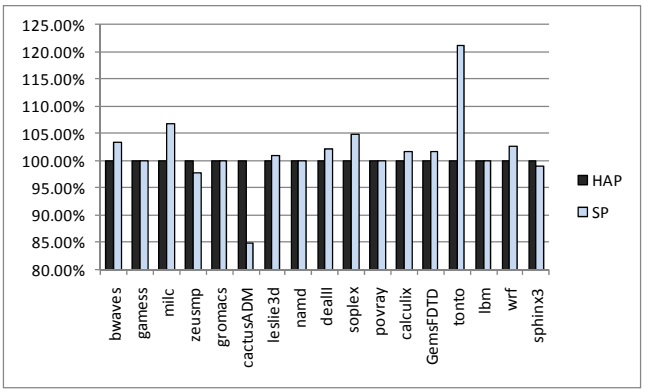

Figure 4. Normalized execution time (SPEC FP)

Since neither HAP nor SP performs better all the time, we can lose performance no matter which one we pick as the default. An intelligent mechanism should be able to exploit the advantages of

both HAP and SP based on the VM behavior. So we propose a dynamic switching mechanism that can switch the paging mode between HAP and SP based on the runtime behavior of the current applications. We name this mechanism *Dynamic Switching Paging* (DSP). DSP relies on online sampling of TLB misses and page faults to make decisions on paging mode switching. We develop a set of heuristics to assist the decision making. Our results show that DSP is able to match or even beat the better performance of SP and HAP for all benchmarks, either in a 32-bit system or a 64-bit system. In the meantime, the overhead of DSP is negligible.

The remainder of this paper is structured as follows. In Section 2 we describe the design of DSP. Section 3 details an implementation of this mechanism based on XEN [2, 14]. Section 4 evaluates DSP using some industry standard benchmarks and compares it with HAP and SP. Section 5 discusses related work. We finally conclude and discuss future work in Section 6.

2. DSP Design

2.1 DSP Functionality

HAP is controlled by the CPU control register. By setting or resetting the corresponding control bit, we can choose whether or not to use HAP on a machine where HAP is supported. Take Intel's EPT as an example. There is a Secondary Processor-Based VM-execution Control Register in the CPU. Bit 1 of this register, defined as "Enable EPT", controls EPT. If this bit is set, EPT is enabled. Otherwise, it is disabled.

To switch to HAP mode, the VMM should prepare a group of page tables as the extended page tables, which map guest physical addresses to host machine addresses. In Section 1, we name this guest physical address to host machine address map the *p2m* map. For the extended tables to take effect, the VMM needs to transfer the root address of the top-level page directory to the hardware virtual machine control structure (VMCS). For most VMM implementations, the *p2m* map is fixed. Therefore, the content of EPT is often fixed as well. When the extended tables are ready, we can enable EPT by setting the control bit.

To switch to SP mode, we need a shadow page table. Because the guest page table is available in both SP mode and HAP mode, the shadow page table can be constructed based on the guest page table and the *p2m* map. The switching thus requires reconstruction of the shadow page table and resetting the EPT control bit.

Since both modes need the *p2m* map, we keep this map intact in both modes. When switching from HAP mode to SP mode, we store the root address of EPT temporarily and restore it to the designed register in VMCS at the time when switching back. In SP mode, the shadow page table should be synchronized with the guest page table, while we do not need a shadow page table in HAP mode. To facilitate quick switching, one approach is to maintain a shadow page table in HAP mode so we do not need to reconstruct it when switching to SP mode. We find that this approach damages the independence of HAP mode and also results in high overhead. We instead rebuild a new shadow page table every time we switch to SP mode. The table is destroyed when leaving SP mode for HAP mode.

To summarize, when switching from HAP mode to SP mode, store the root address of the top level page directory of the *p2m* map, rebuild the shadow page table, and then disable the "Enable EPT" control bit in the Secondary Processor-Based VM-execution Control Register; when switching from SP mode to HAP mode, destroy the shadow page table, restore the root address of the top level page directory of the *p2m* map, and then turn on the "Enable EPT" control bit in the Secondary Processor-Based VM-execution Control Register.

2.2 DSP Tradeoff Analysis

To find out when is a good time to switch between the two paging modes, we need to understand the advantages and disadvantages of each mode. HAP mode eliminates the expensive VM exits and context switches from SP mode when there are needs to synchronize the shadow page table and the guest page table. SP mode enables quicker address translation because it only needs to walk through the shadow page table while HAP mode needs to walk both the guest page table and the *p2m* map, which doubles the number of memory accesses. An ideal switching policy would require predicting the number of VM exits saved by HAP mode and the number of memory accesses saved by SP mode. With an estimation of VM exit penalty and memory access latency, one can design a cost model to determine when to switch between the two modes. Unfortunately, it is difficult to predict either of the two metrics. In HAP mode, there is no shadow page table and thus no VM exits due to shadow-guest synchronization. Although we can monitor the TLB misses and estimate the number of page walks, the MMU cache available in both NPT and EPT eliminates this direct correlation. A TLB miss can hit the MMU cache and thus does not need to walk the page table. Both NPT and EPT come with effective MMU translation caches [3, 4]. Nevertheless, we find that TLB misses are still closely correlated to HAP and SP performance. Rather than estimate the number of VM exits, we take the guest OS page fault statistic as a replacement metric. We observe that HAP mode performs better than SP mode in the applications with a large number of page faults, such as *gcc* and *tonto*, while HAP performs worse in those applications with a small number of page faults but intensive memory accessing and a large number of TLB misses, such as *mcf* and *cactusADM*. Based on the analysis above, DSP switches to HAP mode when we expect frequent page faults in the next period and to SP mode when we foresee frequent TLB misses. To fulfill dynamic switching, we rely on historic TLB miss and page fault information to predict the future trend and make a switching decision.

