PAPI-V: Performance Monitoring for Virtual Machines

Matt Johnson, Heike McCraw, Shirley Moore, Phil Mucci, John Nelson, Dan Terpstra, Vince Weaver
Electrical Engineering and Computer Science Dept.
University of Tennessee
Knoxville, TN 37996

Tushar Mohan
Minimal Metrics

Abstract—**This paper describes extensions to the PAPI hardware counter library for virtual environments, called PAPI-V. The extensions support timing routines, I/O measurements, and processor counters. The PAPI-V extensions will allow application and tool developers to use a familiar interface to obtain relevant hardware performance monitoring information in virtual environments.**

Keywords-performance counters; virutal machines; performance analsis; performance monitoring

I. Introduction

Cloud computing involves use of a hosted computational environment that can provide elastic compute and storage services on demand. Virtualization is a technology that allows multiple virtual machines (VMs) to run on a single physical machine and share its resources. Virtualization is increasingly being used in cloud computing to provide economies of scale, customized environments, fault isolation, and reliability. To address performance concerns with the use of cloud computing for scientific computing, the PAPI-V project is developing a system for hardware performance monitoring in virtualized environments to enable software developers to understand and optimize system and application performance and adapt to changing conditions. To accomplish this goal, the project is extending the widely used Performance API (PAPI) cross-platform library [PAPI] for accessing hardware performance counters [1].

A common approach to observing performance on non-virtualized (i.e., native) systems is to use the hardware performance counters available on modern microprocessors. The counters can be accessed using the portable PAPI library interface [1] or by using one of the many end-user performance analysis tools that use PAPI underneath (e.g., TAU [2], PerfSuite [3], Vampir [4], Scalasca [5]). The counters can provide a wide range of information about processor, cache, and memory performance. Recently developed PAPI components for off-processor counters can provide additional performance information about other components of the system, such as the file system, network, temperature and power consumption, and GPUs [6]. In virtualized systems, however, access to hardware performance counters has typically not been available due to lack of support for counter virtualization.

The PAPI-V project is addressing the following aspects:

1. Timing: The PAPI timing routines are being extended to provide standard real and virtual timers across different Virtual Machine Monitors (VMMs).

2. Component measurements: The existing PAPI I/O, network, and other shared resource components are being extended to provide relevant information in virtualized environments.

3. Virtualization of selected processor hardware counters: A selected set of processor counters considered to be most relevant for application performance analysis and tuning in virtualized environments is being implemented across VMMs.

4. Interpretation of data: A mechanism for defining metrics that correctly quantify the contributions of various factors to overall performance is under development.

The research results will be implemented in the publically available and widely used PAPI library. The PAPI-V extension will allow application and tool developers to use a familiar interface to obtain relevant information for achieving the best possible performance in cloud computing environments.

The remainder of this paper is organized as follows: Section II discusses related work on performance monitoring in virtual environments. Section III discusses timing issues and describes the PAPI timing routines. Section IV discusses issues with I/O performance in virtual environments and describes a PAPI component developed to provide I/O performance measurements. Section V describes a PAPI component developed to provide access to pseudo-counters in VMware environments. Section VI describes preliminary work on providing access to processor counters in virtual environments. Section VII concludes with the project status and future work.

II. Related Work

Xenoprof [7] is a profiler specifically designed for Xen [8], a VMM based on paravirtualization. Based on OProfile [9], Xenoprof takes samples, writes them to disk, and connects samples to programs and ultimately source code. Sampling is driven by performance counter hardware: the profiler programs performance counters to count certain events, and interrupt after a certain number of them has been counted. The interrupt service routine takes a sample. Like OProfile, Xenoprof can only be used in system-wide mode.

perfctr-xen is an infrastructure to provide direct access to hardware performance counters in virtualized environments that use the Xen hypervisor [10]. Perfctr-xen relies on the cooperation of guest kernel and underlying hypervisor to provide profiling tools running in the guest with access to performance counters that is compatible with the APIs used in native, unvirtualized environments. Consequently, frameworks and libraries that rely on PAPI can now be used inside Xen. The author modified both the Xen hypervisor as well as the guest kernel running inside each virtual machine. Perfctr-xen supports both paravirtualized mode as well as hardware virtualization mode and exploits optimizations that avoid trap-and-emulate overhead when possible.

