Evaluating the Performance and Intrusiveness of Virtual Machines for Desktop Grid Computing

Patricio Domingues
School of Technology and Management - Polytechnic Institute of Leiria, Portugal
patricio@estg.ipleiria.pt

Filipe Araujo, Luis Silva
CISUC, Dept. of Informatics Engineering, University of Coimbra, Portugal
{filipius, luis}@dei.uc.pt

Abstract

We experimentally evaluate the performance overhead of the virtual environments VMware Player, QEMU, VirtualPC and VirtualBox on a dual-core machine. Firstly, we assess the performance of a Linux guest OS running on a virtual machine by separately benchmarking the CPU, file I/O and the network bandwidth. These values are compared to the performance achieved when applications are run on a Linux OS directly over the physical machine. Secondly, we measure the impact that a virtual machine running a volunteer @home project worker causes on a host OS. Results show that performance attainable on virtual machines depends simultaneously on the virtual machine software and on the application type, with CPU-bound applications much less impacted than IO-bound ones. Additionally, the performance impact on the host OS caused by a virtual machine using all the virtual CPU, ranges from 10% to 35%, depending on the virtual environment.

1. Introduction

System-level virtual machines that can run unmodified operating systems (OS) are substantially changing computing. In fact, virtual machines have emerged in the last decade as a viable solution for running a full operating system (*guest OS*) on top of a hosting environment (*host OS*). Among other features, virtual machines provide for the easy deployment of virtual OSes, migration, fault tolerance and sandboxing. A major use for virtualization has been software development, software testing and server consolidation [15]. Additionally, virtualization can also play a major role in public resource computing projects, like SETI@home, Einstein@home, Rosetta@home and others [2]. Indeed, several characteristics make virtualization appealing for public resource computing, both from the developers and volunteers point of view. For instance, virtualization provides for an easy deployment of the same computing environment across all participating machines. This includes the operating system and all the software stack that might be required by the desktop grid application. In addition, having a unique and well-known environment across all volunteers considerably eases the task of developers, because they only have to deal with a single platform.

Besides providing for a homogeneous environment, the use of virtual machines for desktop grid computing brings an enhanced security for volunteers. Indeed, the sandboxing isolation offered by system-level virtual machines makes the execution of a foreign application by a volunteer machine much safer. In fact, virtual machines are often used in security-oriented environments for testing potentially malicious software.

Another appealing feature of virtual machines for desktop grid computing lies in the possibility of saving the state of the guest OS to persistent storage. This is done in a transparent manner, requiring no intervention nor modification of the guest OS. This checkpointing feature allows simultaneously for fault tolerance and migration, making possible the exportation of a virtual environment to another physical machine, with the execution being resumed at the remote machine.

Nonetheless the above stated advatanges, wide deployment of virtualization for desktop grids has some hindrances: (1) software licensing, (2) the size of the virtual OS images and (3) the performance impact on both the virtualized environment and on the host OS. Software licensing issues, namely operating system ones can be dealt by resorting to open source OS, such as Linux and BSD. To contain the size of the virtual machine image, one can choose a small footprint distribution, such as *ttylinux* (www.minimalinux.org). However, this will always impose

a download that might not be affordable for all the would-be volunteers.

Finally, resorting to virtual machines for desktop grid computing has implication on performance. Indeed, desktop grid computing aims to use volunteer resources in an efficient manner to maximize the work done. Therefore, before considering the wide-scale adoption of virtual machines for desktop grid computing, the performance impact needs to be assessed. This is precisely the main goal of our study. We split the performance evaluation into two main experiments: one to determine the overhead of running applications in guest OSes, and a second one to assess the overhead that such setting causes on the host OS. For this purpose, we firstly evaluate the raw performance of applications run on virtual machines resorting to benchmarks for CPU, disk I/O and network I/O. We compare the measured values against the performance obtained on the same physical machine, but in a native environment. Secondly, we assess the impact on the performance of a machine that hosts a virtual machine. Specifically, we measure the drop in performance sensed at the host OS level when a guest OS running on a virtual machine is computing a task from the Einstein@home public volunteer project.

We do all performance measurements with four freely available system-level virtual environments: *VMware player* (henceforth VmPlayer), *QEMU*, *Virtual PC* (henceforth virtualPC) and *VirtualBox*. We use Windows XP as the host OS and Linux as the guest OS. The measured values are compared with a native environment, in which the physical machine is running the same Linux distribution.

