**Page Numbers (Slides)**

**Chapter 5 – Cloud Resource Virtualization**



**Chapter 6**

## UNIT-III

1. Identify the conditions for efficient virtualization. Distinguish with diagrams the two basic approaches to Processor Virtualization. **140**
2. Illustrate with necessary diagrams the Xen zero-copy semantics for data transfer using I/O rings. **147**
3. Describe briefly with diagrams Traditional, Hybrid, and Hosted VMs. **137**
4. Discuss briefly the problems faced in the virtualization of the x86 Architecture. **142**
5. Identify with a diagram the layers, and the interfaces between layers in a computer system.**134**
6. Discuss how issues in paravirtualization have been resolved and implemented on the x86-64 Itanium processor through the vBlades VMM project. **152**
7. Compare with a diagram the conventional compilation process of an HLL program and the compilation process for a virtual machine environment that produces portable code. **135**
8. Discuss briefly the Network Optimization Strategies used by Xen. Give the Optimized network architecture of Xen. **149**

## UNIT-IV

1. Illustrate with an example and its diagram how coordination between autonomic system managers can be utilized for cloud resource management. **172**
2. Describe using schematics the ASCA combinatorial auction algorithm. **180**
3. Describe briefly policies for Cloud Resource Management (CRM). **164**
4. List the conditions of the pricing and allocation algorithms for the combinatorial auctions of cloud resources.
5. Describe the two-level control architecture for automatic resource management with a diagram. **170**
6. Describe with timing diagrams the Optimal Partitioning Rule (OPR) and Equal Partitioning Rule (EPR) for partitioning workloads in cloud scheduling.

# UNIT 3 Answers

**1. Identify the conditions for efficient virtualization. Distinguish with diagrams the two basic approaches to Processor Virtualization. 140**

The sufficient conditions for a computer architecture to support virtualization and allow a VMM to operate efficiently :

• A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when the program runs directly on an equivalent machine.

• The VMM should be in complete control of the virtualized resources.

• A statistically significant fraction of machine instructions must be executed without the intervention of the VMM.

There are two basic approaches to processor virtualization:

**1)Full virtualization** requires a virtualizable architecture; the hardware is fully exposed to the guest OS, which runs unchanged, and this ensures that this direct execution mode is efficient.

**2) paravirtualization** is done because some architectures such as x86 are not easily virtualizable.Paravirtualization demands that the guest OS be modified to run under the VMM; In paravirtualization each virtual machine runs on a slightly modified copy of the actual hardware.

Examples: **VMware** VMMs are examples of **full virtualization**. **Xen** and **Denali** are based on **paravirtualization**;

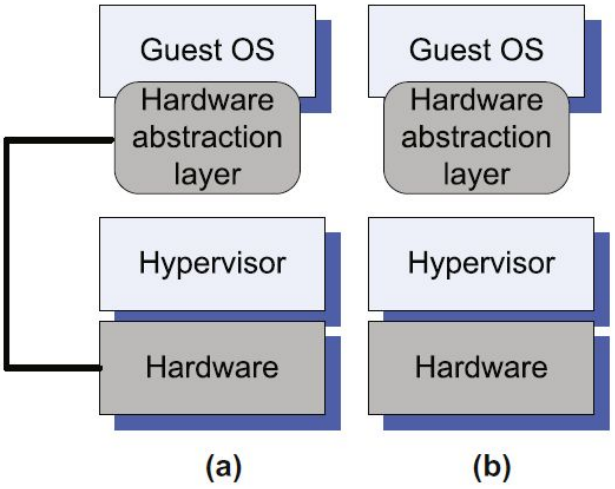
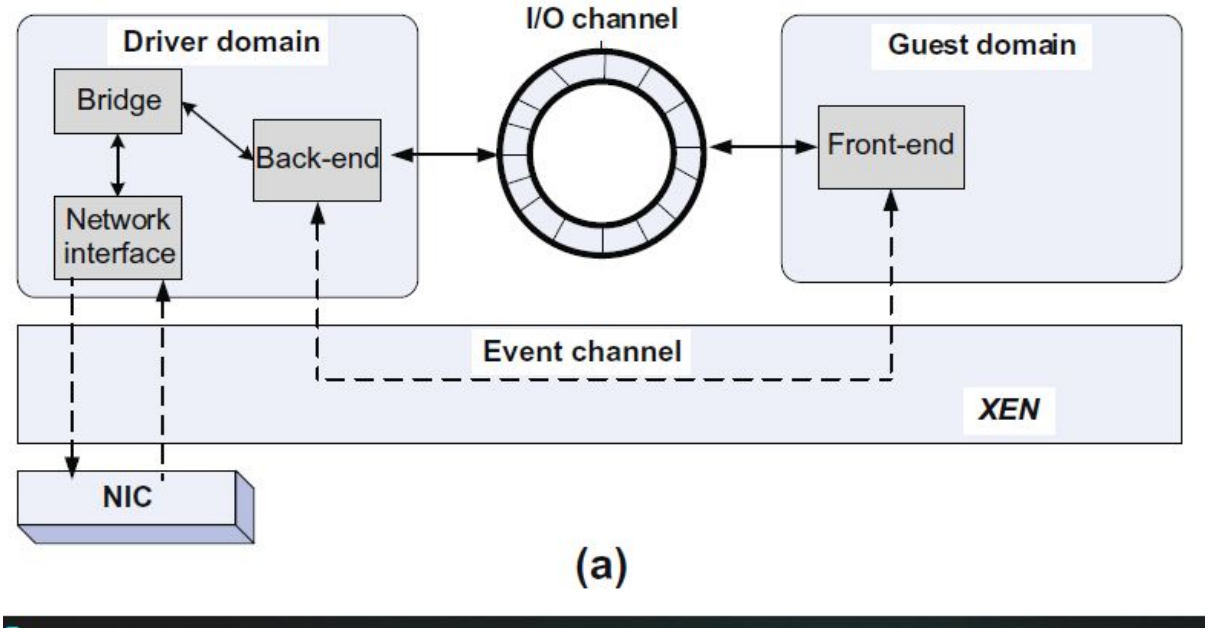



