Virtualization of Linux servers

a comparative study

Fernando Laudares Camargos
Revolution Linux
fernando.laudares@revolutionlinux.com

Gabriel Girard
Université de Sherbrooke
gabriel.girard@usherbrooke.ca

Benoit des Ligneris
Revolution Linux
benoit.des.ligneris@revolutionlinux.com

Abstract

Virtualization of x86 servers has been a hot topic in the last decade, culminating in changes in the architecture's design and the development of new technologies based in different approaches to provide server virtualization.

In this paper, we present a comparative study of six virtualization technologies released under the GPL license. Our analysis is done in two steps. First we evaluate the overhead imposed by the virtualization layers by executing a set of open source benchmarks in the Linux host system and inside the virtual machines. Secondly we analyze the scalability of those technologies by executing a benchmark suite concurrently in multiple virtual machines. Our findings may help users choose the technology that better suits their needs.

1 Introduction

Virtualization is not a new idea [17, 18]. What has changed is its practical purpose. In the begining, virtualization was used as a way of providing multiple access to mainframe systems by running multiple operating systems on the same machine concurrently [31, 30], as well as a safe environment for software development. At that time operating systems were designed for a single user, so the solution to provide access to several users was to run many operating systems in separate virtual machines.

Today, the actual operating systems can provide multiple access to the same machine. There is no more need for running multiple operating systems on the same machine—unless we want them for other reasons. Development is still one use for virtualization, but now the main focus has changed to other applications such as server virtualization. The idea behind server virtualization has always been to make a better use of the available resources—this is being achieved today through a technique called server consolidation. Studies have shown that the majority of data centers found in today's enterprises are organized around a silo-oriented architecture [24], in which each application has its own dedicated physical server. This may have been the right design in the past but today the computational resources of the average physical server exceeds the needs of most server applications, which means a share of those resources is being wasted, only around 15% of them being actually used on average [21, 25]. The solution to avoid this waste of physical resources is then to consolidate a group of those servers in one physical machine. Doing this by virtualizing the underlying infrastructure warrants a greater level of isolation between the servers as well as providing other advantages inherent to server virtualization besides server consolidation, which are covered by many other studies [12, 23, 32]. This concept is illustrated by Figure 1. In this scenario four under-utilized physical servers were consolidated in one.

The virtualization layer that allows the hosting of guest operating systems may be provided by different virtualization solutions. Each solution implements this layer in a different way. One negative side-effect of this model is that the existence of such a layer implies a possible overhead that can affect the performance of applications running inside a virtual machine[26]. Ideally this overhead is minimized.

The main objective of this study is to evaluate six virtualization solutions for Linux released under the GPL

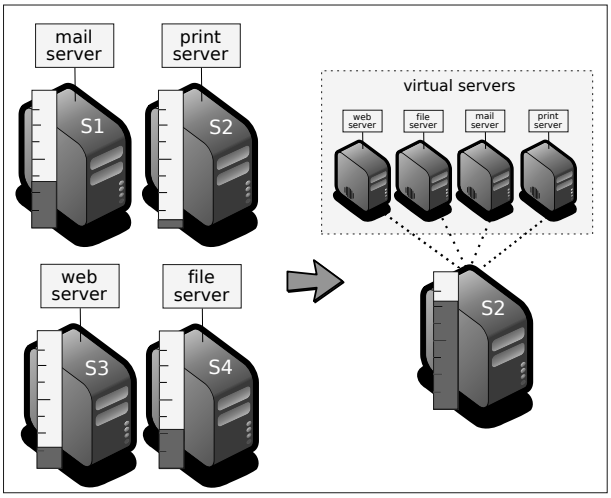

Figure 1: Virtualization as a way to provide server consolidation

license and measure their efficiency. We borrow the definition of *efficiency* from [33], which refers to it as being the combination of *performance* and *scalability*. The stated characteristics of virtualization solutions may lead us to make bad choices if we base ourselves solely on them. This study aims to be practical. We hope our findings may help users to choose the virtualization solutions that better suits their needs.

The rest of this article is structured as follows. Section 2 presents the main points that have motivated us to conduct this study. In Section 3 we present our way of categorizing the different virtualization solutions in groups of technologies that share similar characteristics. Section 4 presents the methodology used to conduct our evaluation. The obtained results are presented and discussed in Section 5. Finally, our conclusions and suggestions for future work are presented in Section 6.

2 Motivation

A number of studies evaluating different virtualization technologies have been published in the past. Table 1 presents some of them chronologically, indicating the year of publication as well as the respective virtualization technologies that were covered by each study. While those studies have been of great value for our understanding of this article's subject, none has covered the main open source virtualization solutions side-by-side and just a few of them were published by independent sources. The main reason for this lack of coverage is that such a work requires a great amount of time. Still we were intrigued by this possibility and we believed many others would be as well.

We have chosen to limit the scope of our study to open source technologies because of the friendly nature of the software user licence, which didn't prevent us publishing our findings nor required them to be scrutinized before a publication permission would eventually be issued. The access to the source code of the software also provides the freedom to adapt it as we need to, which proved to be useful during our experiments.

As users of some of the virtualization technologies that were subjected to our evaluation, our intention with this project was to gain a better view of how they relate to each other. We believe and hope that our feedback may contribute to their continuous development in a positive way.

We have not evaluated the features of the different virtualization technologies on an individual basis. The main characteristics of the majority of those technologies have already been presented in other studies [20, 33, 6].

3 Overview of virtualization technologies

Virtualization technologies differ in the way the virtualization layer is implemented. In 1974, Popek & Goldberg [26] published a paper that presented one way of doing it, which was later referred to as *classic virtualization* [1]. This paper became a reference in the subject and presented a series of requirements for a control program (which is known today as operating system, or *supervisor*) to work as a virtual machine monitor (VMM, also known today as *hypervisor*). Such requirements, organized as a set of properties, limited the construction of such VMMs to a specific machine generation or processor instruction set architecture (ISA) "*(. . .) in which sensitive instructions*[1] *are a subset of privileged instructions*." This characteristic allows the VMM to selectively trap only the privileged instructions issued inside the virtual machines by the guest operating systems and let the remaining instructions be executed directly by the processor. This procedure thus correlates with the stated efficiency property, which is directly associated to a low virtualization overhead.

