USN										

Internal Assessment Test II – May 2019
Answer Solution

Sub:	Cloud Computing				Sub Code:	**18SCS23**	Branch:	CSE
Date:	09/05 /19	Duration:	90 min's	Max Marks: 50	Sem / Sec:	M. Tech (CSE)/ II SEM		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1.	**a) Define Virtualization? Explain Layering and virtualization**	**[07]**	CO3	L2

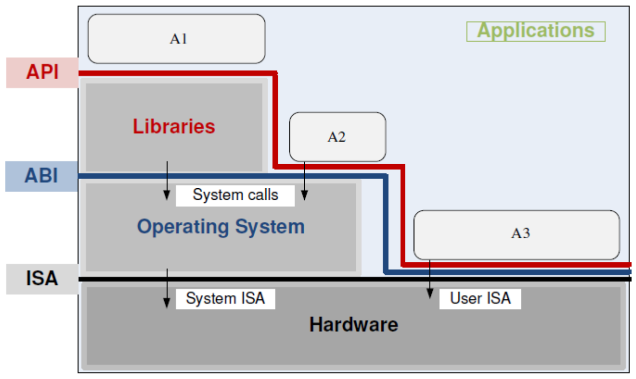
- **Virtualization** abstracts the underlying resources and simplifies their use, isolates users from one another, and supports replication which, in turn, increases the elasticity of the system.

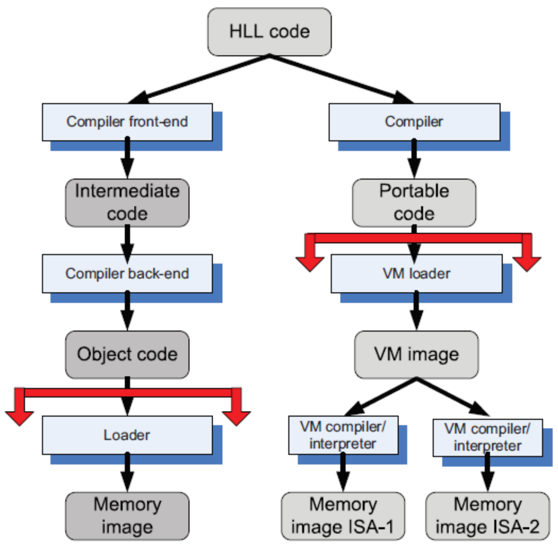
Layering and virtualization

- A common approach to manage system complexity is to identify a set of layers with well defined interfaces among them; the interfaces separate different levels of abstraction.
  - Layering minimizes the interactions among the subsystems and simplifies the description of the subsystems; each subsystem is abstracted through its interfaces with the other subsystems thus, we are able to design, implement, and modify the individual subsystems independently.
  - The ISA (Instruction Set Architecture) defines the set of instructions of a processor; for example, the Intel architecture is represented by the x86-32 and x86-64 instruction sets for systems supporting 32-bit addressing and 64-bit addressing, respectively.
  - The hardware supports two execution modes, a privileged, or kernel mode and a user mode.
  - The instruction set consists of two sets of instructions, privileged instructions that can only be executed in kernel mode and the non-privileged instructions that can be executed in user mode.
  - There are also sensitive instructions that can be executed in kernel and in user mode, but behave differently.
  - The hardware consists of one or more multi-core processors, a system interconnect, (e.g., one or more busses) a memory translation unit,

the main memory, and I/O devices, including one or more networking interfaces.

- Applications written mostly in High Level languages (HLL) often call library modules and are compiled into object code. Privileged operations, such as I/O requests, cannot be executed in user mode; instead, application and library modules issue system calls and the operating system determines if the privileged operations required by the application do not violate system security or integrity and, if so, executes them on behalf of the user.





- 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).

b) Describe the concept of Virtual Machine with respect to cloud

[03] CO3 L2

Virtual Machine (VM)- isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources.

- Process VM - a virtual platform created for an individual process and destroyed once the process terminates.
- System VM - supports an operating system together with many user processes.
- Traditional VM - supports multiple virtual machines and runs directly on the hardware.
- Hybrid VM - shares the hardware with a host operating system and supports multiple virtual machines.
- Hosted VM - runs under a host operating system.

