UNIT II CLOUD ENABLING TECHNOLOGIES (10)

Service Oriented Architecture – REST and Systems of Systems – Web Services – Publish Subscribe Model – Basics of Virtualization – Types of Virtualization – Implementation Levels of Virtualization – Virtualization Structures – Tools and Mechanisms – Virtualization of CPU – Memory – I/O Devices –Virtualization Support and Disaster Recovery.

SOA (Service Oriented Architecture) is built on computer engineering approaches that offer an architectural advancement towards enterprise system. It describes a standard method for requesting services from distributed components and after that the results or outcome is managed. The primary focus of this service oriented approach is on the characteristics of service interface and predictable service behavior. Web Services means a set or combination of industry standards collectively labeled as one. SOA provides a translation and management layer within the cloud architecture that removes the barrier for cloud clients obtaining desired services. Multiple networking and messaging protocols can be written using SOA's client and components and can be used to communicate with each other. SOA provides access to reusable Web services over a TCP/IP network, which makes this an important topic to cloud computing going forward.

Benefits of SOA

With high-tech engineering and enterprise point of view, various offers are provided by SOA which proved to be beneficial. These are:

- Language Neutral Integration: Regardless of the developing language used, the system offers and invoke services through a common mechanism. Programming language neutralization is one of the key benefits of SOA's integration approach.
- Component Reuse: Once an organization built an application component, and offered it as a service, the rest of the organization can utilize that service.
- Organizational Agility: SOA defines building blocks of capabilities provided by software and it offers some service(s) that meet some organizational requirement; which can be recombined and integrated rapidly.
- Leveraging Existing System: This is one of the major use of SOA which is to classify elements or functions of existing applications and make them available to the organizations or enterprise.

Key Benefits Along With Risks of SOA

- Dependence on the network
- Provider cost
- Enterprise standards
- Agility

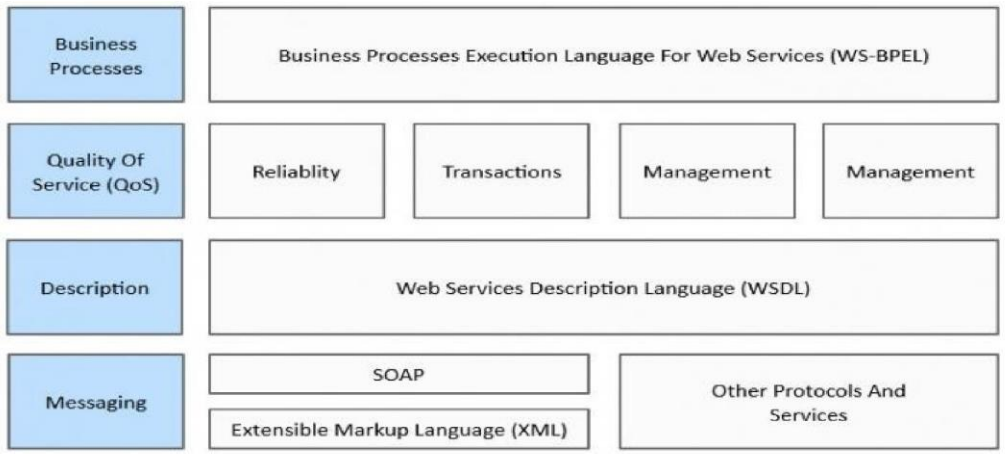
SOA Architecture: SOA architecture is viewed as five horizontal layers.

- Consumer Interface Layer: These are GUI based apps for end users accessing the applications.
- Business Process Layer: These are business-use cases in terms of application.
- Services Layer: These are whole-enterprise, in service inventory.
- Service Component Layer: are used to build the services, such as functional and technical libraries.
- Operational Systems Layer: It contains the data model.

SOA Governance

It is a notable point to differentiate between It governance and SOA governance. IT governance focuses on managing business services whereas SOA governance focuses on managing Business services. Furthermore in service oriented organization, everything should be characterized as a service in an organization. The cost that governance put forward becomes clear when we consider the amount of risk that it eliminates with the good understanding of service, organizational data and processes in order to choose approaches and processes for policies for monitoring and generate performance impact.

