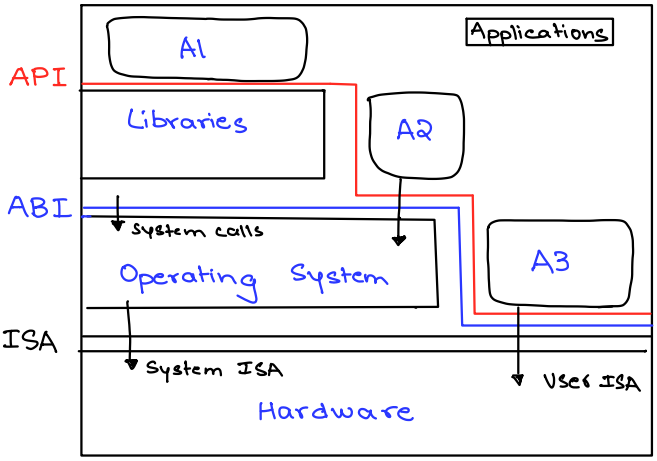
UNIT 3

LAYERING

- managing system complexity - layering
- minimizes interactions
- 2 modes → user & privileged



- 1st interface – ISA – boundary b/w s/ware & h/ware – processor's set of instructions
- ABI (App^n Binary Int.) – H/ware access to the app^n's & library modules. Doesn't include privileged instructions.
- API – defines set of instr the h/ware was designed to execute – gives app^n access to the ISA.

CONDITIONS FOR EFFICIENT VIRTUALIZATION

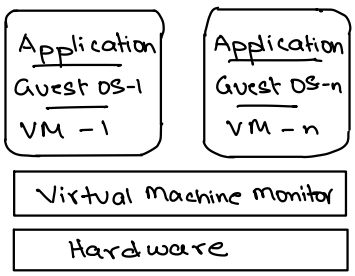
① A program running under the VMM must exhibit behaviour essentially similar to that demonstrated when the app^n runs directly on an equivalent machine.

② The VMM should be in complete control of the virtualized resources.

③ A statistically significant fraction of the machine instructions should be executed without the intervention of the VMM.

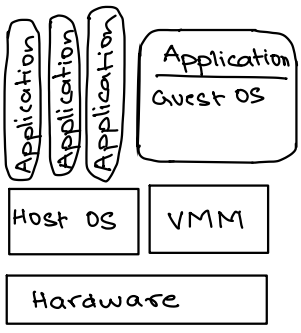
TRADITIONAL

- Thin layer of s/ware that runs directly on the host machine's h/ware
- Main adv. is performance



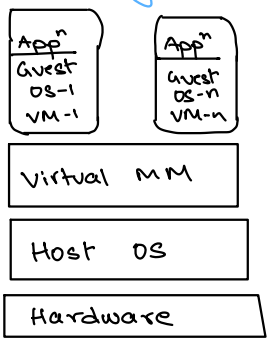
HYBRID

- Shares the h/ware w/ the existing OS.



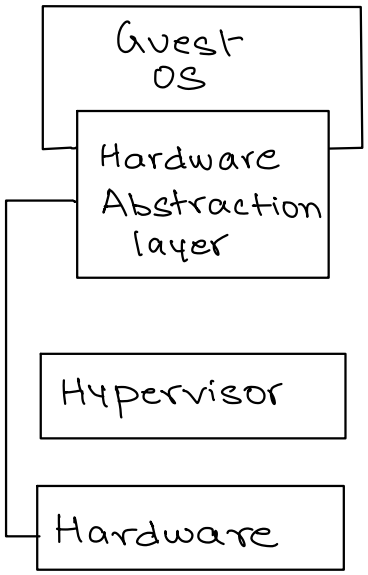
HOSTED

- The VM runs directly on top of the existing OS.