**FIGURE 5.4**

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

**2. Illustrate with necessary diagrams the Xen zero-copy semantics for data transfer using I/O rings. 147**



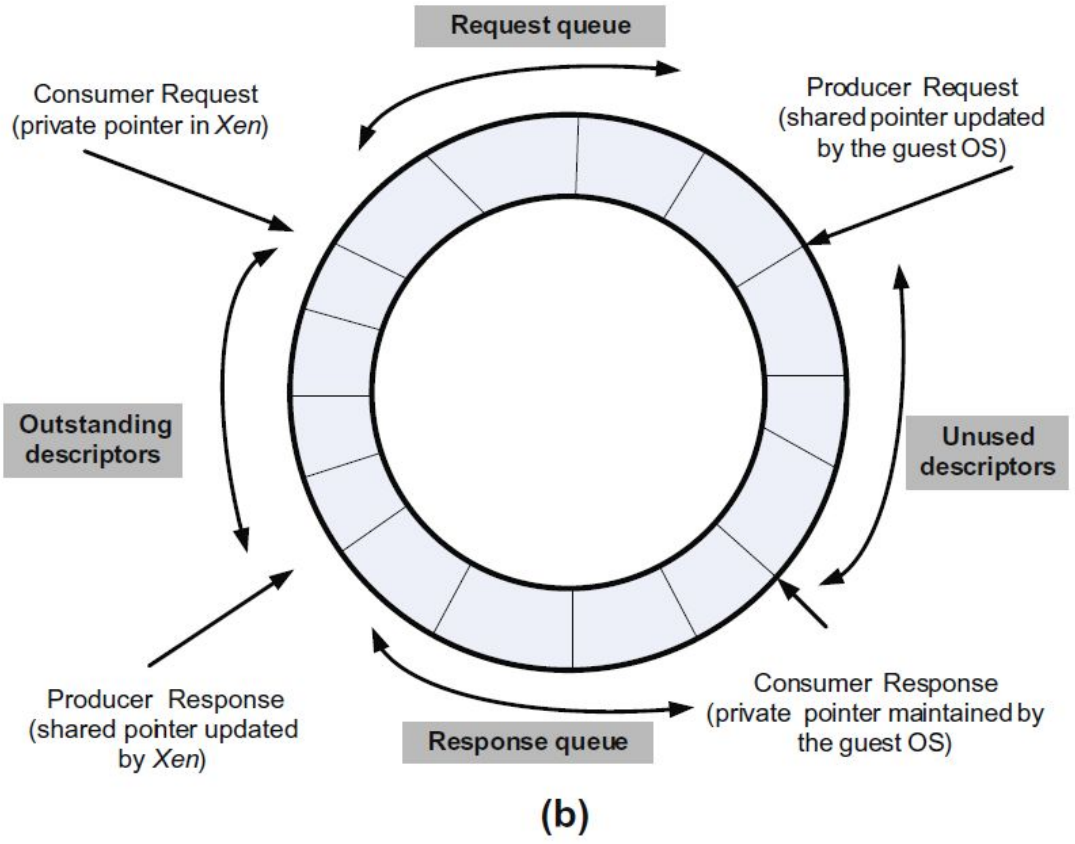



**FIGURE 5.7**

*Xen* zero-copy semantics for data transfer using I/O rings. (a) The communication between a guest domain and the driver domain over an I/O and an event channel; NIC is the Network Interface Controller. (b) The circular ring of buffers.

