Evaluation of the HPC Challenge Benchmarks in Virtualized Environments*

Piotr Luszczek, Eric Meek, Shirley Moore, Dan Terpstra, Vincent M. Weaver, Jack Dongarra

Innovative Computing Laboratory
University of Tennessee Knoxville
{luszczek,shirley,terpstra,vweaver1,dongarra}@eecs.utk.edu

Abstract. This paper evaluates the performance of the HPC Challenge benchmarks in several virtual environments, including VMware, KVM and VirtualBox. The HPC Challenge benchmarks consist of a suite of tests that examine the performance of HPC architectures using kernels with memory access patterns more challenging than those of the High Performance Linpack (HPL) benchmark used in the TOP500 list. The tests include four local (matrix-matrix multiply, STREAM, RandomAccess and FFT) and four global (High Performance Linpack – HPL, parallel matrix transpose – PTRANS, RandomAccess and FFT) kernel benchmarks.
The purpose of our experiments is to evaluate the overheads of the different virtual environments and investigate how different aspects of the system are affected by virtualization. We ran the benchmarks on an 8-core system with Core i7 processors using Open MPI. We did runs on the bare hardware and in each of the virtual environments for a range of problem sizes. As expected, the HPL results had some overhead in all the virtual environments, with the overhead becoming less significant with larger problem sizes. The RandomAccess results show drastically different behavior and we attempt to explain it with pertinent experiments. We show the cause of variability of performance results as well as major causes of measurement error.

1 Introduction

With the advent of cloud computing, more and more workloads are being moved to virtual environments. High Performance Computing (HPC) workloads have been slow to migrate, as it has been unclear what kinds of tradeoffs will occur when running these workloads in such a setup. We evaluated the overheads of

* This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia, Purdue I., and T-U. Dresden.

several different virtual environments and investigated how different aspects of the system are affected by virtualization.

The virtualized environments we investigated were VMware, KVM and VirtualBox. We used the HPC Challenge (HPCC) benchmarks [6] to evaluate these environments. HPC Challenge examines performance of HPC architectures using kernels with memory access patterns more challenging than those of the High Performance Linpack (HPL) benchmark used in the TOP500 list. The tests include four local (matrix-matrix multiply, STREAM, RandomAccess and FFT) and four global (High Performance Linpack – HPL, parallel matrix transpose – PTRANS, RandomAccess and FFT) kernel benchmarks.

We ran the benchmarks on an 8-core system with Core i7 processors using Open MPI. We ran on bare hardware and inside each of the virtual environments for a range of problem sizes.

As expected, the HPL results had some overhead in all the virtual environments, with the overhead becoming more significant with larger problem sizes and VMware having the least overhead. The latency results showed higher latency in the virtual environments, with KVM being the highest.

We do not intend for this paper to provide a definitive answer as to which virtualization technology achieves the highest performance results. Rather, we seek to provide guidance on more generic behavioral features of various virtualization packages and to further the understanding of VM technology paradigms and their implications for performance-conscious users.

2 Related Work

There have been previous works that looked at measuring the overhead of HPC workloads in a virtualized environment. Often the works measure timing external to the guest, or, when they use the guest, they do not explain in great detail what problems they encountered when trying to extrapolate meaningful performance measurements: the very gap we attempt to breach with this paper.

Youseff et al. [11] measured HPC Challenge and ASCI Purple benchmarks. They found that Xen has better memory performance than real hardware, and not much overhead.

Walters et al. [10] compared the overheads of VMWare Server (not ESX), Xen and OpenVZ with Fedora Core 5, Kernel 2.6.16. They used NetPerf and Iozone to measure I/O and the NAS Parallel benchmarks (both serial, OpenMP and MPI) for HPC. They found Xen best in networking, OpenVZ best for filesystems. On serial NAS, most are close to native, some even ran faster. For OpenMP runs, Xen and OpenVZ are close to real hardware, but VMware has large overhead.

Danciu et al. [1] measured both high-performance and high-throughput workloads on Xen, OpenVZ, and Hyper-V. They used LINPACK and Iometer. For timing, they used UDP packets sent out of the guest to avoid timer scaling issues. They found that Xen ran faster than native on many workloads, and that I/O did not scale well when running multiple VMs on the same CPU.