[1] Instructions that may change the current state of the physical machine (IO, for instance).

Study	Year	Evaluated technologies	Version	Kernel version
[2]	2003	Xen	-	2.4.21
		VMware Workstation	3.2	2.4.21
		UML	-	2.4.21
		VMware ESX Server	-	2.4.21
[13]	2004	Xen	-	2.4.26
[27]	2005	Linux-VServer	1.29	2.4.27
		UML	-	2.4.26 / 2.6.7
		Xen	-	2.6.9
[5]	2006	Linux-VServer	1.29	2.4.27
		UML	-	2.4.26 / 2.6.7
		Xen	2.0	2.6.9
		VMware Workstation	3.2	2.4.28
		VMware GSX	-	-
[14]	2006	Linux-VServer	2.0.2rc9 / 2.1.0	2.6.15.4 / 2.6.14.4
		MCR	2.5.1	2.6.15
		OpenVZ	022stab064	2.6.8
		Xen	3.0.1	2.6.12.6
[1]	2006	VMware Player	1.0.1	-
[10]	2006	VMWare ESX Server	3.0	-
[35]	2007	VMware ESX Server	3.0.1 GA	-
		Xen	3.0.3-0	-
[36]	2007	VMware ESX Server	3.0.1	2.6.9
		XenEnterprise	3.2	2.6.9
[11]	2007	Xen	3.0.3 (unstable)	-
		OpenVZ	stable	2.6
[33]	2007	Linux-VServer	2.0.1	2.6.17
		Xen	3.0.2-testing	2.6.16
[16]	2007	Linux-VServer	2.2	2.6.20
[22]	2007	Xen	-	-
		KVM	24	-

Table 1: Summary of studies evaluating virtualization technologies published between 2003 and 2007

Even then the researchers were aware of the fact that not all architectures may be "virtualizable" this way. This realization is specially true for today's most popular architecture, commonly known as x86. The original design of the x86 architecture did not included virtualization [8]. A detailed account on this issue is reported in [28]. In summary, not all sensitive instructions in the x86's architecture are a subset of its privileged instructions. In practice, this prevents the implementation of a VMM capable of selectively trapping only the sensitive instructions executed by the guest operating systems. Instead, it would have to trap all instructions, incurring a considerable system overhead.

Other virtualization technologies have been developed to avoid this issue and implement alternative ways to virtualize the x86 architecture. Some of them rely on the complete or partial emulation of the underlying hardware [36] and provide what is called *full-virtualization*. *QEMU* [4] is a software emulator that emulates the entire hardware stack of a machine and is used as the base for various virtualization projects in this category, like *KQEMU* (an "accelerator" for QEMU that may be used to transform the latter in a virtualization solution [3]), *KVM* and *VirtualBox*. Those solutions are extremely flexible, meaning they can theoretically support any guest operating system developed for the x86 architecture, but are not among the most efficient ones due to the hardware emulation.

A second category contains virtualization solutions that implement a technique called *para-virtualization*. The most important consideration when running more than one operating system concurrently on the same machine is that this class of software is designed to control the machine exclusively. That is why the use of emulation

works so well—in this case, the guest operating system is still the only one controlling the machine, but the machine in question is not the physical machine, but a virtual machine. The key for para-virtualizing a system is to make the guest operating systems aware of the fact that they are being virtualized [12]—and ask them to collaborate. In exchange, the VMM provides an almost direct access to some of the physical resources of the machine. This approach provides an efficient virtualization technology but it is also an extremely invasive technique since it requires important modifications to the guest OS kernel structure. *Xen* is the most well known para-virtualization solution.

The third and last of our categories focus on the *virtualization at the operating system level*. The virtualization layer in this particular implementation is set above the operating system [23]. "Virtual machines" are soft partitions [15], or containers [14], that replicate the environment of the host operating system. This is in theory the most efficient kind of virtualization—yet the less flexible. The efficiency comes from the fact that there is only one kernel in execution at any time, and thus the absence of hypervisor overhead. The lack of flexibility comes from the same reason—one may even run different flavors of Linux, but they will share the same kernel. Linux-VServers and OpenVZ are two examples of OS-level virtualization solutions. Both are available as a patch that can be applied to the Linux kernel.

Recently, the x86 architecture received additional extensions [34, 37] that allows the implementation of a *classic* VMM that responds to Popek & Goldberg's criteria. Whether the use of those extensions will assure the development of more efficient virtualization solutions is still arguable [1]. Nonetheless, since version 2.6.20 the Linux kernel comes with KVM, a simple yet robust piece of software that uses those additional extensions to transform Linux into a hypervisor [29, 19]. More recent versions of Xen also use those extensions, as well as a little dose of emulation, to allow for the virtualization of "closed source" operating systems.

4 Methodology

There is no consensus in the scientific community nor in the industry about what would be the best way to evaluate virtualization solutions. The *Standard Performance Evaluation Corporation* (SPEC) created a committee at the end of 2006 that is studying this matter [7]. VMware and Intel, both members of the committee, stepped ahead and released two different benchmark suites (or workloads, since both suites are composed of already existent and independent benchmarks), named respectively *VMmark* [10] and *vConsolidate* [9]. The workloads that compose both suites are very similar, the difference is in the way each company combines the obtained results to define a scoring system. Neither of these benchmark suites have been considered in this study because they both rely on a benchmark for mail servers called *LoadSim*, which works only with Microsoft's Exchange Server.

The majority of the studies in Table 1 have used benchmarks that target different parts of the system. We have adopted a similar approach. Our analysis is done in two steps. In the first step we evaluate the overhead imposed by the virtualization layers by executing a set of open source benchmarks in the Linux host system and inside the virtual machines. In the second step, we analyze the scalability of those technologies by executing *SysBench*, one of the benchmarks used in the first step, concurrently in multiple virtual machines.

