Module – 02

Virtual Machines and Virtualization of Clusters and Data Centers

Implementation Levels of Virtualization

Virtualization:

Virtualization allows multiple virtual machines (VMs) to run on the same physical hardware, improving resource sharing, performance, and flexibility.

It enhances system efficiency by separating hardware from software. It

has gained importance in distributed and cloud computing.

Levels of Virtualization Implementation

Virtualization can be implemented at various operational layers of the system, including:

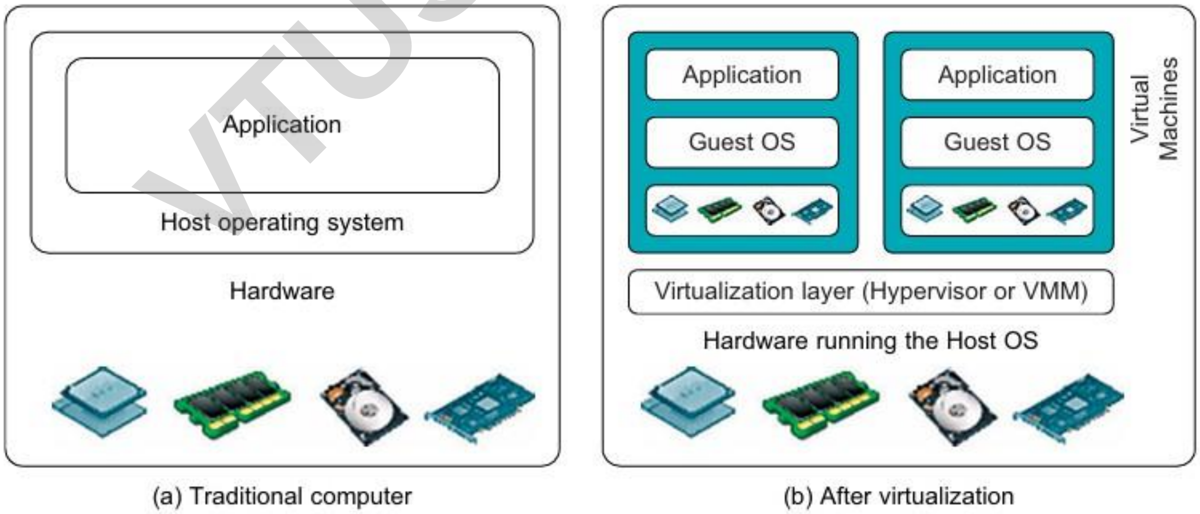


FIGURE 3.1

The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

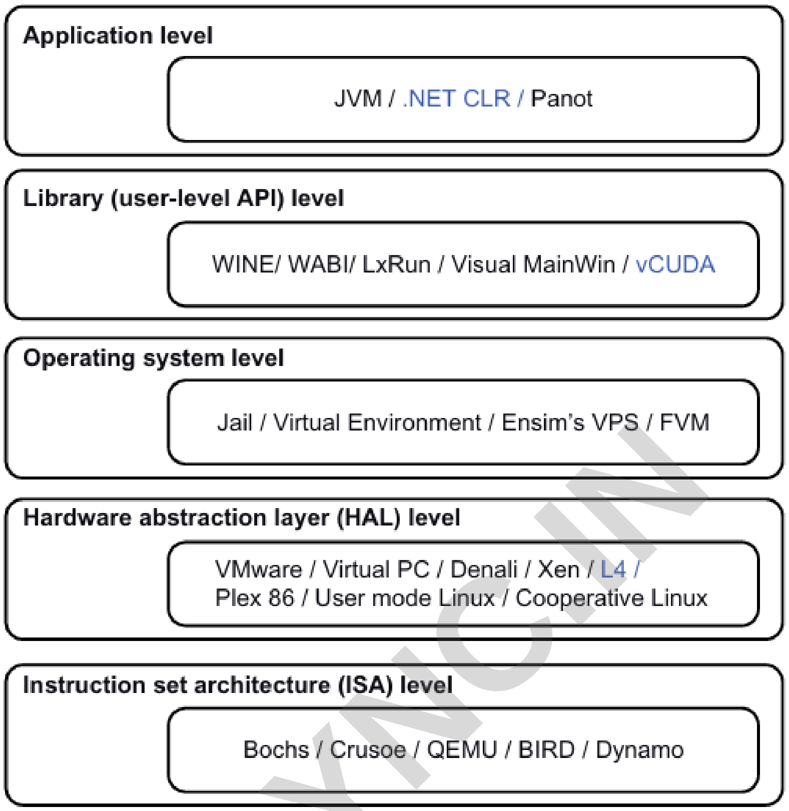

Application level
JVM / .NET CLR / Panot
Library (user-level API) level
WINE/ WABI/ LxRun / Visual MainWin / vCUDA
Operating system level
Jail / Virtual Environment / Ensim's VPS / FVM
Hardware abstraction layer (HAL) level
VMware / Virtual PC / Denali / Xen / L4 /
Plex 86 / User mode Linux / Cooperative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

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture (ISA) Level:

- Virtualizes the instruction set of the host machine to emulate different processor architectures (e.g., running MIPS code on an x86 machine).
- Uses code interpretation or dynamic binary translation for better performance.

Hardware Abstraction Level:

- Virtualizes hardware resources like CPU, memory, and I/O devices to allow multiple users to utilize the hardware concurrently.
- Historical example: IBM VM/370, modern example: Xen hypervisor for x86 machines.

Operating System (OS) Level:

- Creates isolated containers on a single server to allocate hardware resources among users.
- Commonly used in virtual hosting environments and server consolidation.

Library Support Level:

- Virtualizes the communication link between applications and the OS through API hooks.
- Examples include WINE (Windows applications on UNIX) and vCUDA (GPU acceleration within VMs).

User-Application Level:

- Virtualizes individual applications or processes, often called process-level virtualization.
- Examples include the Java Virtual Machine (JVM) and Microsoft .NET CLR.
- Other approaches include application isolation, sandboxing, and application streaming, where the application is isolated from the host OS for easier distribution and removal.

Purpose and Applications:

- Virtualization improves resource utilization, enables running different OS and applications on the same machine, and simplifies the management of distributed systems.
- It plays a key role in enhancing distributed computing, cloud environments, and legacy software support.

Relative Merits of Virtualization Approaches

Comparison Factors: Higher performance, application flexibility, implementation complexity, and application isolation.

Merit Representation: Number of X's in a table (5X = best, 1X = worst). **Performance**

Considerations:

- **Hardware & OS-level virtualization** → Highest performance but expensive.
- **User-level virtualization** → Most complex in terms of application isolation.
- **ISA-level virtualization** → Best application flexibility.

VMM Design Requirements and Providers

Definition & Role of VMM:

- VMM (Virtual Machine Monitor) is a layer between hardware and the operating system.
- Manages hardware resources and captures program interactions with hardware.
- Enables multiple OS instances to run on a single set of hardware.

Requirements of a VMM:

1. **Identical Execution Environment:**
   - Programs should run as if they are on a real machine.
2. **Minimal Performance Overhead:**
   - Should not significantly slow down program execution.
3. **Full Control Over System Resources:**
   - Programs should only access explicitly allocated resources.

Performance Considerations:

VMs share hardware, leading to resource contention.

Timing dependencies and resource availability may cause minor performance differences.

Traditional emulators/simulators offer flexibility but are too slow for real-world use.

Efficiency is ensured by executing most virtual processor instructions directly on hardware.

Resource Control by VMM:

Allocates hardware resources to programs.

Restricts unauthorized access to unallocated resources.

Can reclaim allocated resources under certain conditions.

Challenges in VMM Implementation:

Some processors (e.g., x86) lack full virtualization support.

Solution: Hardware-assisted virtualization modifies hardware to support VMM requirements.

Virtualization Support at the OS Level

Role of OS-Level Virtualization in Cloud Computing

Cloud computing relies on virtualization to shift hardware and management costs to third-party providers.

Two major challenges:

1. **Dynamic resource allocation** – Scaling CPU resources based on demand.
2. **Slow VM instantiation** – Fresh VM boots take time and lack awareness of the application state.

Why OS-Level Virtualization?

- Hardware-level virtualization is slow and inefficient due to redundant VM image storage and performance overhead.
- OS-level virtualization creates multiple **isolated Virtual Execution Environments (VEs) or Containers** within a single OS kernel.
- VEs function like real servers with their own processes, file system, user accounts, and network configurations but share the **same OS kernel**.
- Also known as **single-OS image virtualization**.

Advantages of OS-Level Virtualization

1. **Fast startup/shutdown, low resource use, high scalability.**
2. **State synchronization between VMs and the host OS** – Allows better application state awareness.
3. **Efficiency through resource sharing** – VEs can access most host resources without modifying them.
4. **Overcomes slow VM initialization and application state unawareness** in cloud computing.

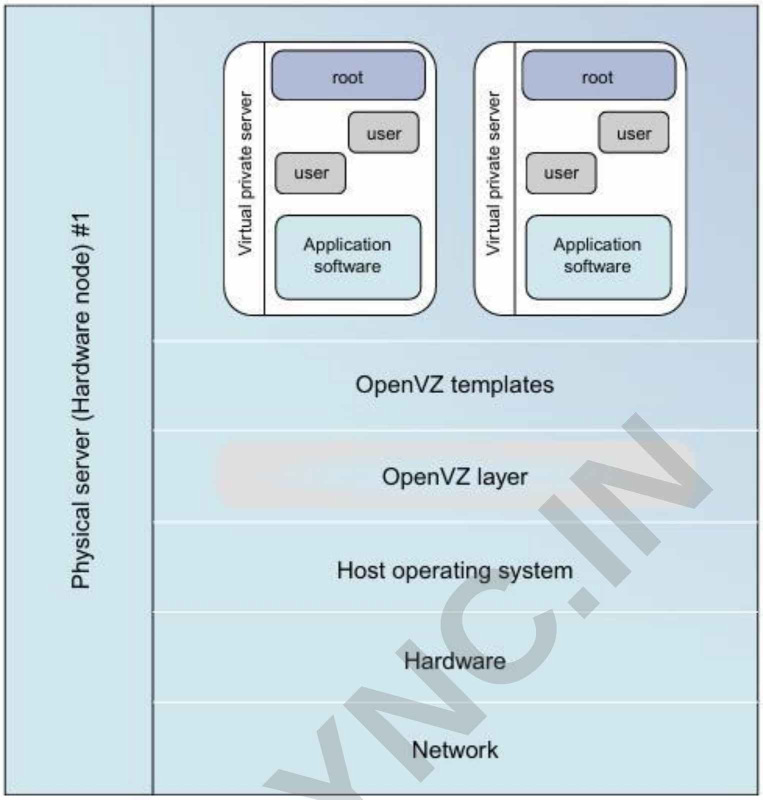


FIGURE 3.3

The OpenVZ virtualization layer inside the host OS, which provides some OS images to create VMs quickly.

(Courtesy of OpenVZ User's Guide [65])

Disadvantages of OS-Level Virtualization

1. **Same OS requirement** – All VMs on a single container must belong to the same OS family (e.g., Windows-based VMs cannot run on a Linux host).
2. **User preference issues** – Some cloud users require different OS types, limiting flexibility.
3. **Resource duplication problem** – If each VM has a full copy of system resources, it leads to high storage and performance costs.

Implementation Considerations

Creating virtual root directories:

1. **Duplicate resources for each VM** (higher cost).

2. **Share most resources and create private copies on demand** (preferred approach).

OS-level virtualization is often a **second choice** due to its limitations compared to hardware-assisted virtualization.

Virtualization on Linux and Windows Platforms

Linux vs. Windows Virtualization

- **Linux-based OS-level virtualization** is well-developed, while **Windows-based OS-level virtualization** is still in research.
- The **Linux kernel provides an abstraction layer** for handling hardware, often requiring patches for new hardware support.
- Most Linux platforms are **not tied to a specific kernel**, allowing multiple VMs to run on the same host.
- **Windows OS virtualization tools** are still experimental, with FVM being an example for the Windows NT platform.

Virtualization Support on the Linux Platform: OpenVZ

OpenVZ is an **open-source container-based virtualization** tool for Linux. It

modifies the Linux kernel to support:

1. **Virtual environments (VPS)** – Each VPS functions like an independent Linux server with its own processes, users, and virtual devices.
2. **Resource management** – Controls CPU, disk space, and memory allocation.
3. **Checkpointing and live migration** – Saves VM state to a file for quick transfer and restoration on another machine.

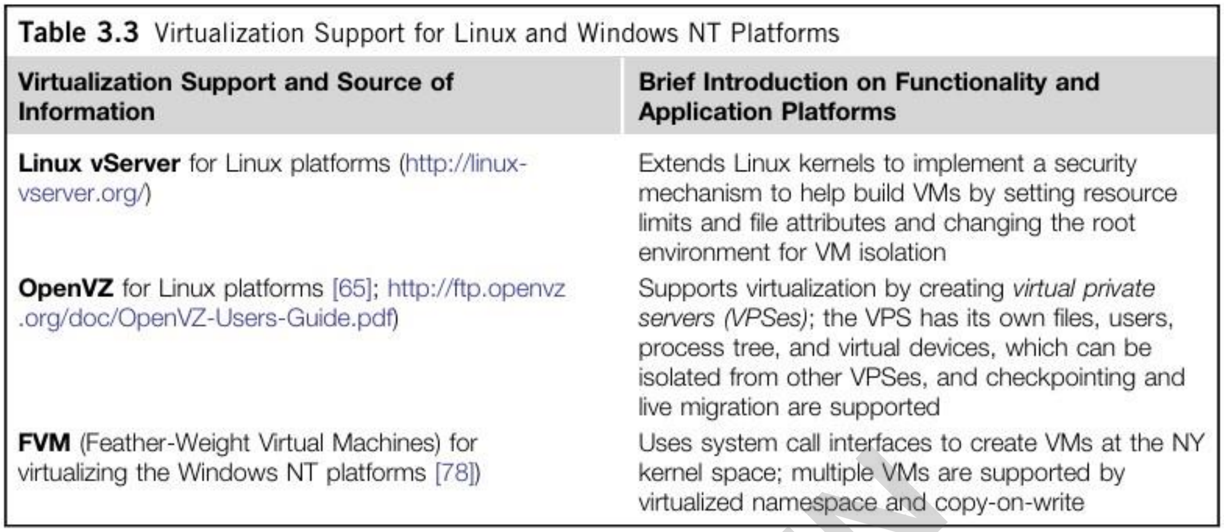


Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Resource Management in OpenVZ:

- **Two-level disk allocation:**
    - First level: Admin allocates disk space for VMs.
    - Second level: VM admin assigns disk space to users.
- **Two-level CPU scheduling:**
    - First level: OpenVZ decides VM priority.
    - Second level: Standard Linux CPU scheduler manages tasks within the VM.
- **20+ resource control parameters** ensure optimized VM usage.

Middleware Support for Virtualization (Library-Level Virtualization)

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation.

This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system.

API call interception and remapping are the key functions performed. This section provides an overview of several library-level virtualization systems.

Namely the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and vCUDA, which are summarized in Table 3.4.

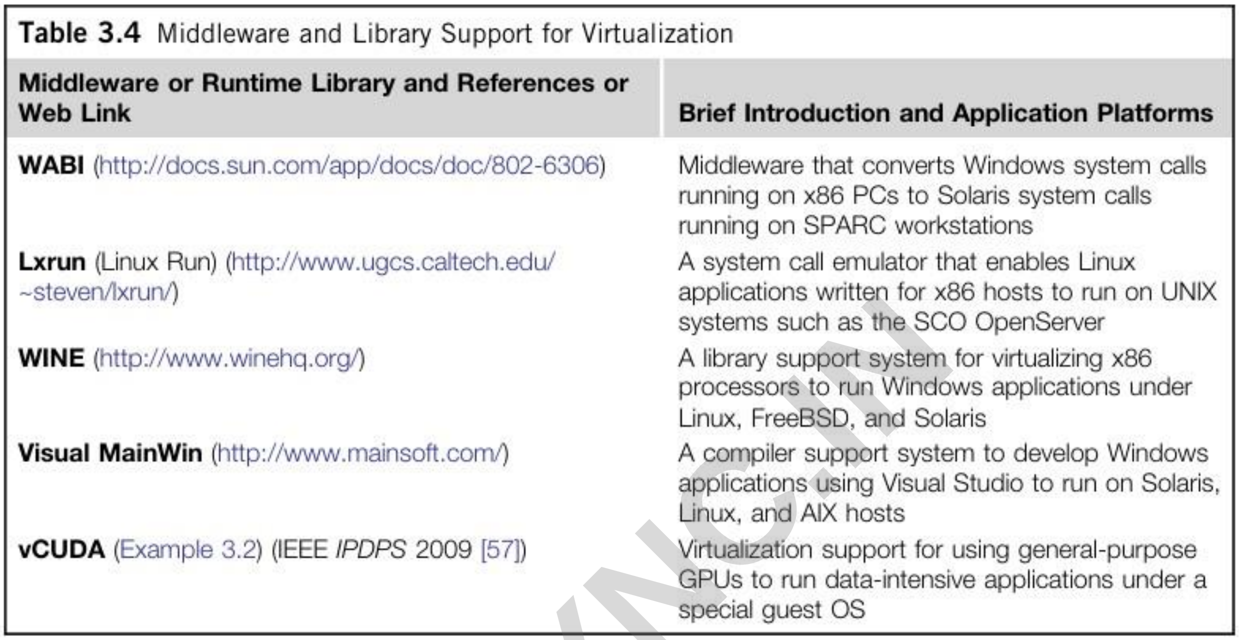


Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

Example 3.2 The vCUDA for Virtualization of General-Purpose GPUs

CUDA is a programming model and library for general-purpose GPUs. It leverages the high performance of GPUs to run compute-intensive applications on host operating systems. However, it is difficult to run CUDA applications on hardware-level VMs directly. vCUDA virtualizes the CUDA library and can be installed on guest OSes. When CUDA applications run on a guest OS and issue a call to the CUDA API, vCUDA intercepts the call and redirects it to the CUDA API running on the host OS. Figure 3.4 shows the basic concept of the vCUDA architecture [57].