An interesting result of this study is the direct relation that seems to exist between the performance delivered by a given virtual machine environment and the impact it causes on the host OS: the higher the performance, the higher is the overhead. In particular, the VmPlayer environment, which delivered the best performance among the studied virtual machines, also caused the highest impact on host.

The remainder of this paper is organized as follows. In Section 2, we present the applications used to assess the performances of virtual machines, while in Section 3 we briefly describe the four virtual machines that were evaluated in this study. Section 4 presents the results. Section 5 summarizes the related work. Finally, Section 6 concludes the paper.

2 Methodology

To assess the performance delivered by the assessed virtual machines, we resorted to the benchmarks listed below. Of these, we wrote *Matrix* and *IOBench*, while the others already existed and are freely available:

- *7Z*: the 7Z application implements, as its default mode, the LZMA algorithm (Lempel-Ziv-Markov chain-Algorithm), which is an improved LZ77 compressor developed by Igor Pavlov [16]. The application has a benchmark mode, which is available through the command line option *b*. Specifically, the benchmark mode returns an execution rate based on the number of instructions that were executed as well as the percentage of CPU that was available to the application. Another useful feature of 7z is the possibility of setting, through the command line switch *-mmt*, the number of threads of the application. This allows for benchmarking multi-core machines. As the 7z benchmark does not access I/O devices, it essentially measures non-floating point CPU performance.

- *Matrix*: this application multiplies two squared matrices of doubles, using a linear (non-optimized) algorithm. We used two matrix sizes: 512×512, and 1024×1024. This benchmark essentially evaluates floating-point CPU performance.

- *IOBench*: to assess IO disk performance, we developed the python program IOBench. It evaluates the performance of the read and write I/O (disk) operations of the filesystem. For this purpose, IOBench executes read and write operations for randomly generated files, whose size ranges from 128 KB to 32 MB. Between each test, the file size is incremented by doubling the precedent one, therefore yielding the sequence 128 KB, 256 KB, 512 KB and so on.

- *NetBench*: NetBench is a wrapper for the *iperf* application [12]. This application measures the network speed for point-to-point communications at the transport layer (TCP and UDP). In this study, the *iperf* was used in its default mode, in which it measures the time required for the transfer of a 10 MB data stream over a TCP connection between a guest OS and a remote machine acting as an *iperf* server. The connecting network was a 100 Mbps Fast Ethernet LAN.

3 Environments

Since Windows is the dominating OS in desktop grid machines [3], we considered Windows XP as the hosting environment throughout this paper. For the guest OS, we selected the Linux Ubuntu distribution. This corresponds to a plausible usage scenario, since due to licensing restrictions, the usage of virtual machines for desktop grids is only viable with license free OS and software.

The applications were run on the following system-level virtual machines: VmPlayer 2.0.2 [21], VirtualBox 1.6.2 [19], QEMU 0.9 with QEMU accelerator 1.3 [5] and Microsoft VirtualPC 2007 6.0.156 [20]. We selected these virtual platforms because (1) they are freely available, (2)

they run on top of Windows XP and (3) they can execute an unmodified Linux image. Moreover, all the considered virtualized environments implement full virtualization, completely simulating an underlying hardware. Next, we succinctly describe the four evaluated virtual machine environments.

3.1 VmWare player

VMware, Inc. was pioneer in the development of virtualization products for the x86 architecture. VmPlayer is a standalone virtualization software that can run guest OSes. The software is optimized, relying on binary translation for achieving nearly native speed.

3.2 QEMU

QEMU is an open source (released under a GPL license) virtualization environment that emulates many hardware platforms. QEMU is being developed by Fabrice Bellard, and it is highly praised for its features and for the large number of architectures (x86, AMD64, SPARC, etc.) that it can emulate. In this study, we complemented QEMU with the QEMU Accelerator which speeds up the execution via binary translation on x86 platforms [5].

3.3 VirtualBox

VirtualBox was released in January 2007 under the GPL version 2 open source license. The product was later acquired by Sun Microsystems in February 2008. Like VmPlayer, VirtualBox resorts to binary translation to achieve full virtualization, while other parts were adopted from QEMU. In this study, we used the open source variant of VirtualBox, version 1.6.2.