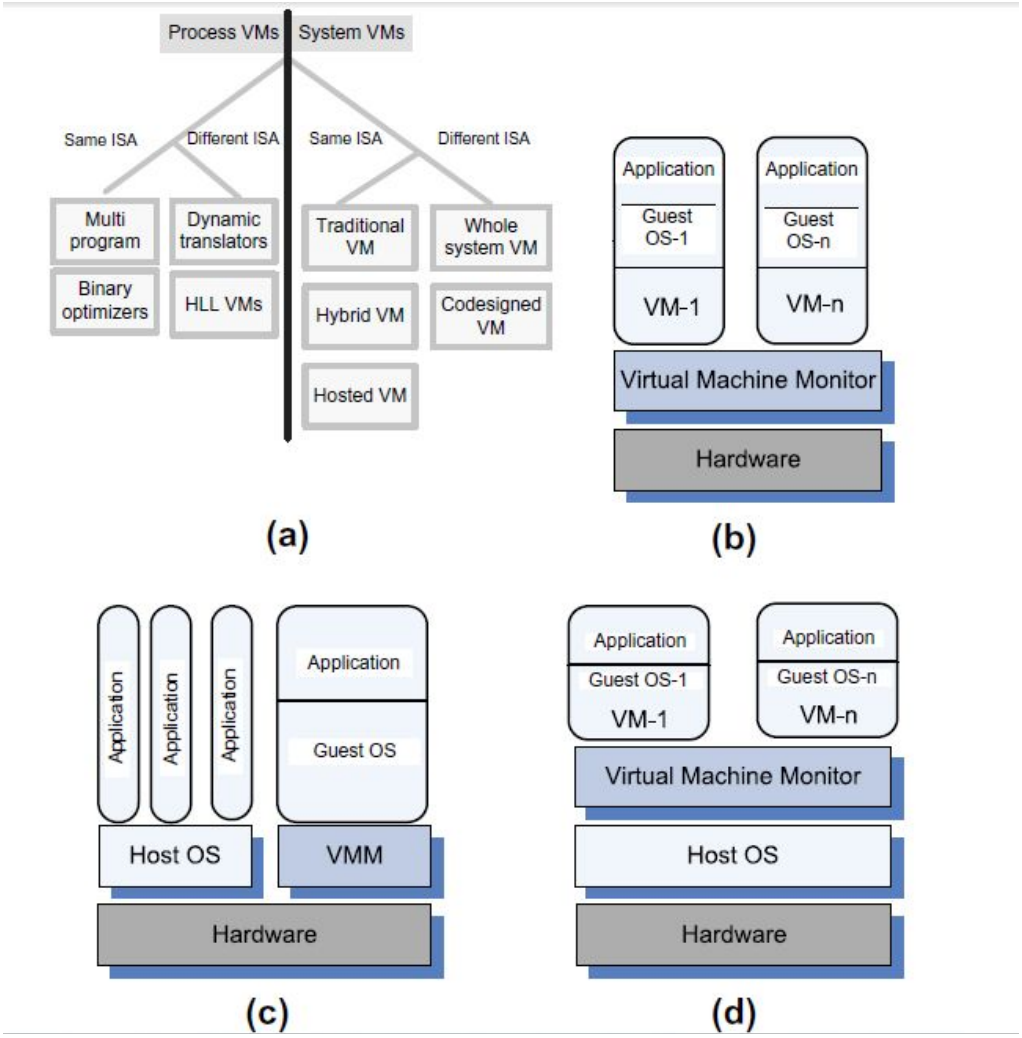
Xen defines abstractions for networking and I/O devices. Split drivers have a front-end in the DomU and a back-end in Dom0; the two communicate via a ring in shared memory. Xen enforces access control for the shared memory and passes synchronization signals. Access control lists (ACLs) are stored in the form of grant tables, with permissions set by the owner of the memory.

Data for I/O and network operations move vertically through the system very efficiently using a set of I/O rings (see Figure 5.7). A ring is a circular queue of descriptors allocated by a domain and accessible within Xen. Descriptors do not contain data; the data buffers are allocated off-band by the guest OS. Memory committed for I/O and network operations is supplied in a manner designed to avoid "cross-talk," and the I/O buffers holding the data are protected by preventing page faults of the corresponding page frames.

Each domain has one or more virtual network interfaces (VIFs) that support the functionality of a network interface card. A VIF is attached to a virtual firewall-router (VFR). Two rings of buffer descriptors, one for packet sending and one for packet

receiving, are supported. To transmit a packet, a guest OS enqueues a buffer descriptor to the send ring, then Xen copies the descriptor and checks safety and finally copies only the packet header, not the payload, and executes the matching rules.

**3. Describe briefly with diagrams Traditional, Hybrid, and Hosted VMs. 137**




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

- **Traditional.** VM also called a "bare metal" VMM. A thin software layer that runs directly on the host machine hardware; its main advantage is performance [see Figure 5.3(b)]. Examples: VMWare ESX, ESXi Servers, Xen, OS370, and Denali.
  • **Hybrid.** The VMMshares the hardware with the existing OS [see Figure 5.3(c)]. Example: VMWare Workstation.
  • **Hosted.** The VM runs on top of an existing OS [see Figure 5.3(d)]. The main advantage of this approach is that the VM is easier to build and install. Another advantage of this solution is that the VMM could use several components of the host OS, such as the scheduler, the pager, and the I/O drivers, rather than providing its own. A price to pay for this simplicity is the increased overhead and associated performance penalty; indeed, the I/O operations, page faults, and scheduling requests from a guest OS are not handled directly by the VMM. Instead, they are passed to the host OS.

**4. Discuss briefly the problems faced in the virtualization of the x86 Architecture. 142**

the problems faced by virtualization of the x86 architecture:
• **Ring deprivileging**. This means that a VMM forces the guest software, the operating system, and the applications to run at a privilege level greater than 0. Recall that the x86 architecture provides four protection rings at levels 0–3. Two solutions are then possible: (a) The (0/1/3) mode, in which the VMM, the OS, and the application run at privilege levels 0, 1, and 3, respectively; or (b) the (0,3,3) mode, in which the VMM, a guest OS, and applications run at privilege levels 0, 3, and 3, respectively. The first mode is not feasible for x86 processors in 64-bit mode, as we shall see shortly.
• **Ring aliasing.** Problems created when a guest OS is forced to run at a privilege level other than that it was originally designed for. For example, when the CR register4 is PUSHed, the current privilege level is also stored on the stack [253].
• **Address space compression**. A VMM uses parts of the guest address space to store several system data structures, such as the interrupt-descriptor table and the global-descriptor table. Such data structures must be protected, but the guest software must have access to them.
• **Non Faulting access to privileged state**. Several instructions, LGDT, SIDT, SLDT, and LTR that load the registers GDTR, IDTR, LDTR, and TR, can only be executed by software running at privilege level 0, because these instructions point to data structures that control the CPU operation. Nevertheless, instructions that store from these registers

fail silently when executed at a privilege level other than 0. This implies that a guest OS executing one of these instructions does not realize that the instruction has failed.