We decided to use Ubuntu 7.10 as the operating system for the host system, as it comes with kernel version 2.6.22-14, which is well supported among all the evaluated virtualization solutions. For the virtual machines we have chosen Ubuntu's Long Time Supported version, which at the time of this study was Ubuntu 6.10. This proved to be a bad decision for a comparative study and one of the major weaknesses of this work, since the majority of the OS utilities, like Rsync and Bzip2, have different versions in each release, not to mention the difference in kernel versions.

The name and version of the benchmarks used in our evaluation are shown in Table 2. Table 3 presents a list of the evaluated virtualization solutions, showing the kernel versions used in the host and guest operating systems.

The compilation of the results is done as follows. All experiments are repeated four times. The first sample was discarded, the presented results for each set of experiments being the median of the second, third, and fourth samples. The results of the experiments conducted inside the virtual machines are normalized using the results of the respective experiments conducted in the non-virtualized environment as a base. The evaluation was done in two steps. The first step focused on the overhead

Benchmark	Version (Host)	Version (VMs)	Unit of measure
Bzip2	1.0.3-0ubuntu2.1	1.0.4-0ubuntu2.1	time
Dbench	3.04	3.04	throughput
Dd (coreutils)	5.93-5ubuntu4	5.97-5.3ubuntu3	throughput
Kernel (build)	linux-2.6.22.14	linux-2.6.22.14	time
Netperf	2.4.4	2.4.4	throughput
Rsync	2.6.6-1ubuntu2.1	2.6.9-5ubuntu1	time
SysBench	0.4.8	0.4.8	throughput

Table 2: List of the benchmarks used in the first step of our evaluation and respective metrics

Virtualization solution	Version	Host Kernel	Guest Kernel
KQEMU	1.3.0 pre11-6	2.6.22.14-kqemu	2.6.15-26-amd64
KVM	58	2.6.22-14-server	2.6.15-26-amd64
Linux-VServer	2.2.0.5	2.6.22.14	2.6.22.14
OpenVZ	5.1	2.6.22-ovz005	2.6.22-ovz005
VirtualBox	1.5.4_OSE/1.5.51_OSE	2.6.22-14-server	2.6.22.14
Xen	3.1.0	2.6.22-14-xen	2.6.22-14-xen

Table 3: Evaluated virtualization solutions and the respective kernel versions

of a single virtual machine. In the second step the experiments are repeated concurrently in *n* virtual machines, with *n*=1,2,4,8,16, and 32.

The experiments were conducted using an IBM/Lenovo desktop configured with an Intel Core 2 Duo 6300 processor, 4G of RAM, an 80GB SATA hard drive and two gigabit network interfaces. The main network interface is connected to a LAN and the other is attached with a cross-over cable to a second machine, which was used as a client for the experiments involving a network. This second machine is a Dell Optiflex GX configured with an Intel Pentium 2 processor, 123M of RAM, a 250GB IDE hard drive and one gigabit interface.

The main machine was rebooted before the beginning of each new set of tests. Before moving on to test the next virtualization solution, the disk was re-formated and the operational system was re-installed from scratch.

In the first step of the evaluation, the virtual machines were configured with 2G of RAM. Table 4 shows the memory allocation used in the second step. The top limit was set to 2039M of RAM because this was the maximum amount supported by QEMU-based virtualization solutions during preliminary tests.

To run the scalability tests we have used Konsole's *Send input to all sessions* function to log in the *n* virtual machines simultaneously from a third machine, connected to the test machine through the LAN interface, and start the benchmark in all VMs simultaneously.

Number of VMs	Memory/VM (in M)
n=1	2039
n=2	1622
n=4	811
n=8	405
n=16	202
n=32	101

Table 4: Memory distribution used in the second step of the evaluation

Figure 2 presents the commands and respective parameters used to execute each one of the benchmarks.

5 Results and discussion

This section presents the results of our experiments. For all charts but the ones representing our scalability evaluations, the results for the different virtualization solutions have been normalized against the results when running without virtualization. Higher bars represent better

Kernel Build

```
$ make defconfig
$ date +%s.%N && make && date +%s.%N
$ make clean
```

Dbench

```
$ /usr/local/bin/dbench -t 300 -D /var/tmp 100
```

Netperf

```
$ netserver            # server side
$ netperf -H <server>  # client side
```

Rsync

```
Experiment 1:
$ date +%s.%N && rsync -av <server>::kernel /var/tmp && date +%s.%N  # client side
  # where 'kernel' is the linux-2.6.22.14 file tree (294M)
$ rm -fr /var/tmp/*
Experiment 2:
$ date +%s.%N && rsync -av <server>::iso /var/tmp && date +%s.%N     # client side
  # where 'iso' is ubuntu-6.06.1-server-i386.iso (433M)
$ rm -fr /var/tmp/*
```

Dd

```
Experiment 1:
$ dd if=/opt/iso/ubuntu-6.06.1-server-i386.iso of=/var/tmp/out.iso
$ rm -fr /var/tmp/*
Experiment 2:
$ dd if=/dev/zero of=/dev/null count=117187560  # 117187560 = 60G
```

Bzip2

```
$ cp /opt/ubuntu-6.06.1-server-i386.iso .
$ date +%s.%N && bzip2 -9 ubuntu-6.06.1-server-i386.iso && date +%s.%N
$ rm ubuntu-6.06.1-server-i386.iso.bz2
```

SysBench

```
$ mysql> create database sbtest;
$ sysbench --test=oltp --mysql-user=root --mysql-host=localhost --debug=off prepare
$ sysbench --test=oltp --mysql-user=root --mysql-host=localhost --debug=off run
```

Figure 2: Commands and parameters used to execute the benchmarks

performance of the virtualization solution for the respective workload.