3.4 VirtualPC

Microsoft's VirtualPC implements full virtualization for the x86 architecture. Although officially VirtualPC does not support Linux as guest OS, we were able to run the Ubuntu distribution without hurdles. However, the lack of official support means that the optimization packages available for Windows guest OSes do not exist for Linux.

4 Results

All tests were conducted on a Core 2 Duo 6600 @2.40 GHz fitted with 1 GB of DDR2 RAM memory. The host OS is Windows XP (service pack 2). The virtual machines were set with 300 MB of virtual RAM. To avoid bias, every test was performed at least 50 times. Additionally, to circumvent the timing imprecision that occur on virtual machines,

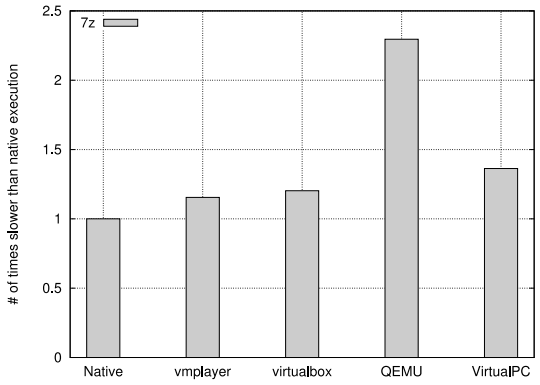

Figure 1. Relative performance of 7z on virtual machines

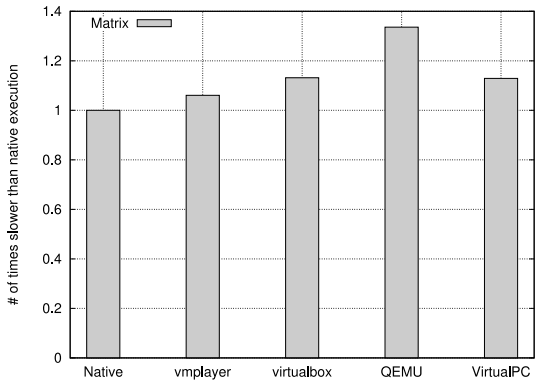

Figure 2. Relative performance of Matrix on virtual machines

especially when the machines are under high load [22], time measurements for executions under virtual machines were done resorting to an external time reference. For that purpose, we used a simple UDP time server running on the host machine.

We present two sets of results: (1) *performance of guest OSes* and (2) the *impact on host OS*. The former measures the performance delivered on the guest OS, while the latter evaluates the impact on the host OS when a guest OS is using up all its virtual CPU.

4.1 Performance of Guest OSes

Figure 1 to Figure 4 aggregate the results for the aforementioned benchmarks. To ease comparisons, results have been normalized against the measurements obtained on a *native* Ubuntu environment (this latter is represented as the unitary value). As the plots represent performance lag relatively to the native environment, smaller values mean bet-

ter performance. The only exception is Figure 4 where absolute values are used: higher values mean better network bandwidth performance.

As shown in Figure 1, for the 7Z benchmark all virtual machines were slower than the native environment. VmPlayer was the best performer, with a 15% performance drop, VirtualBox was 20% slower, while VirtualPC induced a 36% impact on performance relatively to the native execution. QEMU was clearly the worst performer, being more than twice slower than the native environment.

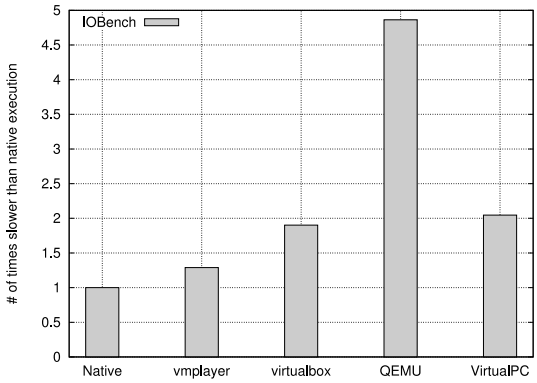

Figure 3. Relative performance of IOBench on virtual machines

Figure 2 displays the relative performance for the Matrix benchmark. Although the performance ordering of results is similar to the ones seen for 7Z, the performance drop is much smaller. Indeed, apart the QEMU virtual machine, which has a 30% performance drop, all other virtual environments induce a performance degradation below 20%. This means that floating-point performance is only marginally deteriorated within virtual environments. Thus, from the performance point-of-view, it is viable to run such applications under a virtual environment.