Han et al. [2] ran Parsec and MPI versions of the NAS Parallel benchmarks on Xen and kernel 2.6.31. They found that the overhead becomes higher when more cores are added.

Huang et al. [4] ran the MPI NAS benchmarks and HPL inside of Xen. They measured performance using the Xenoprof infrastructure and found most of the overhead to be I/O related.

Li et al. [5] ran SPECjvm2008 on a variety of commercial cloud providers. Their metrics include cost as well as performance.

Mei et al. [7] measured performance of webservers using Xenmon and Xentop.

3 Setup

3.1 Self-Monitoring Results

When conducting large HPC experiments on a virtualized cluster, it would be ideal if performance results could be gathered from inside the guest. Most HPC workloads are designed to be measured that way, and doing so requires no change to existing code.

Unfortunately measuring time from within the guest has its own difficulties. These are spelled out in detail by VMware [9]. Time that occurs inside a guest may not correspond at all to outside wallclock time. The virtualization software will try its best to keep things relatively well synchronized, but, especially if multiple guests are running, there are no guarantees.

On modern Linux, either gettimeofday() or clock_gettime() are used by most applications to gather timing information. PAPI, for example, uses clock_gettime() for its timing measurements. The C library translates these calls into kernel calls and executes them either by system call, or by the faster VDSO mechanism that has lower overhead. Linux has a timer layer that supports these calls. There are various underlying timers that can be used to generate the timing information, and an appropriate one is picked at boot time. The virtualized host emulates the underlying hardware and that is the value passed back to the guest. Whether the returned value is "real" time or some sort of massaged virtual time is up to the host.

A list of timers that are available can be found by looking at the file `/proc/timer_list`.

There are other methods of obtaining timing information. The `rdtsc` call reads a 64-bit time-stamp counter on all modern x86 chips. Usually this can be read from user space. VMs like VMware can be configured to pass through the actual system TSC value, allowing access to actual time. Some processor implementations stop or change the frequency of the TSC during power management situations, which can limit the usefulness of this resource.

The RTC (real time clock) can also generate time values and can be accessed directly from user space. However, this timer is typically virtualized.

Others have attempted to obtain real wall clock time measurements by sending messages outside the guest and measuring time there. For example, Danciu et al. [1] send a UDP packet to a remote guest at the start and stop of timing, which allows outside wallclock measurements. We prefer not to do this, as it requires network connectivity to the outside world that might not be available on all HPC virtualized setups.

For our measurements we use the values returned by the HPC Challenge programs, which just call the gettimeofday() interface invoked by MPI_Wtime().

3.2 Statistical Methods

As the VM environments are one step removed from the hardware and, consequently, introduce additional sources of measurement errors, we make an effort to counteract this uncertainty with a number of statistatical techniques. Each of the results we report is a statistical combination of up to five measurements, each of which was taken under the same (or nearly the same) circumstances. One exception is the accuracy-drift experiments from Section 4.2 that were explicitly meant to show variability of performance measurement caused by inconsistent state of the tested VMs. When applicable, we also indicate the standard deviation on our graph charts to indicate the variability and a visual feedback on trustworthiness of the particular measurement.

To combine multiple data points, we use the minimum function for time and the maximum for performance. In our view, these two best filter out randomly injected overheads that could potentially mask the inherent behavior of the virtual environments we tested.

3.3 Hardware and Software Used in Tests

For our tests we used an Intel Core i7 platform. The featured processor was a four-core Intel Xeon X5570 clocked at 2.93 GHz. The VMware Player was version 3.1.4, VirtualBox – 4.0.8, and KVM was compatible with kernel version 2.6.35.

4 Evaluation

4.1 Disparity between CPU and Wall Clock Timers

Among other things, HPCC attempts to underscore the difference between CPU time and wall clock time in performance measurements. The former may simply be measured with a C library call clock() and the latter with either BSD's gettimeofday() or clock_gettime() from real-time extension of POSIX. A common complaint is the low resolution of the primitives for measuring the CPU time. Under greater scrutiny, the complaint stems from the workings of CPU time accounting inside the kernel – the accuracy is closely related to the length of the scheduler tick for computationally intensive processes. As a result, CPU and wall clock time tend to be in large disagreement for such workloads across shorter

