

Selecting A Virtualization System For Grid/P2P Large Scale Emulation

Benjamin Quétier, Vincent Neri, Franck Cappello
INRIA/LRI, Université Paris-Sud, Orsay, France

E-mail: { quetier, neri, fci }@lri.fr

phone: (+33) 1 69 15 4232

fax: (+33) 1 69 15 4213

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. Emulators and experimental tools like Microgrid and PlanetLab use multiple virtual nodes on every platform nodes. Thus, to build a large scale emulator for Grid/P2P experiments, developers have to select a virtualization tool among the available ones.

In this paper, we first present a set of metrics and related methodologies for performances and for usability evaluation. We use these metrics to compare 4 virtual machine technologies (Vserver, Xen, UML and VMware), in the context of large scale emulation.

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

I. INTRODUCTION

A. Virtualization

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. There are several approaches at the hardware and the OS levels. As presented in [1], 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).

Machine virtualization currently plays a significant role in Grid and P2P research platforms. In PlanetLab [2], machine

virtualization is used to confine the experiment of every user in a dedicated domain, referred to as a slice. In MicroGrid [3], virtualization allows Globus applications to run without modification. The Grid eXplorer project (a large scale distributed system emulator) [4] is relying on VM technologies to build a distributed system emulator scaling to 100,000 virtual nodes running over 1000 physical nodes.

Large scale emulation of distributed systems immediately raises the issues of 1) machine virtualization scalability and 2) Network characteristics. In this paper, we only focus on the first issue. Although there are several previous works on QoS for VMs (including research on fair-share [5]) and specific scalability evaluation [6], to the best of our knowledge there are surprisingly only a few published results on comparing the scalability of VMs. 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 II 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 paragraph we present a short survey of system virtualization techniques. Section II motivates and describes the metrics we propose to evaluate the merits of VM tools. Section III 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 IV. In the conclusion, we summarize what we have learned in this study.

B. Survey of existing virtualization tools

In this article, we focus on four virtualization tools representative of four main virtualization techniques :

- VMware [7] which provides a full processor virtualization
- UML [8] with SKAS patch for the kernel replication technique
- Xen [9], [10] and [11] for the paravirtualization
- Vserver [12] based on Security Context

II. METRICS AND METHODOLOGIES

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.

A. 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 tools (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.

A.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_{avn} 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, $O_{vn} = T_{avn} - T_a$ gives the virtualization overhead for n VMs.

A.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: $t_{max} = O_v + t * n$, where O_v is the virtualization overhead for 1 VM.

A.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 and equal access to the machine resources. 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 1 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 2 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 $T_{a2} = 3T_a/2$.

A.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.

B. Usability

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

B.1 Startup time

Startup time is an important characteristic of a virtualization tool, especially in a system where there are frequent re-configurations. 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 $T_{startup} = \max(startup[1, n])$. This metric can be affected by different services launched during the boot sequence (like a ssh server).

B.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_{VMi} the memory used by VMi and M_{tool} the memory (overhead) used by the virtualization system, then: $M_{used(n)} = \text{Sum}(M_{VM}[1, n]) + M_{tool}$.

III. EXPERIMENTAL SPECTRUM AND CONDITIONS

In this section, we present the experimental conditions used to evaluate the merits of UML, VMware, Vserver and Xen. We tested and compared 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 view 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 3000 experiment results.

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 licence.

A. Experimental conditions

To test the four virtualization systems, we have used PC's with AMD Opteron processors (1 per PC) running at 1.8GHz

with 1024Kb of cache, 3 GB of main memory, Ultra ATA hard disk of 80GB and BroadCom TG3 Ethernet interface. Machines are connected together 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 were 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. For Xen, we restricted the number of VMs due to a limitation in the number of IRQ manageable by the machine.

B. 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"
- 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 3 executions of the same application simultaneously on 2 VMs. We measure the performance for every application
- Before every set of tests, the machines are rebooted (except for the startup measurement)

IV. SCALABILITY EVALUATION OF FOUR VIRTUALIZATION SYSTEMS

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

A. Usability

We start presenting the startup time and then present memory and disk occupation for the 4 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.

A.1 Concurrent startup time

We observed that the VM initialization time can be significantly shortened if they have already been launched in a session

	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

Fig. 1. Cold and hot concurrent startup time for the 4 virtualization tools

(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.