The vCUDA employs a client-server model to implement CUDA virtualization. It consists of three user space components: the vCUDA library, a virtual GPU in the guest OS (which acts as a client), and the vCUDA stub in the host OS (which acts as a server). The vCUDA library resides in the guest OS as a substitute for the standard CUDA library. It is responsible for intercepting and redirecting API calls from the client to the stub. Besides these tasks, vCUDA also creates vGPUs and manages them.

The functionality of a vGPU is threefold: It abstracts the GPU structure and gives applications a uniform view of the underlying hardware; when a CUDA application in the guest OS allocates a device's memory the vGPU can return a local virtual address to the application and notify the remote stub to allocate the real device memory, and the vGPU is responsible for storing the CUDA API flow. The vCUDA stub receives

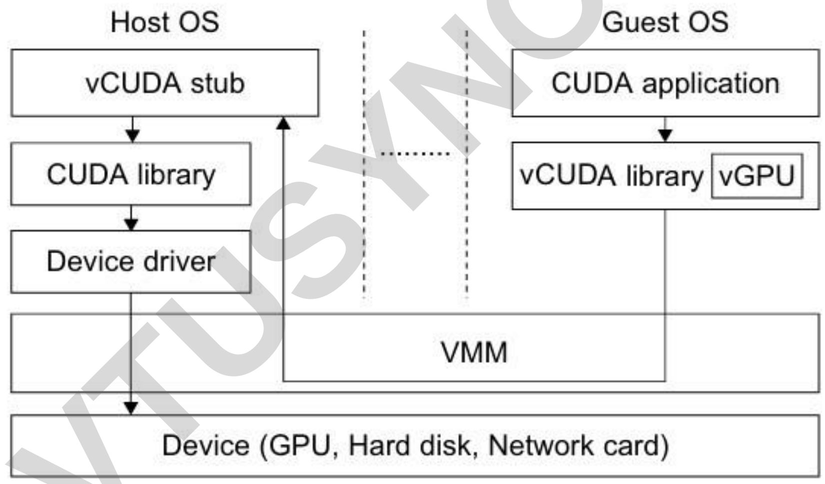


FIGURE 3.4

Basic concept of the vCUDA architecture.

(*Courtesy of Lin Shi, et al. © IEEE [57]*)

and interprets remote requests and creates a corresponding execution context for the API calls from the guest OS, then returns the results to the guest OS. The vCUDA stub also manages actual physical resource allocation.

Virtualization Structure/Tools and Mechanisms

VM Architecture Classes

After virtualization, a **virtualization layer** is inserted between the hardware and OS, converting real hardware into virtual hardware.

This allows multiple OSes (Linux, Windows, etc.) to run simultaneously on a single machine.

There are **three main classes of VM architecture:**

1. **Hypervisor-based virtualization** (VMM – Virtual Machine Monitor)
2. **Paravirtualization**
3. **Host-based virtualization**

Hypervisor and Xen Architecture

The **hypervisor** enables **hardware-level virtualization** by running **directly on bare metal hardware** (CPU, memory, disk, network interfaces).

It acts as an interface between physical hardware and guest OSes.

Types of Hypervisors:

Micro-kernel hypervisor (e.g., **Microsoft Hyper-V**):

- Includes only core functions (memory management, processor scheduling).
- Device drivers and other components are external.
- Smaller hypervisor size.

Monolithic hypervisor (e.g., **VMware ESX**):

- Implements all functions, including device drivers.
- Larger hypervisor size but with better performance and control.

The Xen Hypervisor Architecture

- **Xen** is an **open-source micro-kernel hypervisor** developed at **Cambridge University**.
- Separates **policy (handled by Domain 0) from mechanism (handled by Xen)**.
- **No native device drivers** → Guest OSes handle device management directly.
- **Virtual environment between hardware and OS.**

Components of Xen

Component	Description
Hypervisor	Core virtualization layer between hardware and OS.
Kernel	Supports guest OS execution.
Applications	Runs on guest OS instances.

Xen Domain Structure

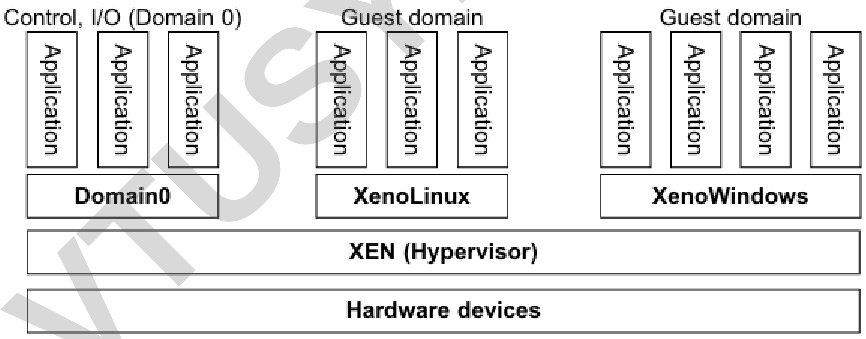


FIGURE 3.5

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

(Courtesy of P. Barham, et al. [7])

Domain 0 (Dom0):

- **Privileged guest OS** with **direct hardware access.**
- Manages guest OS instances (Domain U).
- Controls **resource allocation and device management.**

Domain U (DomU):

- **Unprivileged guest OS instances** running under Xen.
- Cannot access hardware directly.

Security Considerations

- **Domain 0 is the most critical component.** If compromised, the attacker gains full control over all VMs.
- **Security policies are required** to protect Domain 0.

VM State Management and Rollback

Unlike traditional machines (which follow a linear execution path), **VM execution follows a tree structure** where multiple instances can be created at different states.

Benefits of VM state rollback:

- **Error recovery** (rollback to a previous working state).
- **Efficient system distribution** (duplicate VMs for dynamic content).
- **Live migration** (moving running VMs between hosts).

Challenges:

- Security risks in handling VM snapshots and rollbacks.
- Need for strict **access control and auditing.**

Binary Translation with Full Virtualization

Depending on implementation technologies, hardware virtualization can be classified into two cate gories: full virtualization and host-based virtualization.

Full virtualization does not need to modify the host OS. It relies on binary translation to trap and to virtualize the execution of certain sensitive, nonvirtualizable instructions.

The guest OSes and their applications consist of noncritical and critical instructions. In a host-based system, both a host OS and a guest OS are used.

A virtualization software layer is built between the host OS and guest OS. These two classes of VM architecture are introduced next.

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. Why are only critical instructions trapped into the VMM? 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.

Binary Translation Using VMM (Virtual Machine Monitor)

- **Implemented by VMware and other vendors.**
- **VMM is placed at Ring 0** (privileged mode).
- **Guest OS runs at Ring 1**, unaware that it is virtualized.
- **VMM scans and translates** privileged instructions **before execution**.
- **Code caching** helps optimize performance but **increases memory usage**.

Host-Based Virtualization

Runs on a host OS rather than directly on hardware.

The **virtualization layer sits between the host OS and guest OS**.

- Guest OSes and applications can run inside VMs, while other applications can run directly on the host OS.

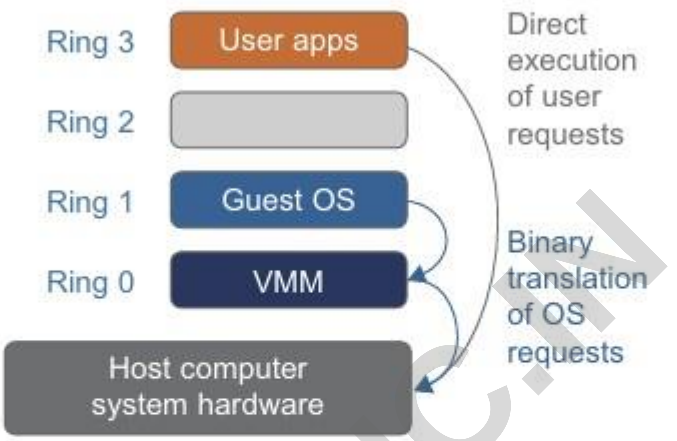

Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

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.

(Courtesy of VM Ware [71])

Advantages:

- **Easier deployment** (no need to modify the host OS).
- **Simplified design** (relies on the host OS for device drivers).
- Works on **various hardware configurations**.

Disadvantages:

- **Lower performance** due to multiple layers of hardware access.
- **Requires binary translation** if guest OS and host hardware have different ISAs.
- **High overhead**, making it less efficient in practice.

Para-Virtualization with Compiler Support Overview

of Para-Virtualization

- **Requires modification of the guest OS kernel** to support virtualization.
- **Provides special APIs (hypercalls)** to replace non-virtualizable OS instructions.
- **Reduces virtualization overhead**, improving performance compared to full virtualization.
- Unlike full virtualization, which relies on **binary translation**, para-virtualization requires **OS kernel modifications**.

Para-Virtualization Architecture

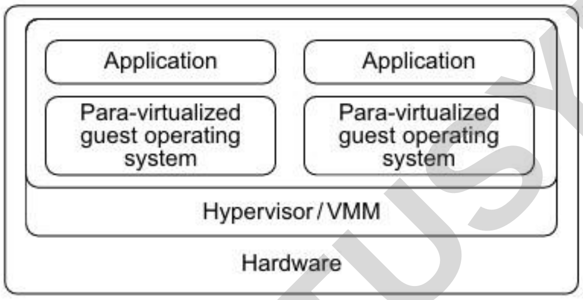

Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor / VMM
Hardware

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.)

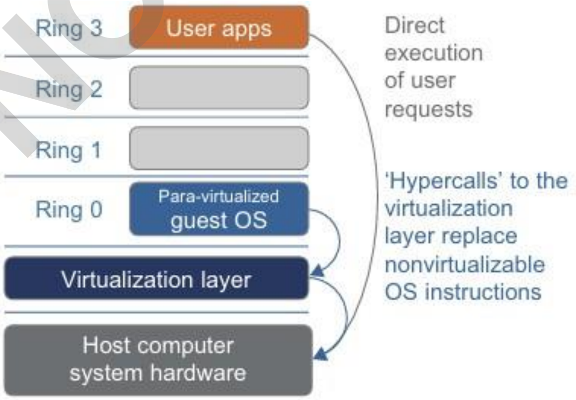

Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

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])

A **virtualization layer is inserted between hardware and OS**.

In the **x86 architecture**, the OS typically runs at **Ring 0** for privileged operations, while applications run at **Ring 3**.

Para-virtualization modifies the guest OS to:

- Run at **Ring 1** instead of Ring 0.

- Replace **non-virtualizable instructions** with **hypercalls** to the **hypervisor/VMM**.

Advantages of Para-Virtualization

- **Improves performance** by eliminating the need for complex **binary translation**.
- **More efficient than full virtualization**, especially for workloads with frequent privileged instructions.
- **Used by popular hypervisors** like **Xen, KVM, and VMware ESX**.

Challenges of Para-Virtualization

- **Requires modifying the OS kernel**, making it **less compatible** with unmodified OSes.
- **Maintaining para-virtualized OS versions is costly**, as OS updates require modifications.
- **Performance benefits depend on workload types**—some workloads benefit greatly, while others do not.

KVM (Kernel-Based VM)

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel.

The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine.

KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

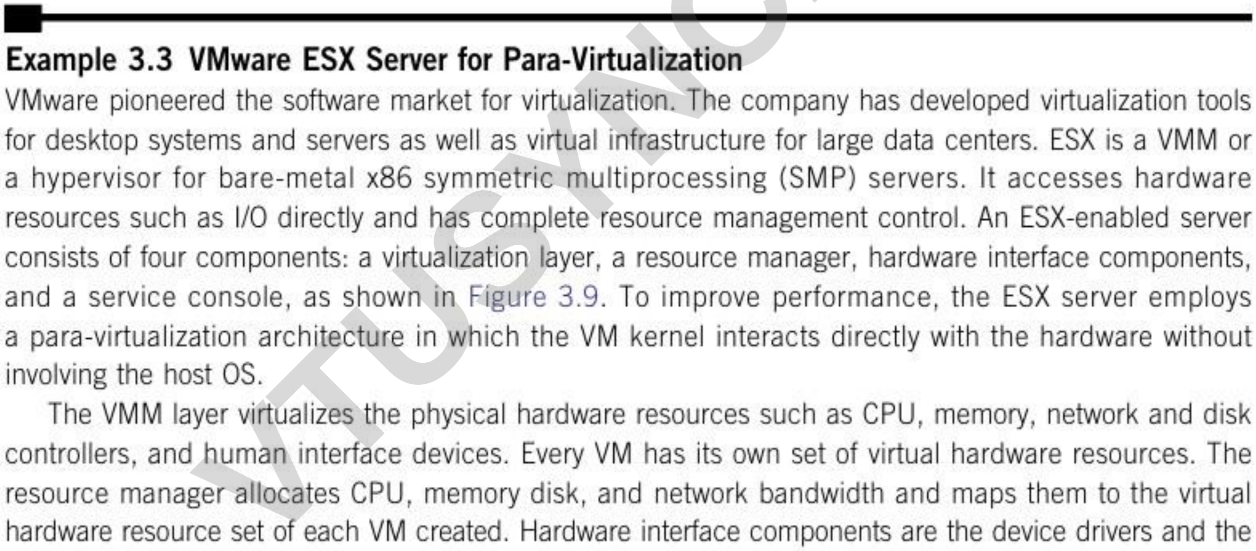
Para-Virtualization with Compiler Support

Unlike full virtualization, which traps privileged instructions at runtime, para-virtualization modifies instructions at compile time.

The OS kernel replaces privileged instructions with hypercalls before execution.

Xen follows this architecture, where:

- The **guest OS runs at Ring 1** instead of Ring 0.
- **Privileged instructions are replaced with hypercalls** to the hypervisor.
- Hypercalls function similarly to **system calls in UNIX** (using service routines).



Example 3.3 VMware ESX Server for Para-Virtualization

VMware pioneered the software market for virtualization. The company has developed virtualization tools for desktop systems and servers as well as virtual infrastructure for large data centers. ESX is a VMM or a hypervisor for bare-metal x86 symmetric multiprocessing (SMP) servers. It accesses hardware resources such as I/O directly and has complete resource management control. An ESX-enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console, as shown in Figure 3.9. To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.

The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources. The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created. Hardware interface components are the device drivers and the

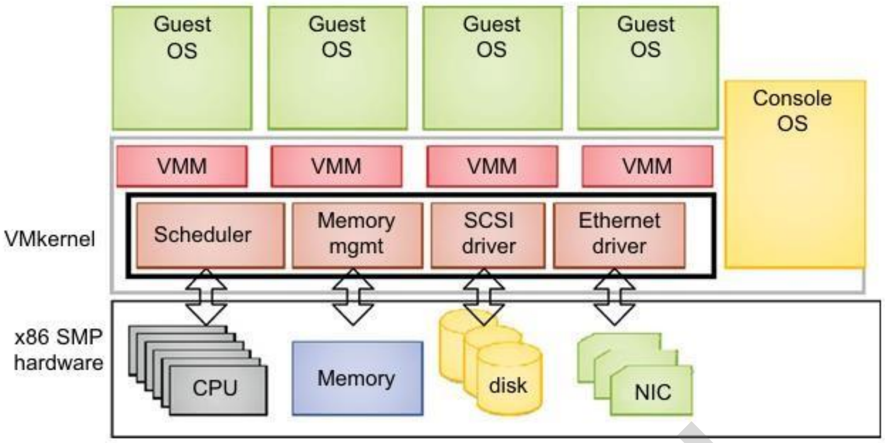


FIGURE 3.9

The VMware ESX server architecture using para-virtualization.