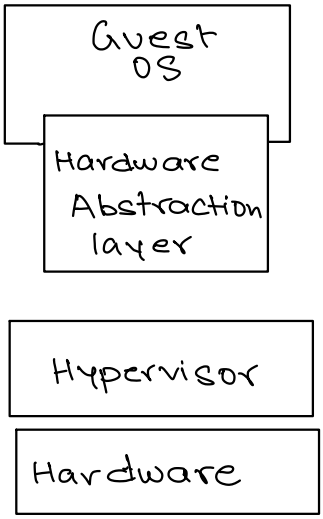
FULL VIRTUALIZATION

- Each VM runs on an exact copy of the actual hlware.
- Requires a fully virtualizable architecture; the hlware is fully exposed to the guest OS, that runs unchanged.



PARAVIRTUALIZATION

- VM runs on a modified copy of the actual hlware.
- It is done as some architectures are not easily virtualizable.
- Demands that the guest OS be modified to run under the VMM.



PROBLEMS FACED BY VIRTUALIZATION OF THE x86 ARCHITECTURE

① RING DEPRIVILEGING

VMM forces the guest sware, the OS, & the app's to run at a privilege level > 0 (not in 64 bit mode)
solns → (0/1/3) mode
→ (0/3/3) mode

② RING ALIASING

Problem when a guest OS is forced to run at a privilege level that it wasn't designed for.

③ ADDRESS SPACE COMPRESSION

VMM uses parts of the guest address space to store system DSs. These DS must be protected, but the guest sware must have access to them.

④ INTERRUPT VIRTUALIZATION

VMM generates a "virtual interrupt" in response to a physical interrupt & delivers it to the target guest OS. Guest OS's have the ability to mask interrupts - complicates the VMM & increases overhead.

⑤ ACCESS TO HIDDEN STATE

Elements of the system state are hidden; there is no mechanism for saving & restoring hidden components after a context switch from 1 VM to another

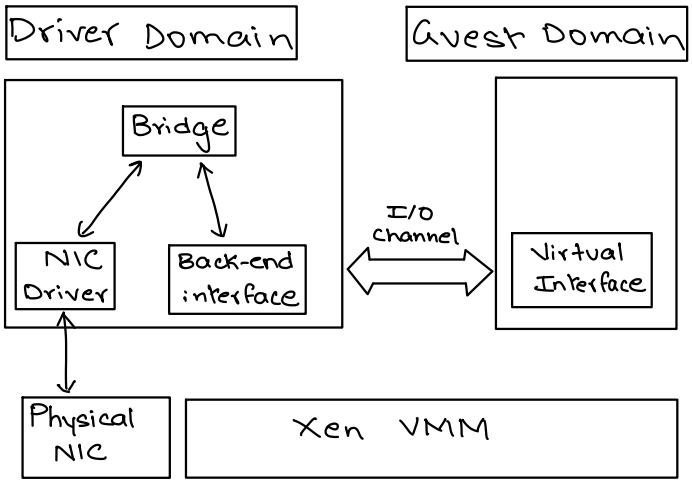
⑥ FREQUENT ACCESS TO PRIVILEGED RESOURCES INCREASES VMM OVERHEAD

The *TPR is frequently used by a guest OS. the VMM must protect access to this reg. & trap all attempts to access it. this can cause performance degradation.