Performance counter virtualization for the hardware-assisted KVM virtual machine monitor that is included in recent versions of the Linux kernel is described in [11]. The implementation uses a save-and-restore mechanism for PMU registers. During inter-domain context switches, the hypervisor saves and restores the PMU registers of a domain. The delivery of overflow interrupts to a domain relies on hardware support provided by architectural virtualization extensions. Such a full virtualization approach has the advantage that it does not require any changes to the guest kernel or user libraries, but it will incur more overhead than a paravirtualization approach since each instruction that changes a configuration register requires a separate trap.

III. Timing Routines

A. Timing Issues

The PAPI-V project aims to use the best and most accurate timers exposed by each VMM to implement a uniform timing interface that can be used across VMMs. This timer standardization will allow the same timing code to be used from within an application regardless of which native or virtualized environment it is running on. Some VMMs support the notion of virtualized time, for example called "apparent time" in VMware [12], whereby the virtual machine can have its own idea of time. Some VMMs support a simple version in which virtualized time is stored as a simple offset; whenever a process requests the current time, the offset is added to the current system time and the sum is returned. Other virtualized time mechanisms, such as VMware's, are more complicated, allowing the virtual machine to fall behind and catch up as needed. Even VMMs that support virtualized time have exceptions that allow the guest operating system to access the normal host system's idea of time.

Most processors have a built-in hardware clock that allows the operating system to measure real and process time. Real time, also called elapsed or wall clock time, is the time according to an external standard since some fixed point such as the start of the life of a process. Process time is the amount of the CPU time used by a process since it was created. Process time is broken down into 1) user CPU time, as called virtual time, which is the amount of time spent executing in user mode; and 2) system CPU time, which the amount of time spent executing in kernel mode. Measurement of process time can be useful for evaluating the performance of a program, including on a per-process or per-thread basis.

B. PAPI Timing Routines

PAPI currently provides two basic timing routines: PAPI_get_real_usec for wall clock time and PAPI_get_virt_usec for process virtual time.

On modern Linux, PAPI uses the following POSIX timers:

- PAPI_get_real_usec(): uses clock_gettime(CLOCK_REALTIME);
- PAPI_get_virt_usec() uses clock_gettime(CLOCK_THREAD_CPUTIME_ID)

Figures 1 and 2 below show the results of measuring PAPI real and virtual times for a naïve matrix-matrix multiply on bare metal and on KVM, respectively, where the system is overloaded with multiple processes doing busy work to compete for CPU cycles. We obtained similar results using VMware Workstation. In Figure 1, PAPI real time increases in a stair-step fashion as more processes are added. PAPI virtual time stays constant for the measured process, indicating that time is only measured while the process of interest is scheduled and running. Figure 2 shows these same measures for KVM where multiple virtual machines are competing for physical resources instead of multiple processes on a single machine. In both cases real and virtual time are indistinguishable, since it appears to PAPI that the measured process has full access to the machine.

The real time is calculated on modern systems using the highest resolution operating system timer available, augmented with the timestamp counter if possible. The virtual time is calculated by the scheduler. It records the time when a process is scheduled in, then the time when it is scheduled out. Total virtual time is a sum of all these intervals. If a virtual machine is descheduled when a process is active, the descheduled time appears to the scheduler as virtual time used by the process. That is why virtual time is reported as the same as real time under a VM – the scheduler is essentially is measuring wall clock time as if it had sole access to the hardware.

In order to improve the accuracy of CPU time accounting on virtual systems, the mechanism must be able to not only distinguish between real and virtual CPU time but also recognize being in involuntary wait states. These wait states are referred to as *steal* time. Starting with kernel version 2.6.11, Linux KVM/Xen added support for steal time accounting that allows the scheduler to report the actual time running properly. This is a sort of a paravirtualized virtual time that gets information from the hypervisor on how often a VM was scheduled out. We plan to use this feature to implement compensation for steal time into the PAPI_get_virt_usec routine.

We have added a steal time component to the PAPI 5.0 development version that allows measuring system-wide steal time under KVM. These measurements can be used to adjust timing measurement to compensate for time stolen by other jobs, as shown in Figure 3. Currently, Linux and KVM only provide system-wide steal time values; per-process values will be needed to automatically and completely adjust PAPI virtual time measurements.

