In 1974 Gerald J. Popek and Robert P. Goldberg gave a set of sufficient conditions for a computer

architecture to support virtualization and allow a VMM to operate efficiently [293]:

• A program running under the VMM should exhibit a behavior essentially identical to that demonstrated

when the program runs directly on an equivalent machine.

• The VMM should be in complete control of the virtualized resources.

• A statistically significant fraction of machine instructions must be executed without the intervention

of the VMM.

Another way to identify an architecture suitable for a virtual machine is to distinguish two classes

of machine instructions: sensitive instructions, which require special precautions at execution time, and

innocuous instructions, which are not sensitive. In turn, sensitive instructions can be:

• Control sensitive, which are instructions that attempt to change either the memory allocation or the

privileged mode.

• Mode sensitive, which are instructions whose behavior is different in the privileged mode.

An equivalent formulation of the conditions for efficient virtualization can be based on this classification

of machine instructions. A VMM for a third-generation (or later) computer can be constructed

if the set of sensitive instructions is a subset of the privileged instructions of that machine. To handle

nonvirtualizable instructions, one could resort to two strategies:

• Binary translation. The VMM monitors the execution of guest operating systems; nonvirtualizable

instructions executed by a guest operating system are replaced with other instructions.

• Paravirtualization. The guest operating system is modified to use only instructions that can be

virtualized.

There are two basic approaches to processor virtualization: full virtualization, in which each virtual

machine runs on an exact copy of the actual hardware, and paravirtualization, in which each virtual

machine runs on a slightly modified copy of the actual hardware (see Figure 5.4). The reasons that

paravirtualization is often adopted are (i) some aspects of the hardware cannot be virtualized; (ii)

to improve performance; and (iii) to present a simpler interface. VMware VMMs are examples of

full virtualization. Xen [41] and Denali [372] are based on paravirtualization; Section 5.8 covers the

strategies to overcome hardware limitations for paravirtualization in Xen.

Full virtualization requires a virtualizable architecture; the hardware is fully exposed to the guest

OS, which runs unchanged, and this ensures that this direct execution mode is efficient. On the other

hand, paravirtualization is done because some architectures such as x86 are not easily virtualizable.

Paravirtualization demands that the guest OS be modified to run under the VMM; furthermore, the

guest OS code must be ported for individual hardware platforms.

Systems such as VMware EX Server support full virtualization on x86 architecture. The virtualization

of the memory management unit (MMU) and the fact that privileged instructions executed by a guest

OS fail silently pose some challenges; for example, to address the latter problem, one has to insert

traps whenever privileged instructions are issued by a guest OS. The system must also maintain shadow

copies of system control structures, such as page tables, and trap every event affecting the state of these

control structures; the overhead of many operations is substantial .

Application performance under a virtual machine is critical; generally, virtualization adds some level

of overhead that negatively affects the performance. In some cases an application running under a VM performs better than one running under a classical OS. This is the case of a policy called cache isolation.

The cache is generally not partitioned equally among processes running under a classical OS, since one

process may use the cache space better than the other. For example, in the case of two processes, one

write-intensive and the other read-intensive, the cache may be aggressively filled by the first. Under

the cache isolation policy the cache is divided between the VMs and it is beneficial to run workloads

competing for cache in two different VMs [324]. The application I/O performance running under a VM

depends on factors such as the disk partition used by the VM, the CPU utilization, the I/O performance

of the competing VMs, and the I/O block size. On a Xen platform, discrepancies between the optimal

choice and the default are as high as 8% to 35% [324].