2.3 DSP Switching Strategy

Both TLB miss and page fault penalties are hardware and system dependent. To estimate the trend, in our implementation, we instead measure page fault frequency and TLB miss frequency, which is the number of page faults and the number of TLB misses, respectively, per thousand retired instructions. To make a decision in DSP, we need a pair of system-dependent thresholds that guard page fault and TLB miss frequencies, respectively. If neither the page fault frequency nor the TLB miss frequency goes beyond its threshold, there would be little difference between HAP and SP mode. DSP should stay in the current mode to avoid the switching cost. If one metric is beyond the threshold and the other is low, DSP needs to take action and switch to the other mode. If both frequencies are high, we need to weigh the relative penalty of each. We introduce a third metric, *P-to-T* ratio, as an estimation of this relative penalty. The P-to-T ratio is the page fault frequency divided by the TLB miss frequency. A third threshold is used to guard the P-to-T ratio.

We manually take a simple machine learning approach to learn the thresholds that determine DSP switching. By training the decision model through the SPEC INT benchmarks on a 32-bit guest, we obtain a heuristic DSP switching algorithm as follows.

1. If the TLB miss frequency is higher than the TLB miss upper-bound threshold and the page fault frequency is lower than 80 percent of the page fault upper-bound threshold, switch from HAP mode to SP mode or stay in SP mode.

2. If the page fault frequency is higher than the page fault upper-bound threshold and the TLB miss frequency is lower than 80 percent of the TLB miss upper-bound threshold, switch from SP mode to HAP mode or stay in HAP mode.
3. If both the TLB miss frequency and the page fault frequency are lower than their lower-bound thresholds, stay in the current paging mode.

For the remaining cases, we will need to use the P-to-T ratio. We notice that the P-to-T ratios show a large range of fluctuations from period to period. We use both a running average of recent P-to-T ratios, called *historic P-to-T ratio*, and the P-to-T ratio in the current monitoring period to help make decision. Below is our policy where step 4 helps avoid divide by 0 exceptions.

4. If either the historic TLB miss frequency or the current TLB miss frequency is zero, switch from SP to HAP or stay in HAP mode.
5. If both the historic average P-to-T ratio and the current P-to-T ratio are bigger than the P-to-T ratio upper-bound threshold, the page fault penalty is more significant than the TLB miss penalty and DSP decides to switch from SP mode to HAP mode or stay in HAP mode.
6. If both the historic average P-to-T ratio and the current P-to-T ratio are lower than the P-to-T ratio lower-bound threshold, the TLB miss penalty is more significant than the page fault penalty. Now DSP switches from HAP mode to SP mode or stays in SP mode.
7. If both the historic average P-to-T ratio and the current P-to-T ratio are between the lower-bound and upper-bound thresholds, neither is significant and there would be little difference between the two paging modes. In this case, the system stays in the current mode.
8. Otherwise, the historic average P-to-T ratio and the current P-to-T ratio fit into different threshold intervals. We cannot decide the trend and the system stays in the current mode.

Figure 5 summarizes the eight policies and shows the workflow of the DSP decision algorithm where the acronyms are listed below.

- FTLB: Frequency of TLB misses
- FPF: Frequency of Page Faults
- HTLB: Historic TLB miss frequency
- HPT: Historic average P-to-T ratio
- CPT: Current P-to-T ratio
- TLBU: TLB miss Upper-bound threshold
- TLBL: TLB miss Lower-bound threshold
- PFU: Page Fault Upper-bound threshold
- PFL: Page Fault Lower-bound threshold
- PTU: P-to-T ratio Upper-bound threshold
- PTL: P-to-T ratio Lower-bound threshold

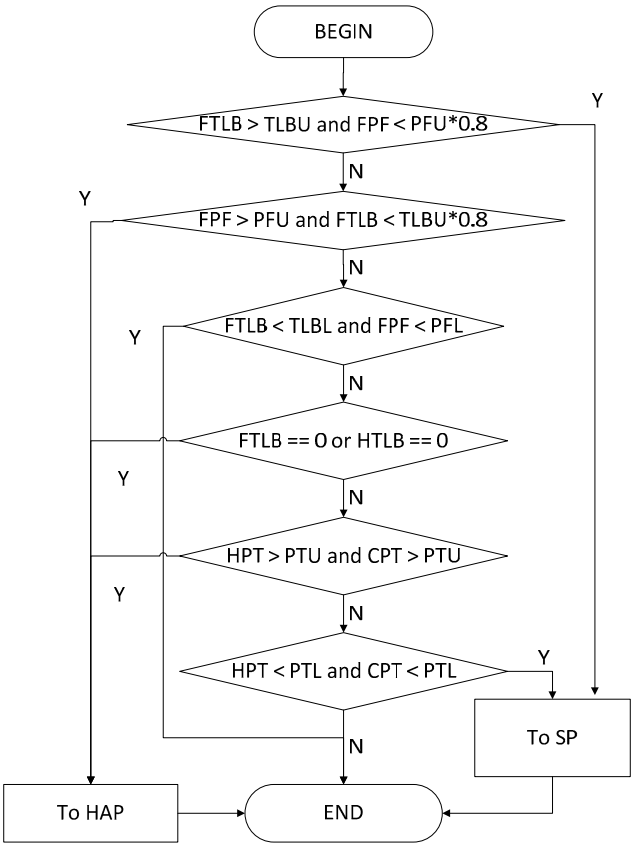

Figure 5. DSP decision diagram

3. DSP Implementation on XEN

We have implemented DSP in XEN 3.3.1. Domain 0 operating system is CentOS 5.4 x86_64 with Linux kernel 2.6.18. Domain U operating system is CentOS 5.4 x86_32 with Linux kernel 2.6.18.

