CS415 Module 9 Part A - Virtualization

Athens State University

Outline

Contents

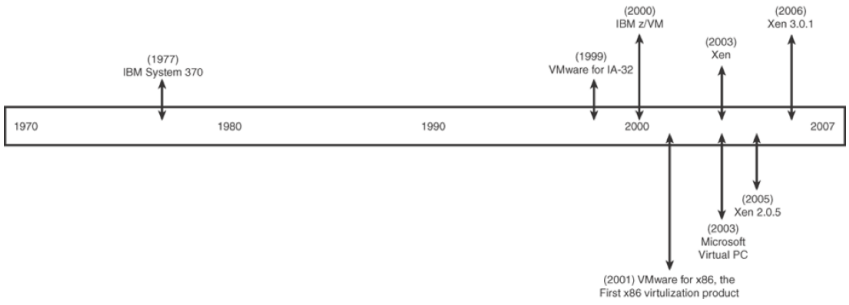
1 Virtualization and Virtual Machines

Virtualization

- **Virtualization**: Separation of a resource or request for a service from the underlying physical delivery of that service
- Consider virtual memory... applications gain access to more memory than physically installed
  - Via background swapping of data to disk storage
- Concept can be extended to other parts of the system

Virtualization is all about abstraction. Much like an operating system abstracts the disk I/O commands from a user through the use of program layers and interfaces, virtualization abstracts the physical hardware from the virtual machines it supports. The virtual machine monitor, or hypervisor, is the software that provides this Abstraction. It acts as a broker, or traffic cop, acting as a proxy for the guests (VMs) as they request and consume resources of the physical host.

History of Virtualization



- IBM S/370
  - CP/CMS, VM/SP, and z/OS
  - Hardware support
- Intel x86 virtualization
  - VMware, XEN, MS Hyper-V
  - Hardware support for virtualization

Terms...

- **Virtual Machine**: Software that provides an operating environment that can host a guest operating system
- **Guest OS**: An OS running in a virtual machine rather than directly upon a physical system
- **Hypervisor**: Software running directly on the hardware but underneath higher-level services that virtualizes the hardware

Design Philosophy

A Virtual Machine (VM) is a software construct that mimics the characteristics of a physical server

A Virtual Machine is a software construct that mimics the characteristics of a physical server. It is configured with some number of processors, some amount of RAM, storage resources, and connectivity through the network ports. Once that VM is created, it can be powered on like a physical server, loaded with an operating system and software solutions, and utilized in the manner of a physical server. Unlike a physical server, this virtual server only sees the resources it has been configured with, not all of the resources of the physical host itself.

This isolation allows a host machine to run many virtual machines, each of them running the same or different copies of an operating system, sharing RAM, storage, and network bandwidth, without problems. An operating system in a virtual machine accesses the resource that is presented to it by the hypervisor. The hypervisor facilitates the translation and I/O from the virtual machine to the physical server devices, and back again to the correct virtual machine. In this way, certain privileged instructions that a "native" operating system would be executing on its hosts hardware are trapped and run by the hypervisor as a proxy for the virtual machine. This creates some performance degradation in the virtualization process, though over time both hardware and software improvements have minimalized this overhead.

Virtual Machine Files

- A configuration file describing the attributes of the virtual machine
  - Defines how many virtual processors (vCPUs) are assigned to the machine, how much RAM is allocated to this machine, I/O devices, and network interfaces
  - Describes what storage the VM can access
- On start-up, additional system administration files are created for logging, memory paging, and other functions.

Virtual machines are made up of files. A typical virtual machine can consist of just a few files. There is a configuration file that describes the attributes of the virtual machine. It contains the server definition, how many virtual processors (vCPUs) are allocated to this virtual machine, how much RAM is allocated, which I/O devices the VM has access to, how many network interface cards (NICs) are in the virtual server, and more. It also describes the storage that the VM can access. Often that storage is presented as virtual disks that exist as additional files in the physical file system. When a virtual machine is powered on, or instantiated, additional files are created for logging, for memory paging, and other functions. That a VM consists of files makes certain functions in a virtual environment much simpler and quicker than in a physical environment. Since the earliest days of computers, backing up data has been a critical function. Since VMs are already files, copying them produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself.