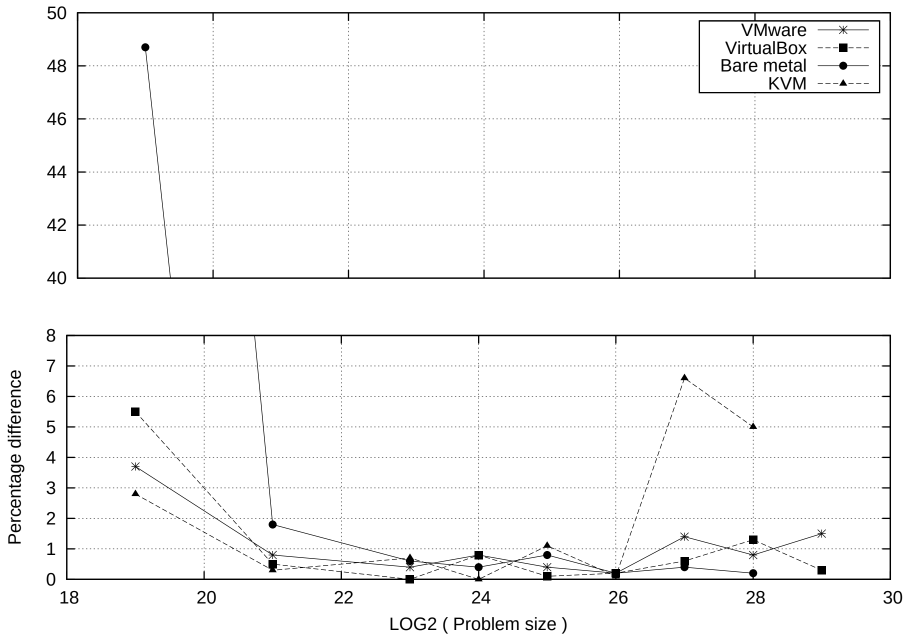

Fig. 1. Variation in percentage difference between the measured CPU and wall clock times for MPIRandomAccess test of HPC Challenge. The vertical axis has been split to offer a better resolution for the majority of data points.

timing periods. This observation is readily confirmed with the results from Figure 1. The figure plots the relative difference between readings from both timers across a large set of input problem sizes:

$$\left|\frac{T_{\text{CPU}}}{T_{\text{wall}}} - 1\right| \times 100\%. \tag{1}$$

We may readily observe nearly a 50% difference between the reported CPU and wall clock times for small problem sizes on bare metal. The discrepancy diminishes for larger problem sizes that require longer processing times and render low timer resultion much less of an issue. In fact, the difference drops below a single percentage point for most problems larger than 2^{22}. Virtual environments do not enjoy such consistent behavior though. For small problem sizes, both timers diverge only by about 5%. Our understanding attributes this behavior to a much greater overhead imposed by virtual environments on system calls such as the timing primitives that require hardware access. More information on the sources of this overhead may be found in Section 2. In summary, for a wide range of problem sizes we observed nearly an order of magnitude difference between the observed behavior of CPU and wall clock time.

