IL NUOVO CIMENTO Vol. 32 C, N. 2 Marzo-Aprile 2009
DOI 10.1393/ncc/i2009-10391-x

Colloquia: CSFI 2008

# Measuring performances of linux hypervisors

A. Chierici, R. Veraldi and D. Salomoni
*INFN-CNAF - Bologna, Italy*

(ricevuto il 4 Agosto 2009; pubblicato online il 4 Settembre 2009)

**Summary.** — Virtualization is a now proven software technology that is rapidly transforming the IT landscape and fundamentally changing the way people make computations and implement services. Recently, all major software producers (*e.g.*, Microsoft and RedHat) developed or acquired virtualization technologies. Our institute (`http://www.cnaf.infn.it`) is a Tier1 for experiments carried on at the Large Hadron Collider at CERN (`http://lhc.web.cern.ch/lhc/`) and is experiencing several benefits from virtualization technologies, like improving fault tolerance, providing efficient hardware resource usage and increasing security. Currently, the virtualization solution we adopted is xen, which is well supported by the Scientific Linux distribution, widely used by the High-Energy Physics (HEP) community. Since Scientific Linux is based on RedHat ES, we felt the need to investigate performances and usability differences with the new kvm technology, recently acquired by RedHat. The case study of this work is the Tier2 site for the LHCb experiment hosted at our institute; all major grid elements for this Tier2 run on xen virtual machines smoothly. We will investigate the impact on performance and stability that a migration to kvm would entail on the Tier2 site, as well as the effort required by a system administrator to deploy the migration.

PACS `01.50.hv` – Computer software and software reviews.
PACS `07.05.-t` – Computers in experimental physics.
PACS `07.05.Bx` – Computer systems: hardware, operating systems, computer languages, and utilities.
PACS `89.20.Ff` – Computer science and technology.

## 1. – Introduction

Infrastructure virtualization, and in particular server virtualization has become very important nowadays because of the tremendous benefits achieved by using it. Virtualization is a technology that allows running a certain number of different and concurrent operating system instances inside a single physical machine. This physical server is divided into multiple isolated virtual environments called guests.

## 2. – Virtualization approaches

There are three popular approaches to server virtualization: full virtualization, paravirtualization and virtualization with hardware support (Hardware Virtual Machine, or HVM).

**2**˙1. *Full virtualization.* – Virtual machines are based on the host/guest paradigm, where each guest runs on a virtual representation of the hardware layer. This approach allows the guest operating system to run without modifications. It also allows the administrator to create guests that use different operating systems. The guest has no knowledge about the host operating system because it is not aware that it is not running on real hardware. This approach does require, however, real computing resources from the host, so an hypervisor to coordinate instructions to the real CPUs is used. The hypervisor is called a virtual machine monitor (VMM), and validates all the guest-issued CPU instructions, managing any executed code that requires additional privileges. VMware and Microsoft Virtual Server both use the full virtualization approach.

**2**˙2. *Para-virtualization.* – The para-virtualized machine approach (PVM) is also based on the host/guest paradigm and uses a virtual machine monitor as well. In this model, however, the VMM actually modifies the guest operating system's code; this modification is called porting. Porting supports the VMM so it can access privileged systems calls sparingly. Like fully virtual machines, para-virtual machines are capable of running multiple operating systems. Xen and UML both use the para-virtual machine model.

**2**˙3. *Hardware virtual machine (HVM).* – Recent innovations in hardware, particularly in CPU, MMU and memory components (notably the Intel VT-x and AMD-V architectures), provide some direct platform-level architectural support for OS virtualization.

HVM offers two key features: first, for unmodified guests, it avoids the need to trap and emulate privileged instructions by enabling guests to run at their native privilege levels; second, it offers para-virtualized guests the VM CALL instruction that calls directly into the hypervisor. This can be used by certain drivers to ensure that guest I/O takes the fastest path into the hypervisors I/O stack. This means that it is possible to recompile para-virtualized drivers inside the guest machine running in HVM environment and load those drivers into the running kernel to achieve para-virtualized I/O performances for an HVM guest.

## 3. – Xen-based virtualization

Xen [1, 2] is a virtual machine monitor that allows several guest operating systems to be executed on the same computer hardware concurrently. A Xen system is structured with the Xen hypervisor as the lowest and most privileged layer. Above this layer one or more guest operating systems are located, which the hypervisor schedules across the physical CPUs. Xen can work both in para-virtualized and HVM mode; in the former the guest operating system must be modified to be executed. Through para-virtualization, Xen can achieve very high performances. The HVM mode offers new instructions to support direct calls by a para-virtualized guest/driver into the hypervisor, typically used for I/O or other so-called hypercalls.