Table 1 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 1 order of magnitude faster than cold startup. Xen does not take advantage of Hot start up. Vserver is much more dynamic than the other virtualization tools. If the virtualization use case requires fast VM creation, then Vserver demonstrates a dramatic advantage compared to other 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).

A.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 2MB and 5MB of memory, obviously, from 200MB 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 [9], 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 4 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.

B. 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.

B.1 Overhead

Figure 2 presents the CPU overhead (execution time of the computational microbenchmark). As explained in the Section II, we compare the execution time of a single application run on the host with 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 1 VM is running, the virtualization overhead is negligible compared to the execution on the host

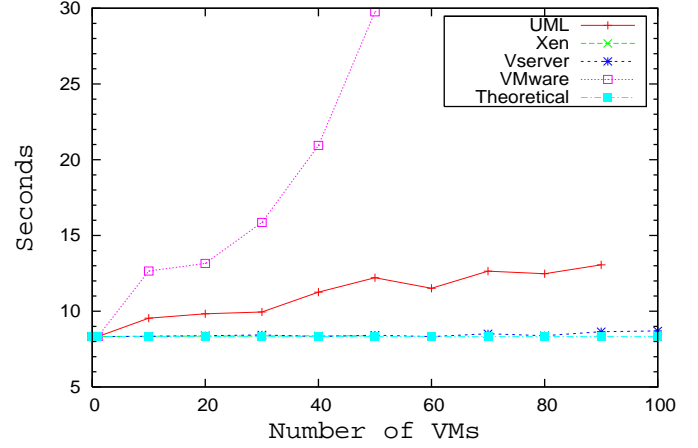


Fig. 2. CPU overhead according to the number of VMs.

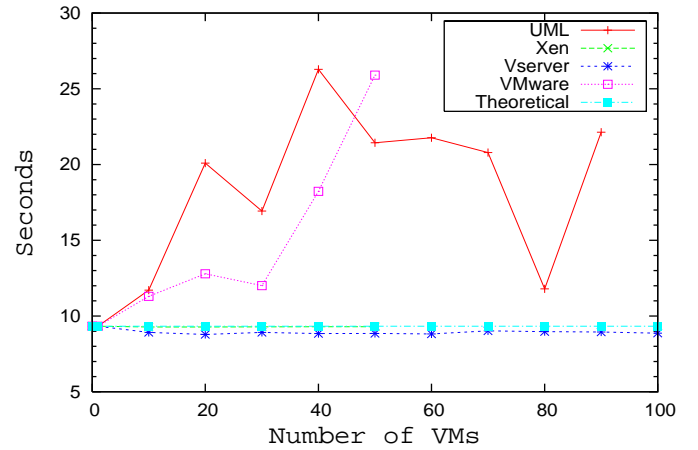


Fig. 3. Memory overhead according to the number of VMs.

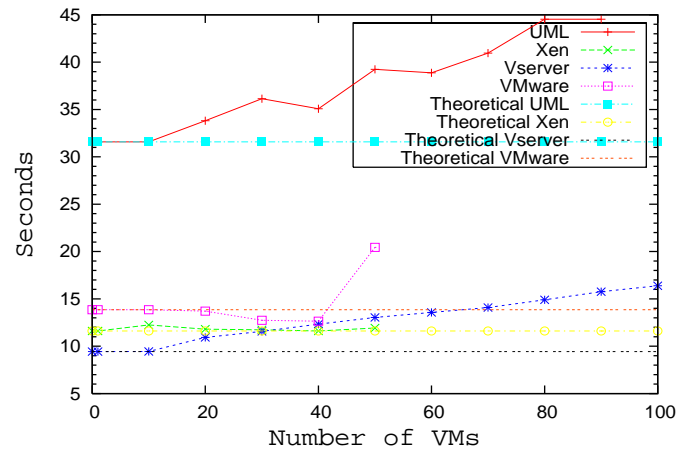


Fig. 4. Disk overhead according to the number of VMs.