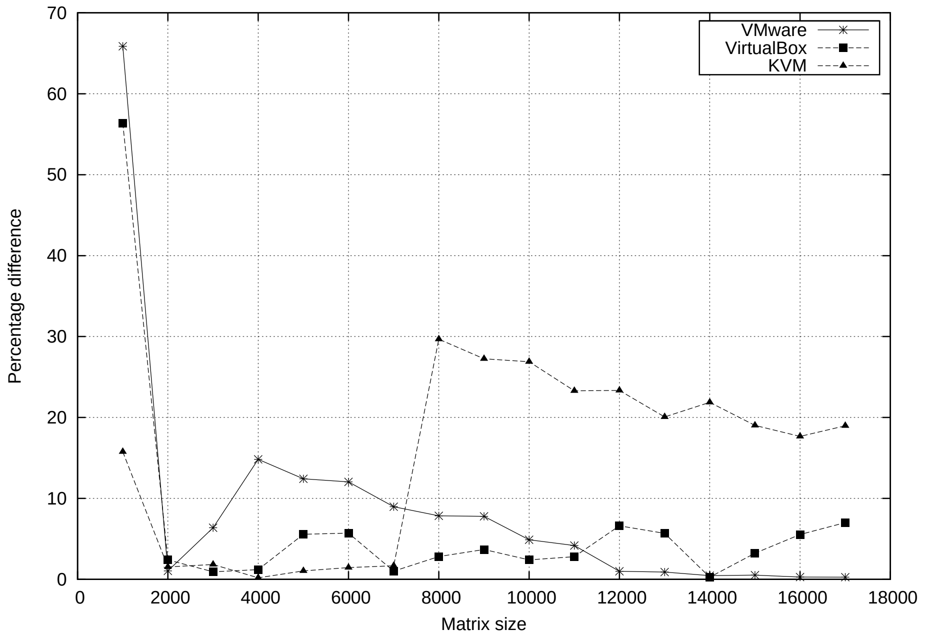

Fig. 2. Variation in percentage difference between the measured wall clock times for HPL (a computationally intensive problem) for ascending and descending orders of problem sizes during execution.

Another common problem is a *timer inversion* whereby the system reports that the process' CPU time exceeds the wall clock time:

$$T_{\text{CPU}} > T_{\text{wall}}. \qquad (2)$$

On bare metal, the timer inversion occurs due to a large difference in relative accuracy of both timers and is most likely to occur when measuring short periods of time that usually result in large sampling error [3]. This is not the case inside all of the virtual machines we tested. The observed likelihood of timer inversion for virtual machines is many-fold greater than the bare metal behavior. In addition, the inversions occur even for some of the largest problem sizes we used: a testament to a much diminished accuracy of the wall clock timer that we attribute to the virtualization overhead.

4.2 Accuracy-Drift of Timers with VM State Accumulation

Another important feature of virtual environments that we investigated was the *accuracy drift* of measurements with respect to the amount of accumulated state within the VMM. This directly relates to a common workflow within benchmarking and performance tuning communities whereby a given portion of the code is run repeatedly until a satisfactory speedup is reached. Our findings indicate that this may lead to inconsistent results due to, in our understanding, a change of state within software underlying the experiment. To illustrate this phenomenon, we ran HPL, part of the HPC Challenge suite, twice with the same set of input problem sizes. In the first instance, we made the runs in ascending order: starting with the smallest problem size and ending with the largest. Then after a reboot, we used the descending order: the largest problem size came first. On bare metal, the resulting performance shows no noticable impact. This is not the case inside the VMs as shown in Figure 2. We plot the percentage difference of times measured for the same input problem size for the ascending and descending orders of exection:

$$\left|\frac{T_{\text{descending}}}{T_{\text{ascending}}} - 1\right| \times 100\%. \tag{3}$$

The most dramatic difference was observed for the smallest problem size of 1000. In fact, it was well over 50% for both VirtualBox and VMware. We attribute this to the short running time of this experiment and, consequently, a large influence of the aforementioned anomaly. Even if we were to dismiss this particular data point, the effects of the accuracy drift are visible across the remaining problem sizes but the pattern of influence is appreciably different. For VMware, the effect gradually abates for large problem sizes after attaining a local maximum of about 15% at 4000. On the contrary, KVM shows comparatively little change for small problem sizes and then drastically increases half way through to stay over 20% for nearly the rest of the large problem sizes. And finally, the behavior of VirtualBox initially resembles that of VMware and later the accuracy drift diminishes to fluctuate under 10%.

From a performance measurement standpoint this resembles the problem faced when benchmarking file systems. The factors influencing the results include the state of the memory file cache and the level of file and directory fragmentation [8]. In virtual environments, we also observe this persistence of state that in the end influences the performance of VMs and the results observed inside its guest operating systems. Clean boot results proved to be the most consistent in our experiments. However, we recognize that, for most users, rebooting the VM after each experiment might not accurately reflect the feasible deployment circumstances.

