Virtualization Introduction

1 Introduction

Hello, and welcome to this introductory lesson on Virtualization.

Virtualization has become the key enabling technology for cloud computing. Virtualization approaches and mechanisms to support them have attracted a lot of attention by system designers and hardware manufacturer in the recent year. In this part of the course we will learn what virtualization and how it does it. In this lesson we will introduce the main concepts and motivate the need for virtualization.

We will start by asking, what is virtualization. If we have used virtualization tools before we understand that virtualization has something to do with running entire systems on other systems. We will clarify the virtualization approach to do this, as opposed to, for example, functional simulation or emulation.

We will then look at what types of virtualization exist.

Central to virtualization is the Virtual Machine Monitor, typically called the "hypervisor". We will get a brief overview of what the hypervisor does.

We will conclude by describing where virtualization comes from and why it is interesting. Virtualization is not new at all. IBM has been using them in their mainframes since the very early 60's, and the venerable IBM VM operating system, named after "virtual machine" has been a great commercial success since its first deployment in 1972. Data centers are using virtualization today for some of the same reasons as in the early mainframes, but a much larger scale.

At the end of this lesson you will master the terminology used in the next lessons on virtualization, where we will explore approaches to virtualization and how to virtualize system resources, including memory and hardware devices.

Let's begin.

2 What is Virtualization?

Let's start by defining what virtualization is in the context of computer systems.

A helpful definition can be found here. A virtual machine is defined as an "efficient", "isolated" "duplicate" of the real machine.

Interestingly, this definition dates back 1974, to the seminal paper in fact by Popek and Goldberg, where they formulate a set of requirements for instruction sets to be able to support virtualization. Virtual machines are clearly not a new thing. The three highlighted terms are important:

- For a virtual machine to be a duplicate of the real machine it has to behave identically to the real machine.
- For it to be isolated, it should be able to share the real machine without interfering with other virtual machine.

- Finally, to be efficient it has to execute at a speed that is comparable to that of the real machine.

Let's have a closer look:

In order for the virtual machine to be a duplicate of the real machine, the software should not be able to distinguish between real and virtual hardware. We will see later that by "software" in this context we really mean the "operating system", which makes this requirement very difficult to satisfy.

Of course, the resources available to the software may be different than those available on the real machine, and the timing is clearly different.

We talked about the need for isolation.

In order to be able to run at speed comparable to that of the real machine, we have to run most of the instructions on the real hardware. Interpreting instructions one-by-one cannot possibly provide an efficient execution.

3 Simulation, Emulation, Virtualization

It may help to put virtualization in a context with simulation and emulation, two other mechanisms that allow the replication of at least the functional model of a system.

- **Simulation** in the context of computer systems is often viewed as limited to replicating the functional model of the system only.
- In comparison, **emulation** attempts to replicate the behavior of the real system. For example, individual instructions and their effect may be replicated. By correctly replicating the behavior of a system, the emulator manages to replicate the functional specification of the system as well. Simulators are therefore sometimes considered as high-level emulators.
- **Virtualization** on the other hand, executes most of the operations on the real hardware, and may emulate or simulate peripheral components of the system.

4 Types of Virtualization

System virtualization can happen at different levels, and we distinguish several types, based on where the virtualization happens.

A the highest level are the virtual machines that we know from "write-once-run-anywhere" development environments, such as Java or .NET. Rather than being executed on the physical machine, a Java or C# program is executed by a virtual machine, the Java Virtual Machine for example, which is really a piece of software that interprets the Java Byte Code instructions on the physical hardware. We call this process-level virtualization, because the virtualization is done inside a process.

On the other end we have "bare metal" virtualization, also called called "type-1" virtualization. In this case a hypervisor, which is a special virtualization software, creates and executes one or more virtual machines on the hardware. Each virtual machine runs its own operating system. Example of this form of virtualization are IBM's VM system and VMWare's ESXi hypervisor.

In "Hosted" or "Type-2" virtualization, there is an operating system, which runs the hypervisor as an execution. The hypervisor in turn hosts the guest virtual machines. Examples of hosted virtualization are VirtualBox and VMWare's Workstation for Windows or Fusion for MacOS. While

convenient for informal use, hosted virtualization causes significantly higher overhead, as each time the guest OS requests system services, this has to go through the hypervisor AND the host OS.

Bare-metal and hosted virtualization are both so-called "system-level" virtualization approaches, because their intent is to virtualize an entire system, to which the guest OSes have access to.

"Operating-System-level" virtualization is useful when one wants to isolate applications from each other. In this case there is no need to run guest operating systems, which makes this approach very light-weight. A well-known example of OS virtualization is Dockers.

In the following we will focus primarily on system-level virtualization.

5 Role of the Hypervisor

We have seen that a central component for system-level virtualization is the hypervisor, which is that program that runs either on the hardware directly or on the operating system to implement and run the virtual machines.

It is also called the virtual machine monitor.

While in a traditional system the operating system sits on top of the hardware directly, in bare-metal virtualization the hypervisor sits between the hardware and what now is one of possibly several guest OSes.