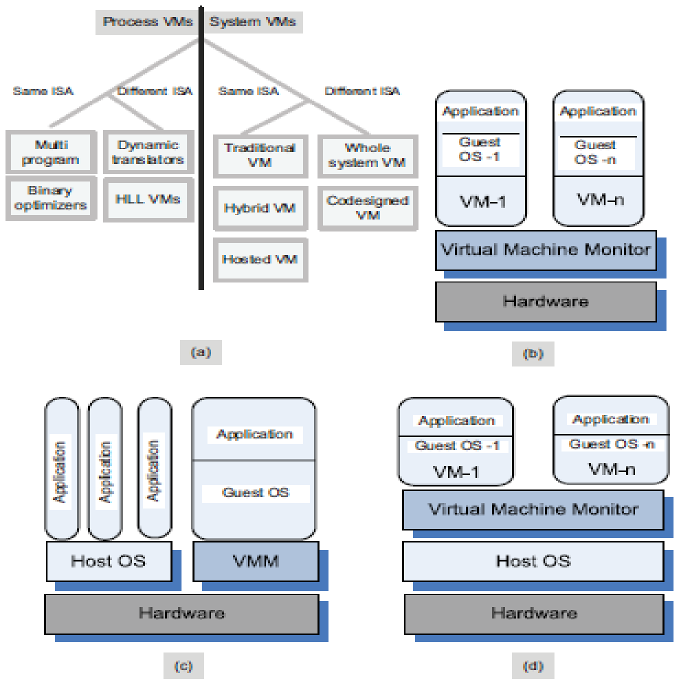
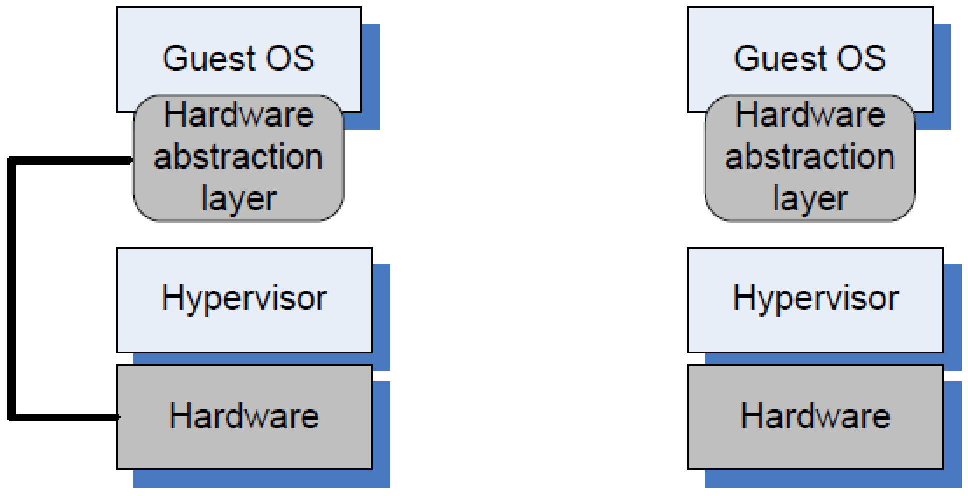


Figure 46: (a) A taxonomy of process and systems VMs for the same and for di Instruction Set Architectures (ISAs). Traditional, Hybrid, and Hosted are three cla VMs for systems with the same ISA. (b) Traditional VMs; the VMM supports m virtual machines and runs directly on the hardware;. (c) Hybrid VM; the VMM sha hardware with a host operating system and supports multiple virtual machines. (d)

2. a) **Write comparison between full virtualization and paravirtualization** **[06]** CO3 L2

Full virtualization and paravirtualization

- Full virtualization – a guest OS can run unchanged under the VMM as if it was running directly on the hardware platform.
- Requires a virtualizable architecture.
- Examples: Vmware.
- **Paravirtualization** - a guest operating system is modified to use only instructions that can be virtualized. Reasons for paravirtualization:
- Some aspects of the hardware cannot be virtualized.
- Improved performance.
- Present a simpler interface.
- Examples: Xen, Denali



(a) Full virtualization (b) Paravirtualization

b) **Write any 4 problems faced by virtualization of the x86 Architecture** **[04]** CO3 L2

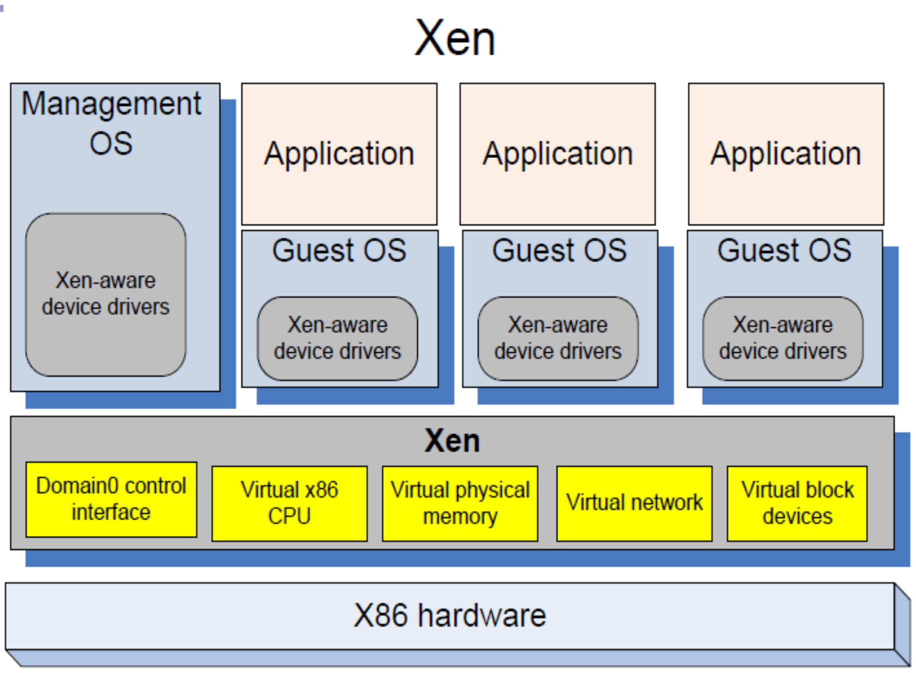
- Ring de-privileging - a VMMs forces the operating system and the applications to run at a privilege level greater than 0.
- Ring aliasing - a guest OS is forced to run at a privilege level other than that it was originally designed for.
- Address space compression - a VMM uses parts of the guest address space to store several system data structures.
- Non-faulting access to privileged state - several store instructions can only be executed at privileged level 0 because they operate on data structures that control the CPU operation. They fail silently when executed at a privilege level other than 0.

- Guest system calls which cause transitions to/from privilege level 0 must be emulated by the VMM.
- Interrupt virtualization - in response to a physical interrupt, the VMM generates a ``virtual interrupt'' and delivers it later to the target guest OS which can mask interrupts.
- 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 protect VMM code from being overwritten by guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish between privilege levels 0, 1, and 2, thus 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.
- The task-priority register 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.