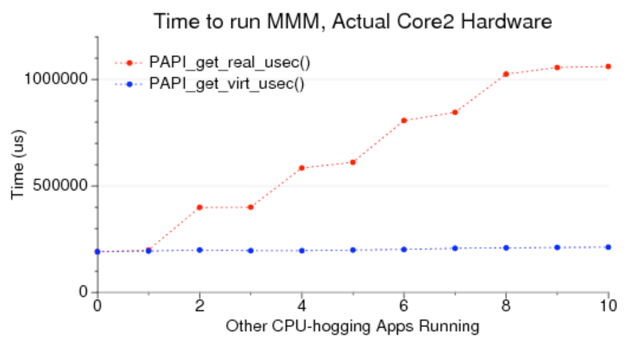


Figure 1. Expected results on bare hardware

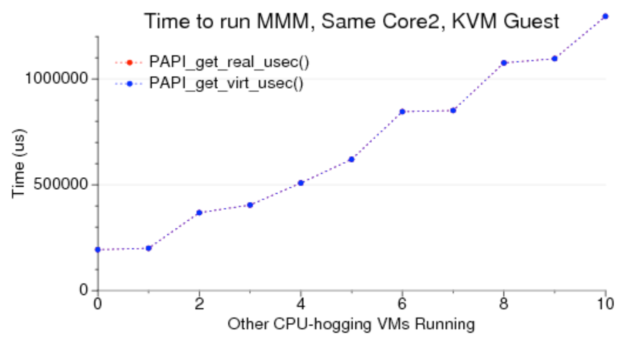


Figure 2. Virtual time on KVM affected by other guests running on system

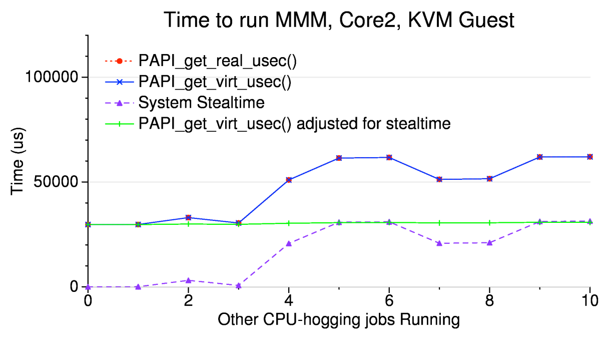


Figure 3. Virtual time adjustment using steal time measurements

IV. I/O PERFORMANCE

Variable I/O performance has been found to significantly impact application performance in virtual environments [13-15]. We have developed an initial version of a PAPI component, called *Application I/O*, or *appio*, for measuring IO performance at application level in virtual environments. In its current forms, *appio* has the following features:

- Intercepts read, write, fread and fwrite
- Supported events: READ_BYTES, READ_CALLS, READ_ERR, READ_SHORT, READ_EOF, READ_BLOCK_SIZE, READ_USEC, WRITE_BYTES, WRITE_CALLS, WRITE_ERR, WRITE_SHORT, WRITE_BLOCK_SIZE, WRITE_USEC
- Works for 32-bit and 64-bit Linux
- Threads-safe (but no aggregation across threads)

We have been testing the *appio* component against IOZone [16], a standard I/O benchmark, comparing I/O performance in virtual environments with performance on bare metal. Results are shown in Figures 4 through 7. The IOzone graphs provide visual verification that the disk read and write subsystems in these two physical and virtual machines are behaving as expected. Data for the bare metal measurements were collected using Ubuntu Linux on a dual Intel Xeon X5550 (Nehalem-EP) with 4 cores per chip clocked at 2.67 GHz and 8 MB of L3 cache. The virtual measurements were collected using a Ubuntu Linux guest hosted by VMware ESX 5.0 on a dual Intel Xeon X7550 (Nehalem-EX) with 8 cores per chip clocked at 2.00 GHz and 18 MB of L3 cache. Because of these differences, the measurements should not be compared numerically, except for the general shapes of the measurement surfaces.

Table 1 compares the read and write rates for a single point on the surfaces shown in Figures 4 through 7. As can been seen, *appio* generally reports higher throughput than IOzone, but the difference between the two measures is small, particularly for the bare metal case. The differences for VMware are somewhat larger, but still less than 10%. From this data we conclude that *appio* can be used effectively to measure and compare application I/O performance in both virtual and physical systems.