As seen on Figure 3, disk I/O is more severely impacted on virtual environments. Indeed, even VmPlayer, which is the fastest disk I/O environment, is 30% slower than a native execution. VirtualBox and VirtualPC are roughly twice slower than the native environment, while QEMU performs extremely poorly, being nearly five times slower than a native execution. This means that conducting disk I/O intensive operations under a system-level machine virtual environment significantly impacts the performance.

Absolute network performances against the native environment are shown in Figure 4. Obviously, the native mode was the fastest, achieving 97.60 Mbps. Since VmPlayer supports a *NAT* and a *bridged* mode for network communications, both were assessed. In bridged mode, VmPlayer yields performance very close to the native execution, delivering a network speed of 96.02 Mbps. This contrasts

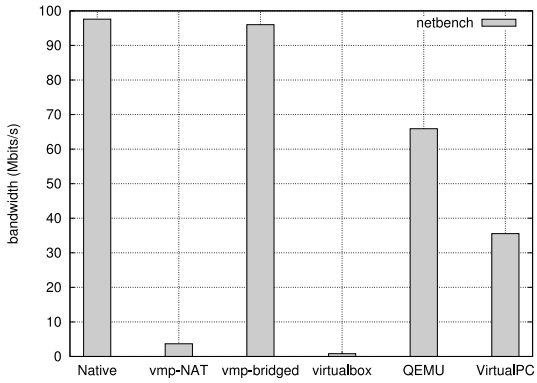

Figure 4. Absolute performance for NetBench on virtual machines

sharply with the NAT mode, which only attains 3.68 Mbps. Contrary to the results for the other benchmarks, QEMU performed quite respectably, averaging 65.91 Mbps, thus being the fastest of the whole set of tested virtual environments, apart VmPlayer in bridged mode. Under VirtualPC, the available bandwidth was 35.56 Mbps. Finally, VirtualBox really underperformed, being nearly 75 times slower than the native execution. The slow speed delivered by the NAT mode is due to the high overhead of this approach [17].

As observed from the CPU and IO benchmarks, while virtual machines moderately degrade performance for CPU-bounded applications and thus can be used for such applications without much loss, impact on IO-bounded applications is much more severe. This makes system-level virtual environments unattractive for running such applications, even considering the benefits of sandboxing and application isolation that such environments deliver.

4.2 Impact on host

An important issue regarding virtual environments for volunteer computing is the performance impact that virtual machines can induce on the hosting machines. Specifically, what is the performance drop felt by applications run on the hosting machine, while a guest system-level virtual machine is executing volunteer tasks? Indeed, if this impact is excessively high, it will most certainly discourage resource owners to volunteer via virtual machines. Next, we assess the overhead on memory and CPU when the virtual CPU of a guest OS is running at full capacity.

4.2.1 Impact on Memory

The memory footprint of system-level virtual machine is defined in its configuration, with the virtual machine committing all the configured memory when it is running. In this

way, the memory consumption is configurable, constant and well-known. This is important for volunteers, since they will know right from the start the amount of memory of their systems which is effectively being volunteered. As stated before, for all experiments reported in this paper, we set the virtual environments with 300 MB. This value might seem high for today's standard machines that have, commonly, 1 or 2 GB of RAM memory. However, we believe that lighter Linux distributions can be used, requiring half of this value. Moreover, 3 and 4 GB are becoming standard on new machines and thus the relative memory space occupied by a guest OS can still be reduced.