3. **Write case study on Xen paravirtualization** **[10]** CO3 L2

- The goal of the Cambridge group - design a VMM capable of scaling to about 100 VMs running standard applications and services without any modifications to the Application Binary Interface (ABI).
- Linux, Minix, NetBSD, FreeBSD, NetWare, and OZONE can operate as paravirtualized Xen guest OS running on x86, x86-64, Itanium, and ARM architectures.
- Xen domain - ensemble of address spaces hosting a guest OS and applications running under the guest OS. Runs on a virtual CPU.
- Dom0 - dedicated to execution of Xen control functions and privileged instructions.
- DomU - a user domain.

- Applications make system calls using hypercalls processed by Xen; privileged instructions issued by a guest OS are paravirtualized and must be validated by Xen.



Xen implementation on x86 architecture

- Xen runs at privilege Level 0, the guest OS at Level 1, and applications at Level 3.
- The x86 architecture does not support either the tagging of TLB entries or the software management of the TLB. Thus, address space switching, when the VMM activates a different OS, requires a complete TLB flush; this has a negative impact on the performance.
- Solution - load Xen in a 64 MB segment at the top of each address space and delegate the management of hardware page tables to the guest OS with minimal intervention from Xen. This region is not accessible or re-mappable by the guest OS.
- Xen schedules individual domains using the Borrowed Virtual Time (BVT) scheduling algorithm.
- A guest OS must register with Xen a description table with the addresses of exception handlers for validation.

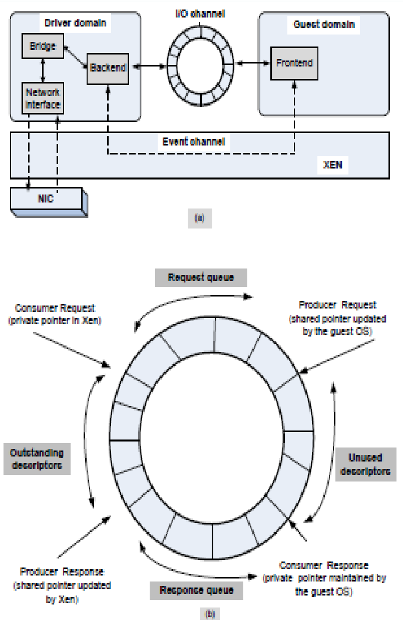
Dom0 components

- XenStore – a Dom0 process.

- Supports a system-wide registry and naming service.
- Implemented as a hierarchical key-value storage.
- A watch function informs listeners of changes of the key in storage they have subscribed to.
- Communicates with guest VMs via shared memory using Dom0 privileges.
- Toolstack - responsible for creating, destroying, and managing the resources and privileges of VMs.
- To create a new VM, a user provides a configuration file describing memory and CPU allocations and device configurations.
- Toolstack parses this file and writes this information in XenStore.
- Takes advantage of Dom0 privileges to map guest memory, to load a kernel and virtual BIOS and to set up initial communication channels with XenStore and with the virtual console when a new VM is created.

Xen abstractions for networking and I/O

- Each domain has one or more Virtual Network Interfaces (VIFs) which support the functionality of a network interface card. A VIF is attached to a Virtual Firewall-Router (VFR).
- Split drivers have a front-end in the DomU and the back-end in Dom0; the two communicate via a ring in shared memory.
- Ring - 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.
- 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, copies only the packet header, not the payload, and executes the matching rules.


Driver domain
Bridge
Backend
Network interface
I/O channel
Guest domain
Frontend
Event channel
XEN
NIC
(a)
Request queue
Consumer Request (private pointer in Xen)
Producer Request (shared pointer updated by the guest OS)
Outstanding descriptors
Unused descriptors
Producer Response (shared pointer updated by Xen)
Response queue
Consumer Response (private pointer maintained by the guest OS)
(b)

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.

4. **a) Explain the policies and mechanisms for resource management** **[05]** CO4 L2

- A policy typically refers to the principles guiding decisions, while mechanisms represent that means to implement policies.
- Cloud resource management policies can be loosely grouped into five classes:
  1. Admission control.
  2. Capacity allocation.
  3. Load balancing.
  4. Energy optimization.
  5. Quality of service (QoS) guarantees.
- The explicit goal of an admission control policy is to prevent the system from accepting workload in violation of high-level system policies; for example, a system may not accept additional workload which would prevent it from completing work already in progress or contracted.
- Limiting the workload requires some knowledge of the global state of the system; in a dynamic system such knowledge, when available, is at best obsolete.

- 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.
- Load balancing and energy optimization can be done locally, but global load balancing and energy optimization policies encounter the same difficulties.
- State information for these models can be too intrusive and unable to provide accurate data. Many techniques are concentrated on system performance in terms of throughput and time in system, but they rarely include energy trade-offs or QoS guarantees.
- The four basic mechanisms for the implementation of resource management policies are:

  • **Control theory**. Control theory uses the feedback to guarantee system stability and predict transient behavior but can be used only to predict local rather than global behavior; Kalman filters have been used for unrealistically simplified models.

  • **Machine learning**. A major advantage of machine learning techniques is that they do not need a performance model of the system this technique could be applied for coordination of several autonomic system managers.

  • **Utility-based**. Utility-based approaches require a performance model and a mechanism to correlate user-level performance with cost.

  • **Market-oriented/economic mechanisms**. Such mechanisms do not require a model of the system, e.g., combinatorial auctions for bundles of resources

b)Explain a two level architecture for resource allocation

[05] CO4 L2

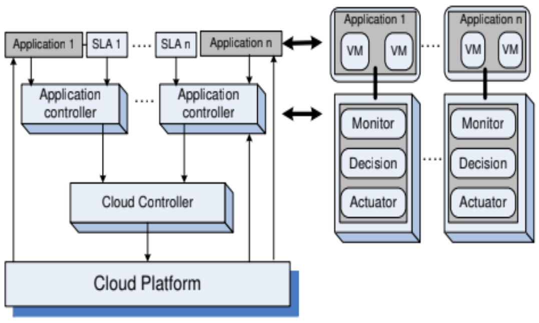


Figure 57: A two-level control architecture; application controllers and cloud controllers work in concert.