VirtualBox is a virtualization solution that allows the user to decide whether or not it should use the virtualization extensions present in the processor. This fact lead us to perform all experiments with this software twice, with (`--hwvirtex on`) and without (`--hwvirtex off`) the use of such extensions. As previously mentioned, QEMU is the base for a considerable number of the virtualization solutions evaluated in this study. Since the virtual machine image used by KVM and KQEMU is also compatible with QEMU, we have also evaluated the latter in the first step of our practical study and included the results whenever we considered it to be appropriate. Our main reason for doing this is to show how

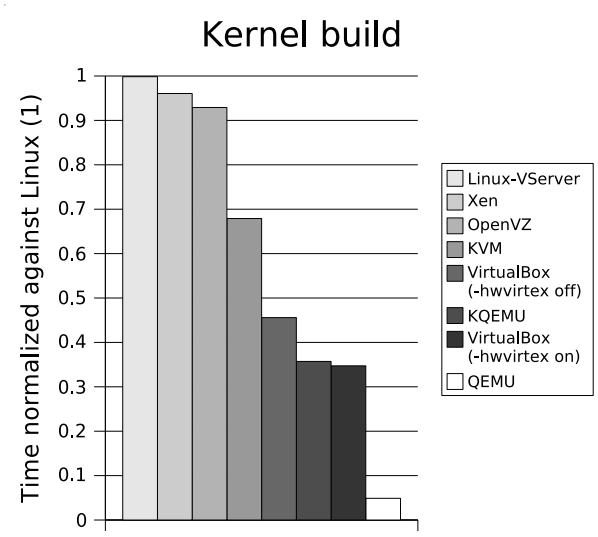

Figure 3: Relative performance of the virtualization solutions for the kernel compilation experiment.

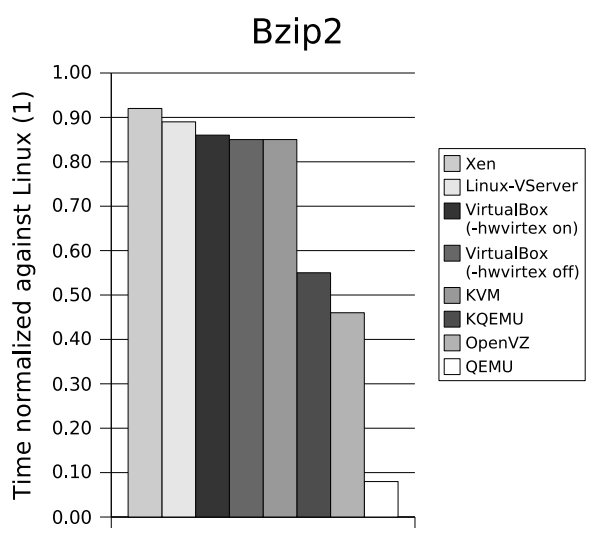

Figure 4: Evaluating the virtualization solutions with Bzip2

much a virtualization layer below QEMU (like KQEMU and KVM) can benefit the performance of the applications running inside it.

Figure 3 shows the results for the *kernel build* experiments. Kernel compilation is a CPU intensive task which involves multiple threads and stress the filesystem in both reading and writing small files. Those characteristics make it for a good overall system performance indication. As expected, virtualization solutions relying on both OS-level and para-virtualization technologies presented a performance close to Linux's. Among the full-virtualization solutions, KVM's performance is far superior.

This first graphic shows a unique situation in our study in which the non-use of the virtualization extensions by VirtualBox results in performance that is higher than when VirtualBox makes use of such extensions to accomplish the same task. In all the other experiments, such a difference in performance will be less significant.

The next experiment is a file compression using *Bzip2*. This is also a CPU intensive task, but with low I/O requirements. The `-9` option used for maximum compression also demands more memory for the process to execute the compression. For this experiment, we have all virtualization solutions performing close to Linux, except for KQEMU and OpenVZ, as shown in Figure 4. The low performance of OpenVZ was a surprise since we expected it to perform close to Linux for all experiments.

Figure 5 shows the results for the experiments with *Dbench*, a file system benchmark that simulates the load placed on a file server. Here, the sole virtualization solution to match Linux closely was Linux-Vserver, the remaining solutions showing performance less than 30% of Linux. This includes Xen, which has shown better performances in other studies [2, 33] for a similar workload. We were not able to successfully run this benchmark with VirtualBox without it crashing for an unknown reason.

The results for our experiments of disk performance done with *dd* are presented in Figure 6. These experiments do not stress the CPU but focus mainly on disk I/O. For the first experiment, which copies a 433M *iso* image file to the same ext3 partition, Linux-VServer presented performance that considerably surpasses that of Linux. VServer's modifications to the kernel clearly benefit this kind of task. Xen and KVM presented good performance while OpenVZ was significantly slower. In the second experiment, 60G of null characters are read from `/dev/zero` and written to a scratch device (`/dev/null`). Since the data is not actually written to the disk this experiment focuses on the I/O operations of the OS without physically stressing the disk. For this experiment, Xen shows a decrease in performance while OpenVZ performs a lot better than in the first experiment, but still shows a considerable overhead when compared to Vserver and KVM. We were unable

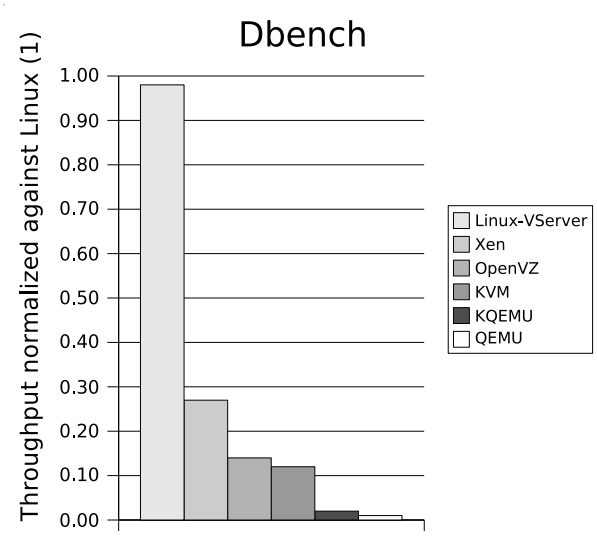

Figure 5: Evaluating the virtualization technologies with Dbench, a file system benchmark