SOA Architecture and Protocols

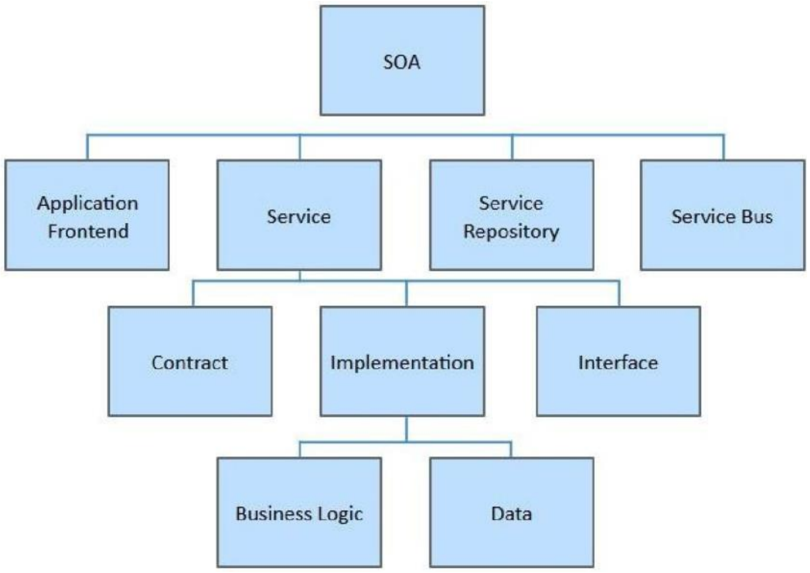

Business Processes
Business Processes Execution Language For Web Services (WS-BPEL)
Quality Of Service (QoS)
Reliablity
Transactions
Management
Management
Description
Web Services Description Language (WSDL)
Messaging
SOAP
Extensible Markup Language (XML)
Other Protocols And Services

Here lies the protocol stack of SOA showing each protocol along with their relationship among each protocol. These components are often programmed to comply with SCA (Service Component Architecture), a language that has broader but not universal industry support. These components are written in BPEL (Business Process Execution Languages), Java, C#, XML etc and can apply to C++ or FORTRAN or other modern multi-purpose languages such as Python, PP or Ruby. With this, SOA has extended the life of many all-time famous applications.

Security in SOA

With the vast use of cloud technology and its on-demand applications, there is a need for well -defined security policies and access control. With the betterment of these issues, the success of SOA architecture will increase. Actions can be taken to ensure security and lessen the risks when dealing with SOE (Service Oriented Environment). We can make policies that will influence the patterns of development and the way services are used. Moreover, the system must be set-up in order to exploit the advantages of public cloud with resilience. Users must include safety practices and carefully evaluate the clauses in these respects.

Elements Of SOA:



Though SOA enjoyed lots achieving the success in the past, the introduction to cloud technology with SOA, renewed the value of SOA.

Representational state transfer (REST)

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.

"Web resources" were first defined on the World Wide Web as documents or files identified by their URLs. However, today they have a much more generic and abstract definition that encompasses everything, entity, or action that can be identified, named, addressed, handled, or performed, in any way whatsoever, on the Web. In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format. The response can confirm that some alteration has been made to the resource state, and the response can provide hypertext links to other related resources.

When HTTP is used, as is most common, the operations (HTTP methods) available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE.

By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running.

The term *representational state transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding's dissertation explained the REST principles that were known as the "HTTP object model" beginning in 1994, and were used in designing the HTTP 1.1 and Uniform Resource Identifiers (URI) standards. The term is intended to evoke an image of how a well-designed Web application behaves: it is a network of Web resources (a virtual state- machine) where the user progresses through the application by selecting resource identifiers such as http://www.example.com/articles/21 and resource operations such as GET or POST (application state transitions), resulting in the next resource's representation (the next application state) being transferred to the end user for their use.

Web Service

Different books and different organizations provide different definitions to Web Services. Some of them are listed here.

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.
- Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
- Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that −

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components −

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows −

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.
- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

Publish–Subscribe

In software architecture, **publish–subscribe** is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.
Publish–subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. Most messaging systems support both the pub/sub and message queue models in their API, e.g. Java Message Service (JMS).
This pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data.

Message filtering

In the publish-subscribe model, subscribers typically receive only a subset of the total messages published. The process of selecting messages for reception and processing is called *filtering*. There are two common forms of filtering: topic-based and content-based.

In a **topic-based** system, messages are published to "topics" or named logical channels. Subscribers in a topic-based system will receive all messages published to the topics to which they subscribe. The publisher is responsible for defining the topics to which subscribers can subscribe.

In a **content-based** system, messages are only delivered to a subscriber if the attributes or content of those messages matches constraints defined by the subscriber. The subscriber is responsible for classifying the messages.

Some systems support a **hybrid** of the two; publishers post messages to a topic while subscribers register content-based subscriptions to one or more topics.

Topologies

In many pub/sub systems, publishers post messages to an intermediary message broker or event bus, and subscribers register subscriptions with that broker, letting the broker perform the filtering. The broker normally performs a store and forward function to route messages from publishers to subscribers. In addition, the broker may prioritize messages in a queue before routing. Subscribers may register for specific messages at build time, initialization time or runtime. In GUI systems, subscribers can be coded to handle user commands (e.g., click of a button), which corresponds to build time registration. Some frameworks and software products use XML configuration files to register subscribers. These configuration files are read at initialization time. The most sophisticated alternative is when subscribers can be added or removed at runtime. This latter approach is used, for example, in database triggers, mailing lists, and RSS.

The Data Distribution Service (DDS) middleware does not use a broker in the middle. Instead, each publisher and subscriber in the pub/sub system shares meta-data about each other via IP multicast. The publisher and the subscribers cache this information locally and route messages based on the discovery of each other in the shared cognizance. In effect, brokerless architectures require publish/subscribe system to construct an overlay network which allows efficient decentralized routing from publishers to subscribers. It was shown by Jon Kleinberg that efficient decentralised routing requires Navigable Small-World topologies. Such Small-World topologies are usually implemented by decentralized or federated publish/subscribe systems.[1] Locality-aware publish/subscribe systems[2] construct Small-World topologies that route subscriptions through short-distance and low-cost links thereby reducing subscription delivery times.