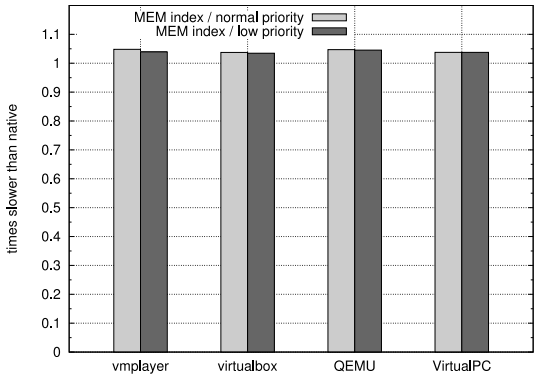

Figure 5. Relative performance (MEM index)

4.2.2 Impact on single-threaded applications

Due to the high costs of IO under virtual machines, which practically prohibits the execution of IO-bound application on virtual machines, we solely assess the impact on the host machine for CPU-bound applications. For this purpose, we measured the behavior of a CPU-oriented benchmark. We ran NBench (described ahead) on the host OS under two scenarios. In the first scenario, we ran the benchmark together with a virtual machine that was processing tasks from the Einstein@home, and thus using the virtual CPU at its full capacity. This corresponds to a real usage scenario. In the second scenario, the benchmark was run on the host OS, with no virtual machine present. Thus, the performance difference between the two scenarios corresponds to the overhead induced by the presence of an active virtual machine.

The NBench benchmark is derived from the well-known ByteMark [6] benchmark. Specifically, the benchmark was ported from Linux, with its C source code compiled with Visual Studio 2003 in release mode. The NBench application relies on well-known algorithms to summarize a computer performance with three numerical indexes: MEM for memory, INT for integer and FP to expose floating point performance. Although these indexes are not suitable for absolute comparisons with NBench original values, since the operating systems and the compilers are different, they can still compare relative performance and thus measure the overhead induced on a host OS, by a running guest virtual machine.

We should emphasize that we could not use the NBench application to evaluate the guest OS. Indeed, NBench resorts to numerous timing measurements of extremely short periods, and the lack of precision of time measurement in virtual machines [22], yields misleading results when the benchmark is run in a virtual environment. However, these timing measurement issues do not exist for executions performed in the host OS.

Figure 5 and Figure 6 plot the performance overhead for the MEM and INT indexes, respectively. Each plot presents the overhead relatively to the performance measured on the hosting OS with no virtual machine. Thus, smaller values mean less overhead. The measurements were taken on the same computing environment (machine, hosting OS and virtual machines) that was used for assessing the performances of virtual machines. To check the influence of the priority level assigned by the host OS to the virtual machine, we first ran the tests with the virtual machine software set to normal priority, and then repeated the experiment with the *idle* level, which is the lowest priority level available in Windows XP. To allow for a better comparison, a normal priority result is plotted next to the corresponding idle priority.

From the plots, it can easily be seen that the highest overhead occurs with the MEM index (Figure 5), but even for the worst case, it is under 5%. For the INT index (Figure 6), overhead averages 2% for all the virtual environments. Finally, practically no overhead was observed regarding floating point (to conserve space, we omit the plot for the FP index). Additionally, all virtual environments performed similarly. Likewise, the priority level assigned by the host OS only marginally influence performance, with both normal and idle levels yielding similar values.

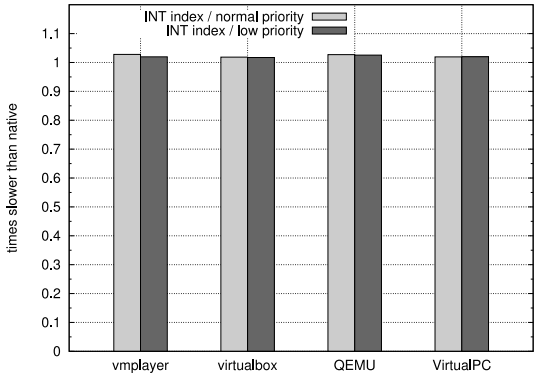

Figure 6. Relative performance (INT index)

The marginal overhead appears to be a consequence of the dual core processor, with the benchmark scheduled by the hosting OS in one core, while the virtual environment is scheduled in the other core. The slight overhead in the MEM index might be due to the fact that some cache collisions occur over the 4 MB level 2 cache, since this cache is shared between the two cores. The results also give proofs that resources can be volunteered without practically no consequence on the performance of the hosting machine.

4.2.3 Impact on multi-threaded applications

To further assess the impact on the host OS caused by running a virtual machine with full use of virtual CPU, we set up the following testbed: in the Linux guest OS, we ran the BOINC client attached to the Einstein@home public volunteer project, thus consuming the whole virtual CPU. To minimize impact, and reproduce real conditions, the execution priority for the virtual machine was set to idle.

