J Grid Computing (2007) 5:83–98
DOI 10.1007/s10723-006-9052-6

Scalability Comparison of Four Host Virtualization Tools

Benjamin Quétier · Vincent Neri · Franck Cappello

Received: 15 February 2006 / Accepted: 11 July 2006 / Published online: 19 September 2006

Abstract Virtualization tools are becoming popular in the context of Grid Computing because they allow running multiple operating systems on a single host and provide a confined execution environment. In several Grid projects, virtualization tools are envisioned to run many virtual machines per host. This immediately raises the issue of virtualization scalability.

In this paper, we compare the scalability merits of Four virtualization tools. First, from a simple experiment, we motivate the need for simple microbenchmarks. Second, we present a set of metrics and related methodologies. We propose four microbenchmarks to measure the different scalability parameters for the different machine resources (CPU, memory disk and network) on three scalability metrics (overhead, linearity and isolation). Third, we compare four virtual machine technologies (Vserver, Xen, UML and VMware).

The results of this study demonstrate that all the compared tools present different behaviors with respect to scalability, in terms of overhead, resource occupation and isolation. Thus this work will help user to select tools according to their application characteristics.

Key words virtual machines · scalability · performance comparison · overhead evaluation

B. Quétier (✉) · V. Neri · F. Cappello
INRIA/LRI, Université Paris-Sud, Orsay, France
e-mail: quetier@lri.fr

V. Neri
e-mail: neri@lri.fr

F. Cappello
e-mail: fci@lri.fr

1 Introduction

Virtualization is becoming a key feature of Grids where it essentially provides the feature of abstracting some specific characteristics of the Grid infrastructure. The concept of a Virtual Machine (VM) is not recent and was proposed very early in the history of computers [14]. There are several approaches at the hardware and the OS levels. As presented in [13], an example of pioneer works was the VM of the IBM 370. VM were identical copies of the hardware where every copy runs its own OS. Currently, there is a trend toward the adoption of the VM concept. Microprocessor vendors (AMD, Intel, IBM, etc.) are proposing new hardware mechanisms to improve virtualization performance in their last generation of products. The main motivations of machine

virtualization in the context of Grid and large scale distributed system experimental platforms (PlanetLab, Grid eXplorer) are the following: a) each VM provides a confined environment where non-trusted applications can be run, b) a VM can limit hardware resource access and usage, through isolation techniques, c) VM allows adapting the runtime environment to the application instead of porting the application to the runtime environment, d) VM allows using dedicated or optimized OS mechanisms (scheduler, virtual memory management, network protocol) for each application, e) Applications and processes running within a VM can be managed as a whole. For example, time allocation can be different for every VM and under the control of a VM manager (also called scheduler, hypervisor or orchestrator).

Because of VM's potential, many Grid projects are studying the integration of machine virtualization in their research and development. In some projects [5], VM is envisioned to reduce the number of Globus Gateways at each Grid site. In [12], a lightweight Grid VM is developed to support Metacomputing while providing strong security and fault tolerance. Some Grid services are proposed to provide machine virtualization in Grid, allowing high security and flexibility by decoupling the service execution from the actual resources [11]. In [22], the authors present an approach for fast VM instantiation in Grid servers. Recently, several projects have extended the concept of machine virtualization to the one of Virtual Clusters. In [16], the authors extend VNET, a virtual private network tool implementing a virtual local area network over a wide area using tunneling, for VMs in Grids. The In-VIGO [1] Grid computing system uses virtualization technologies to create dynamic pools of virtual resources. In SODA [10], an application service is dynamically created on demand and automatically bootstrapped as a set of virtual service nodes; each one being a slice of a real host. Machine virtualization currently plays a significant role in Grid and P2P research platforms. In PlanetLab [3], machine virtualization is used to confine the experiment of every user in a dedicated domain, referred to as a slice. In MicroGrid [15], virtualization allows Globus applications to run without modification. The Grid eXplorer project (a large scale distributed system emulator) [8] is relying on VM technologies to build a distributed system emulator scaling to 100,000 virtual nodes running over 1,000 physical nodes.

As the use of VM become more widespread on Grids, multiple virtual machines or virtual OS will often run on a single Grid node. For example, in [6], The authors suggest assigning VMs on a per user basis. Another study [19] on Internet Services considers strong isolation as a major feature for security between services where data are mostly unshared. Every service is executed in an isolated environment, and the authors consider thousands of VMs per physical node. Other research [21] on attack detection suggest to run a large number of virtual honeypots on dedicated systems within production networks to ease attack data correlation. This trend immediately raises the issue of machine virtualization scalability. Although there are several previous works on QoS for VMs (including research on fair-share [4]) and specific scalability evaluation [19], to the best of our knowledge there are surprisingly only a few published results on comparing the scalability of VMs, which we discuss in Section 2.7. There are neither benchmark suites for comparing VMs, nor clear metrics or methodologies. This paper addresses this lack by 1) motivating, presenting, and discussing metrics and methodologies and 2) using those metrics and methods to compare four VM technologies. The metrics discussed and justified in Section 3 are the following: the overhead of the virtualization technique (in terms of CPUs, memory, disk and network utilization), the effect on performance when the number of VMs increases, the isolation between VMs (also in terms of CPUs, memory, disk and network utilization), and the VM creation time.

In the next section we present a short survey of system virtualization techniques, describing fundamental concepts and popular approaches. Section 3 motivates and describes the metrics we propose to evaluate the scalability of VM tools. Section 4 presents our method of applying these metrics, and the microbenchmark used for the evaluation. We compare four well-known virtualization tools (VMware, Xen, Vserver and UML) in Section 5. In the conclusion, we summarize what we have learned in this study.

2 Machine Virtualization Approaches for Commodity Hardware

There are several approaches for running virtual machines on the resources of a single machine. The article [13] presents a large set of virtualization techniques. In this article we will focus on recent approaches for commodity hardware that allow applications to be run natively within virtual machines.

2.1 Operating System

Classical OS offer a basic resource virtualization. When many processes are running, the kernel's scheduler virtualizes machine's resources (CPU, memory and I/O) using time multiplexing. This approach presents several limitations that make other virtualization approaches more appealing in the context of resource sharing between Grid users and in experimental distributed systems. With the exception of recent OS like Solaris 10 and some experimental OS, most of operating systems do not implement natively the concept of process hierarchy, which is essential for implementing isolation and confinement in machine virtualization. (The OS can not distinguish processes belonging to different virtual machines, and as a result, process scheduling and isolation cannot be done on a per VM basis.) Thus, even though nothing precludes executing the processes of all virtual machines on a single machine, there is no mechanism in an OS ensuring resource fair share between virtual machines. Although, there are some basic mechanisms that allow confinement of virtual machines (chroot, MMU), they do not protect against simple attacks like buffer overrun.