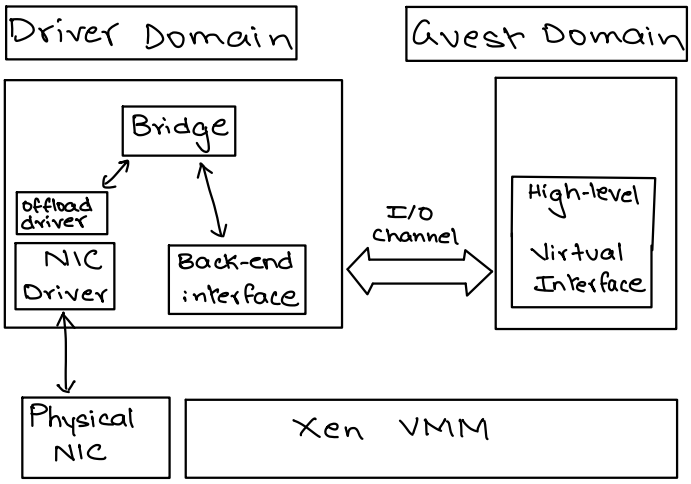
*TPR - Task Priority Register

XEN ARCHITECTURE

ORIGINAL



OPTIMIZED



XEN NETWORK OPTIMIZATIONS

(1) THE VIRTUAL INTERFACE

- The original " " network provides the guest domain w/ the abstraction of a simple low-level network interface supporting sending & receiving primitives.
- This design supports a wide range of physical devices attached to the driver domain but doesn't take advantage of the capabilities of some physical devices.
- These features are supported by the high-level virtual interface of the optimized system

(2) THE I/O CHANNEL

- Rather than copying a data buffer holding a packet, each packet is allocated in a new page & this physical page is remapped to the guest domain.
- This strategy contributes to a better than 4x increase in the send data rate.

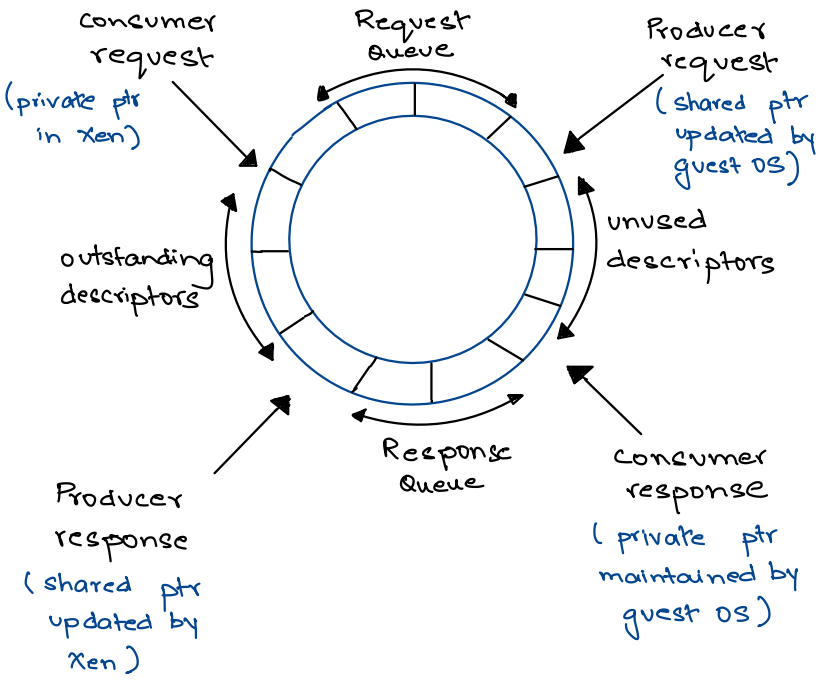
(3) VIRTUAL MEMORY

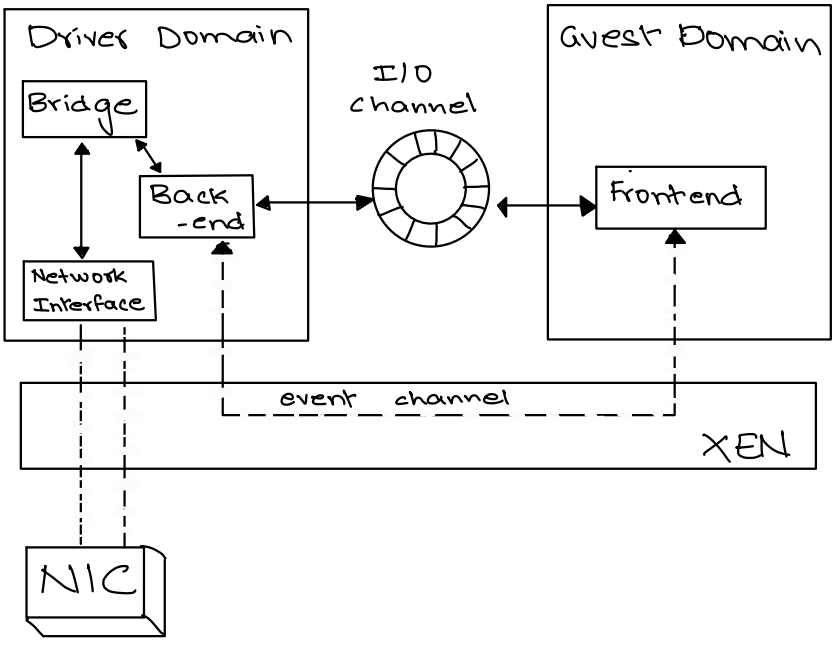
- " " in Xen 2.0 takes advantage of superpage & global page-mapping hardware features.
- A superpage increases the granularity of dynamic address translation – a superpage entry covers 1024 pages of physical memory, & address translation maps a set of contiguous pages to a set of contiguous physical pages.
- This reduces the no. of *TLB misses.
- The optimized version uses a special memory allocator to avoid the problem where the system is forced to use traditional page-mapping rather than superpage mapping