5 Results

In previous sections we have outlined the potential perils of accurate performance evaluation of virtual environments. With these in mind, we attempt to show in this section the performance results we obtained by running the HPC Challenge suite across the tested VMs. We consider the ultimate goal for a VM to match the performance of the bare metal run. In our performance plots then we use a relative metric – the percentage fraction of bare metal performance that is achieved inside a given VM:

$$\frac{\text{performance}_{\text{VM}}}{\text{performance}_{\text{bare metal}}} \times 100\% \tag{4}$$

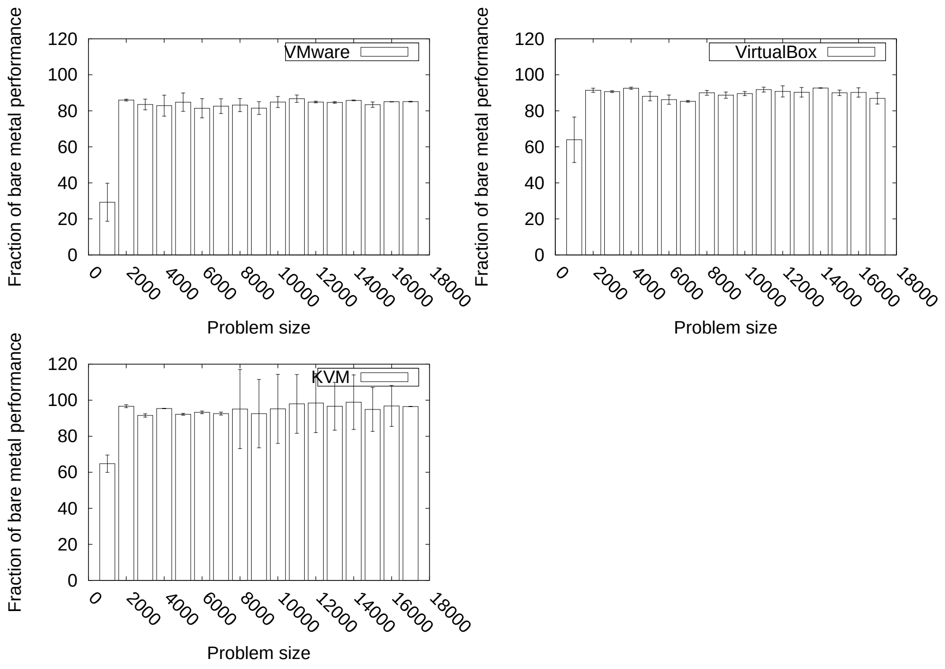

Fig. 3. Percentage of baremetal performance achieved inside VMware, VirtualBox, and KVM for HPC Challenge's HPL test. Each data bar shows the standard deviation bar to indicate the variability of the measurement.

Due to space constraints we cannot present a detailed view of all of our results. Instead, we focus on two tests from the HPC Challenge suite: HPL and MPI-RandomAccess. By selecting these two tests we intend to contrast the behavior of two drastically different workloads. The former represents codes that spend most of their running time inside highly optimized library kernels that nearly optimally utilize the cache hierarchy and exhibit very low OS-level activity which could include servicing TLB misses and network card interrupts required for inter-process communication. Such workloads are expected to suffer little from the introduction of a virtualization layer and our results confirm this as shown in Figure 3. In fact, we observed that virtualization adds very little overhead for such codes and the variability of the results caused by the common overheads is relatively small across a wide range of input problem sizes. On the contrary, MPI-RandomAccess represents workloads that exhibit high demand on the memory subsystem including TLBs and require handling of very large counts of short messages exchanged between processors. Each of these characteristics stresses the bare metal setup and is expected to do so inside a virtualized environment. Our results from Figure 4 fully confirm this prediction. The virtualization overhead is very high and could reach 70% performance loss. Furthermore, a large standard deviation of the measurements indicates to us that long running codes of this type can accumulate state information inside the VM that adversely af-