Simultaneously, in the Windows XP host OS, we run the 7z application in benchmark mode since this application allows to set the number of executing threads. Therefore, we can distinctly measure the impact on the host OS of a running virtual machine, by executing 7z in single and multithreaded mode, by setting the thread number to one and two, respectively. Figures 7 and 8 display the performance measurement delivered by the 7z application while the Einstein@home client is demanding 100% from the CPU of the virtual machine. To ease comparisons, for each environment, the result for the single-threaded 7z execution is plotted to the left of the equivalent result performed under the dual-threaded mode. The entry *no VM* represents the control case, corresponding to results gathered when the virtual machine was not running.

Figure 7 displays the percentage of time the processor is working for the 7z application. A 100% CPU usage for one thread means that the whole CPU was dedicated to the thread running 7z. Likewise, a 180% CPU usage for two threads means that average CPU usage is about 90% for each thread. Therefore, the full availability of two CPUs (cores in this case) is represented by 200%. Confirming results of Section 4.2.2, the plot clearly illustrates that no impact is sensed for single-threaded application, since 7Z reports 100% CPU availability for all except for QEMU, which nonetheless delivers a value close to 100%. However, for dual-threaded applications, the *no VM* execution only achieves 180% CPU usage. This means that 20% of CPU can not be exploited, possibly due to the limitations and overhead of the hardware (contention on memory when both cores try to access memory), OS and of the multithreading subsystem.

Different virtual machines using up all the virtual CPU have different impacts on the host OS. Indeed, while

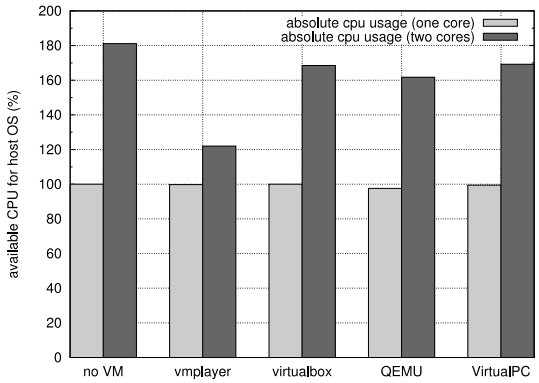

Figure 7. Available % CPU for host OS when guest OS is running at 100%

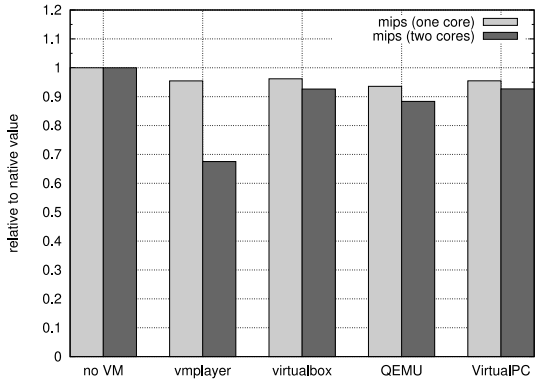

Figure 8. MIPS for 7z when guest OS is running at 100%

QEMU, virtualbox and virtualPC limit the CPU for host OS to approximately 160%, that is, they roughly cause a 20% overhead, VmPlayer induces a steeper penalty, since 7z only reports a 120% CPU availability. This means that VmPlayer causes a 60% overhead, roughly thrice more than the other virtual environments. Figure 8 displays the MIPS metric as computed by the 7z application. Specifically, this metric counts the number of instruction per second performed by 7z. The results shown on the plot correspond to the ratio between the MIPS obtained for the execution on the host OS when a virtual environment is running and the same execution when no virtual machine is present. It can be observed that the presence of VmPlayer reduces MIPS in roughly 30%, while the other virtual environments cause a near 10% degradation. The higher degradation caused by the VmPlayer environment might be due to the more CPU demanding internal optimization techniques that allows this environment to perform faster than the other ones.

5 Related Work

Figueiredo et al. [10] were pioneers on the usage of virtual machines for the execution of grid applications. They address several problems, such as the impact on performance, and the needs for a viable deployment infrastructure. Regarding performance, they report a less than 10% overhead for CPU-bounded benchmarks executed under VMware Workstations 3.0 having both the host OS and the guest OS running Red Hat 7.1. As our study confirms, their performance results still hold on for the VMware platform. Their work has evolved to the WOW system [11]. This system implements fault tolerant virtual cluster over large scale networks by resorting to virtual machines and P2P techniques. The system uses virtualization to provide resource isolation and to wrap the heterogeneity of machines.