*Translation lookaside buffer: memory cache used to reduce the time taken to access a user memory location.

XEN ZERO-COPY SEMANTICS FOR DATA TRANSFER USING I/O RINGS

CIRCULAR RING OF BUFFERS





* IR → Instruction Register
* RSE → Register Stack Engine

vBlades project

- The Itanium processor supports 4 privilege rings – PL0, PL1, PL2, PL3.
- Privileged instructions can only be executed by the kernel at PL0.
- App^n^s run at PL3 & can execute only non-privileged instructions.
- The VMM uses ring compression & runs itself at PL0 & PL1 while forcing a guest OS to run at PL2.
- A problem – privilege leaking – several nonprivileged instructions allow an app^n^ to determine the current privilege level (CPL).
- Itanium was selected because of its multiple functional units & multithreading support.

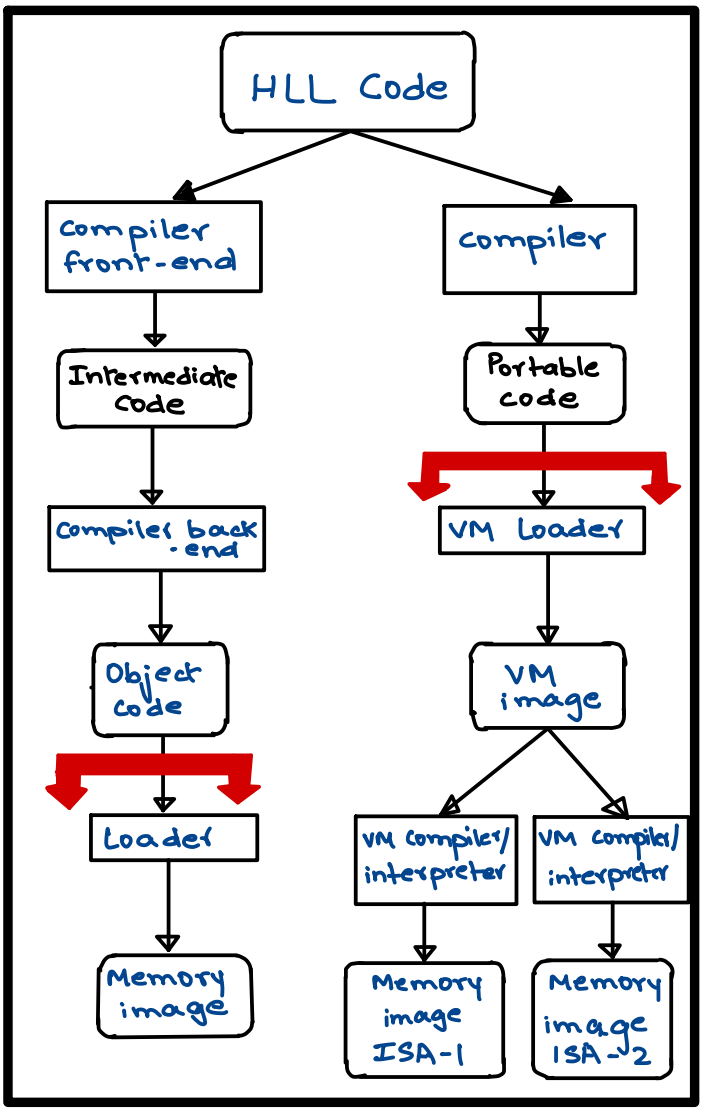
- CPU Virtvalization:
  when a guest OS attempts to execute a privileged instruction the VMM traps & emulates the instruction.
  - Complication – Itanium doesn't have an *IR & the VMM has to use state info to determine whether an instr. is privileged.
  - Complication – caused by *RSE, which operates concurrently wl the processor & may attempt to access memory & generate a page fault.