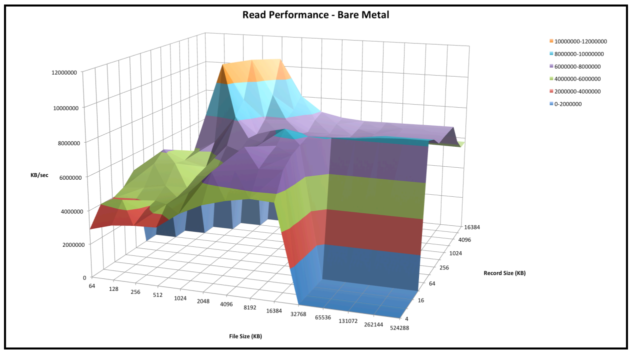


Figure 4. IOzone read performance on bare metal

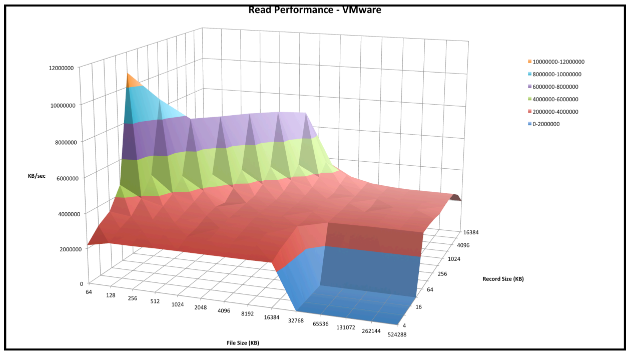
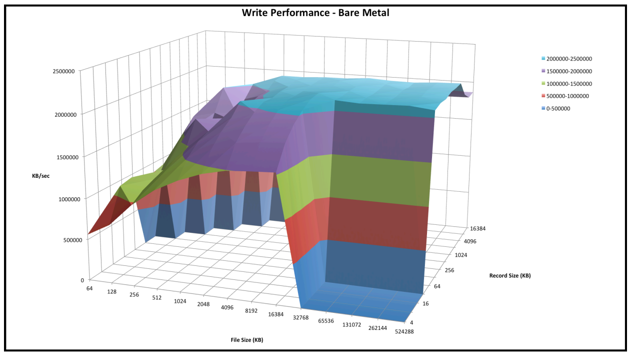


Figure 5. IOzone read performance on VMware ESX



Figre 6. IOzone write performance on bare metal

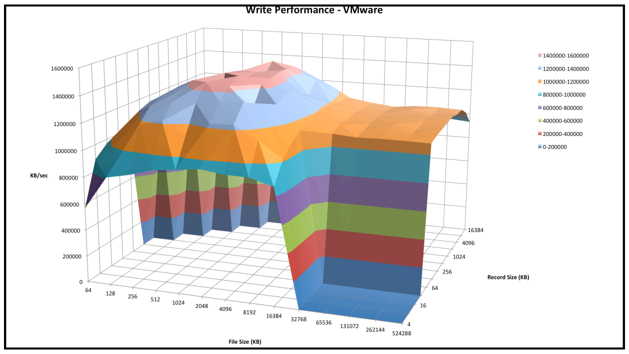


Figure 7. IOzone write performance on VMware ESX

Table 1. Comparison of read and write rates as measured by IOzone and appio

Bare Metal				VMware			
Metric	Type	Rate	Diff	Metric	Type	Rate	Diff
IOzone:	Read	7.1 GB/s	——	IOzone:	Read	3.0 GB/s	——
	Write	2.7 GB/s	——		Write	1.3 GB/s	——
appio:	Read	7.3 GB/s	2.80%	appio:	Read	3.1 GB/s	3.30%
	Write	2.7 GB/s	~ .001 %		Write	1.4 GB/s	7.70%

V. NETWORK PERFORMANCE

One of the promises of Component PAPI is the ability to read performance data from hardware beyond the CPU, such as network interfaces. An example of such a network component is the PAPI Infiniband component. A goal of this project is to extend this functionality to virtual space. This effort has proved to be more time consuming than originally thought, both because of the learning curve involved and because of the need to identify and configure resources for testing. To date we have properly configured two systems with Infiniband and VMware ESX 4 and 5. The ESX 4 system uses the VMware virtualized Infiniband driver and the ESX 5 system uses the DirectPath Infiniband driver. Initial throughput measurements for these two approaches to Infiniband connectivity are shown in Figures 8 and 9.

