RAMAIAH
Institute of Technology

Cloud Computing and Big Data

Subject Code: CS71 (Credits: 4:0:0)

Textbook:

1. Cloud Computing Theory and Practice – **DAN C**. Marinescu – Morgan Kaufmann Elsevier.
2. Cloud Computing A hands - on approach – Arshdeep Bahga & **Vijay madisetti** Universities press
3. Big Data Analytics, Seema Acharya and Subhashini Chellappan. 2nd edition, Wiley India Pvt. Ltd. 2019

NOTE: I declare that the PPT content is picked up from the prescribed course text books or reference material prescribed in the syllabus book and Online Portals.

Unit II

Cloud Resource Virtualization:

- Layering and virtualization, Virtual machine monitors, Virtual machines,
- VM Performance and security isolation,
- Virtualization types, Hardware support for virtualization,
- A performance comparison of virtual machines,
- The darker side of virtualization, Software fault isolation.

Cloud Resource Management and Scheduling:

- Policies and mechanisms for resource management,
- Resource bundling, combinatorial auctions for cloud
- Scheduling algorithms for computing clouds,
- Fair queuing, start time fair queuing,
- Borrowed virtual time
- Resource management and application scaling

RAMAIAH Institute of Technology

Motivation

Three fundamental abstractions are necessary to describe the operation of a computing systems:

(1) processors, (2) memory, (3) communications links and Storage

- As the scale of a system and the size of its users grows, it becomes very challenging to manage its recourses
- **Resource management issues:**
  - provision for peak demands → overprovisioning
  - heterogeneity of hardware and software
  - machine failures

- **Virtualization is a basic enabler of Cloud Computing, it simplifies the management of physical resources for the three abstractions**

  - For example, the state of a virtual machine (VM) running under a virtual machine monitor (VMM) **can de saved and migrated to another server** to balance the load
  - For example, virtualization **allows users to operate in environments they are familiar with**, rather than forcing them to specific ones

RAMAIAH Institute of Technology

Virtualization

- "Virtualization, in computing, refers to **the act of creating a virtual (rather than actual) version of something,** including but not limited to a virtual computer hardware platform, operating system (OS), storage device, or computer network resources."

- Virtualization **abstracts the underlying resources**; simplifies their use; **isolates users from one another**; and **supports replication** **which increases the elasticity of a system**

- In other words, Virtualization is a technique, which allows to **share a single physical instance of a resource or an application** among multiple customers and organizations.

    - Virtualization is the process of converting a **physical IT resource into a virtual IT** resource.

    - Virtualization is a **partitioning of single physical server into multiple logical servers**.
        - Once the physical server is divided, each logical server behaves like a physical server and **can run an operating system and applications independently.**

RAMAIAH
Institute of Technology

Benefits of Virtualization Technology :

- Efficient Usage of Physical Computing Resources
- Increased Availability
- Easy System and Application Testing
- Simple Administration and Less Power Consumption
- Faster Failure Recovery

Issues in Virtualization :

- Server Sharing Performance Issues
- Increased Networking Complexity

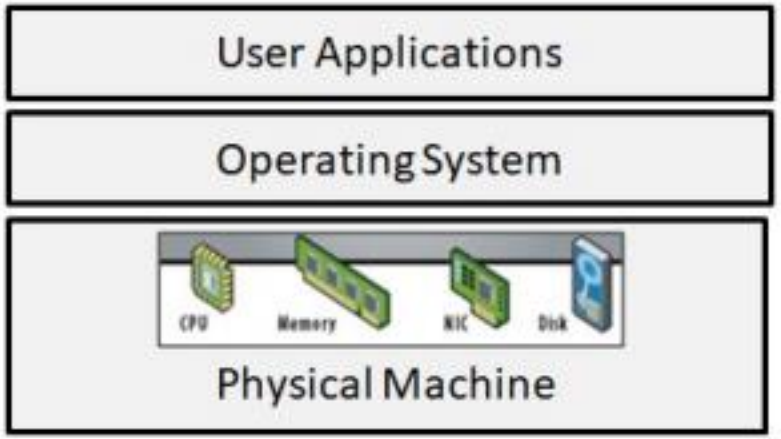


Figure 2.1 Normal Scenario (without virtualization) [33]

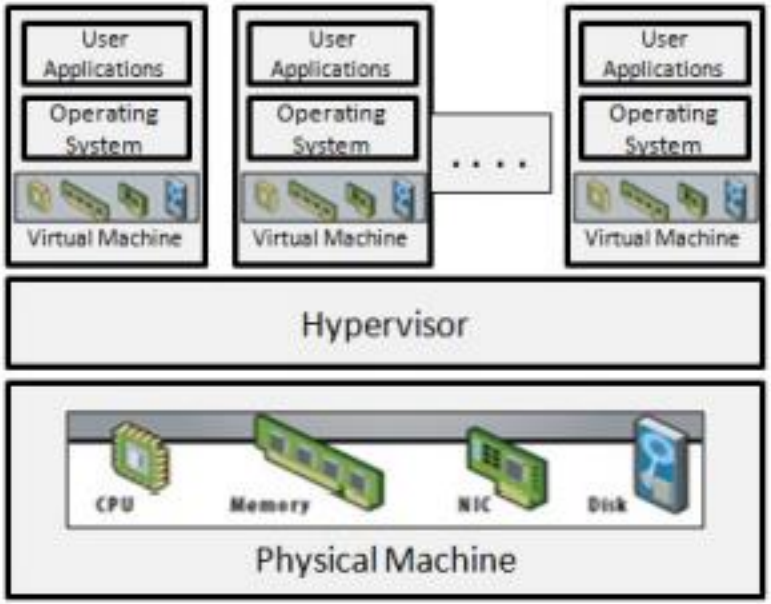


Figure 2.2 Virtualized Environments [33]

RAMAIAH Institute of Technology

Different types of Virtualization

There are many different types of virtualization, including **hardware, software, desktop, memory, storage, data and network virtualization.**

- **Hardware virtualization** is one of the most common types of virtualization. It is synonymous with **platform virtualization**, which occurs through the creation of a virtual machine that behaves **like a real computer or computer operating system.**

- **Desktop virtualization** is also another widely used type of virtualization that creates and stores a client's desktop on a server **that can be remotely accessed by the client over a network.**

- **Software virtualization** is yet another commonly used type of virtualization that allows different versions of an operating system to coexist and run on the same physical machine, providing the ability to run applications in different environments **without the need to invest in additional hardware.**

- **Network Virtualization**: It is a method of combining the available resources in a network **by splitting up the available bandwidth into channels**, each of which is independent from the others and each channel is independent of others and can be assigned to a specific server or device in real time.

- **Storage Virtualization**: It is the pooling of physical storage from multiple network storage devices into what appears to be a single storage device that is managed from a central console. **Storage virtualization is commonly used in storage area networks (SANs).**

- **Server Virtualization**: Server virtualization is the masking of server resources like processors, RAM, operating system etc, from server users. The intention of server virtualization **is to increase the resource sharing and reduce the burden and complexity of computation from users.**

Resource management for a community of users with a wide range of applications running **under different operating systems is a very difficult problem**.

- Resource management becomes even more complex when resources are **oversubscribed and users are uncooperative**.
- In addition to external factors, resource management is affected by internal factors, such as the **heterogeneity of the hardware and software systems,**
- The ability to approximate the global state of the system and to **redistribute the load, the failure rates of different components**, and many other factors.

- The traditional solution for a data center is to **install standard operating systems on individual systems and rely on conventional OS techniques** to ensure resource sharing, application protection, and performance isolation.
- The alternative is **Resource Virtualization**, a technique analyzed in this chapter.

RAMAIAH Institute of Technology

Virtualization

Virtualization is a basic tenet of cloud computing – that simplifies some of the resource management tasks.

For example,

- The state of a **virtual machine (VM)** running under a **virtual machine monitor (VMM)** can be saved and migrated to another server to balance the load.

- At the same time, **virtualization allows users to operate in environments with which they are familiar** rather than forcing them to work in idiosyncratic environments.

- **Resource sharing** in a virtual machine environment requires not only ample hardware support and, in particular, powerful processors but also architectural support for multilevel control.

- Indeed, resources such as **CPU cycles, memory, secondary storage, and I/O and communication bandwidth are shared among several virtual machines**; for each VM, resources must be **shared among multiple instances of an application.**

Virtualization Simulates the interface to a physical object by:

- **Multiplexing**: creates multiple virtual objects from one instance of a physical object. Example - a processor is multiplexed among a number of processes or threads.
- **Aggregation:** creates one virtual object from multiple physical objects. Example - a number of physical disks are aggregated into a RAID disk.

- **Emulation:** constructs a virtual object from a different type of a physical object. Example - a physical disk emulates a Random Access Memory (RAM).

- **Multiplexing and emulation**. Examples - virtual memory with paging multiplexes real memory and disk; a virtual address emulates a real address.

Cloud resource virtualization is important for:

- **System security**, as it allows isolation of services running on the same hardware.
- **Performance and reliability,** as it allows applications to migrate from one platform to another.

RAMAIAH Institute of Technology

Layering & Virtualization

- A common approach to manage system complexity is to identify a set of Layers with well defined interfaces among them.
- **Interfaces separate different levels of abstraction**

Layering – a common approach to manage system complexity.

- Minimizes the **interactions among the subsystems** of a complex system.
- **Simplifies the description of the subsystems;** each subsystem is abstracted through its interfaces with the other subsystems.
- We are able to **design, implement, and modify the individual subsystems independently.**

Layering in a computer system.

- Hardware.
- Software.
  - Operating system.
  - Libraries.
  - Applications.

RAMAIAH
Institute of Technology

Layering and Interfaces between layers in Computer System

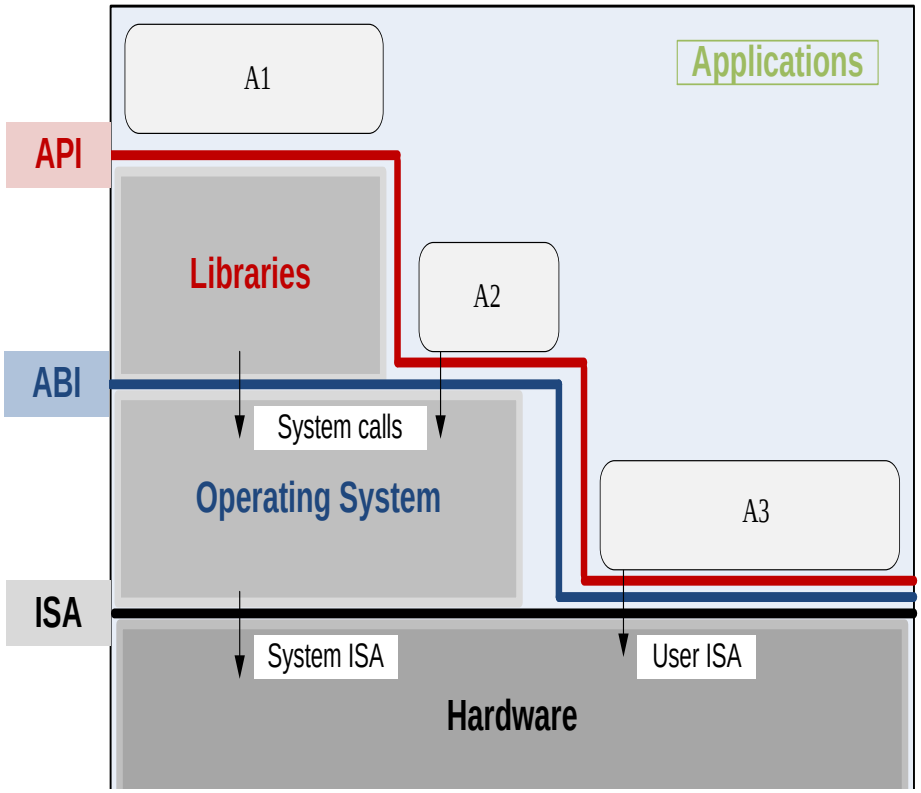


FIGURE 5.1

Layering and interfaces between layers in a computer system. The software components, including applications, libraries, and operating system, interact with the hardware via several interfaces: the *application programming interface* (API), the *application binary interface* (ABI), and the *instruction set architecture* (ISA). An application uses library functions (A1), makes system calls (A2), and executes machine instructions (A3).

RAMAIAH
Institute of Technology

Types of Interfaces

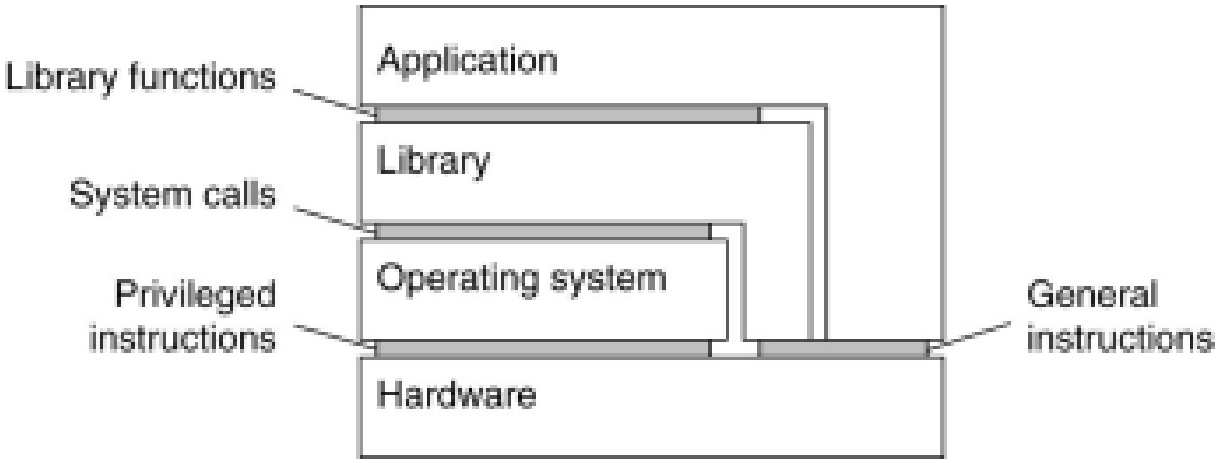


Figure 5.1, which shows the **interfaces among the software components and the hardware.**

- **The hardware consists** of one or more multicore processors, a system interconnect (e.g., one or more buses), a memory translation unit, the main memory, and I/O devices, including one or more networking interfaces.
- **Applications written mostly in high-level languages (HLL)** often call library modules and are compiled into **object code.**
- **Privileged operations,** such as I/O requests, cannot be executed in user mode; instead, **application and library modules issue system calls** and
- The operating system determines whether the privileged operations required by the application do not violate system security or integrity and, if they don't, executes them on behalf of the user.
- **The binaries resulting from the translation of HLL programs are targeted to a specific hardware architecture.**

Instruction Set Architecture (ISA) – at the boundary between hardware and software.
It defines a processors set of instructions. Ex: Intel architecture is represented by x86-32 and x86-64 instruction sets for systems supporting 32-bit addressing and 64-bit addressing.

- The Hardware supports two execution mode:
  - **Privileged or Kernel mode** and
  - a **User Mode.**
- The Instruction Set consist of two sets:
  - **Sensitive Instructions,**
  - **Non Privileged Instructions**.

Application Binary Interface (ABI) – which allows the group consisting of the application and the library modules to access the hardware; the **ABI does not include privileged system instructions, instead it invokes system calls.**

Application Program Interface (API) - defines the set of instructions the hardware was designed **to execute and gives the application access to the ISA**; it includes HLL library calls which often invoke system calls.

RAMAIAH Institute of Technology

Code Portability

- A **process** is the abstraction for the code of an application at execution time;
- A **thread** is a lightweight process.
    - The **ABI** is the projection of the computer system seen by the process, and
    - The **API** is the projection of the system from the perspective of the HLL program.

- **The binaries created by a compiler for a specific ISA and a specific operating system are not portable.** Such code cannot run on a computer with a different ISA or on computers with the same ISA but different operating systems.

- **However, it is possible to compile an HLL program for a VM environment, as shown in Figure 5.2, where portable code is produced** and distributed and then converted by binary translators to the ISA of the host system.