## 4. – KVM-based virtualization

KVM [3] is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, which provides the core virtualization infrastructure, and a processor specific module, kvm-intel.ko or kvm-amd.ko. KVM requires a modified version of qemu, a well-known virtualization software. The kernel component of KVM is included in mainline Linux as of kernel version 2.6.20, while for xen only external support is available. KVM supports I/O para-virtualization using the so called VIRTIO subsystem, consisting of 5 kernel modules.

**4·1.** *KVM in our centre.* – KVM can easily be installed using a yum repository [4]; once installed, the first thing to do is to set up networking. For our virtual machines we currently use public IP addressing configured via a bridged network, so we modified the default kvm start-up script to create a bridge for each network interface; in this way we can choose at boot time which virtual machine can be hooked to which network interface. After this, we need to add a tap interface and assign it to the proper bridge: also in this case we created a special script (/etc/qemu-ifup) that is executed automatically upon guest creation and destruction so that tap interfaces are managed transparently.

## 5. – KVM: qualitative test

We introduced kvm into our environment seamlessly. We are using Quattor [5] as the main installation and configuration tool; Quattor is based on machine profiles describing the configuration designed by system administrators for every machine. Introducing kvm did not require any additional effort or modification to this infrastructure to have every VM to be installed via network boot exactly like xen VMs. We stepped into one little problem involving the "-boot" option, which requires only one parameter, be it network, hard disk or cdrom. After the machine has been installed via network boot, we have to stop and restart it with the option to boot from hard disk; xen does not need this workaround, since the hard disk is the second default boot device.

The qualitative test consisted in installing on a kvm VM an "EGEE/gLite Computing Element" [6] used in production in the LHCb [7] Tier2 we are hosting at CNAF. Previously, the Tier2 CEs were both running on xen VMs, so this gave us the opportunity to make a direct comparison between the two technologies. The VM worked flawlessly for more than 3 weeks not giving users or system administrators any idea of the change that was made: we considered the test fully passed.

While we were doing these tests we decided to install also a CMS [8] secondary squid server and we had the same feedback from the users: no differences in performance were noticed. As a reference, we used this hardware to host the two kvm VMs: 1 dual Intel E5420, 16GB ram, SATA disks on Areca controller.

## 6. – KVM: Quantitative test

Even if kvm passed brilliantly the qualitative test, we needed some quantitative measures to confirm the positive impression so far obtained. For this reason we performed a set of tests targeted at measuring the classic parameters of a machine (CPU, network and disk access), with tools well known to the HEP community. Here is the list of the tools used:

- CPU: hep-spec06 (v1.1) [9], PI computation test
- Network: iperf (v2.0.4) [10]
- Disk access: bonnie++ (v1.94) [11]

To best understand the performance of a kvm virtual machine, we measured the same parameters for a xen machine as well, with both para and hvm virtualization methods, and compared them with a non virtualized machine (the baseline).

**6**˙1. *Test specifications.* – The hardware used for the quantitative test was composed of 1 DELL blade server with a dual intel E5420, 16GB RAM and two 10k SAS disks connected to a LSI Logic RAID controlled (disks configured with RAID0 option). The VMs specs were:

- Xen-para VM: 1 vcpu, 2 GB RAM, disk on a file and on lvm partition
- Xen-hvm VM: 1 vcpu, 2GB RAM, "netfront" network driver, disk on a file and on lvm partition
- KVM VM: 1 vcpu, 2GB RAM, e1000 network driver emulation, disk on a file and lvm partition
- KVM virtio: 1vcpu, 2GB RAM, virtio network driver, virtio disk (on a file and lvm partition)

As for the OS installed:

- Host OS: Scienfific Linux (SL) [12] 5.2 x86_64, kernel 2.6.18-92.1.22.el5
- VM OS: Scientific Linux CERN (SLC) [13] 4.5 i386, kernel 2.6.9-67.0.15.EL.cern
- KVM virtio OS: Scientific Linux CERN (SLC) [13] 5.3 x86_64, kernel 2.6.18-128.1.10.el5

Version of the hypervisors used:

- KVM: 83
- Xen: 3.2.1

**6**˙2. *Benchmarks: HEP-Spec.* – HEP-Spec [9] is the new standard in HEP community for CPU performance benchmarking, and it is based on a subset of the Spec benchmark. We performed a wide number of tests in order to quantify the differences in performance for the various virtualization solutions, compared to a non virtualized CPU.

In fig. 1 we show the performance comparison of the various hypervisors while an increasing number of virtual machines run concurrently on the host. We tested the performance with 7 and 8 concurrent runs in order to understand if the hypervisor requires a dedicated CPU (1 hypervisor+7 VMs is the exact number of cores on our box).

The results of the test shown in fig. 2 confirm that we can "overload" the cores of our machine and run 8 VMs concurrently, without any significant performance loss.

Figure 3 shows a comparison between the HEP-Spec06 benchmark on 8 VMs and the same benchmark run on physical CPUs, as calculated by the HEPiX community [14]. Again, as we can see, the performance loss is so little that we can consider it negligible; we believe in fact that the benefits obtained by virtualization just overcome this minimal performance loss. Table I shows the exact percentage loss compared to physical CPUs. With virtualization technologies, either xen or kvm, the loss is comparable to a CPU downgrade on one industry clock tick (*i.e.* to the immediately slower CPU model).

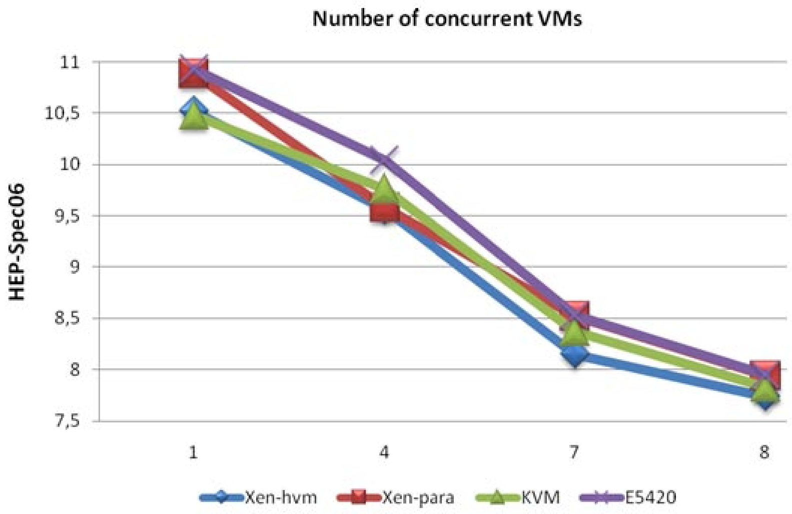

Fig. 1. – XEN *vs.* KVM on dual Intel E5420, single performance measure.

**6**˙3. *Benchmarks: PI test*. – The goal of this test is to measure the CPU performance when multiple virtual machines are in execution, focusing on the CPU distribution against the virtual machines. Each virtual machine is assigned one virtual CPU. We want to see if the total amount of cores is distributed using a fair sharing mechanism or if certain guest machines are using more CPU than others.

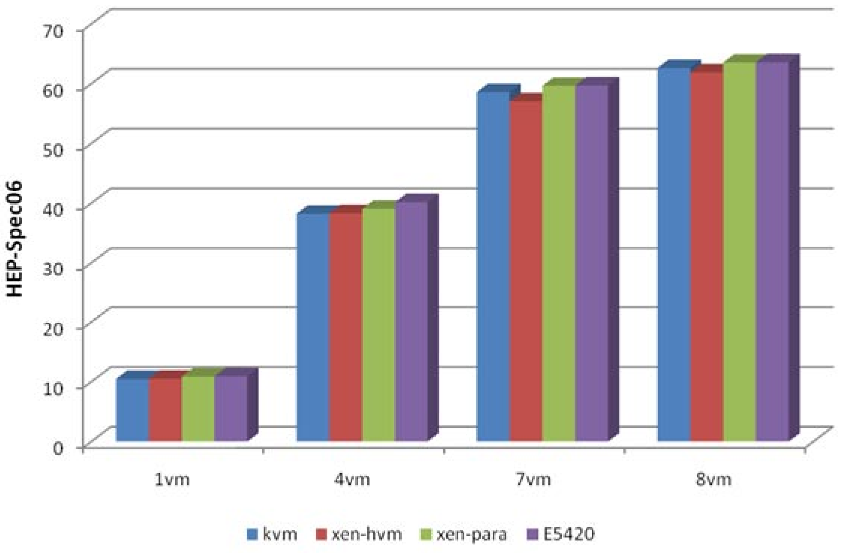

Fig. 2. – VMs *vs.* CPU.