Advantages

Loose coupling

Publishers are loosely coupled to subscribers, and need not even know of their existence. With the topic being the focus, publishers and subscribers are allowed to remain ignorant of system topology. Each can continue to operate as per normal independently of the other. In the traditional tightly coupled client–server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running. Many pub/sub systems decouple not only the locations of the publishers and subscribers but also decouple them temporally. A common strategy used by middleware analysts with such pub/sub systems is to take down a publisher to allow the subscriber to work through the backlog (a form of bandwidth throttling).

Scalability

Pub/sub provides the opportunity for better scalability than traditional client-server, through parallel operation, message caching, tree-based or network-based routing, etc. However, in certain types of tightly coupled, high-volume enterprise environments, as systems scale up to become data centers with thousands of servers sharing the pub/sub infrastructure, current vendor systems often lose this benefit; scalability for pub/sub products under high load in these contexts is a research challenge.

Outside of the enterprise environment, on the other hand, the pub/sub paradigm has proven its scalability to volumes far beyond those of a single data center, providing Internet-wide distributed messaging through web syndication protocols such as RSS and Atom. These syndication protocols accept higher latency and lack of delivery guarantees in exchange for the ability for even a low-end web server to syndicate messages to (potentially) millions of separate subscriber nodes.

Disadvantages

The most serious problems with pub/sub systems are a side-effect of their main advantage: the decoupling of publisher from subscriber.

Message delivery issues

A pub/sub system must be designed carefully to be able to provide stronger system properties that a particular application might require, such as assured delivery.

- The broker in a pub/sub system may be designed to deliver messages for a specified time, but then stop attempting delivery, whether or not it has received confirmation of successful receipt of the message by all subscribers. A pub/sub system designed in this way cannot guarantee delivery of messages to any applications that might require such assured delivery. Tighter coupling of the designs of such a publisher and subscriber pair must be enforced outside of the pub/sub architecture to accomplish such assured delivery (e.g. by requiring the subscriber to publish receipt messages).
- A publisher in a pub/sub system may assume that a subscriber is listening, when in fact it is not. A factory may utilize a pub/sub system where equipment can publish problems or failures to a subscriber that displays and logs those problems. If the logger fails (crashes), equipment problem publishers won't necessarily receive notice of the logger failure, and error messages will not be displayed or recorded by any equipment on the pub/sub system. This is also a design challenge for alternative messaging architectures, such as a client/server system. In a client/server system, when an error logger fails, the system will receive an indication of the error logger (server) failure. However, the client/server system will have to deal with that failure by having redundant logging servers online, or by dynamically spawning fallback logging servers. This adds complexity to the client and server designs, as well as to the client/server architecture as a whole. However, in a pub/sub system, redundant logging subscribers that are exact duplicates of the existing logger can be added to the system to increase logging reliability without any impact to any other equipment on the system. In a pub/sub system, the feature of

assured error message logging can be added incrementally, subsequent to implementing the basic functionality of equipment problem message logging.

Virtualization Techniques in Cloud Computing

Virtualization is the creation of virtual servers, infrastructures, devices and computing resources. Virtualization changes the hardware-software relations and is one of the foundational elements of cloud computing technology that helps utilize the capabilities of cloud computing to the full. Virtualization techniques allow companies to turn virtual their networks, storage, servers, data, desktops and applications.

The Basics of Virtualization

A great example of how virtualization works in your daily life is the separation of your hard drive into different parts. While you may have only one hard drive, your system sees it as two, three or more different and separate segments. Similarly, this technology has been used for a long time. It started as the ability to run multiple operating systems on one hardware set and now it is a vital part of testing and cloud-based computing.

A technology called the Virtual Machine Monitor — also called virtual manager — encapsulates the very basics of virtualization in cloud computing. It is used to separate the physical hardware from its emulated parts. This often includes the CPU's memory, I/O and network traffic. A secondary operating system that is usually interacting with the hardware is now a software emulation of that hardware, and often the guest operating system has no idea it's on the virtualized hardware. Despite the fact that the performance of the virtual system is not equal to the functioning of the ―true hardware‖ operating system, the technology still works because most secondary OSs and applications don't need the full use of the underlying hardware. This allows for greater flexibility, control and isolation by removing the dependency on a given hardware platform.

The layer of software that enables this abstraction is called ―hypervisor‖. A study in the *International Journal of Scientific & Technology Research* defines it as ―a software layer that can monitor and virtualize the resources of a host machine conferring to the user requirements.‖ The most common hypervisor is referred to as Type 1. By talking to the hardware directly, it virtualizes the hardware platform that makes it available to be used by virtual machines. There's also a Type 2 hypervisor, which requires an operating system. Most often, you can find it being used in software testing and laboratory research.

Virtualization vs. Cloud Computing

Unlike virtualization, cloud computing refers to the service that results from that change. It describes the delivery of shared computing resources, SaaS and on-demand services through the Internet. Most of the confusion occurs because virtualization and cloud computing work together to provide different types of services, as is the case with private clouds.