(*Courtesy of VMware [71]*)

VMware ESX Server File System. The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.

irtualization of CPU/Memory and I/O devices

Introduction to Virtualization Support in Hardware

Modern processors (e.g., x86) use hardware-assisted virtualization to support virtual machines efficiently.

The Virtual Machine Monitor (VMM) and guest OS operate in separate modes, ensuring security and isolation.

Sensitive instructions of the guest OS are trapped in the VMM, preventing unauthorized hardware access.

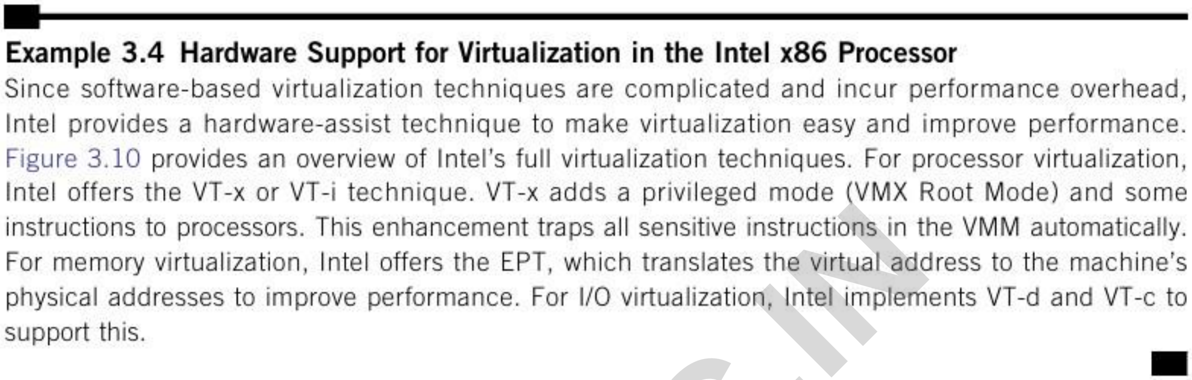
Hardware Support for Virtualization

Processors have two main execution modes:

User Mode: Runs applications with limited access to hardware.

- **Supervisor Mode (Privileged Mode):** Runs the OS kernel and handles critical system operations.

Virtualization complicates execution because multiple OSes run on a single machine.



Example 3.4 Hardware Support for Virtualization in the Intel x86 Processor

Since software-based virtualization techniques are complicated and incur performance overhead, Intel provides a hardware-assist technique to make virtualization easy and improve performance. Figure 3.10 provides an overview of Intel's full virtualization techniques. For processor virtualization, Intel offers the VT-x or VT-i technique. VT-x adds a privileged mode (VMX Root Mode) and some instructions to processors. This enhancement traps all sensitive instructions in the VMM automatically. For memory virtualization, Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance. For I/O virtualization, Intel implements VT-d and VT-c to support this.

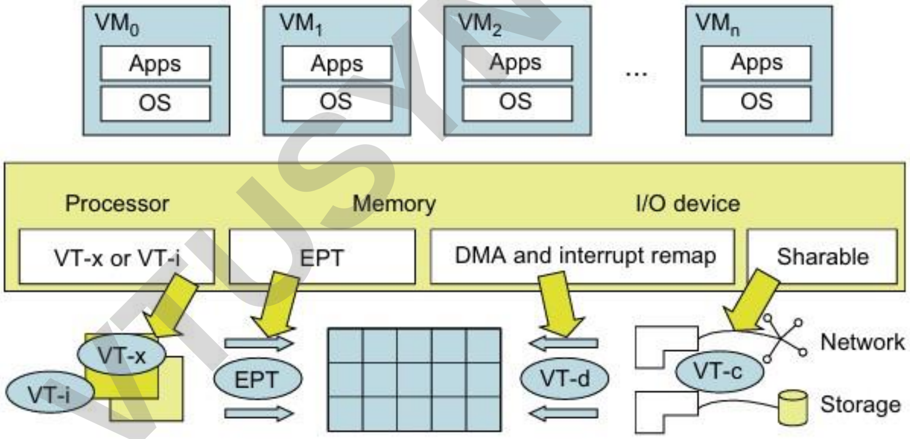


FIGURE 3.10

Intel hardware support for virtualization of processor, memory, and I/O devices.

(Modified from [68], Courtesy of Lizhong Chen, USC)

Examples of hardware-assisted virtualization tools:

- **VMware Workstation** (host-based virtualization).
- **Xen** (hypervisor that modifies Linux as the lowest privileged layer).
- **KVM** (uses Intel VT-x or AMD-V for efficient virtualization).
- **VirtIO** (provides virtualized I/O devices like Ethernet, disk, memory ballooning, and VGA).

CPU Virtualization

VMs execute most instructions in native mode for efficiency, except critical instructions.

Critical instructions are classified into three categories:

1. **Privileged Instructions:** Only execute in privileged mode (Ring 0).
2. **Control-Sensitive Instructions:** Modify system settings or resources.
3. **Behavior-Sensitive Instructions:** Depend on system configuration (e.g., memory access).

CPU virtualization requires trapping privileged instructions so that the VMM can handle them securely.

RISC architectures are naturally virtualizable, as all sensitive instructions are privileged.

x86 architecture is not naturally virtualizable because some sensitive instructions (e.g., SGDT, SMSW) are not privileged and cannot be trapped by the VMM.

Example: System Calls in UNIX and Xen

- **In UNIX systems, system calls trigger the 0x80 interrupt, passing control to the kernel.**
- **In Xen (a para-virtualization system), system calls trigger both 0x80 (guest OS) and 0x82 (hypervisor).**
- **The hypervisor processes privileged operations before returning control to the guest OS.**

Hardware-Assisted CPU Virtualization

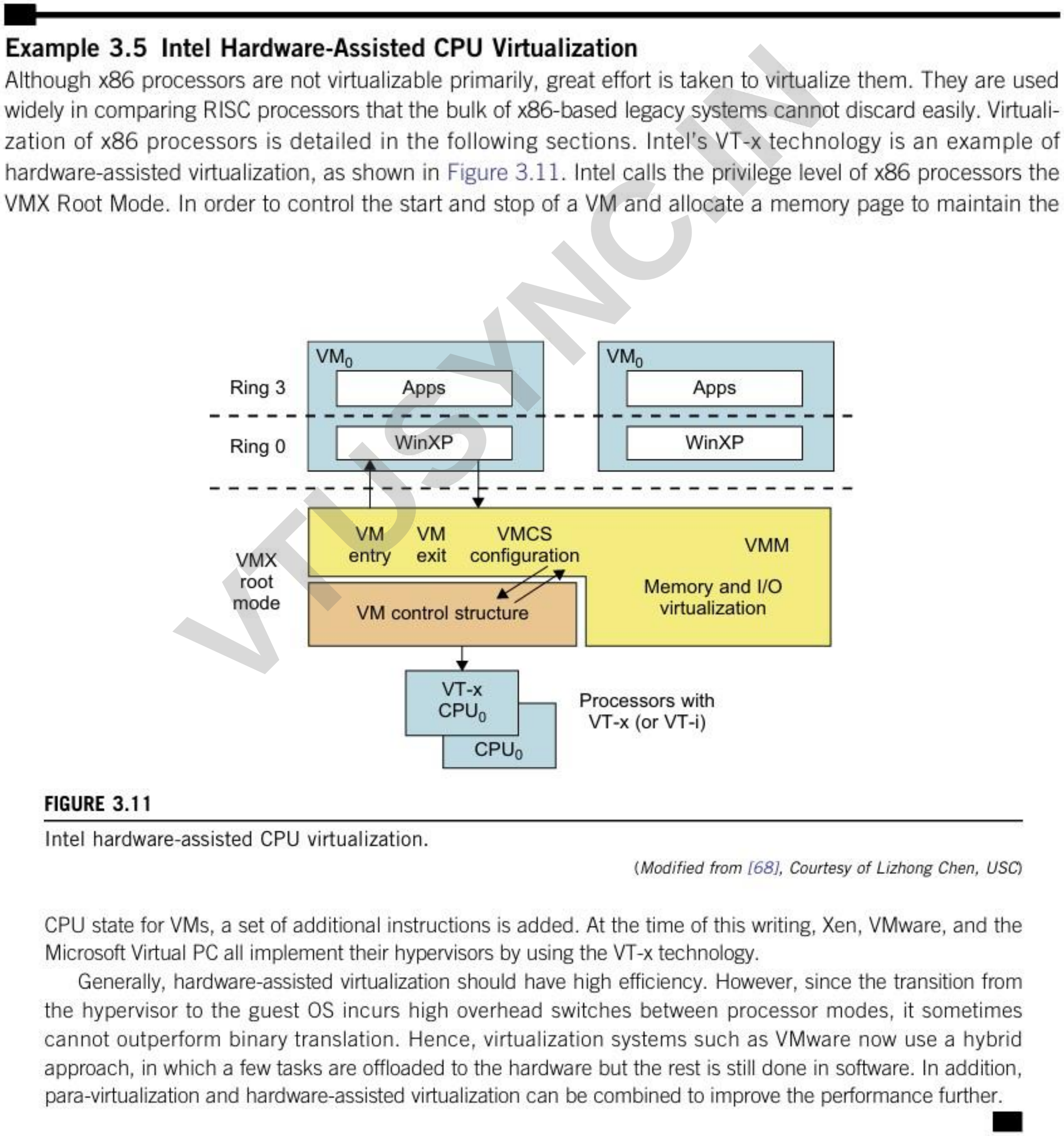
Intel and AMD introduced an additional privilege mode (Ring -1) for virtualization.

Now, the hypervisor runs at Ring -1, while the guest OS runs at Ring 0.

This eliminates the need for complex binary translation (used in full virtualization).

Benefits:

- Simplifies virtualization implementation.
- Allows OSes to run in VMs without modification.
- Traps all privileged instructions in the hypervisor automatically.



Example 3.5 Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtualization of x86 processors is detailed in the following sections. Intel's VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the

FIGURE 3.11

Intel hardware-assisted CPU virtualization.

(*Modified from [68], Courtesy of Lizhong Chen, USC*)

CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

Memory Virtualization

Virtual Memory Mapping in Traditional Systems

- The OS maps virtual memory to machine memory using **page tables** (one-stage mapping).
- Modern x86 CPUs use an **MMU (Memory Management Unit)** and **TLB (Translation Lookaside Buffer)** to optimize memory performance.

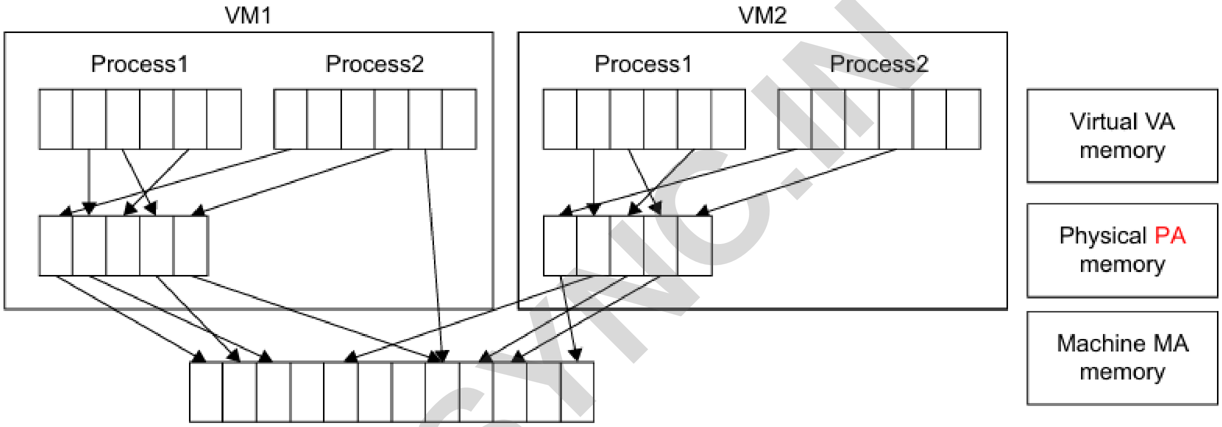


FIGURE 3.12

Two-level memory mapping procedure.

(*Courtesy of R. Rblig, et al.* [68])

Memory Virtualization in Virtualized Environments

- Physical RAM is shared and dynamically allocated among Virtual Machines (VMs).
- A **two-stage mapping** is required:
  - **Guest OS**: Maps virtual memory to guest physical memory.
  - **VMM (Hypervisor)**: Maps guest physical memory to actual machine memory.

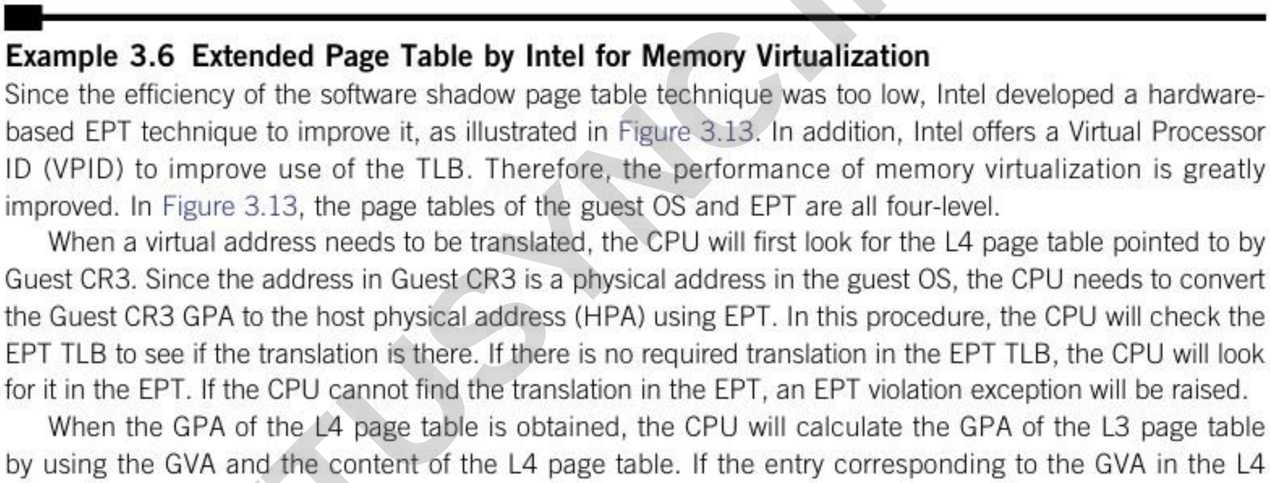
Shadow Page Tables & Nested Paging

- Each guest OS page table has a corresponding **shadow page table** maintained by the VMM.

- This additional layer leads to performance overhead and high memory costs.
- **Nested Paging (Hardware-Assisted Virtualization)**:
    - Reduces the overhead of shadow page tables.
    - Introduced by AMD’s **Barcelona processor (2007)**.

Optimizing Virtual Memory Performance

- VMware uses **shadow page tables** for address translation.
- **TLB hardware** enables direct mapping from virtual memory to machine memory, reducing translation overhead.



Example 3.6 Extended Page Table by Intel for Memory Virtualization

Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.