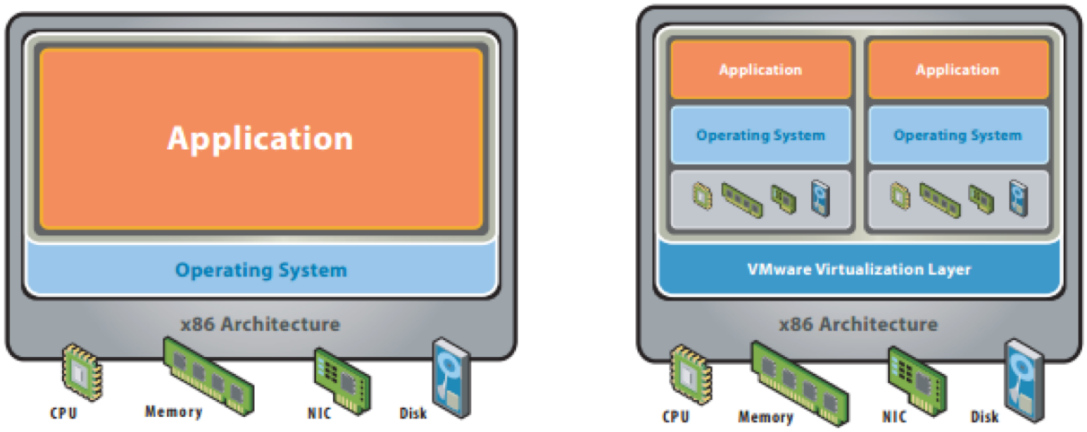
To create a copy of a physical server, additional hardware needs to be acquired, installed, configured, loaded with an operating system, applications, and data, and then patched to the latest revisions, before being turned over to the users. This provisioning can take weeks or even months depending on the processes in places. Since a VM consists of files, by duplicating those files, in a virtual environment there is a perfect copy of the server available in a matter of minutes. There are a few configuration changes to make, server name and IP address to name two, but administrators routinely stand up new virtual machines in minutes or hours, as opposed to months. Another method to rapidly provision new virtual machines is through the use of templates. Templates are virtual machines that cannot be powered on, define the virtual server's configuration, and has all of the operating system and possibly even the application software installed. What hasn't been done are the configuration steps that uniquely identify that virtual server. Creating a new virtual machine from a template consists of providing those unique identifiers and having the provisioning software build a VM from the template and adding in the configuration changes as part of the deployment.

A Different View of "Virtual Machine": Java's JVM

- The Java JVM is an example of a high-level language virtual machine
- Java applications are written to an abstract machine definition
  - The JVM translates this to the appropriate system calls
- Program written in the high-level language will run without concern for the underlying hardware

2 Virtual Infrastructure

Virtual Infrastructure

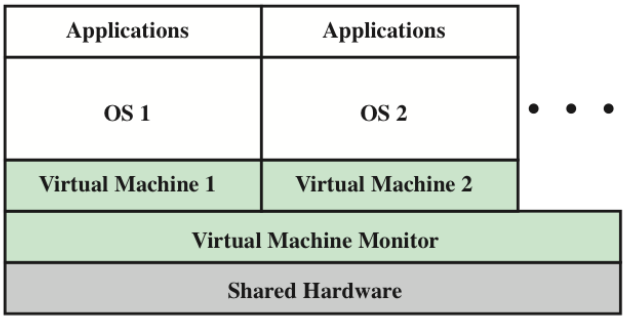


On the left, we have a representation of the relationship between application, operating system, and hardware before virtualization. On the right, we see how the architecture adjusts with the addition of a hypervisor. The hypervisor presents a virtualized view of the hardware to each guest OS. This means the hypervisor has to be able to mediate multiple requests to physical devices from many guest operating systems at the same time.

