Cloud computing Important topics

1. Architecture level for virtualization

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively.

The virtualization software creates the abstraction of VMs by interposing virtualization layer at various levels of a computer system.

Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level (see Figure 3.2)

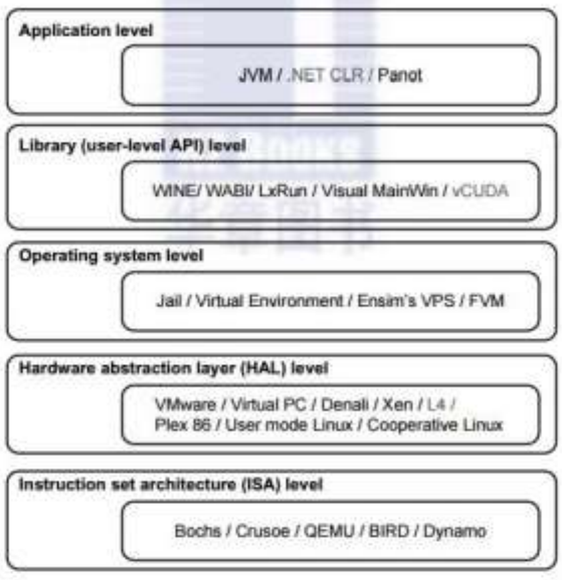
Application level
JVM / .NET CLR / Panot
Library (user-level API) level
WINE/ WABI/ LxRun / Visual MainWin / vCUDA
Operating system level
Jail / Virtual Environment / Ensim's VPS / FVM
Hardware abstraction layer (HAL) level
VMware / Virtual PC / Denali / Xen / L4 / Plex 86 / User mode Linux / Cooperative Linux
Instruction set architecture (ISA) level
Bochs / Crusoe / QEMU / BIRD / Dynamo

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.

1 Instruction Set Architecture Level

- → Instruction Set Architecture (ISA) refers to the set of instructions that a particular CPU (Central Processing Unit) understands and can execute.
- → It serves as an interface between the software and the hardware of a computer system.
- → ISAs can vary significantly between different CPU architectures. Common examples of ISAs include x86 (used in most personal computers), ARM (common in mobile devices and embedded systems), MIPS, PowerPC, and RISC-V, among others.
- → The basic emulation method is through code interpretation.
- → For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions.

2 Hardware Abstraction Level

- → Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM.
- → The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices.
- → The idea was implemented in the IBM VM/370 in the 1960s.
- → More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS applications.

3 Operating System Level

- → At the operating system level, the focus shifts from hardware to software, specifically the software responsible for managing hardware resources and providing services to applications.
- → OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.
- → The kernel is responsible for managing hardware resources such as the CPU, memory, and input/output devices.
- → Operating systems provide user interfaces that allow users to interact with the system and applications.

4 Library Support Level

- → Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS.
- → Library support at the software level provides reusable code modules and functions that simplify development tasks for programmers.
- → Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks.
- → The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts.

5. User-Application Level

- → Virtualization at the application level virtualizes an application as a VM. On a traditional OS,an application often runs as a process.
- → Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL).
- → The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM.
- → Other forms of application-level virtualization are known as application isolation, application sandboxing, or application streaming.

2. Memory virtualization in two level mapping diagram

- → Virtual memory virtualization is similar to the virtual memory support provided by modern operat-ing systems.
- → The operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory.
- → All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.
- → Virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.
- → That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory.
- → The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs.
- → The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.

