IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

Perfect-Xen: A Framework for Performance Counter Virtualization

Asst. Prof. Poonam Gholap[1], Asst. Prof. Rakhee Kundu[2], Asst. Prof. Snehal Mane [3]

Computer Engineering, VESIT, Affiliated to Mumbai University, India [1, 2, 3]

Abstract: Virtualization is a powerful technique used for variety of application domains, including merging cloud environments that provide access to virtual machines as a service. This paper discusses the challenges of performance monitoring inherent to virtualized environments and introduces a technique to virtualize access to low-level performance counters on a per-thread basis. The technique was implemented in perfctr-xen, a framework for the Xen hypervisor that provides an infrastructure for higher-level profilers. This framework supports both accumulative event counts and interrupt-driven event sampling .Paper presents experimental results based on micro benchmarks and SPEC CPU2006 macro benchmarks that show the accuracy and usability of the obtained measurements when compared to native execution.

Keywords: Virtualization, Framework, Hypervisor, Virtual Machine.

I. INTRODUCTION

Virtualization allows multiple instances of an operating system to run on a single computer. Virtualization is the creation of a virtual (rather than actual) version of something, such as an operating system, a server, a storage device or network resources. We probably know a little about virtualization if we have ever divided our hard drive into different partitions. a partition is the logical division of a hard disk drive to create, in effect, two separate hard drives. Operating system virtualization is the use of software to allow a piece of hardware to run multiple operating system images at the same time. The technology got its start on mainframes decades ago, allowing administrators to avoid wasting expensive processing power.

APPLICATIONS OF VIRTUALIZATION:

DESKTOP:

Desktop virtualization is the concept of separating the logical desktop from the physical machine. One form of desktop virtualization, virtual desktop infrastructure (VDI), can be thought as a more advanced form of hardware virtualization. Rather than interacting with a host computer directly via a keyboard, mouse, and monitor, the user interacts with the host computer using another desktop computer or a mobile device by means of a network connection, such as a LAN, WLAN or even the Internet. In addition, the host computer in this scenario becomes a server computer capable of hosting multiple virtual machines at the same time for multiple users. [2]

OPERATING SYSTEM-LEVEL VIRTUALIZATION:

Operating system-level virtualization is a server virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances (often called containers, VEs, VPSs or jails) may look and feel like a real server, from the point of view of its owner.

Network virtualization:

Network virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network. Network virtualization involves platform virtualization, often combined with resource virtualization.

ADVANTAGES VIRTUALIZATION:

- Server consolidation
- Reduced power and cooling
- Green computing
- Ease of deployment and administration
- High availability and disaster recovery

POPULAR VIRTUALIZATION PRODUCTS INCLUDE ALSO KNOWN AS HYPERVISOR: VMWARE

- Microsoft Hyper-V
- Virtual Iron
- Xen

PERFCTR-XEN SUPPORT FOLLOWING TYPE OF VIRTUALIZATION MODES:[4]

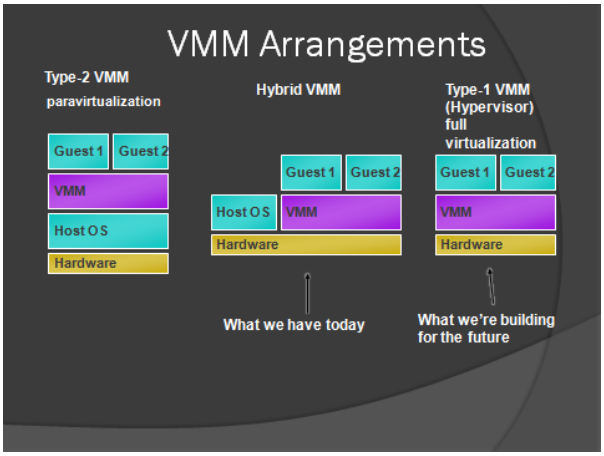

Fig: 1 VMM Arrangements

1. FULL VIRTUALIZATION:

In computer science, full virtualization is a virtualization technique used to provide a certain kind of virtual machine

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

environment, namely, one that is a complete simulation of the underlying hardware. Full virtualization requires that every salient feature of the hardware be reflected into one of several virtual machines – including the full instruction set, input/output operations, interrupts, memory access, and whatever other elements are used by the software that runs on the bare machine, and that is intended to run in a virtual machine. In such an environment, any software capable of execution on the raw hardware can be run in the virtual machine and, in particular, any operating systems.

