Performance isolation is a critical condition for quality-of-service (QoS) guarantees in shared computing

environments. Indeed, if the run-time behavior of an application is affected by other applications running

concurrently and, thus, is competing for CPU cycles, cache, main memory, and disk and network

access, it is rather difficult to predict the completion time. Moreover, it is equally difficult to optimize

the application. Several operating systems, including Linux/RK [270], QLinux [343], and SILK [44],

support some performance isolation, but problems still exist because one has to account for all resources

used and to distribute the overhead for different system activities, including context switching and

paging, to individual users – a problem often described as QoS crosstalk [348].

Processor virtualization presents multiple copies of the same processor or core onmulticore systems.

The code is executed directly by the hardware, whereas processor emulation presents a model of another

hardware system in which instructions are “emulated” in software more slowly than virtualization. An

example is Microsoft’s VirtualPC, which could run on chip sets other than the x86 family. It was used

on Mac hardware until Apple adopted Intel chips.

Traditional operating systems multiplex multiple processes or threads, whereas a virtualization supported

by a VMM multiplexes full operating systems.Obviously, there is a performance penalty because

an OS is considerably more heavyweight than a process and the overhead of context switching is larger.

A VMM executes directly on the hardware a subset of frequently used machine instructions generated by the application and emulates privileged instructions, including device I/O requests. The subset of

the instructions executed directly by the hardware includes arithmetic instructions, memory access, and

branching instructions.

Operating systems use process abstraction not only for resource sharing but also to support isolation.

Unfortunately, this is not sufficient from a security perspective. Once a process is compromised, it is

rather easy for an attacker to penetrate the entire system. On the other hand, the software running on

a virtual machine has the constraints of its own dedicated hardware; it can only access virtual devices

emulated by the software. This layer of software has the potential to provide a level of isolation nearly

equivalent to the isolation presented by two different physical systems. Thus, the virtualization can be

used to improve security in a cloud computing environment.

A VMM is a much simpler and better specified system than a traditional operating system. For

example, the Xen VMM discussed in Section 5.8 has approximately 60,000 lines of code, whereas

the Denali VMM [372] has only about half that, or 30,000 lines of code. The security vulnerability

of VMMs is considerably reduced because the systems expose a much smaller number of privileged

functions. For example, the Xen VMM can be accessed through 28 hypercalls, whereas a standard

Linux allows hundreds (e.g., Linux 2.6.11 allows 289 system calls). In addition to a plethora of system

calls, a traditional operating system supports special devices (e.g., /dev/kmem) and many privileged

programs from a third party (e.g., sendmail and sshd).