Csaba et al. [9] present a practical approach for the use of virtual machines in desktop grid environments. Their main goal is to devise a generic architecture that can be transparently integrated with desktop grid middleware (in particular, the Sztaki DG and extensions are targeted [4]). The proposed architecture relies on a virtual machine base image, which is used to create instances. The number of instances to create in a given machine depends on the hardware, namely on the number of CPU cores. Each virtual machine instance then separately maintains the changes it has performed over the base image. This allows for an easier migration and further strengthen the sandboxing robustness of the system, since no changes can be made to the base image. The authors selected the QEMU virtual environment with its accelerator KQEMU. The choice was motivated by the flexibility of the virtual environment, when compared to other system level virtual machines. Indeed, QEMU provides overlay and copy-on-write images, backdoor access and single user process instance [4]. For Csaba et al., flexibility-related issues have higher precedence than performance, since deployment and manageability are determinant factors when considering virtual environments for desktop grid computing.

Tanaka et al. [18] compare the floating-point performance of VmPlayer-based virtual machines versus OS running in a so-called real machine. Specifically, they resort to the YafRay open source ray tracer, measuring the wall clock time needed for the ray tracing of a fixed scenario under the given virtual and real environments. They considered both Linux and Windows as guest OS, observing that both operating systems induced performance penalties relatively to an execution under a native environment. Similar to our results, both operating systems, when run in a virtual environment, yield performances that are inferior to the native environment. However, the authors found out that as guest OS, Windows delivers slightly worst performance than Linux. They explained this behavior by the higher usage that Windows does of the kernel mode. Indeed, under VmPlayer, instructions in kernel mode are emulated, running much slower than user mode instructions, since these ones are directly executed in the processor.

The philosophy behind the Minimal intrusion Grid (MiG) [1] is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible and that emphasizes the protection of harvested resources. For the purpose of safely harvesting resources, the system resorts to system-level virtual machines for the execution of foreign tasks, thus protecting local resources from buggy and/or malicious guest applications. Similarly to the scenarios benchmarked in this paper, MiG runs a Linux guest OS in the context of a virtual machine which is executed on top of a Windows OS. The study confirms that the system-level virtual machine approach is viable for harvesting desktop grids, at least from an implementation point of view. The study does not provide data regarding overall performance, nor the performance overhead caused by system level virtual machines.

Gonzalez et al. propose the *virtual desktop grid computing* system [13]. Their approach is to combine desktop grid middleware with VmPlayer. They resort to BOINC for distributing a specially crafted *initialization* workunit. This workunits holds the OS virtual image and the worker applications. Once this stage is completed, the BOINC client uses the large initialization workunit to launch VmPlayer with the hosted OS image, which in turn starts the application whose execution is sought within the desktop grid environment. The application can then connect to the project server and requests regular workunits. The main motivation for this approach is to cleanly support legacy applications out of the box. One problem relies to the massive size of the initialization workunit. In an experiment performed by the authors, a 1.4 GB initialization workunit was required. For the time being, this size mostly limits the system to local area environments. However, an optimized scheme for provisioning virtual environments can be used as described by Chadha et al. [7]. Similarly, pure P2P distribution protocol such as BitTorrent can also be considered [8].

The P2P Distributed Virtual Machine (P2P-DVM) [14] is a middleware that promotes desktop grid computing in a peer-to-peer fashion. The system provides support for parallel programming environments such as MPI, PVM and BSP for the execution of desktop grid applications. Regarding virtualization, all processes of the P2P-DVM system run on top of a Linux installation, which is hosted by a virtual environment such as VMWare or Xen (among other possibilities). The goal of using virtual machines is to allow for full isolation between the hosting resources and the guest applications. Interestingly, P2P-DVM implements system-level checkpointing for each of its processes that are run inside the virtual environment.