    - A **dynamic binary translation converts blocks of guest instructions from the portable code to the host instruction** and leads to a significant performance improvement as such blocks are cached and reused.

RAMAIAH
Institute of Technology

Code Portability

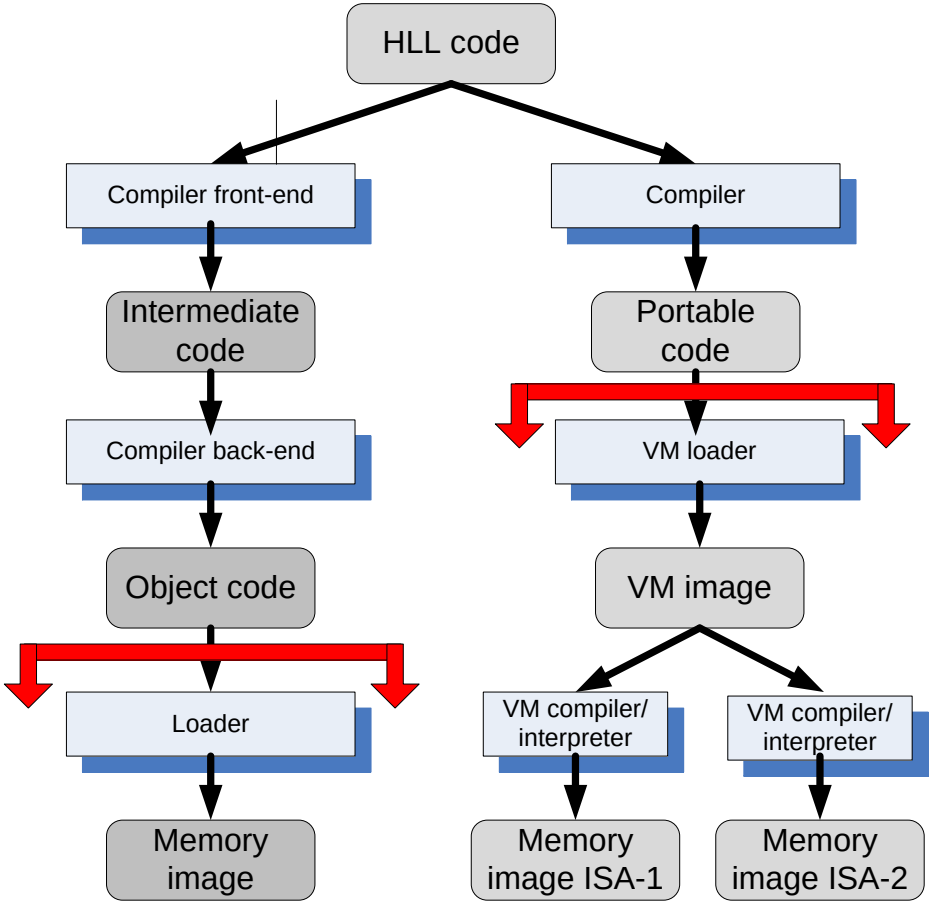


FIGURE 5.2

High-level language (HLL) code can be translated for a specific architecture and operating system. HLL code can also be compiled into portable code and then the portable code translated for systems with different ISAs. The code that is shared/distributed is the object code in the first case and the portable code in the second case.

RAMAIAH Institute of Technology

(VMM / hypervisor)

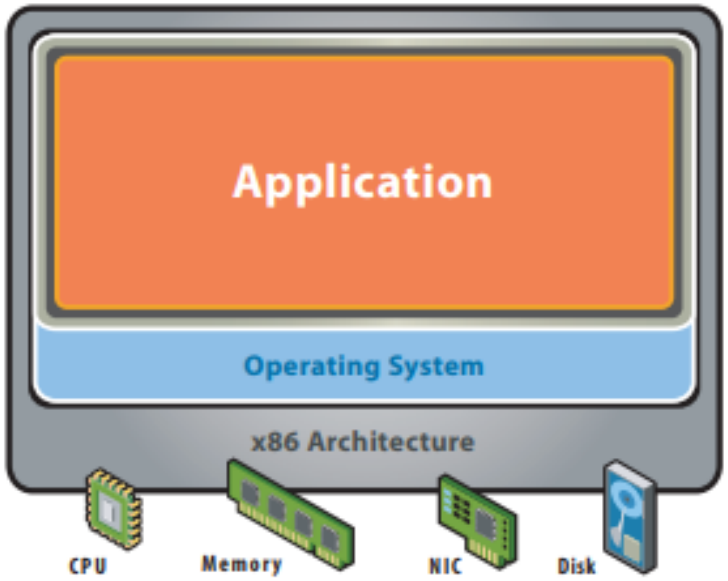
VMM is the software that securely **partitions the resources of a computer system** into **one or more virtual machines (VMs).**

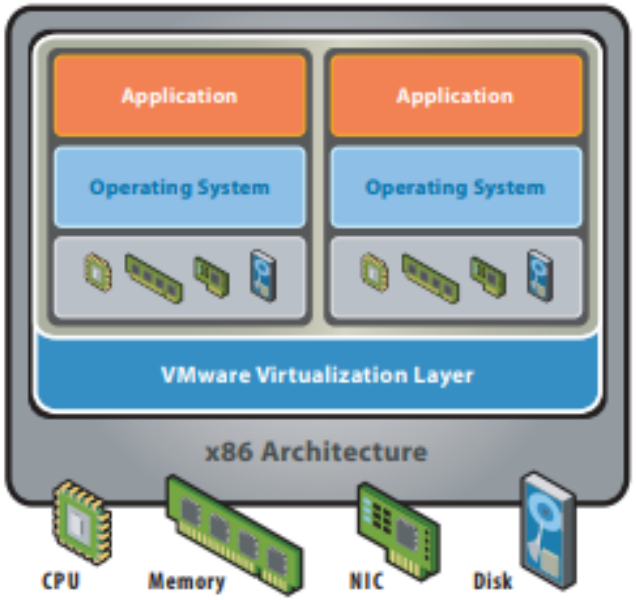
- **A guest OS** is an OS that runs under the control of a VMM rather than directly on the hardware.
- **VMM runs in kernel mode**, a **guest OS runs in user mode**.
- Allows several operating systems to **run concurrently on a single hardware platform**; at the same time.
- VMM **controls how the guest OS uses the hardware resources**.
- Events occurring in one VM do not affect other VM running under same VMM.

- At the same time the VMM enables
  - **Multiple services to share the same platform.**
  - **Live migration - the movement of a server from one platform to another.**
  - **System modification while maintaining backward compatibility with the original system.**
  - **Enforces isolation among the systems, thus security.**

RAMAIAH
Institute of Technology

Virtualization Approaches





Before Virtualization:

- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine often creates conflict
- Underutilized resources
- Inflexible and costly infrastructure

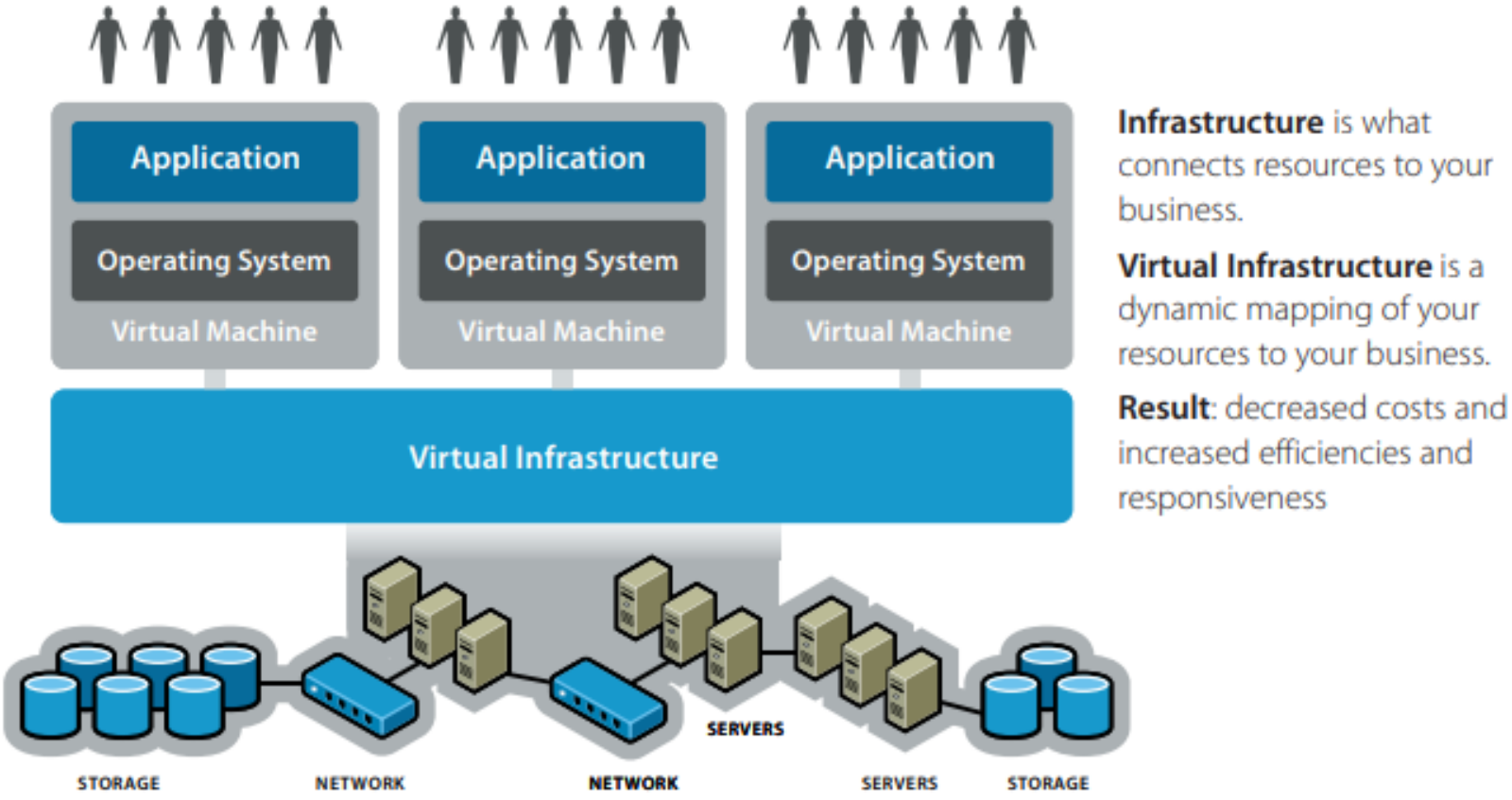
After Virtualization:

- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and application as a single unit by encapsulating them into virtual machines

[Reference]https://www.vmware.com/pdf/virtualization.pdf

RAMAIAH
Institute of Technology

Virtualization Approaches



Infrastructure is what connects resources to your business.

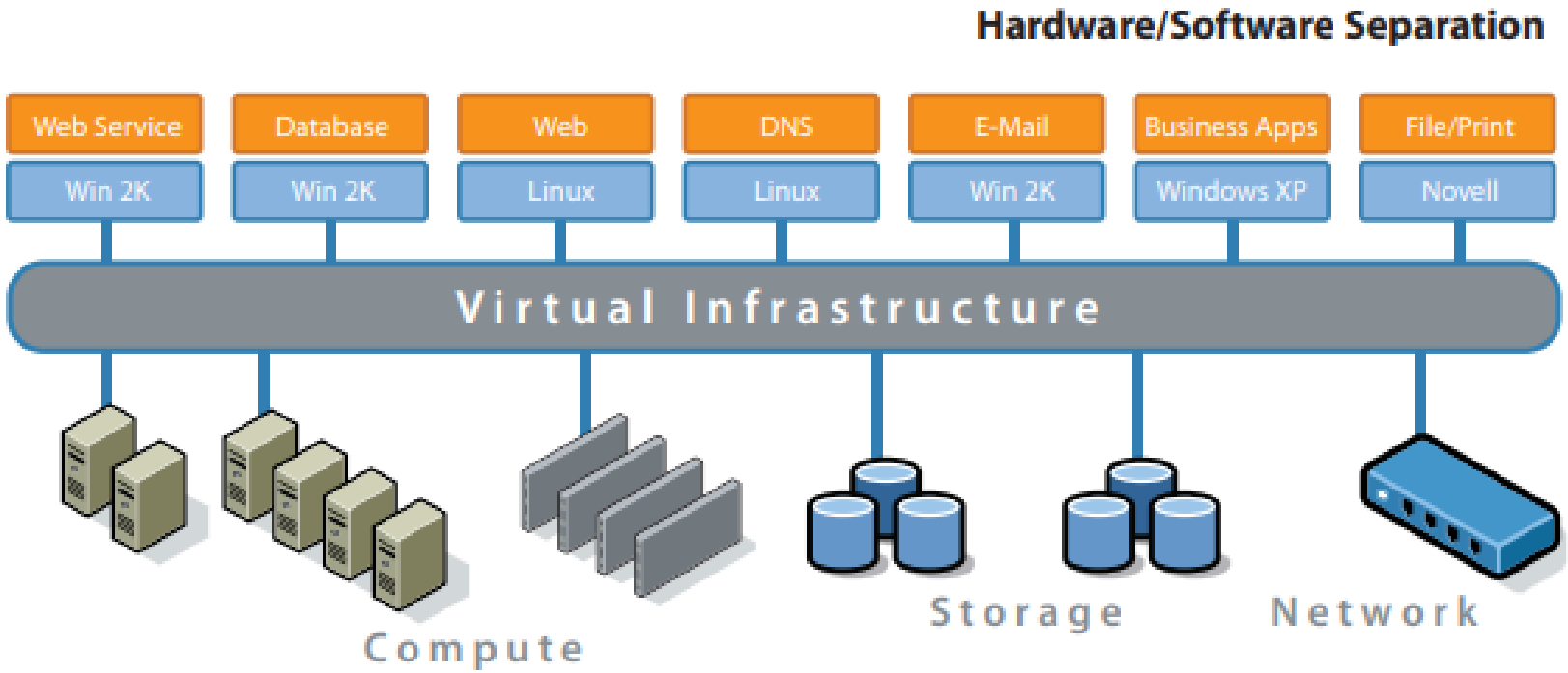
Virtual Infrastructure is a dynamic mapping of your resources to your business.

Result: decreased costs and increased efficiencies and responsiveness

[Reference]https://www.vmware.com/pdf/virtualization.pdf

RAMAIAH
Institute of Technology

Virtualization Approaches



[Reference]https://www.vmware.com/pdf/virtualization.pdf

VMM virtualizes the CPU and the memory

- Traps the privileged instructions executed by a guest OS and **enforces the correctness and safety of the operation.**
- Traps interrupts and dispatches them to the individual guest operating systems.

VMM Controls the virtual memory management.

- **Maintains a shadow page table for each guest OS** and replicates any modification made by the guest OS in its own shadow page table.
- This shadow page table points to the actual page frame and it is used by the Memory Management Unit (MMU) for dynamic address translation.
- **Monitors the system performance and takes corrective actions to avoid performance degradation.** For example, the VMM may swap out a Virtual Machine to avoid thrashing.

RAMAIAH Institute of Technology

Virtual machines (VMs)

VM – is an isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources.

- Each VM appears to be running on the **bare (native)hardware**, giving the appearance of multiple instances of the same computer**, though all are supported by a single physical system.**

A virtual machine (VM) is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as **running applications and programs like a separate computer.**

- A virtual machine, usually known as a guest is created within another computing environment **referred as a "host."**
- **Multiple virtual machines can exist within a single host at one time.**
- **A virtual machine is also known as a guest.**

RAMAIAH Institute of Technology

Two types of VM:

Process & System VMs.

Process Virtual Machine: A process VM is an virtual platform **created for an individual process** and destroyed once the process terminates.

- Virtually all operating systems provide **a process VM** for each one of the applications running, but the more interesting process VMs are those that support binaries compiled on a different instruction set.

System Virtual Machines: A system VM supports an operating system together **with many user processes**. When the VM runs under the control of a normal OS and provides a platform-independent host for a single application, we have an application virtual machine (e.g., Java Virtual Machine [JVM]).

- **A system virtual machine provides a complete system; each VM can run its own OS, which in turn can run multiple applications**.