2. PARAVIRTUALIZATION:

In computing, para virtualization is a virtualization technique that presents a software interface to virtual machines that is similar but not identical to that of the underlying hardware.

The intent of the modified interface is to reduce the portion of the guest's execution time spent performing operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment. The para virtualization provides specially defined 'hooks' to allow the guest(s) and host to request and acknowledge these tasks, which would otherwise be executed in the virtual domain (where execution performance is worse). A successful para virtualized platform may allow the virtual machine monitor (VMM) to be simpler (by relocating execution of critical tasks from the virtual domain to the host domain), and/or reduce the overall performance degradation of machine-execution inside the virtual-guest.

3. HARDWARE-ASSISTED VIRTUALIZATION

The hardware-assisted virtualization is orthogonal to para or full virtualization, and it can be used for the both. Advantages of The hardware-assisted virtualization provide virtual machine monitors (VMM) with simpler and robust implementation. Full-virtualization can be implemented by software only, as we see such products such as VMware as well as Virtual PC and Virtual Server from Microsoft today. However, hardware-assisted virtualization such as IntelR Virtualization Technology (simply IntelR VT hereafter) can improve the robustness, and possibly performance. Disadvantages of Obviously hardware-assisted virtualization requires a
system with the feature, but it is sensible to assume that hardware-assisted virtualization is available on almost all new x86-64-based systems.

4. HYBRID-VIRTUALIZATION:

Hybrid-virtualization that we propose is technically para-virtualization for hardware-assisted virtualization. However, we use this terminology to avoid any confusion caused by the connotation from software-only para-virtualization. And the critical difference is that hybrid-virtualization is simply a set of optimization techniques for hardware-assisted full-virtualization.

II. PROBLEM DEFINITION AND METHODOLOGY ADAPTED

This paper shows comparisons of two Virtualizing hardware event counters perfctr and perfctr xen each is described in detail as follows:

PERFCTR

We chose perfctr because it is widely used and provides the foundation for higher-level libraries and frameworks such as PAPI, HPCToolkit, or PerfExplorer, as shown in Figure 3. It is efficient, lightweight and allows direct access to performance counters in user mode. Perfctr supports a wide range of x86 implementations spanning multiple generations and different vendors, whose hardware event counter implementation can differ significantly. In addition, perfctr works on non-x86 platforms such as PowerPC and ARM and can easily be integrated in any Linux distribution. Perfctr consists of a kernel driver and a user-level library. The kernel driver maintains performance counter-related per-thread data structures, updates them on each context switch, and makes them available to the user-level library via a read-only mapping. Besides miscellaneous architecture-specific information, this perthread data structure contains the following information:

• Control State. Information about which PMU data registers a thread is actively using, which events these registers count, and to which physical register address they are mapped. Similar information is kept with respect to the use of the time-stamp counter, which is also virtualized. Counter State. For each PMU data register, as well as the TSC register, two values are kept: Sumthread, which reflects the thread's accumulated logical event count up to including the last suspension point; and Startthread, which reflects the sampled value of the counter at the last resumption point. Perfctr supports two types of counters: a-mode and i-mode counters. A-mode counters are used by threads to measure the number of events occurring in some region of a program. Before coming to understand how perfctr works we need to understand what it consist of MSRs(machine Specific Registers, First, PMU (Performance Monitoring Unit) configuration registers (e.g., event selectors) need to be re-programmed to reflect the desired event configuration of the thread to be resumed. Second, if the performance counter register contains the logical value of the thread to be resumed, it must be restored (and the value of the outgoing thread must be saved). Otherwise, its value must be sampled and recorded in the corresponding data structure for the thread or domain. When a thread wants to access the logical value of a counter at time t, a user library function issues a RDTSC or RDPMC instruction to obtain the register's physical value Phys(t) and computes the logical value Logthread(t) as

$$\text{Logthread}(t) = \text{Sumthread} + (\text{Phys}(t) - \text{Startthread}) \quad (1)$$