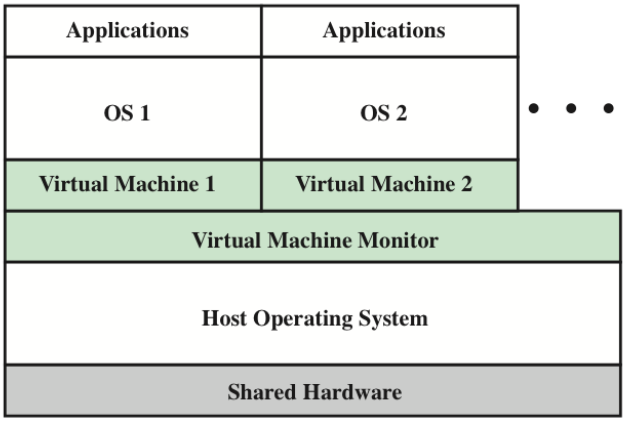
Three Possible Architectures

- *Hosted virtualization*: the virtualization software runs on a host operating system and relies on that OS to talk directly to the underlying hardware
- *Hardware-level virtualization*: A hypervisor lives underneath any other system software and virtualizes devices for all operating system, possibly including the "host" OS
- Para-virtualization

Hypervisor Types



(a) Type 1 VMM



(a) Type 2 VMM

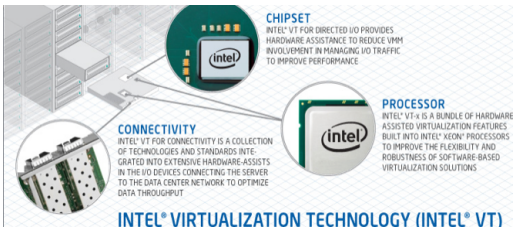
Figure 14.3 Type 1 and Type 2 Virtual Machine Monitors

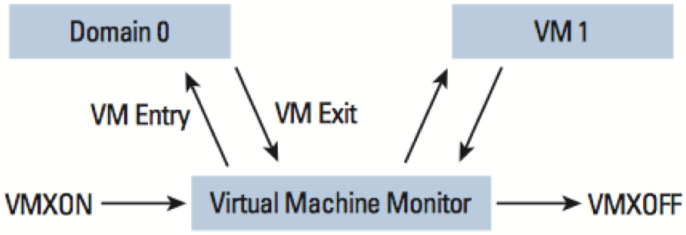
- A Type-1 hypervisor is a thin layer that performs hardware-level device virtualization (VMware ESXi, MS Hyper-V, and Xen)
- A Type-2 hypervisor is an application running on or within a host operating system (VMware Workstation, Parallels, and Oracle Virtual Box)

As mentioned earlier, the hypervisor sits between the hardware and the virtual machines. There are two types of hypervisors, distinguished by whether there is another operating system between the hypervisor and the host. A Type-1 hypervisor is loaded as a thin software layer directly into a physical server, much like an operating system is loaded. Once it is installed and configured, usually just a matter of minutes, the server is then capable of supporting virtual machines as guests. In mature environments, where virtualization hosts are clustered together for increased availability and load balancing, a hypervisor can be staged on a new host, that new host joined to an existing cluster, and VMs can be moved to the new host without any interruption of service. Some examples of Type-1 hypervisors are VMware ESXi, Microsoft Hyper-V, and the various Xen variants. This idea that the hypervisor is loaded onto the "bare metal" of a server is usually a difficult concept for people to understand. They are more comfortable with a solution that works as a traditional application, program code that is loaded on top of a Microsoft Windows or UNIX/Linux operating system environment. This is exactly how a Type-2 hypervisor is deployed. Some examples of Type-2 hypervisors are VMware Workstation and Oracle VM Virtual Box.

There are some important differences between the Type-1 and the Type-2 hypervisors. A Type-1 hypervisor is deployed on a physical host and can directly control the physical resources of that host, whereas a Type-2 hypervisor has an operating system between itself and those resources and relies on the operating system to handle all of the hardware interactions on the hypervisor's behalf. Because of that extra layer, a Type-1 hypervisor has much better performance characteristics than the Type-2 hypervisor. Because a Type-1 hypervisor doesn't compete for resources with an operating system, there are more resources available on the host, and by extension, more virtual machines can be hosted on a virtualization server using a Type-1 hypervisor. Type-1 hypervisors are also considered to be more secure than the Type-2 hypervisors. Virtual machines on a Type-1 hypervisor make resource requests that are handled external to that guest, and they cannot affect other VMs or the hypervisor they are supported by. This is not necessarily true for VMs on a Type-2 hypervisor and a malicious guest could potentially affect more than itself. A Type-1 hypervisor implementation would not require the cost of a host operating system, though a true cost comparison would be a more complicated discussion. Type-2 hypervisors allow a user to take advantage of virtualization without needing to dedicate a server to only that function. Developers who need to run multiple environments as part of their process, in addition to taking advantage of the personal productive workspace that a PC operating system provides, can do both with a Type-2 hypervisor installed as an application on their LINUX or Windows desktop. The virtual machines that are created and used can be migrated or copied from one hypervisor environment to another, reducing deployment time and increasing the accuracy of what is deployed, reducing the time to market of a project.