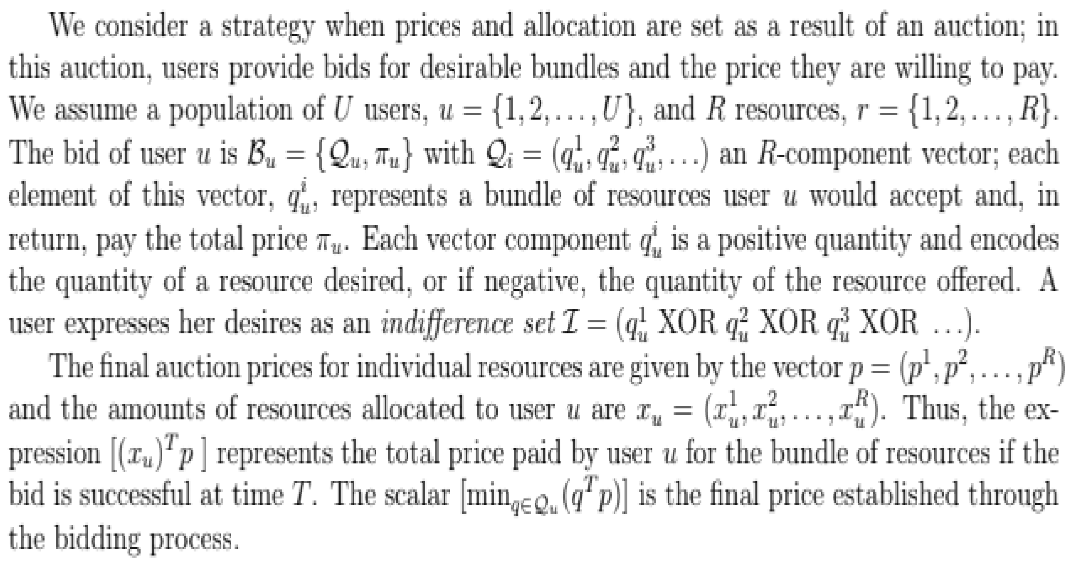
- The Automatic resource management is based on two levels of controllers, one for the service provider and one for the applications.
- The main components of a control system are: the inputs, the control system components, and the outputs.
- The inputs in such models are: the offered workload and the 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 which 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 then the system may become unstable.
- There are three main sources of instability in any control system:
  1. The delay in getting the system reaction after a control action;
  2. The granularity of the control, the fact that a small change enacted by the controllers leads to very large changes of the output;
  3. Oscillations, when the changes of the input are too large and the control is too weak, such that the changes of the input propagate directly to the output.
- Two types of policies are used in autonomic systems:
  (i) threshold-based policies and

(ii) sequential decision policies based on Markovian decision models.

- In the first case, upper and lower bounds on performance trigger adaptation through resource reallocation; such policies are simple and intuitive but require setting per-application thresholds.

5. Explain the concept of resource bundling with supportive algorithms **[10]** CO4 L2

- Resources in a cloud are allocated in bundles; users get maximum benefit from a specific combination of resources. Indeed, along with CPU cycles, an application needs specific amounts of main memory, disk space, network bandwidth, and so on.
- Resource bundling complicates traditional resource allocation models and has generated an interest in economic models and, in particular, in auction algorithms. In the context of cloud computing, an auction is the allocation of resources to the highest bidder.
- **Combinatorial auctions.** Auctions in which participants can bid on combinations of items or packages are called combinatorial auctions such auctions provide a relatively simple, scalable, and tractable solution to cloud resource allocation.
- Two recent combinatorial auction algorithms are the **Simultaneous Clock Auction** and **the Clock Proxy Auction** ;
- The algorithm is called Ascending Clock Auction, (ASCA). In all these algorithms the current price for each resource is represented a "clock" seen by all participants at the auction.
- consider a strategy when prices and allocation are set as a result of an auction; in auction, users provide bids for desirable bundles and the price they are willing to pay.



We consider a strategy when prices and allocation are set as a result of an auction; in this auction, users provide bids for desirable bundles and the price they are willing to pay. We assume a population of U users, $u = \{1, 2, \ldots, U\}$, and R resources, $r = \{1, 2, \ldots, R\}$. The bid of user u is $\mathcal{B}_u = \{\mathcal{Q}_u, \pi_u\}$ with $\mathcal{Q}_i = (q_u^1, q_u^2, q_u^3, \ldots)$ an R-component vector; each element of this vector, q_u^i, represents a bundle of resources user u would accept and, in return, pay the total price π_u. Each vector component q_u^i is a positive quantity and encodes the quantity of a resource desired, or if negative, the quantity of the resource offered. A user expresses her desires as an *indifference set* $\mathcal{I} = (q_u^1 \text{ XOR } q_u^2 \text{ XOR } q_u^3 \text{ XOR } \ldots)$.

The final auction prices for individual resources are given by the vector $p = (p^1, p^2, \ldots, p^R)$ and the amounts of resources allocated to user u are $x_u = (x_u^1, x_u^2, \ldots, x_u^R)$. Thus, the expression $[(x_u)^T p\,]$ represents the total price paid by user u for the bundle of resources if the bid is successful at time T. The scalar $[\min_{q \in \mathcal{Q}_u}(q^T p)]$ is the final price established through the bidding process.