• **Guest system calls**. Two instructions, SYSENTER and SYSEXIT, support low-latency system calls. The first causes a transition to privilege level 0, whereas the second causes a transition from privilege level 0 and fails if executed at a level higher than 0. The VMM must then emulate every guest execution of either of these instructions, which has a negative impact on performance.

• **Interrupt virtualization**. In response to a physical interrupt, the VMM generates a "virtual interrupt" and delivers it later to the target guestOS. But everyOShas the ability to mask interrupts5; thus the virtual interrupt could only be delivered to the guestOS when the interrupt is not masked. Keeping track of all guestOS attempts to mask interrupts greatly complicates theVMMand increases the overhead.

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

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

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

Similar problems exist for the Itanium architecture

**5. Identify with a diagram the layers, and the interfaces between layers in a computer system.134**

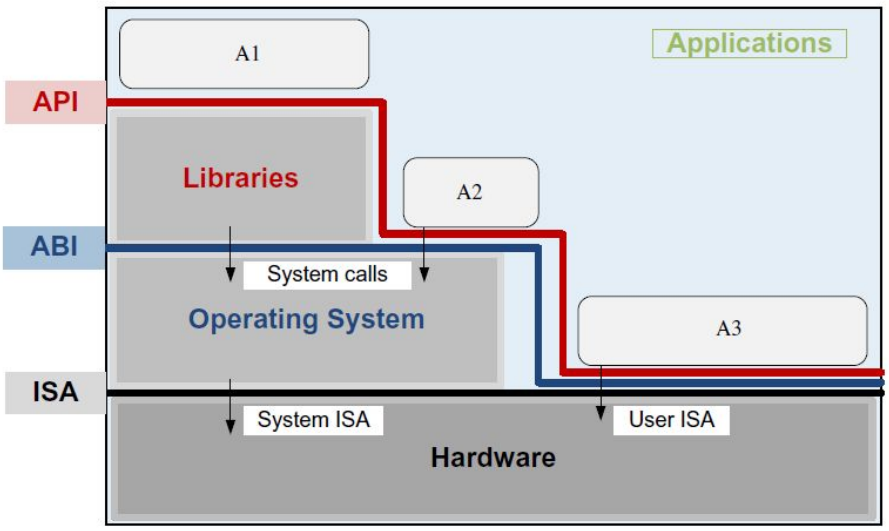



**FIGURE 5.1**

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

The first interface we discuss is the instruction set architecture (ISA) at the boundary of the hardware and the software. The next interface is the application binary interface (ABI), which allows the ensemble consisting of the application and the library modules to access the hardware. The ABI does not include privileged system instructions; instead, it invokes system calls. Finally, the application program interface (API) defines the set of instructions the hardware was designed to execute and gives the application access to the ISA. It includes HLL library calls, which often invoke system calls. The ABI is the projection of the computer system seen by the process, and the API is the projection of the system from the perspective of the HLL program.

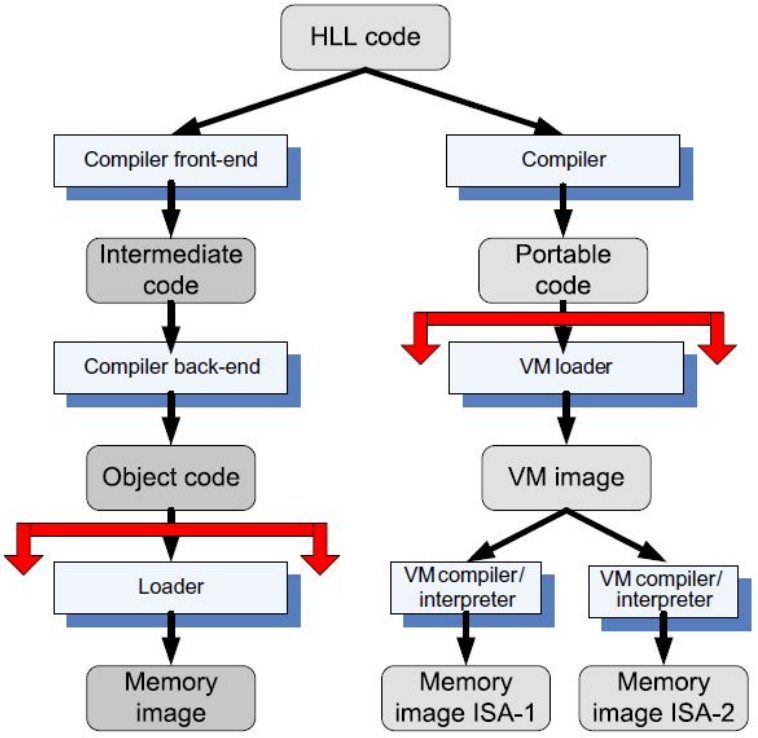
**6. Discuss how issues in paravirtualization have been resolved and implemented on the x86-64 Itanium processor through the vBlades VMM project. 152**

The goal of the vBlades project was to create a VMM for the Itanium family of IA64 Intel processors,14 capable of supporting the execution of multiple operating systems in isolated protection domains with security and privacy enforced by the hardware. The VMM was also expected to support optimal server utilization and allow comprehensive measurement and monitoring for detailed performance analysis.