- A no. of privileged-sensitive instr. behave differently as a function of the privilege level. The VMM replaces each one of them wl a privileged instr. during the dynamic transformation of the instruction stream.

contd.

contd.

- Memory virtualization is guided by the realization that a VMM should not be involved in most read & write operations to prevent a significant degradation of performance, but at the same time the VMM should exercise tight control & prevent a guest OS from acting maliciously.
- The vBlades VMM doesn't allow a guest OS to access memory directly. It inserts an additional layer of indirection called metaphysical addressing b/w virtual & real addressing.
- A guest OS is placed in metaphysical addressing mode. If the address is virtual, the VMM first checks if the guest OS is allowed to access that address, & if it is, it provides the regular address translation. If the address is physical, the VMM is not involved.



UNIT-4

CLOUD RESOURCE MANAGEMENT - POLICIES

① **Admission Control**

The goal of this policy is to prevent the system from accepting workloads in violation of high-level system policies — eg: a system may not accept an additional workload that would prevent it from completing work already in progress. Limiting the workload requires some knowledge about the global state of the system.

② **Capacity Allocation**

It means to allocate resources for individual instances; an instance is an activation of a service

③ **Load Balancing**, ④ **Energy Optimization**

- It refers to evenly distributing the load to a set of servers. In CC, a critical goal is minimizing the cost of providing the service, & , in particular, minimizing the energy consumption.
- This gives us a diff meaning of LB - instead of having the load equally distributed to all servers, we want to concentrate it & use the smallest no. of servers while switching the others to standby mode(server uses less energy).
- LB & energy optimization are correlated & affect the cost of providing the service.

⑤ **QoS guarantees**

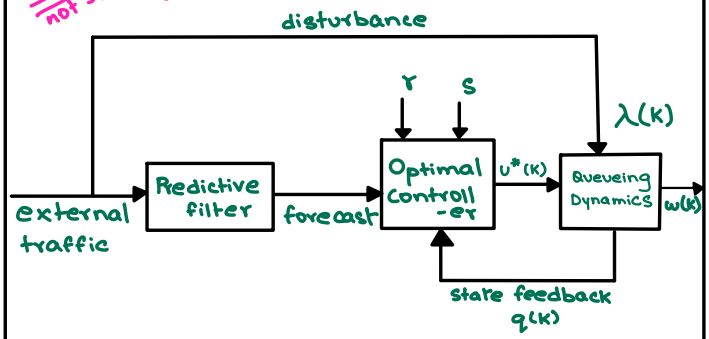
QoS is that aspect of resource management that is probably the most difficult to address &, at the same time, possibly the most critical to the future of CC.

(diagram)

- This is the case of a single processor serving a stream of input requests
- To compute the optimal inputs over a finite horizon, the controller uses feedback regarding the current state, as well as an estimation of the future disturbance due to the environment.

CONTROL THEORY PRINCIPLES — OPTIMAL RESOURCE ALLOCATION

not sure ↘



→ Optimal control generates a sequence of control inputs over a look-ahead horizon while estimating changes in operating conditions.

→ A **convex cost function** has arguments - x(K), the **state** at step K, & u(K), the **control vector**; this cost function is minimized, subject to the constraints imposed by system dynamics.