When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4

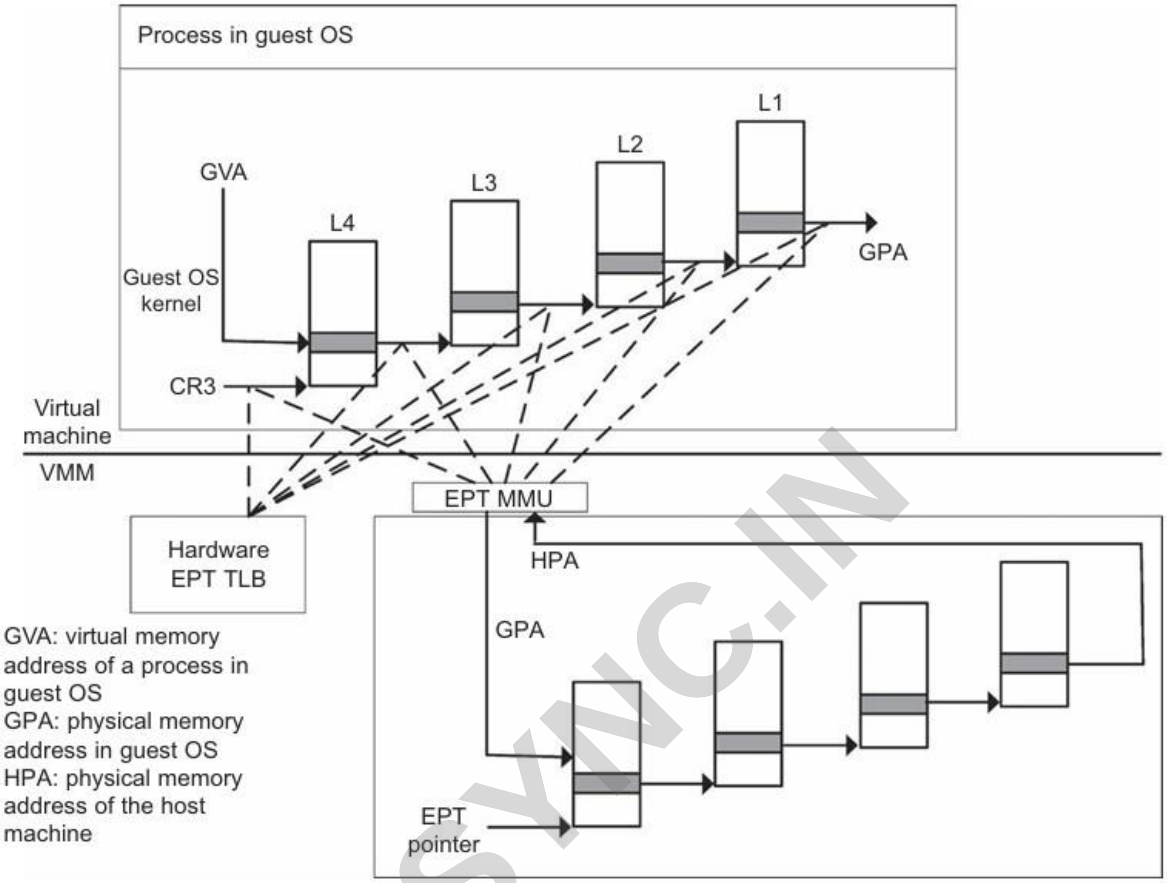

Process in guest OS
L1
L2
L3
L4
GVA
GPA
Guest OS kernel
CR3
Virtual machine
VMM
EPT MMU
Hardware EPT TLB
HPA
GPA
EPT pointer
GVA: virtual memory address of a process in guest OS
GPA: physical memory address in guest OS
HPA: physical memory address of the host machine

FIGURE 3.13

Memory virtualization using EPT by Intel (the EPT is also known as the shadow page table [68]).

page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page table, as described earlier. To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. Therefore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this shortcoming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

I/O Virtualization & Multi-Core Virtualization I/O

Virtualization

I/O virtualization manages routing of I/O requests between virtual devices and shared physical hardware. There are three main approaches:

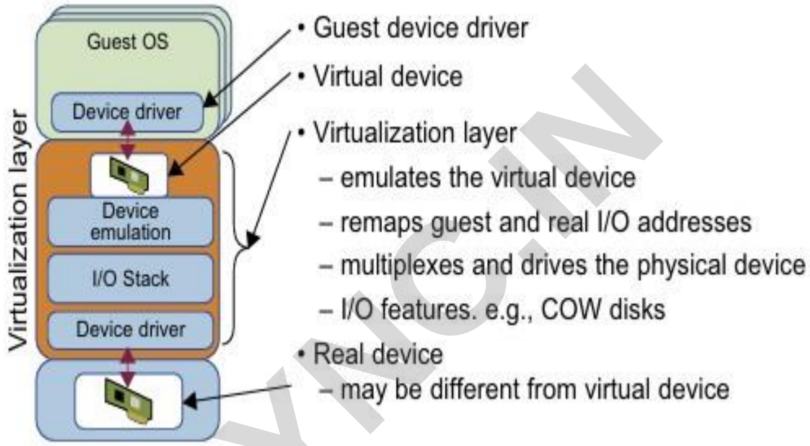

Guest OS
Device driver
Virtualization layer
Device emulation
I/O Stack
Device driver
• Guest device driver
• Virtual device
• Virtualization layer
– emulates the virtual device
– remaps guest and real I/O addresses
– multiplexes and drives the physical device
– I/O features, e.g., COW disks
• Real device
– may be different from virtual device

FIGURE 3.14

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

(Courtesy of V. Chadha, et al. [10] and Y. Dong, et al. [15])

Full Device Emulation

- Emulates real-world devices in software within the **VMM (hypervisor)**.
- The guest OS interacts with virtual devices, and the VMM handles I/O operations.
- **Drawback**: High overhead and lower performance compared to real hardware.

Para-Virtualization (Split Driver Model - Used in Xen)

- Uses **frontend and backend drivers** to handle I/O:
  - **Frontend driver**: Manages I/O requests in the guest OS.

    - **Backend driver**: Runs in the privileged domain (Domain 0) and manages real I/O devices.
- **Pros**: Better performance than full emulation.
- **Cons**: Higher CPU overhead.

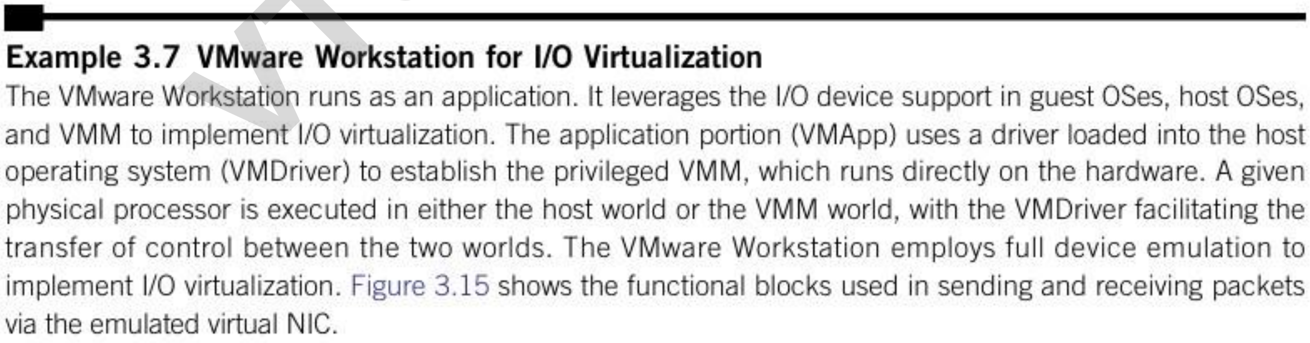
Direct I/O Virtualization

- Allows VMs to directly access physical devices.
- **Pros**: Close-to-native performance, lower CPU cost.
- **Cons**: Limited support for commodity hardware, potential system crashes during workload migration.

Hardware-Assisted I/O Virtualization

Intel VT-d helps remap I/O DMA transfers and device interrupts, allowing direct device access for VMs.

Self-Virtualized I/O (SV-IO) uses multi-core processors to virtualize I/O devices, providing an efficient API for virtualized systems.



Example 3.7 VMware Workstation for I/O Virtualization

The VMware Workstation runs as an application. It leverages the I/O device support in guest OSes, host OSes, and VMM to implement I/O virtualization. The application portion (VMApp) uses a driver loaded into the host operating system (VMDriver) to establish the privileged VMM, which runs directly on the hardware. A given physical processor is executed in either the host world or the VMM world, with the VMDriver facilitating the transfer of control between the two worlds. The VMware Workstation employs full device emulation to implement I/O virtualization. Figure 3.15 shows the functional blocks used in sending and receiving packets via the emulated virtual NIC.

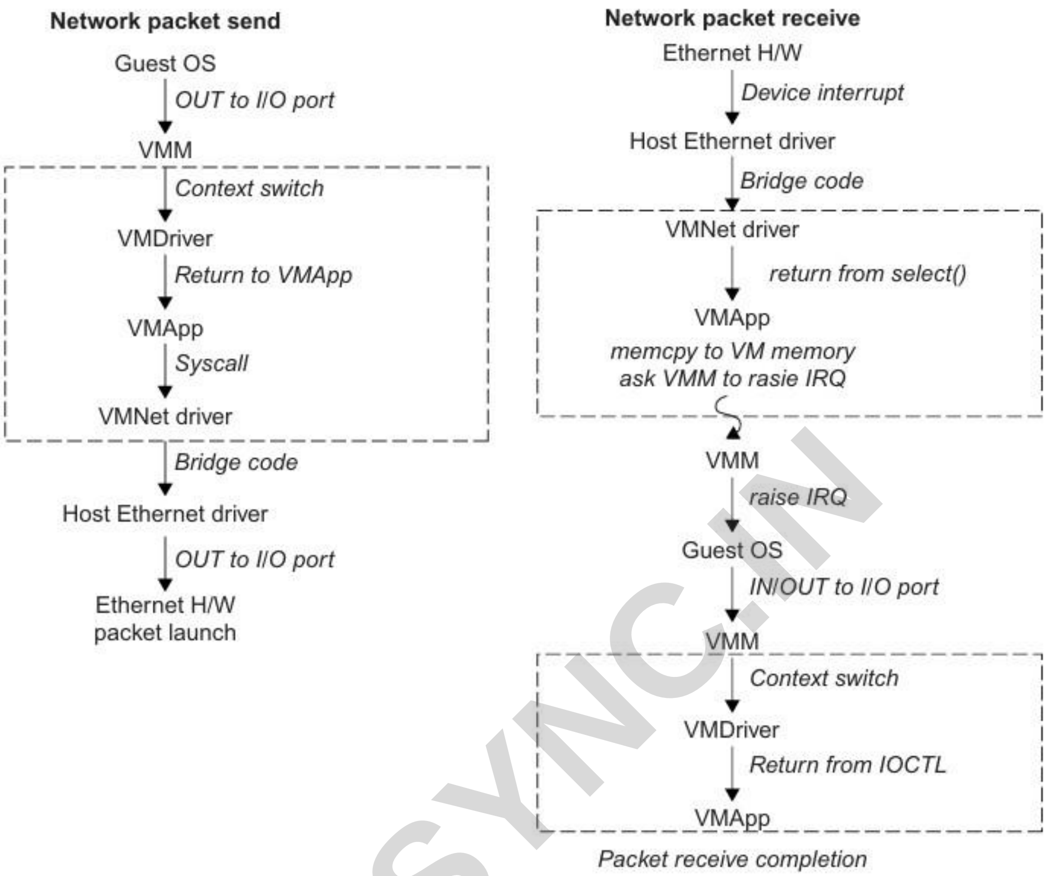


FIGURE 3.15

Functional blocks involved in sending and receiving network packets.

(*Courtesy of VMWare [71]*)

The virtual NIC models an AMD Lance Am79C970A controller. The device driver for a Lance controller in the guest OS initiates packet transmissions by reading and writing a sequence of virtual I/O ports; each read or write switches back to the VMApp to emulate the Lance port accesses. When the last OUT instruction of the sequence is encountered, the Lance emulator calls a normal *write()* to the VMNet driver. The VMNet driver then passes the packet onto the network via a host NIC and then the VMApp switches back to the VMM. The switch raises a virtual interrupt to notify the guest device driver that the packet was sent. Packet receives occur in reverse.

Multi-Core Virtualization

Virtualizing multi-core processors is more complex than uni-core processors due to:

Parallelization Challenges:

- Applications must be explicitly parallelized to utilize all cores efficiently.
- New programming models, languages, and libraries are needed.

Task Scheduling Complexity:

- Scheduling algorithms and resource management policies must optimize performance while handling core assignments.

Dynamic Heterogeneity

- New architectures mix **fat CPU cores** and **thin GPU cores** on the same chip.
- Hardware reliability issues and increased complexity in transistor management make resource allocation more difficult.

Physical vs. Virtual Processor Cores

Virtual CPU (VCPU) Migration:

- Wells et al. proposed a method where VCPUs can move between cores dynamically.
- Reduces inefficiencies in managing processor cores by software.
- Located **below the ISA**, making it transparent to OS and hypervisors.

Virtual Hierarchy in Many-Core Processors

Virtual Hierarchy

- **Many-core chip multiprocessors (CMPs)** enable **space-sharing**, where different jobs are assigned to separate groups of cores for long intervals.
- **Virtual hierarchy** is a **dynamic cache hierarchy** that adapts to workload demands, unlike static physical cache hierarchies.
- Proposed by **Marty and Hill**, this method optimizes **performance isolation** and **cache coherence**.

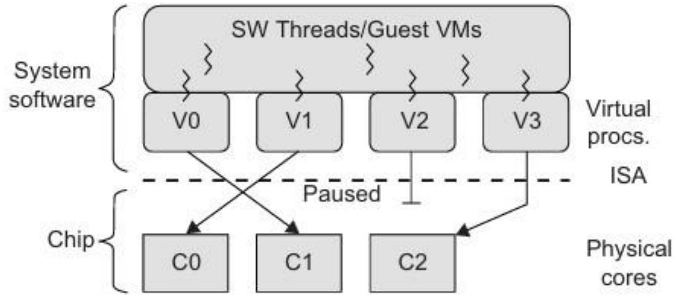


FIGURE 3.16

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.

(Courtesy of Wells, et al. [74])

How Virtual Hierarchy Works

Many-core CMPs typically use **physical cache hierarchies** (L1, L2) with static allocation.

A **virtual hierarchy** dynamically adapts cache levels to workload needs, improving access speed and reducing interference.

benefits:

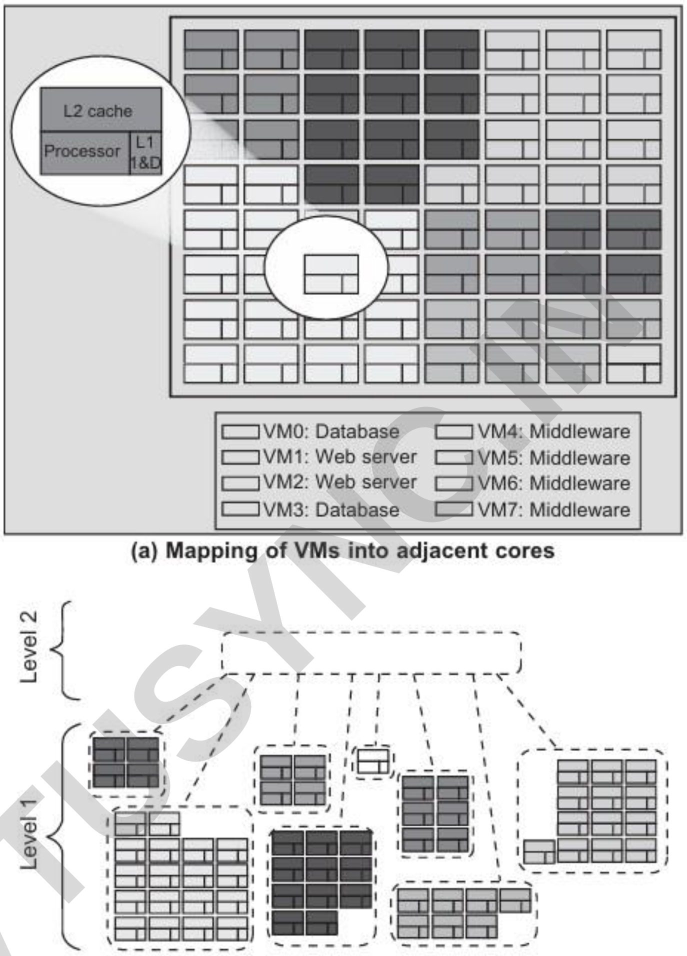
1. **Locates data blocks close to cores** for faster access.
2. **Establishes shared-cache domains** to minimize data transfer delays.
3. **Reduces performance interference** between different workloads.

Space-Sharing Workload Assignment

Workloads are grouped into **virtual clusters** of cores, each assigned to different virtual machines (VMs):

- **VM0 & VM3** → Database workload
- **VM1 & VM2** → Web server workload
- **VM4–VM7** → Middleware workload

Each VM operates in isolation, minimizing cache misses and ensuring efficient resource allocation.



(a) Mapping of VMs into adjacent cores

(b) Multiple virtual clusters assigned to various workloads

FIGURE 3.17

CMP server consolidation by space-sharing of VMs into many cores forming multiple virtual clusters to execute various workloads.

(Courtesy of Marty and Hill [39])

Two-Level Virtual Coherence & Caching Hierarchy

First level:

- Each **VM cluster** operates in isolation.
- Reduces cache miss access time.

- Prevents resource contention between workloads.