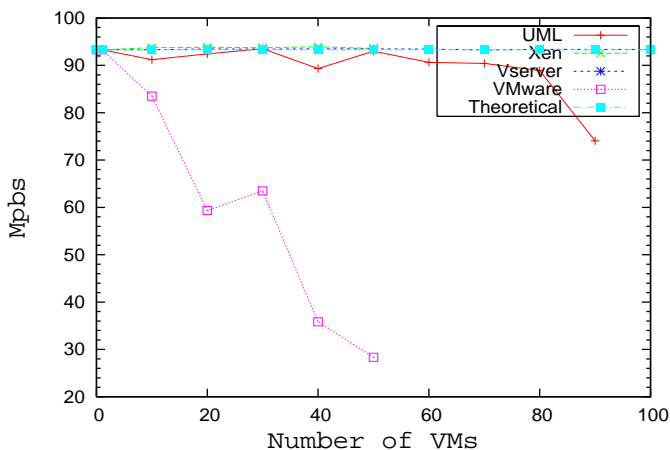


Fig. 5. Network overhead according to the number of VMs.

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 3 we present the memory overhead of increasing the number of concurrently running VMs. The application is still run in 1 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 4 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 4 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.

The first observation is that the different virtualization technologies have different disk overheads. UML exhibits the lowest performance (by a factor of 3) 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 III, 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 10VMs. However it does not increase with the number of VMs in contrary to the one of Vserver. The disk overhead of Vserver increases linearly of 60% for 100 VMs.

Figure 5 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 mi-

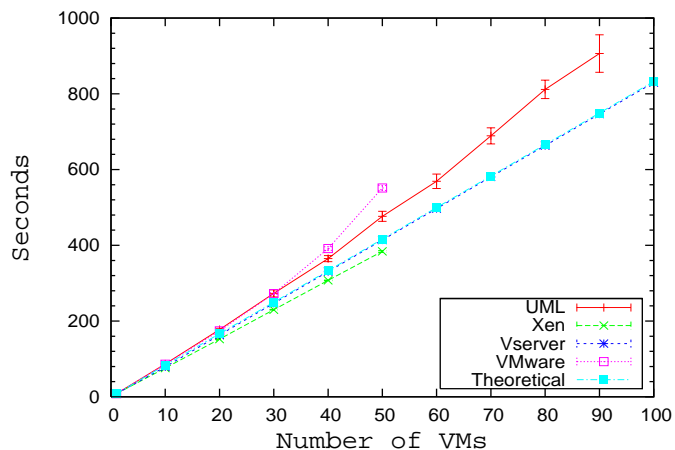


Fig. 6. CPU linearity according to the number of VMs.

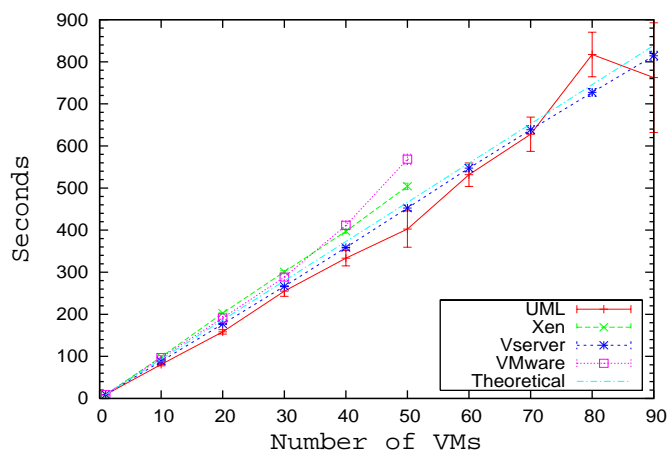


Fig. 7. Memory linearity according to the number of VMs.

nor 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 loses 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.

B.2 Linearity

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

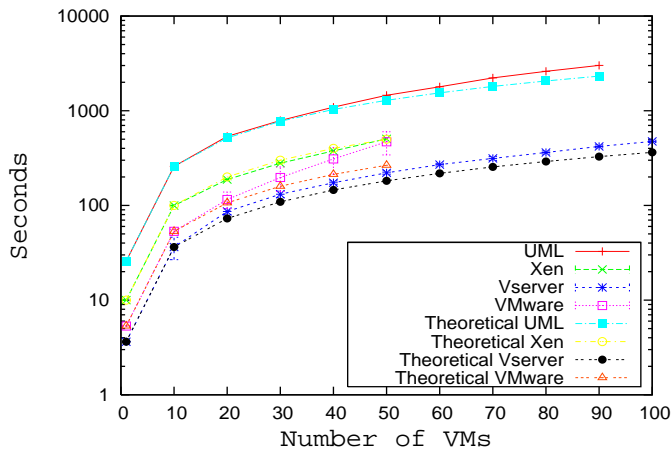


Fig. 8. Disk linearity according to the number of VMs.