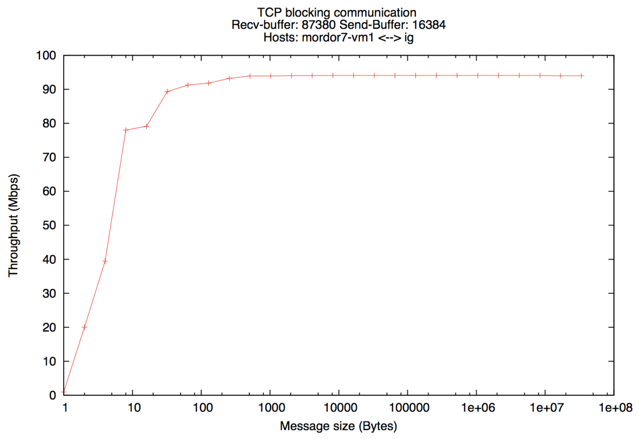


Figure 8. Infiniband throughput in ESX 4.0 with DirectPath disabled

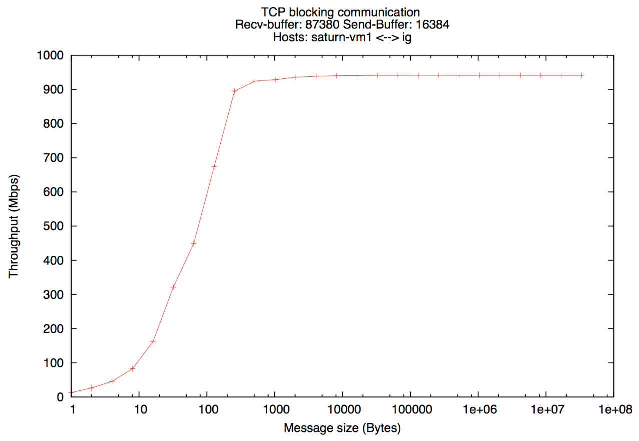


Figure 9. Infiniband throughput in ESX 5.0 with DirectPath enabled

These graphs show a roughly 10-fold difference in throughput with DirectPath enabled. Of course that throughput increase comes at the price of dedicating a network connection to a single virtual machine. In ongoing work we intend to extend the functionality of our existing Infiniband component to both versions of Infinband on VMware and to use these components to measure network performance characteristics on a variety of benchmarks across physical and virtual environments.

VI. VMware Component

The PAPI VMware component is the first PAPI component designed specifically for virtualized environments. This component is available, experimentally, in the current PAPI release. Using VMware's Guest SDK[17], the PAPI team was able to create a component, in a relatively short period of time, that reports both software events and what VMware calls "pseudo performance" counters [12]. VMware makes pseudo performance counters available through an rdpmc instruction to obtain fine-grained time from within the virtual space. However, the monitoring flag on the host machine must be set in order for this to be made available. Therefore, these three timing routines are disabled by default and can be activated dynamically at runtime through an environment variable, PAPI_VMWARE_PSEUDOPERFORMANCE.

The counters made available to PAPI via the VMware Guest SDK are of the type that provides virtual machine statistics such as upper limit of the clock frequency, in MHz, available to the virtual machine, and the minimum clock frequency allocated to the virtual machine. These can give valuable information when evaluating the performance of a virtual machine as it is sharing the CPU with other VMs, as the clock frequency affects peak performance calculations.

VII. Virtualized Processor Counters

The performance monitoring unit (PMU) of a processor typically includes:

- A set of performance counter registers that count the frequency or duration of specific processor events
- A set of performance event select registers used to specify the events that are tracked by the performance counter registers
- A hardware interrupt that can be generated when a counter overflows
- A time stamp counter (TSC) that can be used to count processor clock cycles

The registers used in support of performance monitoring are model-specific registers (MSRs). Each performance counter can be configured to count one event, or measure the duration of one event, at a time. For event counting, the processor increments the counter whenever the event occurs (e.g., a cache miss). For duration measurement, the processor counts the number of processor clock cycles required to complete an event (e.g., the latency of a cache miss). The TSC may be affected by power management events, such as processor frequency changes.

Processor counters are widely used in the scientific computing community for application performance analysis, modeling, and tuning. Events of interest include instruction counts and measurements of cache, memory, and TLB behavior.

We currently have KVM in-guest performance counters working from PAPI. We used a 3.3 kernel on the host, a 3.2 kernel in the guest, and the current git-snapshot of qemu. We exported the "native" CPU inside the guest. The current development version of PAPI compiles and runs fine. PAPI can correctly detect that it is running inside KVM. All of the PAPI acceptance tests pass except for profiling and overflow, as overflow isn't implemented. Most numbers agree between bare metal and KVM, but we are investigating why virtual usec/cycles are higher on KVM.