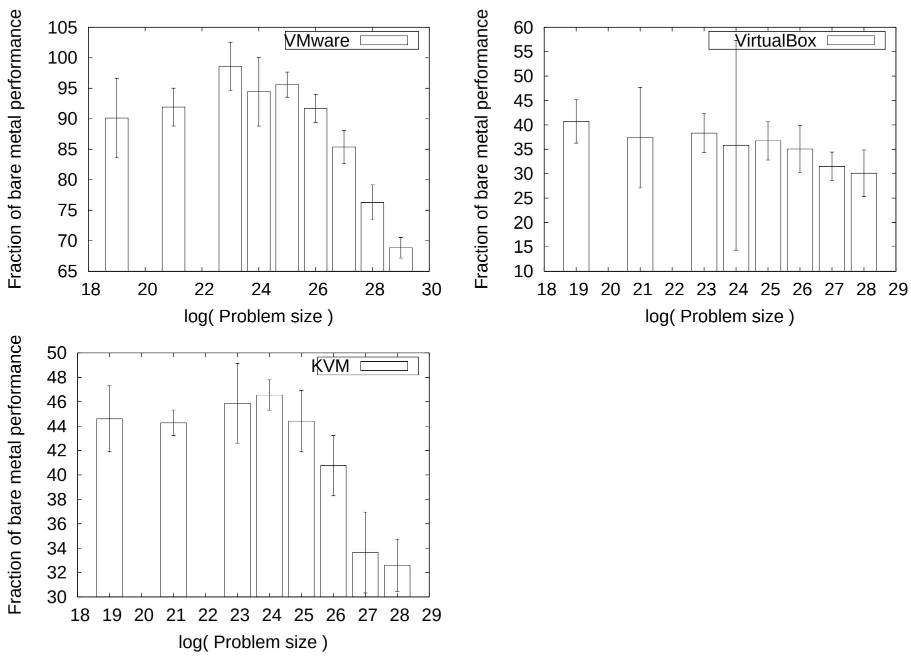

Fig. 4. Percentage of baremetal performance achieved inside VMware, VirtualBox, and KVM for HPC Challenge's MPIRandomAccess test. Each data bar shows the standard deviation bar to indicate the variability of the measurement.

fects the accuracy of the measurements. Such accumulated accuracy drift may persist across a multitude of input problem sizes.

6 Conclusions and Future Work

In this paper we have shown how virtualization exacerbates the common problems of accurate performance measurement and benchmarking. Furthermore, we have observed new obstacles to reliable measurements which we related to the accumulation of state information that is internal to all the tested VMs. This leads us to present our results with clear indicators of their statistical quality and a detailed description of settings and circumstances of the runs to render them repeatable. This, we believe, should be strongly stressed for measuring performance with virtualization enabled – even more so than is customarily practiced for the bare metal runs.

We also showed two drastically different workloads in terms of how they stress the virtualization layer. Our future work will focus on more detailed examination of these workloads and possibly devising new ones that will help us better understand the resulting overheads and performance variability.

References

1. Danciu, V.A., g. Felde, N., Kranzlmüller, D., Lindinger, T.: High-performance aspects in virtualized infrastructures. In: 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management. pp. 25–32 (Oct 2010)
2. Han, J., Ahn, J., Kim, C., Kwon, Y., ri Choi, Y., Huh, J.: The effect of multi-core on HPC applications in virtualized systems. In: 5th Workshop on Virtualization and High-Performance Cloud Computing (Aug 2010)
3. Hines, S., Wyatt, B., Chang, J.M.: Increasing timing resolution for processes and threads in Linux (2000), unpublished.
4. Huang, W., Liu, J., Abali, B., Panda, D.: A case for high performance computing with virtual machines. In: Proceedings of the 20th annual international conference on Supercomputing (2006)
5. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: comparing public cloud providers. In: 10th annual conference on Internet measurement (2010)
6. Luszczek, P., Bailey, D., Dongarra, J., Kepner, J., Lucas, R., Rabenseifner, R., Takahashi, D.: The HPC challenge HPCC benchmark suite. In: SuperComputing 2006 Conference Tutorial (2006)
7. Mei, Y., Liu, L., Pu, X., Sivathanu, S.: Performance measurements and analysis of network I/O applications in virtualized cloud. In: IEEE 3rd International Conference on Cloud Computing. pp. 59–66 (Aug 2010)
8. Smith, K.A., Selzter, M.: File layout and file system performance. Computer Science Technical Report TR-35-94, Harvard University (1994)
9. Timekeeping in VMware Virtual Machines: VMware ESX 4.0/ESXi 4.0, VMware workstation 7.0 information guide
10. Walters, J., Chaudhary, V., Cha, M., Guercio, S.J., Gallo, S.: A comparison of virtualization technologies for HPC. In: 22nd International Conference on Advanced Information Networking and Applications. pp. 861–868 (Mar 2008)
11. Youseff, L., Wolski, R., Gorda, B., Krintz, R.: Paravirtualization for HPC systems. In: Workshop on Xen in High-Performance Cluster and Grid Computing (2006)