3.1 DSP Design in XEN

Figure 6 illustrates our implementation of DSP in the XEN system. Since most management operations of XEN are integrated in the *xm* tools in Domain0, we add two sub-commands in the *xm* tools, *dsp* and *undsp* to enable or disable DSP.

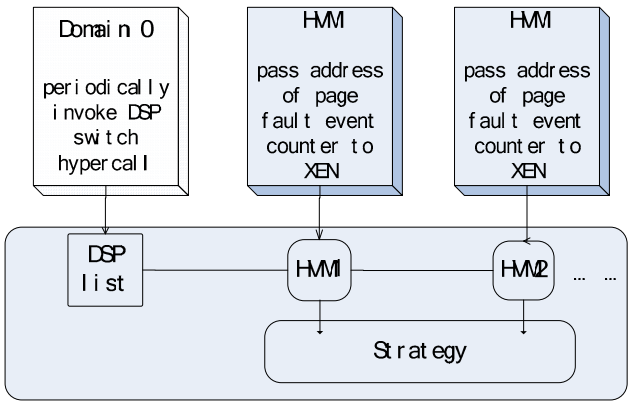

Figure 6. DSP implementation on XEN

We take advantage of the existing timer in XEN to sample a guest OS. In order to count the number of page faults, TLB misses, and retired instructions in the recent period of *T* seconds, we start a timer in the *xend* service process when executing the *xm dsp* command. The timer will invoke the corresponding XEN *hypercall* every *T* seconds, request the XEN hypervisor to collect those statistics and decide whether to change paging mode or not.

We get the number of TLB misses and retired instructions from the processor performance monitor unit (PMU) in the XEN hypervisor. To get the number of page faults, we add a kernel module in each guest OS of interest. When the guest OS starts up, the kernel module will notify the XEN hypervisor of the memory address of the variable that records the number of page faults in the guest OS. The XEN hypervisor can read the variable directly and get the number of page faults efficiently.

All virtual machines using DSP are organized as a list in the hypervisor. Whenever the timer invokes the *hypercall*, the XEN hypervisor will only collect samples for the virtual machines in the DSP list. A virtual machine will be removed from the list when it is destroyed, or when the command *xm undsp* is executed. If the list becomes empty, the timer in the *xend* process will be terminated. This implementation allows enabling or disabling DSP on each virtual machine independently.

The effectiveness of DSP is greatly dependent on the thresholds of page faults, TLB misses, and P-to-T ratios. The thresholds might be quite different on different hardware platforms. Our current approach relies on machine learning and profiling to locate appropriate thresholds for a specific machine. All thresholds can be customized by executing the *xm dsp* command with corresponding parameters. Thus, a user or a system administrator can choose a set of thresholds that fit his/her hardware platform.

3.2 Major Interface Functions

In our implementation, we extend an existing hypercall, *do_hvm_op,* with a new operation, *hvmop_dsp*. Both the kernel module in a guest OS and the DSP timer in the *xend* service process invoke *do_hvm_op* to interact with the XEN hypervisor.

The operation, *hvmop_dsp*, is called on the *hvmop_dsp* branch in the *do_hvm_op* hypercall. According to the parameters, *hvmop_dsp* will perform the following actions respectively.

1. Accept the memory address of the page fault event variable (counter) in a guest OS, and translate the memory address to the corresponding virtual memory address in the VMM.
2. Enable DSP on the target virtual machine and add it to the DSP list.
3. Retrieve the number of page faults, TLB misses, and retired instructions in the current sampling period, and calculate the corresponding frequencies. Call *process_dsp* to make a decision as to whether to change the paging mode according to the strategy introduced in Section 2.3.
4. Disable DSP on the target virtual machine by calling *paging_stop_switch*, and remove the virtual machine from the DSP list; and if the DSP list becomes empty, stop the DSP timer in the *xend* service process. *Paging_stop_switch* will also switch the VM back to the former paging mode before DSP is enabled.

Two functions, *paging_switchto_hap* and *paging_switchto_sp*, are implemented to fulfill paging mode switching. The function *paging_switchto_hap* performs switching from SP mode to HAP mode. In order to complete the switching, it destroys the existing shadow page table, loads the root address of the *p2m* map to the proper register in VMCS, and modifies the Secondary Processor-Based VM-execution Control Register to enable EPT. The function simply returns when the virtual machine is already in HAP mode.

The function, *paging_switchto_sp*, conducts switching from HAP mode to SP mode. In order to complete the switching, it saves the root address of the *p2m* map, rebuilds shadow page tables by constructing an initially empty shadow page tables, and modifies the Secondary Processor-Based VM-execution Control Register to disable EPT. When SP mode starts, the shadow page table will be filled by demand during execution. If the virtual machine is already in SP mode, the function simply returns.

4. Evaluation

In this section, we first run a set of experiments to learn the thresholds for DSP decisions. We then validate the thresholds with a different set of benchmarks or a different guest OS.

4.1 Experimental Environment

We conduct our experiments on a PC with an Intel i7-860 processor, 8GB of memory, and a 1TB SATA hard disk. All 4 cores of the i7-860 processor are enabled while disabling hyperthreading. The hypervisor we use is XEN 3.3.1. We patch our DSP implementation onto it.

Domain0 runs a 64-bit Linux OS, CentOS 5.4 x86_64, and is configured with 3 GB of memory and 2 CPU cores. We install two guest domains, Dom32 and Dom64, running a 32-bit and a 64-bit OS, respectively. Dom32 runs CentOS 5.4 i863 with 3 GB of memory and 1 CPU core. Dom64 runs CentOS 5.4 x86_64 with 3GB of memory and 1 CPU core.