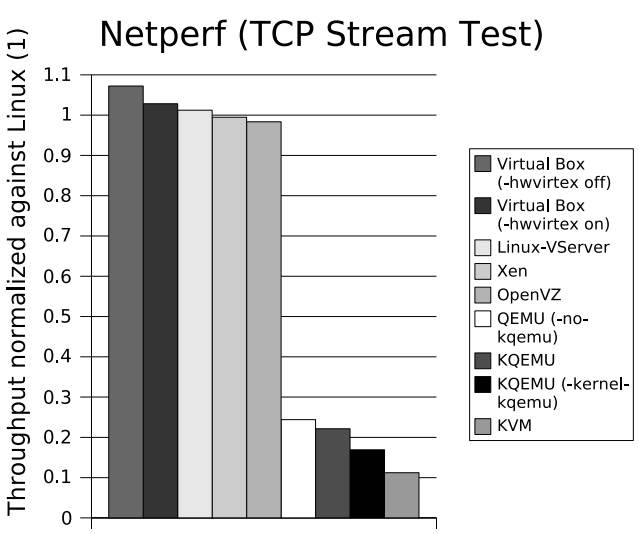

Figure 7: Netperf uses a stream of TCP packets to evaluate the performance of the network

to correctly measure this experiment with KQEMU and VirtualBox, the resultant values for time and throughput being noticeably inaccurate when compared against a wall clock.

Figure 7 shows our results for *Netperf*, a simple network benchmark that uses a stream of TCP packets to evaluate the performance of data exchange. We can clearly differentiate two groups in this experiment: the technologies that are based on QEMU, presenting a poor performance, and the others, which all presented excellent performance. We highlight VirtualBox's performance, possibly due to the use of a special network driver implementation that communicates closely with the physical network interface.

To complement our network evaluation, we have performed two other experiments using *Rsync* to transfer data from a server to a client machine. The first experiment consisted in the transfer of the entire kernel source tree, which is composed by 23741 small files for a total of 294M. The second experiment consisted in the transfer of the same iso image file used in the compression benchmark. Figure 8 presents the results for both experiments. They confirm the strength of OpenVZ for tasks that include transfering data throughout the network. In the opposite sense, these kinds of task reveal one of the major weaknesses of KVM.

Figure 9 presents our results for the *SysBench* database server performance benchmark (OLTP). The workload consisted of a set of 10000 transactions performed against a *MySQL* database. For this workload, Linux-VServer and Xen where the only virtualization solutions to perform close to Linux, while KVM and OpenVZ presented performance half as good.

For the second part of our evaluation, we have chosen the SysBench benchmark to evaluate how the virtualization solutions perform when having to manage and share the physical resources of the server between multiple virtual machines executing the same workload. Figure 10 presents our results for this scenario. The left side of the picture (a) shows the aggregate throughput for each set, which was calculated by multiplying the average throughput of the virtual machines, shown in the right side of the picture (b), by the number of virtual machines executing concurrently.

For all virtualization solutions but KVM and VirtualBox, the biggest aggregate throughput appears when 4 VMs were running concurrently. Xen and KQEMU presented a similar behavior, producing an almost constant aggregate throughput, but with opposite performances: while Xen can be considered the most efficient virtualization solution for this particular workload, the inability of KQEMU to make a good use of the available resources was evident.

The two most interesting results were achieved by VirtualBox and Linux-VServer. The aggregate throughput of the first grew smoothly until the number of running