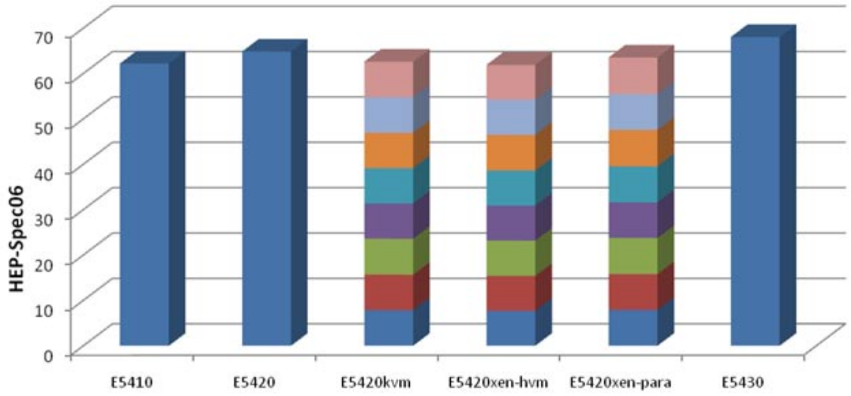

Fig. 3. – 8 VMs aggregate *vs.* CPUs.

To achieve this we used a home-made tool to calculate CPU cycles consumed by each virtual machine (see app. A for the code). We used a arbitrary digits PI calculator with code based on Numerical Recipes [15]. We then used an external C program implementing 2 functions: *start_counter()* right before the call to the PI() function, and *get_counter()* called after PI() function exits. The counter returns the number of CPU cycles used during the PI() function call and execution. We used the RDTSC assembly function which loads the current value of the processor's time-stamp counter into the EDX:EAX registers. The time-stamp counter is contained in a 64-bit MSR. The high-order 32 bits of the MSR are loaded into the EDX register, and the low-order 32 bits are loaded into the EAX register. The processor increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset. We first executed our CPU test on the host and on a single KVM guest machine (see fig. 4): and the result is that there is a difference of about 2% between them.

We then executed the test with 8 KVM guests running at the same (fig. 5) time so that all the 8 cores of our hardware were used. The result is that every guest uses 100% of one single core, the CPU performance is well balanced and it is comparable to the host CPU performance (with a different of around 2%).

In the last test we wanted to use several KVM guests. So we repeated the CPU test with 17 KVM guests running the PI job (fig. 6). Again, the result is that the use of the CPU is well balanced, with an average difference of around 5%.

**6**˙4. *Benchmarks: Iperf*. – Network performance is an essential parameter to measure: with iperf we tested the throughput performances both in inbound and outbound

Table I. – *HEP-Spec % loss.*

| Virtualization technology | % loss from non-Emulated CPU (E5420, 8vm) |
|---|---|
| E5420kvm | 3.42 |
| E5420xen-hvm | 4.55 |
| E5420xen-para | 2.02 |
| E5410 *vs.* E5420 | 4.07 |

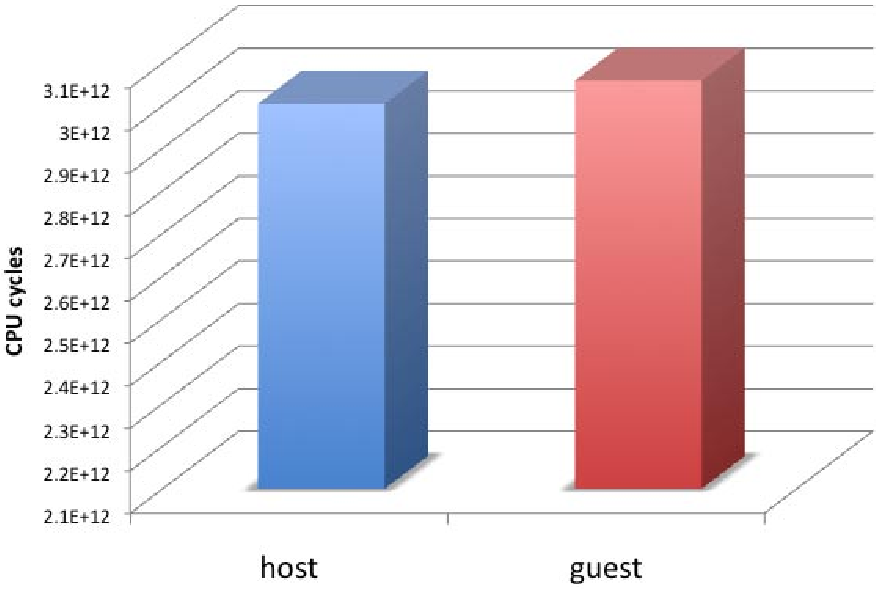

Fig. 4. – CPU comparison between host and KVM guest.