We choose SPEC CPU2006, since the 29 benchmarks in the suite show a variety of memory behavior and DSP is intended to optimize memory virtualization. A memory and CPU intensive benchmark is more suitable than an I/O intensive benchmark to evaluate the effectiveness of DSP.

Table 1. 32-bit VM SPEC INT statistics

PF per 1K inst * 10^7	TLB miss per 1K inst	PF*10^7/TLB (col. 1/col. 2)	Winner
40	0.5	80	SP
125	0.5	250	HAP
0	< 0.5	0	Draw
> 150000	< 1	> 230000	HAP
10000	2.4	4500	HAP
4700	0.3	15000	Draw
1600	16	100	SP
90000	2.7	50000	HAP
0, sometimes 100	0.02	0, sometimes 40000	Draw
0 or 10000	0.3	0 or 30000	HAP
Sometimes>10000	frequently>10		SP
< 50	> 3.9	< 10	SP
> 150000	< 1	> 230000	HAP

4.2 Threshold Selection

In order to find out proper thresholds, we run SPEC INT2006 on Dom32 both in SP mode and in HAP mode. We collect the sample page fault frequency, TLB miss frequency, and historical P-to-T ratio every five seconds. For each benchmark, we select a typical sample value that dominates the whole benchmark. Table 1 lists these samples. Based on this table, we generate thresholds that will result in a correct decision for DSP in most cases. The final thresholds we pick are listed in Table 2.

For all samples values that can help select between HAP and SP, we take their average as the final threshold, expecting it will best fit other programs. We pick the most recent three samples to calculate the historical average ratios. We observe that the three sample points, which denote a 15-second interval, are sufficient to smooth a short-term change in a program. Due to the switching overhead, it is not worth performing switching when there is a short burst of page faults or TLB misses. However, a longer than 15 second interval may result in longer turnaround time. In other words, the system may stay in one mode for too long.

Table 2. Thresholds for DSP decision

	Upper-bound	Lower-bound
Page fault threshold	5000×10^{-7}	100×10^{-7}
TLB miss threshold	10	0.1
P-to-T ratio threshold	200×10^{-7}	150×10^{-7}
Interval for recent history	15 seconds (3 sample points)	

4.3 Sampling Interval Selection

Table 3 shows total TLB misses, page faults and execution times of *mcf* and *gcc* under HAP or SP only. Based on these statistics, we can estimate that the overhead of one TLB miss in HAP mode compared with SP mode is approximately 4 nanoseconds (roughly 12 cycles), and the overhead of one page fault in SP mode compared with HAP mode is around 10 microseconds. Based on the total execution times of the two benchmarks, switching from HAP to SP can save about 100 milliseconds per second on *mcf*, and switching from SP to HAP can save about 300 milliseconds per second on *gcc*. If switching can bring *mcf* or *gcc* to the best paging mode for more than one second, the benefit would overcome the overhead. As both *mcf* and *gcc* are the best cases that benefit most from proper switching, other benchmarks would have to stay in the best paging mode for a longer time to overcome the overhead of paging switching. Based the above analysis, we thus choose an interval at the granularity between one and 10 seconds in our experiments.

Table 3. Total TLB misses and page faults of *mcf* and *gcc*

	TLB miss	page fault	exec time
mcf in HAP mode	12539673680	243491	383
mcf in SP mode	12752691972	346439	329
gcc in HAP mode	1082065972	13320141	403
gcc in SP mode	3198642850	13302782	534

4.4 Overhead

Overhead of paging switching falls in two categories. One is the overhead of switching from SP mode to HAP mode, and the other is the overhead of switching from HAP mode to SP mode. To switch from SP to HAP, we simply load the EPT base address. To switch from HAP to SP, the shadow page table has to be rebuilt, and thus its overhead is larger than switching from SP to HAP.

In order to measure the overhead of switching from SP to HAP, we let the VM initially run in HAP mode. For every second, we invoke a hypercall operation (H-S-H), which will switch the VM from HAP to SP, and then, before returning back to the VM from the hypercall, immediately switch back from SP to HAP. Though the VM has been once in SP mode, but no instruction of the VM has been executed in SP mode. Therefore the shadow page table has never been actually used and it remains empty before switching back to HAP mode. The overhead of H-S-H would be larger than the overhead of single switching from SP to HAP.

Similarly, to measure the overhead of switching from HAP to SP, we let the VM initially run in SP mode. For every second, we invoke another hypercall operation (S-H-S), which will switch the VM from SP to HAP, and then switch immediately from HAP back to SP before returning to the VM. Since the shadow page table is completely destroyed when switching from SP to HAP, after switching back from HAP to SP, the shadow page table has to be rebuilt. The overhead of S-H-S would be larger than the overhead of switching from HAP to SP.