- Operating system-level virtualization allows a physical server to run multiple isolated operating system instances, subject to several constraints; the instances are known as containers, virtual private servers (VPSs), or virtual environments (VEs).

Two types of VM: process & system VMs.

System Virtual Machines:

- A system platform that **supports the sharing of the host computer's physical resources between multiple virtual machines, each running with its own copy of the operating system.**

- **The virtualization technique is provided by a software layer known as a hypervisor**, which can run either on bare hardware or on top of an operating system.

Process Virtual Machine:

- Designed to provide a **platform-independent programming environment** that masks the information of the underlying hardware or operating system and allows program execution to take place in the same way on any given platform

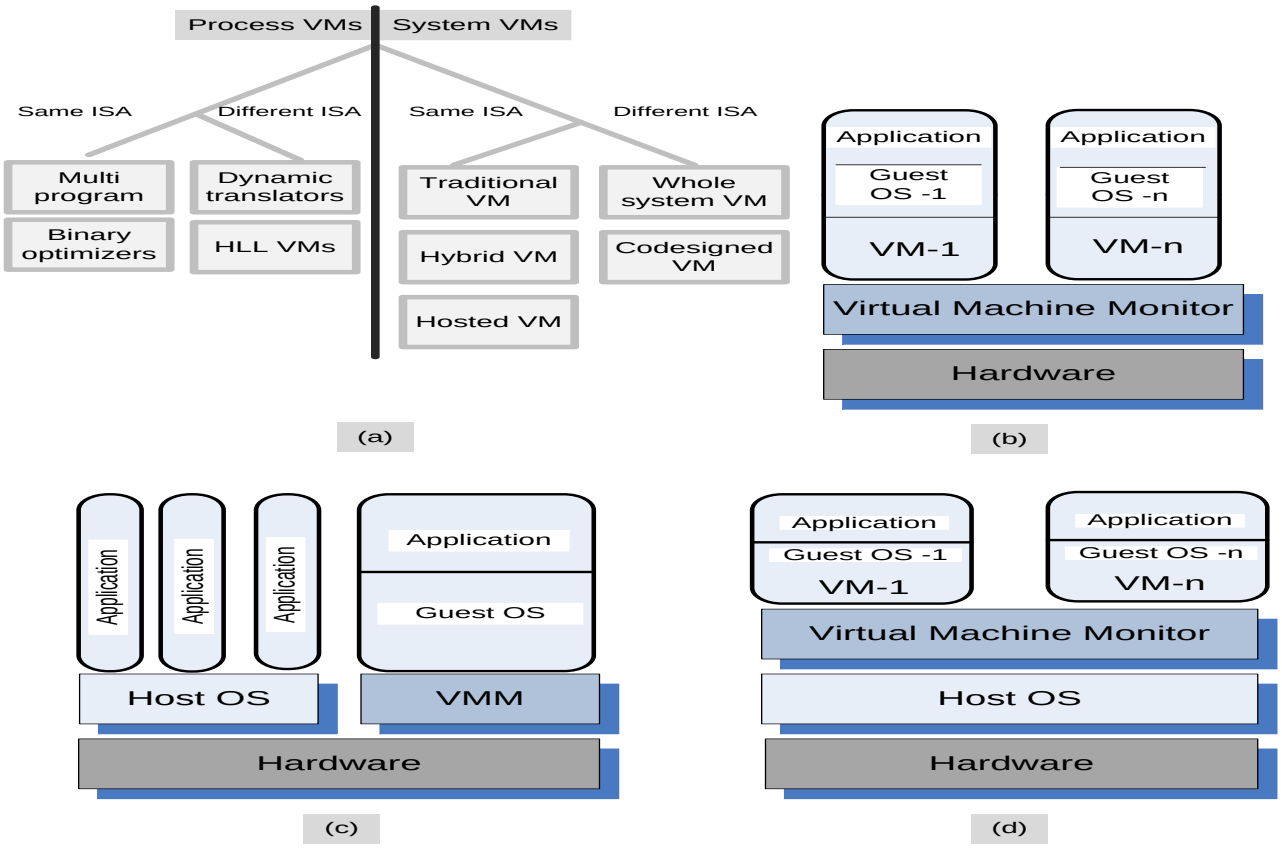


FIGURE 5.3

(a) A taxonomy of process and system VMs for the same and for different ISAs. Traditional, hybrid, and hosted are three classes of VM for systems with the same ISA. (b) Traditional VMs. The VMM supports multiple VMs and runs directly on the hardware. (c) A hybrid VM. The VMM shares the hardware with a host operating system and supports multiple virtual machines. (d) A hosted VM. The VMM runs under a host operating system.

Name	Host ISA	Guest ISA	Host OS	guest OS	Company
Integrity VM	*x86*-64	*x86*-64	HP-Unix	Linux,Windows HP Unix	HP
Power VM	Power	Power	No host OS	Linux, AIX	IBM
z/VM	z-ISA	z-ISA	No host OS	Linux on z-ISA	IBM
Lynx Secure	*x86*	*x86*	No host OS	Linux, Windows	LinuxWorks
Hyper-V Server	*x86*-64	*x86*-64	Windows	Windows	Microsoft
Oracle VM	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows	Oracle
RTS Hypervisor	*x86*	*x86*	No host OS	Linux, Windows	Real Time Systems
SUN xVM	*x86*, SPARC	same as host	No host OS	Linux, Windows	SUN
VMware EX Server	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows Solaris, FreeBSD	VMware
VMware Fusion	*x86*, *x86*-64	*x86*, *x86*-64	MAC OS *x86*	Linux, Windows Solaris, FreeBSD	VMware
VMware Server	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Workstation	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Player	*x86*, *x86*-64	*x86*, *x86*-64	Linux Windows	Linux, Windows Solaris, FreeBSD	VMware
Denali	*x86*	*x86*	Denali	ILVACO, NetBSD	University of Washington
Xen	*x86*, *x86*-64	*x86*, *x86*-64	Linux Solaris	Linux, Solaris NetBSD	University of Cambridge

RAMAIAH Institute of Technology

Performance and security isolation

- **Performance isolation** - **a critical condition for QoS guarantees in shared computing environments.**

  - **Process virtualization** presents multiple copies of the same process or multicore systems. The code is executed directly by the hardware. Whereas, **process emulation** presents **a model of another hardware system in which instructions are "emulated "** in software more slowly than virtualization.

  - **Processor emulation** is a virtualization technology that **allows software compiled for one processor/operating system to run on a system with a different processor/operating system,** without any source code or binary changes.

Performance and security isolation

The run-time behavior of an application is affected by other **applications running concurrently** on the same platform and competing for CPU cycles, cache, main memory, disk and network access. Thus, **it is difficult to predict the completion time**!

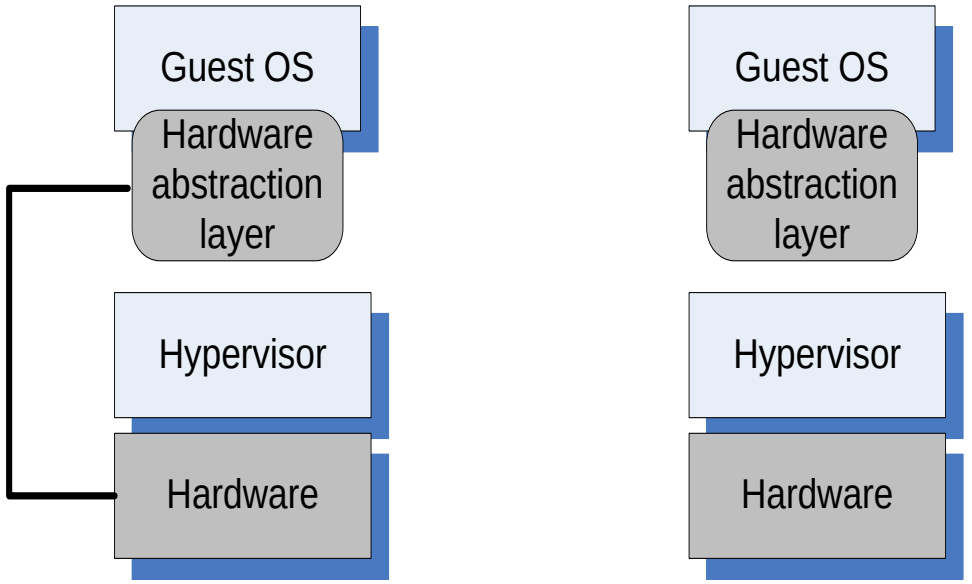
- **A VMM is a much simpler and better specified system than a traditional operating system.**
    - Example - Xen has approximately 60,000 lines of code; Denali has only about half: 30,000
- **The security vulnerability of VMMs is considerably reduced as the systems expose a much smaller number of privileged functions.**
    - For example, Xen VMM has 28 hypercalls while Linux has 100s of system calls

RAMAIAH Institute of Technology

Full virtualization and Par-virtualization

- Full virtualization ,in which each virtual machine runs on **an exact copy** of the actual hardware.
- **(Guest OS is unaware that its in a virtualized environment)**
- **Example: Vmware**

- Paravirtualization , in which each virtual machine runs on **a slightly modified copy** of the actual hardware.
- **(Guest OS is already aware that they are shared hardware)**
- **Example: Xen**



(a) Full virtualization (b) Paravirtualization

FIGURE 5.4

(a) Full virtualization requires the hardware abstraction layer of the guest OS to have some knowledge about the hardware. (b) Paravirtualization avoids this requirement and allows full compatibility at the application binary interface (ABI).

RAMAIAH
Institute of Technology

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 non virtualizable instructions, one could resort to two strategies:
  - **Binary translation.** The VMM monitors the execution of guest operating systems; **non virtualizable instructions executed by a guest operating system are replaced with other instructions**.
  - **Para virtualization**. 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

- **Some aspects of the hardware cannot be virtualized;**
- **To improve performance; and**
- **To present a simpler interface.**

RAMAIAH Institute of Technology

Full virtualization

- Guest operating systems are unaware of each other
- Provide support for unmodified guest operating system.
- **Hypervisor directly interact with the hardware such as CPU, disks.**
- **Hypervisor allow to run multiple OS simultaneously on host computer.**
- Each guest server run on its own operating system

Few implementations: **Oracle's Virtaulbox , VMware server, Microsoft Virtual PC**

Advantages:

- This type of virtualization **provide best isolation and security** for Virtual machine.
- Truly isolated **multiple guest OS can run simultaneously** on same hardware.
- It's only option that requires no hardware assist or OS assist to virtualize sensitive and privileged instructions.

Limitations:

- Full virtualization is **usually bit slower ,because of all emulation**.
- **Hypervisor contain the device driver** and it might be difficult for new device drivers to be installer by users

RAMAIAH Institute of Technology

Para virtualization

- Unlike full virtualization ,**guest servers are aware of one another.**
- Hypervisor does not need large amounts of processing power to manage guest OS.
- The entire system work as a cohesive unit.

Advantages:

- As a guest OS can directly communicate with hypervisor
- **This is efficient virtualization.**
- Allow users to make use of new or modified device drivers.

Limitations:

- Para virtualization requires the guest OS to be modified in order to interact with para virtualization interfaces.
- It requires significant support and maintaibilty issues in production environment.

RAMAIAH Institute of Technology

Hardware virtualization

Hardware virtualization is the virtualization of computers as complete hardware platforms, certain logical abstractions of their componentry, or only the functionality required to run various operating systems.

In computing**, hardware-assisted virtualization** is a platform virtualization approach that enables efficient **full virtualization** using help from hardware capabilities, primarily from the host processors.

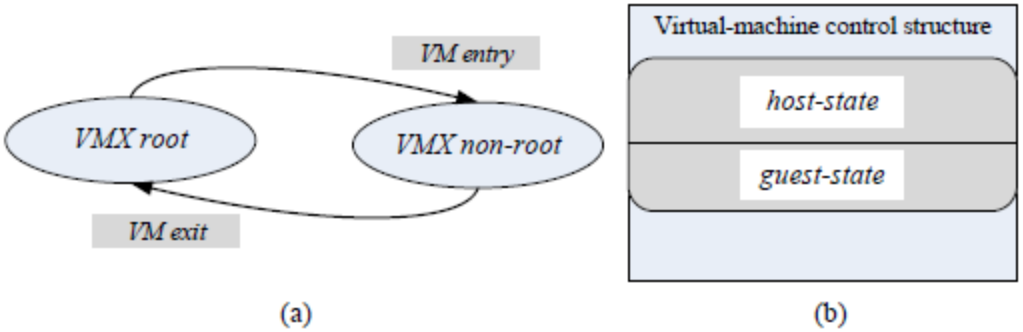
- A full virtualization is used to emulate a complete hardware environment, or virtual machine, in which **an unmodified guest operating system (using the same instruction set as the host machine) effectively executes in complete isolatio**n.

- Hardware-assisted virtualization was added to **x86 processors (Intel VT-x or AMD-V)** in 2005 and 2006 (respectively).

RAMAIAH Institute of Technology

x86 virtualization

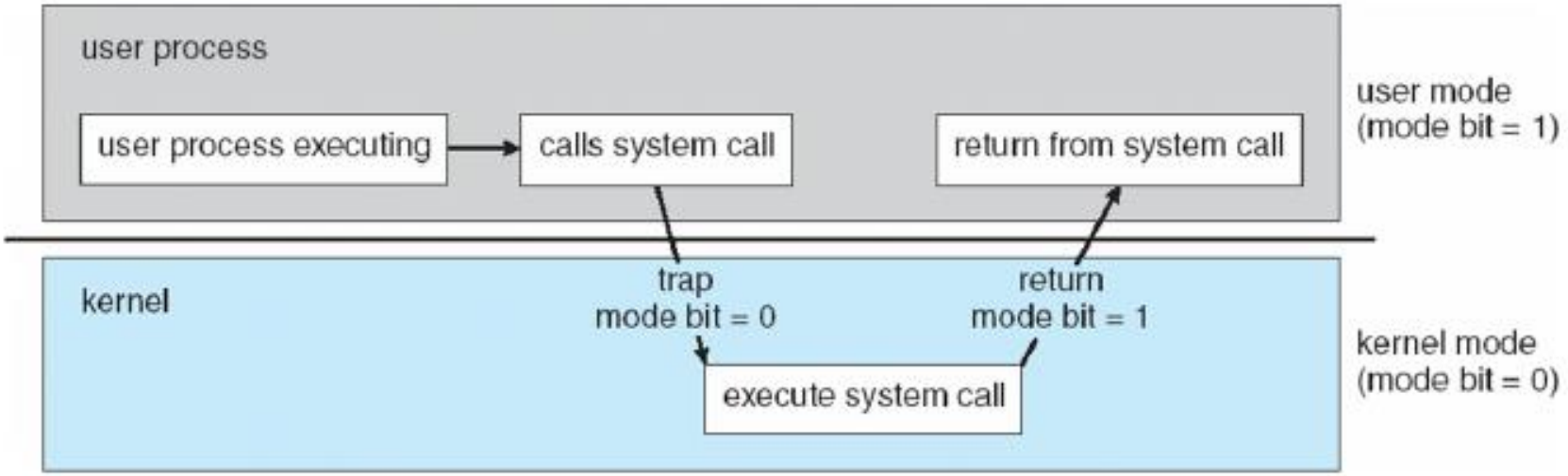
- x86 virtualization is the use of hardware-assisted virtualization capabilities on an x86/x86-64 CPU.
- **Intel virtualization (VT-x)**
- In 2005 Intel released two Pentium 4 models supporting VT-x. VT-x supports two modes of operations (Figure (a)):
    - **VMX root** - for VMM operations.
    - **VMX non-root** - support a VM.
- a new data structure called the **Virtual Machine Control Structure(VMCS)** including host-state and guest-state areas (Figure (b)).
    - **VM entry** - the processor state is loaded from the guest-state of the VM scheduled to run; then the control is transferred from VMM to the VM.
    - **VM exit** - saves the processor state in the guest-state area of the running VM; then it loads the processor state from the host-state area, finally transfers control to the VMM.