directions. The options we used to get the results are: "-w256k -P 5 -t 900", meansing a TCP window size of 256k, 5 parallel connections and a duration of 900 seconds (15 minutes).

As we can see from fig. 7, the behavior of the network is asymmetric with inbound performing better than outbound connection. The situation improves with the increasing number of the nodes, suggesting some sort of limitations hardcoded inside the driver. The situation is better with virtio drivers, where kvm performs almost like xen (fig. 8).

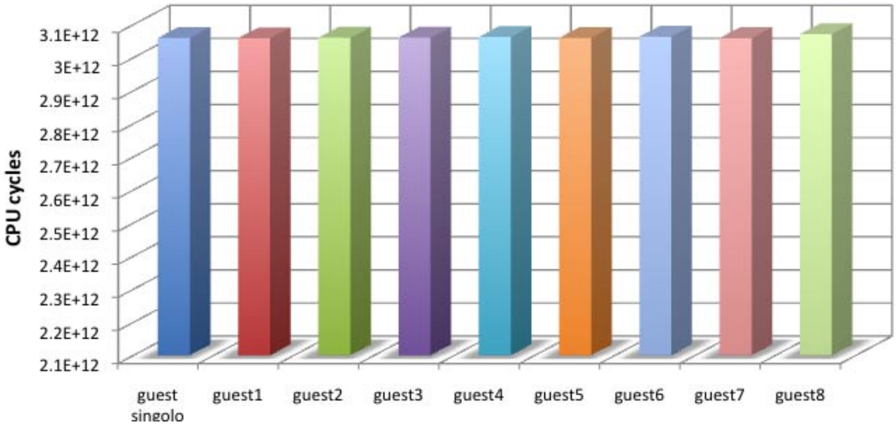

Fig. 5. – CPU comparison between host and 8 concurrent KVM guests.

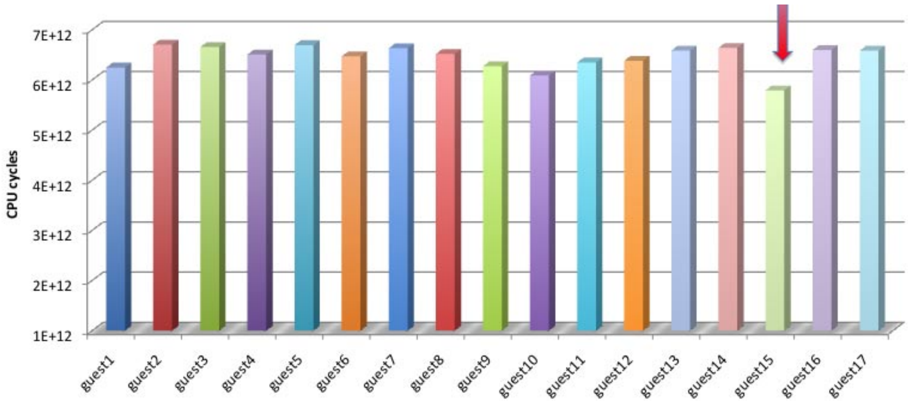

Fig. 6. – CPU comparison between multiple running KVM guests.

Generally speaking, we can say that xen is still behaving better, particularly when kvm is used without the virtio drivers (for example in a machine currently running the EGEE middleware, currently still requiring sl4), and proving to be a better choice in situations where network outbound speed is critical.

**6˙5.** *Benchmarks: bonnie++.* – Disk access speed is critical for every virtualization technology and our tests showed that a lot is still to be improved. In fig. 9 indeed we can see the read/write speed performances for a real machine compared to a virtual one (either xen or kvm).

In fig. 10 we can see a comparison of kvm performances using both standard and virtio drivers, on a lvm partition. The performance does not improve moving to virtio, getting actually even worse, and suggesting that virtio drivers still need to be improved a lot.

