We have seen that aVMMsuch as Xen introduces additional overhead and negatively affects performance

[41,241,242]. The topic of this section is a quantitative analysis of the performance of VMs.We compare

the performance of two virtualization techniques with a standard operating system: a plain-vanilla

Linux referred to as “the base” system. The two VM systems are Xen, based on paravirtualization, and

OpenVZ [281].

First we take a closer look at OpenVZ, a system based on OS-level virtualization. OpenVZ uses a

single patched Linux kernel. The guest operating systems in different containers15 may be different

distributions but must use the same Linux kernel version that the host uses. The lack of flexibility of the

approach for virtualization in OpenVZ is compensated by lower overhead.

The memory allocation in OpenVZ is more flexible than in the case of paravirtualization; memory

not used in one virtual environment can be used by others. The system uses a common file system. Each

virtual environment is a directory of files isolated using chroot. To start a new virtual machine, one

needs to copy the files from one directory to another, create a config file for the virtual machine, and

launch the VM.

OpenVZ has a two-level scheduler: At the first level, the fair-share scheduler allocates CPU time

slices to containers based on cpuunits values; the second level is a standard Linux scheduler that

decides what process to run in that container. The I/O scheduler also has two levels; each container has

an I/O priority, and the scheduler distributes the available I/O bandwidth according to the priorities.

The discussion in [281] is focused on the user’s perspective, thus the performance measures analyzed

are the throughput and the response time. The general question is whether consolidation of the

applications and servers is a good strategy for cloud computing. The specific questions examined are:

• How does the performance scale up with the load?

• What is the impact of a mix of applications?

• What are the implications of the load assignment on individual servers?

There is ample experimental evidence that the load placed on system resources by a single application

varies significantly in time. A time series displaying CPU consumption of a single application in time

clearly illustrates this fact. As we all know, this phenomenon justifies the need for CPU multiplexing

among threads/processes supported by an operating system. The concept of application and server

consolidation is an extension of the idea of creating an aggregate load consisting of several applications

and aggregating a set of servers to accommodate this load. Indeed, the peak resource requirements of

individual applications are very unlikely to be synchronized, and the aggregate load tends to lead to a

better average resource utilization.

The application used in [281] is a two-tier system consisting of an ApacheWeb server and aMySQL

database server. A client of this application starts a session as the user browses through different items in

the database, requests information about individual items, and buys or sells items. Each session requires

the creation of a new thread; thus, an increased load means an increased number of threads. To understand

the potential discrepancies in performance among the three systems, a performance-monitoring tool reports the counters that allow the estimation of (i) the CPU time used by a binary; (ii) the number of

L2-cache misses; and (iii) the number of instructions executed by a binary.

The experimental setups for three different experiments are shown in Figure 5.9 . In the first group

of experiments the two tiers of the application, the Web and the DB, run on a single server for the

Linux, the OpenVZ, and the Xen systems. When the workload increases from 500 to 800 threads, the

throughput increases linearly with the workload. The response time increases only slightly for the base

system and for the OpenVZ system, whereas it increases 600% for the Xen system. For 800 threads the

response time of the Xen system is four times longer than the time for OpenVZ. The CPU consumption

grows linearly with the load in all three systems; the DB consumption represents only 1–4% of it. For a given workload, the Web-tier CPU consumption for the OpenVZ system is close to that of the

base system and is about half of that for the Xen system. The performance analysis tool shows that the

OpenVZ execution has two times more L2-cache misses than the base system, whereas the Xen Dom0

has 2.5 times more and the Xen application domain has 9 times more. Recall that the base system and the

OpenVZ run a Linux OS and the sources of cache misses can be compared directly, whereas Xen runs a

modified Linux kernel. For the Xen-based system the procedure hypervisor\_callback, invoked when

an event occurs, and the procedure evtchn\_do\_upcall, invoked to process an event, are responsible for

32% and 44%, respectively, of the L2-cache misses. The percentage of the instructions invoked by these

two procedures are 40% and 8%, respectively. Most of the L2-cache misses in OpenVZ and the base

system occur in (i) a procedure called do\_anonymous\_pages, used to allocate pages for a particular

application with the percentage of cache misses 32% and 25%, respectively; (ii) the procedures called

\_copy\_to\_user\_ll and \_copy\_ f rom\_user\_ll, used to copy data from user to system buffers and back

with the percentage of cache misses (12+7)%and (10+1)%, respectively. The first figure refers to the

copying from user to system buffers and the second to copying from system buffers to the user space.

The second group of experiments uses two servers, one for the Web and the other for the DB

application, for each one of the three systems. When the load increases from 500 to 800 threads the

throughput increases linearly with the workload. The response time of the Xen system increases only

114%, compared with 600% reported for the first experiments. The CPU time of the base system, the

OpenVZ system, the Xen Dom0, and the User Domain are similar for the Web application. For the DB

application, the CPU time of the OpenVZ system is twice as long as that of the base system, whereas

Dom0 and the User Domain require CPU times of 1.1 and 2.5 times longer than the base system. The

L2-cache misses for theWeb application relative to the base system are the same for OpenVZ, 1.5 times

larger for Dom0 of Xen, and 3.5 times larger for the User Domain. The L2-cache misses for the DB

application relative to the base system are 2 times larger for the OpenVZ, 3.5 larger for Dom0 of Xen,

and 7 times larger for the User Domain.

The third group of experiments uses two servers, one for theWeb and the other for the DB application,

for each one of the three systems but runs four instances of the Web and the DB application on the two

servers. The throughput increases linearly with the workload for the range used in the previous two

experiments, from 500 to 800 threads. The response time remains relatively constant for OpenVZ and

increases 5 times for Xen.

The main conclusion drawn from these experiments is that the virtualization overhead of Xen is considerably

higher than that of OpenVZ and that this is due primarily to L2-cache misses. The performance

degradation when the workload increases is also noticeable for Xen. Another important conclusion is

that hosting multiple tiers of the same application on the same server is not an optimal solution.