Second level:

- Maintains a **globally shared memory** for all VMs.
- Enables **dynamic resource repartitioning** without cache flushes.
- Supports **content-based page sharing** for efficient memory management.

Use Cases & Advantages

- Optimizes multiprogramming & server consolidation workloads.
- Improves system adaptability without major OS or hypervisor changes.
- Ensures performance scalability in many-core processors.

Virtual Clusters and Resource Management

Definition and Overview

- **Physical Clusters**: A collection of **physical servers** interconnected via a **physical network (e.g., LAN)**.
- **Virtual Clusters**: A group of **VMs (Virtual Machines)** running on **distributed physical servers**, interconnected through a **virtual network**.

Design Issues in Virtual Clusters

Live Migration of VMs

- Moving **VMs between physical machines** without downtime.
- Ensures **load balancing**, **fault tolerance**, and **resource optimization**.

Memory and File Migrations

- **Efficient movement of VM memory and file data** across different servers.
- Avoids **performance bottlenecks** and ensures **seamless transitions**.

Dynamic Deployment of Virtual Clusters

- Enables **scalable** and **on-demand provisioning** of VMs.
- Adjusts resources dynamically to **avoid overloading or underutilization**.

Virtual Cluster Configuration and Management

Traditional VM Setup:

- Requires **manual configuration** by an administrator.
- Poor configurations may lead to **performance issues** (e.g., overloading, underutilization).

Cloud-based Virtualization (Example: Amazon EC2)

- **Elastic Computing**: Allows users to **dynamically create, manage, and scale** VMs.
- **User Account Management**: Customers can control VM resources **over time**.

Virtualization Platforms & Bridging Mode

- Platforms like **XenServer** and **VMware ESX Server** support **bridging mode**.
- In bridging mode, all **VMs appear as individual network hosts**.
- VMs can **freely communicate** over the **virtual network interface** and **self-configure**.

Physical vs. Virtual Clusters Physical

Clusters vs. Virtual Clusters

Physical Clusters: Comprise **multiple physical servers** interconnected via **physical networks**.

Virtual Clusters: Comprise **VMs distributed across multiple physical servers** and connected through a **virtual network**.

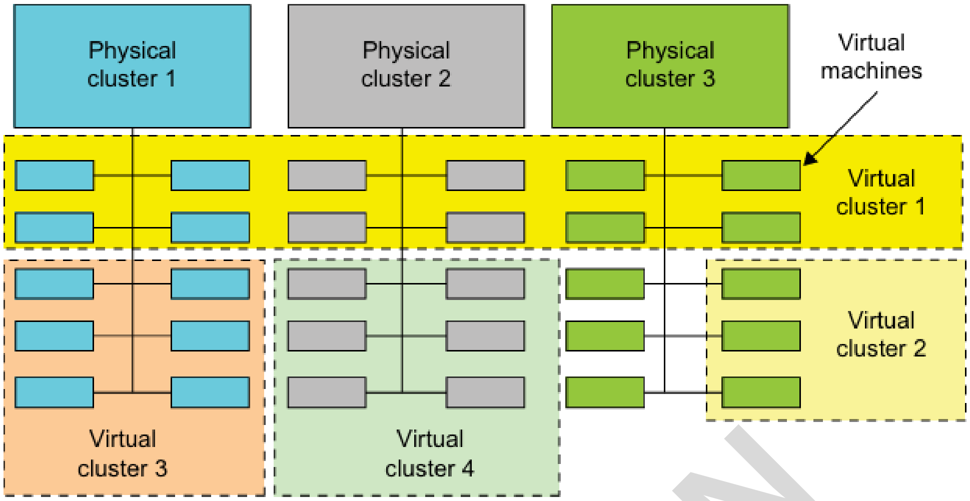


FIGURE 3.18

A cloud platform with four virtual clusters over three physical clusters shaded differently.

(Courtesy of Fan Zhang, Tsinghua University)

Properties of Virtual Clusters

Flexible Node Configuration

- Virtual clusters can include **both physical and virtual machines**.
- A single physical machine can host **multiple VMs** with different **Operating Systems (OSes)**.

Guest vs. Host OS

- Each **VM runs a guest OS**, which may differ from the **host OS** of the physical machine.

Resource Consolidation & Utilization

- VMs help in **consolidating multiple applications** on the same physical server.
- Enhances **server utilization** and improves **application flexibility**.

Distributed Parallelism & Fault Tolerance

- VMs can be **replicated across multiple servers** for better **fault tolerance** and **disaster recovery**.
- If a **physical node fails**, only the VMs running on that node are affected.
- A VM failure **does not impact the host system**.

Scalability & Dynamic Allocation

- The number of nodes in a virtual cluster can **increase or decrease dynamically**, similar to P2P networks.

Virtual Cluster Management & Storage

Efficient VM Deployment & Monitoring: Requires techniques like resource scheduling, load balancing, server consolidation, and fault tolerance.

VM Image Storage:

- A large number of VM images must be stored **efficiently**.
- **Template VMs** can be used to **pre-install common software**, reducing redundancy.
- Users can **customize OS instances** by adding specific **libraries and applications**.

Dynamic Cluster Boundaries

Virtual clusters are **flexible**:

- VMs can be **moved, added, or removed** dynamically.
- They can span **multiple physical clusters** and **adapt to changing workloads**.

Fast Deployment & Scheduling

- Fast deployment involves quickly setting up OS, libraries, and applications on physical nodes.
- VM runtime environments should switch efficiently between different users to optimize resources.
- **Green computing** aims to minimize energy consumption across the cluster, not just on single nodes.
- **Live VM migration** shifts workloads between nodes but can introduce overhead affecting performance.
- Load balancing improves resource utilization and system response times.

High-Performance Virtual Storage

- VMs use **template images** (pre-installed OS and software) to reduce setup time.
- Copy-on-Write (COW) technique minimizes disk space usage by creating small, efficient backup files.
- Storage management should reduce duplicate blocks to optimize disk usage in virtual clusters.

Live VM Migration Steps and Performance Effects

Overview of VM Migration

- In mixed host-guest clusters, **physical nodes** run tasks directly, while **VMs** serve as failover replacements.
- VM failover is more flexible than traditional physical failover but depends on the host's availability.
- **Live VM migration** enables a running VM to move between hosts without service interruption.

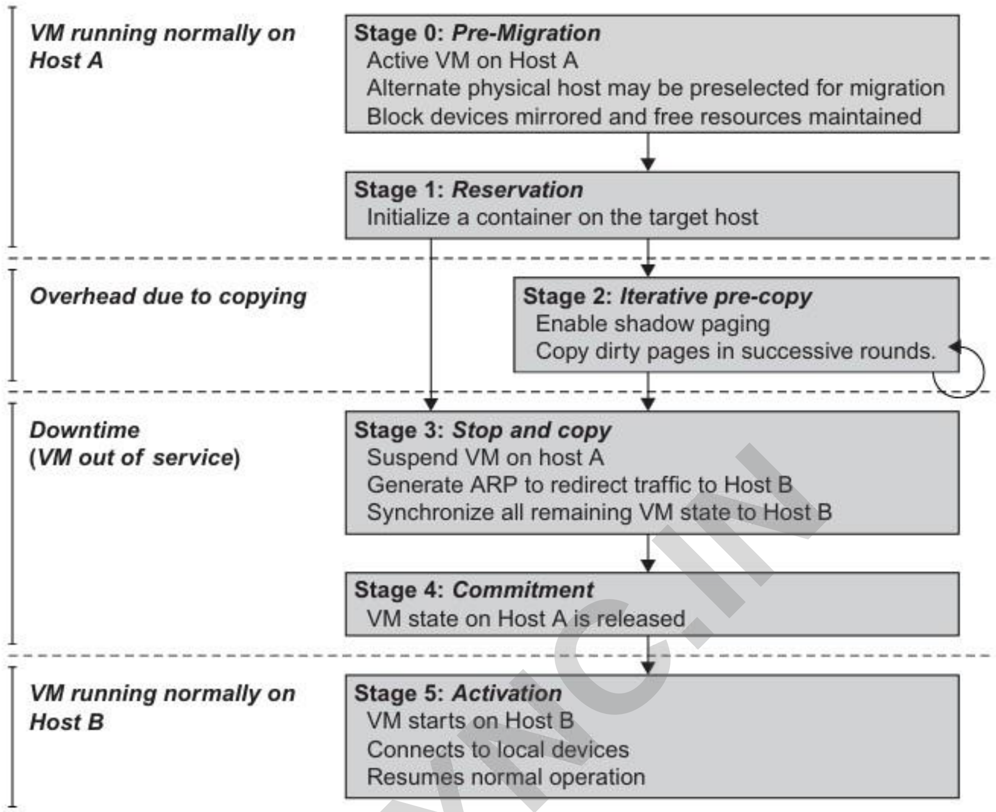

VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime
(VM out of service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

FIGURE 3.20

Live migration process of a VM from one host to another.

(*Courtesy of C. Clark, et al.* [14])

Steps of Live VM Migration

1. **Start Migration (Steps 0 & 1)**
   - Identify the VM to migrate and select the destination host.
   - Migration is triggered automatically (e.g., for load balancing or server consolidation).
2. **Memory Transfer (Step 2)**
   - The entire VM memory is copied to the new host in multiple rounds.
   - Changed memory pages are recopied iteratively until only a small portion remains.
3. **Suspend & Final Data Copy (Step 3)**
   - The VM is temporarily stopped while transferring the final memory pages, CPU, and network states.

- This causes **downtime**, which should be minimized for user experience.

4. **Commit & Activate VM (Steps 4 & 5)**
   - The VM is restored on the new host, resuming execution.
   - The network is redirected to the new VM, and the old one is removed.

Performance Effects of Migration

Data Throughput Impact:

- Before migration: **870 MB/s**
- First memory pre-copy (63 sec): **765 MB/s**
- Further iterations (9.8 sec): **694 MB/s**

Downtime: Only **165 milliseconds**, ensuring minimal disruption.

Minimal Migration Overhead: Critical for **dynamic cluster reconfiguration and disaster recovery**, especially in **cloud computing**.

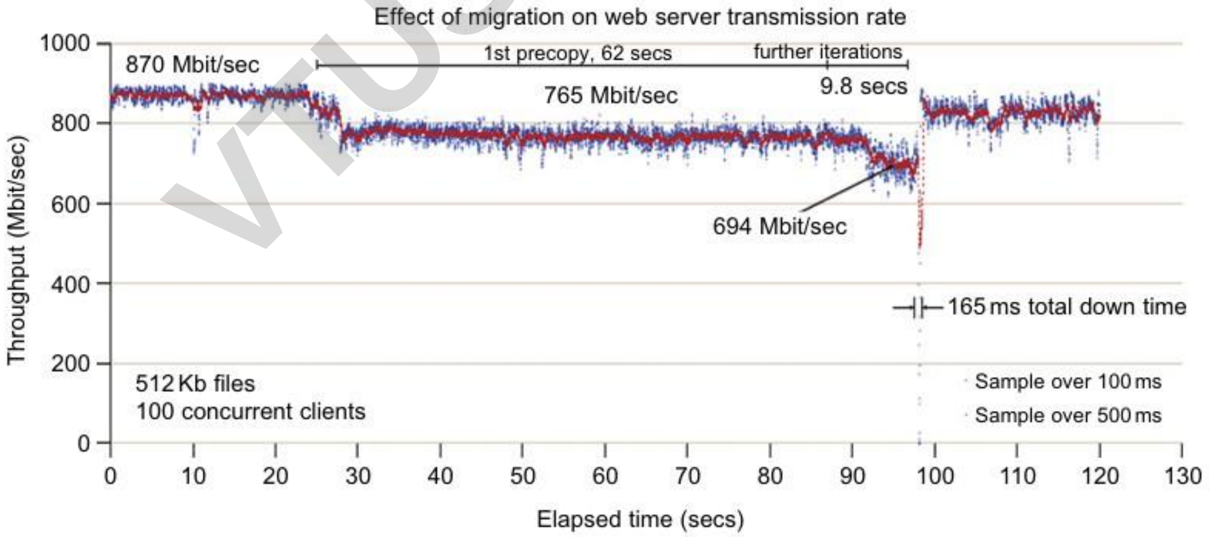


FIGURE 3.21

Effect on data transmission rate of a VM migrated from one failing web server to another.

(Courtesy of C. Clark, et al. [14])

Virtual Cluster Management Approaches

1. **Guest-Based Manager:** Runs within VMs (e.g., openMosix on Xen, Solaris cluster on VMware).
2. **Host-Based Manager:** Runs on physical hosts and can restart VMs after failure (e.g., VMware HA).
3. **Independent Cluster Manager:** Manages both host and guest systems, increasing complexity.
4. **Integrated Cluster Management:** Differentiates between virtual and physical resources for optimal efficiency.

Role of Virtual Clusters

- Used in **cloud computing, HPC, and computational grids**.
- Enables **dynamic resource allocation, quick failover, and efficient workload management**.
- Live migration helps **maintain uptime, reduce network congestion, and support flexible resource scaling**.

Migration of Memory, Files, and Network Resources

Introduction

Due to the high initial cost of clusters—including space, power, and cooling—**leasing or sharing clusters** is a cost-effective approach.

Shared clusters improve resource utilization through **multiplexing** and economies of scale.

Early configuration and management systems help define **service-specific clusters** and allocate physical nodes accordingly.

When migrating a system to another physical node, several key considerations must be addressed.

Memory Migration

Memory migration is a critical aspect of VM migration, as moving a VM's memory from one host to another must be done efficiently.

Memory transfer sizes typically range from hundreds of megabytes to several gigabytes.

Internet Suspend-Resume (ISR) Technique

The ISR technique exploits temporal locality, meaning that memory states in the suspended and resumed VM instances are largely similar.

Mechanism:

- Each file is represented as a tree of small subfiles.
- Both the suspended and resumed VM instances contain a copy of this tree.
- Only changed files are transmitted, reducing data transfer.

Limitations:

- ISR is useful when live migration is not required.
- Downtime is relatively high compared to live migration techniques.

File System Migration

For seamless VM migration, a system must ensure a **consistent, location-independent file system** across all hosts.

Approaches to File System Migration

1. Virtual Disk Mapping

- Each VM is assigned a **virtual disk** that contains the file system.
- The entire disk is moved along with the VM state.
- **Challenges:** High-capacity disks make full disk migration impractical due to **network overhead**.

2. Global Distributed File System

- A **network-accessible file system** eliminates the need for file transfers.
- This reduces migration time and enhances accessibility.

3. ISR-Based Distributed File System

- ISR uses a **distributed file system** as a transport mechanism for VM state transfer.
- The actual file systems **are not mapped directly** onto the distributed system.
- Instead, relevant files are copied **into and out of the local file system** during suspend and resume operations.
- **Advantages:**
  - Simplifies implementation by avoiding direct dependency on distributed file system semantics.
- **Challenges:**
  - The **VMM must store VM virtual disk contents locally**, which must be moved with the VM state.

Network Migration in Virtual Machine (VM) Environments

VM migration involves maintaining network connectivity for the VM without relying on forwarding mechanisms or redirection mechanisms from the original host. This is essential for uninterrupted service during migration.

Some Key Concepts:

Virtual IP and MAC Addresses

- Each VM is assigned a **virtual IP address** and **MAC address**, which are **distinct from the host machine's** address.
- These addresses must be maintained during migration for network communication.
- The **Virtual Machine Monitor (VMM)** maintains a mapping between the virtual IP/MAC addresses and the VM.

Network Migration Mechanism

- When a VM migrates to a new host, the migration must include all protocol states and the **IP address** of the VM.
- On a switched **LAN network**, the migrating host sends an **unsolicited ARP reply**, informing other devices that the VM's IP has moved. This allows peers to update their network configurations to route future packets to the VM's new location.
- If the VM maintains its original **Ethernet MAC address**, the network switch can automatically detect the migration to a new port without requiring further network configuration.

Live Migration of Virtual Machines

Live migration refers to the process of moving a VM from one physical node to another without interrupting the VM's operating system or applications.

This is essential for various **enterprise workloads** such as load balancing, system maintenance, and proactive fault tolerance.

Live Migration Mechanism:

The **precopy approach** is widely used in live migration, where:

- **Memory pages** are transferred to the target node in **multiple iterations**.
- The first round transfers all memory pages, followed by subsequent rounds that only transfer modified (dirty) pages.
- The VM remains **online during migration**, though there is a performance degradation due to the network bandwidth consumption.

Precopy Migration Challenges:

Performance Degradation:

- The migration daemon consumes **network bandwidth** to transfer dirty pages, leading to performance degradation.
- **Rate limiting** can mitigate performance hits, but this **prolongs the migration** process.