2.2 Virtualization Trade-off

There are several degrees of machine virtualization. Some approaches allow installing and running unmodified guest operating systems over a host OS. Other approaches require porting an OS to run concurrently over an abstract machine. Another approach is to use context separation between processes of virtual machines to implement virtualization. As discussed earlier, there are many uses of virtualization in the context of Grid. A first objective in virtualization research is to provide transparency, allowing the installation of several virtual machines on the same physical machine. Depending on the degree of virtualization, transparency may require the use of 'full virtualization' techniques. This approach requires heavy use of x86 ia32 instruction set executed natively by most of the commodity PC since it has not been designed with full virtualization in mind. Limitations concern both execution of some privileged instructions and virtual memory management. A second goal is the portability of the virtualization technique. Some virtualization techniques require the installation of a specific host OS. On the contrary some other approaches impose minimal host OS adaptation to implement virtualization. A third goal is VM performance. The virtualization technique should impose as little overhead as possible to virtual machines. Confinement is a fourth objective. The different virtualization techniques provide different degree security of among virtual machines. A fifth goal is performance isolation. Workload imbalance between virtual machines running on the same machine should not compromise the fair share of the physical machine resources between virtual machines. Unfortunately, these goals may conflict with each other [2]. Consequently, one must consider the cost and benefits of a particular VM. Thus all the VM approaches presented below reflect specific trade-offs.

2.3 Processor Virtualization

VMware and Microsoft virtual server provide full processor virtualization. VMware [17] is a virtualization software for machines based on x86 architecture. There are three versions. VMware workstation and GSX require a host operating system, but are highly portable and do not require any modification of the host OS. WMware ESX is by itself an operating system working as the host. It provides better performance at the cost of reduced portability. The architecture contains three software (VMapp, VMDriver and VMM). In VMware, virtualization works at the processor level. The virtual machine privileged instructions are trapped and virtualized by the VMware process (VMapp). Others instructions are directly executed by the host processor. All hardware

resources of the machine are also virtualized. So, this complete virtualization allow users to install and run concurrently on the same machine different OS based on the x86 architecture, via virtual BIOS. Host and virtual OS are scheduled by the VMdriver (Virtual Machine Driver). I/O initiated by a guest OS are trapped and executed by the host OS.

2.4 UML (Kernel Replication)

User Mode Linux [9] allows launching Linux OS as applications of a host machine running Linux. This feature allows a large variety of configurations and utilizations for machine virtualization not limited to the host machine characteristics. Guest Linux OS are run as user space processes. A special thread manages the separation between the host OS and the guest kernel. Processes of each guest OS are implemented as threads within each guest OS. System call entry and exit of guest OS are trapped and executed by the host OS. An UML host patch named SKAS for Separated Kernel Address Space significantly improves performances of guest machines by separating address spaces and reducing context switches. (The host scheduler manages only one process per guest machine.) This approach offers high transparency and portability at the cost of performance and confinement limitations. However, since the guest OS are run in user space, they are weakly protected compared to 'full virtualization.' Also, the host kernel performs all system calls leading to a bottleneck.

2.5 Operating System Virtualization

Xen [2] is a virtual machine monitor for x86 architecture, allowing the concurrent executions of multiple OS while providing resource isolation and execution confinement between them. Xen relies on the concept of paravirtualization [18] and [20]. It provides a set of abstractions (virtual cpu, virtual memory, virtual network, ...) over which different OS can be ported. These abstractions are not necessarily similar to the actual hardware resources of the host machine. In Xen, a domain0, created at the host machine boot, accesses a control interface and runs the application level management software. The domain0 control interface allows launching and shutting down other domains where guest OS are run. To implement the virtualization, Xen uses several tools. It offers two different schedulers: 'Borrowed Virtual Time' which provides proportional fair share of the CPU time and 'Atropos' which is a soft real time scheduler. This last scheduler provides guarantees about the absolute share of the CPU time, with a facility for sharing slack CPU time on a best-effort basis. For memory virtualization, Xen reserves for each VM a certain amount of the physical memory. This amount is fixed at the VM's creation but it can be adapted if a user needs more or less memory (without quitting or rebooting the VM). For the network, each VM has one or more network interfaces and communication between interfaces is implemented using two token ring buffers (one for sending, one for receiving). Xen is not transparent, since guest OS have to be ported before being installed and run. It provides a high degree of confinement, in-between CPU virtualization and resource virtualization (see below). Performance evaluation compared to VMware and UML demonstrates outstanding performance. Compared to VMware, UML and Vserver, Xen is much less portable, since it requires the installation of a specific host OS. Denali [19] shares the same general principle of Xen. It requires to port guest operating systems to the Denali isolation kernel.

2.6 Resources Virtualization

Vserver [7] is a kernel patch based on partitioning, using what its authors call a 'security context.' It creates independent Virtual Private Servers (VPS). These VPS are running simultaneously on a physical machine and share its hardware resources. To implement VPS confinement, Vserver uses a context separation ensuring that only processes belonging to the same context are visible between each other. This is implemented by modifying the host OS scheduler and adding a 'context tag' to the process id. To force the VPS users to stay in their context, Vserver uses a security improved version of the linux 'chroot' command which forbids users to access directories belonging to other contexts. Management of

other resources like memory also benefits from confinement add-ons to ensure context separation. Compared to UML and Xen, Vserver does not need to launch one kernel for every VPS. However Vserver requires patching the host OS kernel, reducing its portability merit. It does not allow installing and executing guest OS or kernels, which significantly limits its transparency. However, since there is no intermediary between virtual hosts processes and host kernel, the performance of the Vserver is similar to that of the virtual hosts of the host OS. Indeed, the Vserver avoids the technical issue raised by some IA32 privileged instructions and virtual memory management.

2.7 Existing Results Concerning the Scalability of Machine Virtualization Tools

Existing results on scalability evaluation concern low level virtualization mechanisms and rather coarse-grain applications. Most of the evaluations were performed with different tests, methodologies and metrics. Very few results compare the various virtualization tools.