RAMAIAH
Institute of Technology

Dual-mode operation allows OS to protect itself and other system components

- **User mode and kernel mode**
- **Mode bit provided by hardware**
  - Ability to distinguish when system is running user or kernel code
  - Some instructions are privileged, only executable in kernel mode
  - System call changes mode to kernel, return resets it to user



Three classes of machine instructions:

- **Privileged instructions can be executed in kernel mode**. When attempted to be executed in user mode, they cause a trap and so executed in kernel mode.
- **Nonprivileged instructions the ones that can be executed in user mode**
- **Sensitive instructions can be executed in either kernel or user** but they behave differently.

RAMAIAH Institute of Technology

Challenges/Problems faced by x86 CPU Virtualization

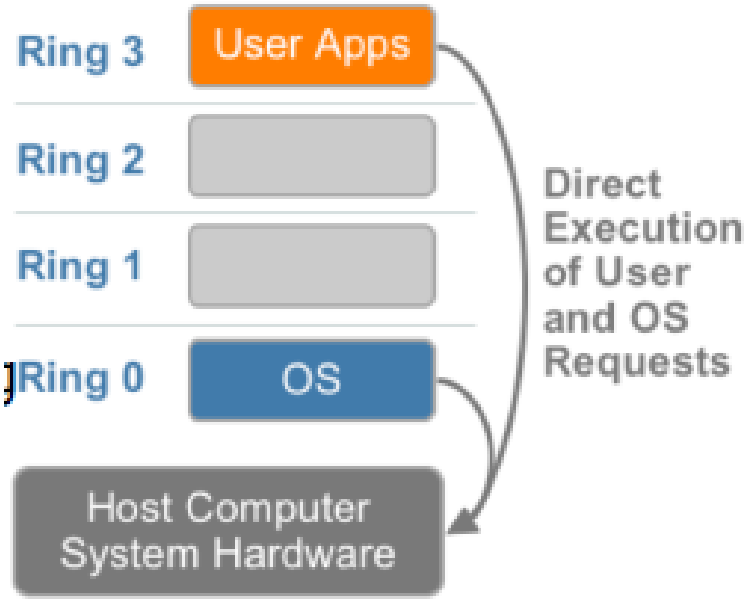
- **Ring de-privileging** –

- a VMMs forces the guest software, operating system and the applications to run at a privilege level greater than 0.
- The x86 architecture provides **Four layers of privilege execution→rings(level 0-3)**

Two solutions are then possible:

- The (0/1/3) mode, in which the VMM, the OS, and the application run at privilege levels 0, 1, and 3, respectively; or

- The (0,3,3) mode, in which the VMM, a guest OS, and applications run at privilege levels 0, 3, and 3, respectively.



RAMAIAH
Institute of Technology

Problem faced by virtualization of the x86 architecture:

- **Ring aliasing** - a guest OS is forced to run at a privilege level other than that it was originally designed for.
- **Address space compression** - a VMM uses parts of the guest address space to store several system data structures.
- **Non-faulting access to privileged state** - several store instructions can only be executed at privileged level 0 because they operate on data structures that control the CPU operation. They fail silently when executed at a privilege level other than 0.
- **Guest system calls** which cause transitions to/from privilege level 0 must be emulated by the VMM.
- **Interrupt virtualization** - in response to a physical interrupt, the VMM generates a ``virtual interrupt'' and delivers it later to the target guest OS which can mask interrupts.

RAMAIAH
Institute of Technology

Problem faced by virtualization of the x86 architecture:

- **Access to hidden state** - elements of the system state, e.g., descriptor caches for segment registers, are hidden; there is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.

- **Ring compression** - paging and segmentation protect VMM code from being overwritten by guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish between privilege levels 0, 1, and 2, thus the guest OS must run at privilege level 3, the so called (0/3/3) mode. Privilege levels 1 and 2 cannot be used thus, the name ring compression.

- **Frequent access to privileged resources increases VMM overhead:** The task-priority register is frequently used by a guest OS; the VMM must protect the access to this register and trap all attempts to access it. This can cause a significant performance degradation.

Overview of x86 Virtualization

- virtualization layer is added between the **hardware** and **operating system** as seen in Figure 2.
- This virtualization layer allows multiple operating system instances to run concurrently within virtual machines on a single computer,
- **dynamically partitioning and sharing the available physical resources** such as CPU, storage, memory and I/O devices.

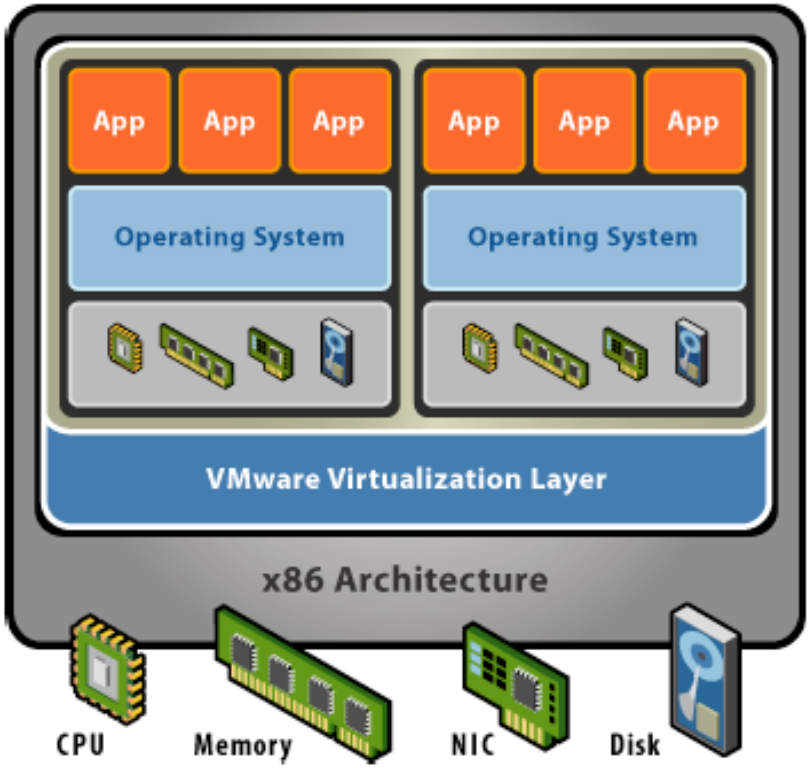


Figure 2 – x86 virtualization layer

- For industry standard x86 systems, virtualization approaches use either a hosted or a hypervisor architecture.

- A **hosted architecture** installs and runs the virtualization layer as **an application on top of an operating system** and supports the broadest range of hardware configurations.

- In contrast, a **hypervisor (bare-metal) architecture** installs the virtualization layer directly on a clean x86-based system.

- Since it has direct access to the hardware resources rather than going through an operating system, **a hypervisor is more efficient than a hosted architecture** and delivers greater scalability, robustness and performance.

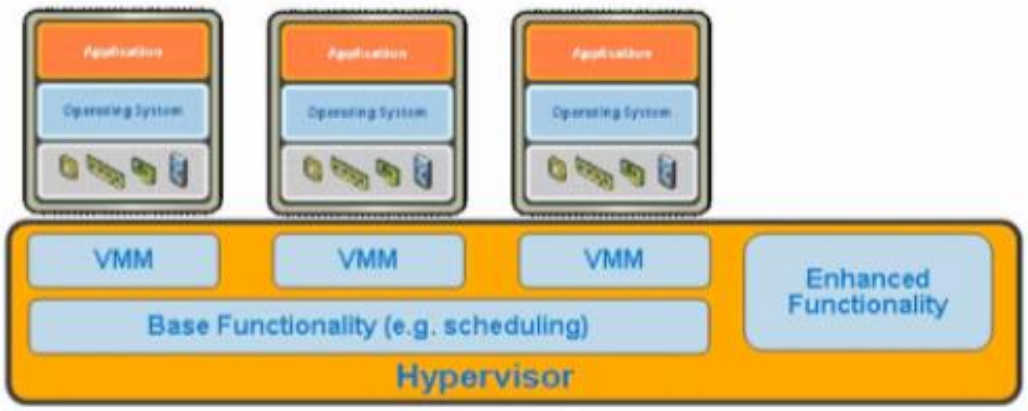


Figure 3 – The hypervisor manages virtual machine monitors that host virtual machines

Figure 3, the virtualization layer is a hypervisor running directly on the hardware.. Each VMM running on the hypervisor implements the **virtual machine hardware abstraction** and is responsible for running a guest OS. **Each VMM has to partition and share the** CPU, memory and I/O devices to successfully virtualize the system

The Challenges of x86 Hardware Virtualization

X86 operating systems are **designed to run directly on the bare-metal hardware**, so they naturally assume they fully 'own' the computer hardware. As shown in Figure 4,

the x86 architecture **offers four levels of privilege** known as Ring 0, 1, 2 and 3 to operating systems and applications to manage access to the computer hardware.

- While **user level applications** typically run in Ring 3,

- the **operating system needs to have direct access** to the memory and hardware and must execute its privileged instructions in Ring 0.

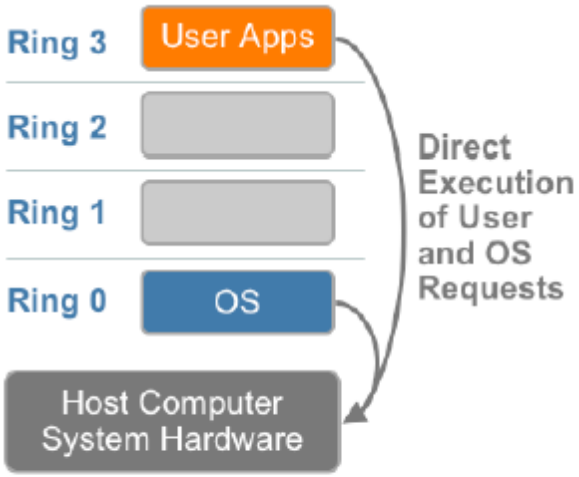


Figure 4 – x86 privilege level architecture without virtualization

Virtualizing the x86 architecture requires placing a virtualization layer under the operating system (which expects to be in the most privileged Ring 0) to create and manage the virtual machines that deliver shared resources. Further complicating the situation, **some sensitive instructions can't effectively be virtualized as they have different semantics when they are not executed in Ring 0.** The difficulty in trapping and translating these sensitive and privileged instruction requests at runtime was the challenge that originally made x86 architecture virtualization look impossible.

VMware resolved the challenge in 1998, developing **binary translation techniques** that allow the VMM to run in Ring 0 for isolation and performance,

while moving the operating system to a user level ring with greater privilege than applications in Ring 3 but less privilege than the virtual machine monitor in Ring 0.

Three alternative techniques now exist for handling sensitive and privileged instructions to virtualize the CPU on the x86 architecture:

- **Full virtualization using binary translation**
- **OS assisted virtualization or paravirtualization**
- **Hardware assisted virtualization (first generation)**

Technique 1 – Full Virtualization using Binary Translation

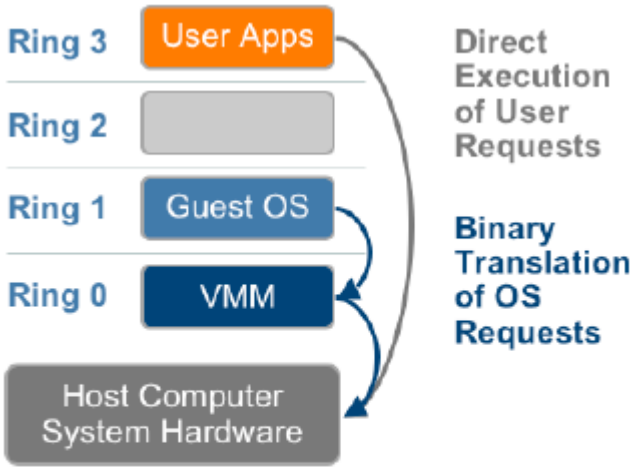


Figure 5 – The binary translation approach to x86 virtualization

The guest OS is not aware it is being virtualized and requires no modification.

Full virtualization is the only option that requires no hardware assist or operating system assist to virtualize sensitive and privileged instructions.

The hypervisor translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed.

Full virtualization offers the best isolation and security for virtual machines, and simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware.

Technique 2 – OS Assisted Virtualization or Paravirtualization

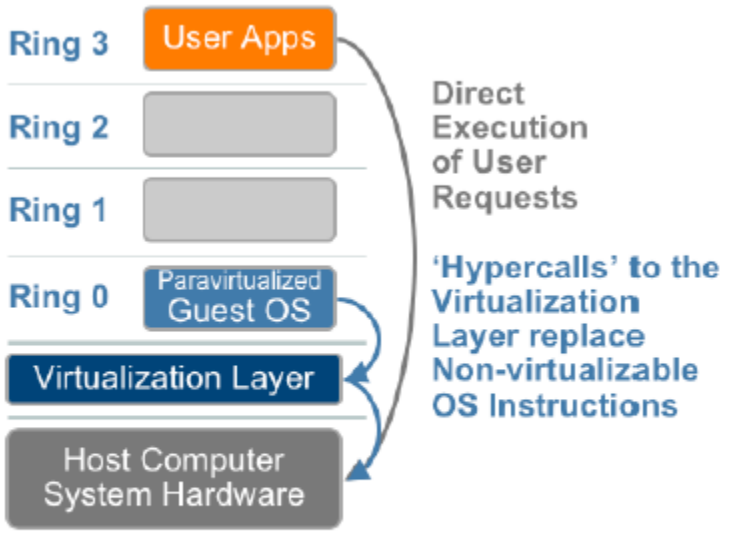


Figure 6 – The Paravirtualization approach to x86 Virtualization

Paravirtualization is different from full virtualization, where the unmodified OS does not know it is virtualized and sensitive OS calls are trapped using binary translation.

As paravirtualization cannot support unmodified operating systems (e.g. Windows 2000/XP), its compatibility and portability is poor.

Paravirtualization can also introduce significant support and maintainability issues in production environments as it requires deep OS kernel modifications.

The open source Xen project is an example of paravirtualization that virtualizes the processor and memory using a modified Linux kernel and virtualizes the I/O using custom guest OS device drivers

Technique 3 – Hardware Assisted Virtualization

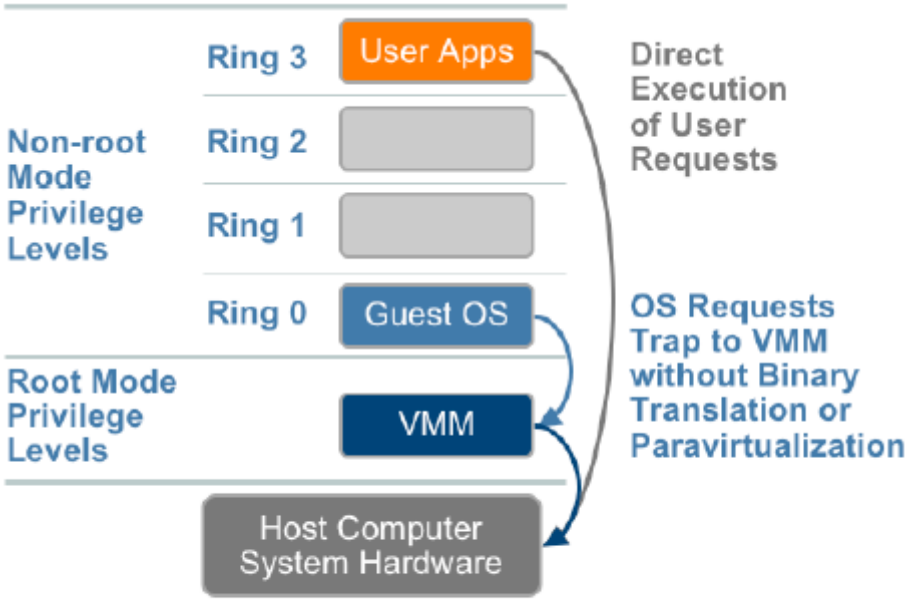


Figure 7 – The hardware assist approach to x86 virtualization

Hardware vendors are rapidly embracing virtualization and developing new features to simplify virtualization techniques.