On each context switch, the perfctr kernel driver updates the accumulated value of the outgoing thread as

$$\text{Sumthread} \leftarrow \text{Sumthread} + (\text{Phys} - \text{Startthread})$$

to account for the events during the last scheduling period. In addition, the Startthread value
of the thread to be resumed is reset as Startthread ← Phys. Note that the actual physical register value is not changed on a context switch for a-mode counters. I-mode counters, which are used for sampling, trigger interrupts after a certain number of events has occurred, which represents the sampling period. Since the value at which an overflow interrupt is triggered is fixed at 0 and cannot be

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

programmed, the physical register must be set to a small negative value whose absolute value represents the desired length of the sampling period. I-mode counters are treated differently during a context switch: their physical value is saved on suspend and restored on resume. The Sumthread field maintains the counter's accumulated logical value as for amode counters. The Startthread field is used to record the physical value when a thread is suspended. Consequently, the logical value of an i-mode counter can also be obtained using equation (1).

When an overflow occurs, perfctr handles this interrupt, identifies the register(s) that have overflowed and updates Sumthread, then disables further event counting for these registers Before coming to understand how perfctr works we need to understand what it consist of MSRs (machine Specific Registers, First, PMU (Performance Monitoring Unit) configuration registers (e.g., event selectors) need to be re-programmed to reflect the desired event configuration of the thread to be resumed. Second, if the performance counter register contains the logical value of the thread to be resumed, it must be restored (and the value of the outgoing thread must be saved).Otherwise, its value must be sampled and recorded in the corresponding data structure for the thread or domain.

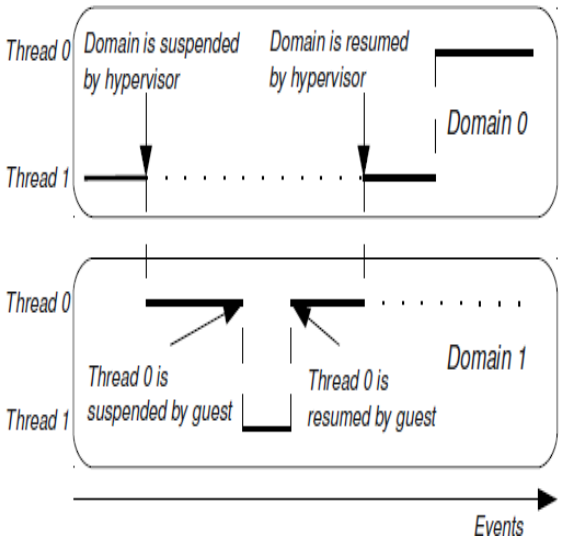

Fig 2: context switching in virtualized environment

States	Sumthread= Sumthread+ (Phys(t) −Start thread)	Startthread =Phys	Phys (t)	Logthread =Sum thread+ (Phys(t) − Start thread)
Thread start	0	10		
Thread suspend	2		12	0+(12-10)=2
Thread resume	2	14		
Thread suspend	4		16	2+(16-14)=4

Table1: A- mode counter (native)

A-MODE COUNTERS

- Log thread(t) = Sum thread + (Phys(t) − Start thread)
- Sum thread =Sum thread+(Phys−Start thread)
- Start thread = Phys

	Sum thread	Start thread= Phys (when thread is suspended)	Log thread= Sum thread + Phys(t)-Start thread
T0	0	-5	0+(-5)-(-0)=5
	5(store)	-5(store)	
T1	5(restore)	-5(restore)	
T2	5	-5	5+((-3)-(-5))=7
		-3(-5+2)	

Table2: I-mode counter (native)

I-mode counters

- Phy reg loaded with negative value , Saving and storing at each switch
- Log thread=Sum thread =Sum thread+(Phys−Start thread)
- Start thread=Phys when thread is suspended

Disadvantage of perfctr :

We first considered having the hypervisor update each thread's counter state directly on the guest kernel's behalf. This approaches the advantage that no changes to the perfctr user library are required. However, it would create undesirable coupling between the hypervisor and the guest kernel implementations, because the hypervisor would need to traverse guest kernel data structures.

THE PERFCTR-XEN FRAMEWORK

Perfctr-xen includes a hypervisor driver, a guest kernel driver, and a modified user-level library. perfctr-xen supports both paravirtualized mode as well as hardware virtualization mode due to perfctr disadvantages, we decided to split the control and counter state in two parts. At the guest kernel level, a per-thread data structure is maintained. At the hypervisor level, a per-VCPU data structure is maintained for each virtual CPU that is assigned to a guest domain. The hypervisor provides read-only access to this data structure to the guest kernel, who in turn maps it into the address space of each thread using performance counters.