Regarding low level mechanisms for evaluation, in [11], the results concern the cloning time of virtual machines with VMware for different sizes of virtual machines. The experiment mostly considered VMware, up to 16 VMs per physical node. The exponential increase of the cloning time according to the number of running VMs is presented but not explained. Other evaluations of low level mechanism performance are presented in [19]: interrupt and idle process management, VM machine memory footprint in the kernel and memory. Concerning the coarse-grain tests, the benchmarks used in [2] are Lmbench, SPEC CPU, OSDB and SPEC WEB. In [19], application tests use a Web Server and Quake II server. Scalability evaluations with the Web server concern the 'request per second' metric and are performed in 'in-core' and 'out-of-core' conditions. For Quake II server, the metrics are the throughput and latency to process updates sent by clients. The paper presenting Xen [2] details some scalability comparisons between Linux and XEN, running multiple instances of the SPECWEB99 either as processes of Linux or as an application in XEN VM. The paper also presents some results about overhead in memory occupation of XEN when running 128 VMs the on the same machine. However, there is no comparison with other machine virtualization technologies. Moreover, no result is presented about fair sharing among virtual machines. The authors of [11], present one comparison result considering the cloning time of VMware and UML, but there is no analysis of this result.

Altogether, there is a lack of global comparison of the scalability merits of virtual machine technologies using the same metrics and methodology.

3 Metrics and Methodologies

As discussed in the previous section, several comparison results already exist about VM scalability. The results concerning low level virtual machine mechanisms mostly exhibit the design properties of these mechanisms. However 1) a comparison of the different technologies at this level is not always possible since the code of several tools is not available and 2) these results have little significance for users who need to run applications or services within VM and want to understand the performance and capacity they can count on. Other existing comparison results mainly consider specific two applications: web servers and quake servers. There are other uses of VM technologies and results obtained from specific applications may not be applicable to other applications with different characteristics. So, an application with different characteristics might not be able to apply the results to differentiate one VM from another. From the user point of view, as for other real computers, the relevant information is the performance of the resources taken individually (CPU, Memory, Disk and Network), their capacity, how their usage changes when several VMs are running on the same hardware and the time required to start an execution on a VM, when it involves the creation of the VM.

To motivate further the use of resource oriented micro-benchmarks, we will demonstrate that complex applications cannot be used to understand the VM scalability, even on a simple scenario. Let us consider that, for security reasons, the administrator of a Grid server decides

to launch a confined instance of a web server for every client. Every web server application is confined by a different VM. Two clients connect to the Grid server; the first one sending two HTTP GET requests and the second client issues only a single HTTP GET request (we assume that the three requests target different 600 MB files). Two VMs are launched to host the two web servers. We consider one essential criterion of VM scalability: performance isolation. As discussed later, performance isolation is a property of the VM manager to distribute the resources utilization in a fair way. Figure 1 presents the execution times (in seconds) for the three simultaneous WGET using raw HTTP or HTTPS on a server running the Apache software. In the latter case, the CPU also contributes significantly to the execution time for encoding and decoding (the measurements have been obtained in the cluster presented in Section 5).

Results in Figure 1 clearly demonstrate different behaviors between the virtualization tools. Some tools (Vserver and Xen) seem to balance the load between the three processes. The other tools (UML and VMware) seem to balance the load between the two virtual machines. UML and VMware perform significantly worse compared to Vserver and Xen. Users of VM need to know if the four VM technologies will behave identically (e.g. with respect to load) with their applications which certainly exhibit different balances between resource usage. Thus, these kinds of results are clearly not enough to help them in their decision. We believe that an evaluation of the scalability of VM on a per resource basis will provide better insight to the user about the behavior of VM technologies on their applications and will help VM designers in discovering potential limitations and properties of their VM technologies.

	Vserver	XEN	UML	VMware
HTTP wget				
VM1(1)	22 s	21 s	140 s	36 s
VM1(2)	21 s	22 s	140 s	36 s
VM2	23 s	22 s	45 s	27 s
HTTPS wget				
VM1(1)	50 s	49 s	428 s	120 s
VM1(2)	49 s	51s	430s	119 s
VM2	52 s	50 s	287 s	85 s

Figure 1 Resource isolation evaluation of four virtualization tools on a simple scenario.

Thus we propose a set of metrics, on a per resource basis, divided in two classes: performance and usability. For every metric, we propose a method to evaluate the merit of the virtualization system. This method will be used in the next section to compare the virtualization tools.

3.1 Performances

In this part, we will focus on metrics related to the machine virtualization performances. A virtualization tool should exhibit a low constant virtualization overhead and allow performance isolation between virtualized machines, independent of the number of running virtual machines. For all metrics, we provide a simple formula, which will be used in the evaluation section as a performance reference. These formulas are also intended to give a strict definition of the metrics. All these metrics will be used for the four resources of virtual machines (CPU, memory, network and disk). As discussed in the previous section, real applications and OS benchmarks often mobilize several resources simultaneously or concentrate on OS specific operations, making it difficult to understand the impact of the virtualization tools on every resource. So, we deliberately choose very simple microbenchmarks, each of which stresses a single resource. Their simplicity allows us to show the scalability limitations of virtualization tools, on a per resource basis.

CPU measurements use a program that iteratively computes an estimate of $\sqrt{2}$. This test was chosen because it minimizes the memory usage. Memory measurements use a program stressing the memory and with negligible computation: repeated sums of two big matrices in a third one. $C_{i,j} = A_{i,j} + B_{i,j} + C_{i,j}$ with the following initialization: $A_{i,j} = B_{i,j} = i + j$ and $C_{i,j} = 0$. Network measurements use the netperf benchmark. Disk measurement use a disk duplication tool (dd in

read only). The memory and computation consumption of netperf and dd are negligible compared on the activity they impose to the network and disk.

3.1.1 *Overhead*

To evaluate the virtualization overhead due to the virtualization mechanisms, we compare the execution time of an application running on a non-virtualized OS (T_a) with another instance of the same application run within a single VM (T_{av}). The overhead may be negligible for a single VM and becomes significant when several VMs run concurrently (context switch overhead may exist even if no application is executed on the other VMs). So we also compare T_a with T_{anv} when n VMs run concurrently. In this scenario, only a single VM actually runs the application. The other $n-1$ VMs are free of application. $O_v = T_{av} - T_a$ gives a reference virtualization overhead, $O_{vn} = T_{anv} - T_a$ gives the virtualization overhead for n VMs.

3.1.2 *Linearity*

To evaluate changes in the performance as the number of running virtual machines is increased, we first measure the execution time of an application running on a single virtual machine. Then we measure the execution time of the same application running concurrently on several virtual machines (with the same replicated data sets). If the virtualization is constant (i.e. independent of the number of virtual machines), then the maximum of the application execution times should be an affine function of the number of virtual machines running the application. If an application takes a time t to run on a virtual machine, then, if this application is executed concurrently on n VMs, the maximum of the application execution time is: $tmax = O_v + t \times n$, where O_v is the virtualization overhead for 1 VM. Some virtualization approaches may have a more complex behavior, exhibiting execution times according to the number of running VMs greater than the one driven by the virtualization overhead only. In such case, we will consider that there is at least a second component in the overhead, in addition to the one imposed by the virtualization mechanisms.