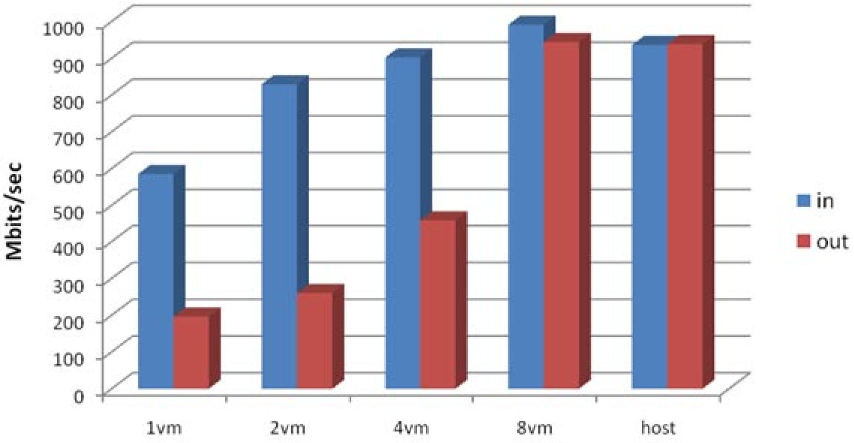

Fig. 7. – KVM Network Performance.

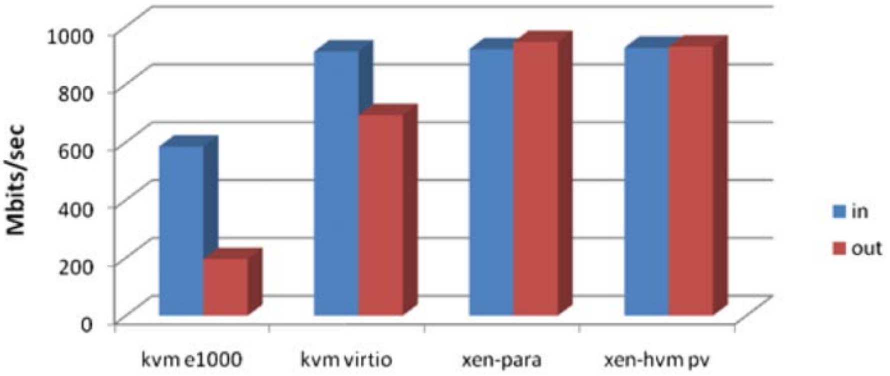

Fig. 8. – KVM virtio Network Performance compared to xen.

Figure 11 depicts performance comparison between disk-on-a-file and an lvm partition. As we can see, performances of the lvm partition are generally better, even if not to the degree we were expecting. What disappointed us were the writing performances compared to a non emulated machine: in some cases the virtualized solution got just 10% of the non-virtualized performance. The Xen-para solution is for sure the better solution here, but when we move to fully virtualized approaches (either with xen-hvm and kvm) performances are almost the same. Another disappointing result showed by the tests is the general performance of concurrent access of VMs to the physical disk. In these circumstances (figs. 10 and 11) all the solutions perform very badly compared to the real machine, with xen-para taking the lead, even if only for read access. All the solutions perform very badly in disk writing. There is probably quite some room for improvements in future releases of the hypervisors in this area.

Finally, fig. 12 shows a global performance comparison between all the virtualization approaches using both disk-on-a-file and lvm partition when running 8 concurrent VMs.

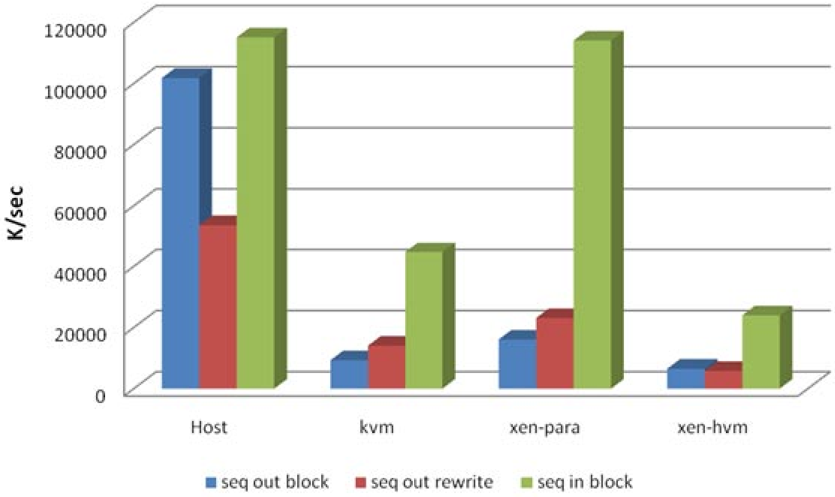

Fig. 9. – 2GB RAM, 4GB data set, 1vm comparison.