Convergence Issues:

- Some applications may not have small writable working sets, causing difficulties in convergence, which might require additional migration iterations.

Memory Transfer Volume:

- The large volume of data transferred during the migration process is a key limitation in **precopy-based migration**.

Checkpointing and Trace/Replay (CR/TR-Motion) Migration

To address the limitations of **precopy migration**, an alternative approach using **checkpointing and trace/replay** (CR/TR-Motion) has been proposed.

- **CR/TR-Motion** transfers an **execution trace file** rather than the dirty memory pages, significantly reducing the **amount of transferred data**.
- **Advantages:**
    - **Drastically reduces total migration time** and downtime.
    - **Log files** (execution traces) are much smaller than dirty pages, leading to a more efficient migration.
- **Limitations:**
    - The approach is effective only if the **log replay rate** exceeds the **log growth rate**. The differences between the source and target nodes may limit its effectiveness in some scenarios.

Postcopy Migration

Postcopy migration transfers all memory pages at once, reducing the baseline migration time. However, it introduces significant **downtime** due to latency as memory pages are fetched from the source node before the VM can be resumed on the target node.

Advantages:

- **Reduced total migration time** as the baseline transfer is done only once.

Challenges:

- Higher **downtime** compared to precopy due to the latency in fetching pages.

Compression-Based Optimization for Memory Migration

With advancements in **multicore** and **many-core machines**, there is a possibility to compress memory pages to reduce the data transfer required during migration.

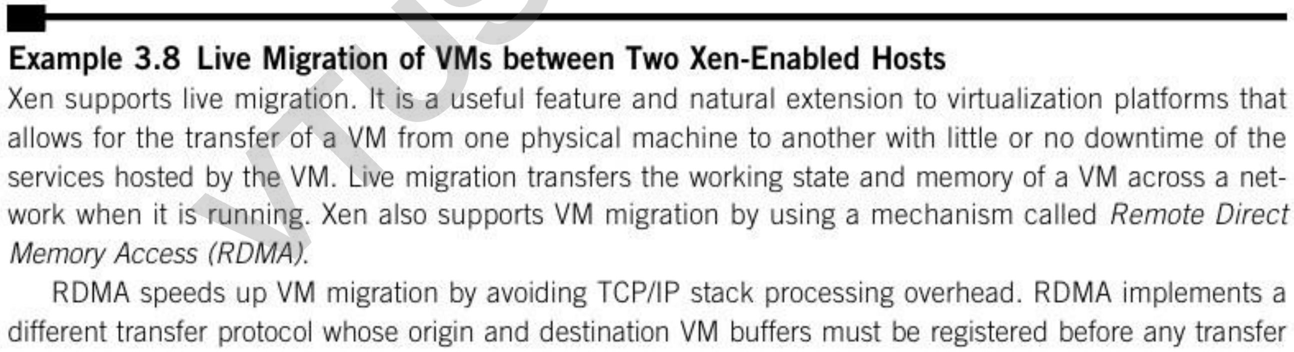
Compression Algorithms:

- **Memory compression** reduces the **amount of data** transferred during migration.
- **Decompression** is fast and doesn't require significant memory, thus improving migration efficiency.

Live Migration Using Xen Hypervisor

Xen is a widely used **Virtual Machine Monitor (VMM)** that supports live migration by utilizing a **send/recv model** to transfer VM states between source and target hosts.

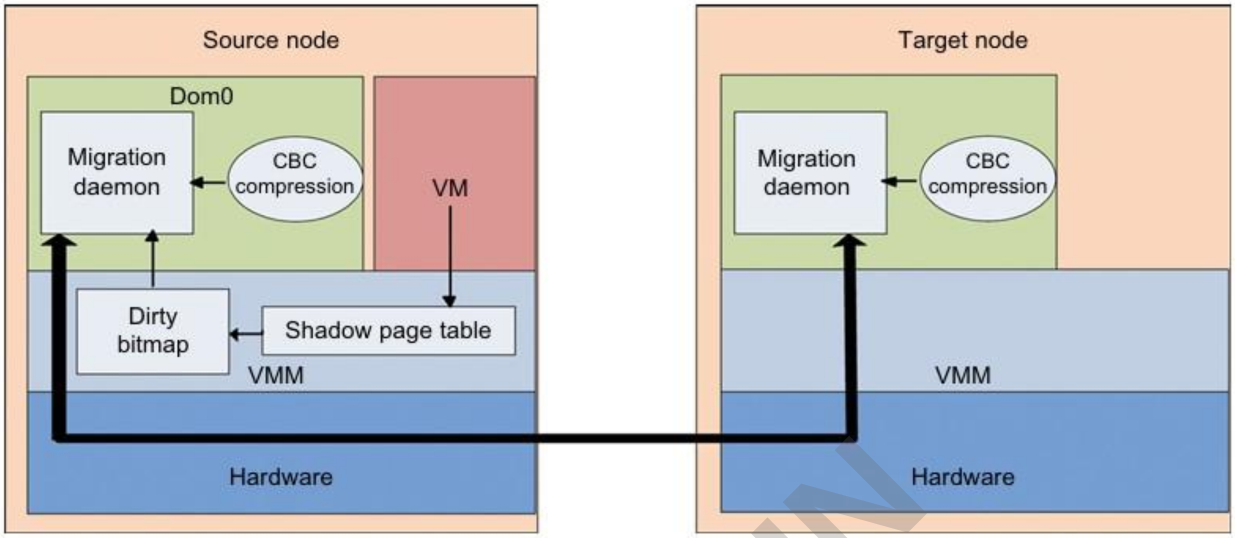
- **Dom0** (the control domain) manages the migration process, including the creation, termination, or migration of VMs across hosts.



Example 3.8 Live Migration of VMs between Two Xen-Enabled Hosts

Xen supports live migration. It is a useful feature and natural extension to virtualization platforms that allows for the transfer of a VM from one physical machine to another with little or no downtime of the services hosted by the VM. Live migration transfers the working state and memory of a VM across a network when it is running. Xen also supports VM migration by using a mechanism called *Remote Direct Memory Access (RDMA)*.

RDMA speeds up VM migration by avoiding TCP/IP stack processing overhead. RDMA implements a different transfer protocol whose origin and destination VM buffers must be registered before any transfer



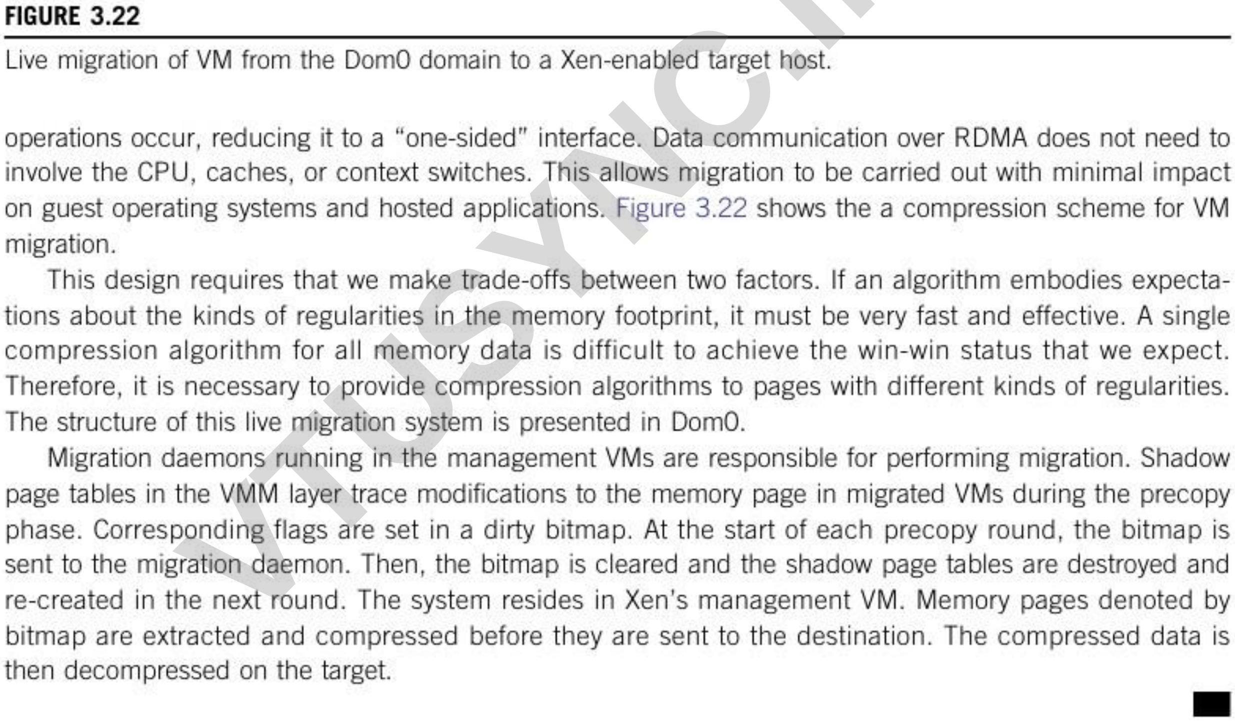


FIGURE 3.22

Live migration of VM from the Dom0 domain to a Xen-enabled target host.

operations occur, reducing it to a "one-sided" interface. Data communication over RDMA does not need to involve the CPU, caches, or context switches. This allows migration to be carried out with minimal impact on guest operating systems and hosted applications. Figure 3.22 shows the a compression scheme for VM migration.

This design requires that we make trade-offs between two factors. If an algorithm embodies expectations about the kinds of regularities in the memory footprint, it must be very fast and effective. A single compression algorithm for all memory data is difficult to achieve the win-win status that we expect. Therefore, it is necessary to provide compression algorithms to pages with different kinds of regularities. The structure of this live migration system is presented in Dom0.

Migration daemons running in the management VMs are responsible for performing migration. Shadow page tables in the VMM layer trace modifications to the memory page in migrated VMs during the precopy phase. Corresponding flags are set in a dirty bitmap. At the start of each precopy round, the bitmap is sent to the migration daemon. Then, the bitmap is cleared and the shadow page tables are destroyed and re-created in the next round. The system resides in Xen's management VM. Memory pages denoted by bitmap are extracted and compressed before they are sent to the destination. The compressed data is then decompressed on the target.

Virtual Cluster Research and Dynamic Deployment

Several virtual cluster research projects have focused on **dynamic deployment** to improve the flexibility and resource allocation of VMs across clusters:

1. **Cellular Disco at Stanford:**

- A **shared-memory multiprocessor** system designed to handle dynamic VM migrations.

2. **INRIA Virtual Cluster:**
   - Created to test **parallel algorithm performance** across VMs.
3. **COD and VIOLIN Clusters:**
   - Other clusters that are studied for performance and dynamic VM management.

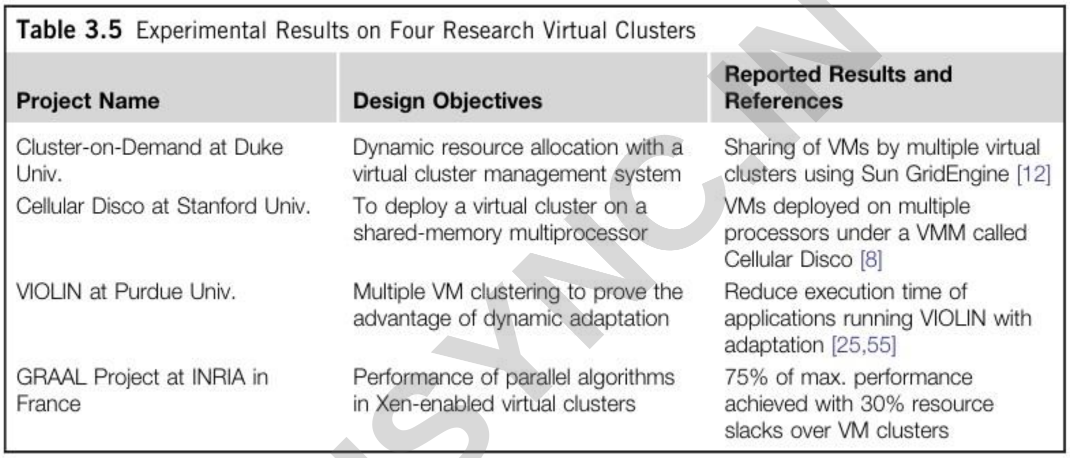


Table 3.5 Experimental Results on Four Research Virtual Clusters

Project Name	Design Objectives	Reported Results and References
Cluster-on-Demand at Duke Univ.	Dynamic resource allocation with a virtual cluster management system	Sharing of VMs by multiple virtual clusters using Sun GridEngine [12]
Cellular Disco at Stanford Univ.	To deploy a virtual cluster on a shared-memory multiprocessor	VMs deployed on multiple processors under a VMM called Cellular Disco [8]
VIOLIN at Purdue Univ.	Multiple VM clustering to prove the advantage of dynamic adaptation	Reduce execution time of applications running VIOLIN with adaptation [25,55]
GRAAL Project at INRIA in France	Performance of parallel algorithms in Xen-enabled virtual clusters	75% of max. performance achieved with 30% resource slacks over VM clusters

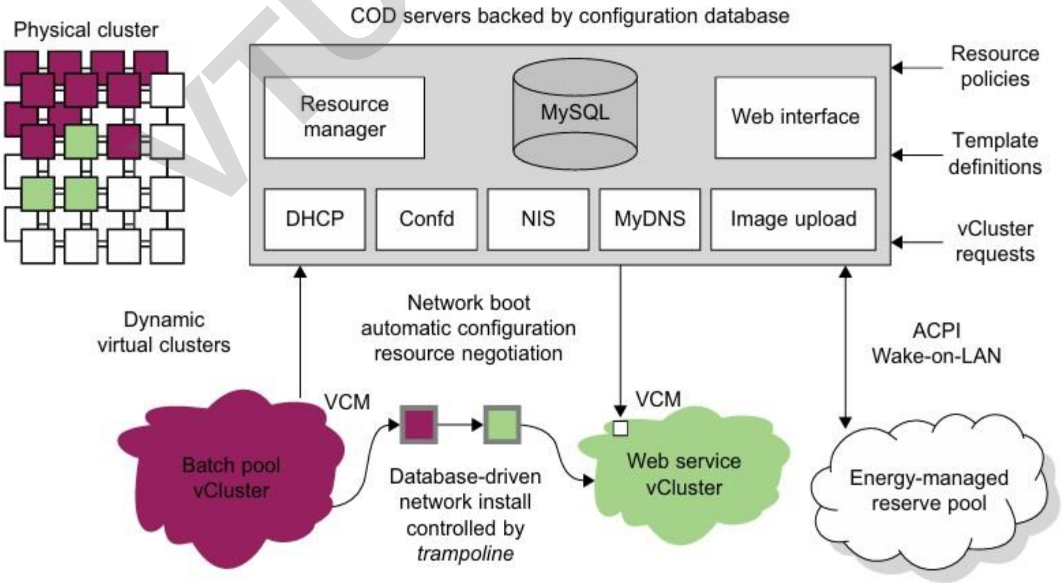


FIGURE 3.23

COD partitioning a physical cluster into multiple virtual clusters.

(Courtesy of Jeff Chase, et al., HPDC-2003 © IEEE [12])

These projects demonstrate the potential benefits of dynamic VM migration within virtual clusters, ensuring resources are efficiently allocated based on demand

Virtualization for Data Center Automation

Growth of Data Centers

Major IT companies like **Google, Amazon, Microsoft, HP, Apple, and IBM** are investing in data centers.

Billions of dollars are spent on **data-center construction and automation**.

Automation enables **dynamic resource allocation** for millions of users with **QoS and cost-efficiency**.

Role of Virtualization in Data-Center Automation

Virtualization and **cloud computing** drive automation. **Market**

growth (2006-2011):

- **2006:** $1.044 billion market share, dominated by **production consolidation and software development**.
- **2011 (projected):** $3.2 billion market share, expanding into **high availability (HA), utility computing, and workload balancing**.

Virtualization reduces **planned downtime** and enhances **mobility and scalability**.

Developments in Virtualization

High availability (HA) ensures system uptime.

Backup services improve disaster recovery.

Workload balancing optimizes resource allocation.

Service-oriented automation and **policy-based management** enhance efficiency.

Server Consolidation in Data Centers

Workloads are categorized into:

- **Chatty workloads:** Fluctuating demand (e.g., video streaming at night).
- **Noninteractive workloads:** Consistent demand (e.g., high-performance computing).

Problem: Underutilized servers waste resources (hardware, space, power, management costs).

Solution: Virtualization-based **server consolidation** optimizes resource utilization.

Benefits of Server Virtualization

Improves hardware utilization by consolidating multiple servers into fewer machines.