3.1.3 *Performance Isolation*

There are different definitions of performance isolation and no consensus yet about how VM should behave with respect to this parameter. In principle, performance isolation ensures that in a situation of load imbalance between VMs, all VMs will still get an equal access to the machine resources. As demonstrated in the previous test with a web server, the VM technologies have different behaviors. Discussions with VM designers and users lead us to the conclusion that, for some of them, the performance isolation should be performed between processes while the others consider that it should be performed between VMs. Because this topic is still under discussion, we will only present the respective isolation characteristics of the VM technologies without deciding which policy is 'correct.'

To measure performance isolation we run two VMs concurrently (VM1 and VM2), executing one application on VM1 and two copies of the same application on VM2. In the case of a scheduling by process: all VMs should finish at the same time. In the case of a scheduling by VM: before the end of the application running on VM1, half of the machine resources should be allocated to every VM. After the end of the application running on VM1, half of the resource should be allocated to every of the two applications running on VM2. The application of VM1 should terminate the execution before VM2. More formally if a) T_a is the execution time of an application on a single VM, b) T_{a1} and T_{a2} are the execution times of the same application on VM1 and VM2, then performance isolation should be illustrated by the following relation: $T_{a1} = 2T_a$ and $T_{a2} = 3T_a$.

3.1.4 *Communications Between Virtual Machines*

Several VMs will run on the same machine, possibly exchanging messages. The communication performance (bandwidth and latency) depends on how the virtualization implements internal communication between VMs. The performance may also evolve with the number of running VMs on

the host machine. The benchmark used to evaluate the inter-VMs communication performance is based on Netperf. Each VM runs a Netperf server and Netperf client, and all VMs are connected in a ring topology. All Netperf executions are synchronized before measurement. We run the test with 10 concurrent VMs. The result of this test should give the bandwidth available for every VM.

3.2 Usability

In this part, we present criteria distinguishing virtualization systems by some usability criteria. Indeed, some factors restrict the number of simultaneously running VMs.

3.2.1 *Startup Time*

Startup time is an important characteristic of a virtualization tool, especially in a system where there are frequent reconfigurations. It is a part of the responsiveness of the virtualization tool. We consider the following metric: the time to launch n VMs, all VMs being started concurrently. Let $T_{startup}$ be the time to boot all n VMs, we have $T_{startup} = \max(\text{startup}[1, n])$. This metric can be affected by different services launched during the boot sequence (like a ssh server).

3.2.2 *Memory Occupation*

Memory occupation is a factor limiting the number of VMs running simultaneously. In some systems like UML, the amount of memory allocated to every VM can be specified. In some others virtualization tools, we need to measure the memory occupation of every VM. We also measure the memory occupation of the virtualization tools (another aspect of the memory overhead) and how it varies with the number of running VM. To evaluate the memory occupation, we measure the physical memory occupation before the execution of the virtualization system (just after the boot of the physical machine). Then we measure the memory when all VMs are booted and spin waiting (no application is running). On some systems like Xen, VM are launched from another VM (domain0). The physical memory of the host machine is not entirely visible from this VM (it can only access its own memory). A configuration file stores the amount of memory to be reserved for every VM. A simple runtime check allows verifying this value. Globally: let $M_{used(n)}$ be the total memory occupation for n VMs, M_{VM_i} the memory used by VM_i and M_{tool} the memory (overhead) used by the virtualization system, then: $M_{used(n)} = \text{Sum}(M_{VM_{[1,n]}}) + M_{tool}$.

4 Experimental Spectrum and Conditions

In this section, we present the experimental conditions used to evaluate the scalability merits of UML, VMware, Vserver and Xen. We tested and compared scalability merits of these four tools. For all virtualization tools we evaluated separately four resource types (CPU, memory, disk and network). The evaluation measured three main parameters: linearity, overhead, and performance isolation. Altogether this gives a three dimensional set requiring a set of 48 distinct experiments and systems settings. Linearity as well as overhead measurements multiplied the number of experiments due to the necessity to scale the number of virtual machine from 1 to 100. Thus the following tables and graphs present a set of about 3,000 experiment results: for linearity and overhead 10 measures repeated 10 times each with 16 configurations (four resources and four virtualization tools) and for isolation 16 configurations repeated 10 times so we get $2 \times (10 \times 10 \times 16) + (16 \times 10) = 3,360$ experiments.

The four virtualization systems tested and compared in this paper are:

- UML version for kernel 2.6.7
- Vserver version 1.29
- Xen version 2.0
- VMware Workstation 3.2 and GSX version. We will only give limited comments on VMware GSX versions due to the strict limitation of result publication imposed by the VMware license.

4.1 Experimental Conditions

To test the four virtualization systems, we have used PCs with AMD Opteron processors (one per

PC) running at 1.8 GHz with 1,024 KB of cache, 3 GB of main memory, Ultra ATA hard disk of 80 GB and BroadCom TG3 Ethernet interface. For some tests, several machines are connected together by a Netgear FS605V2 100 Mbps Ethernet switch. The HTTP GET tests use a 1 Gbps Ethernet connection between the PCs, with a D-LINK D65-1216T Gigabit Ethernet switch.

Here are the host and guest kernel settings for the different virtualization tools:

- **UML**: Linux with a 2.6.7 kernel patched with SKAS (Separate Kernel Address Space) for the physical machine and Linux with a 2.4.26 kernel for every VM.
- **VMware**: the host OS was Linux with a 2.4.28 kernel and the guest kernels were 2.4.28.
- **Vserver**: The host OS was Linux with a 2.4.27 kernel patched with linux-vserver-1.29 and util-vserver-0.30 to manage VM.
- **Xen**: The host OS was Linux with a 2.6.9 kernel patched for Xen the guest OS ware running 2.6.9 kernels patched for every VM.

These settings correspond to those recommended by the virtualization tools developers. To improve fairness we tried Vserver with the kernel 2.6. However, this version is known to be unstable and we were not able to run all the tests correctly with this version. For VMware, Linux 2.4 kernel allows launching more VMs (50) than with kernel 2.6 because of memory occupation.

4.2 Measurement Protocol

The following summarizes the measurement protocol:

- Time measurements related to CPU and memory use 'gettimeofday' (microsecond precision)
- Time measurements related to disk use the value returned by 'dd'
- Time measurements are done inside the host.
- For network measurements, we use Netperf which returns a bandwidth value from a fixed timing
- When many applications need to be launched simultaneously, we use 'rsh' in parallel to launch them on the VMs. The time between the first launch and the last one is insignificant compared to the application execution time
- For performance isolation tests, we launch three executions of the same application simultaneously on two VMs. We measure the performance for every application
- Before every set of tests, the machines are rebooted (except for the startup measurement)

5 Scalability Evaluation of Four Virtualization Systems

We presents the usability results in the first part and the performance comparison in the second part.