Pricing and allocation algorithms. A pricing and allocation algorithm partitions the set of users in 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, 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 then 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.

Table 15: The constraints for a combinatorial auction algorithm

$x_u \in \{0 \cup \mathcal{Q}_u\},\ \forall u$	-a user gets all resources or nothing
$\sum_u x_u \leq 0$	-final allocation leads to a net surplus of resources
$\pi_u \geq (x_u)^T p,\ \forall u \in \mathcal{W}$	-auction winners are willing to pay the final price
$(x_u)^T p = \min_{q \in \mathcal{Q}_u}(q^T p),\ \forall u \in \mathcal{W}$	-winners get the cheapest bundle in $\mathcal{I}$
$\pi_u < \min_{q \in \mathcal{Q}_u}(q^T p),\ \forall u \in \mathcal{L}$	-the bids of the losers are below the final price
$p \geq 0$	-prices must be non-negative

The ASCA combinatorial auction algorithm. Informally, in the ASCA algorithm [332] 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. Then the *excess vector*

$$z(t) = \sum_u x_u(t) \tag{67}$$

is computed. If all its components are negative, then the auction stops; negative components mean that the demand does not exceed the offer. If the demand is larger than the offer, $z(t) \geq 0$, then the auctioneer increases the price for items with a positive excess demand and solicits bids at the new price. Note that the algorithm satisfies conditions 1 through 6; all users discover the price at the same time and pay or receive a "fair" payment relative to uniform resource prices, the computation is tractable, and the execution time is linear in the number of participants at the auction and the number of resources. The computation is robust, generates plausible results regardless of the initial parameters of the system.

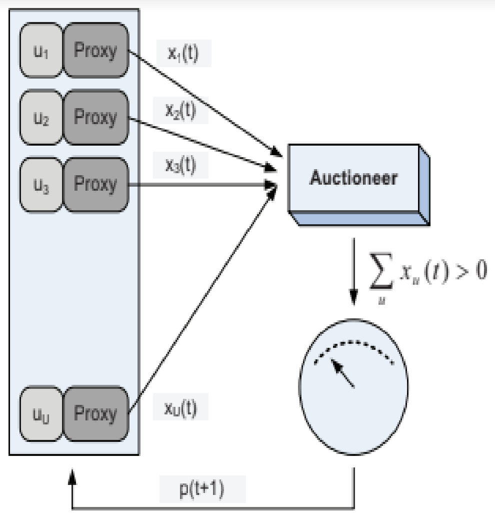


Figure 61: The schematics of the ASCA algorithm; to allow for a single round auction users are represented by proxies which place the bids $x_u(t)$. The auctioneer determines if there is an excess demand and, in that case, it raises the price of resources for which the demand exceeds the supply and requests new bids.

There is a slight complication as the algorithm involves user bidding in multiple round to address this problem the user proxies automatically adjust their demands on behalf the actual bidders, as shown in Figure 61. These proxies can be modeled as functions whic compute the "best bundle" from each $\mathcal{Q}_u$ set given the current price

$$\mathcal{Q}_u = \begin{cases} \hat{q}_u & \text{if } \hat{q}_u^T p \leq \pi_u \quad \text{with } \hat{q}_u \in \arg\min(q_u^T p) \\ 0 & \text{otherwise} \end{cases}$$

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

```
1 :  set $t = 0, p(0) = \bar{p}$
2 :  **loop**
3 :     collect bids $x_u(t) = \mathcal{G}_u(p(t)), \quad \forall u$
4 :     calculate excess demand $z(t) = \sum_u x_u(t)$
5 :     **if** $z(t) < 0$ **then**
6 :        break
7 :     **else**
8 :        update prices $p(t+1) = p(t) + g(x(t), p(t))$
9 :        $t \leftarrow t + 1$
10 :     **end if**
11 :  **end loop**
```

In this algorithm $g(x(t), p(t))$ is the function for setting the price increase. This functio can be correlated with the excess demand $z(t)$ as in $g(x(t), p(t)) = \alpha z(t)^+$ (the notation x means $\max(x, 0)$) with α a positive number. An alternative is to ensure that the price doe not increase by an amount larger than δ; in that case $g(x(t), p(t)) = \min(\alpha z(t)^+, \delta e)$ wit $e = (1, 1, \ldots, 1)$ is an R-dimensional vector and minimization is done componentwise.

[10] CO4 L2

6. **Explain Fair queuing and start-time fair queuing**

Fair queuing:

benefit from a larger bandwidth.

The *fair queuing (FQ)* algorithm in [102] proposes a solution to this problem. First, it introduces a *bit-by-bit round-robin (BR)* strategy; as the name implies, in this rather impractical scheme a single bit from each queue is transmitted and the queues are visited in a round-robin fashion. Let $R(t)$ be the number of rounds of the BR algorithm up to time t and $N_{active}(t)$ the number of active flows through the switch. Call t_i^a the time when the packet i of flow a, of size P_i^a bits arrives and call S_i^a and F_i^a the values of $R(t)$ when the first and the last bit, respectively, of the packet i of flow a are transmitted. Then,