The per-VCPU data structure is modeled after the per-thread data structure used in the native version of perfctr For each PMU data register, as well as forthe TSC register, the hypervisor maintains two values per VCPU: Startvcpu and Sumvcpu.

Startvcpu represents the sampled value of the counter at the most recent resumption point of the domain or thread (whichever happened last). If the hypervisor resumes a domain,it directly updates Startvcpu after sampling the counter. If the guest kernel resumes a thread, it requests via a hypercall that the hypervisor record the sampled

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

value in Startvcpu. The same hypercall is also used to activate this thread's counter-related control state.
The field Sumvcpu represents the cumulative number of events incurred by this domain since the last intra-domain thread resumption point until the most recent domain suspension point. It is set to zero on each intra-domain switch during the hypercall that notifies the hypervisor that the guest kernel resumed a thread. On each inter-domain context switch, the perfctr-xen hypervisor driver updates the accumulated value of the outgoing VCPU as

$$\text{Sumvcpu} \leftarrow \text{Sumvcpu} + (\text{Phys} - \text{Startvcpu}) \quad (1)$$

to account for the events incurredsince the last intra- or inter-domain resumption point.The perfctr-xen guest kernel driver maintains the value Sumthread for each thread as in the native case, which represents the cumulative number of events up to the last thread suspension point. A counter's logical value at time t is computed as

$$\text{Logthread}(t) = \text{Sumthread} + (\text{Phys\_}(t) - \text{Start\_thread}) \quad (2)$$

Phys_(t) represents the adjusted physical value that accounts for possible VCPU preemption, which is computed as

$$\text{Phys\_}(t) = \text{Sumvcpu} + (\text{Phys}(t) - \text{Startvcpu}) \quad (3)$$

Thus, the logical value represents the sum of the cumulative number of events until the last thread suspension point, plus the number of events encountered from there until the last domain resumption point while the domain was active, plus the events encountered since then until t, reduced by an adjusted start value Start_thread. The adjusted thread start value Start_thread compensates for the requirement that each intra-domain context switch includes a hypercall. Since this hyper call is introduced by our framework, we wish to exclude any events occurring during its execution. Right before resuming a guest thread, the guest kernel driver computes
Start_thread = Phys_(tr) (after returning from the hypercall at time tr)

$$\text{Start\_ thread} = \text{Sumvcpu} + (\text{Phys}(tr) - \text{Startvcpu}) \quad (4)$$

This adjustment excludes any events incurred between when the hypervisor sampled the counter during the hypercall and tr.

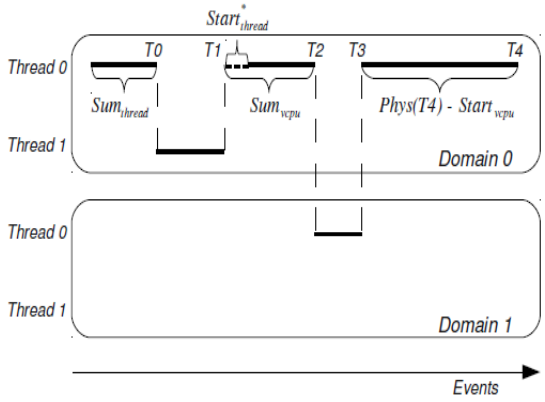

Fig:3 example scenario for virtualized perfctr counter (perfctr-xen)

The inclusion of the term Sumvcpu ensures that all such events are excluded, even if the domain was suspended and resumed during the hypercall by the preemptive scheduler.

- Sum vcpu =Sum vcpu + (Phys−Start vcpu)
- Log thread(t) = Sum thread + (Phys*(t) − Start*thread)
- Phys*(t) = Sum vcpu + (Phys(t) − Start vcpu)
- Start*thread = Phys*(tr) after returning from the hypercall at time tr
- Start*thread = Sum vcpu + (Phys(tr) − Start vcpu)