We first review the features of the Itanium processor that are important for virtualization, starting with the observation that the hardware supports four privilege rings, PL0, PL1, PL2, and PL3. Privileged instructions can only be executed by the kernel running at level PL0, whereas applications run at level PL3 and can only execute non privileged instructions; PL2 and PL4 rings are generally not used. The VMMuses ring compression and runs itself
at PL0 and PL1 while forcing a guest OS to run at PL2. A first problem, called privilege leaking, is that several non privileged instructions allow an application to determine the current privilege level (CPL); thus, a guest OS may not accept to boot or run or may itself attempt to make use of all four privilege rings.
Itanium Was selected because of its multiple functional units and multithreading support. The Itanium processor has 30 functional units: six general-purpose ALUs, two integer units, one shift unit, four data cache units, six multimedia units, two parallel shift units, one parallel multiply, one population count,
three branch units, two 82-bit floating-point multiply-accumulate units, and two SIMD floating-point multiply-accumulate units. A 128-bit instruction word contains three instructions; the fetch mechanism can read up to two instruction words per clock from the L1 cache into the pipeline. Each unit can execute a particular subset of the instruction set. The hardware supports 64-bit addressing; it has 32 64-bit general-purpose registers numbered from R0 to R31 and 96 automatically renumbered registers, R32 through R127, used by procedure calls.When a procedure is entered, the alloc instruction specifies the registers the procedure can access by setting the bits of a 7-bit field that controls the register usage. An illegal read operation from such a register out
of range returns a zero value, whereas an illegal write operation to it is trapped as an illegal instruction. The Itanium processor supports isolation of the address spaces of different processes with eight privileged region registers. The Processor Abstraction Layer (PAL) firmware allows the caller to set the values in the region register. The VMM intercepts the privileged instruction issued by the guest OS to its PAL and partitions the set of address spaces among the guest OSs to ensure isolation. Each guest is limited to 218 address spaces.
The hardware has an IVA register to maintain the address of the interruption vector table. The entries in this table control both the interrupt delivery and the interrupt state collection. Different types of interrupts activate different interrupt handlers pointed from this table, provided that the particular interrupt is not

disabled. Each guest OS maintains its own version of this vector table and has its own IVA register. The hypervisor uses the guest OS IVA register to give control to the guest interrupt handler when an interrupt occurs.

**7. Compare with a diagram the conventional compilation process of an HLL program and the compilation process for a virtual machine environment that produces portable code. 135**




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

The binaries created by a compiler for a specific ISA and a specific operating system are not portable. Such code cannot run on a computer with a different ISA or on computers with the same ISA but with different operating systems. However, it is possible to compile an HLL program for a VM environment, where portable code is produced and distributed and then converted by binary translators to the ISA of the host system. A dynamic binary translation converts blocks of guest instructions from the portable code

to the host instruction and leads to a significant performance improvement as such blocks are cached and reused.

**8. Discuss briefly the Network Optimization Strategies used by Xen. Give the Optimized network architecture of Xen. 149**

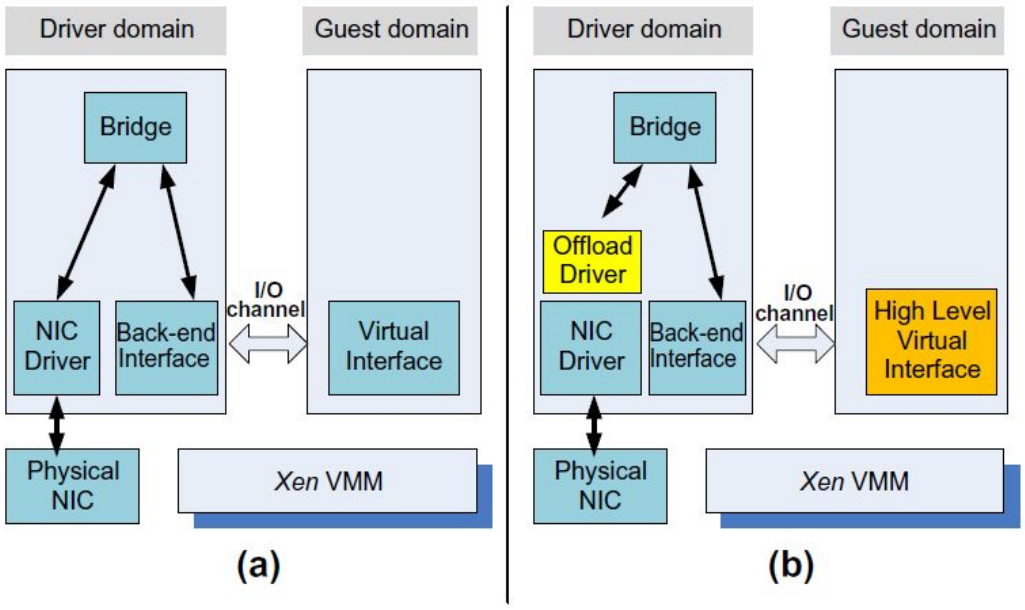



**FIGURE 5.8**

*Xen* network architecture. (a) The original architecture. (b) The optimized architecture.