5.1 Usability

We start presenting the startup time and then present memory and disk occupation for the four virtualization tools. Note that the startup time could also be considered as a performance criterion, since some Grid applications will require the dynamic instantiation of virtual machines.

5.1.1 *Concurrent Startup Time*

We observed that the VM initialization time can be significantly shortened if they have already been launched in a session (already in memory). So we present two sets of results: cold startup (VMs are not in memory) and hot startup when VMs are already resident in memory.

For Xen, we restricted the number of VMs due to a limitation in the number of IRQ manageable by the machine.

Figure 2 demonstrates that Vserver outperforms Xen and UML. In these systems, each VM launches its own kernel. Vserver uses the physical machine kernel for all VMs. For Vserver as well as for UML, hot startup is about one order of magnitude faster than cold startup. Xen does not take advantage of Hot start up. Vserver is much more dynamic that the other virtualization tools. If the virtualization use case requires fast VM creation, then Vserver demonstrates a dramatic advantage compared to other virtualization tools.

	Cold start up	Hot start up
Vserver (100 VMs)	460s	30s
UML (100 VMs)	1040s	160s
Xen (50 VMs)	860s	860s
VMware (50 VMs)	1400s	810s

Figure 2 Cold and hot concurrent startup time for the four virtualization tools.

To our knowledge, no previous paper has presented such scalability result highlighting a significant cost of virtualization tools based on system virtualization (paravirtualization).

5.1.2 *Memory Occupation*

Memory occupation has not been measured by an experiment. Xen, UML and VMware run one kernel for every virtual machine. Since typical kernel configurations occupy between 2 MB and 5 MB of memory, obviously, from 200 MB to 1/2 GB of memory is required to run 100 VMs. This is a significant difference compared to Vserver, which only needs a single kernel instance whatever the number of VMs is. Note that, in [2], the authors present OS configuration obtaining a minimum memory footprint of 6.4 MB for each Xen domain. There is also an important difference in memory management between the four virtualization tools. Vserver VMs use and share the physical memory of the host machine. VMs are not confined in memory to some specific limit. In UML and Xen, VMs use the memory reserved for them.

5.2 Performance

In this part we present and comment result related to the virtualization overhead, linearity and performance isolation for the four resources of the physical machine.

5.2.1 *Overhead*

Figure 3 presents the CPU overhead (execution time of the computational microbenchmark). As explained in Section 3, we compare the execution time of a single application run on the host with

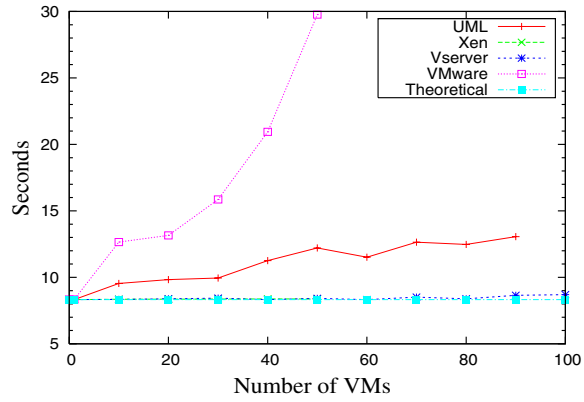

Figure 3 CPU overhead according to the number of VMs.

the execution of the same application running on a VM while the number of VMs is increasing from 1 to 100. The theoretical value of the overhead corresponds to the one when only a single VM is running.

We can observe that when one VM is running, the virtualization overhead is negligible compared to the execution on the host OS for all virtualization tools. When the number of concurrent VMs is increasing, VMware and UML exhibit strong deviation from the theoretical curve. This deviation is almost linear for UML and quadratic or exponential for VMware. We have observed the same phenomena for VMware GSX. Xen and Vserver exhibit near optimal scalability concerning the overhead parameter (curves like theoretical).

In Figure 4 we present the memory overhead of increasing the number of concurrently running VMs. The application is still run in one VM.

The behavior of Xen and Vserver is nearly perfect keeping the same negligible overhead whatever is the number of running VMs. VMware and UML behaviors exhibit a large deviation from the optimal. UML's memory overhead is erratic while VMware's overhead increases strongly with the number of VMs.

Figure 5 (left) presents the disk overhead. We still measure the execution time of the application while increasing the number of VMs from 1 to 100. We use different kernels for the four virtualization systems (2.6 and 2.4) and disks performances are not the same function of which kernel is running. Consequently we give one theoretical curve by virtualization system.

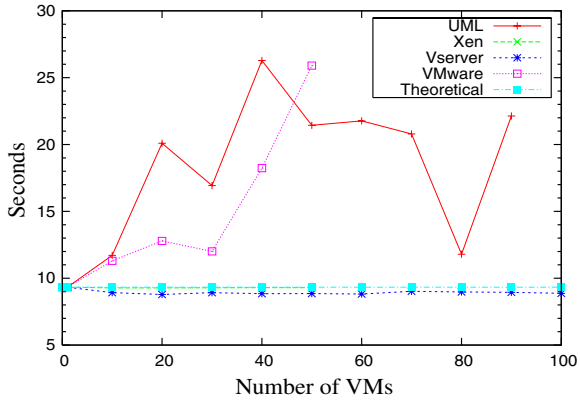

Figure 4 Memory overhead according to the number of VMs.

The first observation is that the different virtualization technologies have different disk overheads. UML exhibits the lowest performance (by a factor of three) and has the highest overhead, increasing linearly with the number of VMs. VMware has a disk overhead as much as 40% higher than the one of Vserver. Unfortunately, as explained in the Section 4, we were not able to run more than 50 VMs. The overhead increases sharply from 40 to 50 VMs. Xen has a higher overhead compared to Vserver at 10 VMs. However it does not increase with the number of VMs in contrary to the one of Vserser. The disk overhead of Vserver increases linearly of 60% for 100 VMs.

Figure 5 (right) presents the network overhead. For this test we measure application bandwidth reduction when using from 1 to 100 VMs compared to the host alone. The bandwidth is nearly optimal for Xen, UML and Vserver, albeit UML exhibits some minor deviations until 80 VMs and a 20% bandwidth degradation for 90 VMs. VMware has a stronger performance degradation, increasing with the number of VMs from 20% for 10 VMs to about 80% for 50 VMs.