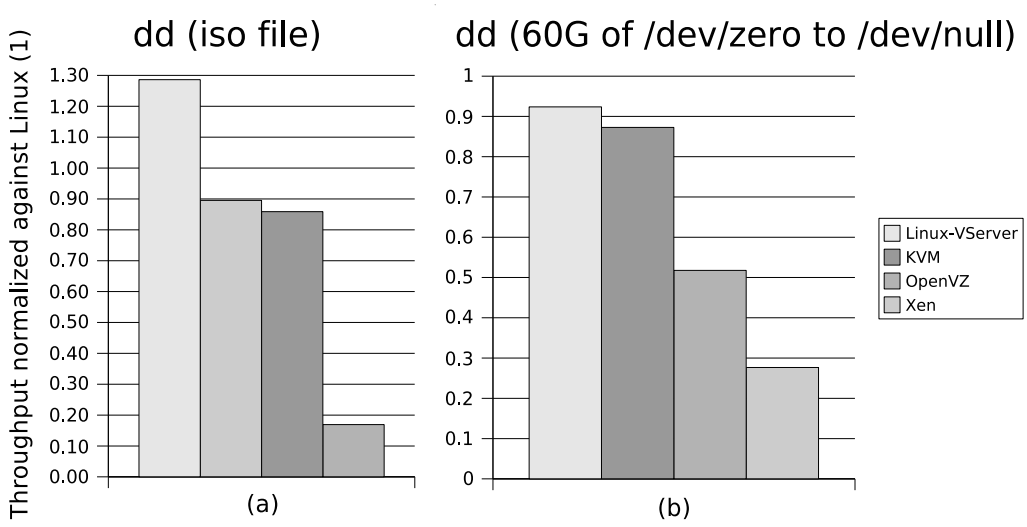

Figure 6: Using *dd* to evaluate disk performance

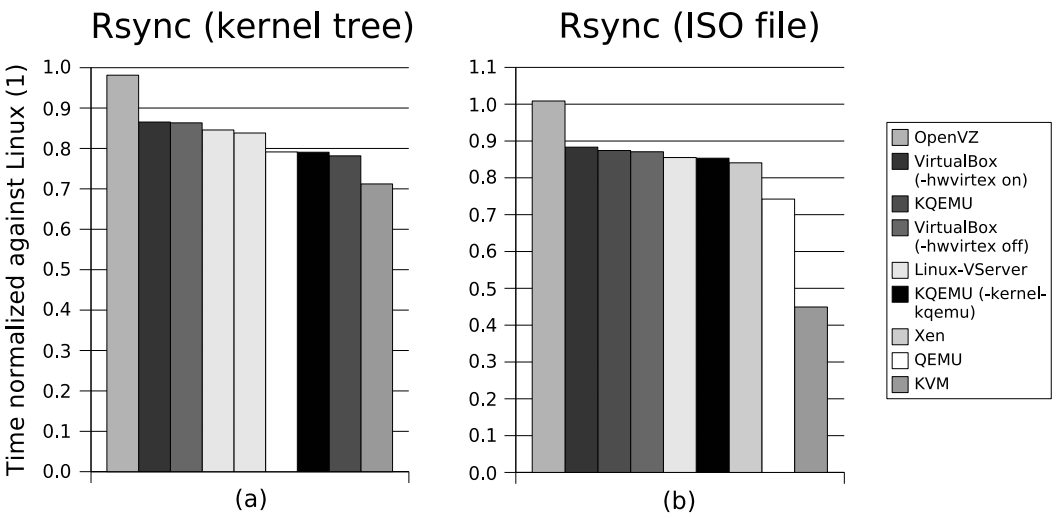

Figure 8: Evaluating data transfer in the network with *Rsync*

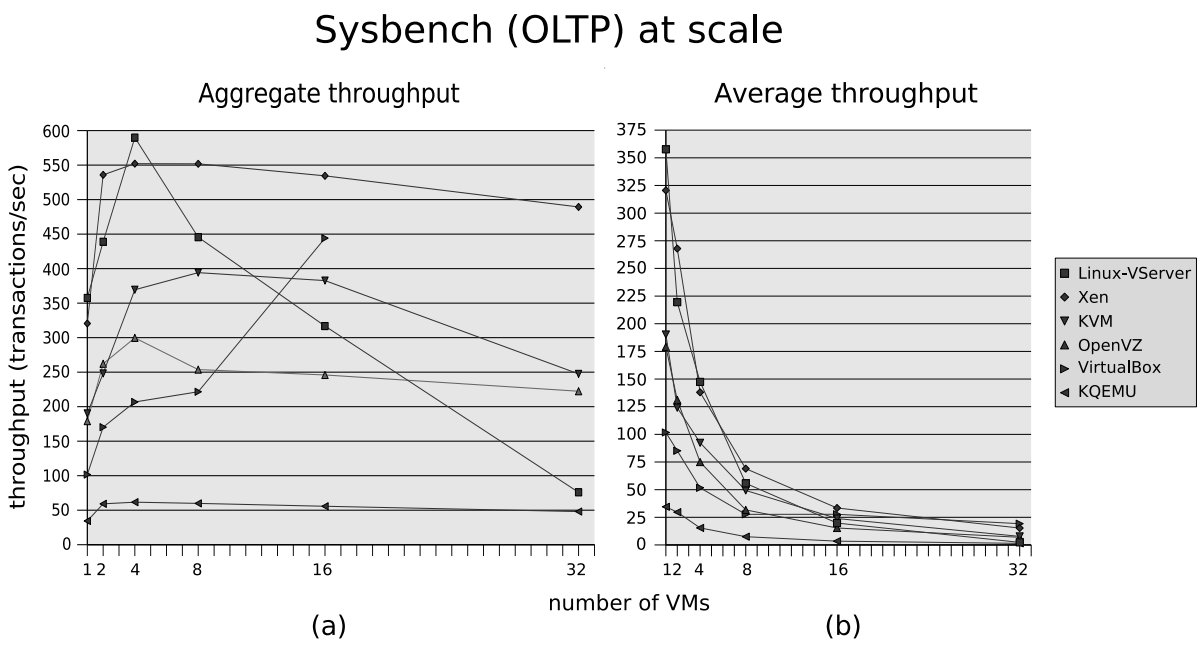

Figure 10: Evaluating the capacity of the virtualization solutions to manage and share the physical available resources with *SysBench*: the left side of the picture (a) shows the aggregate throughput (average throughput per VM x number of VMs) while the right side of the picture (b) shows the average throughput per VM

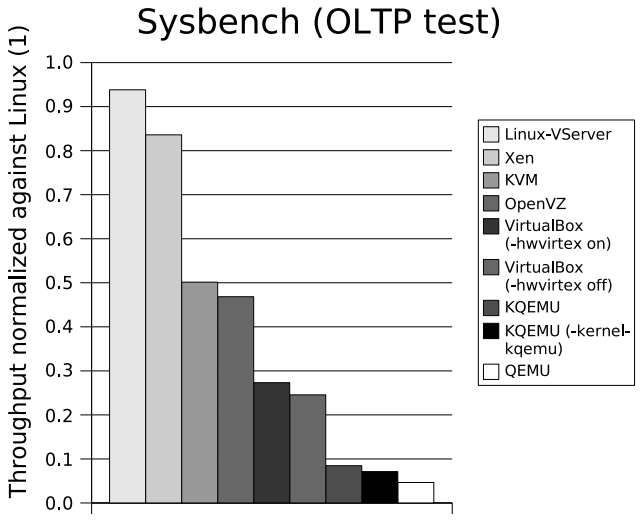

Figure 9: Using the *SysBench* OLTP benchmark to evaluate the performance of a database server

virtual machines reached 8. When we doubled the number of virtual machines to 16, the average throughput per VM remained the same, duplicating the total aggregate throughput produced. We were not able, though, to execute the experiment with 32 VMs, as the available memory per VM was insufficient to run the benchmark. With Linux-VServer, the aggregate throughput obtained for two and eight VMs (or vservers) was almost the same, but it fell considerably when running 16 and 32 vservers concurrently.

This is not the behavior we are used to seeing in production systems running this virtualization solution. We have contacted the authors of [16], who pointed some issues that could help explain Linux-VServer's performance for this particular workload at scale. In summary, they suggested to try different kernel I/O schedulers and timer frequencies, and also executing the same experiment directly in the host OS with and without VServer's kernel patch, and compare the results with those of VServer. Figure 11 summarizes this analysis. In the legend, the data between parenthesis are respectively the Ubuntu version, the kernel timer frequency, and the kernel I/O scheduler used in each experiment.