→ The discrete-time optimal control problem is to determine the sequence of control variables u(i), u(i+1), .. u(n-1) to minimize the expression:

$$J(i) = \phi(n, x(n)) + \sum_{K=i}^{n-1} L^K(x(K), u(K))$$

$\phi(n, x(n))$: cost function of the final step

$\sum_{K=i}^{n-1} L^K(x(K), u(K))$: Time-varying cost function at the intermediate step K ova [i,n]

→ The minimization is subject to :

$$x(K+1) = f^K(x(K), u(K))$$

$x(K+1)$: system state at time K+1; f^K: function of; $x(K)$: state at time K; $u(K)$: input at time K

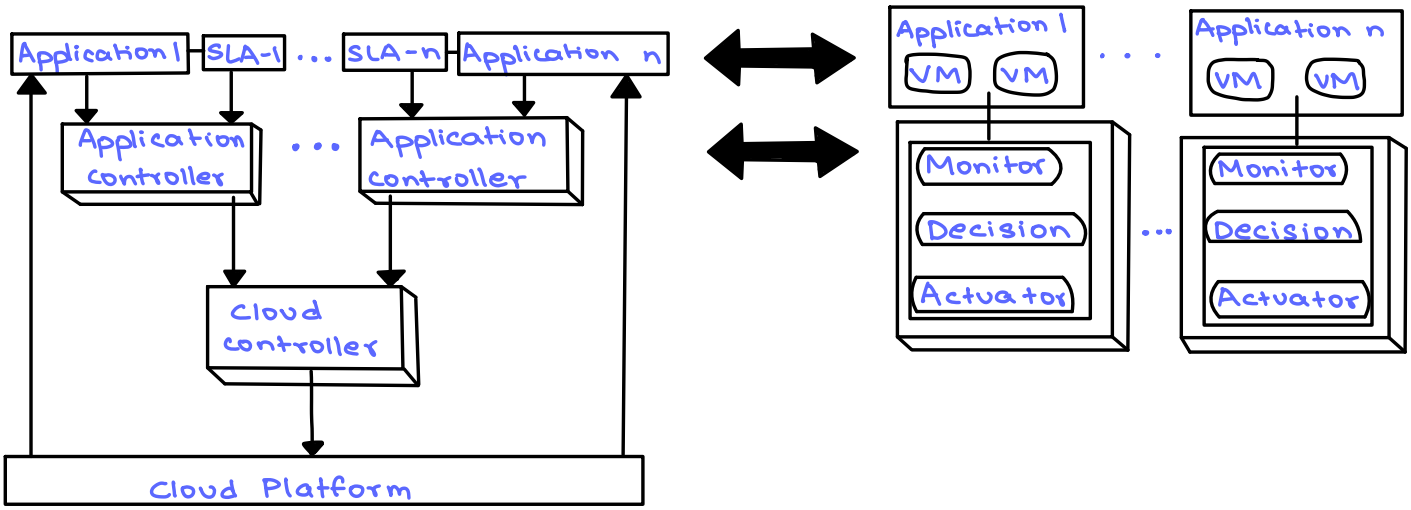
→ A technique to solve this problem - Lagrange multiplier(λ) method to maximize a function $g(x,y)$ subject to constraint $h(x,y) = K$

$$\Lambda(x, y, \lambda) = g(x,y) + \lambda \times [h(x,y) - K]$$

→ A necessary condition for optimality is that (x, y, λ) is a stationary point for $\Lambda(x, y, \lambda)$, or,