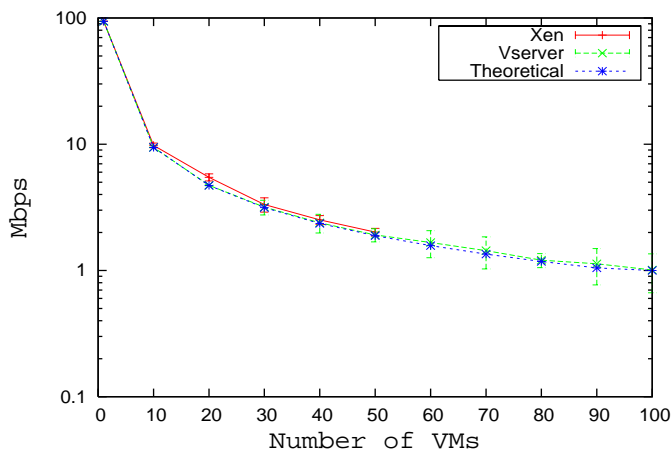


Fig. 9. Network linearity according to the number of VMs.

a normal linearity the 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 1 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 6 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 7 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 exhibit a poor linearity and UML results have a strong standard deviation.

The overhead of UML and VMware B.1 explain the bad results of this two virtualization tools for the CPU and memory

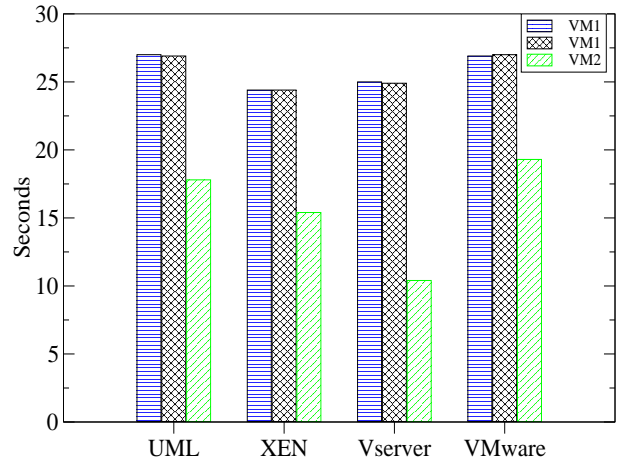


Fig. 10. CPU performance isolation.

linearity benchmark.

Figure 8 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 9 presents the network linearity. Curves are plotted on a logarithmic scale, so deviations are large. Xen and Vserver 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 envision adapting the Netperf code in order to cope with this problem.

B.3 Performance Isolation

Figures 10, 11, 12, 13 present the Performance Isolation (PI) of the 4 virtualization tools for the simple scenario where 2 VMs are running 3 microbenchmarks (2 on 1 VM and 1 on the other VM). First we observe, concerning the absolute performance of the 4 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 2 instances of the benchmark running on the 2 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 experi-

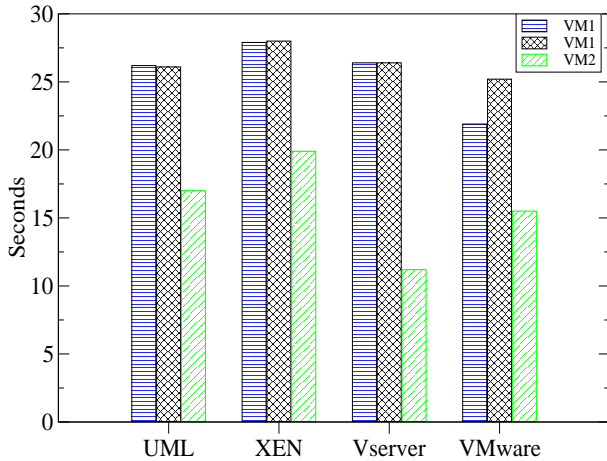


Fig. 11. Memory performance isolation.