The Vserver patch used in the experiment did not

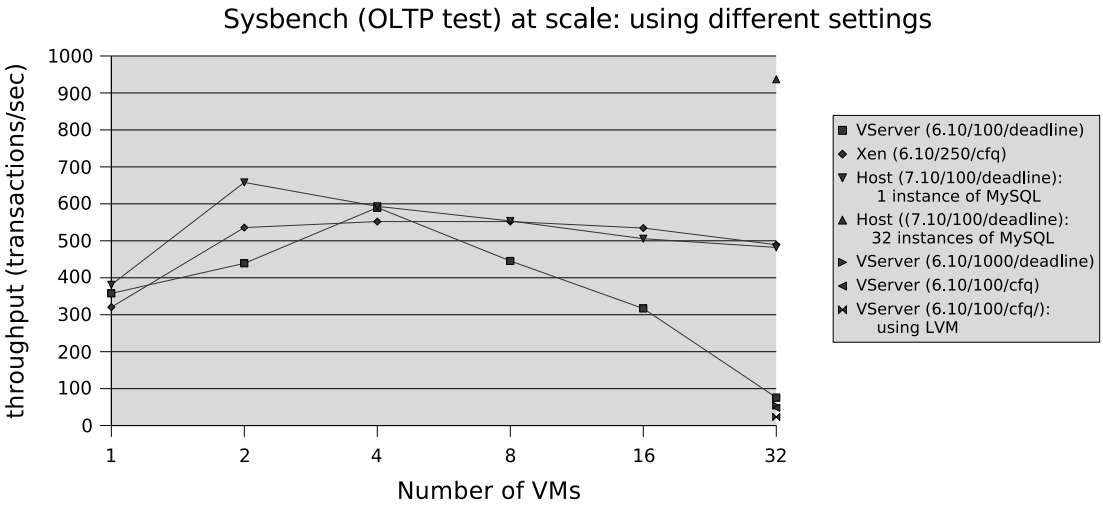

Figure 11: Repeating the scalability evaluation with Linux-VServer, Xen, and Linux using different settings.

change the kernel standard timer frequency, set to 100 Hz, nor the I/O scheduler, *deadline*. Xen uses a different configuration, setting the kernel timer frequency to 250 Hz and selecting a different I/O scheduler, called *Completely Fair Queuing*(cfq). We have repeated the experiment with VServer for n=32 VMs using different combinations of timer frequency and I/O scheduler. We have also experimented installing each vserver in individual LVM partitions. Neither of those configurations gave better results.

To eliminate the possibility of this being a kernel-related issue we repeated the experiment with 1, 2, 4, 8, and 32 instances of SysBench running in the host system and accessing the same database. The resulting performance curve resembles more the one of Xen. The aggregate throughput decreases slowly when there is more than 4 instances executing concurrently and does not follow the pattern of VServer. We also repeated this experiment running 32 instances of SysBench that connected each to different databases, hosted by 32 *mysqld* servers running on different ports of the host system. This configuration achieved by far the best aggregate throughput in our experiments.

The PlanetLab project uses Linux-VServer to share part of the available resources of their clusters of machines in "slices" [33]. We tried a patch used in the project that fixes a CPU scheduler related bug present in the version of VServer we used in our experiments, with no better results. Finally, we decided to try a different workload, the kernel build benchmark. For VServer and Xen experiments we used the same configuration as in our previous experiments. For the experiment in the host system we have simply used multiple kernel source trees. Figure 12 summarizes the results, showing the aggregate time for each set of experiments.

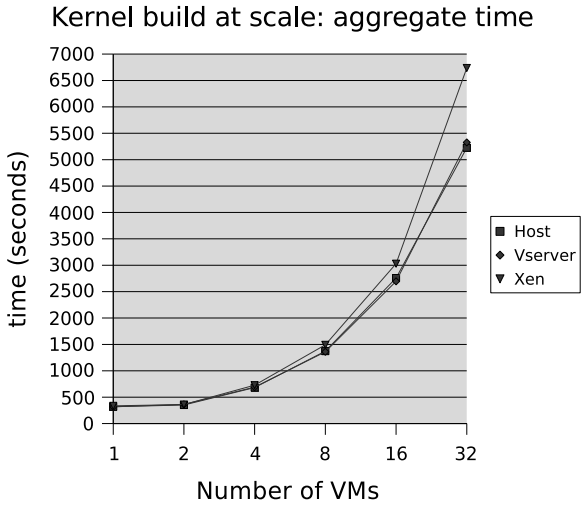

Figure 12: Evaluation of the scalability of Linux-VServer and Xen using the kernel build as benchmark and Linux as reference

For this particular workload, Linux-VServer performs as well as the Linux host system, while Xen follows closely but shows a small overhead that becomes more significant when the experiment is done with 32 VMs.

The results we had with this experiment helps to shown how the nature of the workload affects the performance of the virtualization solutions, and how Linux-VServer can deliver the performance we expect from it.

6 Conclusion

This paper evaluated the efficiency of six open source virtualization solutions for Linux. In the first part of our study, we used several benchmarks to analyze the performance of different virtualisation solutions under different types of load and related it to the raw performance of Linux, observing the resulting overhead. In the second part of our study, we evaluated the scalability of those virtualization solutions by running the same benchmark concurrently in multiple virtual machines.

For the first part of our evaluation, Linux-VServer performed close to Linux in all experiments, showing little to no overhead in all situations but one, in which it surpassed Linux's own performance for disk I/O. When executing SysBench at scale, thought, VServer failed to deliver the expected aggregate throughput, specially when the benchmark was running in more than four virtual machines concurrently. A closer look at the problem indicates that it is not directly related to Linux's kernel. We tried different combinations of kernel I/O schedulers and timer frequencies with no better results. To be fair, we repeated the experiment with Linux, Xen, and Linux-VServer using the kernel build benchmark instead of SysBench. This time VServer's performance was comparable to Linux's while Xen showed a small overhead that became more significant when running the experiment concurrently in 32 VMs.

Xen performed fairly well in all experiments but the one using the file system benchmark Dbench. In fact, no other virtualization solution had good results for this benchmark, with the exception of Linux-Vserver. Xen was also the solution that presented the best aggregate throughput when executing SysBench at scale.

KVM performed quite well for a full-virtualization solution. Although it makes for a good development tool, our results indicate it should be avoided for running application services that rely heavily on network I/O. On the other hand, the performance of OpenVZ was disappointing, except when the workload included data transfer throughout the network, which proved to be a strength of this virtualization solution. VirtualBox also showed good results for experiments that focused on the network but did not performed as well as the others, with the exception of the file compression benchmark. Finally, KQEMU may also be a good candidate in the area of development but for now its use should be avoided in production systems.