6 Conclusion

We measured the performance drop that applications suffer when they run on a guest OS of a virtual machine. We also evaluated the impact on local resources, measured at the host OS level, caused by running desktop grid tasks under virtual environments. All experiments were performed with a Core 2 Duo machine i) to evaluate how this architecture copes with virtual machines and ii) to assess the performances that such architecture can deliver when running simultaneously Windows XP as host OS and an Ubuntu Linux as guest OS. In particular, we show that a machine fitted with a dual core processor can withstand, with marginal impact on its performance, the presence of a virtual machine as long as only single threaded applications are run in the host OS. However, multi-threaded applications running at the host OS suffer a performance drop that ranges from 10% to 35% compared to the same execution in the native environment.

For the applications executed on the virtual environment, the performance impact depends on the application type and on the virtual machine software. For CPU-bounded application, the overhead is acceptable, since it revolves around 15% to 30% for the virtualized environments. The only exception is QEMU whose high overhead practically halves the execution speed. However, both disk IO and network IO performances are severely penalized on the studied virtual machines and thus the execution of IO-bound applications should not be considered on such environments.

Acknowledgment

This work was partially supported by Fundação para a Ciência e a Tecnologia under the project GRID/GRI/81727/2006, "GRID para simulação e análise de dados de ATLAS/LHC".

References

[1] R. Andersen and B. Vinter. Harvesting idle windows cpu cycles for grid computing. In H. R. Arabnia, editor, *Proceedings of the 2006 International Conference on Grid Computing & Applications (GCA 2006)*, pages 121–126, Las Vegas, Nevada, USA, June 2006. CSREA Press.

[2] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing, 2004, Pittsburgh, USA*, 2004.

[3] D. P. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing. In *IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 73–80, 2006.

[4] Z. Balaton, G. Gombas, P. Kacsuk, A. Kornafeld, J. Kovacs, A. C. Marosi, G. Vida, N. Podhorszki, and T. Kiss. SZTAKI desktop grid: a modular and scalable way of building large computing grids. In *IPDPS'07*, 2007.

[5] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track. April 2005, Anaheim, CA, USA*, pages 41–46, april 2005.

[6] Bytemark (http://www.byte.com/bmark/bmark.htm), 2008.

[7] V. Chadha, D. Wolinsky, and R. Figueiredo. Provisioning of virtual environments for wide area desktop grids through redirect-on-write distributed file system. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008)*, pages 1–8, April 2008.

[8] F. Costa, L. Silva, G. Fedak, and I. Kelley. Optimizing the data distribution layer of BOINC with BitTorrent. In *IPDPS'08*, pages 1–8, 2008.

[9] A. Csaba Marosi, P. Kacsuk, G. Fedak, and O. Lodygensky. Using virtual machines in desktop grid clients for application sandboxing. Technical Report TR-0140, Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID - Network of Excellence, August 2008.

[10] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *ICDCS'03*, 2003.

[11] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. *Journal of Grid Computing*, 5(2), 2007.

[12] Iperf - The TCP/UDP Bandwidth Measurement Tool (http://dast.nlanr.net/Projects/Iperf/), 2008.

[13] D. Lombraa-Gonzalez, F. de Vega, L. Trujillo, G. Olague, and B. Segal. Customizable execution environments with virtual desktop grid computing. In *Parallel and Distributed Computing and Systems, PDCS*, 2007.

[14] L. Ni, A. Harwood, and P. J. Stuckey. Realizing the e-science desktop peer using a peer-to-peer distributed virtual machine middleware. In *MCG '06: 4th International Workshop on Middleware for Grid Computing*, 2006.

[15] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin. Performance evaluation of virtualization technologies for server consolidation. Technical Report HPL-2007-59R1, HP Labs, September 2008.

[16] I. Pavlov. 7-Zip (http://www.7-zip.org), 2008.

[17] S. Rixner. Network virtualization: breaking the performance barrier. *ACM Queue*, 6(1):36–41, 2008.

[18] K. Tanaka, M. Uehara, and H. Mori. A case study of a linux grid on windows using virtual machines. In *AINAW '08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 195–200, 2008.

[19] VirtualBox (http://www.virtualbox.org), 2008.

[20] VirtualPC, Microsoft Inc. (http://www.microsoft.com), 2008.

[21] VmWare, Inc. (http://www.vmware.com), 2008.

[22] Time keeping in VmWare Virtual Machines (http://www.vmware.com/pdf/vmware_timekeeping.pdf), 2007.