The cloud often includes virtualization products as a part of their service package. The difference is that a true cloud provides the self-service feature, elasticity, automated management, scalability and pay-as-you-go service that is not inherent to the technology.

Types of Virtualization in Cloud Computing

Here are six methodologies to look at when talking about virtualization techniques in cloud computing:

Network Virtualization

Network virtualization in cloud computing is a method of combining the available resources in a network by splitting up the available bandwidth into different channels, each being separate and distinguished. They can be either assigned to a particular server or device or stay unassigned completely — all in real time. The idea is that the technology disguises the true complexity of the network by separating it into parts that are easy to manage, much like your segmented hard drive makes it easier for you to manage files.

Storage Virtualization

Using this technique gives the user an ability to pool the hardware storage space from several interconnected storage devices into a simulated single storage device that is managed from one single command console. This storage technique is often used in storage area networks. Storage manipulation in the cloud is mostly used for backup, archiving, and recovering of data by hiding the real and physical complex storage architecture. Administrators can implement it with software applications or by employing hardware and software hybrid appliances.

Server Virtualization

This technique is the masking of server resources. It simulates physical servers by changing their identity, numbers, processors and operating systems. This spares the user from continuously managing complex server resources. It also makes a lot of resources available for sharing and utilizing, while maintaining the capacity to expand them when needed.

Data Virtualization

This kind of cloud computing virtualization technique is abstracting the technical details usually used in data management, such as location, performance or format, in favor of broader access and more resiliency that are directly related to business needs.

Desktop Virtualizing

As compared to other types of virtualization in cloud computing, this model enables you to emulate a workstation load, rather than a server. This allows the user to access the desktop remotely. Since the workstation is essentially running in a data center server, access to it can be both more secure and portable.

Application Virtualization

Software virtualization in cloud computing abstracts the application layer, separating it from the operating system. This way the application can run in an encapsulated form without being dependant upon the operating system underneath. In addition to providing a level of isolation, an application created for one OS can run on a completely different operating system.

IMPLEMENTATION LEVELS OF VIRTUALIZATION

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The idea of VMs can be dated back to the 1960s [53]. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility. Hardware resources (CPU, memory, I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers. This virtualization technology has been revitalized as the demand for distributed and cloud computing increased sharply in recent years.

The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks, and storage. In this chapter we will discuss VMs and their applications for building distributed systems. According to a 2009 Gartner Report, virtualization was the top strategic technology poised to change the computer industry. With sufficient storage, any computer platform can be installed in another host computer, even if they use processors with different instruction sets and run with distinct operating systems on the same hardware.

1. Levels of Virtualization Implementation

A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure 3.1(a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer as shown in Figure 3.1(b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM) [54]. The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels, as we will discuss shortly. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level

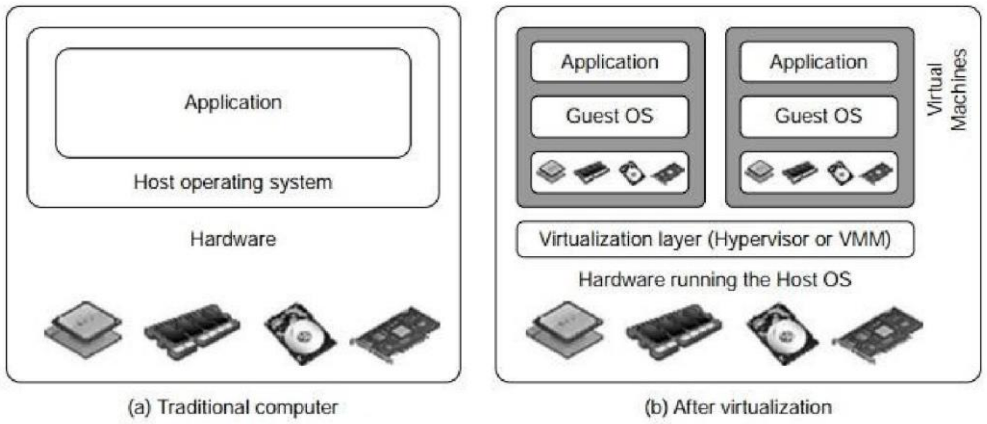
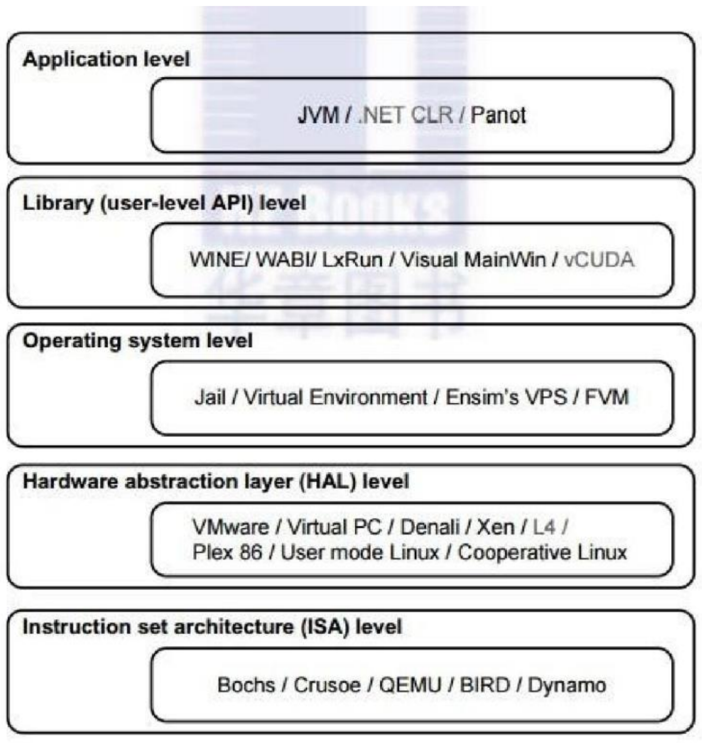