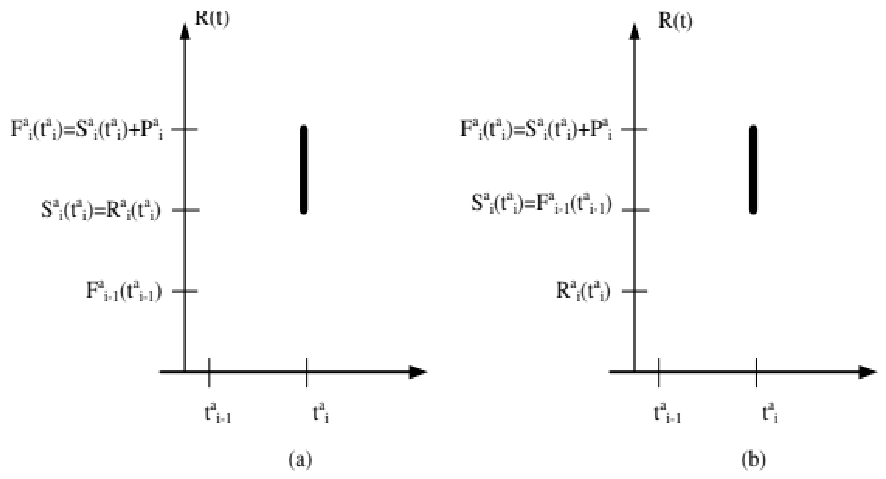

R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=R^a_i(t^a_i)$
$F^a_{i-1}(t^a_{i-1})$
t^a_{i-1}
t^a_i
(a)
R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=F^a_{i-1}(t^a_{i-1})$
$R^a_i(t^a_i)$
t^a_{i-1}
t^a_i
(b)

Figure 63: Transmission of a packet i of flow a arriving at time t_i^a of size P_i^a bits. The transmission starts at time $S_i^a = \max[F_{i-1}^a, R(t_i^a)]$ and ends at time $F_i^a = S_i^a + P_i^a$ with $R(t)$ the number of rounds of the algorithm. (a) The case $F_{i-1}^a < R(t_i^a)$. (b) The case $F_{i-1}^a \geq R(t_i^a)$.

$$F_i^a = S_i^a + P_i^a \quad \text{and} \quad S_i^a = \max[F_{i-1}^a, R(t_i^a)]. \tag{69}$$

The quantities $R(t), N_{active}(t), S_i^a$ and F_i^a depend only on the arrival time of the packets, t_i^a, and not on their transmission time, provided that a flow a is active as long as

$$R(t) \leq F_i^a \quad \text{when} \quad i = \max(j|t_j^a \leq t). \tag{70}$$

The authors of [102] use for packet-by-packet transmission time the following non-preemptive scheduling rule which emulates the BR strategy: *the next packet to be transmitted is the one with the smallest F_i^a.* A preemptive version of the algorithm requires that the transmission of the current packet be interrupted as soon as one with a shorter finishing time, F_i^a, arrives.

A fair allocation of the bandwidth does not have an effect on the timing of the transmission. A possible strategy is to allow less delay for the flows using less than their fair share of the bandwidth. The same paper [102] proposes the introduction of a quantity called the *bid*, B_i^a, and scheduling the packet transmission based on its value. The bid is defined as

$$B_i^a = P_i^a + \max[F_{i-1}^a, (R(t_i^a) - \delta)], \tag{71}$$

Start-time fair queuing:

A hierarchical CPU scheduler for multimedia operating systems was proposed in [142]. The basic idea of the *start-time fair queuing (SFQ)* algorithm is to organize the consumers of the CPU bandwidth in a tree structure; the root node is the processor and the leaves of this tree are the threads of each application. A scheduler acts at each level of the hierarchy; the fraction of the processor bandwidth, B, allocated to the intermediate node i is

$$\frac{B_i}{B} = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{72}$$

with $w_j, 1 \leq j \leq n$, the weight of the n children of node i, see the example in Figure 64.

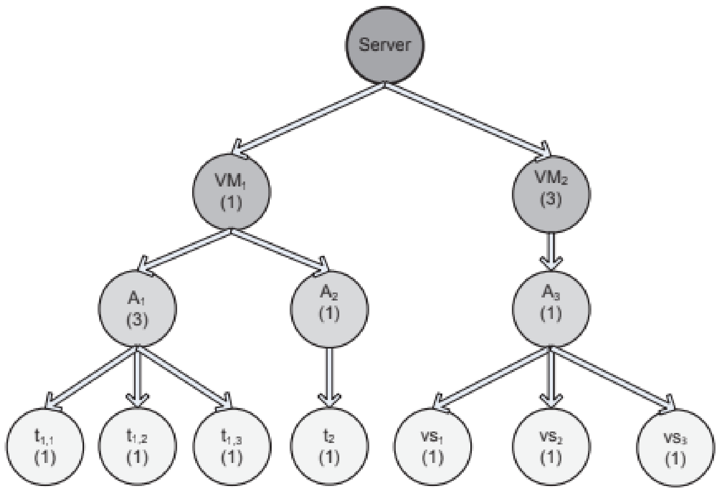


Figure 64: The SFQ tree for scheduling when two virtual machines VM_1 and VM_2 run on a powerful server. VM_1 runs two best-effort applications A_1, with three threads $t_{1,1}, t_{1,2}$, and $t_{1,3}$, and A_2 with a single thread t_2; VM_2 runs a video-streaming application A_3 with three threads vs_1, vs_2, and vs_3. The weights of virtual machines, applications, and individual threads are shown in paranthesis.