Table 4. Switching overhead of H-S-H

benchmark	Conf. Interval HAP	Conf. Interval H-S-H	#switches	ovhd (%)
perlbench	487.95 ± 0.40	488.62 ± 0.46	489	0.14
bzip2	859.54 ± 0.05	858.70 ± 0.11	859	-0.10
gcc	402.56 ± 3.37	406.38 ± 4.34	406	0.95
mcf	383.00 ± 1.98	380.23 ± 1.59	380	-0.72
gobmk	636.53 ± 0.23	636.38 ± 0.15	636	-0.02
hmmer	1240.70 ± 0.08	1240.06 ± 0.06	1240	-0.05
sjeng	727.12 ± 0.15	729.74 ± 0.18	730	0.36
libquantum	812.87 ± 0.79	813.00 ± 0.93	813	0.02
h264ref	1045.03 ± 0.35	1045.88 ± 1.01	1046	0.08
omnetpp	379.24 ± 0.29	378.86 ± 0.20	379	-0.10
astar	664.04 ± 0.58	663.30 ± 0.32	663	-0.11
xalancbmk	341.54 ± 0.60	341.46 ± 0.17	341	-0.02
bwaves	773.29 ± 1.55	771.44 ± 1.38	771	-0.24
gamess	1431.17 ± 0.56	1429.26 ± 0.68	1429	-0.13
milc	511.27 ± 0.84	511.02 ± 1.25	511	-0.05
zeusmp	839.16 ± 0.96	838.16 ± 1.29	838	-0.12
gromacs	1050.90 ± 0.85	1049.80 ± 0.62	1050	-0.11
cactusADM	1589.21 ± 3.80	1581.87 ± 3.57	1582	-0.46
leslie3d	891.25 ± 0.39	891.03 ± 0.85	891	-0.03
namd	640.94 ± 0.38	640.07 ± 0.17	640	-0.13
dealII	606.92 ± 0.09	605.67 ± 0.25	606	-0.21
soplex	354.31 ± 0.55	354.27 ± 0.36	364	-0.01
povray	332.41 ± 0.35	333.26 ± 1.25	333	0.26
calculix	1764.04 ± 0.83	1762.44 ± 0.33	1762	-0.09
GemsFDTD	836.53 ± 1.26	834.75 ± 0.83	835	-0.21
tonto	941.64 ± 0.46	940.91 ± 0.46	941	-0.08
lbm	405.74 ± 0.33	405.23 ± 0.50	405	-0.13
wrf	1145.45 ± 0.24	1144.22 ± 0.76	1144	-0.11
sphinx3	708.50 ± 2.13	709.11 ± 1.45	709	0.09

We run each benchmark seven times, respectively, at HAP, SP, H-S-H, and S-H-S modes. To avoid measurement noise and to report measurement errors, we pick the top 6 execution times and calculate the confidence intervals. Columns 2 and 3 in Tables 4 and 5 show the confidence intervals of execution times. The number of switches (column 4) is from the run with median execution time which is roughly equal to the mean execution time since switching is performed every second. The percentage overhead (column 5) for H-S-H is measured as ($mean_{H\text{-}S\text{-}H} - mean_{HAP}$) / $mean_{HAP}$ *100%. The overhead calculation for S-H-S is similar.

As shown in Table 4, the average overheads of H-S-H on most benchmarks are negative although they are all within one percent. Benchmark *gcc* yields the largest overhead which is still less than 1%. We notice that the confidence intervals of HAP and H-S-H overlap for a majority of benchmarks where the overheads are shown in grey. So we can conclude that the overhead of H-S-H is statistically negligible. However, since the overheads are dominantly negative, we suspect H-S-H may have some slightly positive impact on cache or TLB, which calls for further investigation.

Table 5. Switching overhead of S-H-S

benchmark	Conf. Interval SP	Conf. Interval S-H-S	#switch	ovhd (%)
perlbench	488.29 ± 0.52	494.32 ± 0.58	494	1.23
bzip2	858.05 ± 0.07	861.11 ± 0.19	861	0.36
gcc	534.16 ± 0.19	539.27 ± 0.30	539	0.96
mcf	329.11 ± 0.70	345.03 ± 0.86	345	4.84
gobmk	634.04 ± 0.04	636.94 ± 0.12	637	0.46
hmmer	1237.54 ± 0.10	1240.29 ± 0.08	1240	0.22
sjeng	723.47 ± 0.24	736.44 ± 0.17	736	1.79
libquantum	806.05 ± 0.80	811.21 ± 0.35	811	0.64
h264ref	1043.31 ± 0.68	1046.59 ± 0.61	1047	0.31
omnetpp	371.01 ± 0.25	374.06 ± 0.13	374	0.82
astar	648.26 ± 2.89	649.83 ± 0.12	650	0.24
xalancbmk	336.47 ± 0.19	341.78 ± 0.38	342	1.58
bwaves	790.95 ± 2.60	813.52 ± 3.93	814	2.85
gamess	1427.78 ± 0.50	1430.66 ± 0.83	1430	0.20
milc	547.50 ± 0.91	566.16 ± 1.16	566	3.41
zeusmp	816.55 ± 0.20	836.19 ± 0.83	836	2.40
gromacs	1047.90 ± 0.49	1049.94 ± 0.56	1050	0.19
cactusADM	1343.24 ± 4.64	1370.80 ± 3.14	1371	2.05
leslie3d	900.24 ± 0.39	907.33 ± 0.45	907	0.79
namd	639.34 ± 0.31	640.83 ± 0.22	641	0.23
dealII	614.32 ± 0.55	618.65 ± 0.49	619	0.70
soplex	368.44 ± 0.30	376.07 ± 0.46	376	2.07
povray	332.02 ± 0.11	332.87 ± 0.19	333	0.26
calculix	1790.88 ± 0.67	1799.67 ± 0.69	1800	0.49
GemsFDTD	845.10 ± 0.83	904.73 ± 0.83	905	7.06
tonto	1129.54 ± 0.81	1137.89 ± 1.50	1138	0.74
lbm	404.75 ± 2.11	417.98 ± 0.40	418	3.27
wrf	1181.33 ± 0.18	1198.07 ± 0.66	1198	1.42
sphinx3	701.91 ± 0.91	705.42 ± 1.67	705	0.50

S-H-S causes more noticeable overhead as destroying and restoring the shadow page tables definitely cost time. Particularly, for the benchmarks with large working set and thus likely large page table, the overhead is more prominent. So it is not a surprise for *GemsFDTD* and *mcf*, the two benchmarks with the largest working set sizes in SPEC CPU2006, to take the top two spots on overhead [7]. For all other benchmarks, the overheads are all below 5% and most of them are within 1%.