FIGURE 3.1

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



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

FIGURE 3.2

Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture Level

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary code writ- ten for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V- ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level

Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently. The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86- based machines to run Linux or other guest OS applications. We will discuss hardware virtualization approaches in more detail in Section 3.3.

Operating System Level

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hard-ware and software in data centers. The containers behave like real servers. OS- level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server. OS-level virtualization is depicted in Section 3.1.3.

Library Support Level

Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS. Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization. Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration. This approach is detailed in Section 3.1.4.

User-Application Level

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL) VMs. In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM.

Other forms of application-level virtualization are known as application isolation, application sandboxing, or application streaming. The process involves wrapping the application in a layer that is isolated from the host OS and other applications. The result is an application that is much easier to distribute and remove from user workstations. An example is the LANDesk application virtuali-zation platform which deploys software applications as self- contained, executable files in an isolated environment without requiring installation, system modifications, or elevated security privileges.

Relative Merits of Different Approaches

Table 3.1 compares the relative merits of implementing virtualization at various levels. The column headings correspond to four technical merits.

-Higher Performance

-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.
Each row corresponds to a particular level of virtualization.

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.

Virtualization of CPU, Memory, and I/O Devices

1. Hardware Support for Virtualization 2. CPU Virtualization 3. Memory Virtualization 4. I/O Virtualization 5. Virtualization in Multi-Core Processors

VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, mode switching is completed by hardware. For the x86 architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

1. Hardware Support for Virtualization

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Example 3.4 discusses Intel's hardware support approach. At the time of this writing, many hardware virtualization products were available. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host- based virtualization. Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts. Actually, Xen modifies Linux as the lowest and most privileged layer, or a hypervisor.

One or more guest OS can run on top of the hypervisor. KVM (Kernel-based Virtual Machine) is a Linux kernel virtualization infrastructure. KVM can support hardware-assisted virtualization and paravirtualization by using the Intel VT-x or AMD-v and VirtIO framework, respectively. The VirtIO framework includes a paravirtual Ethernet card, a disk I/O controller, a balloon device for adjusting
//;//;?:"?"::'''''''''''
"

}
}

}
}}

}’’
“

guest memory usage, and a VGA graphics interface using VMware drivers. Example 3.4 Hardware Support for Virtualization in the Intel x86 Processor

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

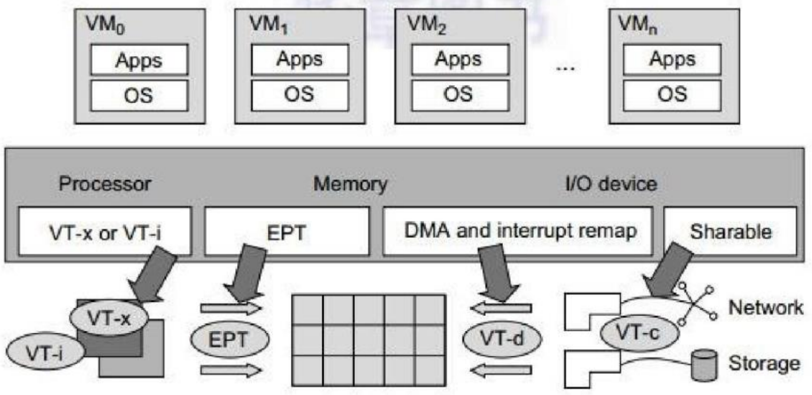


FIGURE 3.10

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

2. CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: privileged instructions, control-sensitive instructions, and behavior-sensitive instructions. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control- sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior-sensitive instructions of a VM are exe- cuted, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions. When these instructions execute in virtualization, they cannot be trapped in the VMM.

On a native UNIX-like system, a system call triggers the 80h interrupt and passes control to the OS kernel. The interrupt handler in the kernel is then invoked to process the system call. On a para-virtualization system such as Xen, a system call in the guest OS first triggers the 80h interrupt nor-mally. Almost at the same time, the 82h interrupt in the hypervisor is triggered. Incidentally, control is passed on to the hypervisor as well. When the hypervisor completes its task for the guest OS system call, it passes control back to the guest OS kernel. Certainly, the guest OS kernel may also invoke the hypercall while it's running. Although paravirtualization of a CPU lets unmodified applications run in the VM, it causes a small performance penalty.