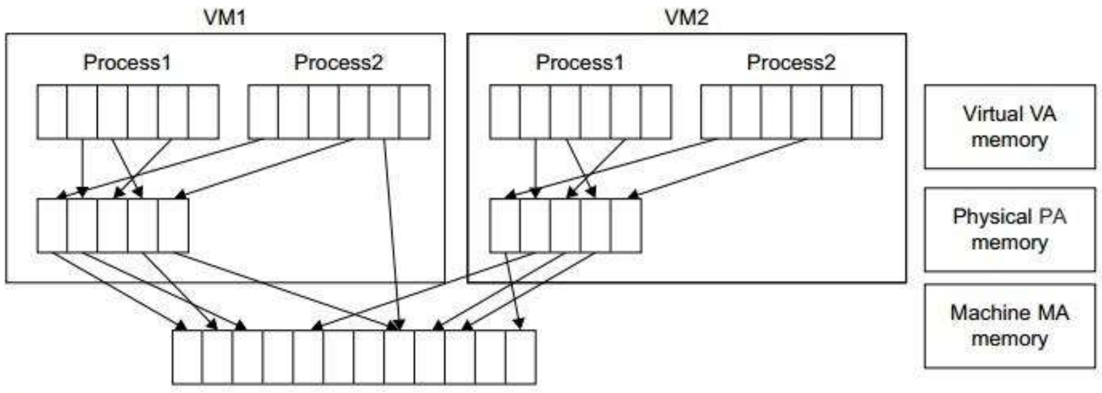

FIGURE 3.12

Two-level memory mapping procedure.

→ Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table.

→ Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS.

→ Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor.

→ Consequently, the performance overhead and cost of memory will be very high.

→ When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup.

→ The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007.

→ It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

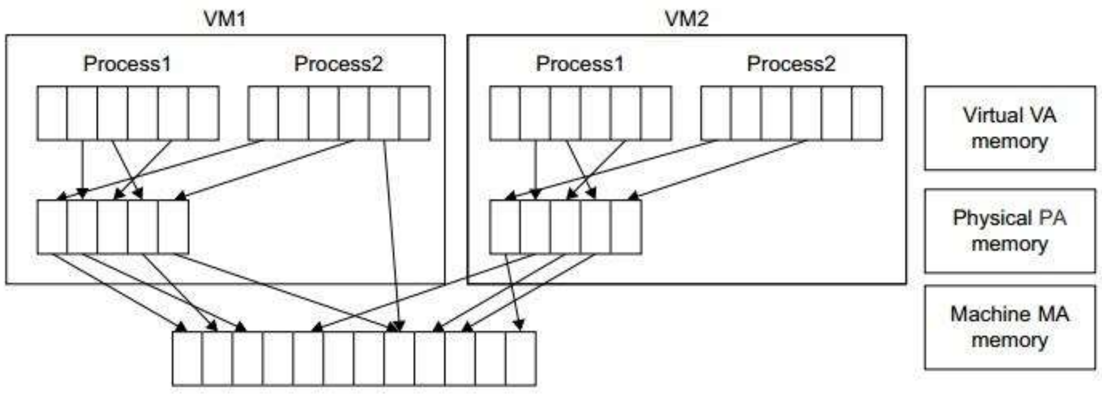

FIGURE 3.12

Two-level memory mapping procedure.

3.CPU Virtualization

- → A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode.
- → Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency.
- → The critical instructions are divided into three categories: privileged instructions, control-sensitive instructions, and behavior-sensitive instructions.
- → Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.
- → A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode.
- → This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions. When these instruc-tions execute in virtualization, they cannot be trapped in the VMM.
- → On a native UNIX-like system, a system call triggers the 80h interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call.
- → Almost at the same time, the 82h interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well.
- → Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

4.Para virtualization with compiler support and para virtualization architecture

Para-Virtualization with Compiler Support:

- → Para-virtualization needs to modify the guest operating systems.
- → A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system.
- → The virtualization layer can be inserted at different positions in a machine soft-ware stack.

- → Figure 3.7 illustrates the concept of a paravirtualized VM architecture. The guest operating systems are para-virtualized.
- → They are assisted by an intelligent compiler to replace the nonvirtualizable OS instructions by hypercalls as illustrated in Figure 3.8.The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3.

para virtualization architecture:

- → When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS.
- → In Figure 3.8, we show that para virtualization replaces nonvirtualizable instructions with hypercalls that communicate directly with the hypervisor or VMM.
- → Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well.
- → Finally, the performance advantage of para-virtualization varies greatly due to workload variations.
- → The main problem in full virtualization is its low performance in binary translation. To speed up binary translation is difficult.
- → Therefore, many virtualization products employ the para-virtualization architecture. The popular Xen, KVM, and VMware ESX are good examples.