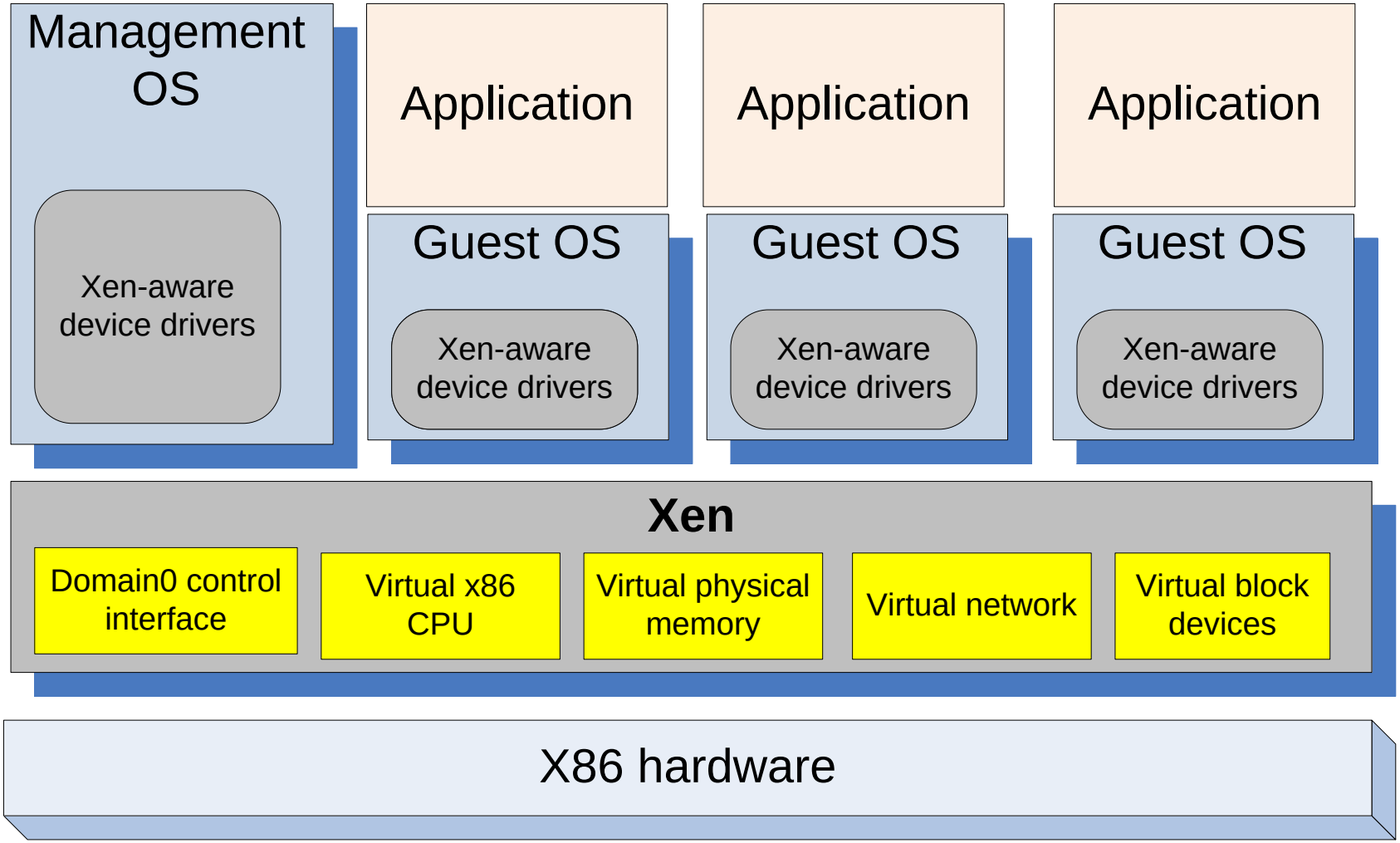
First generation enhancements include Intel Virtualization Technology (VT-x) and AMD's AMD-V which both target privileged instructions with a new CPU execution mode feature that allows the VMM to run in a new root mode below ring 0.

As depicted in Figure 7, privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for either binary translation or paravirtualization.

The guest state is stored in Virtual Machine Control Structures (VT-x) or Virtual Machine Control Blocks (AMD-V).

- Processors with Intel VT and AMD-V became available in 2006, so only newer systems contain these hardware assist features.

	Full Virtualization with Binary Translation	Hardware Assisted Virtualization	OS Assisted Virtualization / Paravirtualization
Technique	Binary Translation and Direct Execution	Exit to Root Mode on Privileged Instructions	Hypercalls
Guest Modification / Compatibility	Unmodified Guest OS Excellent compatibility	Unmodified Guest OS Excellent compatibility	Guest OS codified to issue Hypercalls so it can't run on Native Hardware or other Hypervisors Poor compatibility; Not available on Windows OSes
Performance	Good	Fair Current performance lags Binary Translation virtualization on various workloads but will improve over time	Better in certain cases
Used By	VMware, Microsoft, Parallels	VMware, Microsoft, Parallels, Xen	VMware, Xen
Guest OS Hypervisor Independent?	Yes	Yes	XenLinux runs only on Xen Hypervisor VMI-Linux is Hypervisor agnostic


Management OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Application
Guest OS
Xen-aware device drivers
Xen
Domain0 control interface
Virtual x86 CPU
Virtual physical memory
Virtual network
Virtual block devices
X86 hardware

Performance comparison of virtual machines

We have seen that a VMM such as Xen introduces additional overhead and negatively affects performance…..Will Compare the performance of **Xen and OpenVZ**

- The questions examined are:
  - How the performance scales up with the load?
  - What is the impact of a mix of applications?
  - What are the implications of the load assignment on individual servers?

- **The main conclusions:**
  - The virtualization **overhead of Xen is considerably higher than that of OpenVZ** and that this is due primarily to L2-cache misses.
  - The **performance degradation when the workload increases is also noticeable for Xen**.
  - Hosting multiple tiers of the same application on the same server is not an optimal solution.

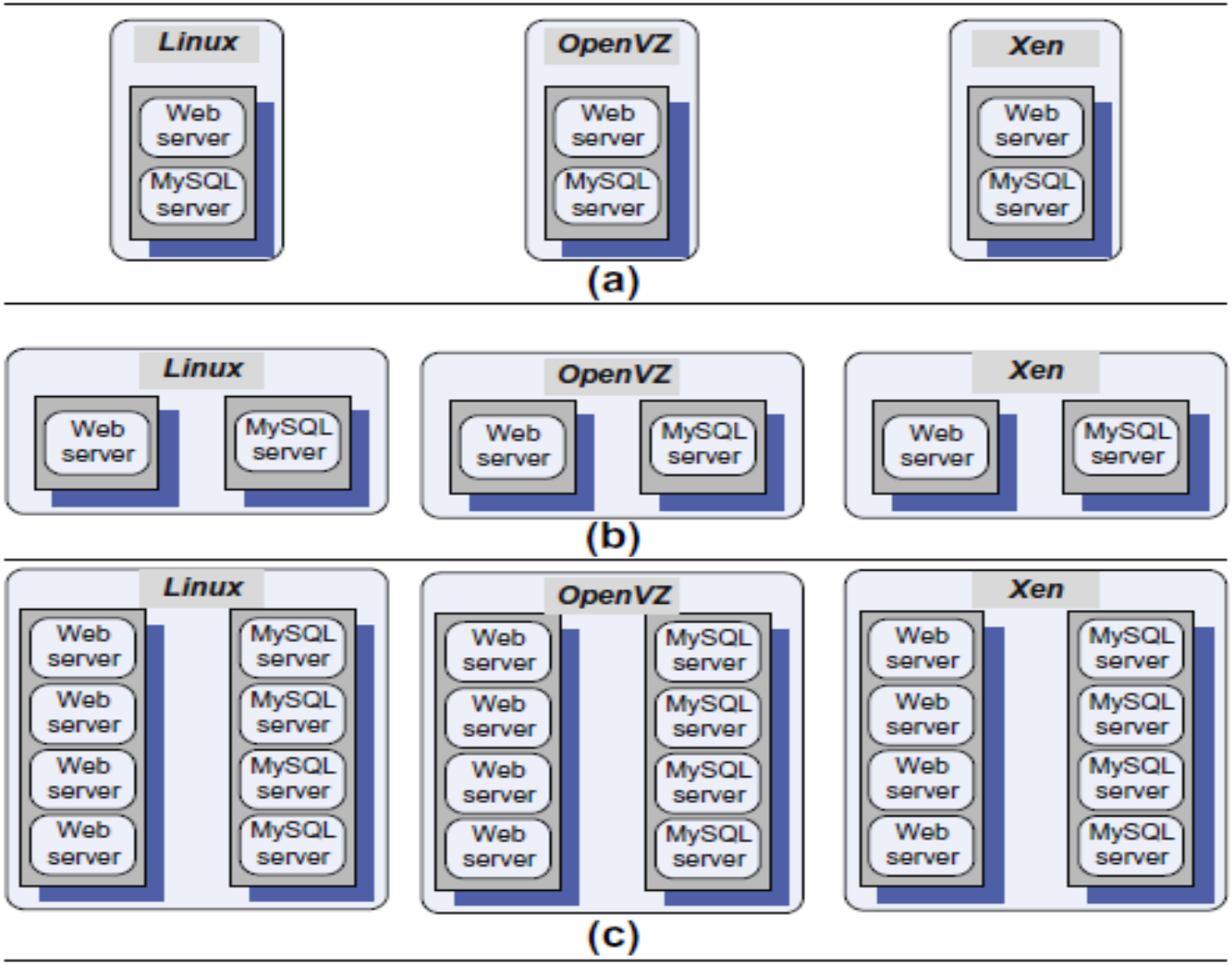


FIGURE 5.9

The setup for the performance comparison of a native `Linux` system with the *OpenVZ* and *Xen* systems. The applications are a Web server and a MySQL database server. (a) In the first experiment, the Web and the DB share a single system. (b) In the second experiment, the Web and the DB run on two different systems. (c) In the third experiment, the Web and the DB run on two different systems and each has four instances.

The experimental setups for three different experiments are shown in Figure 5.9 .

- **In the first group of experiments** **the two tiers of the application, the Web and the DB**, **run on a single server** for the Linux, the OpenVZ, and the Xen systems.

    - When the workload increases from 500 to 800 threads, The throughput increases linearly with the workload.

    - The response time increases only slightly for the base system and for the OpenVZ system, whereas it increases 600% for the Xen system.

    - For 800 threads the response time of the Xen system is four times longer than the time for OpenVZ.

    - The CPU consumption grows linearly with the load in all three systems; the DB consumption represents only 1–4% of it.

For a given workload, the Web-tier CPU consumption for the OpenVZ system is close to that of the base system and is about half of that for the Xen system.

- The performance analysis tool shows that the OpenVZ execution has two times more L2-cache misses than the base system,
- whereas the Xen Dom0 has 2.5 times more and the Xen application domain has 9 times more.

- **The second group of experiments** uses two servers, one for the Web and the other for the DB application, for each one of the three systems.

  - When the load increases from 500 to 800 threads the throughput increases linearly with the workload.
  - The response time of the Xen system increases only 114%, compared with 600% reported for the first experiments. The CPU time of the base system,
  - The OpenVZ system, the Xen Dom0, and the User Domain are similar for the Web application.

- **The third group of experiments uses** two servers, one for the Web and the other for the DB application,

  - for each one of the three systems but runs four instances of the Web and the DB application on the two servers.
  - The throughput increases linearly with the workload for the range used in the previous two experiments, from 500 to 800 threads.
  - The response time remains relatively constant for OpenVZ and increases 5 times for Xen

The main conclusion drawn from these experiments is that

- the virtualization overhead of Xen is considerably higher than that of OpenVZ and that this is due primarily to L2-cache misses.

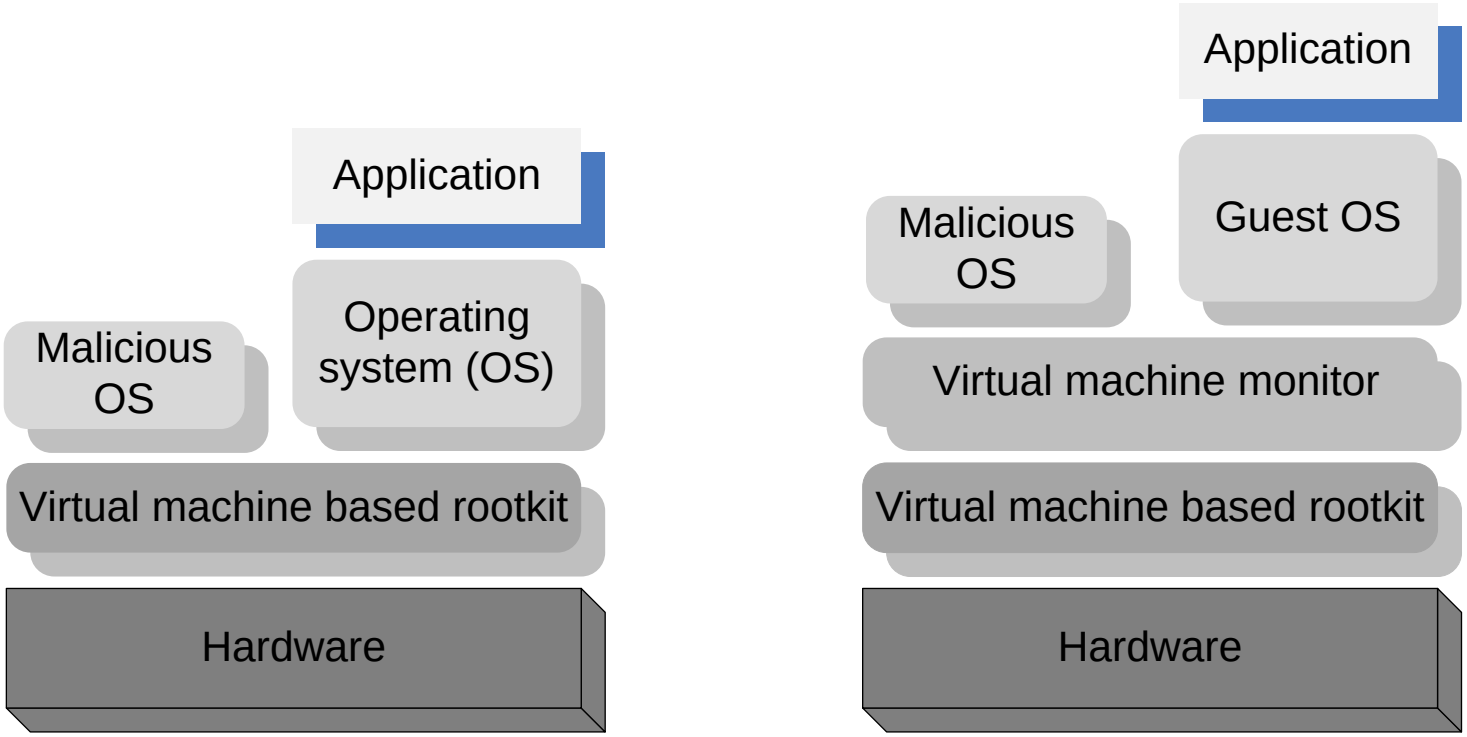
- The performance degradation when the workload increases is also noticeable for Xen.

- Another important conclusion is that hosting multiple tiers of the same application on the same server is not an optimal solution.

The darker side of virtualization

In a layered structure, a defense mechanism **at some layer can be disabled by malware running at a layer below it.** a VMM allows a guest operating system to run on virtual hardware. The VMM offers to the guest operating systems a hardware abstraction and mediates its access to the physical hardware

- It is feasible to insert a *rogue VMM*, **a Virtual-Machine Based Rootkit (VMBR) between the physical hardware and an operating system.**
- Rootkit - malware with a privileged access to a system.
- The VMBR can **enable a separate malicious OS to run surreptitiously and make this malicious OS invisible to the guest OS** and to the application running under it.
- Under the protection of the VMBR, the malicious OS could:
    - **observe the data, the events, or the state of the target system.**
    - **run services, such as spam relays or distributed denial-of-service attacks.**
    - **interfere with the application.**



The insertion of a Virtual-Machine Based Rootkit (VMBR) as the lowest layer of the software stack running on the physical hardware;

(a) **below an operating system; (b) below a legitimate virtual machine monitor.** The VMBR enables a malicious OS to run surreptitiously and makes it invisible to the genuine or the guest OS and to the application.

How such an insertion is possible

The malware runs either inside a VMM or with the support of a VMM;

But a VMM is a very potent engine for the malware. It prevents the software of the guest operating system or the application from detecting malicious activities.

A VMBR can record key strokes, system state, data buffers sent to or received from the network, and data to be written to or read from the disk with impunity; moreover, it can change any data at will.

The only way for a VMBR to take control of a system is to modify the boot sequence and to first load the malware and only then load the legitimate VMM or the operating system. This is only possible if the attacker has root privileges.

Software fault isolation

- Software fault isolation (SFI) offers a technical solution **for sandboxing binary code** of questionable provenance that can affect security in cloud computing.

- Insecure and tampered VM images are one of the security threats because binary codes of questionable provenance for native plug-ins to a Web browser can pose a security threat when Web browsers are used to access cloud services

- The application of the **sandboxing technology for two modern CPU architectures**, ARM and 64-bit x86. ARM is a load/store architecture with 32-bit instruction and 16 general-purpose registers

Table 5.4 The features of the SFI for the native client on the *x86*-32, *x86*-64, and *ARM*. ILP stands for instruction-level parallelism.

Feature/Architecture	*x86*-32	*x86*-64	*ARM*
Addressable memory	1 GB	4 GB	1 GB
Virtual base address	Any	44 GB	0
Data model	ILP 32	ILP 32	ILP 32
Reserved registers	0 of 8	1 of 16	0 of 16
Data address mask	None	Implicit in result width	Explicit instruction
Control address mask	Explicit instruction	Explicit instruction	Explicit instruction
Bundle size (bytes)	32	32	16
Data in text segment	Forbidden	Forbidden	Allowed
Safe address registers	All	RSP, RBP	SP
Out-of-sandbox store	Trap	Wraps mod 4 GB	No effect
Out-of-sandbox jump	Trap	Wraps mod 4 GB	Wraps mod 1 GB

Cloud Resource Management and Scheduling

Cloud Resource Management and Scheduling:

- Policies and mechanisms for resource management,
- Resource bundling, combinatorial auctions for cloud
- Scheduling algorithms for computing clouds,
- Fair queuing, start time fair queuing,
- Borrowed virtual time
- Resource management and application scaling

- Resource management is the practice of planning, scheduling, and allocating people, money, and technology to a project or program.

- Resource management is a core function required for any cloud system or man-made system. and **inefficient resource management has a direct negative effect on performance and cost**, while it can also indirectly affect system functionality, becoming too expensive or ineffective due to poor performance.

- It affects the **three basic criteria for the evaluation of a system**:
  - **Functionality. (indirect)**
  - **Performance.(direct negative effect)**
  - **Cost. (direct negative effect)**

- Cloud resource management requires **complex policies and decisions** for multi-objective optimization.

- The Strategies for resource management is associated with the three cloud delivery models. IaaS, PaaS,SaaS differ from one another.

- **Policies and mechanisms for resource allocation.**

  - **Policy → principles guiding decisions.**
  - **Mechanisms → the means to implement policies.**

Cloud resource management polices can be loosely grouped into five classes:

Admission control → The goal is to prevent the system from accepting workload in violation of high-level system policies. Example:

- workload requires **some a system may not accept an additional workload** that would prevent it from completing work already in progress or contracted.
- **Limiting the knowledge of the global state** of the system.

Capacity allocation → It means allocate resources for individual instances **(instance is an activations of a service).**

Load balancing → distribute the workload evenly among the servers.

- Consider 4 identical server A,B,C,D whose relative loads are 80%,60%.40% and 20% respectively of their capacity.
- As a result of perfect load balancing all servers would end with the same load – 50% of each server capacity. In cloud computing critical **goal is minimizing the cost of providing the service and minimizing the energy consumption.**
- Hence load from D is shifted to A and C is shifted to B. Thus **A and B will be loaded at full capacity**, whereas **C and D will be switched to standby mode**.

Energy optimization → minimization of energy consumption.

Quality of service (QoS) guarantees → ability to satisfy timing or other conditions specified by a **Service Level Agreement.**

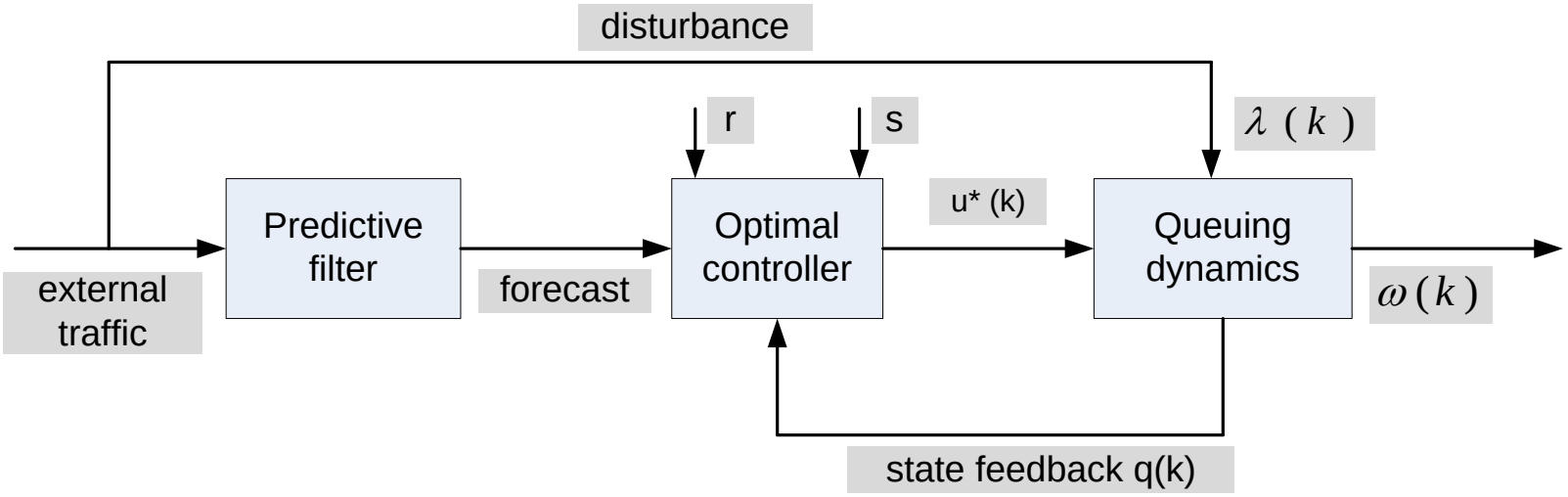
The four Mechanisms for the implementation of resource management policies

- **Control theory → uses the feedback** to guarantee system stability and predict transient behavior. It can be used only to predict local rather than global behavior.

- **Machine learning →** A major advantage is that they do not need a performance model of the system. This technique could be applied to coordination of several autonomous system managers.

- **Utility-based →** It require a performance model and a mechanism to correlate user-level performance with cost.

- **Market-oriented/economic →** do not require a model of the system, e.g., combinatorial auctions (**auction: It is a process of buying and selling goods or services**)for bundles of resources.

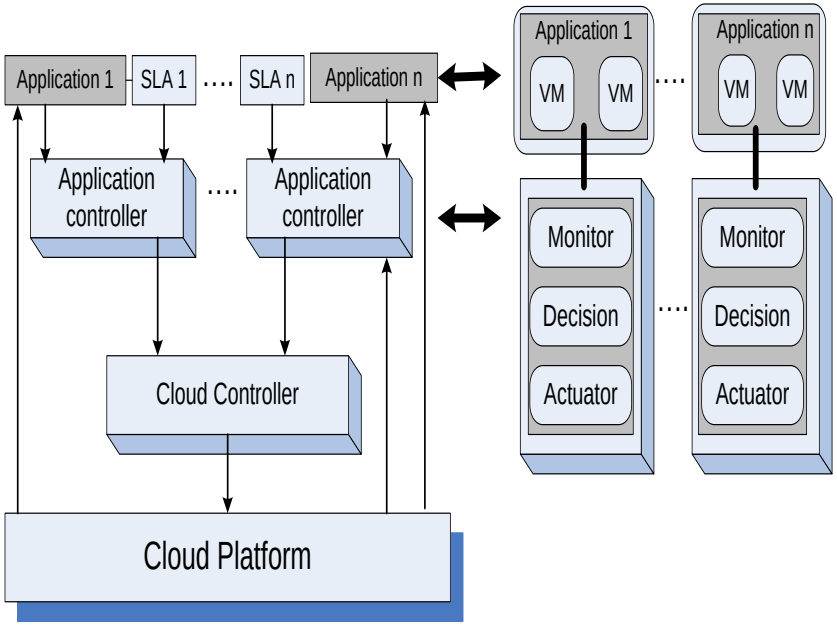


Control Theory: The controller uses the **feedback regarding the current state** and the **estimation of the future disturbance** due to environment to compute the optimal inputs over a finite horizon**.**

***r* and *s* are the weighting factors of the performance index.**

Stability of a Two-level cloud controller Architecture:

- The actions of the **control system should be carried out in a rhythm(strong)** that does not lead to instability.
- **Adjustments should only be carried out after the performance of the system has stabilized.**
- The actions consist of **allocation/deallocation of one or more virtual machines**. Sometimes allocation/dealocation of a single VM required by one of the threshold may cause crossing of the other, another source of instability.



Feed back control based on dynamic thresholds:

The elements involved in control systems are

-Sensors (**measures the parameters of interest**, then transmit the measured values to monitor.)

-Monitors(determines whether the **system behavior should be changed or not**)

-Actuators (if changes required, **actuators carry out the necessary actions)**

Feedback control based on : Dynamic Thresholding

Algorithm: **Proportional Thresholding**

- Compute the **integral value of the high and the low threshold** as averages of the maximum and, respectively, the minimum of the processor utilization over the process history.
    - **Request additional VMs** when the **average value of the CPU utilization** over the current time slice **exceeds the high threshold.**
    - **Release a VM** when the average value of the CPU utilization over the current time **slice falls below the low threshold.**

Conclusions

- **Dynamic thresholds perform better than the static ones.**
- **Two thresholds are better than one.**

Resource bundling: Combinatorial auctions for cloud resources

- **Resources in a cloud are allocated in bundles.(pakages)**
- Users get maximum benefit from a specific combination of resources: CPU cycles, main memory, disk space, network bandwidth, and so on.
- Resource bundling complicates traditional resource allocation models and has generated an interest in economic models and, in particular, in auction algorithms.
- **In the context of cloud computing, an auction is the allocation of resources to the highest bidder.**

- **Auctions in which participants can bid on combinations of items, or packages are called combinatorial auctions.**

- Such auctions provide a relatively simple , scalable and tractable solution to cloud resource allocation.

- Combinatorial auction algorithms:

    - Simultaneous clock auction ,
    - clock proxy auction,
    - **Ascending clock auction (ASCA)**

In all these algorithms , **the current price for each resource is represented by a "clock"** seen by all participants at the auction.

Combinatorial auctions for cloud resources

- **Prices and allocation are set as a result of an auction.**
- **Users provide bids for desirable bundles and the price they are willing to pay.**

Assume a population of **users U, u={ 1,2,…,U} and resources R, r={1,2,…,R}.**

The bid of user u is $B_{u} = \{ Q_u , \prod_u \}$,$Q_{i=} \{ q_u^1, q_u^2, q_u^{3\ldots} \}$
an R-Component Vector.

-q_u^1- **bundle of resource user u would accept**

-$\prod_u$- **total price to be paid for the resource**

q_u^i - is a positive quantity – resource desired
- is a negative quantity – resource offered

User desire is Indifferent set I = (q_u^1 XORq_u^2XORq_u^3)

The bidding process aims to optimize an objective function *f(x,p)*.

Constraints for Combinatorial auction algorithm

1. The user either gets one of the bundles it has opted for or nothing. No partial allocation is acceptable.
2. The system awards only available resources ; only offered can be allocated.
3. The bid of winners exceeds the final price
4. The winners get the least expensive bundles in their indifference set.
5. Losers bid below the final price.
6. All prices are positive numbers.

Table 6.4 The constraints for a combinatorial auction algorithm.

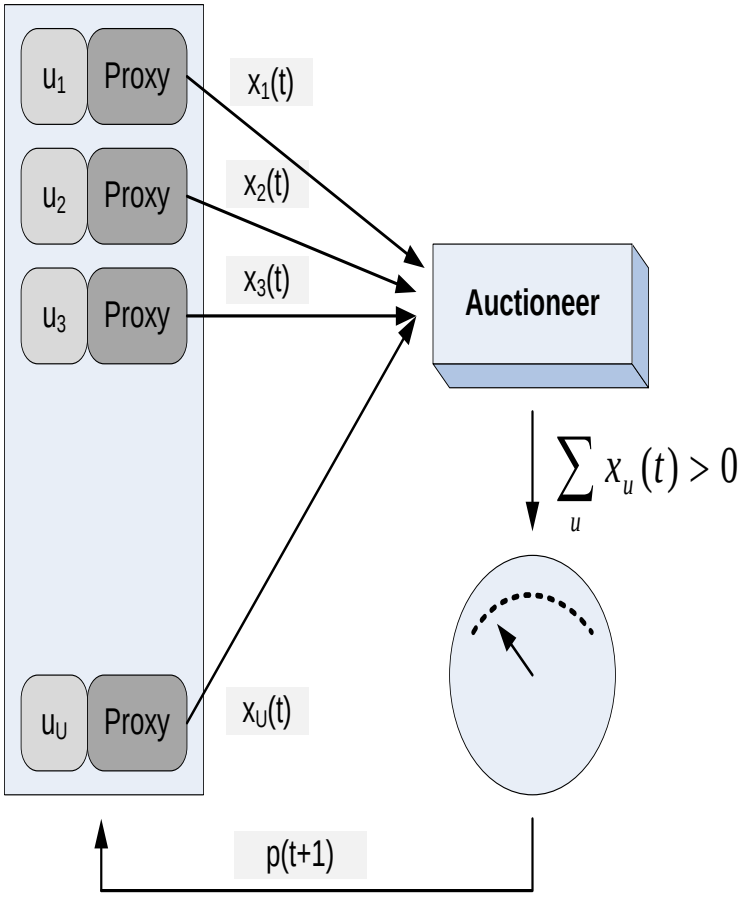
$x_u \in \{0 \cup \mathcal{Q}_u\}, \forall u$	A user gets all resources or nothing.
$\sum_u x_u \leqslant 0$	Final allocation leads to a net surplus of resources.
$\pi_u \geqslant (x_u)^T p, \forall u \in \mathcal{W}$	Auction winners are willing to pay the final price.
$(x_u)^T p = \min_{q \in \mathcal{Q}_u} (q^T p), \forall u \in \mathcal{W}$	Winners get the cheapest bundle in $\mathcal{I}$.
$\pi_u < \min_{q \in \mathcal{Q}_u} (q^T p), \forall u \in \mathcal{L}$	The bids of the losers are below the final price.
$p \geqslant 0$	Prices must be nonnegative.

Pricing and allocation algorithms

A pricing and allocation algorithm partitions the set of users in **two disjoint sets, winners and losers.**

Desirable properties of a pricing algorithm:

- **Be computationally tractable**; traditional combinatorial auction algorithms e.g., Vickey-Clarke-Groves (VLG) are not computationally tractable.
- **Scale well -** given the scale of the system and the number of requests for service, scalability is a necessary condition.
- **Be objective -** partitioning in winners and losers should only be based on the price of a user's bid; if the price exceeds the threshold then the user is a winner, otherwise the user is a loser.
- **Be fair -** make sure that the prices are uniform, all winners within a given resource pool pay the same price.
- Indicate clearly at the end of the auction the **unit prices for each resource pool.**
- Indicate clearly to all participants the **relationship between the supply and the demand in the system.**



The input to the ASCA algorithm: U users, R resources, $\bar{p}$ the starting price, and the update increment function, $g : (x, p) \mapsto \mathbb{R}^{\mathbb{R}}$. The pseudocode of the algorithm is:

1: set $t = 0$, $p(0) = \bar{p}$
2: **loop**
3: collect bids $x_u(t) = \mathcal{G}_u(p(t))$, $\forall u$
4: calculate excess demand $z(t) = \sum_u x_u(t)$
5: **if** $z(t) < 0$ **then**
6: **break**
7: **else**
8: update prices $p(t+1) = p(t) + g(x(t), p(t))$
9: $t \leftarrow t + 1$
10: **end if**
11: **end loop**

Ascending Clock Auction, (ASCA) → the current price for each resource is represented by a "clock" seen by all participants at the auction. The algorithm involves user bidding in multiple rounds; to address this problem the user proxies automatically adjust their demands on behalf of the actual bidders.

Scheduling algorithms for computing clouds

Scheduling → It is a critical component of cloud resource management. It is responsible for resource sharing/multiplexing at several levels:

- A server can be shared among **several virtual machines**.
- A virtual machine could support **several applications**.
- An application may consist of **multiple threads**.

A scheduling algorithm should be efficient, fair, and starvation-free. The objectives of a scheduler :

- **Batch system →**
  - **maximize throughput(the number of jobs completed in one unit of time) and**
  - **minimize turnaround time(the time between job submission and its completion).**
- **Real-time system → meet the deadlines and be predictable.**

Two different dimensions of resource management must be addressed by a scheduling policy:

- **The amount or quality of the resources allocated**
- **The timing when access to resource is granted**

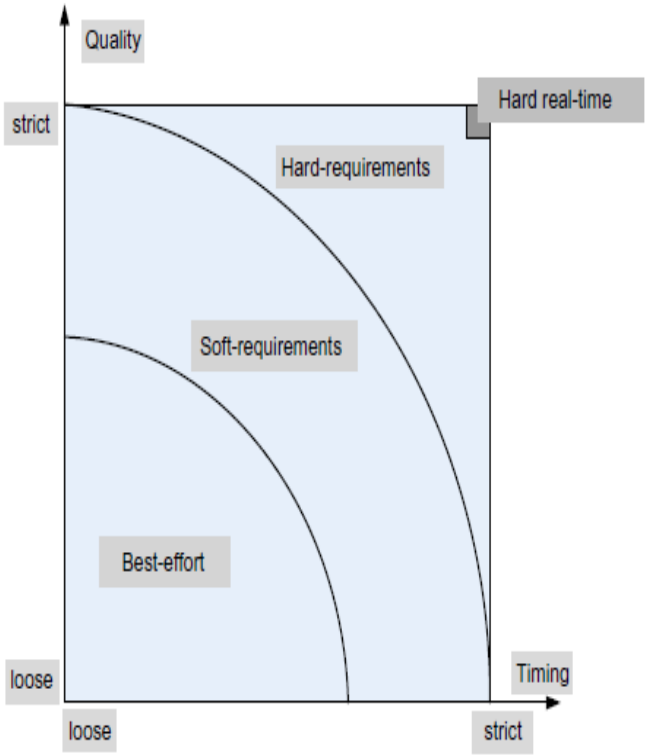


FIGURE 6.7

Best-effort policies do not impose requirements regarding either the amount of resources allocated to an application or the timing when an application is scheduled. Soft-requirements allocation policies require statistically guaranteed amounts and timing constraints; hard-requirements allocation policies demand strict timing and precise amounts of resources.

Best-effort policies → do not impose(force) requirements regarding either the amount of resources allocated to an application, or the timing when an application is scheduled.

- **Soft-requirements policies** → require statistically guaranteed amounts and timing constraints
- **Hard-requirements policies** → are the most challenging because they demand strict timing and precise amounts of resources.

Figure 6.7 identifies **several broad classes of resource allocation** requirements in the space defined by these two dimensions: best-effort, soft requirements, and hard requirements.

Fair Scheduling Algorithms:

Max-Min fairness criterion:

- Consider a resource with **Bandwidth B** shared among **n users** who have equal rights.
  - Each user requests an **amount b_i and receives B_i.**
- Then according to max-min criterion, following conditions for fair allocation:

  C_1: $B_i <= b_i$ (The amount received by any user is not larger than the amount requested.)

  C_2: B_{min} should not be higher (if the minimum allocation of any user is B_{min} **no allocation satisfying C1 has a higher B_{min} than the current allocation.**)

  C_3: $B-B_{min}$ (**When we remove the user receiving the minimum allocation Bmin and then reduce the total amount of resources available from B** to($B-B_{min}$) ,the condition C_2 remains recursively true.)

Fairness criterion for CPU scheduling

- A fairness criterion for CPU scheduling requires that the amount of work in the **time interval t1 to t2** of two runnable **threads a and b ,**
- $\Omega_a(t1,t2)$ and $\Omega_b(t1,t2)$, minimize the expression

$$\left| \frac{\Omega_a(t1,t2)}{\omega_a} - \frac{\Omega_b(t1,t2)}{\omega_b} \right|$$

- Where ω_a and ω_b are the **weights of the thread a and b** respectively

The **quality-of-service (QoS) requirements** differ for different classes of cloud applications and demand different scheduling policies.

- Best-effort applications such as batch applications and analytics do not require QoS guarantees.
- **Round-robin, FCFS, shortest-job-first (SJF),** and priority algorithms are among the most common scheduling algorithms for best-effort applications

Fair queuing

- **Fair scheduling algorithm** can be used for **scheduling packet transmission** , as well as threads.

- **Interconnection networks** allow cloud servers to communicate with one another and with users.

    - These **n/w consists** of communication links of limited bandwidth and switches/routers/gateways of limited capacity.

    - **When the load exceeds its capacity , a switch starts dropping packets**, because it has limited buffers for the switching fabric and for the outgoing links , as well as CPU cycles.

- A switch must handle flows and pairs of source destination endpoints of the traffic. Thus **a scheduling algorithm has to manage several quantities at the same time.**

    - **bandwidth:** amount of data each flow is allowed to transport.
    - **timing:** when the packets of individual flows are transmitted
    - **Buffer space:** space allocated to each flow.

The first strategy to avoid network congestion is to use FCFS

Solution_1: first strategy : FCFS.

- To avoid network congestion – **FCFS.** But does not guarantees fairness;

Solution_2: Fair Queuing(FQ) proposes that **separate queues, one per flow** , be maintained by a switch and the queues be serviced in a **round robin (RR) manner.**

- manages buffer space management, not bandwidth allocation. Indeed a flow transporting large packets will benefit from a larger bandwidth.

Solution_3: bit by bit RR (BR)

- impractical scheme , a single bit from each queue is transmitted and the queues are visited in a round robin fashion.

    - **R(t) – number of rounds aof the BR algorithm up to time t.**
    - **$N_{active(t)}$ – number of active flows through the switch.**

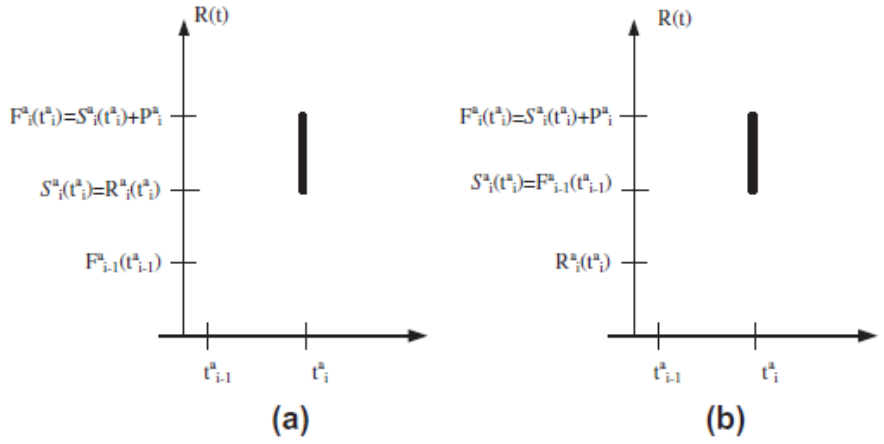


FIGURE 6.8

Transmission of a packet i of flow a arriving at time t_i^a of size P_i^a bits. The transmission starts at time $S_i^a = \max[F_{i-1}^a, R(t_i^a)]$ and ends at time $F_i^a = S_i^a + P_i^a$ with $R(t)$ the number of rounds of the algorithm. (a) The case $F_{i-1}^a < R(t_i^a)$. (b) The case $F_{i-1}^a \geqslant R(t_i^a)$.

The **transmission of packet *i*** of a flow can only start after the packet is available and the transmission of the previous packet has finished.

- **(a)The new packet arrives after the previous has finished.**
- **(b)The new packet arrives before the previous one was finished.**

the switch. Call t_i^a the time when the packet i of flow a, of size P_i^a bits arrives, and call S_i^a and F_i^a the values of $R(t)$ when the first and the last bit, respectively, of the packet i of flow a are transmitted. Then,

$$F_i^a = S_i^a + P_i^a \quad \text{and} \quad S_i^a = \max\left[F_{i-1}^a, R(t_i^a)\right]. \tag{6.28}$$

The quantities $R(t)$, $N_{active}(t)$, S_i^a, and F_i^a depend only on the arrival time of the packets, t_i^a, and not on their transmission time, provided that a flow a is active as long as

$$R(t) \leqslant F_i^a \quad \text{when} \quad i = \max\left(j | t_i^a \leqslant t\right). \tag{6.29}$$

$$B_i^a = P_i^a + \max\left[F_{i-1}^a, \left(R\left(t_i^a\right) - \delta\right)\right], \tag{6.30}$$

Start-time fair queuing

The basic idea of the **start-time fair queuing (SFQ) algorithm** is to organize the consumers of the CPU bandwidth in a tree structure; the root node is the **processor** and the leaves of this tree are the **threads of each application**.

- A scheduler acts at each level of the hierarchy. The fraction of the processor bandwidth, B, allocated to the intermediate node i is

$$\frac{B_i}{B} = \frac{w_i}{\sum_{j=1}^{n} w_j} \qquad (6.31)$$

with w_j, $1 \leqslant j \leqslant n$, the weight of the n children of node i; see the example in Figure 6.9.

- When a **virtual machine is not active**, its bandwidth is reallocated to the other VMs active at the time.
- When **one of the applications of a virtual machine is not active**, its allocation is transferred to the other applications running on the same VM.
- Similarly, if **one of the threads of an application is not runnable**, its allocation is transferred to the other threads of the applications.

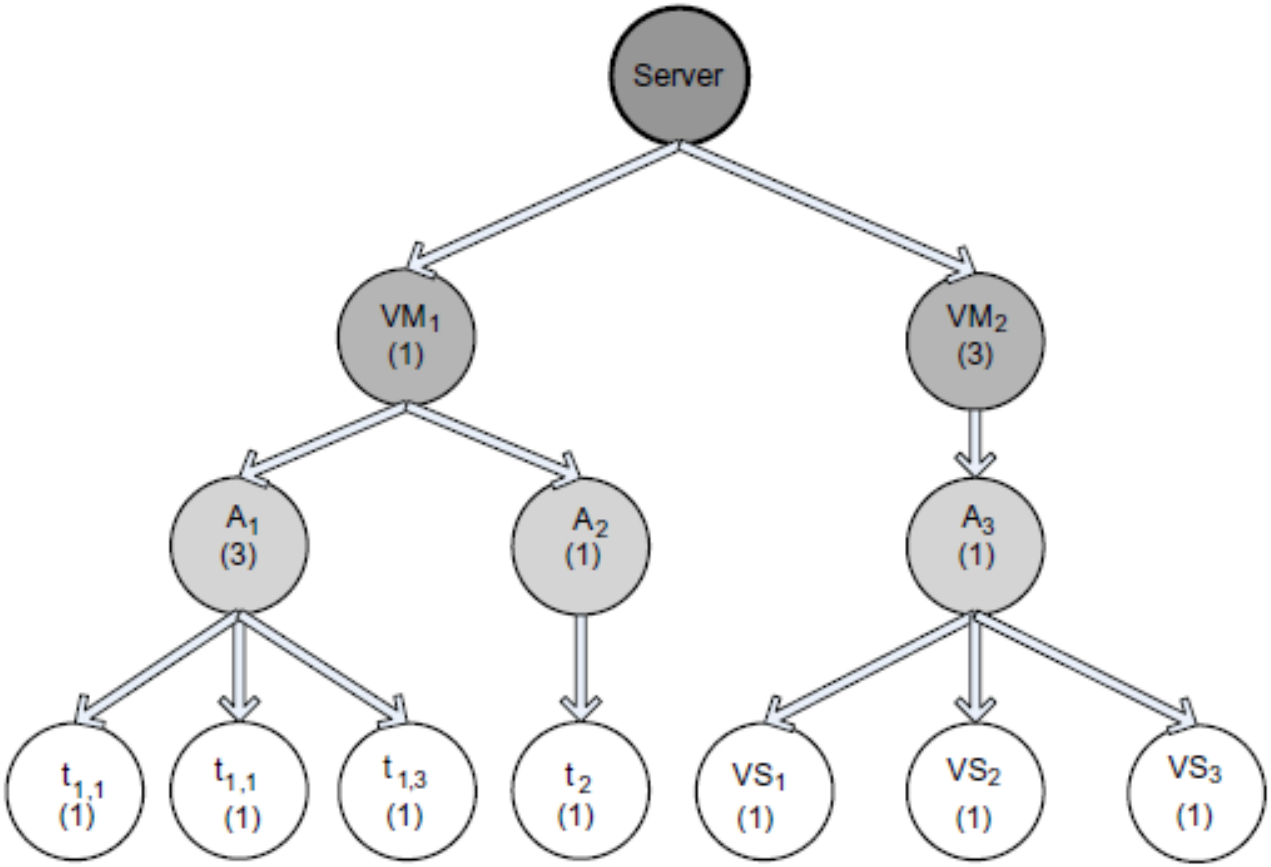


FIGURE 6.9

The SFQ tree for scheduling when two virtual machines, VM_1 and VM_2, run on a powerful server. VM_1 runs two best-effort applications A_1, with three threads $t_{1,1}$, $t_{1,2}$, and $t_{1,3}$, and A_2 with a single thread, t_2. VM_2 runs a video-streaming application, A_3, with three threads vs_1, vs_2, and vs_3. The weights of virtual machines, applications, and individual threads are shown in parenthesis.

- ***va(t)* and *vb(t)* the virtual time** of threads *a* and *b*, respectively, at real time *t*.
  - The virtual time of the scheduler at time *t* is denoted by *v(t)*.
  - Call ***q* the time quantum** of the scheduler in milliseconds.
  - The ***threads a and b*** have their time quanta, ***qa*** and ***qb,***
  - weighted by ***wa*** and ***wb***, respectively
- Thus, in the example, **the time quanta of the two threads** are **q/wa** and **q/wb,** respectively
- The ***i* -th activation of thread *a*** will **start** at the **virtual time *Sia*** and will **finish** at **virtual time *Fia***
- We call τj the real time of the *j* -th invocation of the scheduler.

An SFQ scheduler follows several rules:

R1. The threads are serviced in the order of their virtual start-up time; ties are broken arbitrarily.

R2. The **virtual startup time** of the *i* -th activation of thread *x* is

$$S_x^i(t) = \max\left[v\left(\tau^j\right), F_x^{(i-1)}(t)\right] \quad \text{and} \quad S_x^0 = 0. \tag{6.32}$$

R3. The **virtual finish time** of the i -th activation of thread x is

$$F_x^i(t) = S_x^i(t) + \frac{q}{w_x}. \tag{6.33}$$

R4. The virtual time of all threads is initially zero, v0x = 0. The **virtual time v(t)** at real time t is computed as follows:

$$v(t) = \begin{cases} \text{Virtual start time of the thread in service at time } t, & \text{if CPU is busy} \\ \text{Maximum finish virtual time of any thread,} & \text{if CPU is idle.} \end{cases} \tag{6.34}$$

In this description of the algorithm we have included the real time t to stress the dependence of all events in virtual time on the real time. To simplify the notation we use in our examples the real time as the index of the event. In other words, S_a^6 means the virtual start-up time of thread a at real time $t = 6$.

Example. The following example illustrates the application of the SFQ algorithm when there are two threads with the weights $w_a = 1$ and $w_b = 4$ and the time quantum is $q = 12$ (see Figure 6.10.)