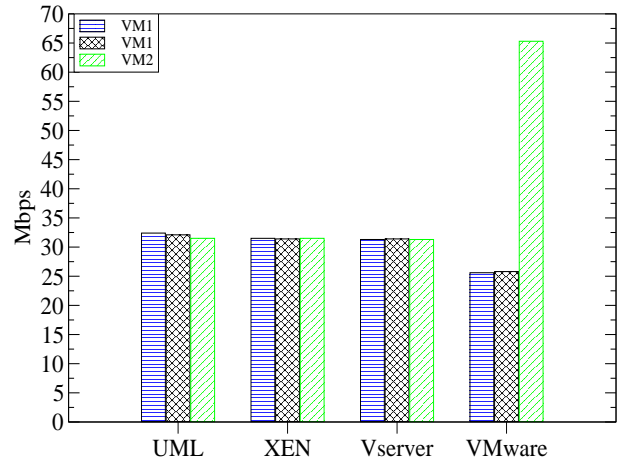


Fig. 13. Network performance isolation.

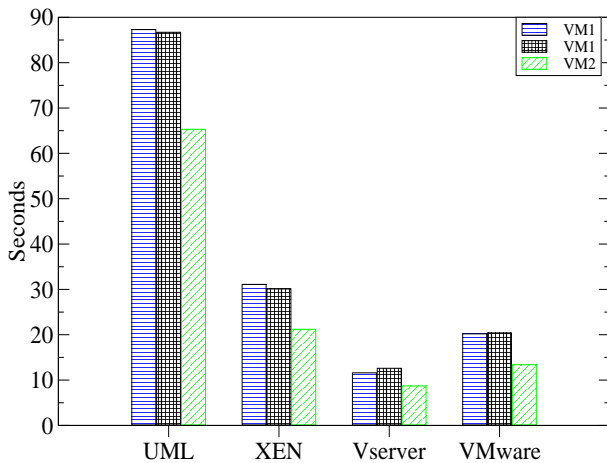


Fig. 12. Disk performance isolation.

ments 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.

B.4 Communication between VMs inside a single host

Table 14 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. 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

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

Fig. 14. Inter-VM communication performance when 10 VMs are executing Netperf inside a host

adds another layer of overhead, still requiring kernel to kernel communications through a specific device but these kernels are run in user mode.

V. CONCLUSION

The evaluation of machine virtualization tools is a difficult exercise. 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 used these microbenchmarks and metrics to better understand the scalability limitations and merits of virtualization tools. Thus, we have compared 4 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.

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 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.

Altogether we believe that 1) the result of this study will help users to select the VM technologies corresponding to the char-

acteristics of their application and 2) the proposed metrics and benchmarks could help the VM designers by evaluating their technology with a third point of view (close to user needs), between real applications and low level VM mechanisms benchmarks.

REFERENCES

- [1] Amit Singh. An introduction to virtualization. In <http://www.kernelthread.com/publications/virtualization/>, March 2004.
- [2] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating systems support for planetary-scale network services. In *First Symposium on Network Systems Design and Implementation, NSDI'04*, Mar 2004.
- [3] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 53. IEEE Computer Society, 2000.
- [4] Grid Explorer project. In <http://www.lri.fr/fci/GdX>.
- [5] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Retrofitting quality of service into a time-sharing operating system. In *Proceedings of the USENIX 1999 Annual Technical Conference*, June, 1999.
- [6] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the denali isolation kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.
- [7] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.
- [8] Hans-Jorg Hoxer, Kerstin Buchacker, and Volkmar Sieh. Implementing a user mode linux with minimal changes from original kernel. In *Proceedings of 9th International Linux System Technology Conference*, Cologne, Germany, September 2002.
- [9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM Press, 2003.
- [10] A. Whitaker, M. Shaw, and S. D. Gribble. Lightweight virtual machines for distributed and networked applications. Technical Report 02-02-01, University of Washington, 2002.
- [11] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the denali isolation kernel. In *OSDI*, 2002.
- [12] J. Gelinas. Virtual private servers and security contexts. In http://www.solucorp.qc.ca/miscprj/s_context.hc, 2003.