For the consolidation of Linux servers, virtualization technologies such as para-virtualization and OS-level virtualization seem to make more efficient use of the available physical resources. However, our findings indicate that the scalability of virtualization solutions may be directly related to the nature of the workload. Except for Linux-VServer and Xen, we have not used different workloads in the second part of our work and we suggest that this should be took in consideration for future studies. It would also be interesting to repeat the scalability experiments using a mix of different workloads. With the exception of web hosting centers, there are few production systems interested in running multiple instances of the same workload in the same physical server.

References

[1] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM Press.

[2] 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, New York, NY, USA, 2003. ACM Press.

[3] Daniel Bartholomew. Qemu: a multihost, multitarget emulator. *Linux J.*, 2006(145):3, May 2006.

[4] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[5] Franck Cappello Benjamin Quetier, Vincent Neri. Selecting a virtualization system for grid/p2p

large scale emulation. In *Proc of the Workshop on Experimental Grid testbeds for the assessment of large-scale distributed applications and tools (EXPGRID'06), Paris, France, 19-23 june*, 2006.

[6] Lucas Bonnet. Etat de l'art des solutions libres de virtualisation pour une petite entreprise. Livre blanc, Bearstech, Decembre 2007. `http://bearstech.com/files/LB-virtualisationEntrepriseBearstech.pdf`.

[7] Standard Performance Evaluation Corporation. Spec virtualization committee, April 2008. `http://www.spec.org/specvirtualization/`.

[8] Simon Crosby and David Brown. The virtualization reality. *Queue*, 4(10):34–41, 2007.

[9] Casazza et al. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal*, 10(3):243–252, 2006.

[10] Makhija et al. Vmmark: A scalable benchmark for virtualized systems. Tech reaport, VMware, Inc., 2006.

[11] Padala et al. Performance evaluation of virtualization technologies for server consolidation. Tech Reaport HPL-2007-59, HP Laboratories Palo Alto, 2007.

[12] Justin M. Forbes. Why virtualization fragmentation sucks. In *Proceedings of the Linux Symposium*, volume 1, pages 125–129, Ottawa, ON, Canada, June 2007.

[13] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, and Mark Williamson. Safe hardware access with the xen vir tual machine monitor. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for On-Demand IT In frastructure*, Boston, MA, USA, October 2004.

[14] Cedric Le Goater, Daniel Lezcano, Clement Calmels, Dave Hansen, Serge E. Hallyn, and Hubertus Franke. Making applications mobile under linux. In *Proceedings of the Linux Symposium*, volume 1, pages 347–368, Ottawa, ON, Canada, July 2006.

[15] Risto Haukioja and Neil Dunbar. Introduction to linux virtualization solutions. Technical report, Hewlett-Packard Development Company, L.P., September 2006.

[16] Marc E. Fiuczynski Herbert Potzl. Linux-vserver: Resource efficient os-level virtualization. In *Proceedings of the Linux Symposium*, volume 2, pages 151–160, Ottawa, ON, Canada, June 2007.

[17] IBM. Driving business value with a virtualized infrastructure. Technical report, International Business Machines Corporation, March 2007.

[18] M. Tim Jones. Virtual linux, December 2006. `http://www.ibm.com/developerworks/library/l-linuxvirt/index.htlm`.

[19] M. Tim Jones. Discover the linux kernel virtual machine, April 2007. `http://www.ibm.com/developerworks/linux/library/l-linux-kvm/`.

[20] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, Ottawa, ON, Canada, June 2007.

[21] Tim Klassel and Jeffrey Peck. The rise of the virtual machine and the real impact it will have. Technical report, Thomas Eisel Partners, 2006.

[22] Jun Nakajima and Asit K. Mallick. Hybrid-virtualization: Enhanced virtualization for linux. In *Proceedings of the Linux Symposium*, volume 2, pages 87–96, Ottawa, ON, Canada, June 2007.

[23] Susanta Nanda and Tzi cker Chiueh. A Survey on Virtualization Technologies. Rpe report, State University of New York, 2005.

[24] Pradeep Padala. Adaptive control of virtualized resources in utility computing environments. In *Proc of the EUROSYS (EUROSYS'07), Lisboa, Portugal, March 21-23*, 2007.

[25] John Pflueger and Sharon Hanson. Data center efficiency in the scalable enterprise. *Dell Power Solutions*, pages 08–14, February 2007.

[26] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[27] Benjamin Quetier and Vincent Neri. V-meter: Microbenchmark pour evaluer les utilitaires de virtualisation dans la perspective de systemes d'emulation a grande echelle. In *16eme Rencontres Francophones du Parallelisme (RenPar'16)*, Le Croisic, France, Avril 2005.

[28] J. Robin and C. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor, 2000.

[29] Michael D. Day Ryan A. Harper and Anthony N. Liguori. Using kvm to run xen guests without xen. In *Proceedings of the Linux Symposium*, volume 1, pages 179–188, Ottawa, ON, Canada, June 2007.

[30] Love H. Seawright and Richard A. MacKinnon. Vm/370 - a study of multiplicity and usefulness. *IBM Systems Journal*, 18(1):4–17, 1979.

[31] B. D. Shriver, J. W. Anderson, L. J. Waguespack, D. M. Hyams, and R. A. Bombet. An implementation scheme for a virtual machine monitor to be realized on user - microprogrammable minicomputers. In *ACM 76: Proceedings of the annual conference*, pages 226–232, New York, NY, USA, 1976. ACM Press.

[32] Amit Singh. An introduction to virtualization, 2006. `http://www.kernelthread.com/publications/virtualization/`.

[33] Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proc of the EUROSYS (EUROSYS'07), Lisboa, Portugal, March 21-23*, 2007.

[34] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[35] VMware. A performance comparison of hypervisors. Technical report, VMware, Inc., 2007.

[36] XenSource. A performance comparison of commercial hypervisors. Technical report, XenSource, Inc., 2007.

[37] Alan Zeichick. Processor-based virtualization, amd64 style, part i. Technical report, Advanced Micro Devices, 2006. `http://developer.amd.com/article_print.jsp?id=14`.

Proceedings of the Linux Symposium

Volume One

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*