Note that we report an extreme upper bound on overhead here. Our actual implementation suggests a 5-second interval and, for all benchmarks, DSP performs only a couple of dozens of switching at most. Compared to hundreds of switching in Tables 4 and 5, the actual overhead would be negligible.

4.5 Validation Using SPEC INT

We apply the thresholds picked in Section 4.2 back to SPEC INT2006. Figure 7 shows the results where the bars are normalized execution times. The performance of DSP can almost always reach the better case of HAP and SP. For each individual benchmark, the performance difference between DSP and the higher one of HAP and SP is within 1%. For *gcc*, the performance of DSP is 34% better than SP, and for *mcf*, the performance of DSP is 13% better than HAP. The "total" bars in Figure 7 show the normalized total execution time of all integer benchmarks. DSP shows a slight improvement over both HAP and SP.

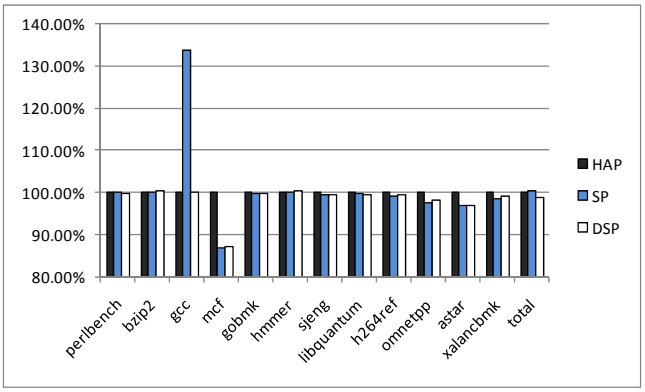

Figure 7. SPEC INT on 32-bit VM.

Figure 8 shows the distribution of execution time of each SPEC CPU2006 benchmark in HAP mode and in SP mode, when DSP is enabled. We also report the floating pointing benchmarks here for comparison purpose. The numbers over the bars are the number of times of paging switching. Note that, for most benchmarks, one mode dominates the execution time. Often there are no or only a few switches. As discussed in Section 4.4, the overhead of switching is very small, thus, the overhead of the DSP mechanism itself is negligible. Once a proper paging mode is selected, the performance gain comes directly from the mode itself.

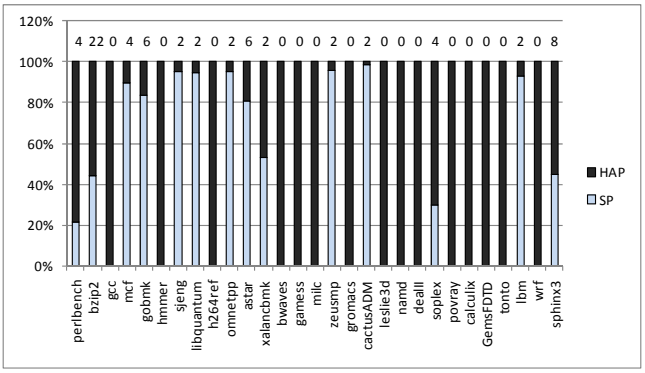

Figure 8. Time distribution of each benchmark in different paging mode (5s interval)

All experimental results in this section so far are based on a 5-second sampling interval. We further compare the performance of

DSP under three different intervals, 2 seconds, 5 seconds and 10 seconds. As shown in Figure 9, the differences among 2-second, 5-second and 10-second intervals are not significant. Note that the number of page faults and TLB misses are measured on a per thousand instruction basis. They do not vary remarkably for most benchmarks among these three intervals. Moreover, most SPEC CPU2006 benchmarks have distinct phases that are longer than 10 seconds, DSP would make the same decision under different timer intervals.

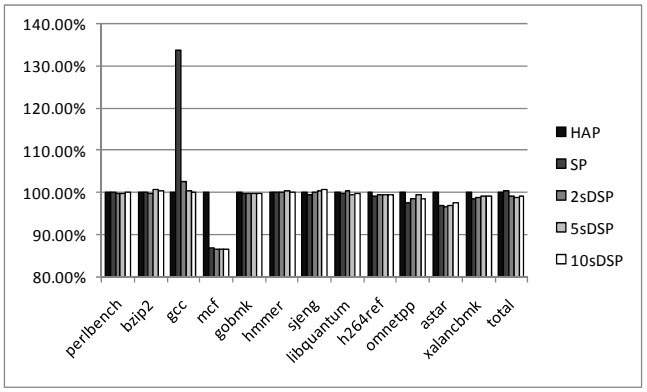

Figure 9. Effects of length of sample intervals

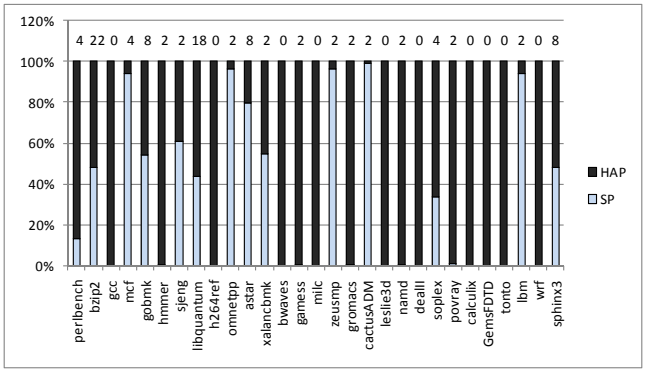

Figure 10. Time distribution of each benchmark in different paging mode (2s interval)

Figures 10 and 11 show that the distribution of execution time and the number of mode switches under the 2-second and 10-second intervals, respectively. Comparing Figure 8 with Figures 10 and 11, we can see that the distribution is almost the same for each benchmark except for *libquantum*, and the number of switches can slightly vary. For *libquantum*, DSP spends over a half of time in HAP mode with the 2-second interval, while only 6% of time with the 5-second or 10-second interval. We observe that *libquantum* shows frequent yet short bursts of page faults. DSP under the 2-second interval switches to HAP when encountering a burst and switches back to SP in the next interval. While under the 5-second and 10-second intervals, DSP stays in SP mode most of time because the bursts are smoothed in a longer interval.