Initially $S_a^0 = 0$, $S_b^0 = 0$, $v_a(0) = 0$, and $v_b(0) = 0$. Thread b blocks at time $t = 24$ and wakes up at time $t = 60$.

The scheduling decisions are made as follows:

1. $t = 0$: We have a tie, $S_a^0 = S_b^0$, and arbitrarily thread b is chosen to run first. The virtual finish time of thread b is
$$F_b^0 = S_b^0 + q/w_b = 0 + 12/4 = 3. \tag{6.35}$$
2. $t = 3$: Both threads are runnable and thread b was in service; thus, $v(3) = S_b^0 = 0$; then
$$S_b^1 = \max[v(3), F_b^0] = \max(0, 3) = 3. \tag{6.36}$$
But $S_a^0 < S_b^1$, thus thread a is selected to run. Its virtual finish time is
$$F_a^0 = S_a^0 + q/w_a = 0 + 12/1 = 12. \tag{6.37}$$
3. $t = 15$: Both threads are runnable, and thread a was in service at this time; thus,
$$v(15) = S_a^0 = 0 \tag{6.38}$$
and
$$S_a^1 = \max[v(15), F_a^0] = \max[0, 12] = 12. \tag{6.39}$$
As $S_b^1 = 3 < 12$, thread b is selected to run; the virtual finish time of thread b is now
$$F_b^1 = S_b^1 + q/w_b = 3 + 12/4 = 6. \tag{6.40}$$

4. $t = 18$: Both threads are runnable, and thread b was in service at this time; thus,

$$v(18) = S_b^1 = 3 \tag{6.41}$$

and

$$S_b^2 = \max[v(18), F_b^1] = \max[3, 6] = 6. \tag{6.42}$$

As $S_b^2 < S_a^1 = 12$, thread b is selected to run again; its virtual finish time is

$$F_b^2 = S_b^2 + q/w_b = 6 + 12/4 = 9. \tag{6.43}$$

5. $t = 21$: Both threads are runnable, and thread b was in service at this time; thus,

$$v(21) = S_b^2 = 6 \tag{6.44}$$

and

$$S_b^3 = \max[v(21), F_b^2] = \max[6, 9] = 9. \tag{6.45}$$

As $S_b^2 < S_a^1 = 12$, thread b is selected to run again; its virtual finish time is

$$F_b^3 = S_b^3 + q/w_b = 9 + 12/4 = 12. \tag{6.46}$$

6. $t = 24$: Thread b was in service at this time; thus,

$$v(24) = S_b^3 = 9 \tag{6.47}$$

$$S_b^4 = \max[v(24), F_b^3] = \max[9, 12] = 12. \tag{6.48}$$

Thread b is suspended till $t = 60$; thus, thread a is activated. Its virtual finish time is

$$F_a^1 = S_a^1 + q/w_a = 12 + 12/1 = 24. \tag{6.49}$$

7. $t = 36$: Thread a was in service and the only runnable thread at this time; thus,

$$v(36) = S_a^1 = 12 \tag{6.50}$$

and

$$S_a^2 = \max[v(36), F_a^2] = \max[12, 24] = 24. \tag{6.51}$$

Then,

$$F_a^2 = S_a^2 + q/w_a = 24 + 12/1 = 36. \tag{6.52}$$

8. $t = 48$: Thread a was in service and is the only runnable thread at this time; thus,

$$v(48) = S_a^2 = 24 \tag{6.53}$$

and

$$S_a^3 = \max[v(48), F_a^2] = \max[24, 36] = 36. \tag{6.54}$$

Then,

$$F_a^3 = S_a^3 + q/w_a = 36 + 12/1 = 48. \tag{6.55}$$

9. $t = 60$: Thread a was in service at this time; thus,

$$v(60) = S_a^3 = 36 \tag{6.56}$$

and

$$S_a^4 = \max[v(60), F_a^3] = \max[36, 48] = 48. \tag{6.57}$$

But now thread b is runnable and $S_b^4 = 12$.
Thus, thread b is activated and

$$F_b^4 = S_b^4 + q/w_b = 12 + 12/4 = 15. \tag{6.58}$$

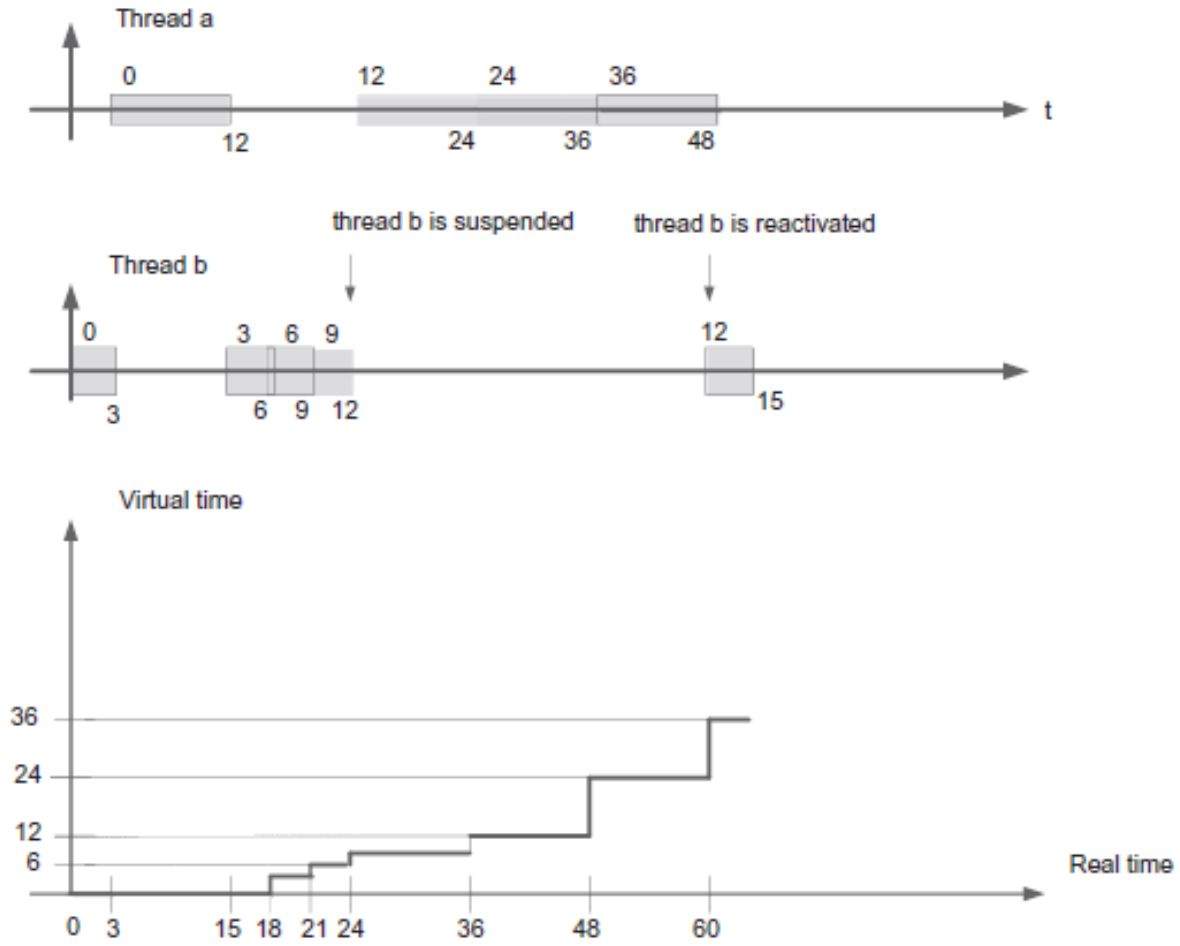


FIGURE 6.10

Top, the virtual start-up time $S_a(t)$ and $S_b(t)$ and the virtual finish time $F_a(t)$ and $F_b(t)$ function of the real time t for each activation of threads a and b, respectively, are marked at the top and bottom of the box representing a running thread. The virtual time of the scheduler $v(t)$ function of the real time is shown on the bottom graph.

Borrowed virtual time

Objective - support low-latency dispatching of real-time applications, and weighted sharing of CPU among several classes of applications.

- Like SFQ , BVT supports scheduling of a mix of applications- hard, soft real time constraints ,best effort.

A thread ***i*** has

- **an effective virtual time, E_i.**
- **an actual virtual time, A_i.**
- **a virtual time warp, W_i.**

- The scheduler thread maintains its own **scheduler virtual time (SVT)** defined as the minimum actual virtual time A_j of any thread.

- The threads are dispatched in the order of their effective virtual time E_i, a policy called the **Earliest Virtual Time (EVT).**

- The virtual warp time allows a thread to acquire an earlier effective time ie., to borrow virtual time from its future CPU allocation.

- The virtual warp time is enabled when the variable *warpBack* is set.

In this case a latency-sensitive thread gains dispatching preference as

$$E_i \leftarrow \begin{cases} A_i & \text{if} \quad warpBack = OFF \\ A_i - W_i & \text{if} \quad warpBack = ON. \end{cases} \tag{6.59}$$

- The algorithm measures time in ***minimum charging units(mcu)*** and uses time quantum called ***Context switch allowance(C)***
- which measures the real time a thread is allowed to run when competing with other threads, measured in multiples of *mcu*.
- example: mcu=100μsec C= 100 msec

Context switches are triggered by events such as:

- the running thread is blocked waiting for an event to occur.
- the time quantum expires and an interrupt occurs.
- when a thread becomes runnable after sleeping.
- When the thread becomes runnable after sleeping , its actual virtual time is updated as follows:

$$A_i \leftarrow \max[A_i, SVT]. \tag{6.60}$$

This policy prevents a thread sleeping for a long time to claim control of the CPU for a longer period of time than it deserves.

If there are no interrupts, threads are allowed to run for the same amount of virtual time. Individual threads have weights; a thread with a larger weight consumes its virtual time more slowly. In practice, each thread τ_i maintains a constant k_i and uses its weight w_i to compute the amount Δ used to advance its actual virtual time upon completion of a run:

$$A_i \leftarrow A_i + \Delta. \tag{6.61}$$

Given two threads a and b,

$$\Delta = \frac{k_a}{w_a} = \frac{k_b}{w_b}. \tag{6.62}$$

The EVT policy requires that every time the actual virtual time is updated, a context switch from the current running thread τ_i to a thread τ_j occurs if

$$A_j \leqslant A_i - \frac{C}{w_i}. \tag{6.63}$$

Example 1. The following example illustrates the application of the BVT algorithm for scheduling two threads a and b of best-effort applications. The first thread has a weight twice that of the second, $w_a = 2w_b$; when $k_a = 180$ and $k_b = 90$, then $\Delta = 90$.

We consider periods of real-time allocation of $C = 9$ *mcu*. The two threads a and b are allowed to run for $2C/3 = 6$ *mcu* and $C/3 = 3$ *mcu*, respectively.

Threads a and b are activated at times

$$\begin{aligned} a &: 0, 5, 5+9=14, 14+9=23, 23+9=32, 32+9=41, \ldots \\ b &: 2, 2+9=11, 11+9=20, 20+9=29, 29+9=38, \ldots \end{aligned} \tag{6.64}$$

The context switches occur at real times:

$$2, 5, 11, 14, 20, 23, 29, 32, 38, 41, \ldots \tag{6.65}$$

The time is expressed in units of *mcu*. The initial run is a shorter one, consists of only 3 *mcu*; a context switch occurs when a, which runs first, exceeds b by 2 *mcu*.

Table 6.5 shows the effective virtual time of the two threads at the time of each context switch. At that moment, its actual virtual time is incremented by an amount equal to Δ if the thread was allowed

Table 6.5 The real time of the context switch and the effective virtual time $E_a(t)$ and $E_b(t)$ at the time of a context switch. There is no time warp, so the effective virtual time is the same as the actual virtual time. At time $t = 0$, $E_a(0) = E_b(0) = 0$ and we choose thread a to run.

Context Switch	Real Time	Running Thread	Effective Virtual Time of the Running Thread
1	$t = 2$	a	$E_a(2) = A_a(2) = A_a(0) + \Delta/3 = 30$ b runs next as $E_b(2) = 0 < E_a(2) = 30$
2	$t = 5$	b	$E_b(5) = A_b(5) = A_b(0) + \Delta = 90$ a runs next as $E_a(5) = 30 < E_b(5) = 90$
3	$t = 11$	a	$E_a(11) = A_a(11) = A_a(2) + \Delta = 120$ b runs next as $E_b(11) = 90 < E_a(11) = 120$
4	$t = 14$	b	$E_b(14) = A_b(14) = A_b(5) + \Delta = 180$ a runs next as $E_a(14) = 120 < E_b(14) = 180$
5	$t = 20$	a	$E_a(20) = A_a(20) = A_a(11) + \Delta = 210$ b runs next as $E_b(20) = 180 < E_a(20) = 210$
6	$t = 23$	b	$E_b(23) = A_b(23) = A_b(14) + \Delta = 270$ a runs next as $E_a(23) = 210 < E_b(23) = 270$
7	$t = 29$	a	$E_a(29) = A_a(29) = A_a(20) + \Delta = 300$ b runs next as $E_b(29) = 270 < E_a(29) = 300$
8	$t = 32$	b	$E_b(32) = A_b(32) = A_b(23) + \Delta = 360$ a runs next as $E_a(32) = 300 < E_b(32) = 360$.
9	$t = 38$	a	$E_a(38) = A_a(38) = A_a(29) + \Delta = 390$ b runs next as $E_b(11) = 360 < E_a(11) = 390$
10	$t = 41$	b	$E_b(41) = A_b(41) = A_b(32) + \Delta = 450$ a runs next as $E_a(41) = 390 < E_b(41) = 450$

Resource management and application scaling

- The demand for computing resources, such as CPU cycles, primary and secondary storage, and network bandwidth, depends heavily on the volume of data processed by an application.

- The demand for resources can be a function of the time of day, can monotonically increase or decrease in time, or can experience predictable or unpredictable peaks.

For example,

  - a new Web service will experience a low request rate when the service is first introduced and the load will exponentially increase if the service is successful.

  - A service for income tax processing will experience a peak around the tax filling deadline, whereas access to a service provided by Federal Emergency Management Agency (FEMA)will increase dramatically after a natural disaster.

The question we address is: How scaling can actually be implemented in a cloud when a very large number of applications exhibit this often unpredictable behavior.

We distinguish two scaling modes: vertical and horizontal.

Vertical scaling keeps the number of VMs of an application constant, but increases the amount of resources allocated to each one of them.

- This can be done either by migrating the VMs to more powerful servers or by keeping the VMs on the same servers but increasing their share of the CPU time.

- The first alternative involves additional overhead; the VMis stopped, a snapshot of it is taken, the file is transported to a more powerful server, and, finally, the VM is restated at the new site.

Horizontal scaling is the most common mode of scaling on a cloud; it is done by increasing the number of VMs as the load increases and reducing the number of VMs when the load decreases.

- Often, this leads to an increase in communication bandwidth consumed by the application. Load balancing among the running VMs is critical to this mode of operation.

- For a very large application, multiple load balancers may need to cooperate with one another. In some instances the load balancing is done by a front-end server that distributes incoming requests of a transaction-oriented system to back-end servers.

In the case of an **arbitrarily divisible application** the workload can be partitioned dynamically; as the load increases, the system can allocate additional VMs to process the additional workload.

Most cloud applications belong to this class, which justifies our statement that **horizontal scaling is the most common scaling mode.**

There are several strategies to support scaling.

- Automatic VM scaling uses predefined metrics, e.g., CPU utilization, to make scaling decisions.

- Automatic scaling requires sensors to monitor the state of VMs and servers; controllers make decisions based on the information about the state of the cloud, often using a statemachine model for decision making. Amazon and Rightscale (www.rightscale.com) offer automatic scaling.

- In the case of AWS the CloudWatch service supports applications monitoring and allows a user to set up conditions for automatic migrations.

- Nonscalable or single-load balancers are also used for horizontal scaling.

- The Elastic Load Balancing service from Amazon automatically distributes incoming application traffic across multiple EC2 instances. Another service, the Elastic Beanstalk, allows dynamic scaling between a low and a high number of instances specified by the user (see Section 3.1). T