The role of the hypervisor could be described as that of a meta-operating-system. It partitions the hardware among the virtual machines and the rest of the system. It schedules the execution of the guest operating systems. All accesses to shared resources go through the hypervisor, and the hypervisor switches between the guest OS contexts. This is called a "world switch". A world switch between the VMM and the host worlds involves saving and restoring all user and system visible state on the CPU, and is thus more heavyweight than a normal process switch.

Since the hypervisor controls all resources, there are a few implications, which we will discuss in later lessons. First, the hypervisor must run in privileged mode, similarly to the kernel of the operating system in a traditional system. The guest OS on the other hand must run in non-privileged mode, which, as we will discuss in details, is not easy to accomplish, given that it is an operating system. The guest OS will be bound to make lots of privileged instructions, which must be intercepted in some way by the hypervisor and handled appropriately.

6 Why Virtualization?

Why would one use virtualization? Historically, it was used to enable sharing of expensive mainframes. Starting with CP-40, a very early prototype of a hypervisor in the early 60's IBM, developed a series of hypervisors. CP stands for "control program". The first commercial hypervisor was Control-Program 67 for the IBM/360. The IBM VM system, for "virtual machine" which came out in 1972 was a major technology for IBM, which ran their mainframes for a long time. VM was very flexible, and virtual machines could run all kinds of guests, (5) such as very light-weight single-user systems, primarily CMS, or mainstream OS, such as MVS, AIX, which is the IBM version of UNIX, and mainframe Linux. or second level VM systems, to basically stack virtual machines on top of each other, or specialized OS subsystems, which typically managed shared resources, such as shared files, printer spooling, and so on. Overall, VM was ideally suited to support all kinds of legacy systems. All one needed was a virtual machine to host the legacy system, after which it could be ported to newer-generation machines.

Other than for mainframes, virtualization largely disappeared in the 80's. There were two reasons for this:

1. Time sharing, which was implemented by running light-weight single-user OS such as CMS in virtual machines, became natively supported by other operating systems, such as UNIX.

2. There was no interest in consolidating multiple machines onto a single physical machine, in the way many operators did with IBM's VM. The emergence of mini-computer and later personal computers made hardware simply too cheap to worry about consolidation.

Since the early 2000's, virtualization has gone through a major revival, primarily due to its ability to provide isolation of virtual machines from each other.

The isolation provided by running applications in separate virtual machines, provides great quality of service, as one can provision each virtual machine with an amount of memory, and scheduling algorithms in the hypervisor are able to fairly allocate CPU bandwidth to virtual machines.

The security is improved as well, since there is very controlled resource sharing across machines.

Finally, the hypervisor is able to run multiple, different OSs concurrently.

The state of the guest OS is completely encapsulated and can be safely stored to a file. The state can be later loaded from the file and restored.

The file containing the state can be distributed and so the virtual machine replicated.

Similarly, the virtual machine can be migrated by saving it to file and restoring it on a different host. Multiple virtual machines can be consolidated on a single host in this way.

By saving the state of a virtual machine to stable storage, one de-facto generates a checkpoint. If the system crashes, one can go back to the last saved state.

Finally, encapsulation supports debugging as well.

Guernot Heiser, a well-known researcher in operating systems and virtualization, once famously stated that having virtualization for these reasons would be mostly unnecessary, if OSes were designed correctly to do their job.

Another important reason to use virtualization today is cloud computing.

Cloud data center providers run large numbers of machines to host virtual machines that in turn are populated by paying customers. In this example, two independent companies consolidate their systems by packaging them as virtual machines and having the cloud provider host them as guest on the hypervisors in their systems.

The cloud provider can host the virtual machines in a cost-effective manner by significantly over-allocating the resources and relying on so-called statistical multiplexing to satisfy the customers. Taking advantage of Statistical Multiplexing means that one relies on the fact that not all clients request the maximum use of their resource at the same time. If they do, they briefly experience poor service. This happens sufficiently rarely to warrant the risk, in particular if risk can be compensated by an appropriate discount.

The cost of maintenance is reduced because of the large scale at which cloud providers operate. Economy of scale allows for the efficient operations of the hosts. In particular, the operation of large numbers of hosts allows for more efficient paradigms. Sometimes this is called "cattle vs. pets", where large-scale operations use a cattle approach, where individual hosts are operated with extremely little overhead, while a "pet" paradigm would require a high-maintenance way of operating hosts.

On-demand provisioning is another great advantage of using cloud providers. As the demand for computing grows, the cloud provider can easily replicate the guest across the data center to accommodate the additional demand. If the demand subsides, the workload can be consolidated back to a few guests.

Similarly, the cloud operator can easily scale or balance the load by migrating virtual machines across the data center.

6.1 Summary

We have come to the conclusion of this introductory lesson on Virtualization.

We have clarified what virtualization is, and we have explored different types of virtualization.

We have learned about the role of the hypervisor, which is a central component of virtualization.

We have learned about history and motivation for virtualization.

After this introduction we will be exploring the mechanics of virtualization. The main problem is to run operating system kernel, which typically run in kernel mode, as guests on top of hypervisors, which means that they have to run in non-privileged mode. Similarly, we will have to re-visit how memory management and device management can be done in an environment where the guest Oses and the hypervisor manage these resources, sometimes with conflicting goals.

Thank you for watching.