Hardware-Assisted CPU Virtualization

This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification. Example 3.5 Intel Hardware-Assisted CPU Virtualization

Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtuali-zation of x86 processors is detailed in the following sections. Intel's VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

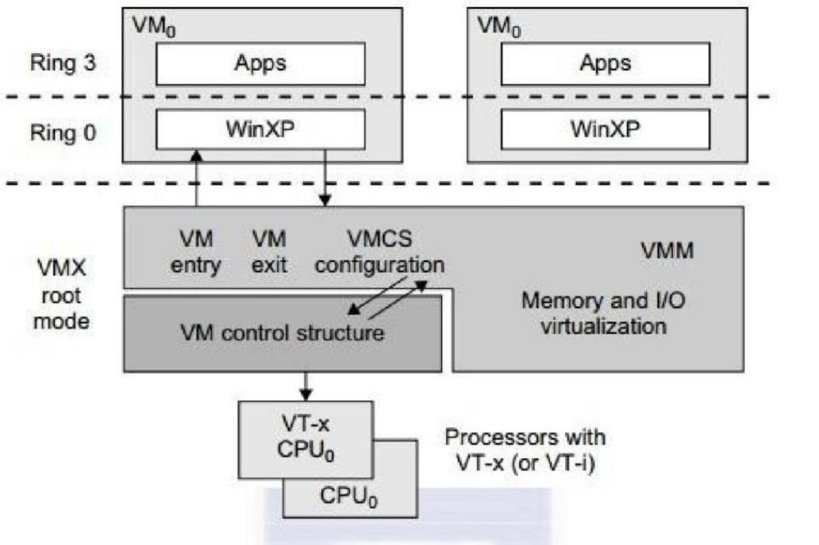


FIGURE 3.11

Intel hardware-assisted CPU virtualization.

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

3. Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.

That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of

VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.

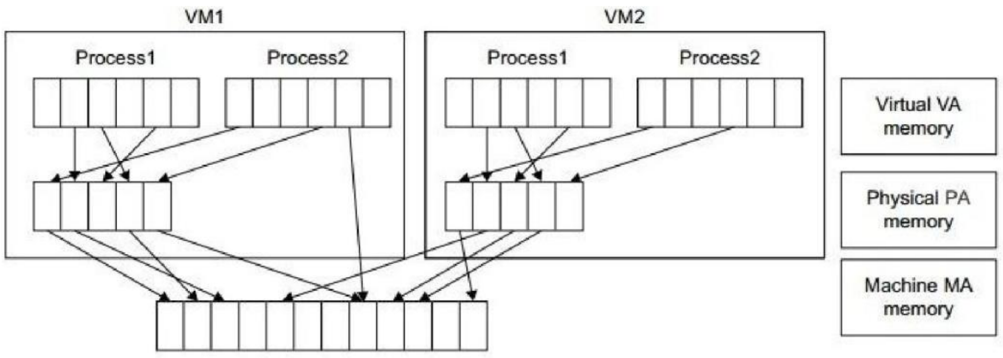


FIGURE 3.12

Two-level memory mapping procedure.

Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded. Consequently, the performance overhead and cost of memory will be very high.

VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging. Example 3.6 Extended Page Table by Intel for Memory Virtualization. Since the efficiency of the software shadow page table technique was too low, Intel developed a hardware-based EPT technique to improve it, as illustrated in Figure 3.13. In addition, Intel offers a Virtual Processor ID (VPID) to improve use of the TLB. Therefore, the performance of memory virtualization is greatly improved. In Figure 3.13, the page tables of the guest OS and EPT are all four-level.

When a virtual address needs to be translated, the CPU will first look for the L4 page table pointed to by Guest CR3. Since the address in Guest CR3 is a physical address in the guest OS, the CPU needs to convert the Guest CR3 GPA to the host physical address (HPA) using EPT. In this procedure, the CPU will check the EPT TLB to see if the translation is there. If there is no required translation in the EPT TLB, the CPU will look for it in the EPT. If the CPU cannot find the translation in the EPT, an EPT violation exception will be raised.When the GPA of the L4 page table is obtained, the CPU will calculate the GPA of the L3 page table by using the GVA and the content of the L4 page table. If the entry corresponding to the GVA in the L4 page table is a page fault, the CPU will generate a page fault interrupt and will let the guest OS kernel handle the interrupt. When the PGA of the L3 page table is obtained, the CPU will look for the EPT to get the HPA of the L3 page

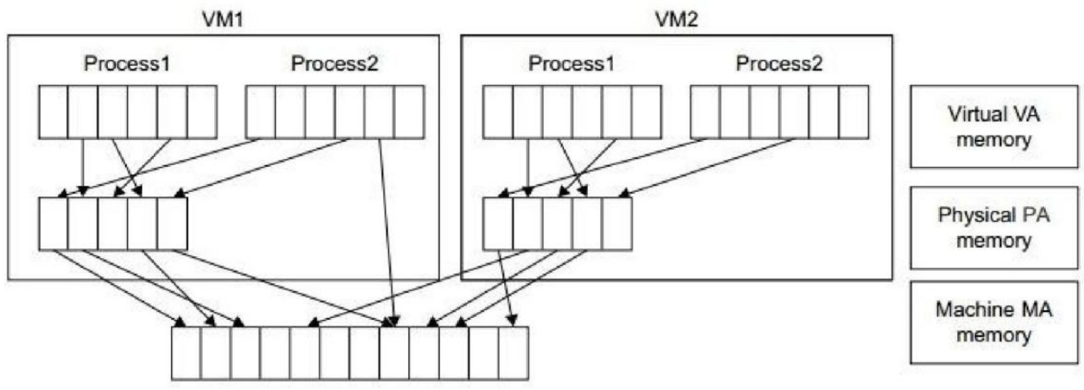


FIGURE 3.12
Two-level memory mapping procedure.

table, as described earlier.