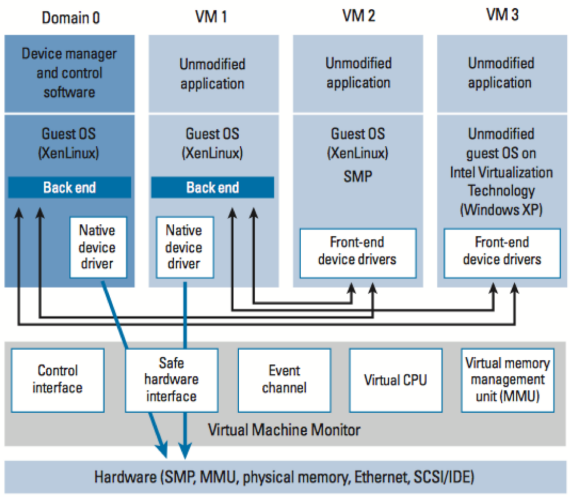
Hardware Support





Processor manufacturers AMD and Intel have added functionality to their processors to enhance performance with hypervisors. AMD-V and Intel's VT-x designate the hardware-assisted virtualization extensions that the hypervisors can take advantage of during processing. Intel processors offer an extra instruction set called Virtual Machine Extensions (VMX). By having some of these instructions as part of the processor, the hypervisors no longer need to maintain these functions as part of their codebase, the code itself can be smaller and more efficient, and the operations they support are much faster as they occur entirely on the processor.

Example: The XEN Architecture



Paravirtualization

- **Paravirtualization**: Software assisted virtualization technique that uses special APIs to link virtual machines with the hypervisor to optimize their performance
- Software is provided as specialized services, loadable kernel modules, and device drivers to allow an OS and hypervisor to work without the overhead of hypervisor transitions
- Supported since 2008 in most general Linux distributions
- Also component of the MSFT Hyper-v hypervisor, with focus on the server versions

As virtualization became more prevalent in corporations, both hardware and software vendors looked for ways to provide even more efficiencies. The operating system in the virtual machine, Linux or Microsoft Windows, has specialized paravirtualization support as part of the kernel, as well as specific paravirtualization

drivers that allow the OS and hypervisor to work together more efficiently with the overhead of the hypervisor translations. This software-assisted virtualization offers optimized virtualization support on servers with or without processors that provide virtualization extensions. Paravirtualization support has been offered as part of many of the general Linux distributions since 2008.

3 Hardware and Device Driver Issues

Two Approaches To Virtualizing A CPU

- Emulate a chip as software and provide access to that resource
  - Method used by QEMU and the Android Emulator in the Android SDK
  - Mixed solutions are available if the OS supports paravirtualization
- Provide segments of processing on the physical processors
  - The virtual processors in the VM is viewed as additional processes by the host
  - This is how most of the virtualization hypervisors offer processor resources to guest OSes.

In a virtual environment, there are two main strategies for providing processor resources. The first is to emulate a chip as software and provide access to that resource. Examples of this method are QEMU and the Android Emulator in the Android SDK. They have the benefit of being easily transportable since they are not platform dependent, but they are not very efficient from a performance standpoint as the emulation process is resource intensive.

The second model doesn't actually virtualize processors but provides segments of processing time on the physical processors (pCPUs) of the virtualization host to the virtual processors of the virtual machines hosted on the physical server. This is how most of the virtualization hypervisors offer processor resources to their guests. When the operating system in a virtual machine passes instructions to the processor, the hypervisor intercepts the request. It then schedules time on the host's physical processors, sends the request for execution, and returns the results to the VM's operating system. This insures the most efficient use of the available processor resources on the physical server. To add some complexity, when multiple VMs are contending for processor, the hypervisor acts as the traffic controller, scheduling processor time for each VM's request as well as directing the requests and data to and from the virtual machines.

Ring 0

- Native OSes manage hardware by acting as the intermediary between application requests and the hardware
- For security reasons, the OS is the only thing that is running in "Ring 0", the most-permissive protection level
- Hypervisors must also run in Ring 0

Native operating systems manage hardware by acting as the intermediary between application code requests and the hardware. As requests for data or processing are made, the operating system passes these to the correct device drivers, through the physical controllers, to the storage or I/O devices, and back again. The operating system is the central router of information and controls access to all of the physical resources of the hardware. One key function of the operating system is to help prevent malicious or accidental system calls from disrupting the applications or the operating system itself. Protection rings describe level of access or privilege inside of a computer system and many operating systems and processor architectures take advantage of this security model. The most trusted layer is often called Ring 0 (zero) and is where the operating system kernel works and can interact directly with hardware. Rings 1 and 2 are where device

drivers execute while user applications run in the least trusted area, Ring 3. In practice, though, Rings 1 and 2 are not often used, simplifying the model to trusted and untrusted execution spaces. Application code cannot directly interact with hardware since it runs in Ring 3 and needs the operating system to execute the code on its behalf in Ring 0. This separation prevents unprivileged code from causing untrusted actions like a system shutdown or an unauthorized access of data from a disk or network connection.

Hypervisors run in Ring 0 controlling hardware access for the virtual machines they host. The operating systems in those virtual machines also believe that they run in Ring 0, and in a way they do, but only on the virtual hardware that is created as part of the virtual machine. In the case of a system shutdown, the operating system on the guest would request a shutdown command in Ring 0. The hypervisor intercepts the request; otherwise the physical server would be shutdown, causing havoc for the hypervisor and any other virtual machines being hosted. Instead, the hypervisor replies to the guest operating system that the shutdown is proceeding as requested, which allows the guest operating system to complete and necessary software shutdown processes.

Memory Management

- The hypervisor manages page sharing for guest OSes running in VMs
  - As result, the VMs are unaware of what's happening in the physical system
- Ballooning
  - To address this problem, the hypervisor includes a *balloon driver* that inflates and presses the guest OS to flush pages to disk
  - Once pages are cleared, the hypervisor deflates the balloon driver and the physical pages can be used for other VMs
- *Memory overcommit*: the capability to allocate more memory than physically exists on a host

Since the hypervisor manages page sharing, the virtual machine operating systems are unaware of what is happening in the physical system. Another strategy for efficient memory use is akin to thin provisioning in storage management. This allows an administrator to allocate more storage to a user than is actually present in the system. The reason is to provide a high water mark that often is never approached. The same can be done with virtual machine memory. We allocate 1GB of memory but that is what is seen by the VM operating system. The hypervisor can use some portion of that allocated memory for another VM by reclaiming older pages that are not being used. The reclamation process is done through ballooning . The hypervisor activates a balloon driver that (virtually) inflates and presses the guest operating system to flush pages to disk. Once the pages are cleared, the balloon driver deflates and the hypervisor can use the physical memory for other VMs. This process happens during times of memory contention. If our 1GB VMs used half of their memory on average, nine VMs would require only 4.5GB with the remainder as a shared pool managed by the hypervisor and some for the hypervisor overhead. Even if we host an additional three 1GB VMs, there is still a shared reserve. This capability to allocate more memory than physical exists on a host is called memory overcommit . It is not uncommon for virtualized environments to have between 1.2 and 1.5 times the memory allocated, and in extreme cases, many times more.

There are additional memory management techniques that provide better resource utilization. In all cases, the operating systems in the virtual machines see and have access to the amount of memory that has been allocated to them. The hypervisor manages that access to the physical memory to insure that all requests are serviced in a timely manner without impacting the virtual machines. In cases where more physical memory is required than is available, the hypervisor will be forced to resort to paging to disk. In multiple host cluster environments, virtual machines can be automatically live migrated to other hosts when certain resources become scarce.

I/O in a Virtual Environment

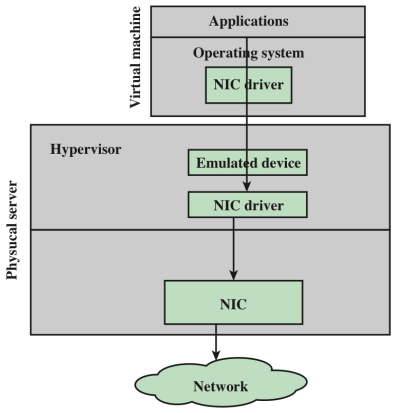


Figure 14.5 I/O in a Virtual Environment

Application performance is often directly linked to the bandwidth that a server has been allocated. Whether it is storage access that has been bottlenecked or constrained traffic to the network, either case will cause an application to be perceived as underperforming. In this way, during the virtualization of workloads, I/O virtualization is a critical item. The architecture of how I/O is managed in a virtual environment is straightforward (Figure 14.5). In the virtual machine, the operating system makes a call to the device driver as it would in a physical server. The device driver then connects with the device; though in the case of the virtual server, the device is an emulated device that is staged and managed by the hypervisor. These emulated devices are usually a common actual device, such as an Intel e1000 network interface card or simple generic SGVA or IDE controllers. This virtual device plugs into the hypervisor's I/O stack that communicates with the device driver that is mapped to a physical device in the host server, translating guest I/O addresses to the physical host I/O addresses. The hypervisor controls and monitors the requests between the virtual machine's device driver, through the I/O stack, out the physical device, and back again, routing the I/O calls to the correct devices on the correct virtual machines. There are some architectural differences between vendors, but the basic model is similar.

4 Case Studies

VMware ESXi

- A commerically available hypervisor that provides a Type-1 baremetal hypervisor to host virtual machines on servers
- VMware was one of the first VM vendors in the Intel x86 marketplace in the late 1990s

VMware ESXi Architecture

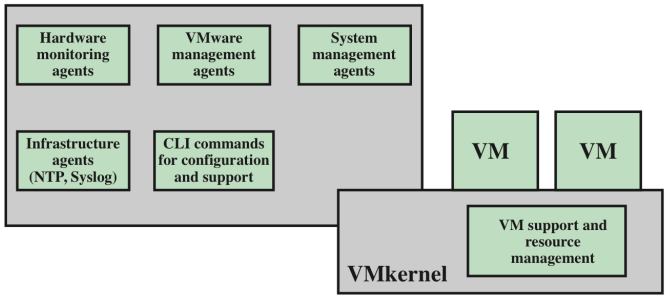


Figure 14.6 ESX

The virtualization kernel (VMkernel) is the core of the hypervisor and performs all of the virtualization functions. In earlier releases of ESX (Figure 14.6), the hypervisor was deployed alongside a Linux installation that served as a management layer. Certain management functions like logging, name services, and often third-party agents for backup or hardware monitoring were installed on this service console. It also made a great place for administrators to run other scripts and programs. The service console had two issues. The first was that it was considerably larger than the hypervisor; a typical install required about 32MB for the hypervisor and about 900MB for the service console. The second was that the Linux-based service console was a well-understood interface and system and was vulnerable to attack by malware or people. VMware then re-architected ESX to be installed and managed without the service console.

VMware ESXi Architecture

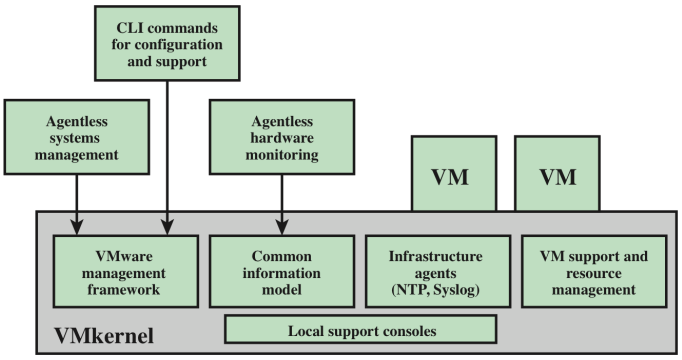


Figure 14.7 ESXi

This new architecture, dubbed ESXi, the "i" for integrated, has all of the management services as part of the VMkernel. This provides a smaller and much more secure package than before. Current versions are in the neighborhood of about 100MB. This small size allows server vendors to deliver hardware with ESXi already available on flash memory in the server. Configuration management, monitoring, and scripting are now all available through command line interface utilities. Third-party agents are also run in the VMkernel after being certified and digitally signed. This allows, for example, a server vendor who provides hardware

monitoring, to include an agent in the VMkernel that can seamlessly return hardware metrics like internal temperature or component statuses to either VMware management tools or other management tools.

XEN Paravirtualization

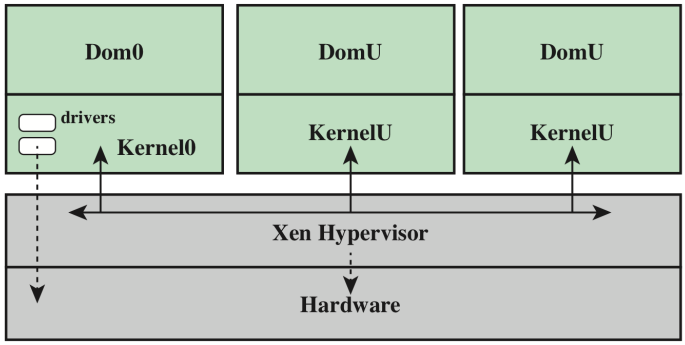


Figure 14.8 Xen

In the early 2000s an effort based in Cambridge University led to the development of the Xen, an open-source hypervisor. Over time, and as the need for virtualization increased, many hypervisor variants have come out of the main Xen branch. Today, in addition to the open-source hypervisor, there are a number of Xen-based commercial hypervisor offering from Citrix, Oracle, and others. Architected differently than the VMware model, Xen requires a dedicated operating system or domain to work with the hypervisor, similar to the VMware service console. This initial domain is known as domain zero (Dom0), runs the Xen tool stack, and as the privileged area, has direct access to the hardware.

Many versions of Linux contain a Xen hypervisor that is capable of creating a virtual environment. Some of these are CentOS, Debian, Fedora, Ubuntu, OracleVM, Red Hat (RHEL), SUSE, and XenServer. Companies that used Xen-based virtualization solutions do so due to the lower (or no) cost of the software or due to their own in-house Linux expertise.

Guests on Xen are unprivileged domains, or sometimes user domains, referred to as DomU. Dom0 provides access to network and storage resources to the guests via BackEnd drivers that communicate with the FrontEnd drivers in DomU. Unless there are pass-through devices configured (usually USB), all of the network and storage I/O is handled through Dom0. Since Dom0 is itself an instance of Linux, if something unexpected happens to it, all of the virtual machines it supports will be affected. Standard operating system maintenance like patching also can potentially affect the overall availability.

Microsoft Hyper-V

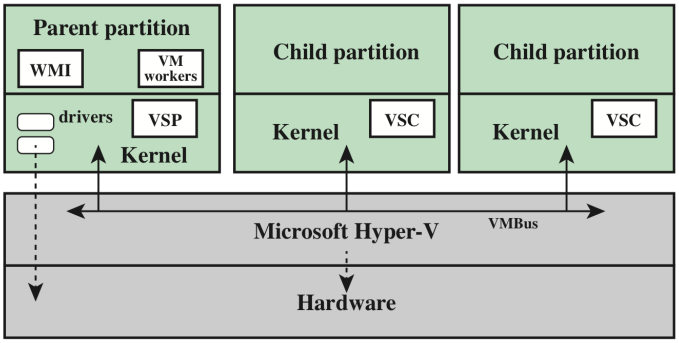


Figure 14.9 Hyper-V

Microsoft has had a number of virtualization technologies, including Virtual Server, a Type-2 hypervisor offering that was acquired in 2005 and is still available today at no cost. Microsoft Hyper-V, a Type-1 hypervisor, was first released in 2008 as part of the Windows Server 2008 Operating System release. Similar to the Xen architecture, Hyper-V has a parent partition that serves as an administrative adjunct to the Type-1 hypervisor. Guest virtual machines are designated as child partitions. The parent partition runs the Windows Server operating system in addition to its functions, such as managing the hypervisor, the guest partitions, and the devices drivers. Similar to the FrontEnd and BackEnd drivers in Xen, the parent partition in Hyper-V uses a Virtualization Service Provider (VSP) to provide device services to the child partitions. The child partitions communicate with the VSPs using a Virtualization Service Client (or Consumer) (VSC) for their I/O needs.

Microsoft Hyper-V has similar availability challenges to Xen due to the operating system needs in the parent partition, the resource contention an extra copy of Windows requires on the server, and the single I/O conduit. From a feature standpoint, Hyper-V is very robust, though not as widely used as ESXi since it is still relatively new to the marketplace. As time passes and new functionality appears, adoption will probably increase