Enhances disaster recovery and **backup services**.

Reduces Total Cost of Ownership (TCO):

- Fewer servers needed → lower maintenance, power, and cooling costs.

Improves availability and business continuity:

- Guest OS crashes do not affect host OS or other VMs.
- VMs can be migrated between servers seamlessly.

Automated Resource Management in Virtualized Data Centers **Key**

factors:

- **Resource scheduling** and **power management**.

- **Performance optimization** through analytical models.

Scheduling levels:

- VM-level, server-level, and data-center-level scheduling.

Dynamic CPU allocation:

- Based on **VM utilization and application QoS metrics**.
- **Two-level resource management** using **local (VM-level) and global (server-level) controllers**.

Challenges and Future Scope

- **Optimizing memory management** in **multicore processors (CMPs)**.
- **Power budgeting strategies** to balance power saving and data-center performance.
- **Inter-VM communication and memory access protocols** to enhance efficiency.

Server Consolidation in Data Centers

Types of Workloads in Data Centers

Chatty workloads: Burst at peak times and remain idle otherwise (e.g., web video services).

Noninteractive workloads: Do not require human intervention after submission (e.g., high-performance computing).

Challenge: Workloads have different resource demands, leading to **underutilized servers** when resources are allocated for peak demand.

Problems with Static Resource Allocation

Servers are often **underutilized**, wasting:

- **Hardware resources**
- **Space and power**
- **Management costs**

Resource optimization is needed at the level of **CPU, memory, and network interfaces**.

Virtualization-Based Server Consolidation

Key approach: Reduces the number of physical servers while optimizing resource use.

More effective than other consolidation techniques (e.g., centralized and physical consolidation).

Virtualization allows **fine-grained resource allocation**, improving flexibility.

Benefits of Server Virtualization

- **Enhances hardware utilization:** Underutilized servers are consolidated into fewer machines.
- **Facilitates backup and disaster recovery.**
- **Improves agility in resource provisioning:** VM images can be cloned and deployed quickly.
- **Reduces Total Cost of Ownership (TCO):**
  - Fewer servers → lower maintenance, power, cooling, and cabling costs.
- **Improves availability and business continuity:**
  - Guest OS crashes do not affect host OS or other VMs.
  - Virtual servers can be **migrated seamlessly** between physical machines.

Challenges in Virtualized Data Centers

Resource scheduling complexity: Needs optimization at multiple levels:

- **VM level**
- **Server level**
- **Data-center level**

Dynamic CPU allocation:

- Based on **VM utilization and application-level QoS metrics**.
- Adjusts resources automatically for varying workloads.

Two-level resource management:

- **Local controller** (VM level) and **Global controller** (server level) work together for **autonomic resource allocation**.

Multicore Processors and Virtualization

Challenges with CMP (Chip Multiprocessing):

- Memory systems are not fully optimized for virtualization.
- Need to **reduce memory access time** and **minimize inter-VM interference**.
- **VM-aware power budgeting** required for **balancing power savings and performance**.

Future Considerations for Optimized Virtualized Data Centers

Improving inter-VM communication protocols.

Enhancing memory sharing and management in CMP-based servers.

Integrating power management policies with VM-aware scheduling.

Addressing heterogeneity in workloads for better performance and efficiency.

Virtual Storage Management

Storage Virtualization in System Virtualization

Traditional Storage Virtualization: Aggregation and repartitioning of physical disks for use by physical machines.

In System Virtualization: Virtual storage includes the storage managed by **Virtual Machine Monitors (VMMs)** and **guest OSes**, with the data classified into two categories:

- **VM images** (specific to virtual environments)
- **Application data** (similar to traditional OS environments).

Challenges in Virtual Storage

Encapsulation and Isolation:

- VMs provide isolation between guest OSes, allowing multiple VMs to run on a physical machine.
- **Storage systems** struggle to keep up with system and CPU advancements, becoming the **bottleneck** in VM deployment.

Storage Management Issues:

- Guest OS storage operations behave as though accessing a real hard disk, but they cannot directly access the physical disk.
- Multiple guest OSes may **compete for disk resources** when running on the same machine.
- The storage management layer of the **underlying VMM** is much more complex than traditional guest OS management.

VM Storage Primitives:

- Operations like **remapping volumes across hosts** and **checkpointing disks** are **complicated** and sometimes **unavailable**.

Problems in Virtual Storage Management

Flooded VM images: Large numbers of VMs in data centers create excessive VM images, consuming significant storage space.

Storage Management Complexity: Current storage management techniques don't cater to virtualization needs, making it difficult to manage and optimize storage.

Solutions in Virtual Storage Management

Parallax (Distributed Storage System for Virtualization):

- A **customized storage solution** designed for virtual environments, aimed at simplifying management and improving performance.
- **Content Addressable Storage (CAS):** Reduces the total size of VM images, supporting large VM-based systems in data centers.
- **Federated Storage VMs**: Moves traditional storage features into a **federation of storage VMs** that share physical hosts with the VMs they serve, improving management and performance.

Parallax System Architecture

Storage Appliance VM:

- Acts as a **block virtualization layer** between the VMs and physical storage devices.
- Provides a **virtual disk** for each VM on the same physical machine.
- Supports various system virtualization techniques, such as **paravirtualization** and **full virtualization**.

Benefits of Parallax

- Simplifies storage management in virtualized data centers.
- Reduces the **storage footprint** by reducing the size of VM images.
- **Improves performance** by handling virtual storage operations more efficiently.

Example 3.11 Parallax Providing Virtual Disks to Client VMs from a Large Common Shared Physical Disk

The architecture of Parallax is scalable and especially suitable for use in cluster-based environments. Figure 3.26 shows a high-level view of the structure of a Parallax-based cluster. A cluster-wide administrative domain manages all storage appliance VMs, which makes storage management easy. The storage appliance

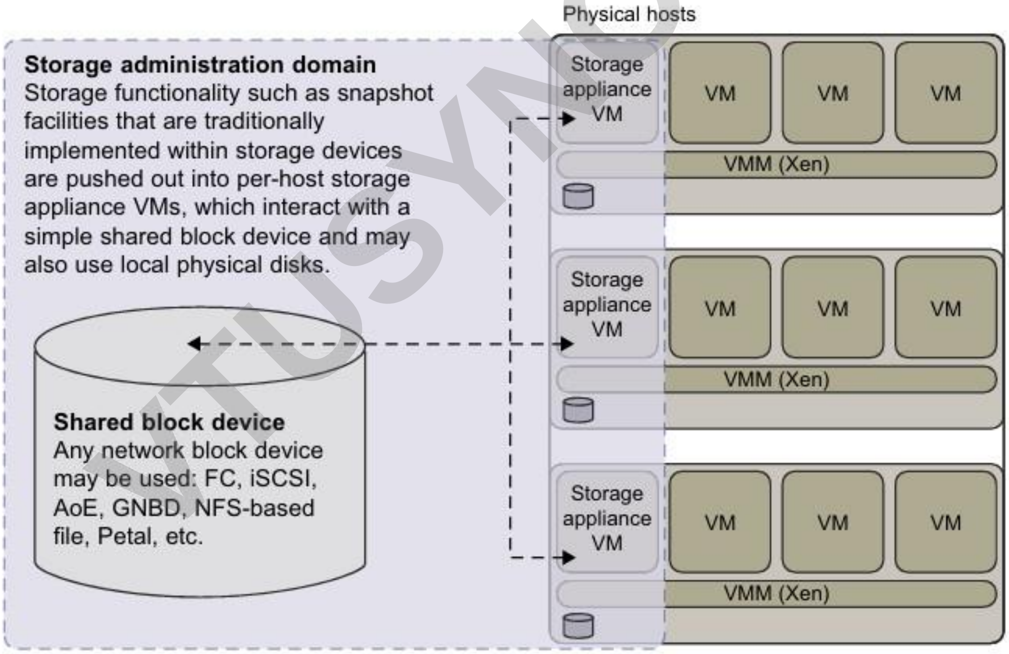
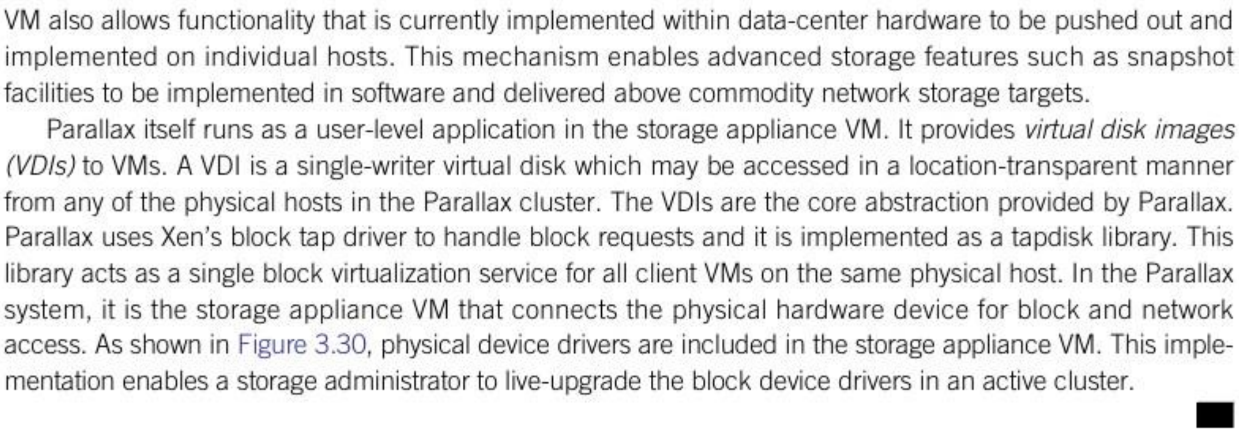

Physical hosts
Storage administration domain
Storage functionality such as snapshot facilities that are traditionally implemented within storage devices are pushed out into per-host storage appliance VMs, which interact with a simple shared block device and may also use local physical disks.
Shared block device
Any network block device may be used: FC, iSCSI, AoE, GNBD, NFS-based file, Petal, etc.
Storage appliance VM
VM
VM
VM
VMM (Xen)
Storage appliance VM
VM
VM
VM
VMM (Xen)
Storage appliance VM
VM
VM
VM
VMM (Xen)

FIGURE 3.26

Parallax is a set of per-host storage appliances that share access to a common block device and presents virtual disks to client VMs.

(Courtesy of D. Meyer, et al. [43])



VM also allows functionality that is currently implemented within data-center hardware to be pushed out and implemented on individual hosts. This mechanism enables advanced storage features such as snapshot facilities to be implemented in software and delivered above commodity network storage targets.

Parallax itself runs as a user-level application in the storage appliance VM. It provides *virtual disk images (VDIs)* to VMs. A VDI is a single-writer virtual disk which may be accessed in a location-transparent manner from any of the physical hosts in the Parallax cluster. The VDIs are the core abstraction provided by Parallax. Parallax uses Xen's block tap driver to handle block requests and it is implemented as a tapdisk library. This library acts as a single block virtualization service for all client VMs on the same physical host. In the Parallax system, it is the storage appliance VM that connects the physical hardware device for block and network access. As shown in Figure 3.30, physical device drivers are included in the storage appliance VM. This implementation enables a storage administrator to live-upgrade the block device drivers in an active cluster. ■

Cloud OS for Virtualized Data Centers

To serve as cloud providers**, data centers must be virtualized.**

Several Virtual Infrastructure (VI) managers **and** Cloud OSes **are designed for managing** virtualized data centers **efficiently.**

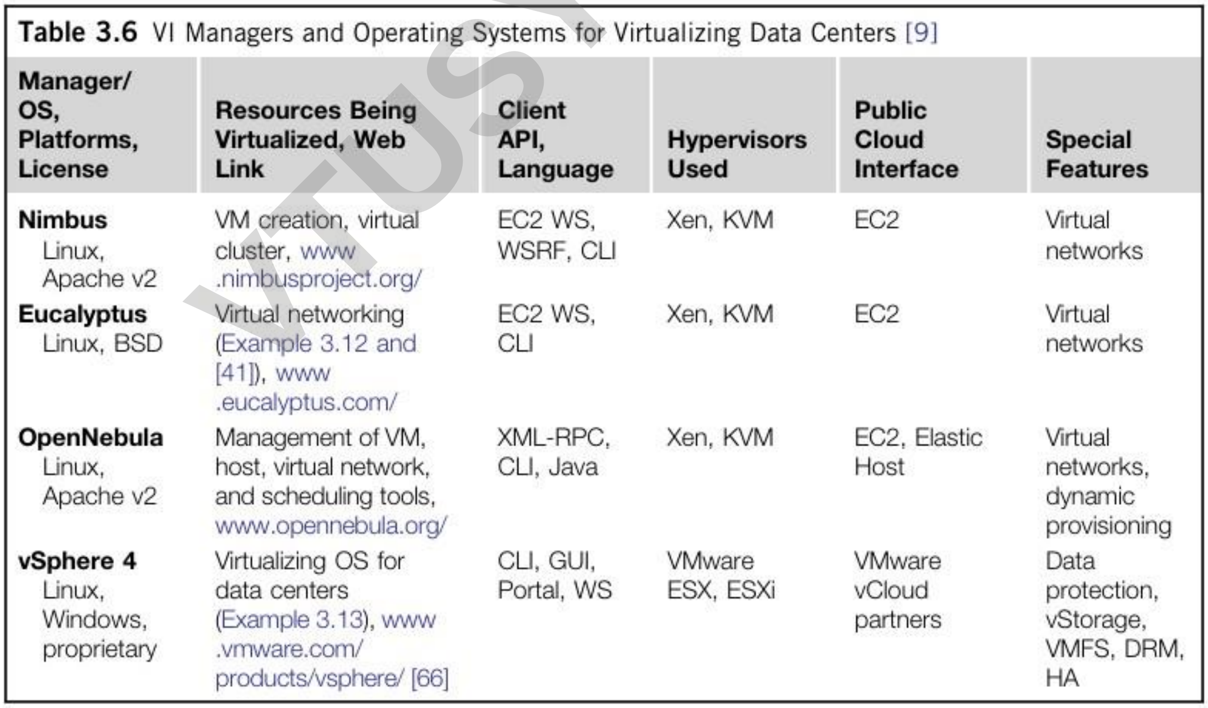


Table 3.6 VI Managers and Operating Systems for Virtualizing Data Centers [9]

Manager/ OS, Platforms, License	Resources Being Virtualized, Web Link	Client API, Language	Hypervisors Used	Public Cloud Interface	Special Features
Nimbus Linux, Apache v2	VM creation, virtual cluster, www.nimbusproject.org/	EC2 WS, WSRF, CLI	Xen, KVM	EC2	Virtual networks
Eucalyptus Linux, BSD	Virtual networking (Example 3.12 and [41]), www.eucalyptus.com/	EC2 WS, CLI	Xen, KVM	EC2	Virtual networks
OpenNebula Linux, Apache v2	Management of VM, host, virtual network, and scheduling tools, www.opennebula.org/	XML-RPC, CLI, Java	Xen, KVM	EC2, Elastic Host	Virtual networks, dynamic provisioning
vSphere 4 Linux, Windows, proprietary	Virtualizing OS for data centers (Example 3.13), www.vmware.com/products/vsphere/ [66]	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vCloud partners	Data protection, vStorage, VMFS, DRM, HA

Example 3.12 Eucalyptus for Virtual Networking of Private Cloud

Eucalyptus is an open source software system (Figure 3.27) intended mainly for supporting Infrastructure as a Service (IaaS) clouds. The system primarily supports virtual networking and the management of VMs; virtual storage is not supported. Its purpose is to build private clouds that can interact with end users through Ethernet or the Internet. The system also supports interaction with other private clouds or public clouds over the Internet. The system is short on security and other desired features for general-purpose grid or cloud applications.

The designers of Eucalyptus [45] implemented each high-level system component as a stand-alone web service. Each web service exposes a well-defined language-agnostic API in the form of a WSDL document containing both operations that the service can perform and input/output data structures.

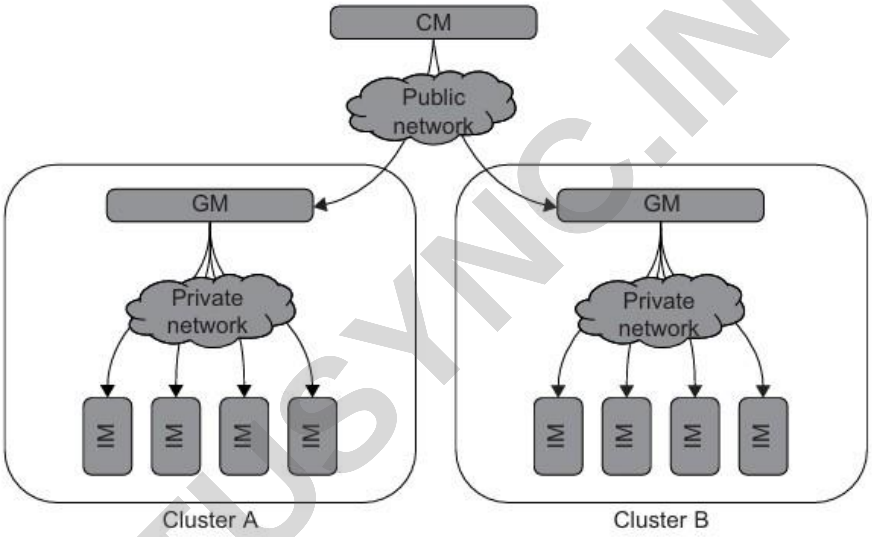


FIGURE 3.27

Eucalyptus for building private clouds by establishing virtual networks over the VMs linking through Ethernet and the Internet.

(*Courtesy of D. Nurmi, et al.* [45])

Example 3.13 VMware vSphere 4 as a Commercial Cloud OS [66]

The vSphere 4 offers a hardware and software ecosystem developed by VMware and released in April 2009. vSphere extends earlier virtualization software products by VMware, namely the VMware Workstation, ESX for server virtualization, and Virtual Infrastructure for server clusters. Figure 3.28 shows vSphere's

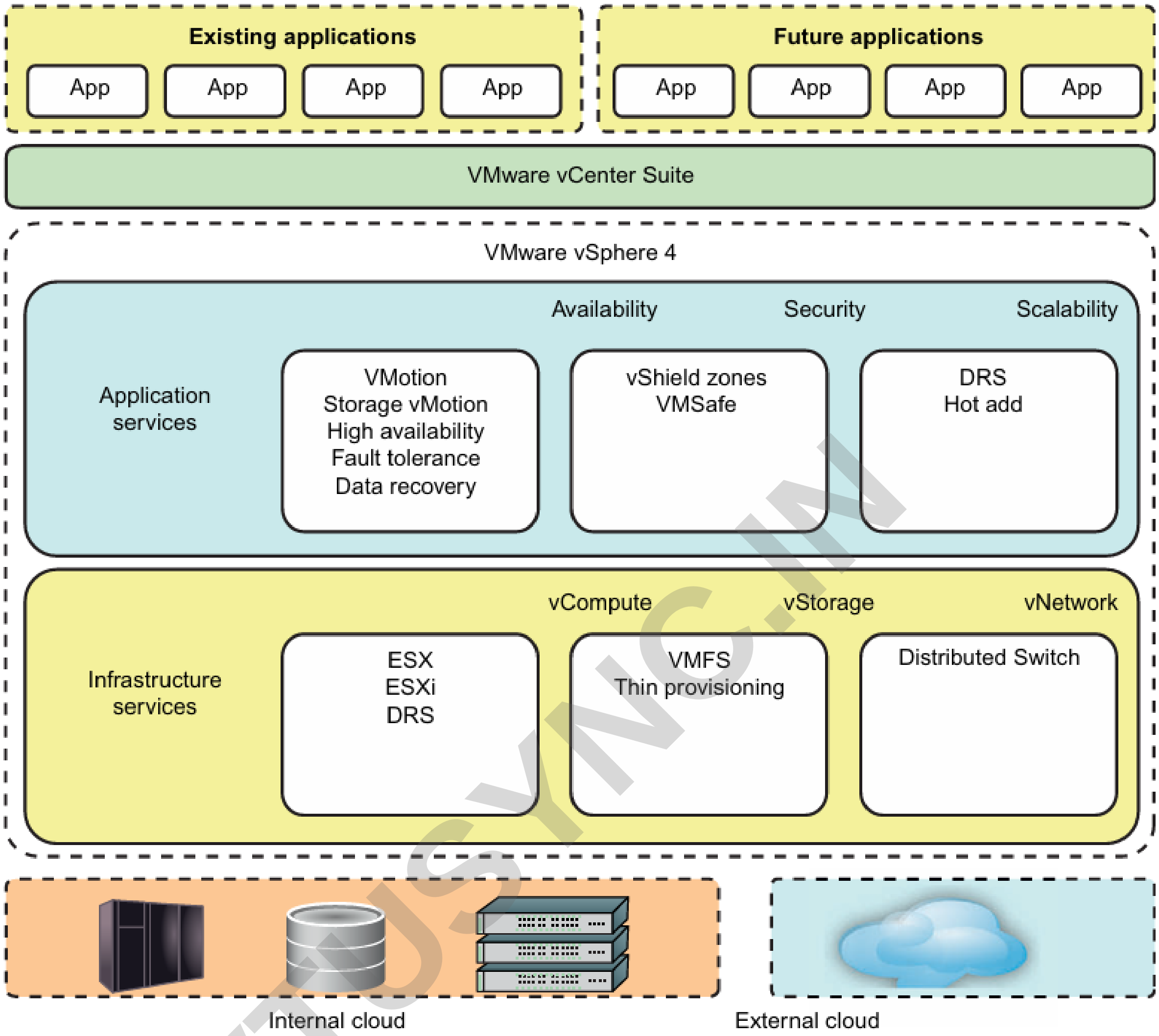


FIGURE 3.28

vSphere/4, a cloud operating system that manages compute, storage, and network resources over virtualized data centers.

(Courtesy of VMware, April 2010 [72])

Functions of VI Managers:

Create and manage VMs

Aggregate VMs into virtual clusters to provide elastic computing resources

Virtual networks support (Nimbus, Eucalyptus)

Dynamic resource provisioning and advance reservations (OpenNebula)

Virtual storage & data protection (vSphere 4)

Trust Management in Virtualized Data Centers

Role of the Virtual Machine Monitor (VMM) in Security:

The **VMM (Hypervisor)** creates and manages VMs, acting as an interface between **OS and hardware**.

A VM is **fully encapsulated**, meaning its entire state can be copied, moved, and deleted like a file.

The **VMM is the foundation of security** in a virtual system, controlling how VMs access hardware resources.

Typically, one **management VM** is privileged to create, suspend, resume, or delete other VMs.

Security Risks in VMM-based Virtualized Environments:

VMM or Management VM Compromise:

- If an attacker gains control over the VMM or management VM, **all VMs and the entire system are at risk**.

Random Number Reuse Issue:

- VMs can be **rolled back** to a previous state, causing old **random numbers** to be reused.
- This weakens **session key security** in cryptographic protocols.
- TCP hijacking attacks can occur due to **reuse of initial sequence numbers**.

VM-Based Intrusion Detection Systems (IDS)

Intrusion: Unauthorized access to a system via local or network-based attacks.

Intrusion Detection System (IDS): Detects and recognizes these unauthorized actions. IDS can be classified into:

- **Host-based IDS (HIDS):** Runs on individual VMs, but can be compromised if the VM is attacked.
- **Network-based IDS (NIDS):** Monitors network traffic but **may miss sophisticated attacks**.

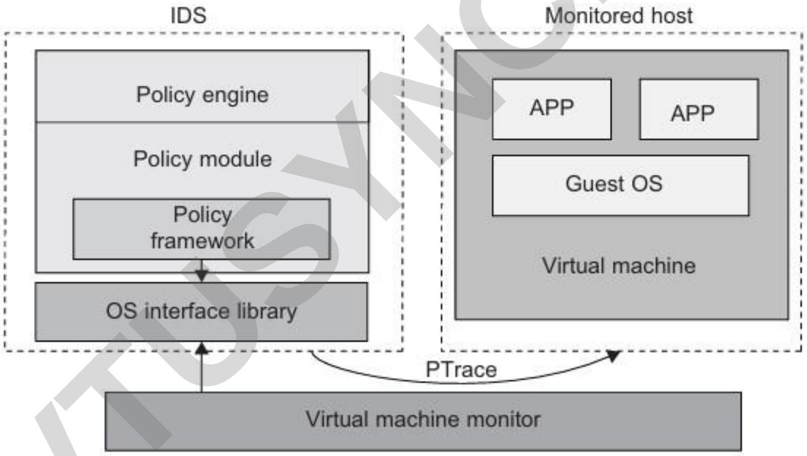
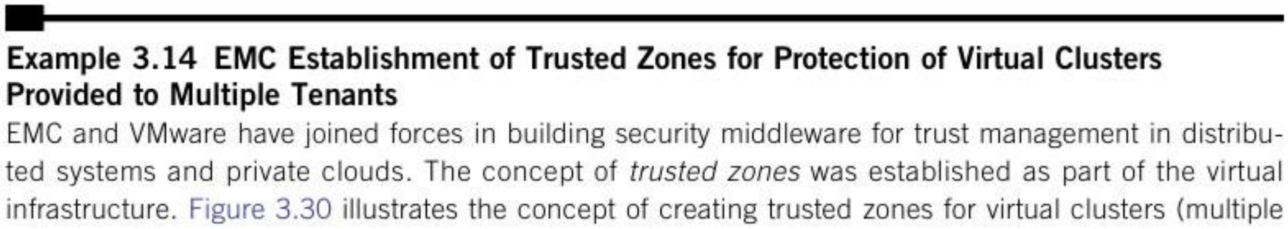


FIGURE 3.29

The architecture of livewire for intrusion detection using a dedicated VM.

(*Courtesy of Garfinkel and Rosenblum, 2002 [17]*)



Example 3.14 EMC Establishment of Trusted Zones for Protection of Virtual Clusters Provided to Multiple Tenants

EMC and VMware have joined forces in building security middleware for trust management in distributed systems and private clouds. The concept of *trusted zones* was established as part of the virtual infrastructure. Figure 3.30 illustrates the concept of creating trusted zones for virtual clusters (multiple

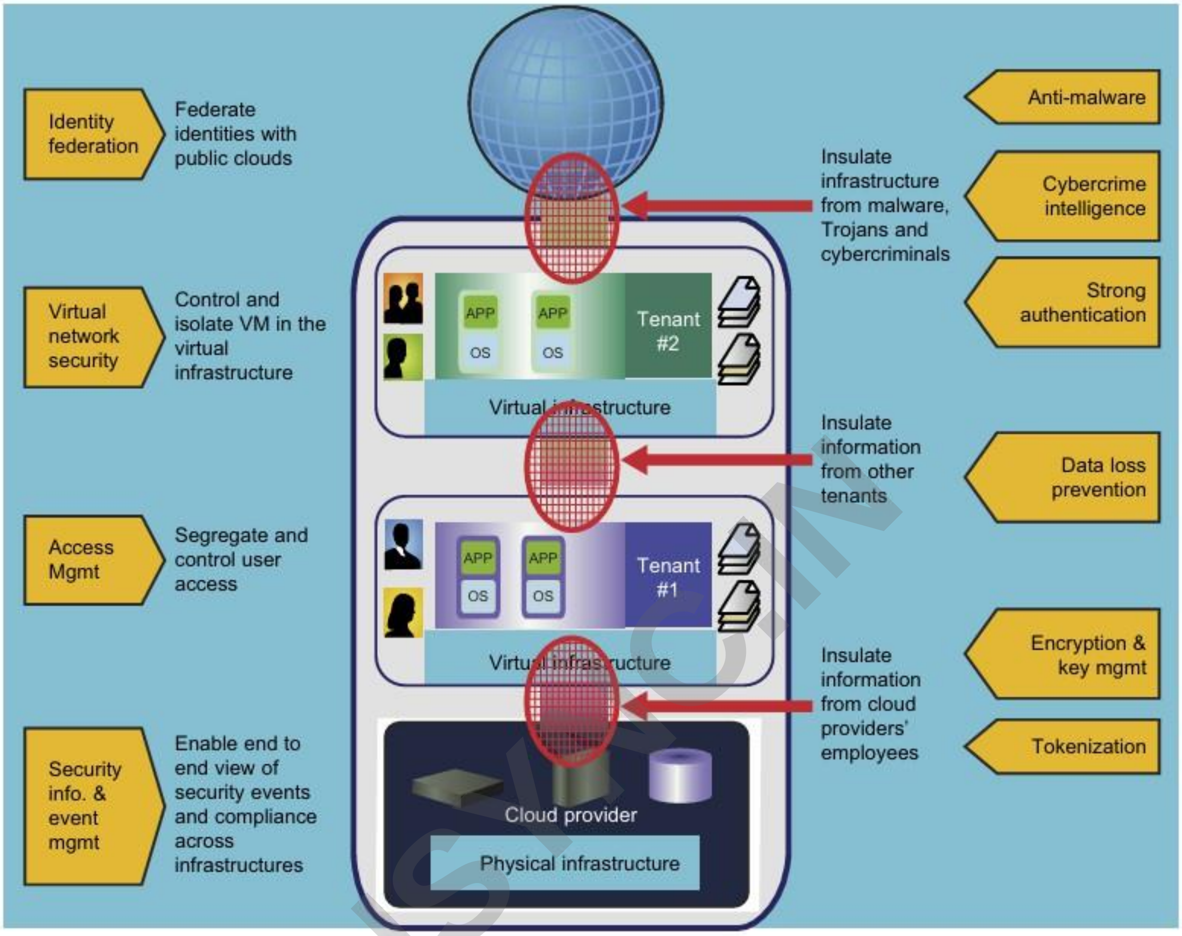

Identity federation
Federate identities with public clouds
Virtual network security
Control and isolate VM in the virtual infrastructure
Access Mgmt
Segregate and control user access
Security info. & event mgmt
Enable end to end view of security events and compliance across infrastructures
APP
OS
Tenant #2
Virtual infrastructure
APP
OS
Tenant #1
Virtual infrastructure
Cloud provider
Physical infrastructure
Insulate infrastructure from malware, Trojans and cybercriminals
Insulate information from other tenants
Insulate information from cloud providers' employees
Anti-malware
Cybercrime intelligence
Strong authentication
Data loss prevention
Encryption & key mgmt
Tokenization

FIGURE 3.30

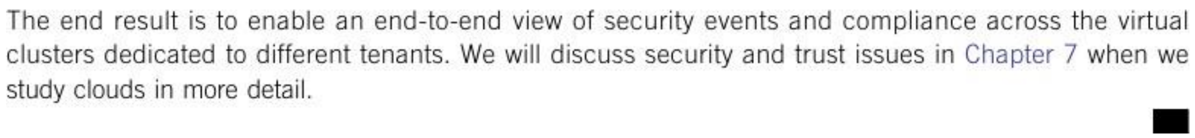
Techniques for establishing trusted zones for virtual cluster insulation and VM isolation.

(*Courtesy of L. Nick, EMC [40]*)

applications and OSes for each tenant) provisioned in separate virtual environments. The physical infrastructure is shown at the bottom, and marked as a cloud provider. The virtual clusters or infrastructures are shown in the upper boxes for two tenants. The public cloud is associated with the global user communities at the top.

The arrowed boxes on the left and the brief description between the arrows and the zoning boxes are security functions and actions taken at the four levels from the users to the providers. The small circles between the four boxes refer to interactions between users and providers and among the users themselves. The arrowed boxes on the right are those functions and actions applied between the tenant environments, the provider, and the global communities.

Almost all available countermeasures, such as anti-virus, worm containment, intrusion detection, encryption and decryption mechanisms, are applied here to insulate the trusted zones and isolate the VMs for private tenants. The main innovation here is to establish the trust zones among the virtual clusters.


The end result is to enable an end-to-end view of security events and compliance across the virtual clusters dedicated to different tenants. We will discuss security and trust issues in Chapter 7 when we study clouds in more detail.

■

Advantages of VM-Based IDS in Virtualization:

Isolation: Even if a VM is compromised, **other VMs remain unaffected**.

VMM Security Monitoring: The **VMM can monitor and audit** access requests for hardware and system software.

Combination of HIDS & NIDS:

- Provides **system-level monitoring (HIDS)**.
- Offers **network-level detection (NIDS)**.

Two Implementation Methods for VM-Based IDS:

Independent IDS Process in Each VM or a High-Privileged Management VM:

- IDS runs separately in every VM.
- A privileged VM manages IDS functions.

IDS Integrated into the VMM (Full Privilege Access):

- IDS is built into the hypervisor, giving **direct hardware access**.
- More effective but also increases the risk if compromised.

Additional Intrusion Prevention Methods:

IDS Logs:

- Analyzing attack patterns is critical.
- Logs are used for security monitoring but must be **protected from tampering**.

Honeypots & Honeynets:

- **Honeypots** simulate vulnerable systems to lure attackers.
- **Honeynets** use multiple honeypots to analyze attack behaviors.
- **Virtual honeypots** use VMs as **decoy targets** to track attackers.