An SFQ scheduler follows several rules:

- (R1) The threads are serviced in the order of their virtual start up time; ties are broken arbitrarily.
- (R2) The virtual startup time of the i-th activation of thread x is

$$S_x^i(t) = \max\left[v(\tau^j), F_x^{(i-1)}(t)\right] \quad \text{and} \quad S_x^0 = 0. \tag{73}$$

  The condition for thread i to be started is that thread $(i-1)$ has finished and that the scheduler is active.
- (R3) The virtual finish time of the i-th activation of thread x is

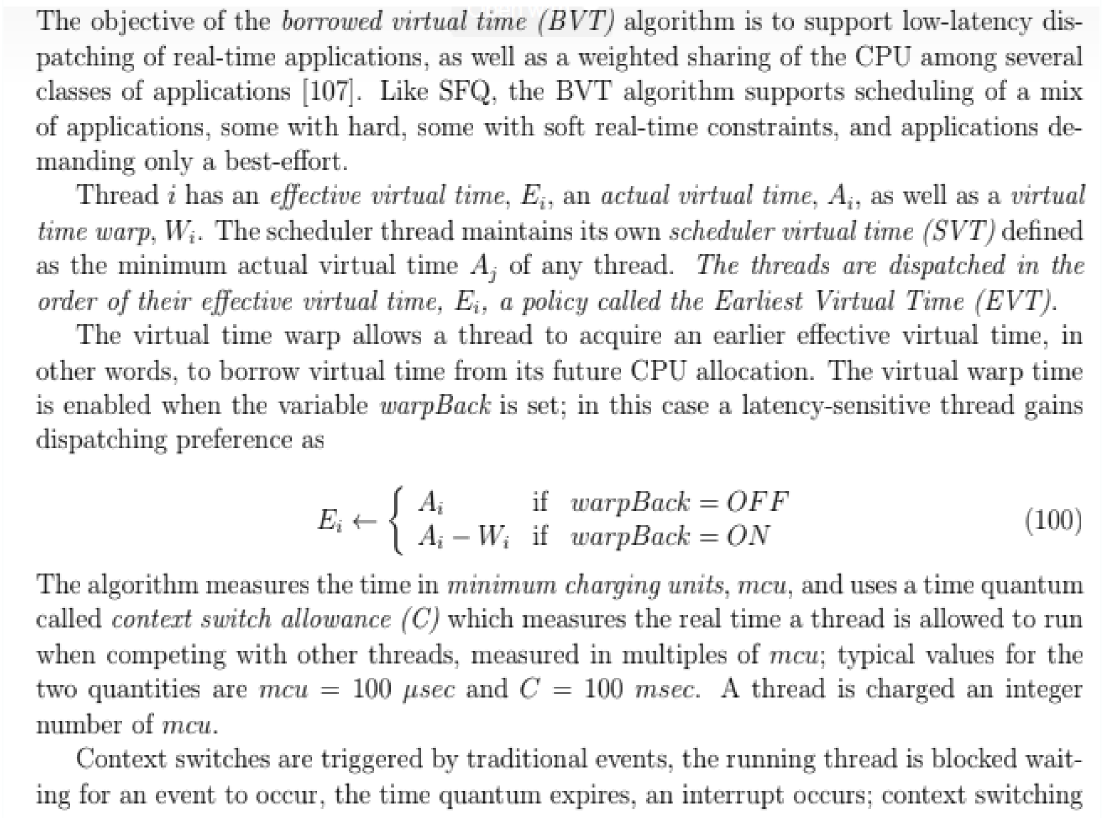
$$F_x^i(t) = S_x^i(t) + \frac{q}{w_x}. \tag{74}$$

  A thread is stopped when its time quantum has expired; its time quantum is the time quantum of the scheduler divided by the weight of the thread.
- (R4) The virtual time of all threads is initially zero, $v_x^0 = 0$. The virtual time $v(t)$ at real time t is computed as follows:

$$v(t) = \begin{cases} \text{Virtual start time of the thread in service at time } t, & \text{if CPU is busy} \\ \text{Maximum finish virtual time of any thread,} & \text{if CPU is idle} \end{cases}$$

7. **Describe the concept of a) Borrowed Virtual Time** **[05]** CO4 L2

Borrowed Virtual Time:



The objective of the *borrowed virtual time (BVT)* algorithm is to support low-latency dispatching of real-time applications, as well as a weighted sharing of the CPU among several classes of applications [107]. Like SFQ, the BVT algorithm supports scheduling of a mix of applications, some with hard, some with soft real-time constraints, and applications demanding only a best-effort.

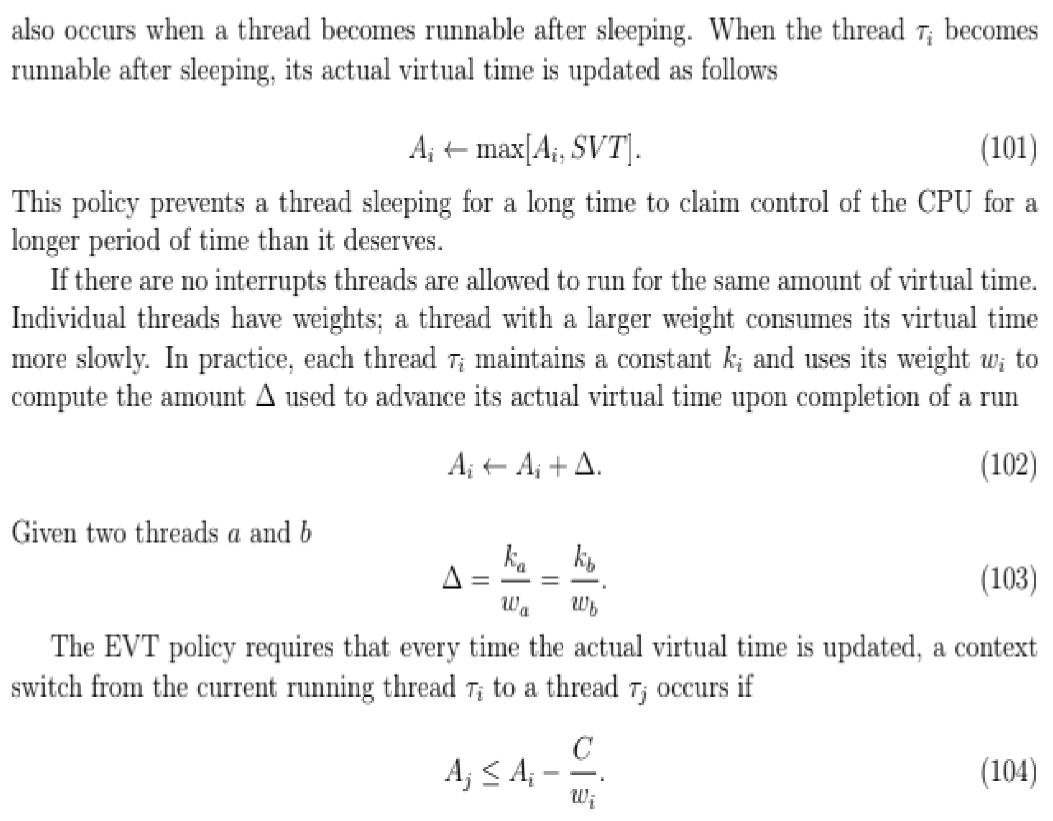
Thread i has an *effective virtual time*, E_i, an *actual virtual time*, A_i, as well as a *virtual time warp*, W_i. The scheduler thread maintains its own *scheduler virtual time (SVT)* defined as the minimum actual virtual time A_j of any thread. *The threads are dispatched in the order of their effective virtual time, E_i, a policy called the Earliest Virtual Time (EVT).*