In general, the 2-second interval typically causes the most mode switching while the 10-second interval the least. So we observe some slight performance difference among the three intervals. It is basically a tradeoff between switching overhead and performance gain due to paging switching. As shown in Figure 9, *bzip2*, *sjeng*, *astar* and *xalancbm* perform slightly better under the 2-second interval while *gcc* and *libquantum* prefer the 5-second interval.

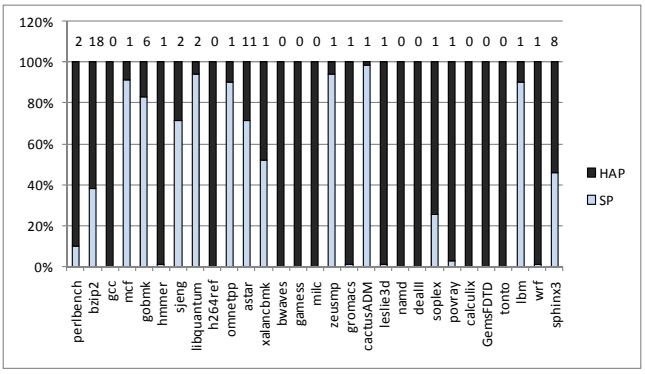

Figure 11. Time distribution of each benchmark in different paging mode (10s interval)

4.6 Validation Using SPEC FP and 64-bit System

This section first validates the same set of thresholds using SPEC FP2006 in Dom32 with DSP enabled and then using a 64-bit guest for all SPEC CPU2006 benchmarks.

Figure 12 shows the results for SPEC FP under a 32-bit guest. Again, DSP always matches the better performance of HAP and SP. DSP achieves a 2% improvement over SP and HAP for SPEC FP. It obtains both the 15% performance gain by SP for *cactusADM* and the 22% gain for *tonto* by HAP.

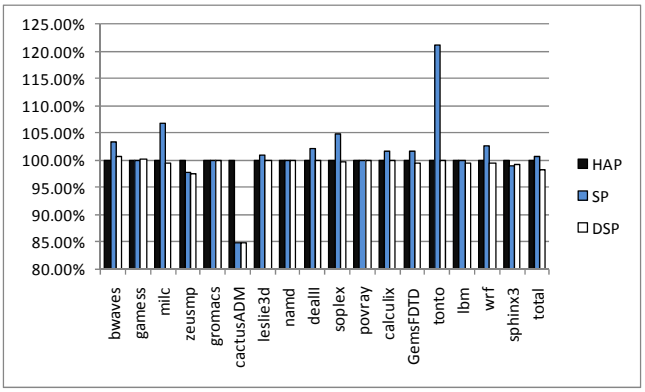

Figure 12. SPEC FP on 32-bit VM

Figures 13 and 14 show the results on the 64-bit guest. Although HAP loses it advantage for most benchmarks, DSP is still able to catch the better performance of the two modes. For the two benchmarks, *gcc* and *milc*, where HAP shows a notable performance advantage over SP, DSP is able to keep the system in HAP mode. For the benchmarks where SP shows significant advantages, DSP switches to SP mode for a majority of time. For *cactusADM*, DSP indeed shows a slight improvement over SP, which performs 14% better than HAP. Benchmark *wrf* cannot be compiled on the 64-bit OS and thus is not reported here.