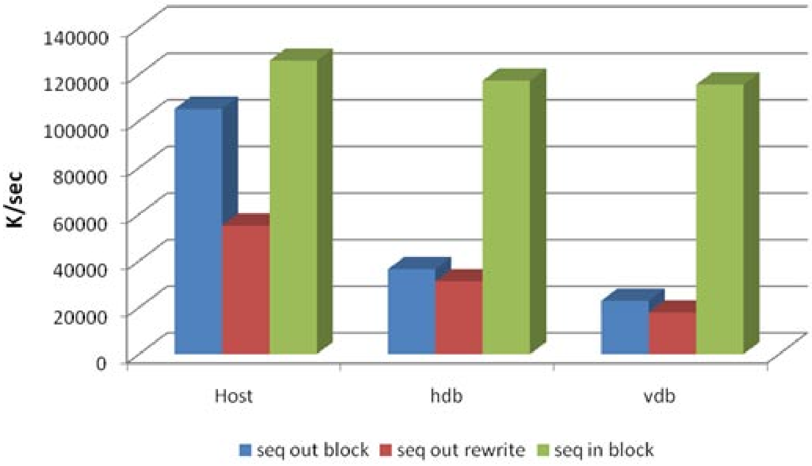

Fig. 10. – 2GB RAM, 4GB data set, 1vm comparison, standard *vs.* virtio driver.

In this case I/O performances are really bad, with each VMs getting just a tiny bit of the performance the real node could achieve if not virtualized. Also in this case xen-para outperforms the other solutions.

The next step in hardware virtualization will hopefully be to implement special instructions in the hardware for I/O in a way similar to what has been done for the CPU; as we have shown, CPU-intensive task are already performing brilliantly.

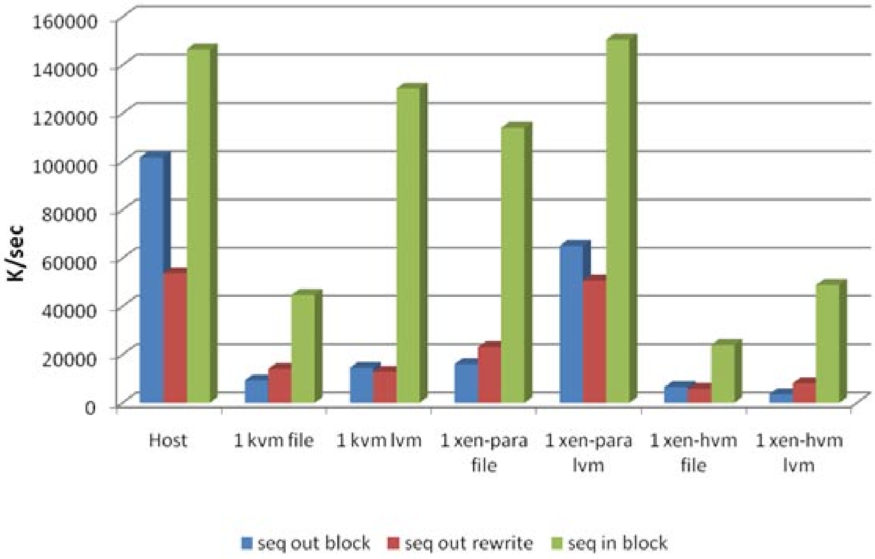

Fig. 11. – 2GB RAM, 4GB data set, 1vm, file *vs.* lvm partition.