The Xen network optimization strategies are:-

**(i) the virtual interface-**The original virtual network interface provides the guest domain with the abstraction of a simple low-level network interface supporting sending and receiving primitives. This design supports a wide range of physical devices attached to the driver domain but does not take advantage of the capabilities of some physical NICs such as checksum offload .

**(ii) the I/O channel-**The next target of the optimization effort is the communication between the guest domain and the driver domain. Rather than copying a data buffer holding a packet, each packet is allocated in a new page and then the physical page containing the packet is remapped into the target domain.. The optimization is based on the observation that there is no need to remap the entire packet; As a result, the optimized implementation is based on an "out-of-band" channel used by the guest domain to provide the bridge with the packet MAC header.

**(iii) the virtual memory.** :The third optimization covers virtual memory. Virtual memory in Xen 2.0 takes advantage of the superpage and global page-mapping hardware features available on Pentium and Pentium Pro processors. A superpage

increases the granularity of the dynamic address translation; When new processes are created, the guest OS must allocate read-only pages for the page tables of the address spaces running under the guest OS, and that forces the system to use traditional page mapping rather than superpage mapping. The optimized version uses a special memory allocator to avoid this problem.

# UNIT-IV Answers

1. **Illustrate with an example and its diagram how coordination between autonomic system managers can be utilized for cloud resource management. 172**

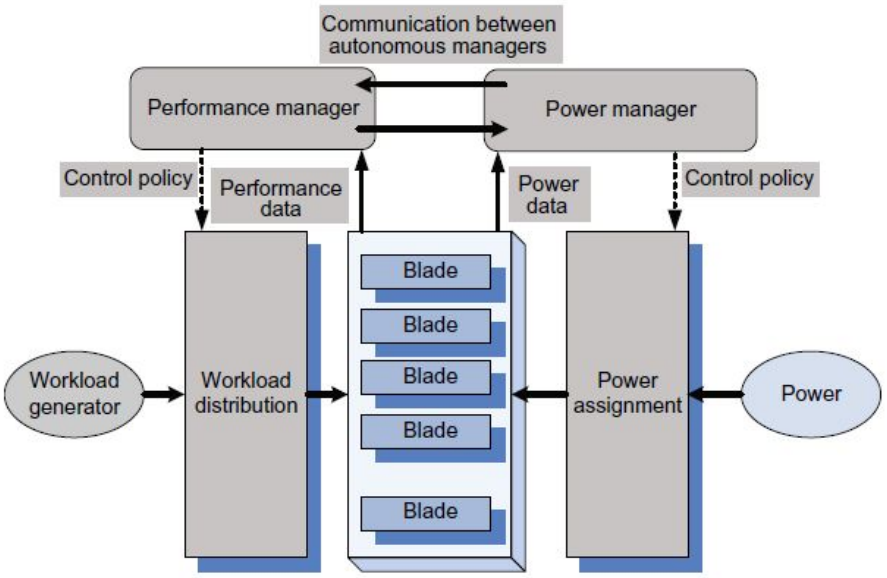



**FIGURE 6.3**

Autonomous performance and power managers cooperate to ensure SLA prescribed performance and energy optimization. They are fed with performance and power data and implement the performance and power management policies, respectively.

Virtually all modern processors support dynamic voltage scaling (DVS) as a mechanism for energy saving. Indeed, the energy dissipation scales quadratically with the supply voltage. The power management controls the CPU frequency and, thus, the rate of instruction execution. For some compute-intensive workloads the performance decreases

linearly with the CPU clock frequency, whereas for others the effect of lower clock frequency is less noticeable or nonexistent. The clock frequency of individual blades/servers is controlled by a power manager, typically implemented in the firmware; it adjusts the clock frequency several times a second.

The approach to coordinating power and performance management in [187] is based on several ideas:

- Use a joint utility function for power and performance. The joint performance-power utility function, $U_{pp}(R, P)$, is a function of the response time, $R$, and the power, $P$, and it can be of the form

$$U_{pp}(R, P) = U(R) - \epsilon \times P \quad \text{or} \quad U_{pp}(R, P) = \frac{U(R)}{P}, \tag{6.18}$$

  with $U(R)$ the utility function based on response time only and $\epsilon$ a parameter to weight the influence of the two factors, response time and power.
- Identify a minimal set of parameters to be exchanged between the two managers.
- Set up a power cap for individual systems based on the utility-optimized power management policy.
- Use a standard performance manager modified only to accept input from the power manager regarding the frequency determined according to the power management policy. The power manager consists of Tcl (Tool Command Language) and C programs to compute the per-server (per-blade) power caps and send them via IPMI[5] to the firmware controlling the blade power. The power manager and the performance manager interact, but no negotiation between the two agents is involved.

- Use standard software systems. For example, use the WebSphere Extended Deployment (WXD), middleware that supports setting performance targets for individual Web applications and for the monitor response time, and periodically recompute the resource allocation parameters to meet the targets set. Use the Wide-Spectrum Stress Tool from the IBM Web Services Toolkit as a workload generator.