An analysis of the virtual machine approaches in UML and VMware lead us to suspect a high scheduling overhead as the reason behind their high overhead on the four resources virtualization. To confirm this hypothesis, we have instrumented the Linux kernel in order to measure the scheduler activity. When the machine is idle (the idle process is scheduled), the scheduler is activated between 4 and 8 times per second. When a single UML or VMware VM is running (VM being idle too), the host scheduler is called nearly 400 times per second. As a consequence, the host scheduler looses processor cycles to schedule idle VMs. Vserver does not yield any overhead because of its implementation. Actually, creating a new inactive VM leads to a new process 'context,' not any scheduled process, even an idle one, being forked. Thus, schedule time measurement is not suitable for this architecture.

5.2.2 *Linearity*

In contrary with the overhead measurement, for linearity we scale the number of VMs executing the same application. For a normal linearity the

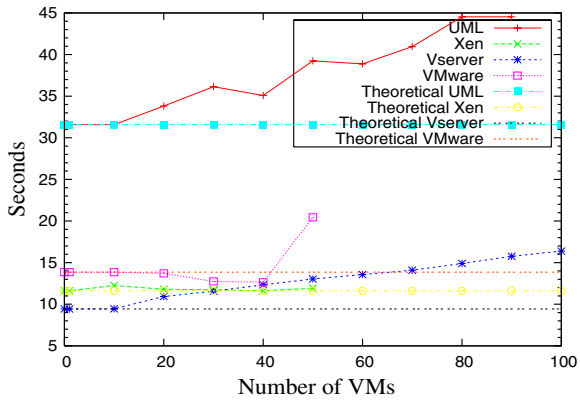

Figure 5 Disk overhead according to the number of VMs.

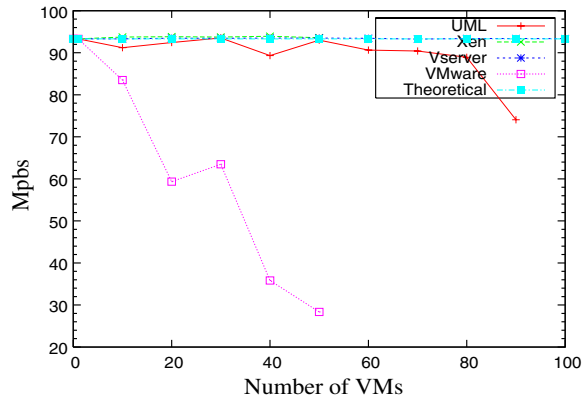

Figure 6 Network overhead according to the number of VMs.

execution time should increase linearly as a function of the number of concurrent running VMs. All presented results are means of the execution time measured on all running VMs. We also give the standard deviation to check how the resource is allocated to the concurrent VMs. Since the behaviors of the different virtualization tools for CPU, Memory and network are similar for one VM, we use only one theoretical curve as reference: the one considering Vserver as reference. For disk linearity, we show a theoretical curve for every virtualization tool.

Figure 7 presents the CPU linearity according to the number of concurrent VMs running the application. CPU linearity of Vserver and Xen is perfect from 1 to 100 VMs, Xen being faster compared to the theoretical reference based on Vserver. VMware and UML exhibit a poor linearity.

Figure 8 presents the memory linearity according to the number of concurrent VMs running the application. We observe the memory linearity of Vserver and Xen is perfect. VMware exhibits a poor linearity and UML results have a strong standard deviation.

The overhead of UML and VMware 5.2.1 explains the bad results of this two virtualization tools for the CPU and memory linearity benchmark.

Figure 9 presents the disk linearity. As already observed for the disk overhead, the four virtualization tools exhibit different behaviors concerning the disk. So we also plotted the theoretical curve for all tools. Curves are plotted on a logarithmic scale, so deviations are large. VMware, UML and Vserver deviate from their theoretical curve, which is not the case of Xen.

Figure 10 presents the network linearity. Curves are plotted on a logarithmic scale, so deviations are large. Xen and Vserve exhibit a rather optimal linearity. Unfortunately, we were not able to get meaningful experiment results with VMware and UML concerning the network scalability using Netperf. We suspect some timing and coordination weaknesses leading to inaccurate measurement. We envisage adapting the Netperf code in order to cope with this problem. *The final version of the article will present the measurements with the adapted Netperf version.*

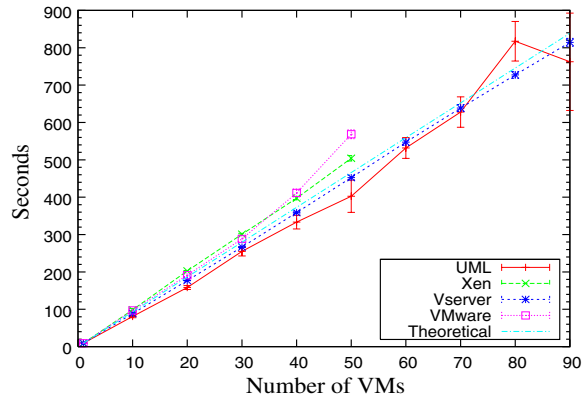

Figure 8 Memory linearity according to the number of VMs.

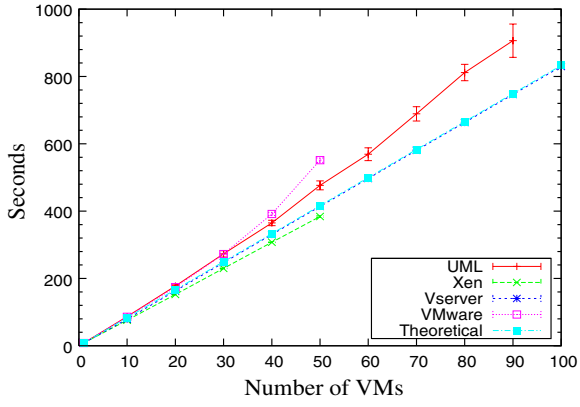

Figure 7 CPU linearity according to the number of VMs.

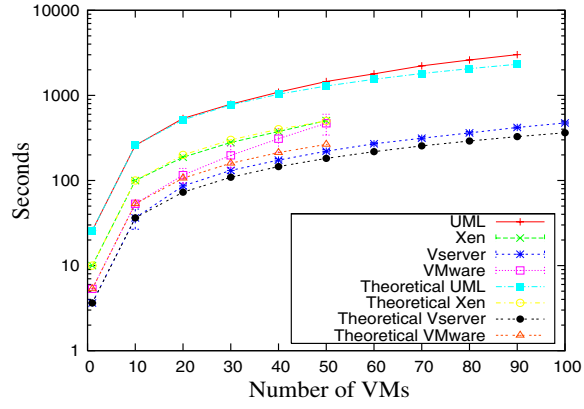

Figure 9 Disk linearity according to the number of VMs.

