CHAPTER 8

Virtualization technology

If one had to choose a single technology that has been most influential in enabling the cloud computing paradigm, it would have to be virtualization. As we have seen earlier in Chapter 1, virtualization is not new, and dates back to the early mainframes as a means of sharing computing resources amongst users. Today, besides underpinning cloud computing platforms, virtualization is revolutionizing the way enterprise data centers are built and managed, paving the way for enterprises to deploy 'private cloud' infrastructure within their data centers.

8.1 VIRTUAL MACHINE TECHNOLOGY

We begin with an overview of virtual machine technology: In general, any means by which many different users are able simultaneously to interact with a computing system while each perceiving that they have an entire 'virtual machine' to themselves, is a form of virtualization. In this general sense, a traditional multiprogramming operating system, such as Linux, is also a form of virtualization, since it allows each user process to access system resources oblivious of other processes. The abstraction provided to each process is the set of OS system calls and any hardware instructions accessible to user-level processes. Extensions, such as 'user mode Linux' [17] offer a more complete virtual abstraction where each user is not even aware of other user's processes, and can login as an administrator, i.e. 'root,' to their own seemingly

private operating system. 'Virtual private servers' are another such abstraction [36]. At a higher level of abstraction are virtual machines based on high-level languages, such as the Java virtual machine (JVM) which itself runs as an operating system process but provides a system-independent abstraction of the machine to an application written in the Java language. Such abstractions, which present an abstraction at the OS system call layer or higher, are called *process virtual machines*. Some cloud platforms, such as Google's App Engine and Microsoft's Azure, also provide a process virtual machine abstraction in the context of a web-based architecture.

More commonly, however, the virtual machines we usually refer to when discussing virtualization in enterprises or for infrastructure clouds such as Amazon's EC2 are *system virtual machines* that offer a complete hardware instruction set as the abstraction provided to users of different virtual machines. In this model many system virtual machine (VM) instances share the same physical hardware through a virtual machine monitor (VMM), also commonly referred to as a *hypervisor*. Each such system VM can run an independent operating system instance; thus the same physical machine can have many instances of, say Linux *and* Windows, running on it simultaneously. The system VM approach is preferred because it provides complete isolation between VMs as well as the highest possible flexibility, with each VM seeing a complete machine instruction set, against which any applications for that architecture are guaranteed to run.

It is the virtual machine monitor that enables a physical machine to be virtualized into different VMs. Where does this software itself run? A *host* VMM is implemented as a process running on a *host* operating system that has been installed on the machine in the normal manner. Multiple *guest* operating systems can be installed on different VMs that each run as operating system processes under the supervision of the VMM. A *native* VMM, on the other hand, does not require a host operating system, and runs directly on the physical machine (or more colloquially on 'bare metal'). In this sense, a native VMM can be viewed as a special type of operating system, since it supports multiprogramming across different VMs, with its 'system calls' being hardware instructions! Figure 8.1 illustrates the difference between process virtual machines, host VMMs and native VMMs. Most commonly used VMMs, such as the open source Xen hypervisor as well as products from VMware are available in both hosted as well as native versions; for example the hosted Xen (HXen) project and VMware Workstation products are hosted VMMs, whereas the more popularly used XenServer (or just Xen) and VMware ESX Server products are native VMMs.

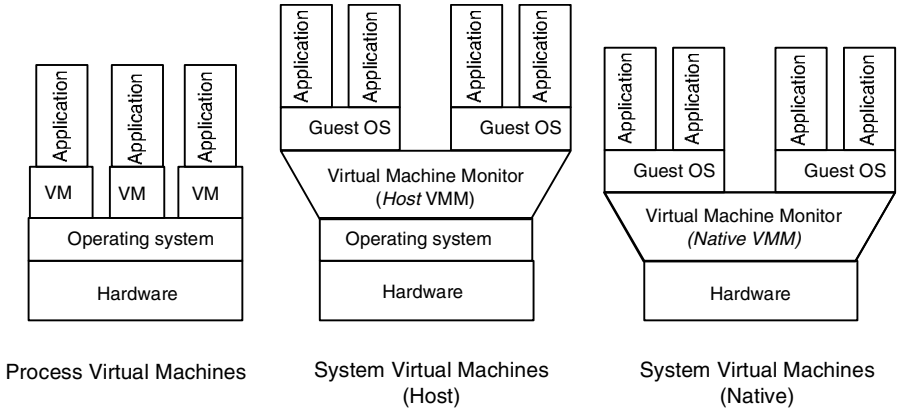

FIGURE 8.1. Virtual machines

In the next section we shall briefly describe how system virtual machines are implemented *efficiently* and how individual virtual machines actually run.

8.1.1 System virtual machines

A system virtual machine monitor needs to provide each virtual machine the illusion that it has access to a complete independent hardware system through a full instruction set. In a sense, this is very similar to the need for a time-sharing operating system to provide different processes access to hardware resources in their allotted time intervals of execution. However, there are fundamental differences between the 'virtual machine' as perceived by a traditional operating system processes and a true system VM:

1. Processes under an operating system are allowed access to hardware through system calls, whereas a system VMM needs to provide a full hardware instruction set for use by each virtual machine
2. Each system virtual machine needs to be able to run a full operating system, while itself maintaining isolation with other virtual machines.

Going forward we will focus our discussion on native VMMs that run directly on the hardware, like an operating system; native VMMs are more efficient and therefore the ones used in practice within enterprises as well as cloud platforms. One way a native system VMM could work is by *emulating* instructions of the target instruction set and maintaining the state of

different virtual machines at all levels of memory hierarchy (including registers etc.) *indirectly* in *memory* and switching between these as and when required, in a manner similar to how virtual memory page tables for different processes are maintained by an operating system. In cases where the target hardware instruction set and actual machine architecture are different, emulation and indirection is unavoidable, and, understandably, inefficient. However, in cases where the target instruction set is the same as that of the actual hardware on which the native VMM is running, the VMM can be implemented more efficiently.

An *efficient* native VMM attempts to run the instructions of each of is virtual machines natively on the hardware, and while doing so also maintain the state of the machine at its proper location in the memory hierarchy, in much the same manner as an operating system runs process code natively as far as possible except when required.

Let us first recall how an operating system runs a process: The process state is first loaded into memory and registers, then the program counter is reset so that process code runs from thereon. The process runs until a timer event occurs, at which point the operating system switches the process and resets the timer via a special *privileged* instruction. The key to this mechanism is the presence of privileged instructions, such as resetting the timer interrupt, which cause a *trap* (a program generated interrupt) when run in 'user' mode instead of 'system' mode. Thus, no *user* process can set the timer interrupt, since this instruction is privileged and always traps, in this case to the operating system.

Thus, it should be possible to build a VMM in exactly the same manner as an operating system, by trapping the privileged instructions and running all others natively on the hardware. Clearly the privileged instructions themselves need to be *emulated*, so that when an operating system running in a virtual machine attempts to, say, set the timer interrupt, it actually sets a *virtual* timer interrupt. Such a VMM, where only privileged instructions need to be emulated, is the most efficient native VMM possible, as formally proved in [45].

However, in reality it is not always possible to achieve this level of efficiency. There are some instruction sets (including the popular Intel IA-32, better known as x86) where some non-privileged instructions behave differently depending on whether they are called in user mode or system mode. Such instruction sets implicitly assume that there will be only one operating system (or equivalent) program that needs access to privileged instructions, a natural assumption in the absence of virtualization. However, such instructions pose a problem for virtual machines, in which the operating system is actually

running in user mode rather than system mode. Thus, it is necessary for the VMM to also emulate such instructions in addition to all privileged instructions. Newer editions of the x86 family have begun to include 'hardware support' for virtualization, where such anomalous behavior can be rectified by exploiting additional hardware features, resulting in a more efficient implementation of virtualization: For example, Intel's VT-x ('Vanderpool') technology includes a new VMX mode of operation. When VMX is enabled there is a new 'root' mode of operation exclusively for use by the VMM; in non-root mode all standard modes of operation are available for the OS and applications, including a 'system' mode which is at a lower level of privilege than what the VMM enjoys. We do not discuss system virtual machines in more detail here, as the purpose of this discussion was to give some insight into the issues that are involved through a few examples; a detailed treatment can be found in [58].

8.1.2 Virtual machines and elastic computing

We have seen how virtual machine technology enables decoupling physical hardware from the virtual machines that run on them. Virtual machines can have different instruction sets from the physical hardware if needed. Even if the instruction sets are the same (which is needed for efficiency), the size and number of the physical resources seen by each virtual machine need not be the same as that of the physical machine, and in fact will usually be different. The VMM partitions the actual physical resources in time, such as with I/O and network devices, or space, as with storage and memory. In the case of multiple CPUs, compute power can also be partitioned in time (using traditional time slices), or in space, in which case each CPU is reserved for a subset of virtual machines.