The virtual time warp allows a thread to acquire an earlier effective virtual time, in other words, to borrow virtual time from its future CPU allocation. The virtual warp time is enabled when the variable *warpBack* is set; in this case a latency-sensitive thread gains dispatching preference as

$$E_i \leftarrow \begin{cases} A_i & \text{if} \quad warpBack = OFF \\ A_i - W_i & \text{if} \quad warpBack = ON \end{cases} \tag{100}$$

The algorithm measures the time in *minimum charging units*, *mcu*, and uses a time quantum called *context switch allowance (C)* which measures the real time a thread is allowed to run when competing with other threads, measured in multiples of *mcu*; typical values for the two quantities are $mcu = 100\ \mu sec$ and $C = 100\ msec$. A thread is charged an integer number of *mcu*.

Context switches are triggered by traditional events, the running thread is blocked waiting for an event to occur, the time quantum expires, an interrupt occurs; context switching



also occurs when a thread becomes runnable after sleeping. When the thread τ_i becomes runnable after sleeping, its actual virtual time is updated as follows

$$A_i \leftarrow \max[A_i, SVT]. \tag{101}$$

This policy prevents a thread sleeping for a long time to claim control of the CPU for a longer period of time than it deserves.

If there are no interrupts threads are allowed to run for the same amount of virtual time. Individual threads have weights; a thread with a larger weight consumes its virtual time more slowly. In practice, each thread τ_i maintains a constant k_i and uses its weight w_i to compute the amount Δ used to advance its actual virtual time upon completion of a run

$$A_i \leftarrow A_i + \Delta. \tag{102}$$

Given two threads a and b

$$\Delta = \frac{k_a}{w_a} = \frac{k_b}{w_b}. \tag{103}$$

The EVT policy requires that every time the actual virtual time is updated, a context switch from the current running thread τ_i to a thread τ_j occurs if

$$A_j \leq A_i - \frac{C}{w_i}. \tag{104}$$

b)Utility based model for cloud-based web services:

[05]	CO4	L2

A *utility function* relates the "benefits" of an activity or service with the "cost" to provide the service. For example, the benefit could be revenue and the cost could be the power consumption.

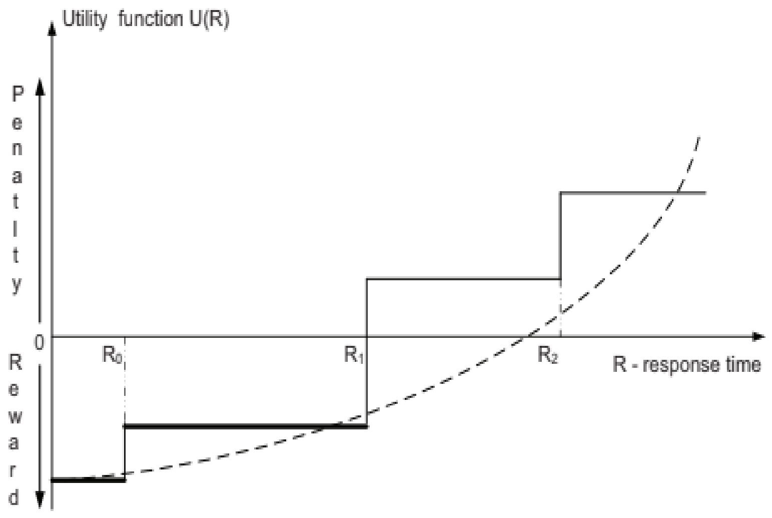


Figure 59: The utility function $U(R)$ is a series of step functions with jumps corresponding to the response time, $R = R_0|R_1|R_2$, when the reward and the penalty levels change according to the SLA. The dotted line shows a quadratic approximation of the utility function.

A service level agreement (SLA) often specifies the rewards as well as penalties associated with specific performance metrics. Sometimes the quality of services translates into average response time; this is the case of cloud-based web services when the SLA often specifies explicitly this requirement. For example, Figure 59 shows the case when the performance metrics is R, the response time. The largest reward can be obtained when $R \leq R_0$; a slightly lower reward corresponds to $R_0 < R \leq R_1$; when $R_1 < R \leq R_2$, instead of gaining a reward, the provider of service pays a small penalty; the penalty increases when $R > R_2$. A utility function, $U(R)$, which captures this behavior is a sequence of step functions; the