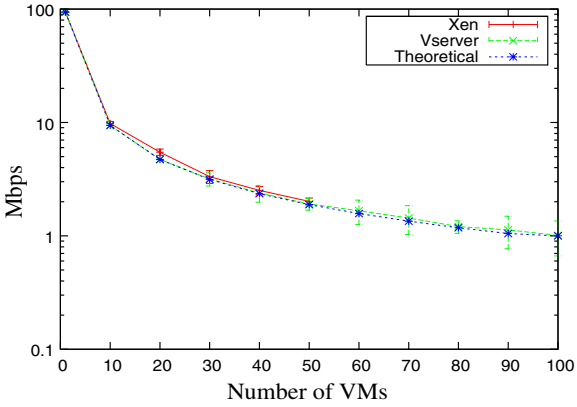

Figure 10 Network linearity according to the number of VMs.

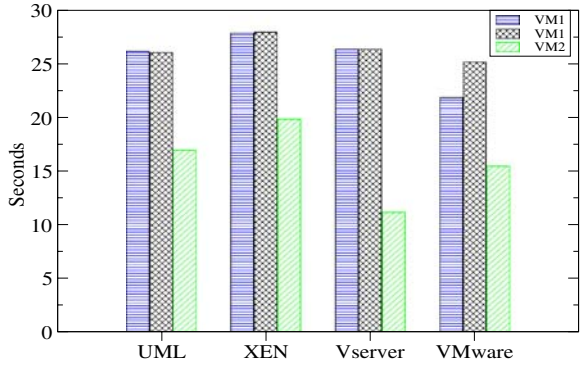

Figure 12 Memory performance isolation.

5.2.3 *Performance Isolation*

Figures 11 and 12 present the Performance Isolation (PI) of the four virtualization tools for the simple scenario where two VMs are running three microbenchmarks (two on one VM and one on the other VM). First we observe, concerning the absolute performance of the four virtualization tools, 1) similar performance on CPU, memory and network resources and 2) high performance differences on the disk resource. Vserver and Xen provide the best performance, Vserver outperforming Xen on the disk benchmark. Second, we observe that PI is between VMs (similar execution times between two instances of the benchmark running on the two VMs and a shorter execution time for the third instance executed on a single VM) except for the network, where all virtualization tools, but VMware, exhibit no PI between VMs. In order to explain the behavior of VMware, we have investigated several hypotheses: a) PI implemented in the virtual network device within VMs and b) PI implemented in the VMware orchestrator (VM monitor) using packet drops. However, the experiments conducted to validate these hypotheses did not confirmed them. We suspect that the VM monitor explicitly implements fare share mechanism ensuring the PI between VMs.

These results explain the behaviors of the four virtualization tools in our very first test (HTTP GET and HTTPS GET) used as a motivating example in Section 3. The results in Figures 11, 12, 13, and 14 clearly show that VMware demonstrates a full PI between VMs, which is consistent with our first test. We can also observe that UML, despite its poor network PI, exhibits PI for the

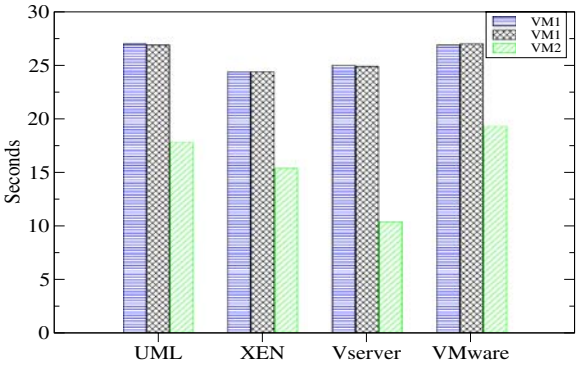

Figure 11 CPU performance isolation.

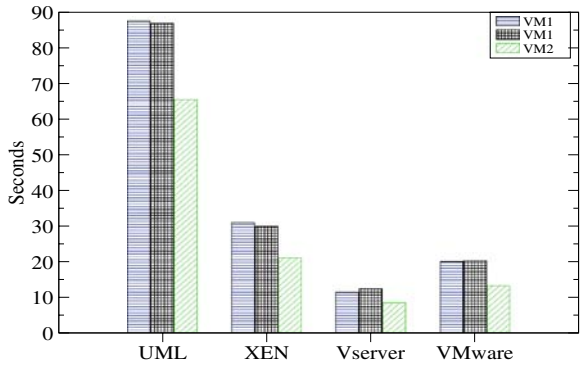

Figure 13 Disk performance isolation.

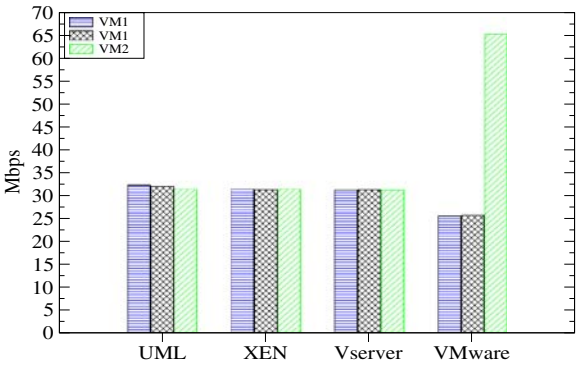

Figure 14 Network performance isolation.

other resources. However, since 1) the disk is the main solicited resource in the HTTP GET tests (the network uses 1 Gbps Ethernet connection between machines) and 2) UML shows a very high overhead on the Disk resource, the disk becomes the dominant factor of the execution time. This explains why UML appears to provide PI between VMs on the HTTP GET tests. Xen and Vserver do not exhibit PI between VMs on the HTTP GET tests. This behavior could be explained by 1) they have good performance on the disk making the network the dominant factor in the execution time of the HTTP GET tests and 2) they demonstrate no PI between VMs on the network.

5.2.4 *Communication Between VMs Inside a Single Host*

Figure 15 presents the communication performance for 10 concurrent VMs executing the Netperf microbenchmark.

These results clearly demonstrate a large performance difference between the virtualization tools. Surprisingly Xen performance is far lower than the one of Vserver. The only explanation we have about this performance degradation is related to the architecture of Vserver and Xen.

UML	Vserver	Xen	VMware
18 MB/s	543 MB/s	93 MB/s	105MB/s

Figure 15 Inter-VM communication performance when 10 VMs are executing Netperf inside a host.

In Vserver, all the VMs are running as processes of the same kernel and inter-process communication is crossing the host loopback device. In Xen, communications should traverse a specific device connecting the kernels and called a bridge. Thus the communications are not managed at the same level in both virtualization tools. UML adds another layer of overhead, still requiring kernel to kernel communications through a specific device but these kernels are run in user mode.