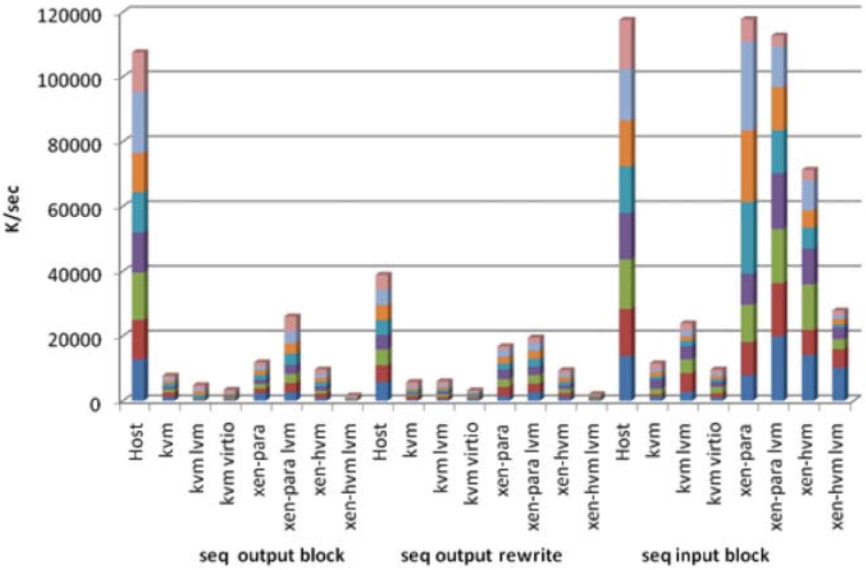

Fig. 12. – 2GB RAM, 4GB data set, 8vm, aggregate disk-on-a-file and lvm partition.

## 7. – Conclusions

During our tests kvm showed great stability and reliability: it never crashed and integrated seamlessly into our computing farm, without requiring any additional effort to system administrators. Our benchmarks showed that the CPU performance provided by the virtualization layer is comparable to the one provided by xen and, in some cases it is even better.

Network performances are fair, showing some strange asymmetric behavior when using standard network emulation; things inprove when using virtio drivers.

Disk I/O seems to be the most problematic aspect with KVM: VMs get poor performances, particularly when multiple machines concurrently access the disk. Both kvm- and xen-based VMs showed poor performances with this aspect, with either lvm partitions or the more common disk-on-a-file approach.

Compating xen and kvm performance-wise, the results show that, even if the kvm solution appears very promising in the medium term, xen hypervisors seem to be the best solution at the moment, particularly when using the para-virtualized approach.

## Appendix A.

### PI Code

```
#include <stdio.h>
#include <stdlib.h>

/* Initialize the cycle counter */
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;
```

```
/* Set *hi and *lo to the high and low order bits of the cycle counter.
Implementation requires assembly code to use the rdtsc instruction. */
void access_counter(unsigned *hi, unsigned *lo)
{
 asm("rdtsc; movl %%edx,%0; movl %%eax,%1" /* Read cycle counter */
 : "=r" (*hi), "=r" (*lo) /* and move results to */
 : /* No input */ /* the two outputs */
 : "%edx", "%eax");
}

/* Record the current value of the cycle counter. */
void start_counter()
{
  access_counter(&cyc_hi, &cyc_lo);
}

/* Return the number of cycles since the last call to start_counter. */
double get_counter()
{
 unsigned ncyc_hi, ncyc_lo;
 unsigned hi, lo, borrow;
 double result;

 /* Get cycle counter */
 access_counter(&ncyc_hi, &ncyc_lo);

 /* Do double precision subtraction */
 lo = ncyc_lo - cyc_lo;
 borrow = lo > ncyc_lo;
 hi = ncyc_hi - cyc_hi - borrow;
 result = (double) hi * (1 << 30) * 4 + lo;
 if (result < 0) {
   fprintf(stderr, "Error: counter returns neg value: %.0f\n", result);
 }
 return result;
}
```

## REFERENCES

[1] Xen Website: http://www.xen.org.
[2] Xen unofficial repository: http://www.gitco.de/repo.
[3] KVM Website: http://www.linux-kvm.org.
[4] KVM unofficial repository: http://www.lfarkas.org/linux/packages/centos.
[5] Quattor Website: http://ww.quattor.org.
[6] EGEE/gLite Website: http://glite.web.cern.ch/glite/.
[7] LHCb Website: http://lhcb.web.cern.ch/lhcb/.
[8] CMS Website: http://cms.web.cern.ch/cms/index.html.
[9] HEP-SPEC06 Website: https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiBenchHEPSPEC.
[10] Iperf Website: http://sourceforge.net/projects/iperf/.

[11] Bonnie++ Version 1.94: http://freshmeat.net/projects/bonnie/releases/283268.
[12] Scientific Linux Website: https://www.scientificlinux.org/.
[13] Scientific Linux CERN Website: http://linux.web.cern.ch/linux/.
[14] HEPiX Website: http://www.hepix.org.
[15] Numerical Recipes: *The Art of Scientific Computing*, third edition (Cambridge University Press) 2007.