*(Page 174 )*

2. **Describe using schematics the ASCA combinatorial auction algorithm. 180**

Informally, in the ASCA algorithm the participants at the auction specify the resource and the quantities of that resource offered or desired at the price listed for that time slot.If all its components are negative, the auction stops; negative components mean that the demand does not exceed the offer. If the demand is larger than the offer the auctioneer increases the price for items with a positive excess demand and solicits bids at the new price.An auctioning algorithm is very appealing because it supports resource bundling and does not require a model of the system

Cloud_Computing_Theory_And_Practice.pdf - Adobe Reader

File Edit View Window Help

Open 181 (200 of 415) 91.4% Tools Fill & Sign Comment

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

```
1: set t = 0, p(0) = p̃
2: loop
3:   collect bids x_u(t) = G_u(p(t)), ∀u
4:   calculate excess demand z(t) = Σ_u x_u(t)
5:   if z(t) <0 then
6:     break
7:     else
```



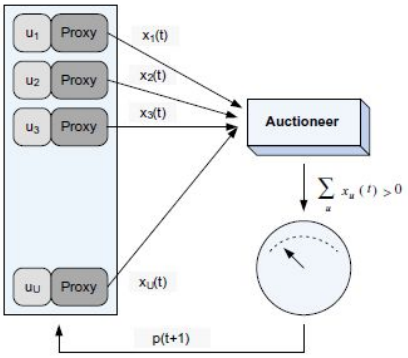



**FIGURE 6.6**

The schematics of the ASCA algorithm. To allow for a single round, auction users are represented by proxies that place the bids $x_u(t)$. The auctioneer determines whether there is an excess demand and, in that case, raises the price of resources for which the demand exceeds the supply and requests new bids.

```
8:       update prices p(t + 1) = p(t) + g(x(t), p(t))
9:       t ← t + 1
10:  end if
11: end loop
```

In this algorithm $g(x(t), p(t))$ is the function for setting the price increase. This function can be correlated with the excess demand $z(t)$, as in $g(x(t), p(t)) = \alpha z(t)^+$ (the notation $x^+$ means max $(x, 0)$)

3. **Describe briefly policies for Cloud Resource Management (CRM). 164**

A policy typically refers to the principal guiding decisions, whereas mechanisms represent the means to

implement policies. Separation of policies from mechanisms is a guiding principle in computer science.

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

**1. Admission control.-**The explicit goal of an admission control policy is to prevent the system from accepting workloads in violation of high-level system policies; for example, a system may not accept an additional workload that would prevent it from completing work already in progress or contracted.

**2.Capacity allocation** means to allocate resources for individual instances; an instance is an activation of a service. Locating resources subject to multiple global optimization constraints requires a search of a very large search space when the state of individual systems changes rapidly.

**3. Load balancing &. Energy optimization.**.Load balancing and energy optimization can be done locally, but global load-balancing and energy optimization policies encounter the same difficulties as the one we have already discussed. Load balancing
and energy optimization are correlated and affect the cost of providing the services In cloud computing a critical goal is minimizing the cost of providing the service and, in particular, minimizing the energy consumption.

**5. Quality-of-service (QoS) guarantees.-** It is that aspect of
resource management that is probably the most difficult to address and, at the same time, possibly the most critical to the future of cloud computing.

**4. List the conditions of the pricing and allocation algorithms for the combinatorial auctions of cloud resources.**

**Pricing and Allocation Algorithms.** A pricing and allocation algorithm partitions the set of users into two disjoint sets, winners and losers, denoted as W and L, respectively. The algorithm should:

**1. Be computationally tractable.** Traditional combinatorial auction algorithms such as Vickey-Clarke- Groves (VLG) fail this criteria, because they are not computationally tractable.

**2. Scale well.** Given the scale of the system and the number of requests for service, scalability is a necessary condition.

**3. Be objective.** Partitioning in winners and losers should only be based on the price πu of a user's bid. If the price exceeds the threshold, the user is a winner; otherwise the user is a loser.

**4. Be fair.** Make sure that the prices are uniform. All winners within a given resource pool pay the same price.

5. Indicate clearly at the end of the auction the unit prices for each resource pool.

6. Indicate clearly to all participants the relationship between the supply and the demand in the system.

The function to be maximized is

$$\max_{x,p} f(x, p).$$

5. **Describe the two-level control architecture for automatic resource management with a diagram. 170**

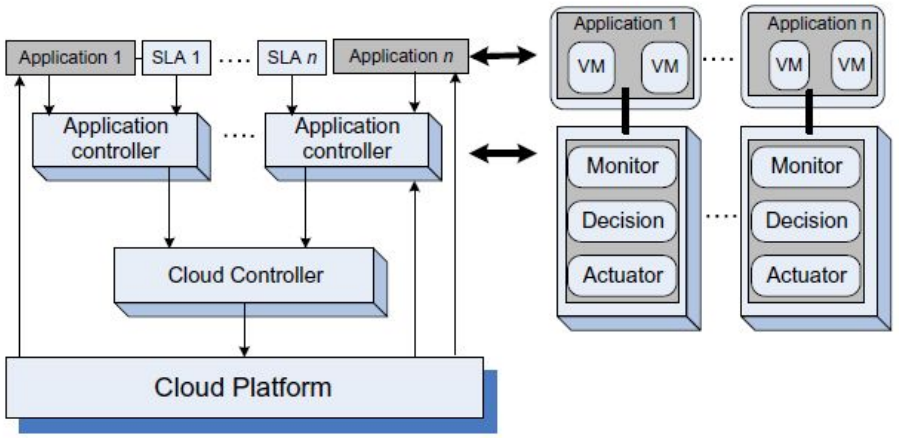



**FIGURE 6.2**

A two-level control architecture. Application controllers and cloud controllers work in concert.