6 Conclusion

The evaluation of machine virtualization tools is a difficult exercise. We first motivated the use of microbenchmark to better understand the scalability limitations and merits of virtualization tools. We have described a set of metrics (overhead, linearity and isolation), and related microbenchmarks for the CPU, memory, disk and network resources. These metrics allow testing many aspects of these systems, performance as well as usability. We have compared four virtualization tools using this methodology: VMware, UML, Vserver and Xen. We clearly noticed strong limitations with VMware and UML, as previously published by other authors, but we have provided a detailed evaluation, identifying overhead, linearity and performance isolation limitations for all machine resources. Vserver and Xen clearly provide the better performance. However, there is still room for improvements in Vserver and Xen, since they do not provide performance isolation for the network between VMs (which is desirable for some users) and Xen suffers from low inter-virtual machine communication performance. A significant limitation of Vserver is that it cannot run kernels for guest virtual machines different to the hosting one. But, compared to Xen, its architecture saves a lot of memory space when running many virtual machines, since only one kernel is shared by all VMs. VMware has the advantage of providing performance isolation for all resources. It also allows running unmodified guest OS at the cost of a high overhead and poor linearity with respect to scalability.

According to their current respective merits and limitations, the compared VM technologies will match efficiently different application scenarios depending on their need in guest OS configuration, performance isolation and scalability. VMware clearly fits scenarios requiring small number of VMs and performance isolation between VMs. Such scenarios are likely to occur for Grid servers ensuring QoS from service level agreement (SLA). In addition, VMware accepts dynamic instantiation of user defined runtime environment, including specialized OS. Xen will match scenarios where many users or applications are deployed, possibly on ported OSes, on the same hardware with a 'best effort' or opportunistic like QoS (the performance of every VM will depend on the workload of the others). Vserver will accommodate more VMs and provide high performance communication between the VMs. But the application should be compliant with the VM hosting kernel. Vserver will match for example scenarios where the number of physical nodes running a distributed application with a fixed number of processes may evolve from time to time. UML is the only one which can be runned by an unprivileged user.

Another feature we can consider for choosing a virtualization tool is the ease of deployment of the virtualization package. Indeed VMware for example can run on both Windows and Linux; it is easy to install it and the graphic configuration interface is very useful. These functionalities are important for a user who wants to quickly test a virtualization tool.

Altogether we believe that 1) the result of this study will help users to select the VM technologies corresponding to the characteristics of their application and 2) the proposed metrics and benchmarks could help the VM designers by evaluating their technology from a third point of view (close to user needs), between real applications and low level VM mechanisms benchmarks.

For future work, it could be interesting to study the virtualization with other architectures such as bi or quadri processors (SMP) and dual or multicore processors. Indeed these architecture are increasingly used in clusters. A comparison with the recent hardware virtualization (Virtualization Technology for Intel processors and Pacifica for AMD processors) could also be consider.

References

1. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., Zhu, X.: From virtualized resources to virtual computing Grids: the in-vigo system. In: ELSEVIER (ed.) Future Generation Computer Systems (2004)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–177. SOSP, ACM Press, New York (2003)
3. Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., Wawrzoniak, M.: Operating systems support for planetary-scale network services. In: First Symposium on Network Systems Design and Implementation, NSDI '04, San Francisco, California, March 2004
4. Bruno, J., Brustoloni, J., Gabber, E., Ozden, B., Silberschatz, A.: Retrofitting quality of service into a time-sharing operating system. In: Proceedings of the USENIX Annual Technical Conference, Monterey, California, June 1999
5. Childs, S., Coghlan, B., OCallaghan, D., Quigley, G., Walsh, J.: A single-computer Grid gateway using virtual machines. In: Proceedings of AINA '05: The IEEE 19th International Conference on Advanced Information Networking and Applications, Taiwan. IEEE Computer Society, March 2005
6. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A case for Grid computing on virtual machines. In: Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDS, Providence, Rhode Island. IEEE Computer Society, 2003
7. Gelinas, J.: Virtual private servers and security contexts. http://www.solucorp.qc.ca/miscprj/s_context.hc, 2003
8. Grid Explorer project. http://www.lri.fr/~{fci/GdX}
9. Hoxer, H.J., Buchacker, K., Sieh, V.: Implementing a user mode linux with minimal changes from original kernel. In: Proceedings of 9th International Linux System Technology Conference, Cologne, Germany, September 2002
10. Jiang, X., Xu, D.: Soda: a service-on-demand architecture for application service hosting utility platforms. In: Proceedings of The 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), Seattle, Washington, June 2003
11. Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B., Figueiredo, R.J.: Vmplants: Providing and managing virtual machine execution environments for Grid

computing. In: Supercomputing '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing (CDROM), Pittsburgh, Pennsylvania. IEEE Computer Society, 2004
12. NAREGI project. http://www.naregi.org/index_e.html/
13. Singh, A.: An introduction to virtualization. http://www.kernelthread.com/publications/virtualization/. March 2004
14. Smith, J., Ravi, N.: In: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann, San Francisco, California (2005)
15. Song, H.J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., Chien, A.: The microGrid: a scientific tool for modeling computational Grids. In: Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), Dallas, Texas, p. 53. IEEE Computer Society (2000)
16. Sundararaj, A., Dinda, P.: Towards virtual networks for virtual machine Grid computing. In: Proceedings of the third USENIX Virtual Machine Research and Technology Symposium (VM 04), San Jose, California, May 2004
17. Waldspurger, C.A.: Memory resource management in VMware ESX server. SIGOPS Oper. Syst. Rev. **36**(SI), 181–194 (2002)
18. Whitaker, A., Shaw, M., Gribble, S.D.: Lightweight virtual machines for distributed and networked applications. Technical Report 02-02-01, University of Washington, Seattle, Washington, 2002
19. Whitaker, A., Shaw, M., Gribble, S.D.: Scale and performance in the denali isolation kernel. In: Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI), Boston, Massachusetts, December 2002
20. Whitaker, A., Shaw, M., Gribble, S.D.: Scale and performance in the denali isolation kernel. In: OSDI, Boston, Massachusetts, 2002
21. Xu, D., Jiang, X.: Collapsar: a vm-based architecture for network attack detention center. In: Proceedings of the 13th USENIX Security Symposium (Security '04), San Diego, California, August 2004
22. Zhao, M., Zhang, J., Figueiredo, R.J.: Distributed file system support for virtual machines in Grid computing. In: Proceedings HPDC13: The Thirteenth IEEE International Symposium on High-Performance Distributed Computing, Honolulu, Hawaii, June 2004