# Virtual Machines

The concept of virtual machines existed long before cloud computing was invented. The technologies used to implement virtual machines can be divided into three broad categories:

- Software emulation
- Para-virtualization
- Full virtualization

*Software emulation:* The idea of using software to emulate a computer has many applications. One use involves accommodating heterogeneous computers. Suppose a user who has a particular type of computer, C1, wants to run a program, *P*, that has been compiled for another type of computer, C2. A piece of software called a *software emulator* solves the problem. The emulator treats program P as data by repeatedly reading an instruction, doing whatever C2 would do, and then moving to the next instruction.

- The software emulation approach has been especially popular as a way to make a new programming language available quickly.
- Before a language can be used on a particular type of computer, someone must write a compiler that translates programs into instructions the computer understands.
- Writing a compiler is time-consuming and error prone, and often limits how quickly a new language can be widely adopted.
- Software emulation allows a language designer to write one compiler and then use the compiler on all types of computers.
- To implement a compiler for a new programming language, use an existing language to build two tools: a compiler and a software emulator.
- To simplify writing the tools, define a hypothetical computer (i.e., a type of virtual machine).
- Build a compiler that translates programs written in the new language into code for the hypothetical machine.
- Build a software emulator that reads code for the hypothetical machine and carries out the steps the code specifies.
- The community uses the term *byte code* to describe instructions for the hypothetical computer, calls the compiler a *byte code compiler*, and calls the emulator a *byte code interpreter*.
- Once the compiler and software emulator have been written, porting them to a new type of computer is much easier than building a new compiler.

*Para-virtualization: A*n approach to virtualization which allows multiple operating systems to run on a single computer at the same time.

Instead, para-virtualization uses a piece of software known as a *hypervisor* to control multiple operating systems and move the processors among them analogous to the way a single operating system moves the processor among a set of running processes.

Unlike software emulation, para-virtualization allows software to run at high speed. That is, para-virtualization avoids the inherent overhead of software emulation and allows the processor to execute instructions directly with no extra software involved.

To prohibit a given operating system from commandeering the hardware, paravirtualization requires the code for an operating system to be altered before it can be used.

All *privileged instructions* (i..e., instructions used to gain control of the hardware and I/O devices) must be replaced by calls to hypervisor functions.

Thus, when an operating system tries to perform I/O, control passes to the hypervisor, which can choose whether to perform the requested operation on behalf of the operating system or deny the request.

- *Full virtualization*. Like para-virtualization, the approach known as *full virtualization* allows multiple operating systems to run on the same computer at the same time.
- Furthermore, full virtualization avoids the overhead of software emulation and allows operating system code to run unaltered.

The full virtualization technologies currently used to support Virtual Machines(VMs) in cloud data centers have three key properties:

Emulation of commercial instruction sets

Isolated facilities and operation

Efficient, low-overhead execution

*Emulation of commercial instruction sets*.

To a customer, a VM appears to be identical to a conventional computer, including the complete instruction set. Code that has been compiled to run on a commercial computer will run on a VM unchanged. In fact, a VM can boot and run a commercial operating system, such as *Microsoft Windows* or *Linux*.

*Isolated facilities and operation*. Although multiple VMs on a given server can each run an operating system at the same time, the underlying system completely isolates each VM from the others. From the point of view of an operating system running on a VM, the operating system thinks it controls all of physical memory, a set of I/O devices, and the processor, including all cores.

*Efficient, low-overhead execution*. The most important and surprising aspect of VM technology arises from the low overhead a VM imposes. In fact, until one understands how a VM works, performance seems impossibly fast because most instructions, especially those used in application code execute natively (i.e., at full hardware speed).

- The general idea behind VM is straightforward: load software onto a server that allows the cloud provider to create one or more VMs. Allow the tenant who owns each VM to boot an operating system on the VM, and then use the operating system to launch one or more applications.
- The key piece of software responsible for creating and managing VMs is known as a *hypervisor*. We think of a hypervisor as controlling the underlying hardware.
- Each VM the hypervisor creates is independent of other VMs. Figure 5.1 illustrates the conceptual organization of a server running hypervisor software, and a set of VMs that each run an operating system and apps.