We installed VMware Workstation Technology Preview 2012, and after resolving some license key issues we successfully ran it on our SandyBridge-EP machine. We were able to successfully compile PAPI from within VMware. The PAPI utilities show that we can properly detect that we are running inside of VMware and also the setup of the virtual hardware (number of virtual CPUs, number of virtual cores per CPUs). Compared to KVM, we don't see the same discrepancy between virtual cycles and seconds on the physical platform versus the virtual platform as we encountered with KVM. The ratio of virtual cycles to virtual seconds on physical hardware is fairly consistent to what we get from within VMware.

VIII. Project Status and Future Work

We are in the first year of a three-year project to implement PAPI for virtual environments. The initial implementations of the I/O and VMware components can be obtained from the PAPI website at http://icl.eecs.utk.edu/papi/. Timing routines and access to process counters will be provided as soon as we can ensure reliable and validated results.

We have completed a series of redesigns and modifications of the PAPI library to refactor it to support virtual environments. One such change is simply exposing the existence of a virtual machine environment to the user space application. Another is refactoring operating system dependencies away from CPU dependencies to make the architecture more robust. These changes will be released as PAPI 5.0 (PAPI-V) sometime in the summer of 2012, once final feature implementation and testing are complete. We anticipate that further enhancements related to virtual environments will be incorporated into the library as we and the user community gain experience with this release.

We welcome comments and suggestions from the community on desired functionality for hardware performance monitoring in virtual environments, as well as feedback on our initial implementations.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1117058 and by VMware.

References

[1] Browne, S., Dongarra, J., Garner, N., London, K., Mucci, P. "A Scalable Cross-Platform Infrastructure for Application Performance Tuning

Using Hardware Counters," Proceedings of SuperComputing 2000 (SC'00), Dallas, TX, November 2000.

[2] Shende, S. and A.D. Malony, *The TAU Parallel Performance System.* International Journal of High Performance Computing Applications, 2006. **20**(2): p. 287-311.

[3] 12. Kufrin, R., PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux, in 6th International Conference on Linux Clusters: The HPC Revolution (LCI 2005), April 2005: Chapel Hill, NC.

[4] 13. Brunst, H., et al., *Tools for scalable parallel program analysis: Vampir NG, MARMOT, and DeWiz.* International Journal of Computational Science and Engineering, 2009. **4**(3): p. 149-161.

[5] 14. Wolf, F., et al., *Automatic analysis of inefficiency patterns in parallel applications.* Concurrency and Computation: Practice and Experience, 2007. **19**(11): p. 1481-1496.

[6] Terpstra, D., et al., Collecting Performance Data with PAPI-C, in Tools for High Performance Computing, 2009: Dresden, Germany. p. 157-173.

[7] Menon, A., et al., *Diagnosing Performance Overheads in the Xen Virtual Machine Environment*, in *First ACM/USENIX Conference on Virtual Execution Environments (VEE'05),* June 2005.

[8] Barham, P., et al., *Xen and the Art of Virtualization*, in *19th ACM Symposium on Operating Systems Principles (SOSP'03),* 2003, ACM: Bolton Landing, NY. p. 164-177.

[9] *OProfile website*. Available from: http://oprofile.souceforge.net/.

[10] Nikolaev, R. and G. Back, *Perfctr-Xen: A Framework for Performance Counter Virtualization*, in *2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011),* March 2011: Newport Beach, CA.

[11] Du, J., N. Sehrawat, and W. Zwaenepoel, Performance Profiling of Virtual Machines, in 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011), March 2011, ACM: Newport Beach, CA.

[12] *Timekeeping in VMware Virtual Machines*, 2010, VMware, Inc.: Palo Alto, CA.

[13] Nanos, A., G. Goumas, and N. Koziris, *Exploring I/O Virtualization Data Paths for MPI Applications in a Cluster of VMs: A Networking Perspective*, in *5th Workshop on Virtualization in High Performance Cloud Computing (VHPC'10),* 2010: Naples, Italy.

[14] Kim, H., et al., *Task-aware Virtual Machine Scheduling for I/O Performance*, in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'09),* 2009: Washington, DC.

[15] Ongaro, D., A.L. Cox, and S. Rixner, *Scheduling I/O in Virtual Machine Monitors*, in *International Conference on Virtual Environments,* 2008: Seattle, WA.

[16] IOzone Filesystem Benchmark, http://www.iozone.org/

[17] VMware, vSphere Guest SDK Documentation, http://www.vmware.com/support/developer/guest-sdk/