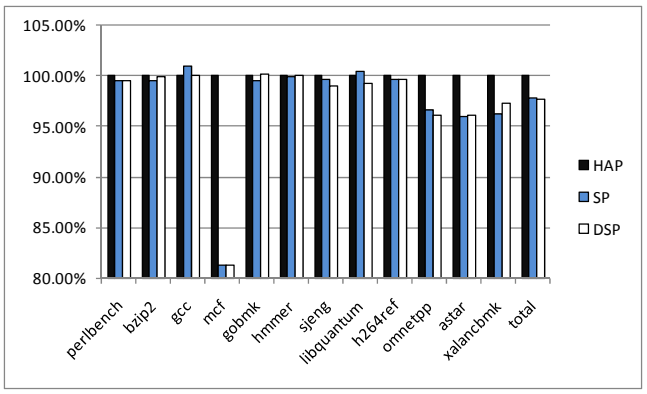

Figure 13. SPEC INT on 64-bit VM

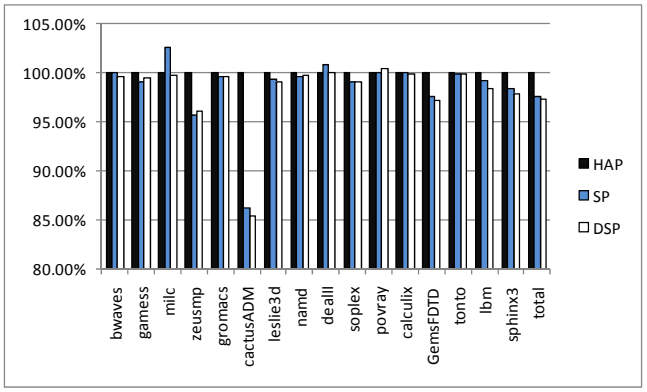

Figure 14. SPEC FP on 64-bit VM

5. Related Work

Keith Adams and Ole Agesen describe an implementation of a software VMM that employs Binary Translation (BT) for x86 guest execution [1]. Their experimental results show that BT outperforms traditional shadow paging with a previous generation of hardware-assisted virtualization available at the point of their research. They also survey software and hardware opportunities on virtualization, pointing out the potential problem of page walk for hardware MMU support such as AMD's "nested paging" and Intel's EPT, and mention several ways to optimize. One solution proposed by VMware is to reduce the overhead of page walking by applying only large pages both in Guest OS and in VMM, and therefore, TLB misses will be reduced dramatically [10, 11]. However, large pages can cause memory pressure and fragments. Gillespie also provides a comparison between EPT and shadow paging but offers no design and implementation details [6].

Ravi Bhargava *et al.* present an in-depth examination of the 2D page table walk of AMD's "nested paging" and options for decreasing the page walk penalty [4]. They use the processor's *page walk cache* to exploit the strong reuse of page entry references. By applying large pages with *page walk cache*, the hypervisor can improve the guest performance by 3%-22%. Barr *et al.* later explore the design space of MMU caches [3].

Hardware assisted virtualization has become more mature nowadays than its initial version. Neither software virtualization nor hardware assisted virtualization will win in all cases. Our solution is based on two mature techniques, shadow paging mode and hardware assisted paging mode, and involves no compatibility issue. It can also work orthogonally with large page mode for more performance improvement.

As another attempt, Wang *et al.* propose *Dynamic Memory Para-virtualization* (DMP) to improve the performance of memory virtualization in [13]. For a VM in SP mode, DMP instruments the VMM to dynamically modify the Guest OS kernel binary code, so as to make use of the guest page table to find machine addresses. Host machine addresses are filled in the page table entries instead of guest physical memory addresses. When the guest OS reads or writes these page table entries, the modified kernel code is executed to translate host machine addresses back to guest physical addresses for the guest to use. DMP can achieve close to native performance. However, DMP introduces security issues, which prevents it from being applied in a security-sensitive public service environment. Our DSP solution brings the benefits in performance without any security risks.

6. Discussion and Conclusion

To reduce the overhead of memory virtualization, both Intel and AMD have provided hardware virtualization support. Unfortunately, hardware assisted virtualization does not always bring performance benefit. In this paper, we compare the performance of hardware-assisted paging and conventional shadow paging, and show that either of them can significantly outperform the other in certain benchmarks. We design and implement an effective paging scheme that dynamically switches between the two page modes and thus exploits the advantages of both. Based on online sampling of TLB misses and guest page faults, we emulate a machine learning process to pick a set of thresholds to help DSP make choices in paging modes. We implement this system in the open source XEN environment and share our patch with the research community. The code patch is now available at http://www.cs.mtu.edu/~zlwang/dsp.tar.gz. We plan to automate the threshold selection using support vector machines and compare it with our manual selection results. Our experimental results show that HAP is not as effective in the 64-bit system as in the 32-bit system. It is not a surprise as the 64-bit system doubles the levels of the page table and thus the penalty of page walk. It is worth a further study in effective hardware support for 64-bit systems.

Acknowledgments

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2007CB310900, National Science Foundation of China under Grant No.90718028 and No.60873052, National High Technology Research 863 Program of China under Grant No.2008AA01Z112, and MOE-Intel Information Technology Foundation under Grant No.MOE-INTEL-10-06. Zhenlin Wang is also supported by NSF Career CCF0643664. We also owe thanks to the anonymous reviewers and Steve Carr for their valuable comments and suggestions.

References

[1] K. Adams and O. Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. In *ASPLOS '06: Proceeding of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006 .

[2] P. Barham, B. Dragovic, K. Fraser, *et al*. Xen and the art of virtualization. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. p. 177, 2003.

[3] T. W. Barr, A. L. Cox, and S. Rixner. Translation Caching: Skip, Don't Walk the Page Table. In *ISCA '10: Proceedings of International Symposium on Computer Architecture*, 2010.

[4] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating Two-Dimensional Page Walks for Virtualized Systems. In *ASPLOS*

'08: Proceeding of International Conference on Architectural Support for Programming Languages and Operating Systems, 2008.

[5] S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. *US Patent*, 6397242, Oct. 1998.

[6] M. Gillespie. Best Practices for Paravirtualization Enhancements from Intel® Virtualization Technology: EPT and VT-d. June 1, 2009. http://software.intel.com/en-us/articles/best-practices-for-paravirtualization-enhancements-from-intel-virtualization-technology-ept-and-vt-d/.

[7] D. Gove. CPU2006 working set size. In *SIGARCH Comput. Archit. News*, 35(1):90–96, 2007.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin and A. Liguori. kvm: the Linux virtual machine monitor**.** In *Linux Symposium*. Jan, 2007.

[9] SPEC. http://www.spec.org.

[10] VMware. Performance Evaluation of Intel EPT Hardware Assist. Copyright® VMware. Inc.

[11] VMware. Large Page Performance: ESX Server 3.5 and ESX Server 3i v3.5, Copyright® 2008 VMware. Inc.

[12] C. Waldspurger. Memory resource management in VMware ESX serve. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*. Dec, 2002.

[13] X. Wang, Y. Sun, Y. Luo *et al*. Dynamic memory paravirtualization transparent to guest OS. In *Science China Information Sciences*, January 2010, 53(1): 77-88.

[14] Xen. http://www.xen.org.