SPEC CPU2006 BENCHMARKS

We used the SPEC CPU2006 benchmarks as macrobenchmarks to show the correctness of our implementation and provide error esti- mates for CPU and memory bounded workloads. Native mode execution is again used as reference point. Since Dom0 is a paravirtualized domain in Xen, we used the Dom1 and Dom2 domains for tests that include fully-virtualized domains. (To exclude any possible effect of Dom0, we pinned it to a dedicated core.) We considered 5scenarios: (1) Native mode execution. (2) Fully-virtualized domains Dom1 and Dom2, each running on a dedicated core (DC). (3) Fully-virtualized domains Dom1 and Dom2 running on the same core (SC). (4) Paravirtualized domains Dom0 and Dom1, each running on a dedicated core (DC). (5) Paravirtualized domains Dom0 and Dom1 running on the same core (SC). The official SPEC distribution contains a large set of different benchmarks. We ran all of them using the 'train' problem size and recorded the total number of events counted during their execution. Since some benchmarks were executed under different data sets, we calculated the cumulative event counter values for all data sets.
We present results for a subset of benchmarks only, choosing those for which both a non-negligible number of events was counted and for which the difference between the scenarios was largest; these represent the relative weakest performance of our framework. In Figure 5, the results for the cycle counts reported by the virtualized TSC are shown. If the benchmarks execution were unaffected by virtualization, and if our framework achieved the same accuracy as perfctr running natively, we would expect to obtain the same results for all test scenarios for a given benchmark. This is true for most benchmarks, although 3 benchmarks (mcf, astar, and lbm) show significant deviations for the fully virtualized configuration.

When counting the number of instructions retired (Figure 7), we did not observe any significant differences. Figures 6 and 8 display the number of L2 cache references and misses, respectively. Since these events are more strongly influencedby environmental factors inherent to the virtualized environment, they show slightly larger deviations, particularly for the number of cache misses. For example, libquantum shows a significant drop in the number of cache misses observed, although the number of cache references is roughly the same. These effect warrant further investigation to ascertain if they indeed reflect environmental circumstances or are caused by inadvertent interactions with the measurement framework.

DOI 10.17148/IARJSET.2015.21210

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

Time	Phys (t)	Start vcpu =Phys	Sumvcpu=0 (intradomain) Sumvcpu= Sumvcpu+ (Phys-Startvcpu) (interdomain)	Phys*(t)= Sumvcpu+ (Phys(t)- startvcpu)	Phys*(tr)= Start*thread= Sumvcpu+Phys (tr)-Startvcpu	Logthread(t)= Sumthread+ (Phys*(t)- start*(t))	Sumthread = Log thread
T0	10	0	0	10+(10 -0)=20	10+0-0=10	0+(20-10)=10	0
			0+10-0=10				10
T1		13	0 (intradomain switch)		0+(14-13)=1		
T2	20		0+(20-14)=6 (interdomain switch)	6+(20-13)=13	6+(14-13)=7	10+(13-7)=16	16
T3		22	6				
T4	30		6+(30-22)=14	14+(3022)=22	14+(22-22)=14	16+(22-14)=24	24

Table3: example for A-mode (perfctr-xen)

Time	Startvcpu=Phys(intradomain) Startvcpu=Startthread(interdomain)	Sumthread= Sumvcpu	Startthread=Startvcpu (intradomain)Startthread= Negative sample period(interdomain)	Sumvcpu=Sumvcpu + (Phys-Startthread)
T0	-10	0	-5(saved)(step4)	0 (step1)
	-5(phys as intradomain)(stp2)	5(saved) (Step3)		0+(-5-(-10))=5
T1	-5(restored)	5(restored)		
T2	-5	5	-5	5+(-2-(-5))=8
	-2(-5+3)	2(saved)	-2(saved)	
T3	-2(restored)	8 (restored)		
		8	-2	
T4	-2	8	-2=(-5-(-3))	
	-2 (phys as interdomain)			

Table4: example for I-mode (perfctr-xen)