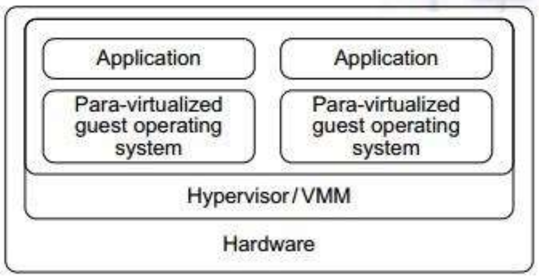
Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor/VMM
Hardware

FIGURE 3.7

Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization process (See Figure 3.8 for more details.)

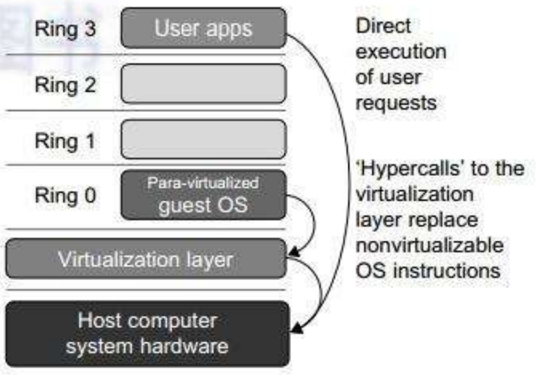
Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

FIGURE 3.8

The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls.

(Courtesy of VMWare [71])

5.Design and explain the architecture of Xen

- → Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism.
- → The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure 3.5.
- → It just provides a mechanism by which a guest OS can have direct access to the physical devices. As a result, the size of the Xen hypervisor is kept rather small.
- → Xen provides a virtual environment located between the hardware and the OS.
- → A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].
- → The core components of a Xen system are the hypervisor, kernel, and applications. The organi-zation of the three components is important.

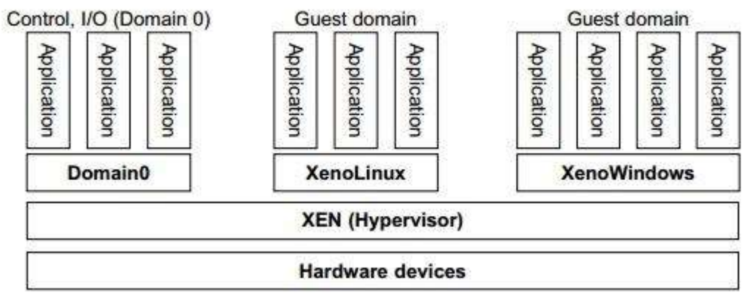

FIGURE 3.5

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

- → The guest OS, which has control ability, is called Domain 0, and the others are called Domain U.
- → It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices.
- → Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).
- → Its management VM is named Domain 0, which has the privilege to manage other VMs implemented on the same host.
- → Traditionally, a machine's lifetime can be envisioned as a straight line where the current state of the machine is a point that progresses monotonically as the software executes.

→ VMs are allowed to roll back to previous states in their execution (e.g., to fix configuration errors) or rerun from the same point many times (e.g., as a means of distributing dynamic content or circulating a “live” system image).

6. Full virtualization

→ With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software.

→ Both the hypervisor and VMM approaches are considered full virtualization.

→ This is because binary translation can incur a large performance overhead.

→ Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do.

→ Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security

7. Binary host based virtualization

→ An alternative VM architecture is to install a virtualization layer on top of the host OS.

→ This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer.

→ Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly.

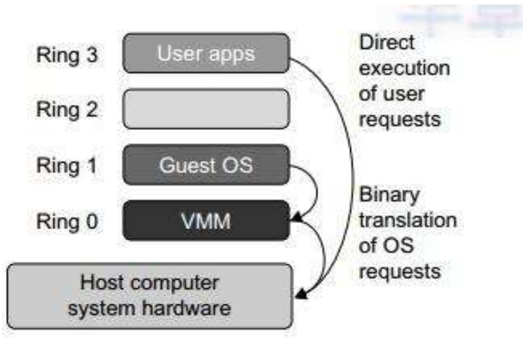
Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

FIGURE 3.6

Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host.

→ This host-based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS.

→ The virtualizing software can rely on the host OS to provide device drivers and other low-level services.

→ Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low.

→ When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly.