

Module-2

Virtual Machines and Virtualization of Clusters and Data Centers: Implementation Levels of Virtualization, Virtualization Structure/Tools and Mechanisms, Virtualization of CPU/Memory and I/O devices, Virtual Clusters and Resource Management, Virtualization for Data Center Automation.

Textbook 1: Chapter 3: 3.1 to 3.5

3.1 Implementation Levels of Virtualization

Virtualization: Virtualization is the process of creating virtual instances of computing resources—such as servers, storage, networks, or operating systems—on a single physical machine. It allows multiple virtual environments to run independently on the same hardware, improving resource utilization, scalability, and flexibility.

Difference Between Traditional Computers and Virtualized Computers

Feature	Traditional Computer 	Virtualized Computer 
Hardware Usage	Dedicated to a single OS and applications.	Multiple virtual instances share the same physical hardware.
Operating System	Runs one OS per physical machine.	Can run multiple OSES (Windows, Linux, etc.) on the same machine.
Resource Utilization	Often underutilized, as resources are fixed.	Maximizes hardware efficiency by sharing CPU, RAM, and storage among VMs.
Isolation	All applications share the same OS.	Each virtual instance (VM or container) is isolated from others.
Scalability	Adding resources requires new physical machines.	Can dynamically allocate more resources without new hardware.
Security	If one application crashes, it can affect the whole system.	Virtual instances are independent, reducing the impact of failures.
Deployment	Manual setup and configuration per machine.	Can quickly deploy and clone VMs/containers.
Maintenance	Requires individual system updates and backups.	Easier to maintain, as snapshots and templates allow fast recovery.
Cost	Higher hardware costs, as each OS requires a separate machine.	Lower costs by running multiple virtual instances on fewer physical machines.
Flexibility	Limited flexibility; one system per hardware.	Can run different OSES and applications simultaneously.

Example Scenarios

◆ **Traditional Computer:** A single physical server running Windows, dedicated to one task, such as hosting a database.

◆ **Virtualized Computer:** A physical server running multiple VMs—one with Windows for the database, another with Linux for a web server, and another for testing.

Benefits of Virtualization

1. **Better Resource Utilization** – Maximizes hardware efficiency by running multiple VMs on a single machine.
2. **Cost Savings** – Reduces hardware, power, and maintenance costs.
3. **Scalability & Flexibility** – Easily scale resources as needed.
4. **Improved Security & Isolation** – VMs are isolated from each other, reducing security risks.
5. **Disaster Recovery** – Snapshots and backups allow quick recovery in case of failures.

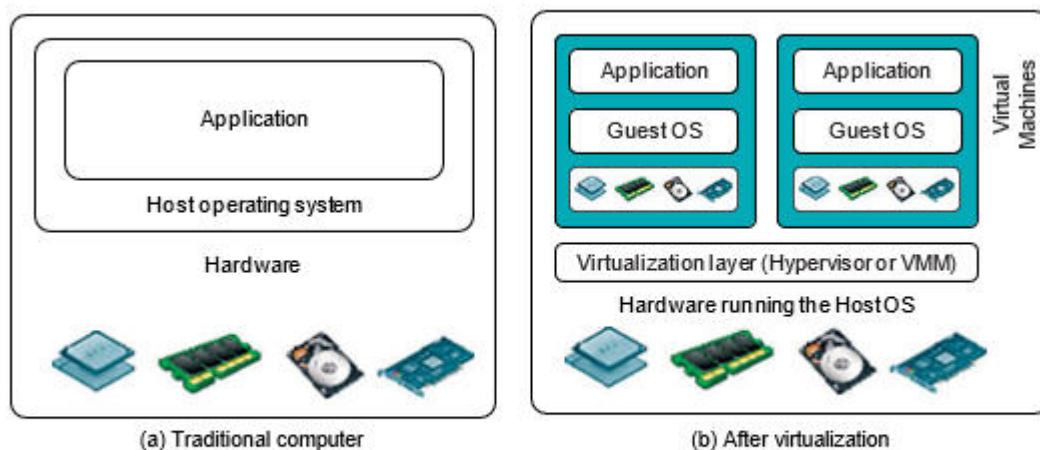


FIGURE 3.1

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

Virtualization Layer

The **virtualization layer** is a software layer that abstracts physical hardware resources (CPU, memory, storage, network, etc.) and presents them as virtual resources to applications and operating systems. It acts as a bridge between the physical hardware and virtual instances, ensuring proper allocation, isolation, and management of resources.

Role of the Virtualization Layer

- **Abstraction:** Hides the complexity of hardware, allowing virtual instances to operate independently.
- **Isolation:** Ensures that virtual instances (VMs or containers) don't interfere with one another.
- **Resource Allocation:** Dynamically allocates resources based on demand, ensuring efficient usage.

- **Compatibility:** Enables different operating systems and applications to coexist on the same physical machine.
- **Portability:** Allows virtual machines or containers to be easily moved across physical hosts.

Benefits of the Virtualization Layer

- **Efficient Use of Resources:** Consolidates multiple workloads onto fewer physical machines.
- **Scalability:** Enables dynamic allocation of resources to meet changing demands
- **Improved Security:** Isolates virtual instances to minimize security risks.
- **Flexibility:** Supports diverse operating systems and applications on the same hardware.
- **Disaster Recovery:** Facilitates backups, snapshots, and migrations for rapid recovery.

What is a Virtual Machine Monitor (VMM)?

- A **Virtual Machine Monitor (VMM)**, also called a **hypervisor**, is a software layer or hardware component responsible for managing and running virtual machines (VMs) on a host system. It provides the necessary abstraction and resource allocation to allow multiple virtual machines to share the same physical hardware.

Difference between Virtualization layer and VMM

Aspects	VMM	Virtualization Layer
Definition .	A component (software or hardware) that creates, manages, and runs virtual machines by abstracting hardware.	The broader abstraction layer that virtualizes resources (e.g., hardware, OS, applications, or networks).
Role	Directly responsible for managing VMs and their access to hardware resources.	Encompasses the VMM along with other components like APIs, drivers, and management tools.
Scope	Focused specifically on running and managing virtual machines.	Includes hardware, VMM, virtual networking, storage, and resource abstraction tools.
Examples	VMware ESXi, Microsoft Hyper-V, KVM.	Includes hypervisors (like VMM), SDN (software-defined networking), and SDS (software-defined storage).
Primary Function	- Creates and runs virtual machines. - Provides resource isolation and sharing for VMs.	- Abstracts all physical resources. - Enables dynamic management of hardware, networks, and applications.
Abstraction Level	Focuses on hardware abstraction for virtual machines.	Operates at multiple levels, including hardware, operating systems, storage, and networks.
Key Components	- Virtual CPUs. - Virtual memory management. - Device emulation. - Virtual storage.	- VMM. - Management tools (e.g., VMware vCenter). - Virtual switches (e.g., Open vSwitch). - APIs and drivers.

Example Scenarios	- Running multiple virtual machines on a single physical server. - Isolating workloads using hypervisors like Xen or KVM.	- Managing data center resources using platforms like VMware vSphere. - Supporting cloud platforms (e.g., AWS or Azure).
--------------------------	---	--

3.1.1 Levels of Virtualization Implementation

Virtualization is done in *five abstract levels* ranging from hardware to applications

3.1.1.1 Instruction Set Architecture Level:

At the **ISA level**, virtualization primarily focuses on enabling the execution of instructions written for one architecture on a host machine with a different architecture. The process is achieved through **instruction set emulation**, which can either interpret instructions one-by-one or dynamically translate blocks of instructions to improve efficiency.

Note: Emulation refers to the process of mimicking or imitating the behavior of one system, device, or software by another system. In the context of computing, emulation allows one machine or software to replicate the functions of another, enabling programs, applications, or operating systems designed for one platform to run on a different platform.

How ISA Virtualization Works

1. Instruction Emulation:

- The source ISA (e.g., MIPS) is emulated on the target ISA (e.g., x86) through a software layer.
 - The software layer interprets or translates the source instructions into target machine instructions.
2. **Virtual ISA (V-ISA):**
- A **virtual instruction set architecture** acts as an abstraction, making it possible for various source ISAs to execute on the same host machine by translating and optimizing the instructions.
 - A software layer, added to the compiler, facilitates this translation and manages differences between ISAs.

Key Techniques in ISA Virtualization

1. **Code Interpretation:**
 - **Process:** An interpreter program translates source instructions into host (native) instructions one-by-one during execution.
 - **Characteristics:**
 - Simple to implement.
 - High overhead due to the need to process each instruction individually.
 - **Performance:** Slow, as each source instruction may require tens or even hundreds of native instructions to execute.
2. **Dynamic Binary Translation:**
 - **Process:**
 - Instead of interpreting instructions one-by-one, this method translates **blocks of source instructions** (basic blocks, traces, or superblocks) into target instructions.
 - The translated blocks are cached, so subsequent executions do not need re-translation.
 - **Characteristics:**
 - Faster than interpretation due to caching and reuse of translated instructions.
 - Optimization opportunities arise from analyzing multiple instructions in a block.
 - **Performance:** Significantly better than interpretation but requires more complex implementation.
3. **Binary Translation and Optimization:**
 - **Purpose:** Enhance performance and reduce the overhead of translation.
 - **Methods:**
 - **Static Binary Translation:** Translates the entire binary code before execution, which avoids runtime translation but can miss opportunities for runtime optimizations.
 - **Dynamic Binary Translation:** Translates instructions at runtime, enabling better adaptability to runtime conditions.
 - **Dynamic Optimizations:** Includes reordering, inlining, and loop unrolling to improve the efficiency of translated code.

ISA-level virtualization via instruction set emulation opens immense possibilities for running diverse workloads across platforms, supporting legacy systems, and enabling hardware

independence. The shift from simple interpretation to more advanced techniques like dynamic binary translation and optimizations has significantly improved its performance and applicability, making it a key enabler for cross-platform software execution.

Advantages of ISA Virtualization

1. **Legacy Code Support:**
 - Allows legacy binary code (e.g., for MIPS or PowerPC) to run on newer hardware (e.g., x86 or ARM).
 - Extends the lifespan of legacy software without needing hardware redesign.
2. **Cross-Architecture Compatibility:**
 - Applications can run on hardware with different ISAs, enhancing portability and flexibility.
3. **Facilitates Hardware Upgrades:**
 - Software compiled for older processors can run on modern processors, easing transitions to new architectures.
4. **Enables Platform Independence:**
 - Virtual ISAs abstract the underlying hardware, enabling software to operate across heterogeneous platforms.

Challenges and Limitations

1. **Performance Overhead:**
 - Emulating an ISA on another is inherently slower due to instruction-by-instruction interpretation or translation.
 - Dynamic binary translation improves performance but still adds runtime overhead.
2. **Complexity:**
 - Implementing dynamic binary translation and optimizations requires advanced techniques and significant development effort.
3. **Scalability:**
 - Supporting highly diverse ISAs can become challenging, especially when optimizing performance for multiple architectures.

3.1.1.2 Hardware abstraction level:

Hardware-level virtualization is a method of virtualization implemented directly on the physical hardware to enable the creation and management of virtual machines (VMs). It abstracts and virtualizes a computer's physical resources—such as CPU, memory, and input/output (I/O) devices—allowing multiple operating systems to run concurrently on the same physical machine.

Key Features of Hardware-Level Virtualization

1. **Bare-Metal Hypervisors:**
 - A **hypervisor** (Type 1) operates directly on the hardware without requiring an underlying host operating system.
 - It creates and manages virtual hardware environments for virtual machines.
2. **Resource Virtualization:**

- Virtualizes hardware components such as CPUs, memory, network interfaces, and storage.
- VMs appear to have dedicated hardware, even though they share the underlying physical resources.
- 3. **Improved Hardware Utilization:**
 - Allows multiple users or workloads to share the same hardware, increasing resource utilization and efficiency.
- 4. **Isolation:**
 - Each VM operates in isolation, meaning that the failure or compromise of one VM does not affect others.

Advantages of Hardware-Level Virtualization

1. **High Performance:**
 - Since the hypervisor runs directly on hardware, it minimizes overhead and provides near-native performance for VMs.
2. **Scalability:**
 - Easily supports multiple VMs, enabling efficient use of physical server resources.
3. **Fault Isolation:**
 - Problems in one VM (e.g., OS crashes or software bugs) do not impact other VMs or the host system.
4. **Versatility:**
 - Supports running different operating systems or environments on the same physical hardware.

3.1.1.3 Operating System Level

Operating System (OS) level virtualization is a type of virtualization that operates at the OS kernel layer, creating isolated environments called **containers** or **virtual environments** within a single instance of an operating system. This approach allows multiple isolated user spaces to run on the same physical hardware while sharing the same operating system kernel.

Key Features of OS-Level Virtualization

1. **Single OS Kernel:**
 - All containers share the same underlying OS kernel, eliminating the need for separate kernels for each environment.
 - More lightweight compared to traditional hardware-level virtualization since there's no need to emulate hardware.
2. **Isolated Environments (Containers):**
 - Containers behave like independent servers, with their own libraries, binaries, and configuration files.
 - Processes running inside one container are isolated from processes in other containers.
3. **Efficient Resource Utilization:**
 - OS-level virtualization efficiently shares hardware resources like CPU, memory, and storage across containers.
 - Reduces overhead compared to full virtualization, as there is no need for a hypervisor or virtual hardware.

Advantages of OS-Level Virtualization

1. **Lightweight:** Containers consume fewer resources compared to traditional VMs because they do not require a full guest operating system.
2. **High Performance:** Since all containers share the same OS kernel, there is minimal overhead, resulting in near-native performance.
3. **Scalability:** Containers can be created, started, stopped, and destroyed quickly, making them ideal for dynamic environments.
4. **Isolation:** Although containers share the same kernel, they provide process and file system isolation, ensuring that one container does not interfere with another.
5. **Ease of Deployment:** Containers package applications with their dependencies, making them portable across different environments.

Disadvantages of OS-Level Virtualization

1. **Single OS Limitation:** Since all containers share the same kernel, they must use the same operating system. For example, you cannot run a Windows container on a Linux host.
2. **Weaker Isolation:** Compared to hardware-level virtualization, OS-level virtualization provides less isolation. If the kernel is compromised, all containers are at risk.
3. **Compatibility Issues:** Applications that require specific kernel modules or features not supported by the shared kernel may face compatibility challenges.

3.1.1.4 Library Support Level:

Library-level virtualization focuses on virtualizing the interface between an application and the operating system by intercepting or emulating the API (Application Programming Interface) calls. This type of virtualization is lightweight and primarily targets specific application needs, rather than virtualizing entire operating systems or hardware.

Key Features of Library-Level Virtualization

1. **API Hooks:**
 - Applications typically interact with the operating system via APIs exported by user-level libraries.
 - Library-level virtualization works by **intercepting API calls** and redirecting them to virtualized implementations.
2. **Controlled Communication:**
 - Virtualization happens by managing the communication link between the application and the underlying system.
 - This approach avoids direct interaction with the operating system and replaces it with controlled, virtualized responses.
3. **Application-Specific Virtualization:**
 - Focused on enabling specific features or compatibility, such as supporting applications from one environment on another.

How Library-Level Virtualization Works

- Applications are written to use standard library calls for their functionality, such as file access, networking, or graphics.

- Library-level virtualization intercepts these calls (using API hooks) and replaces the original functionality with emulated or redirected behavior.

Library-level virtualization is a targeted and lightweight form of virtualization that focuses on emulating user-level APIs to enable application compatibility and portability. It plays a critical role in scenarios like running software across platforms, leveraging hardware features in virtualized environments, and extending the life of legacy applications. While it may not provide the full isolation or flexibility of OS- or hardware-level virtualization, its efficiency and simplicity make it invaluable for specific use cases.

3.1.1.5 User-Application Level:

User-Application-level virtualization refers to the virtualization of individual applications, treating each application as a virtualized entity, often running as an isolated process. It is also known as **process-level virtualization** since the virtualization focuses on isolating and managing application processes rather than entire systems or hardware.

Key Features of User-Application-Level Virtualization

1. **High-Level Language (HLL) Virtual Machines:**
 - A common form of application-level virtualization involves **High-Level Language (HLL) Virtual Machines**, which provide an abstraction layer for running applications written in a specific language.
 - The virtualization layer operates as an application program on top of the OS, exporting a virtual machine abstraction.
 - Examples:
 - **Java Virtual Machine (JVM):** Executes Java bytecode, enabling Java applications to run on any platform with a JVM implementation.
 - **Microsoft .NET Common Language Runtime (CLR):** Runs programs written in .NET-supported languages like C# and VB.NET.
2. **Application Isolation and Sandboxing:**
 - Another form of user-application level virtualization involves isolating applications from the host OS and other applications.
 - This approach wraps the application in a layer that prevents it from interfering with system resources or other applications.
3. **Application Streaming:**
 - Applications are deployed as self-contained packages that can be executed in a virtualized environment without traditional installation processes.
 - The application runs in an isolated sandbox, reducing dependency on the host OS.

Approaches to User-Application Level Virtualization

1. High-Level Language (HLL) Virtualization:
2. Application Isolation and Sandboxing:

3. Application Streaming

Advantages of User-Application Level Virtualization

1. **Cross-Platform Compatibility:**
 - Applications written for an abstract VM (e.g., JVM, CLR) can run on any system with the corresponding VM implementation.
2. **Improved Security:**
 - Applications are isolated from the host OS and other applications, reducing the risk of system compromise or interference.
3. **Simplified Deployment:**
 - Applications can be distributed as self-contained packages, eliminating the need for complex installation procedures or OS-level dependencies.
4. **Resource Efficiency:**
 - Compared to hardware- or OS-level virtualization, application-level virtualization has lower overhead as it focuses only on individual processes.
5. **Portability:**
 - Virtualized applications can be easily moved between systems or platforms.

Disadvantages of User-Application Level Virtualization

1. **Performance Overhead:**
 - Running applications in a virtualized environment may introduce some latency compared to native execution.
2. **Limited Scope:**
 - Unlike OS- or hardware-level virtualization, application-level virtualization cannot provide a full OS environment or support multiple users.
3. **Compatibility Challenges:**
 - Not all applications can be easily virtualized, especially those with tight integration with the underlying OS or hardware.

3.1.1.6 Relative Merits of Different Approaches:

Table 3.1 Relative Merits of Virtualization at Various Levels (More "X"s Means Higher Merit, with a Maximum of 5 X's)				
Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

In the above table, the column headings correspond to four technical merits. “*Higher Performance*” and “*Application Flexibility*” are self-explanatory. “*Implementation*

Complexity” implies the cost to implement that particular virtualization level. **Application Isolation**” refers to the effort required to isolate resources committed to different VMs.

The number of X’s in the table cells reflects the advantage points of each implementation level. Five X’s implies the best case and one X implies the worst case. Overall, hardware and OS support will yield the highest performance. However, the hardware and application levels are also the most expensive to implement. User isolation is the most difficult to achieve. ISA implementation offers the best application flexibility.

3.1.2 VMM Design Requirements and providers

Hardware-level virtualization adds a layer, the **Virtual Machine Monitor (VMM)**, between the hardware and operating systems. The VMM manages hardware resources and allows multiple operating systems to run simultaneously on a single hardware setup by virtualizing components like the CPU. A VMM must meet three key requirements:

1. Provide an environment essentially identical to the original machine.
2. Minimize performance overhead.
3. Maintain complete control over system resources.

While VMMs ensure functional equivalence to physical hardware, two performance-related exceptions are allowed: resource availability limitations (when multiple VMs share resources) and timing dependencies (due to software layers and concurrent VMs).

Efficiency is crucial for VMMs, as slow emulators or interpreters are unsuitable for real machines. To ensure performance, most virtual processor instructions should execute directly on physical hardware without VMM intervention.

The VMM manages resources by allocating them to programs, restricting unauthorized access, and regaining control when needed. However, implementing VMMs can be challenging on certain processor architectures (e.g., x86), where privileged instructions cannot always be trapped. Processors not designed for virtualization may require hardware modifications to meet VMM requirements, a method known as **hardware-assisted virtualization**.

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Key Observations:

- VMware Workstation supports a wide range of guest operating systems and uses full virtualization.
- VMware ESX Server eliminates a host OS, running directly on hardware with para-virtualization.
- Xen supports diverse host OSs and uses a hypervisor-based architecture.
- KVM runs exclusively on Linux hosts and supports para-virtualization for multiple architectures.

3.1.3 Virtualization Support at the OS level:

Cloud computing, enabled by VM technology, shifts the cost and responsibility of managing computational centers to third parties, resembling the role of banks. While transformative, it faces two significant challenges:

1. **Dynamic Resource Scaling:** Adapting to varying computational demands, where tasks may require a single CPU at times and hundreds of CPUs at others.
2. **Slow VM Instantiation:** Current methods for creating VMs, such as fresh boots or replicating templates, are slow and do not account for the current application state.

To address these challenges and enhance cloud computing efficiency, significant research and development are needed.

3.1.3.1 Why OS-Level Virtualization?

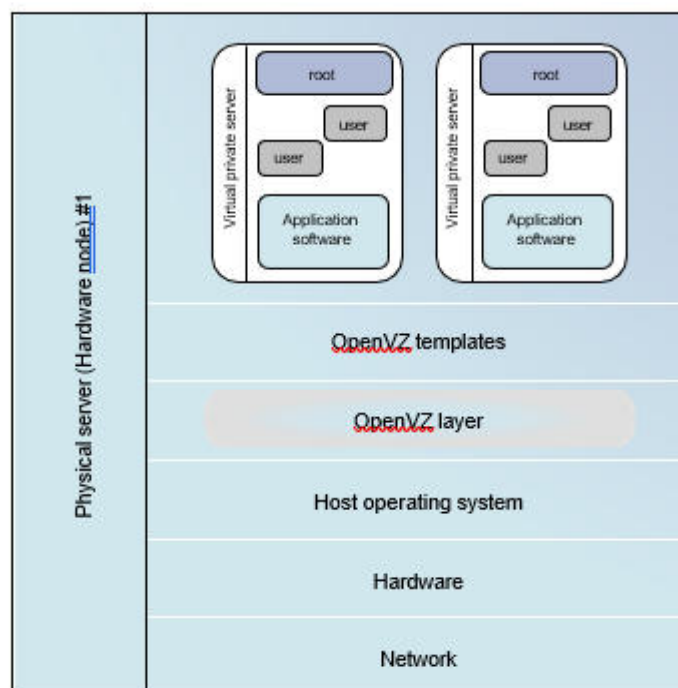


FIGURE 3.3

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

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

OS-level virtualization addresses the challenges of hardware-level virtualization, such as slow initialization, storage issues due to repeated content in VM images, slow performance,

and the need for para-virtualization or hardware modifications. It introduces a virtualization layer within the operating system to partition physical resources and enable multiple isolated virtual environments (VEs), also called containers or Virtual Private Systems (VPS).

VEs share the same OS kernel but appear as independent servers to users, each with its own processes, file system, user accounts, network settings, and more. This approach, known as single-OS image virtualization, is an efficient alternative to hardware-level virtualization. Figure 3.3 illustrates operating systems virtualization from the point of view of a machine stack.

3.1.3.2 Advantages of OS Extensions

OS-level virtualization offers two key advantages over hardware-level virtualization:

1. **Efficiency and Scalability:** OS-level VMs have low startup/shutdown costs, minimal resource requirements, and high scalability.
2. **State Synchronization:** VMs can synchronize state changes with the host environment when needed.

These benefits are enabled by two mechanisms:

- All VMs share a single OS kernel, reducing overhead.
- The virtualization layer allows VMs to access host resources without modifying them.

In cloud computing, these features address the slow initialization of hardware-level VMs and their inability to account for the current application state.

3.1.3.3 Disadvantages of OS Extensions

The primary disadvantage of OS-level virtualization is that all VMs on a single container must belong to the same operating system family. For example, a Linux-based container cannot run a Windows OS. This limitation challenges its usability in cloud computing, where users may prefer different operating systems.

To implement OS-level virtualization:

- A virtualization layer is added to the OS to partition hardware resources for isolated virtual environments (VMs).
- VMs share the same OS kernel, and access requests are redirected to their respective resource partitions.
- Virtual root directories can be created using methods like the **chroot** command in UNIX systems.

Two methods for managing resource partitions are:

1. **Duplicating resources for each VM:** This incurs high resource costs and overhead.
2. **Sharing most resources with the host and creating private copies on demand:** This is more efficient and commonly used.

Due to its limitations and overhead in some scenarios, OS-level virtualization is often considered a secondary choice compared to hardware-assisted virtualization.

3.1.3.4 Virtualization on Linux or Windows Platforms

OS-level virtualization systems are predominantly Linux-based, while Windows-based platforms are still in the research phase. The Linux kernel provides an abstraction layer that allows software to interact with resources without needing hardware-specific details. However, new hardware may require updates or patches to the Linux kernel for extended functionality.

Key points about Linux OS-level virtualization:

- Most Linux platforms are not tied to a specific kernel, enabling a host to run multiple VMs simultaneously on the same hardware.
- Linux-based tools, such as **Linux vServer** and **OpenVZ**, support running applications from other platforms through virtualization.
- On Windows, **FVM** is a specific tool developed for OS-level virtualization on the Windows NT platform.

Example 3.1: Virtualization Support for the Linux Platform

OpenVZ is an open-source, container-based virtualization tool for Linux platforms. It enables the creation of virtual private servers (VPSes) that operate like independent Linux servers. OpenVZ modifies the Linux kernel to provide features like virtualization, resource management, isolation, and checkpointing.

Key Features:

1. **Isolation:**
 - Each VPS has its own files, user accounts, process tree, virtual network, virtual devices, and interprocess communication (IPC) mechanisms.
2. **Resource Management:**
 - **Disk Allocation:** Two levels:
 - *First level:* The OpenVZ server administrator assigns disk space limits to each VM.
 - *Second level:* VM administrators manage disk quotas for users and groups.
 - **CPU Scheduling:**
 - *First level:* OpenVZ's scheduler allocates time slices based on virtual CPU priority and limits.
 - *Second level:* Uses the standard Linux CPU scheduler.
 - **Resource Control:** OpenVZ has ~20 parameters to control VM resource usage.
3. **Checkpointing and Live Migration:**
 - Allows saving the complete state of a VM to a disk file, transferring it to another machine, and restoring it there.
 - The process takes only a few seconds, although network connection re-establishment causes minor delays.

Advantages:

- Efficient resource management and isolation.
- Quick VM migration with minimal downtime.

Challenges:

- Delay in processing due to migrating active network connections.

Table 3.3 provides a summary of OS-level virtualization tools

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 <i>virtual private servers (VPSes)</i> ; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other <i>VPSes</i> , 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 <i>NT</i> kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

3.1.4 Middleware Support for Virtualization

Library-level virtualization, also referred to as user-level Application Binary Interface (ABI) or API emulation, enables execution environments to run specific programs on a platform without the need to virtualize the entire operating system. The key functions include **API call interception** and **remapping**.

Overview of Library-Level Virtualization Systems/ Middleware and Library Support for Virtualization:

1. **WABI:**
 - Middleware that translates **Windows system calls** into **Solaris system calls**, allowing Windows applications to run on Solaris systems.
2. **Lxrun:**
 - A **system call emulator** enabling Linux applications designed for x86 hosts to run on UNIX systems.
3. **WINE:**
 - Provides **library support** to virtualize x86 processors, enabling Windows applications to run on UNIX-based systems.
4. **Visual MainWin:**
 - A **compiler support system** that allows developers to use **Visual Studio** to create Windows applications capable of running on some UNIX hosts.

5. vCUDA:

- A virtualization solution for **CUDA**, enabling applications requiring GPU acceleration to utilize GPU resources remotely. (Discussed in detail in Example 3.2.)

Key Benefits:

- Enables cross-platform application execution without the overhead of a full virtualized operating system.
- Useful for running **alien programs** efficiently on different platforms.

Challenges:

- Performance depends on the accuracy and efficiency of **API call remapping**.
- Limited to specific program compatibility based on system call emulation.

Example 3.2 vCUDA for Virtualization of General-Purpose GPUs

vCUDA is a virtualization solution designed to enable CUDA applications to run on guest operating systems (OSes) by virtualizing the CUDA library. It allows compute-intensive applications to leverage GPU acceleration in a virtualized environment.

Key Features of vCUDA:

1. Purpose:

- Virtualizes the CUDA library for guest OSes, enabling CUDA applications to execute GPU-based tasks indirectly through the host OS.

2. Architecture:

- Follows a **client-server model** with three main components:
 - **vCUDA Library:**
 - Resides in the guest OS as a substitute for the standard CUDA library.
 - Intercepts and redirects API calls to the host OS.
 - Manages virtual GPUs (vGPUs).
 - **Virtual GPU (vGPU):**
 - Abstracts GPU hardware, provides a uniform interface, and manages device memory allocation.
 - Tracks and stores CUDA API flow.
 - **vCUDA Stub:**
 - Resides in the host OS.
 - Receives and interprets requests from the guest OS.
 - Creates execution contexts for CUDA API calls and manages the physical GPU resources.

3. Functionality of vGPU:

- Abstracts the GPU structure, giving applications a consistent view of hardware.
- Handles memory allocation by mapping virtual addresses in the guest OS to real device memory in the host OS.
- Stores the flow of CUDA API calls for proper execution.

4. Workflow:

- CUDA applications on the guest OS send API calls to the vCUDA library.
- The **vCUDA library** redirects these calls to the **vCUDA stub** on the host OS.
- The **vCUDA stub** processes the requests, executes them on the physical GPU, and returns results to the guest OS.

Benefits of vCUDA:

- Enables **GPU acceleration** in virtualized environments without directly running CUDA on hardware-level VMs.
- Efficiently handles resource allocation and execution across guest and host OSes.

Challenges:

- Relies heavily on the client-server architecture and the efficiency of API call redirection.
- Performance may depend on the complexity of GPU tasks and overhead of virtualization.

3.2 Virtualization Structures / Tools And Mechanisms

There are three typical classes of VM architectures, differentiated by the placement of the virtualization layer in the system stack. Virtualization transforms a machine's architecture by inserting a virtualization layer between the hardware and the operating system. This layer converts real hardware into virtual hardware, enabling different operating systems (e.g., Linux and Windows) to run simultaneously on the same physical machine.

Classes of VM Architectures:

1. **Hypervisor Architecture (or VMM):**
 - The hypervisor operates directly on the hardware layer, acting as the virtualization layer.
 - It manages the hardware resources and virtualizes them for guest operating systems.
 - Supports multiple OS instances on the same hardware efficiently.
2. **Para-Virtualization:**
 - A modified version of the guest OS collaborates with the virtualization layer.
 - Offers better performance compared to traditional hypervisors by reducing the overhead of virtualization.
3. **Host-Based Virtualization:**
 - The virtualization layer runs as an application on top of an existing host operating system.
 - Easier to implement but less efficient due to the additional layer of the host OS.

Key Points:

- The **virtualization layer** is crucial for translating real hardware into virtual hardware.
- These architectures enable flexibility in running multiple operating systems on the same machine.
- Hypervisors (or VMMs) and other approaches vary in performance, complexity, and implementation.

3.2.1 Hypervisor and Xen Architecture

The hypervisor (or Virtual Machine Monitor, VMM) enables hardware-level virtualization by acting as an intermediate layer between physical hardware (e.g., CPU, memory, disk, network interfaces) and the operating systems (OS). It facilitates the creation of virtual resources that guest OSes and applications can utilize.

Types of Hypervisor Architectures:

1. **Micro-Kernel Hypervisor:**
 - Only includes essential and unchanging functionalities, such as physical memory management and processor scheduling.
 - Device drivers and other changeable components are kept outside the hypervisor.
 - Examples: Microsoft Hyper-V.
 - **Advantages:** Smaller code size, reduced complexity, and easier maintainability.
2. **Monolithic Hypervisor:**
 - Integrates all functionalities, including device drivers, within the hypervisor itself.
 - Examples: VMware ESX for server virtualization.
 - **Advantages:** Comprehensive functionality but with a larger codebase and potential complexity.

Key Features of a Hypervisor:

- Supports virtualized access to physical hardware through **hypercalls** for guest OSes and applications.
- Converts physical devices into virtual resources for use by virtual machines (VMs).
- Plays a critical role in resource management and scheduling for multiple VMs.

These architectures allow efficient use of physical hardware while enabling multiple OSes to run simultaneously.

3.2.1.1 The Xen Architecture

Xen is an open-source microkernel hypervisor developed at Cambridge University. It acts as a virtualization layer between hardware and the operating system, enabling multiple guest OS instances to run simultaneously. The core components of a Xen system include the **hypervisor, kernel, and applications**.

A key feature of Xen is **Domain 0 (Dom0)**, a privileged guest OS that manages hardware access and resource allocation for other guest domains (**Domain U**). Since Dom0 controls the entire system, its security is critical. If compromised, an attacker can control all virtual machines.

Xen allows users to manage VMs flexibly creating, copying, migrating, and rolling back instances. However, this flexibility also introduces security risks, as VMs can revert to previous states, potentially reintroducing vulnerabilities. Unlike traditional machines with a linear execution timeline, Xen VMs form a **tree-like execution state**, enabling multiple instances and rollbacks, which benefits system management but also creates security challenges.

Note: A key feature of Xen is **Domain 0 (Dom0)**, a privileged virtual machine responsible for managing hardware, I/O operations, and other guest VMs (**Domain U**). Dom0 is the first OS to load and has direct hardware access, allowing it to allocate resources and manage devices for unprivileged guest domains.

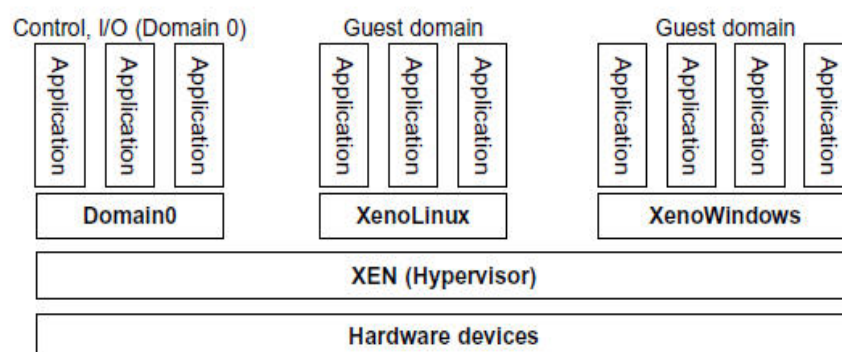


FIGURE 3.5

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])

3.2.2 Binary Translation with Full Virtualization

Hardware virtualization can be categorized into **full virtualization** and **host-based virtualization** based on implementation technologies.

Full Virtualization

- Full virtualization allows a guest OS to run **without modification** by using **binary translation** to handle critical instructions.
- **Noncritical instructions** execute directly on hardware for efficiency, while **critical instructions** are trapped and emulated by the **Virtual Machine Monitor (VMM)**.
- VMware and similar solutions place the **VMM at Ring 0** and the **guest OS at Ring 1**, ensuring complete decoupling from hardware.
- Although binary translation enables compatibility, it introduces **performance overhead**, particularly for **I/O-intensive applications**. Performance on x86 systems typically reaches **80% to 97%** of native execution speed.

Host-Based Virtualization

- In this approach, a **host OS manages hardware**, with a **virtualization layer** placed on top to run guest OSes.
- Unlike full virtualization, the host OS remains in control, **providing drivers and low-level services**, simplifying deployment.
- However, performance suffers due to **multiple layers of mapping**—every hardware request must pass through four layers, leading to significant slowdowns.
- If the **guest OS and hardware have different ISAs**, binary translation is required, further reducing performance.

While host-based virtualization offers **flexibility**, it is generally less efficient than **full virtualization with a VMM**.

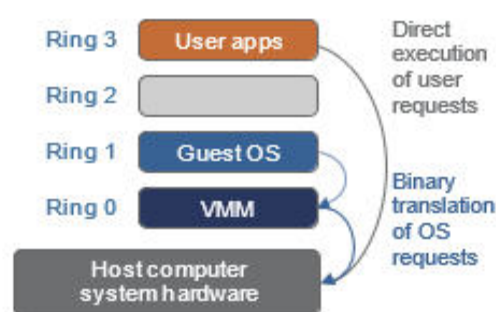


FIGURE 3.6
Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simple instructions on the same host.

3.2.3 Para-Virtualization with Compiler Support

Para-virtualization reduces virtualization overhead by modifying the guest OS to replace **privileged instructions** with **hypercalls** that communicate directly with the hypervisor.

Unlike full virtualization, which relies on **binary translation**, para-virtualization improves performance but introduces compatibility and maintenance challenges.

Challenges of Para-Virtualization

1. **Compatibility & Portability Issues:** Since para-virtualization modifies the guest OS, supporting unmodified OSes becomes difficult.
2. **High Maintenance Costs:** OS kernel modifications require ongoing updates and support.
3. **Variable Performance Gains:** The performance improvement depends on the workload and system architecture.

Para-Virtualization Architecture

Para-Virtualization with Compiler Support

- **Guest OS Modification:** The OS kernel is modified, but user applications may also need changes.
- **Hypercalls:** Privileged instructions that would normally run at **Ring 0** are replaced with **hypercalls** to the hypervisor.
- **Intelligent Compiler:** A specialized compiler assists in identifying and replacing **nonvirtualizable instructions** with hypercalls, optimizing performance.
- **Improved Efficiency:** Compared to full virtualization, para-virtualization significantly **reduces overhead**, making VM execution **closer to native performance**.
- **Limitation:** Since the guest OS is modified, it **cannot run directly on physical hardware** without a hypervisor.

Para-Virtualization on x86 Architecture

- Traditional **x86 processors** have four **privilege levels (Rings 0–3)**:
 - **Ring 0:** Kernel (full control of hardware)

- **Ring 3:** User applications (restricted access)
- In para-virtualization, the guest OS **cannot execute at Ring 0** directly. Instead, it operates at a lower ring, using **hypercalls** to interact with the hypervisor at Ring 0.

Due to the inefficiency of binary translation, many virtualization solutions, including **Xen, KVM, and VMware ESX**, use para-virtualization.

KVM (Kernel-Based Virtual Machine)

- A **Linux-based para-virtualization system**, integrated into the Linux 2.6.20 kernel.
- Leverages **Linux's existing memory management and scheduling**, making it more lightweight than a full hypervisor.
- Supports **unmodified guest OSes**, including Windows, Linux, Solaris, and UNIX variants, with **hardware-assisted virtualization**.

Example 3.3 VMware ESX Server for Para-Virtualization

VMware pioneered the virtualization market, providing solutions for desktops, servers, and data centers. VMware ESX is a bare-metal hypervisor designed for x86 symmetric multiprocessing (SMP) servers, enabling efficient virtualization by directly managing hardware resources.

Key Components of VMware ESX Server

1. Virtualization Layer (VMM Layer)
 - The Virtual Machine Monitor (VMM) virtualizes CPU, memory, network, disk controllers, and human interface devices.
 - Each VM has its own set of virtual hardware resources, isolated from others.
2. Resource Manager
 - Allocates CPU, memory, storage, and network bandwidth to VMs.
 - Maps physical resources to virtual hardware resources for efficient distribution.
3. Hardware Interface Components
 - Includes device drivers that facilitate direct access to I/O devices.
 - Uses a VMkernel-based para-virtualization architecture for improved performance.
4. Service Console (Legacy in ESX, removed in ESXi)
 - Handles system booting, initialization of VMM and Resource Manager, and administrative functions.

Para-Virtualization in ESX

- The VMkernel interacts directly with the hardware, bypassing the need for a host OS.
- Para-virtualized drivers (e.g., VMXNET for networking, PVSCSI for disk I/O) improve performance.
- Provides better efficiency than full virtualization while supporting unmodified guest OSes via hardware-assisted virtualization (Intel VT, AMD-V).

By leveraging para-virtualization, VMware ESX optimizes performance, enhances resource management, and ensures scalability, making it a preferred choice for enterprise data center virtualization.

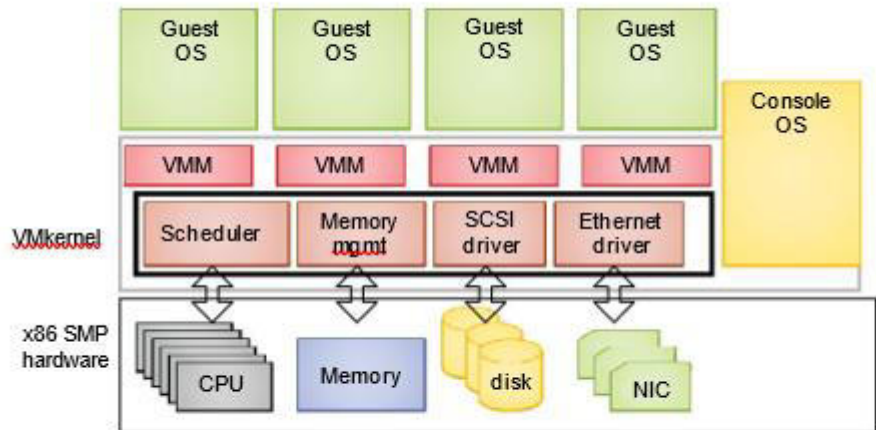


FIGURE 3.9

The VMware ESX [server](#) architecture using para-virtualization.

3.3 Virtualization of CPU, Memory, And I/O Devices

3.3.1 Hardware Support for Virtualization

Modern processors support multiple processes running simultaneously, but they require protection mechanisms to prevent system crashes. This is achieved by dividing execution into user mode and supervisor mode:

- User Mode: Runs unprivileged instructions, restricting direct hardware access.
- Supervisor Mode: Runs privileged instructions, allowing direct control over system hardware.

In virtualized environments, ensuring OS and application correctness is more complex due to additional abstraction layers.

Hardware Virtualization Products

Several solutions leverage hardware support for virtualization, including:

1. VMware Workstation
 - A host-based virtualization software suite for x86 and x86-64 systems.
 - Runs multiple VMs simultaneously on a host OS.
2. Xen Hypervisor
 - Works on IA-32, x86-64, Itanium, and PowerPC 970 architectures.
 - Modifies Linux to function as a hypervisor, controlling guest OSes.
3. KVM (Kernel-Based Virtual Machine)
 - Integrated into the Linux kernel as a virtualization infrastructure.

- Supports hardware-assisted virtualization (Intel VT-x, AMD V) and paravirtualization via the VirtIO framework.
- VirtIO components include:
 - Paravirtual Ethernet Card (for networking).
 - Disk I/O Controller (optimized storage access).
 - Balloon Device (dynamically adjusts VM memory allocation).
 - VGA Graphics Interface (enhanced graphics performance using VMware drivers).

By leveraging hardware-assisted virtualization, modern hypervisors like VMware, Xen, and KVM achieve higher efficiency, reduced overhead, and improved performance for virtual machines.

Example 3.4: Hardware Support for Virtualization in the Intel x86 Processor

Software-based virtualization methods are complex and introduce performance overhead. To address this, Intel provides hardware-assisted virtualization techniques that simplify implementation and improve efficiency.

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.

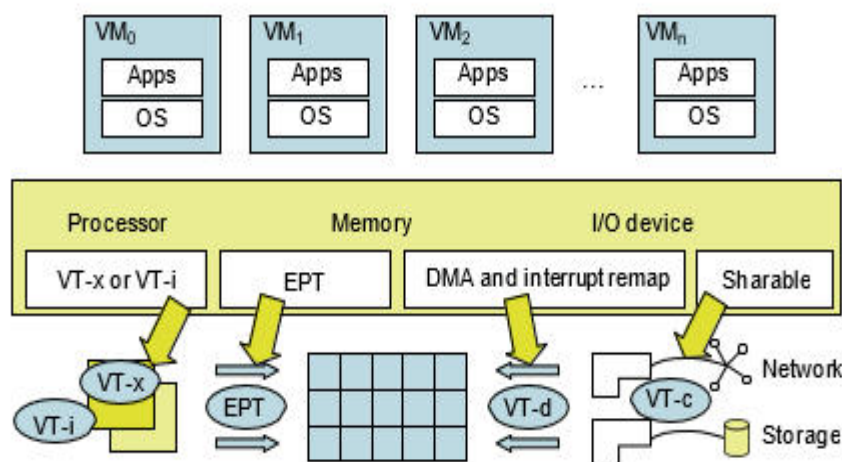


FIGURE 3.10

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

3.3.2 CPU Virtualization

A Virtual Machine (VM) replicates a real computer system, executing most instructions on the host processor in native mode for efficiency. However, critical instructions must be carefully managed to ensure stability and correctness.

Types of Critical Instructions

1. Privileged Instructions
 - Execute only in privileged mode.
 - If executed in user mode, they trigger a trap.
2. Control-Sensitive Instructions
 - Modify system resources (e.g., changing memory configuration).
3. Behavior-Sensitive Instructions
 - Change behavior based on system configuration (e.g., memory load/store operations).

Virtualizability of CPU Architectures

- RISC CPUs (e.g., PowerPC, SPARC) are naturally virtualizable since all sensitive instructions are privileged.
- x86 CPUs were not originally designed for virtualization, as some sensitive instructions (e.g., SGDT, SMSW) are not privileged.
 - These instructions bypass the VMM, making virtualization difficult without software-based techniques like binary translation.

Paravirtualization in CPU Virtualization

- In a UNIX-like system, a system call triggers an 80h interrupt, transferring control to the OS kernel.
- In a paravirtualized system (e.g., Xen), the system call:
 1. Triggers the 80h interrupt in the guest OS.
 2. Also triggers the 82h interrupt in the hypervisor.
 3. The hypervisor processes the call before returning control to the guest OS kernel.
- Benefit: Runs unmodified applications in a VM.
- *Drawback: Small performance overhead due to additional hypervisor intervention.*

Hardware-Assisted CPU Virtualization

To simplify virtualization, Intel and AMD introduced hardware extensions:

- Intel VT-x / AMD-V: Adds a new privilege mode (Ring -1).
 - Hypervisor runs at Ring -1.
 - Guest OS runs at Ring 0 (normal OS mode).
 - All privileged instructions are automatically trapped into the hypervisor.

Advantages of Hardware-Assisted Virtualization:

- i. Removes the need for binary translation (improving performance).
- ii. Allows unmodified operating systems to run in virtual machines.
- iii. Simplifies virtualization implementation.

Example 3.5: Intel Hardware-Assisted CPU Virtualization

Virtualizing x86 Processors

- Unlike RISC processors, x86 processors were not originally virtualizable.
- Since x86-based legacy systems are widely used, virtualization is necessary.

Intel VT-x Technology

- Introduces VMX Root Mode, an additional privilege level.
- Adds special instructions to:
 - Start/stop VMs.
 - Allocate memory pages to store CPU states.
- Used in hypervisors like Xen, VMware, and Microsoft Virtual PC.

Performance Considerations

- High efficiency expected, but switching between hypervisor and guest OS causes overhead.
- Hybrid Approach (used by VMware):
 - Offloads some tasks to hardware while keeping others in software.
- Combining Para-Virtualization with Hardware-Assisted Virtualization further boosts performance.

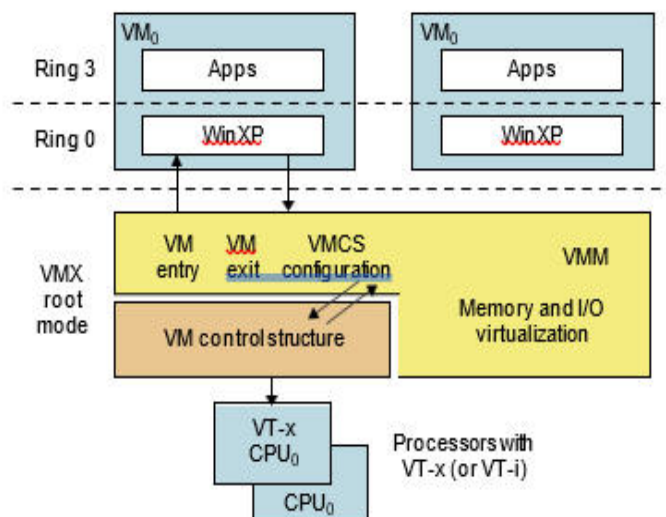


FIGURE 3.11

Intel hardware-assisted CPU virtualization.

3.3.3 Memory Virtualization

1. Virtual Memory Virtualization Basics

- Similar to traditional virtual memory in modern OSes.
- Standard OSes use page tables for one-stage mapping (Virtual Memory → Physical Memory).
- Virtualized environments require a two-stage mapping:
 - Guest OS: Virtual Memory → Physical Memory (Guest View).
 - VMM: Physical Memory → Machine Memory (Actual Hardware).

2. Memory Management in Virtualized Systems

- Memory Management Unit (MMU) and Translation Lookaside Buffer (TLB) help optimize performance.
- Guest OS controls virtual-to-physical mapping but cannot directly access machine memory.
- VMM (Hypervisor) handles actual memory allocation to prevent conflicts.

3. Shadow Page Tables & Nested Paging

- Shadow Page Table (SPT):
 - Maintained by the VMM to map guest physical memory → machine memory.
 - Reduces guest OS overhead but increases memory usage and performance costs.
- Nested Page Tables (NPT):
 - Adds another layer of address translation.
 - First introduced in AMD Barcelona (2007) as hardware-assisted memory virtualization.
 - Reduces VMM overhead by offloading translation to hardware.

4. VMware's Approach

- Uses shadow page tables to map virtual memory to machine memory.
- TLB caching avoids repeated translations, improving performance.
- AMD and Intel now support hardware-assisted memory virtualization to reduce overhead.

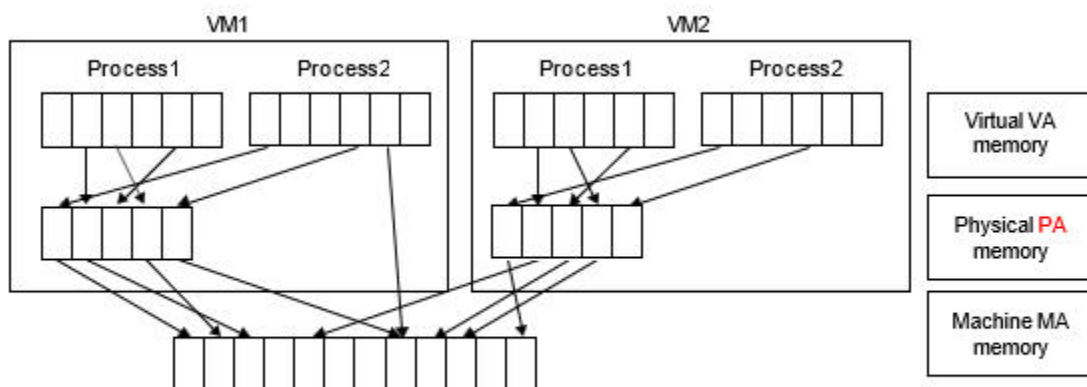


FIGURE 3.12

Two-level memory mapping procedure.

Example 3.6 Extended Page Table by Intel for Memory Virtualization

1. The Problem with Shadow Page Tables

- Software-based shadow page tables were inefficient and caused high performance overhead.
- Frequent memory lookups and context switches slowed down virtualized environments.

2. Intel's EPT Solution

- Hardware-assisted memory virtualization that eliminates the need for shadow page tables.
- Works with Virtual Processor ID (VPID) to optimize Translation Lookaside Buffer (TLB) usage.
- Reduces memory lookup time and improves performance significantly.

3. How EPT Works

- Uses a four-level page table hierarchy (same as guest OS page tables).
- Translation Process:
 1. Guest OS uses Guest CR3 (Control Register 3) to point to L4 page table.
 2. CPU must translate Guest Physical Address (GPA) to Host Physical Address (HPA) using EPT.
 3. The CPU first checks the EPT TLB for an existing translation.
 4. If not found, it searches the EPT page tables (up to 5 times in the worst case).
 5. If still not found, an EPT violation exception is triggered.
 6. The CPU will access memory multiple times to resolve the mapping (up to 20 memory accesses).

4. Optimization: EPT TLB Expansion

- Intel increased the size of EPT TLB to store more translations and reduce memory accesses.
- This dramatically improves memory access speed and virtualization efficiency.

5. Impact of EPT on Virtualization

- Removes the need for shadow page tables, reducing overhead.
- Works seamlessly with Intel VT-x for faster CPU virtualization.
- Improves performance for workloads running in virtualized environments like VMware, Xen, and KVM.

3.3.4 I/O Virtualization

I/O virtualization allows virtual machines (VMs) to share and access physical I/O devices efficiently. There are three main methods to implement I/O virtualization:

1. Full Device Emulation

- The VMM (Virtual Machine Monitor) emulates well-known hardware devices.
- Guest OS interacts with a virtual device, which is mapped to a real physical device by the virtualization layer.
- Functions performed by the virtualization layer:
 - Emulates device features like interrupts, DMA, and device enumeration.
 - Remaps guest OS I/O addresses to real device addresses.
 - Multiplexes and routes I/O requests from multiple VMs.
 - Supports advanced features like Copy-on-Write (COW) disks.
- Advantages:
 - Allows compatibility with unmodified guest OSes.
- Disadvantages:
 - High overhead and slow performance due to software emulation.

2. Para-Virtualization (Split Driver Model)

- Used in Xen and other virtualization platforms.

- Uses a frontend driver (inside the guest OS) and a backend driver (inside the hypervisor or privileged VM).
- How it works:
 - Frontend driver (Domain U): Handles I/O requests from guest OS.
 - Backend driver (Domain 0): Manages real I/O devices and routes data between VMs.
 - Communication occurs via a shared memory block.
- Advantages:
 - Better performance than full device emulation.
- Disadvantages:
 - Higher CPU overhead due to the need for additional processing.

3. Direct I/O Virtualization

- Allows direct access between the VM and the physical device.
- Benefits:
 - Near-native performance because I/O operations bypass software emulation.
 - Lower CPU overhead compared to para-virtualization.
- Challenges:
 - Current implementations focus mainly on networking for mainframes.
 - Difficulties in reclaiming and reassigning physical devices after VM migration.
 - Risk of arbitrary device states (e.g., DMA errors leading to system crashes).

4. Hardware-Assisted I/O Virtualization

- Intel VT-d technology supports I/O DMA remapping and device interrupt remapping.
- Helps unmodified, specialized, or virtualization-aware guest OSes run efficiently.

5. Self-Virtualized I/O (SV-IO)

- Uses multicore processors to manage I/O virtualization tasks.
- Virtual Interface (VIF) Model:
 - Provides virtual devices with an access API for VMs.
 - Provides a management API for the VMM.
 - Each virtual device (VIF) has a unique ID and uses message queues for communication.
- Applications:
 - Virtual network interfaces
 - Virtual block devices (disks)
 - Virtual camera devices

Summary

Method	Performance	CPU Overhead	Hardware Dependency
Full Device Emulation	Low	High	No
Para-Virtualization	Moderate	Moderate	Yes (Modified OS)
Direct I/O	High	Low	Yes (Special Hardware)

Method	Performance	CPU Overhead	Hardware Dependency
Hardware-Assisted I/O (VT-d)	High	Low	Yes
Self-Virtualized I/O (SV-IO)	High	Low-Moderate	Yes

I/O virtualization continues to evolve, with **hardware-assisted methods** like **VT-d** and **SV-IO** improving efficiency and reducing overhead.

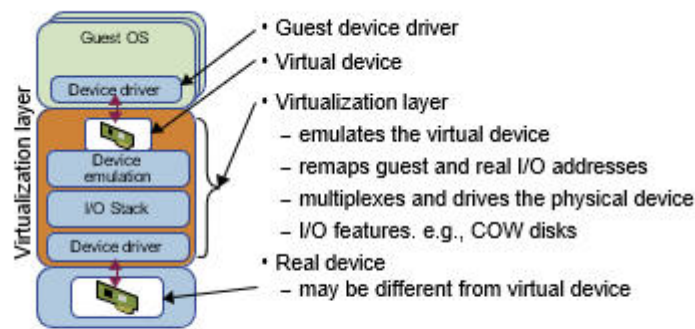


FIGURE 3.14

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.

Example 3.7 VMware Workstation for I/O Virtualization

VMware Workstation is a hosted hypervisor that runs as an application on a host operating system (OS). It relies on guest OS, host OS, and VMM (Virtual Machine Monitor) to implement I/O virtualization efficiently.

How VMware Workstation Implements I/O Virtualization

1. VMAp (Application Layer):
 - Runs as an application inside the host OS.
 - Manages VM operations and user interactions.
2. VMDriver (Host OS Driver):
 - Loaded into the host OS kernel.
 - Acts as a bridge between VMAp and VMM.
 - Facilitates control transfers between the host OS and the VMM world.
3. Virtual Machine Monitor (VMM):
 - Runs at a privileged level directly on the hardware.
 - Manages the execution of guest OSes and handles virtualization tasks.
4. Processor Execution:
 - A physical processor operates in two distinct states:
 - Host World: Executes tasks for the host OS.
 - VMM World: Runs the guest OS and virtualized workloads.
 - VMDriver facilitates transitions between these two execution states.

Advantages of VMware Workstation's I/O Virtualization

1. Broad hardware compatibility (supports various guest OSes without modification).
2. Runs as a regular application on host OS, making installation and management simple.
3. Leverages host OS drivers for better I/O support.

Disadvantages

1. Performance overhead due to full device emulation.
2. Slower than direct I/O virtualization approaches.

3.3.5 Virtualization in multicore Processors

As multi-core processors become more prevalent, virtualizing them presents unique challenges compared to single-core processors. While multi-core CPUs offer higher performance by integrating multiple cores on a single chip, virtualization introduces complexities in task scheduling, parallel execution, and resource management.

Challenges in Multi-Core Virtualization

1. Parallelism & Programming Models

- Applications must be parallelized to fully utilize multiple cores.
- New programming models, languages, and libraries are required to simplify parallel programming.

2. Task Scheduling & Resource Management

- Efficient scheduling algorithms are needed to distribute tasks across cores.
- Resource allocation policies must balance performance, complexity, and power efficiency.

3. Dynamic Heterogeneity

- The integration of different types of cores (fat CPU cores & thin GPU cores) on the same chip makes resource management more complex.
- As transistor reliability decreases and complexity increases, system designers must adapt scheduling techniques dynamically.

Multi-core virtualization presents challenges in parallel execution, task scheduling, and resource management. Applications need to be parallelized, and efficient scheduling is crucial for performance. Dynamic heterogeneity, where CPU and GPU cores coexist, further complicates resource allocation.

To address these challenges:

- Virtual Processor Cores (VCPUs) abstract low-level hardware management, allowing dynamic core migration and suspension.
- Virtual Hierarchies adapt cache and coherence structures to optimize workload distribution, reducing cache misses and improving isolation between VMs.

These advancements enhance performance, flexibility, and efficiency, making multi-core virtualization ideal for server consolidation and cloud computing.

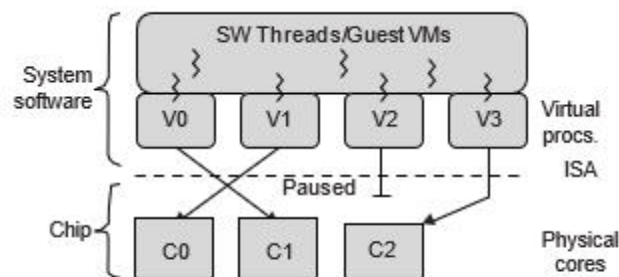


FIGURE 3.16

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

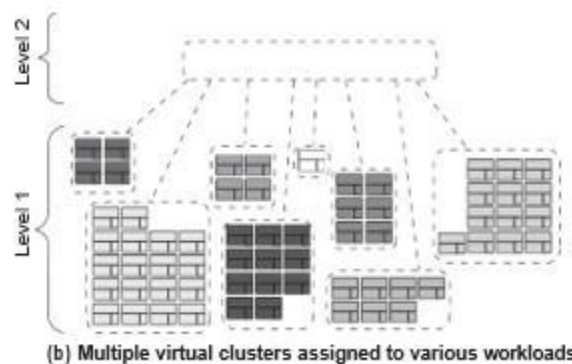
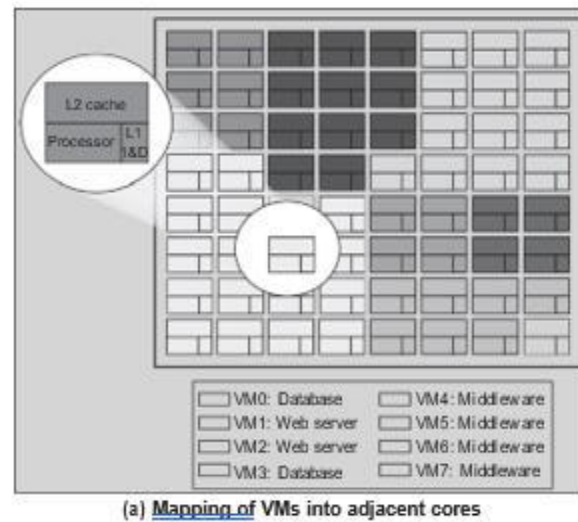


FIGURE 3.17

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

3.4 VIRTUAL CLUSTERS AND RESOURCE MANAGEMENT

What are Virtual Clusters?

A physical cluster is a group of physical servers connected through a network. In contrast, a virtual cluster consists of virtual machines (VMs) spread across multiple physical machines. These VMs are connected via a virtual network and can be managed flexibly.

Key Characteristics of Virtual Clusters:

- VMs can run on physical machines and be assigned dynamically.
- Each VM runs its own operating system, which can differ from the host system.
- VMs improve server utilization by running multiple applications on the same physical hardware.
- VMs can be replicated across multiple servers to enhance performance, fault tolerance, and disaster recovery.
- Virtual clusters can expand or shrink dynamically, depending on demand.
- If a physical machine fails, only the VMs on that machine are affected, while other VMs and the host system continue running.

Challenges in Virtual Cluster Management:**1. Fast Deployment & Scheduling**

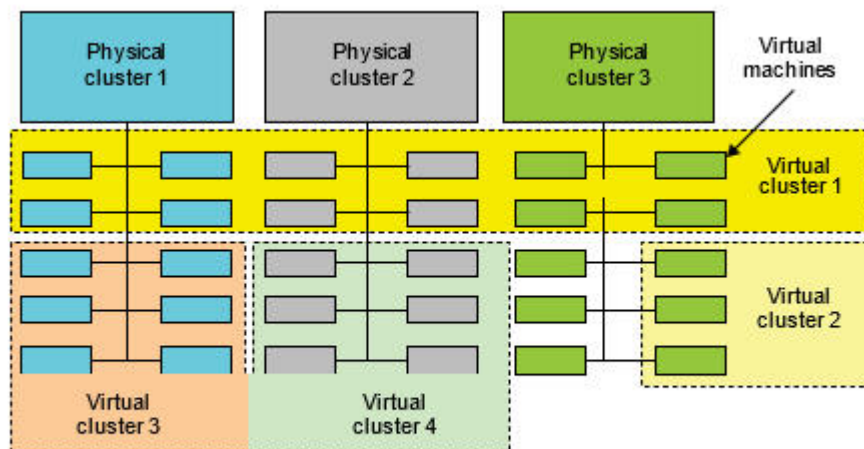
- Virtual clusters must be quickly set up, shut down, and switched to optimize resource use.
- Green computing techniques aim to save energy, but live VM migration can cause performance overhead.
- Load balancing strategies help distribute workloads efficiently across VMs.

2. High-Performance Virtual Storage

- VM images (software templates) must be stored efficiently to reduce space and improve deployment speed.
- Copy-on-Write (COW) technology allows creating new VMs quickly by modifying existing templates instead of duplicating entire disk images.
- Automating VM configurations can save time when managing large numbers of VMs.

Conclusion:

Virtual clusters provide flexibility, efficient resource usage, and better fault tolerance. However, they require careful management for fast deployment, effective load balancing, and optimized storage. Strategies like automated configuration and optimized migration help improve performance while reducing overhead.

**FIGURE 3.18**

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

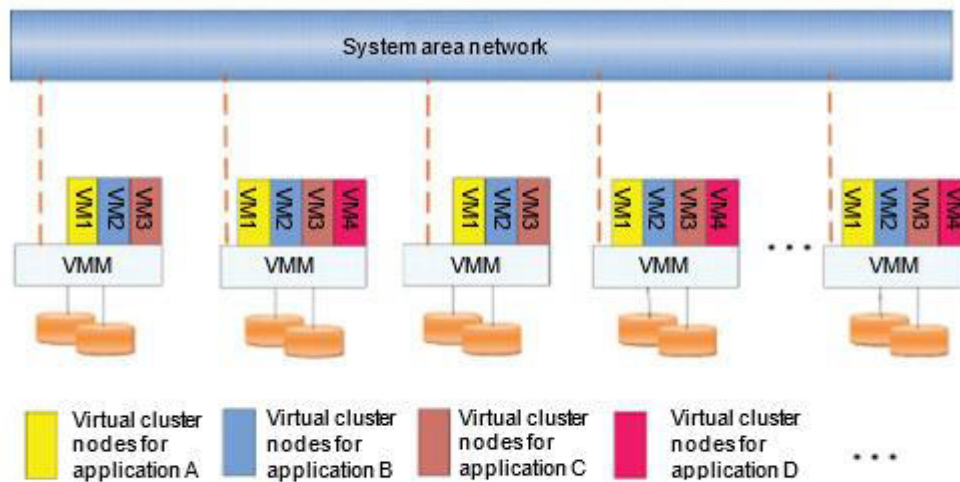


FIGURE 3.19

The concept of a virtual cluster based on application partitioning.

3.4.2 Live VM Migration Steps and Performance Effects

In a mixed-node cluster, virtual machines (VMs) typically run on physical hosts, but if a host fails, its VM role can be taken over by another VM on a different host. This enables flexible failover compared to traditional physical-to-physical failover. However, if a host fails, its VMs also fail, which can be mitigated through live VM migration.

Virtual Cluster Management Approaches:

1. Guest-Based Manager: The cluster manager runs inside the guest OS (e.g., OpenMosix, Sun's Oasis).
2. Host-Based Manager: The cluster manager runs on the host OS, supervising VMs (e.g., VMware HA).
3. Independent Manager: Both guest and host have separate cluster managers, increasing complexity.
4. Integrated Cluster Management: A unified manager controls both virtual and physical resources.

Live VM Migration Process (6 Steps):

1. Start Migration: Identify the VM and destination host, often triggered by load balancing or server consolidation strategies.
2. Memory Transfer: The VM's memory is copied to the destination host in multiple rounds, ensuring minimal disruption.
3. Suspend and Final Copy: The VM pauses briefly to transfer the last memory portion, CPU, and network states.
4. Commit and Activate: The destination host loads the VM state and resumes execution.
5. Redirect Network & Cleanup: The network redirects to the new VM, and the old VM is removed.

Performance Effects:

- The first memory copy takes 63 seconds, reducing network speed from 870 MB/s to 765 MB/s.
- Additional memory copy rounds further reduce speed to 694 MB/s in 9.8 seconds.
- The total downtime is only 165 milliseconds, ensuring minimal service disruption.

Key Benefits of Live Migration:

- Ensures continuous service availability in cloud computing, HPC, and computational grids.
- Enables dynamic resource allocation on demand.
- Supports disaster recovery with minimal overhead.
- Prevents resource contention with careful network and CPU usage planning.

Live VM migration enhances cloud computing by enabling seamless workload balancing and minimizing downtime during host failures. Platforms like VMware and Xen support these migrations, allowing multiple VMs to run efficiently on a shared physical infrastructure.

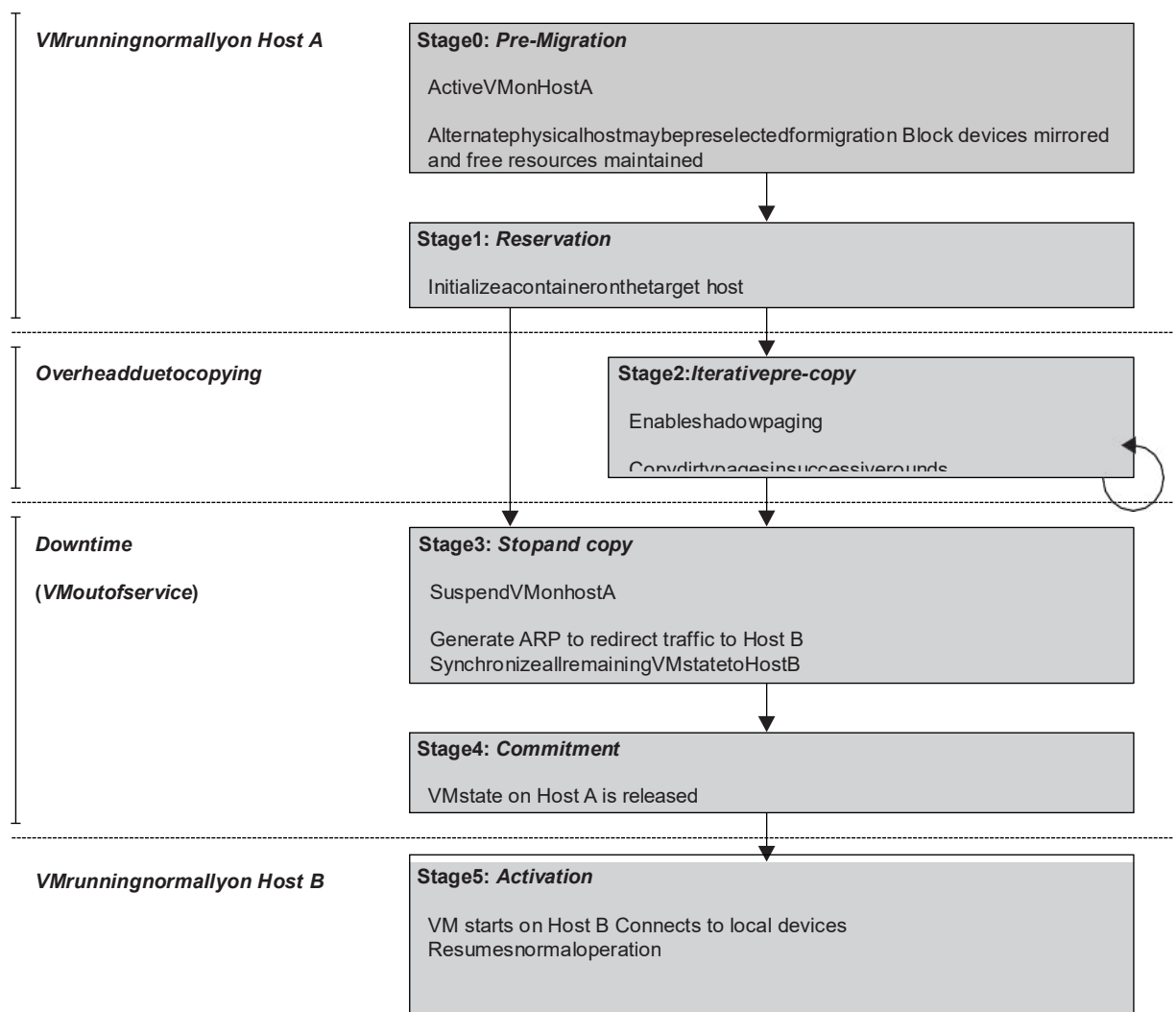


Figure. 3.20 Live migration process of a VM from one host to another.

3.4.3 Migration of Memory, Files, and Network Resources

Shared clusters reduce costs and improve resource utilization. When migrating a system to a new physical node, key factors include **memory migration, file system migration, and network migration.**

1. Memory Migration

- VM memory migration involves moving memory states efficiently.
- The Internet Suspend-Resume (ISR) technique uses temporal locality to transfer only changed data.
- A tree-based file structure minimizes data transfer by copying only modified parts.
- ISR results in high downtime, making it unsuitable for live migrations.

2. File System Migration

- VM migration requires a consistent, location-independent file system accessible on all hosts.
- Two approaches:
 1. Virtual Disk Transport (copies entire disk but is slow for large data).
 2. Global Distributed File System (avoids copying, supports direct network access).
- Smart Copying uses spatial locality, transferring only differences between the old and new locations.
- Proactive State Transfer predicts the destination and pre-transfers critical data.

3. Network Migration

- Migrating VMs must maintain network connections without disruption.
- Each VM has a virtual IP and MAC address, separate from the host machine.
- Unsolicited ARP replies notify network peers of the VM's new location.
- Switched networks detect new VM locations and reroute traffic automatically.

4. Live VM Migration Techniques

- Precopy Approach:
 - Transfers all memory pages first, then iteratively copies only modified pages.
 - Reduces downtime but increases total migration time.
- Checkpoint/Recovery & Trace/Replay (CR/TR-Motion):
 - Transfers execution logs instead of dirty pages, minimizing migration time.
 - Limited by differences in source and target system performance.
- Postcopy Approach:
 - Transfers memory pages once but has higher downtime due to fetch delays.
- Memory Compression:
 - Uses spare CPU resources to compress memory pages before transfer, reducing data size.

5. Live Migration Using Xen

- Xen Hypervisor allows multiple OSes to share hardware.
- Domain 0 (Dom0) manages VM creation, termination, and migration.

- Xen uses a send/receive model to transfer VM states efficiently.

Key Takeaways

- Live migration ensures minimal downtime while keeping services running.
- Optimized memory, file, and network transfer techniques improve efficiency.
- Different migration strategies (Precopy, Postcopy, CR/TR-Motion, and Compression) balance speed and downtime based on system needs.
- Xen provides a reliable VM migration framework for enterprise environments.

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

What is Live Migration?

Live migration is the process of moving a running Virtual Machine (VM) from one physical machine to another without stopping its operations. This means users experience little to no downtime while the VM is being transferred.

How Does Xen Handle Live Migration?

Xen uses a method called Remote Direct Memory Access (RDMA) to speed up migration by bypassing traditional network communication protocols like TCP/IP. This avoids unnecessary processing by the CPU, making the transfer faster and more efficient.

Key Steps in Xen's Live Migration Process:

1. Migration Daemon (a background process) manages the migration.
2. Shadow Page Tables track memory changes during the migration process.
3. Dirty Bitmaps record modified memory pages. These pages are updated and sent to the new machine in precopy rounds.
4. Memory pages are compressed before transfer to reduce data size and improve speed.
5. The new host decompresses the memory pages and resumes the VM.

Trade-offs in Migration

- The compression algorithm must be fast and effective for different types of memory data.
- Using a single compression method for all memory pages is not efficient because different memory types require different strategies.

Conclusion

Live migration in Xen, enhanced by RDMA, allows seamless VM transfer with minimal impact on performance. Techniques like precopying, dirty bitmaps, and compression improve efficiency while ensuring smooth operation.

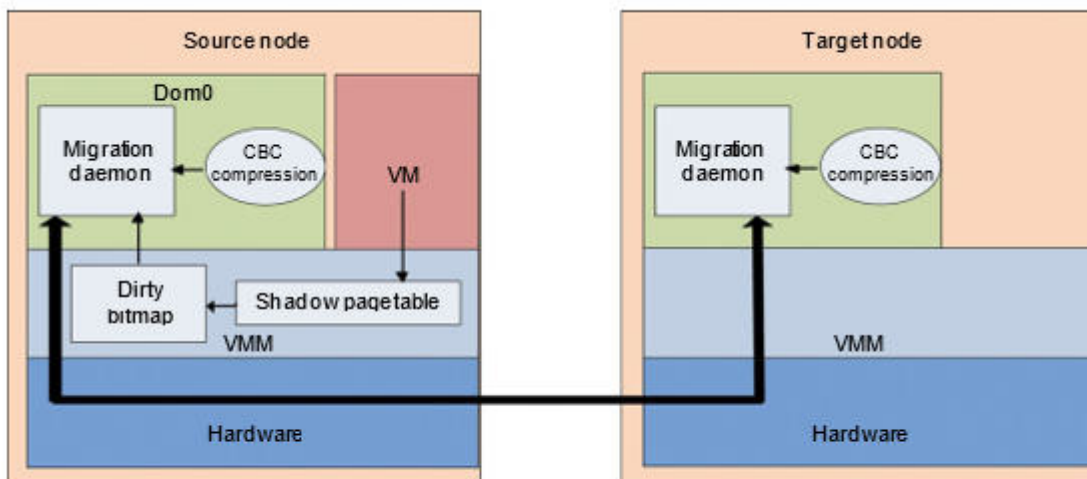


FIGURE 3.22

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

3.4.4 Dynamic Deployment of Virtual Clusters

What is Dynamic Deployment of Virtual Clusters?

Dynamic deployment allows virtual clusters (vClusters) to change in size, move, or adapt to resource demands. This helps in efficient resource management, improved performance, and cost savings in cloud computing.

Research Projects on Virtual Clusters:

1. Cluster-on-Demand (COD) – Duke University
 - Goal: Dynamically allocate servers to multiple vClusters.
 - Results: Efficient sharing of VMs using Sun GridEngine.
2. Cellular Disco – Stanford University
 - Goal: Deploy a virtual cluster on a shared-memory multiprocessor system.
 - Results: Multiple VMs successfully managed under Cellular Disco Virtual Machine Monitor (VMM).
3. VIOLIN – Purdue University
 - Goal: Improve performance through dynamic adaptation of VMs.
 - Results: Reduced execution time for parallel applications by adapting to changing workloads.
4. GRAAL – INRIA, France
 - Goal: Evaluate parallel algorithms in Xen-enabled virtual clusters.
 - Results: Achieved 75% of maximum performance while using only 30% of the total resources.

Example 3.9: COD (Cluster-on-Demand) – Duke University

COD is a virtual cluster management system that allows automatic resizing and reallocation of clusters.

- Uses Sun GridEngine for scheduling workloads.
- Users can request resources dynamically via a web interface.
- Figure 3.24 shows how the number of servers in different vClusters changed over eight days based on demand.

- Helps with load balancing, resource reservation, and automatic provisioning.

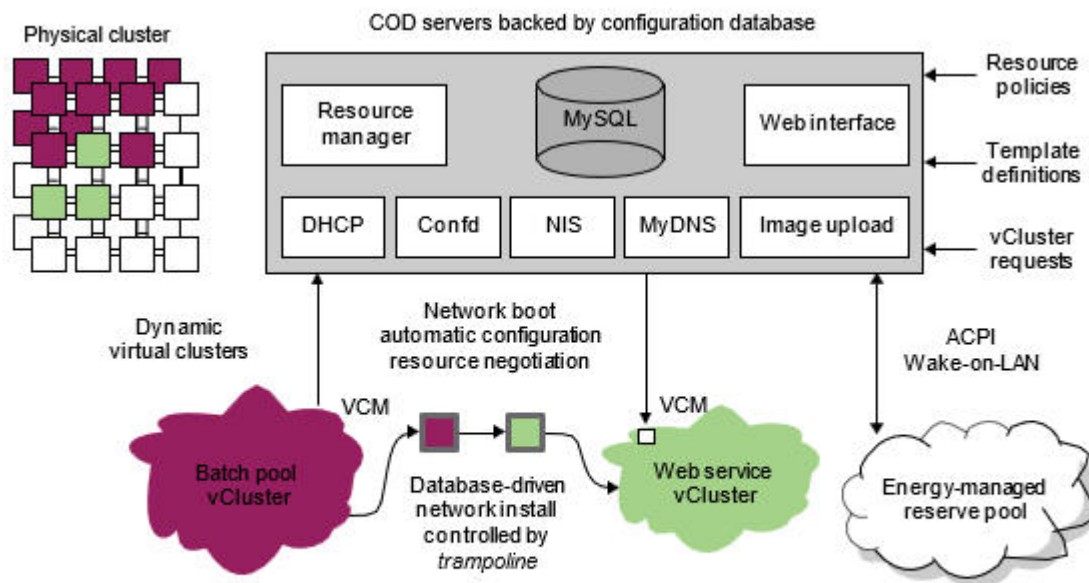


FIGURE 3.23

COD partitioning a physical cluster into multiple virtual clusters.

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

Example 3.10: VIOLIN – Purdue University

VIOLIN focuses on live VM migration to dynamically adjust virtual clusters for better resource use.

- Supports multiple virtual environments (VIOLIN 1–5) running on shared physical clusters.
- Can move VMs between clusters as needed without disrupting applications.
- Key result: Increased resource utilization with only a 1% increase in execution time.

Note:

- Virtual clusters help efficiently manage and allocate resources.
- COD and VIOLIN show that dynamic adaptation can significantly improve resource utilization.
- Live migration allows VMs to be moved with minimal downtime.
- These techniques enable scalable, flexible, and cost-effective cloud computing solutions.

3.5 VIRTUALIZATION FOR DATA-CENTER AUTOMATION

Data centers have expanded rapidly, with major IT companies like Google, Amazon, and Microsoft investing heavily in automation. This automation dynamically allocates hardware, software, and database resources to millions of users while ensuring cost-effectiveness and Quality of Service (QoS). The rise of virtualization and cloud computing has driven this transformation, with market growth from \$1.04 billion in 2006 to a projected \$3.2 billion by 2011.

Server Consolidation in Data Centers

Data centers handle heterogeneous workloads, categorized as:

- Chatty workloads (e.g., web video services) that have fluctuating demand.
- Noninteractive workloads (e.g., high-performance computing) that require consistent resource allocation.

To meet peak demand, resources are often statically allocated, leading to underutilized servers and wasted costs in hardware, space, and power. Server consolidation—particularly virtualization-based consolidation—optimizes resource management by reducing physical servers and improving hardware utilization.

Benefits of Server Virtualization

- Enhances hardware utilization by consolidating underutilized servers.
- Improves resource provisioning through agile VM deployment.
- Reduces costs, including server purchases, maintenance, power, cooling, and cabling.
- Enhances availability and business continuity, as VM crashes do not impact other VMs or the host system.

Challenges and Optimization Strategies

- Resource Scheduling: Efficient, multi-level schedulers improve utilization and QoS.
- Dynamic CPU Allocation: Adjusts resources based on VM utilization and workload demand.
- Two-Level Resource Management: Uses local VM controllers and global server controllers for optimized allocation.
- Power Management: A VM-aware power budgeting scheme balances power savings with performance while addressing hardware heterogeneity.

By leveraging virtualization and multicore processing (CMP), data centers can enhance efficiency, but optimization in memory access, VM reassignment, and power management remains a challenge.

Virtual Storage Management

- Storage virtualization in system environments differs from traditional storage aggregation, focusing on VM image management and application data handling.
- A key challenge is storage bottlenecks, as VMs compete for disk resources.
- Parallax, a distributed storage system, addresses these challenges by offering scalable virtual disks across a common physical storage device. It enhances storage flexibility, reduces storage footprint, and supports advanced features like snapshots.

Overall, virtualization significantly improves data-center efficiency by optimizing server and storage management, reducing costs, and enabling scalable, automated resource allocation.

Example 3.11 – Parallax Virtual Storage System

Parallax is a scalable virtual storage system designed for cluster-based environments. It enables efficient storage management by using a set of per-host storage appliances that share access to a common block device.

Key Features of Parallax:

- Cluster-Wide Storage Management: A centralized administrative domain manages all storage appliance VMs.
- Pushes Storage Functionality to Hosts: Features like snapshots are implemented in software instead of dedicated hardware.

- Virtual Disk Images (VDIs): Provides single-writer virtual disks, accessible from any physical host in the cluster.
- Efficient Block Virtualization: Uses Xen's block tap driver and tapdisk library for handling block storage requests across VMs.
- Storage Appliance VM: Acts as an intermediary between client VMs and physical hardware, facilitating live upgrades of block device drivers.

Parallax enhances flexibility, scalability, and ease of storage management in virtualized data centers by integrating advanced block storage virtualization techniques.

3.5.3 Cloud OS for Virtualized Data Centers

To function as cloud providers, data centers must be virtualized using Virtual Infrastructure (VI) managers and Cloud OSes. Table 3.6 outlines four such platforms:

1. Nimbus (Open-source)
2. Eucalyptus (Open-source)
3. OpenNebula (Open-source)
4. vSphere 4 (Proprietary, VMware)

Key Features of VI Managers & Cloud OSes:

- VM Creation & Management: All platforms support virtual machines and virtual clusters for elastic cloud resources.
- Virtual Networking: Nimbus, Eucalyptus, and OpenNebula offer virtual network support, enabling flexible communication between VMs.

- Dynamic Resource Provisioning: OpenNebula stands out by allowing advance reservations of cloud resources.
- Hypervisor Support:
 - Nimbus, Eucalyptus, and OpenNebula use Xen & KVM for virtualization.
 - vSphere 4 utilizes VMware ESX & ESXi hypervisors.
- Virtual Storage & Data Protection: Only vSphere 4 supports virtual storage along with networking and data protection.

Example 3.12 Eucalyptus for Virtual Networking of Private Cloud

Eucalyptus is an open-source software system designed for private cloud infrastructure and Infrastructure as a Service (IaaS). It enables virtual networking and VM management, but does not support virtual storage.

Key Features of Eucalyptus:

- Private Cloud Deployment:
 - Supports Ethernet and Internet-based networking to connect VMs.
 - Can interact with public and private clouds.
- Component-based Web Services Architecture:
 - Uses WS-Security policies for secure communication.
 - Web services expose language-agnostic APIs via WSDL documents.
- Resource Managers in Eucalyptus:
 - Instance Manager: Manages VM execution, inspection, and termination.
 - Group Manager: Handles scheduling and virtual network management.
 - Cloud Manager: Central user entry-point, queries nodes, makes scheduling decisions.
- AWS Compatibility:
 - Works like Amazon EC2 APIs and supports S3 storage emulation.
 - Compatible with SOAP, REST, and CLI-based management.
- Platform Support:

- Installed on Linux-based platforms.

Eucalyptus provides a flexible and scalable solution for private cloud networking but lacks some security and general-purpose cloud features.

Example 3.13 VMware vSphere 4 as a Commercial Cloud OS

vSphere 4, released by VMware in April 2009, is a virtualization platform designed for private cloud management. It extends earlier VMware products like Workstation, ESX, and Virtual Infrastructure. The system interacts with applications through vCenter and provides infrastructure and application services.

The infrastructure services include three components:

- vCompute (ESX, ESXi, DRS)
- vStorage (VMS, thin provisioning)
- vNetwork (distributed switching and networking)

The application services focus on:

- Availability (VMotion, Storage VMotion, HA, Fault Tolerance, Data Recovery)
- Security (vShield Zones, VMsafe)
- Scalability (DRS, Hot Add)

Users must understand vCenter interfaces to manage applications effectively. More details are available on the vSphere 4 website.

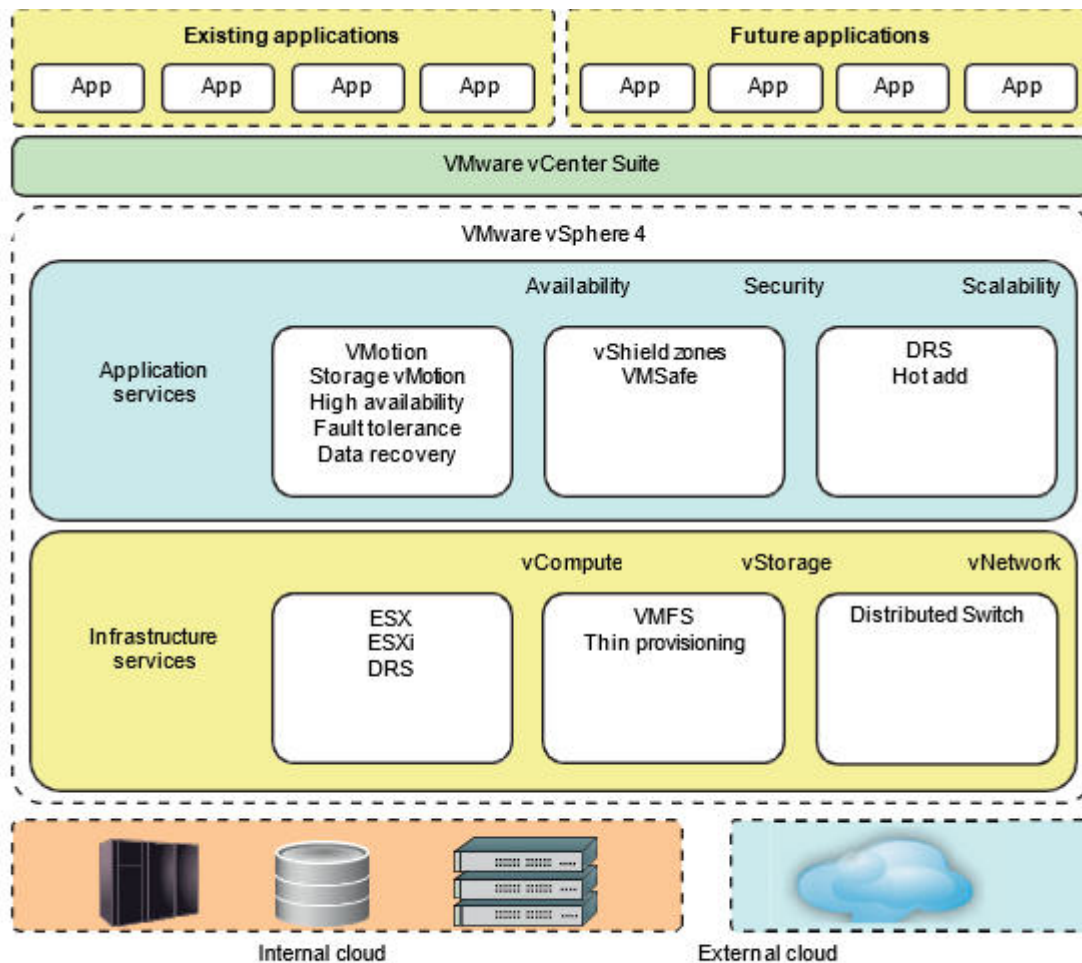


FIGURE 3.28

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

3.5.4 Trust Management in Virtualized Data Centers

A Virtual Machine Monitor (VMM) creates and manages Virtual Machines (VMs) by acting as a software layer between the operating system and hardware. It provides secure isolation and manages access to hardware resources, making it the foundation of security in virtualized environments. However, if a hacker compromises the VMM or management VM, the entire system is at risk. Security issues also arise from random number reuse, which can lead to encryption vulnerabilities and TCP hijacking attacks.

VM-Based Intrusion Detection

Intrusion Detection Systems (IDS) help identify unauthorized access. IDS can be:

- Host-based IDS (HIDS) – Runs on a monitored system but is vulnerable to attacks.
- Network-based IDS (NIDS) – Monitors network traffic but struggles to detect fake actions.

A VM-based IDS leverages virtualization to isolate VMs, preventing compromised VMs from affecting others. The Virtual Machine Monitor (VMM) can audit access requests, combining the strengths of HIDS and NIDS. There are two methods for implementation:

1. IDS as an independent process inside a high-privileged VM on the VMM.
2. IDS integrated into the VMM with direct hardware access.

Garfinkel and Rosenblum proposed a VMM-based IDS that monitors guest VMs using a policy framework and trace-based security enforcement. However, logs used for analysis can be compromised if the operating system is attacked.

Honeypots and Honeynets

Besides IDS, honeypots and honeynets are used to detect attacks by tricking attackers into interacting with fake systems. Honeypots can be physical or virtual, and in virtual honeypots, the host OS and VMM must be protected to prevent attacks from guest VMs.

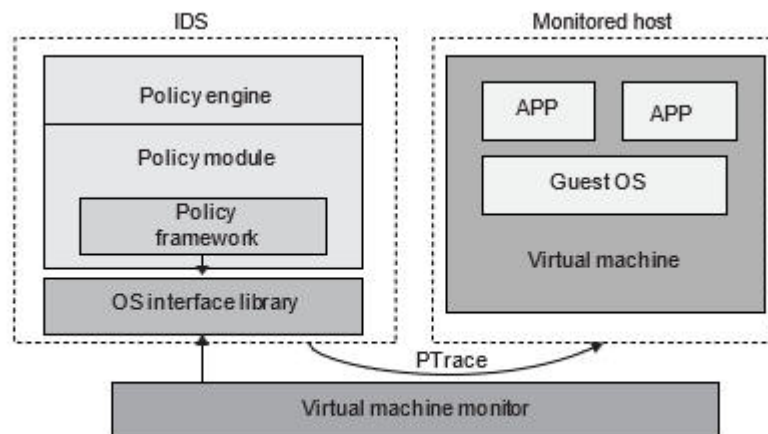


FIGURE 3.29

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

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

EMC and VMware collaborated to develop security middleware for trust management in distributed systems and private clouds. The concept of trusted zones was introduced to enhance security in virtual clusters, where multiple applications and OS instances for different tenants operate in separate virtual environments.

Trusted Zones Architecture

- Physical Infrastructure (Cloud Provider) – Forms the foundation at the bottom.
- Virtual Clusters (Tenants) – Separate virtual environments for different tenants.
- Public Cloud (Global Users) – Represents broader user communities.
- Security Measures – Includes anti-virus, worm containment, intrusion detection, and encryption.

The trusted zones ensure secure isolation of VMs while allowing controlled interactions among tenants, providers, and global communities. This approach strengthens security in private cloud environments.

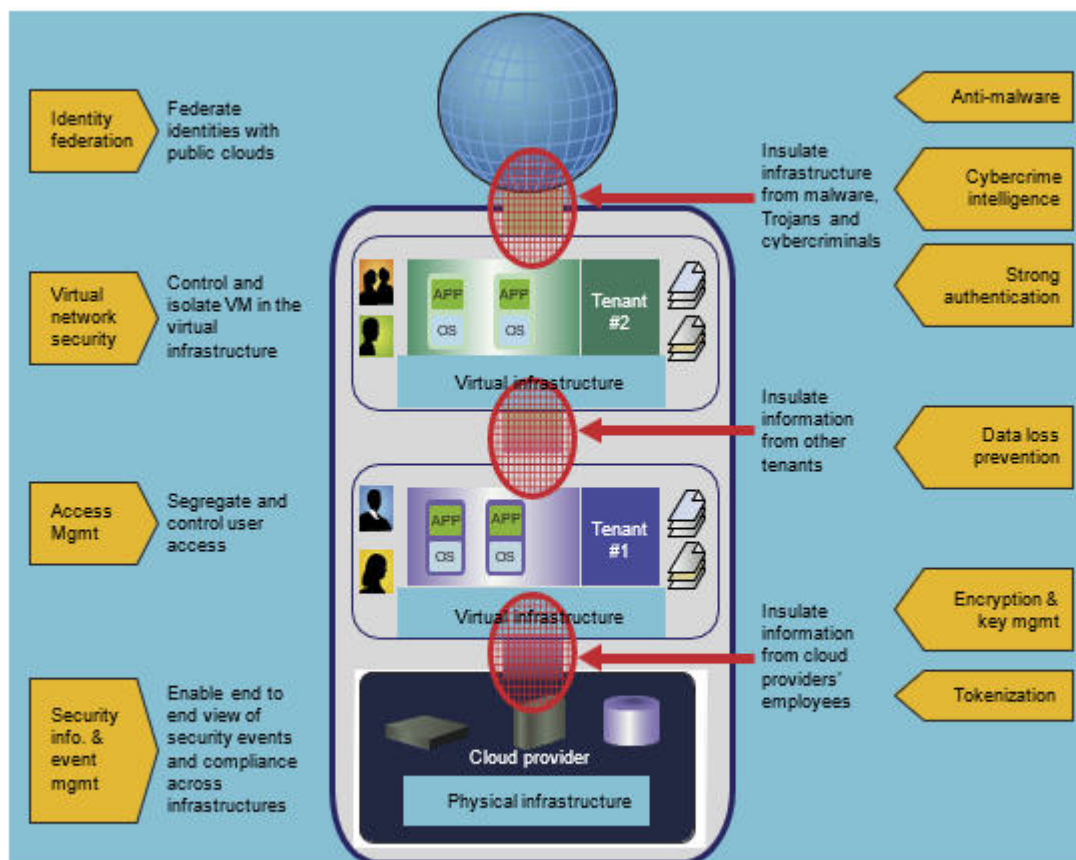


FIGURE 3.30

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