The term 'elastic computing' has become popular when discussing cloud computing. The Amazon 'elastic' cloud computing platform makes extensive use of virtualization based on the Xen hypervisor. Reserving and booting a server instance on the Amazon EC cloud provisions and starts a virtual machine on one of Amazon's servers. The configuration of the required virtual machine can be chosen from a set of options (see Chapter 5). The user of the 'virtual instance' is unaware and oblivious to which physical server the instance has been booted on, as well as the resource characteristics of the physical machine.

An 'elastic' multi-server environment is one which is completely virtualized, with all hardware resources running under a set of *cooperating* virtual

machine monitors and in which provisioning of virtual machines is largely automated and can be dynamically controlled according to demand. In general, any multi-server environment can be made 'elastic' using virtualization in much the same manner as has been done in Amazon's cloud, and this is what many enterprise virtualization projects attempt to do. The key success factors in achieving such elasticity is the degree of *automation* that can be achieved across multiple VMMs working together to maximize utilization. The *scale* of such operations is also important, which in the case of Amazon's cloud runs into tens of thousands of servers, if not more. The larger the scale, the greater the potential for amortizing demand effciently across the available capacity while also giving users an illusion of 'infinite' computing resources.

Technology to achieve elastic computing at scale is, today, largely proprietary and in the hands of the major cloud providers. Some automated provisioning technology is available in the public domain or commercially off the shelf (see Chapter 17), and is being used by many enterprises in their internal data center automation efforts. Apart from many startup companies, VMware's VirtualCentre product suite aims to provide this capability through its 'VCloud' architecture.

We shall discuss the features of an elastic data center in more detail later in this chapter; first we cover virtual machine migration, which is a pre-requisite for many of these capabilities.

8.1.3 Virtual machine migration

Another feature that is crucial for advanced 'elastic' infrastructure capabilities is 'in-flight' migration of virtual machines, such as provided in VMware's VMotion product. This feature, which should also be considered a key component for 'elasticity,' enables a virtual machine running on one physical machine to be suspended, its state saved and transported to or accessed from another physical machine where it is resumes execution from exactly the same state.

Virtual machine migration has been studied in the systems research community [49] as well as in related areas such as grid computing [29]. Migrating a virtual machine involves capturing and copying the entire state of the machine at a snapshot in time, including processor and memory state as well as all virtual hardware resources such as BIOS, devices or network MAC addresses. In principle, this also includes the entire disk space, including system and user directories as well as swap space used for virtual memory operating system scheduling. Clearly, the complete state of a typical server is likely to be quite large. In a closely networked multi-server environment, such as a cloud data

center, one may assume that some persistent storage can be easily accessed and mounted from different servers, such as through a storage area network or simply networked file systems; thus a large part of the system disk, including user directories or software can easily be transferred to the new server, using this mechanism. Even so, the remaining state, which needs to include swap and memory apart from other hardware states, can still be gigabytes in size, so migrating this efficiently still requires some careful design.

Let us see how VMware's VMotion carries out in-flight migration of a virtual machine between physical servers: VMotion waits until the virtual machine is found to be in a stable state, after which all changes to machine state start getting logged. VMotion then copies the contents of memory, as well as disk-resident data belonging to either the guest operating system or applications, to the target server. This is the *baseline* copy; it is not the final copy because the virtual machine continues to run on the original server during this process. Next the virtual machine is suspended and the last remaining changes in memory and state since the baseline, which were being logged, are sent to the target server, where the final state is computed, following which the virtual machine is activated and resumes from its last state.

8.2 Virtualization applications in enterprises

A number of enterprises are engaged in virtualization projects that aim to gradually relocate operating systems and applications running directly on physical machines to virtual machines. The motivation is to exploit the additional VMM layer between hardware and systems software for introducing a number of new capabilities that can potentially ease the complexity and risk of managing large data centers. Here we outline some of the more compelling cases for using virtualization in large enterprises.

8.2.1 Security through virtualization

Modern data centers are all necessarily connected to the world outside via the internet and are thereby open to malicious attacks and intrusion. A number of techniques have been developed to secure these systems, such as firewalls, proxy filters, tools for logging and monitoring system activity and intrusion detection systems. Each of these security solutions can be significantly enhanced using virtualization.

For example, many intrusion detection systems (IDS) traditionally run on the network and operate by monitoring network traffic for suspicious behavior