To get the HPA corresponding to a GVA, the CPU needs to look for the EPT five times, and each time, the memory needs to be accessed four times. There-fore, there are 20 memory accesses in the worst case, which is still very slow. To overcome this short-coming, Intel increased the size of the EPT TLB to decrease the number of memory accesses.

4. I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. At the time of this writing, there are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O. Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real- world devices.

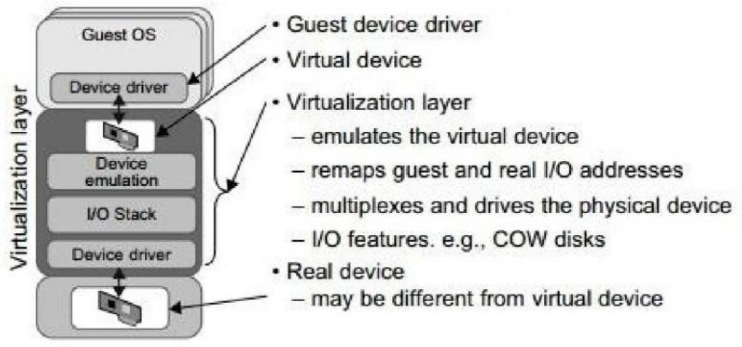

Guest OS
Device driver
Device emulation
I/O Stack
Device driver
Virtualization layer
• Guest device driver
• Virtual device
• Virtualization layer
– emulates the virtual device
– remaps guest and real I/O addresses
– multiplexes and drives the physical device
– I/O features. e.g., COW disks
• Real device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

FIGURE 3.14

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

All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices. The full device emulation approach is shown in Figure 3.14.

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates [10,15]. The para- virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend dri-ver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices. For example, when a physical device is reclaimed (required by workload migration) for later reassign-ment, it may have been set to an arbitrary state (e.g., DMA to some arbitrary memory locations) that can function incorrectly or even crash the whole system. Since software- based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical. Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or –virtualization-aware‖ guest OSes.

Another way to help I/O virtualization is via self-virtualized I/O (SV-IO) [47]. The key idea of SV-IO is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. It provides virtual devices and an associated access API to VMs and a management API to the VMM. SV-IO defines one virtual interface (VIF) for every kind of virtua-lized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two mes-sage queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

Example 3.7 VMware Workstation for I/O Virtualization

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

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

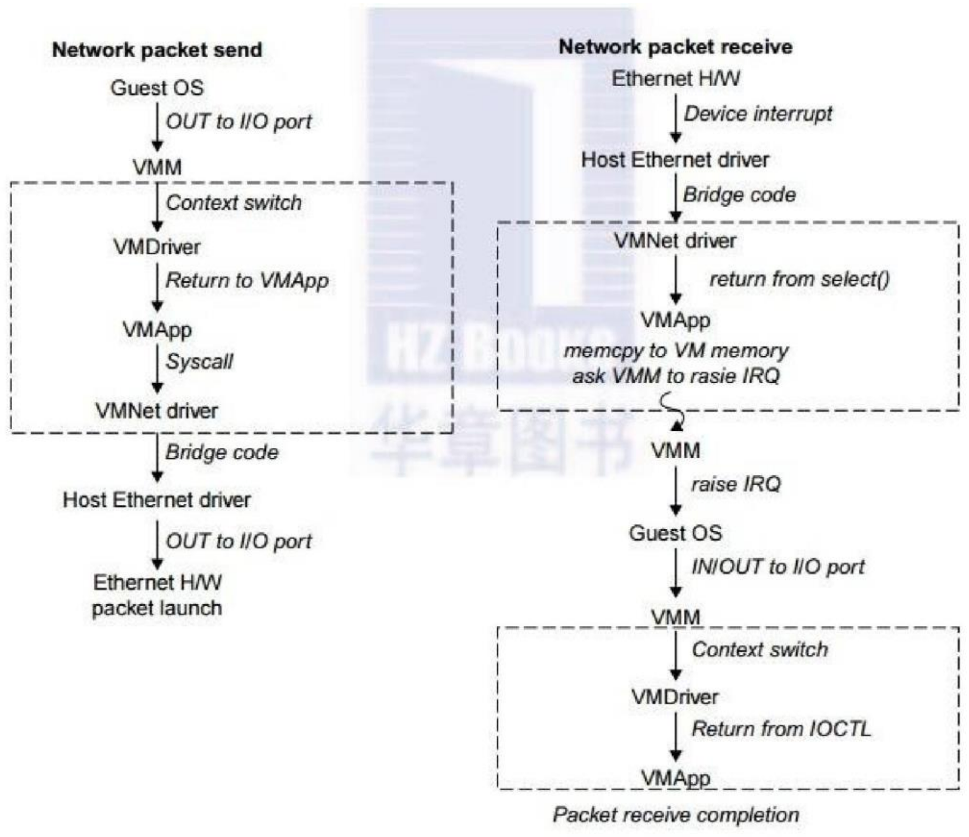


FIGURE 3.15

Functional blocks involved in sending and receiving network packets.

5. Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors [33,66].

Physical versus Virtual Processor Cores

Wells, et al. [74] proposed a multicore virtualization method to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

Virtual Hierarchy

The emerging many-core chip multiprocessors (CMPs) provides a new computing landscape. Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. This idea was originally suggested by Marty and Hill [39]. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

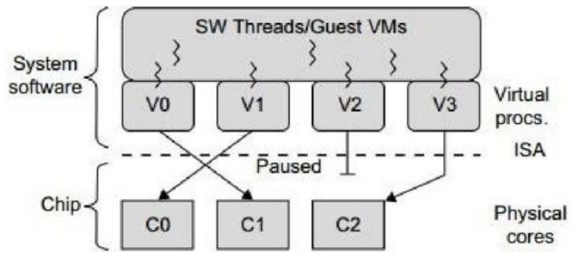


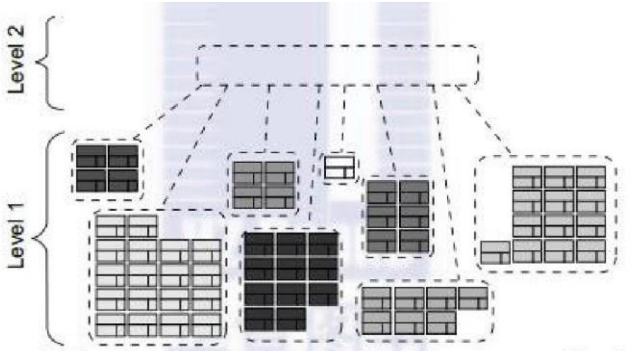
FIGURE 3.16

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

Today’s many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads [39]. The hierarchy’s first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also pro-vide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.The idea is illustrated in

Figure 3.17(a). Space sharing is applied to assign three workloads to three clusters of virtual cores: namely VM0 and VM3 for database workload, VM1 and VM2 for web server workload, and VM4–VM7 for middleware workload. The basic assumption is that each workload runs in its own VM. However, space sharing applies equally within a single operating system. Statically distributing the directory among tiles can do much better, provided operating sys-tems or hypervisors carefully map virtual pages to physical frames. Marty and Hill suggested a two-level virtual coherence and caching hierarchy that harmonizes with the assignment of tiles to the virtual clusters of VMs.

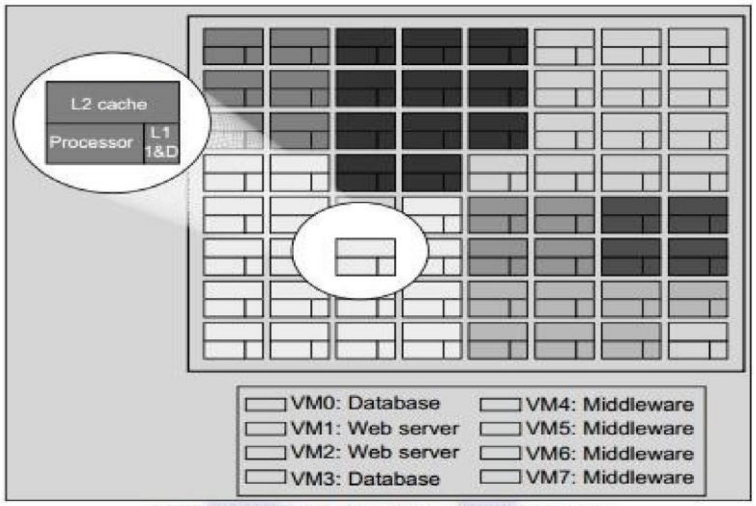
Figure 3.17(b) illustrates a logical view of such a virtual cluster hierarchy in two levels. Each VM operates in a isolated fashion at the first level.



(b) Multiple virtual clusters assigned to various workloads

FIGURE 3.17

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



(a) Mapping of VMs into adjacent cores

This will minimize both miss access time and performance interference with other workloads or VMs. Moreover, the shared resources of cache capacity, inter-connect links, and miss handling are mostly isolated between VMs. The second level maintains a globally shared memory. This facilitates dynamically repartitioning resources without costly cache flushes. Furthermore, maintaining globally shared memory minimizes changes to existing system software and allows virtualization features such as content-based page sharing. A virtual hierarchy adapts to space-shared workloads like multiprogramming and server consolidation. Figure 3.17 shows a case study focused on consolidated server workloads in a tiled architecture. This many-core mapping scheme can also optimize for space-shared multiprogrammed workloads in a single-OS environment.