The automatic resource management is based on two levels of controllers, one for the service provider and one for the application. The main components of a control system are the inputs, the control system components, and the outputs. The inputs in such models are the offered workload and the policies for admission control, the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees in the cloud. The system components are sensors used to estimate relevant measures of performance and controllers that implement various policies; the output is the resource allocations to the individual applications . The controllers use the feedback provided by sensors to stabilize the system; stability is related to the change of the output. If the change is too large, the system may become unstable. In our context the system could experience thrashing, the amount of useful time dedicated to the execution of applications becomes increasingly small and most of the system resources are occupied by management functions.

6. **Describe with timing diagrams the Optimal Partitioning Rule (OPR) and Equal Partitioning Rule (EPR) for partitioning workloads in cloud scheduling.**
   *(Page 195-199 Out of Syllabus???)*

**NOT IN SYLLABUS test 2**

**Problem 7.** *Use the start-time fair queuing (SFQ) scheduling algorithm to compute the virtual startup and the virtual finish time for two threads a and b with weights $w_a = 1$ and $w_b = 5$ when the time quantum is $q = 15$ and thread b blocks at time $t = 24$ and wakes up at time $t = 60$. Plot the virtual time of the scheduler function of the real time.*

51

As in Problem 6, we consider two threads with the weights $w_a = 1$ and $w_b = 5$ and the time quantum is $q = 15$, and thread $b$ blocks at time $t = 24$ and wakes up at time $t = 60$.

Initially $S_a^0 = 0$, $S_b^0 = 0$, $v_a(0) = 0$, and $v_b(0) = 0$. The scheduling decisions are made as follows:

1. t=0: we have a tie, $S_a^0 = S_b^0$ and arbitrarily thread $b$ is chosen to run first; the virtual finish time of thread $b$ is

1. t=0: we have a tie, $S_a^0 = S_b^0$ and arbitrarily thread $b$ is chosen to run first; the virtual finish time of thread $b$ is

$$F_b^0 = S_b^0 + q/w_b = 0 + 15/5 = 3. \tag{22}$$

2. t=3: both threads are runnable and thread $b$ was in service, thus, $v(3) = S_b^0 = 0$; then

$$S_b^1 = \max[v(3), F_b^0] = \max(0, 3) = 3. \tag{23}$$

But $S_a^0 < S_b^1$ thus thread $a$ is selected to run. Its virtual finish time is

$$F_a^0 = S_a^0 + q/w_a = 0 + 15/1 = 15. \tag{24}$$

3. t=18: both threads are runnable and thread $a$ was in service at this time thus,

$$v(18) = S_a^0 = 0 \tag{25}$$

and

$$S_a^1 = \max[v(18), F_a^0] = \max[0, 15] = 15. \tag{26}$$

As $S_a^1 = 3 < 12$, thread $b$ is selected to run; the virtual finish time of thread $b$ is now

and

$$S_a^1 = \max[v(18), F_a^0] = \max[0, 15] = 15. \tag{26}$$

As $S_b^1 = 3 < 12$, thread $b$ is selected to run; the virtual finish time of thread $b$ is now

$$F_b^1 = S_b^1 + q/w_b = 3 + 15/5 = 6. \tag{27}$$

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

$$v(21) = S_b^1 = 3 \tag{28}$$

and

$$S_b^2 = \max[v(21), F_b^1] = \max[3, 6] = 6. \tag{29}$$

As $S_b^2 < S_a^1 = 15$, thread $b$ is selected to run again; its virtual finish time is

$$F_b^2 = S_b^2 + q/w_b = 6 + 15/5 = 9. \tag{30}$$

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

$$v(24) = S_b^2 = 6 \tag{31}$$

$$S_b^3 = \max[v(24), F_b^2] = \max[6, 9] = 9. \tag{32}$$

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

$$F_a^1 = S_a^1 + q/w_a = 9 + 15/1 = 24. \tag{33}$$

6. t=39: thread $a$ was in service and it is the only runnable thread at this time, thus,

$$v(39) = S_a^1 = 15 \tag{34}$$

and

$$S_a^1 = \max[v(39), F_a^1] = \max[15, 24] = 24. \tag{35}$$

Then,

$$F_a^2 = S_a^2 + q/w_a = 24 + 15/1 = 39. \tag{36}$$

7. t=54: thread $a$ was in service and it is the only runnable thread at this time thus,

$$v(54) = S_a^1 = 24 \tag{37}$$

and

$$S_a^2 = \max[v(54), F_a^2] = \max[24, 39] = 39. \tag{38}$$

Then,

$$F_a^2 = S_a^3 + q/w_a = 39 + 15/1 = 54. \tag{39}$$

8. t=69: thread $a$ was in service at this time, thus,

$$v(69) = S_a^2 = 39 \tag{40}$$

and

$$S_a^2 = \max[v(54), F_a^2] = \max[24, 39] = 39. \tag{38}$$

Then,

$$F_a^2 = S_a^3 + q/w_a = 39 + 15/1 = 54. \tag{39}$$

8. t=69: thread $a$ was in service at this time, thus,

$$v(69) = S_a^2 = 39 \tag{40}$$

and

$$S_a^4 = \max[v(69), F_a^2] = \max[39, 54] = 54. \tag{41}$$

But now thread $b$ is runnable and $S_b^3 = 9$.

Thus, thread $b$ is activated and

$$F_b^3 = S_b^3 + q/w_b = 9 + 15/5 = 12. \tag{42}$$