SPEC CPU2006 BENCHMARKS

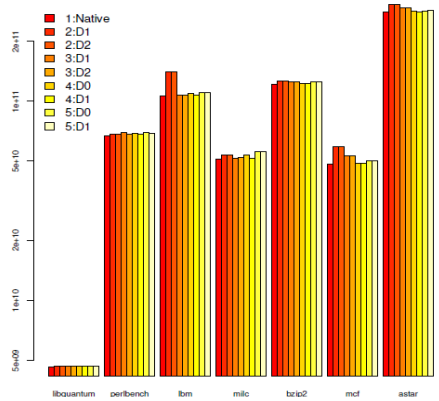

Fig5. SPEC CPU2006, Time Stampcounter

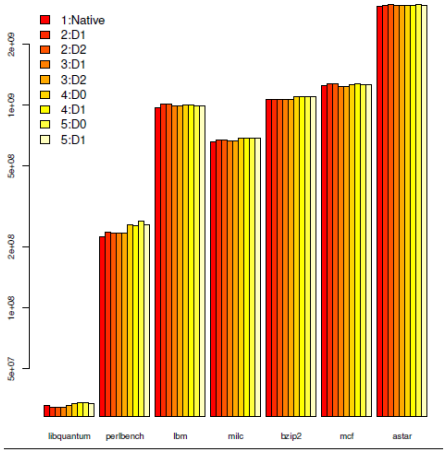

Fig 6. SPEC CPU2006, L2 Cache

DOI 10.17148/IARJSET.2015.21210

IARJSET

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 12, December 2015

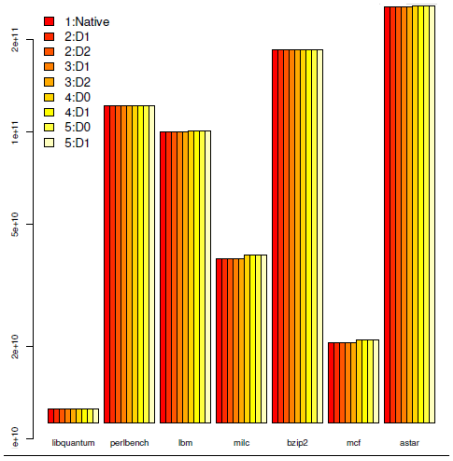

Fig7. SPEC CPU2006, Instructions Retired

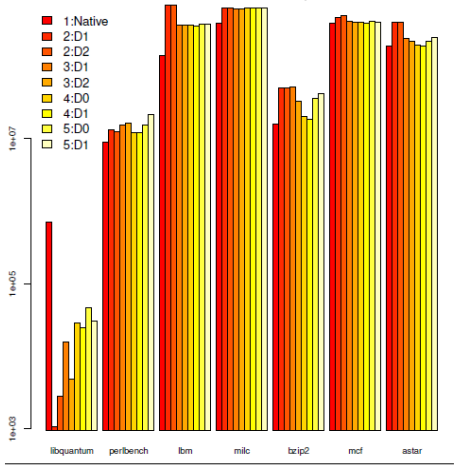

Fig8. SPEC CPU2006, L2 Cache Misses

III. CONCLUSION

This paper presented perfctr-xen, a novel performance counter framework for the Xen hypervisor which we have developed. perfctr-xen extends the existing perfctr framework so it can be used in virtual machine environments running under the Xen hypervisor. perfctr-xen supports both paravirtualized guest and guests using hardware-based virtualization. It provides a hybrid mode in which paravirtualization techniques are applied to hardware-assisted guest virtual machine. The technical contributions of this paper are the following: (1) application of an offsetting technique that allows direct access to logical per-thread counter values from user mode while avoiding the costs ssociated with saving and restoring physical PMU data registers; (2) the optimization of guest and hypervisor communication to minimize and amortize the costs associated with their coordination, while avoiding the costs of trapping and emulating counter-related instructions; (3) a technique for increasing the accuracy of performance monitoring by correcting for monitoring overhead. Perfctr-xen enables the use of higher-level profiling frameworks such as PAPI or HPCToolkit in those environments, without requiring changes to them. As such, it addresses an urgent need in emerging IaaS cloud environments.

REFERENCES

[1] Xen at Wikipedia http://en.wikipedia.org/wiki/Xen
[2] Virtualization at Wikipedia- http://en.wikipedia.org/wiki/Virtualization
[3] Performance Monitor Counters- http://technet.microsoft.com/en-us/library/cc768048.aspx
[4] Hybrid-Virtualization—Enhanced Virtualization for Linux Intel Open Source Technology Center
[5] Performance Profiling in a Virtualized Environment. http://static.usenix.org/event/hotcloud10/tech/full_papers/Du.pdf
[6] Xen in detail Xen http://www.xen.org/
[7] http://www.linuxjournal.com/article/9764
[8] What_is_Xen. http://wiki.xen.org/wiki/Xen_Overview#
[9] http://nehttp://technet.microsoft.com/en-us/library