$$\nabla_{x,y,\lambda} \Lambda(x, y, \lambda) = 0$$

TWO-LEVEL CONTROL ARCHITECTURE

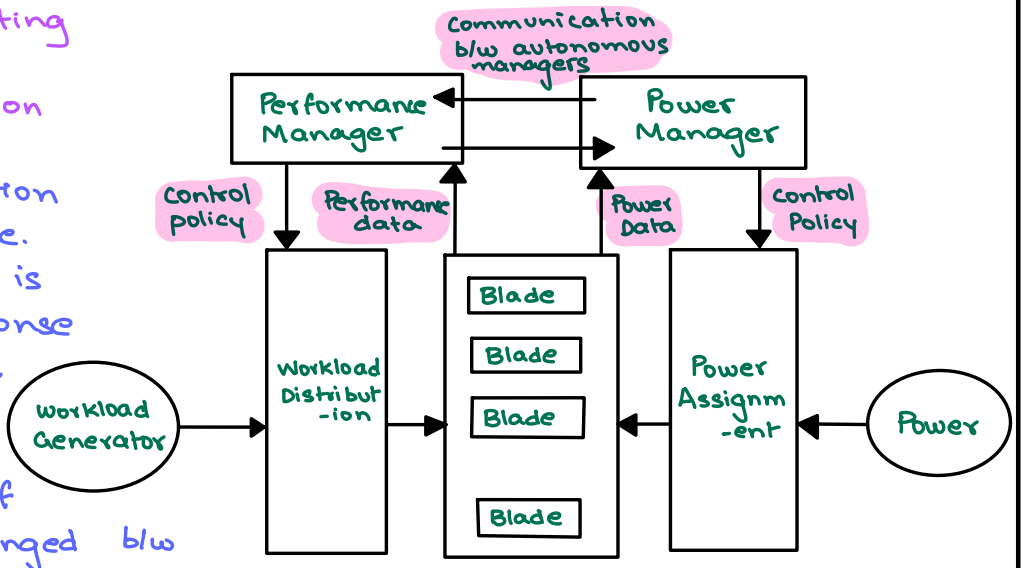


- The automatic resource management is based on 2 levels of controllers – one for the service provider & one for the application.
- The main components of a control system are the inputs, the control system components, & the outputs.
- The inputs in such models are the offered workload & the policies for admission control, the capacity allocation, the load balancing, the energy optimization, & the QoS guarantees in the cloud.
- The system components are sensors used to estimate relevant measures of performance & 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.

COORDINATION OF AUTONOMIC PERFORMANCE MANAGERS

* The approach to coordinating power & performance management is based on several ideas:
- Use a joint utility function for power & performance. This function, $U_{pp}(R,P)$ is a function of the response time R, & the power P.

$$U_{pp}(R,P) = \frac{U(R)}{P}$$



- Identify a minimal set of parameters to be exchanged b/w the 2 managers.
- Set up a power cap for individual systems based on the utility-optimized power management policy
- Use a std performance manager modified only to accept input from the power manager regarding the frequency determined according to

contd.

the power management policy.

- Use standard software systems.

utility function $U'(p_k, n_c) = U_{pp}(R(p_k, n_c), P(p_k, n_c))$

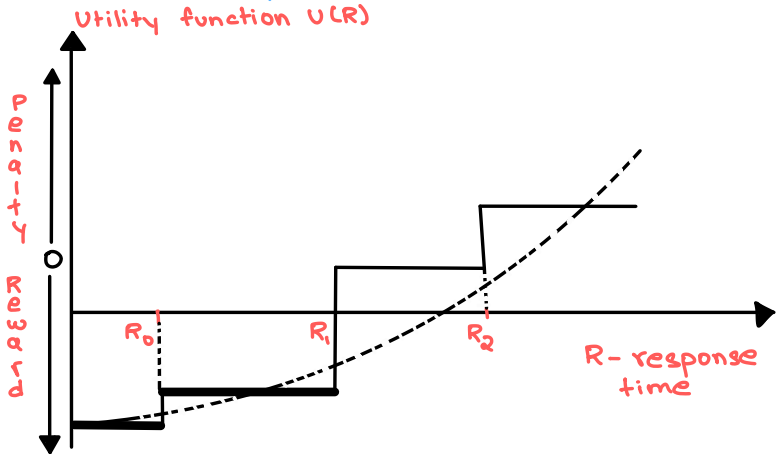
p_k: power cap; n_c: No. of clients

UTILITY FUNCTION – PERFORMANCE METRIC : RESPONSE TIM (R)

→ A utility function relates the "benefits" of an activity or service wl the "cost" to provide the service

→ A service-level-agreement (SLA) often specifies the rewards & penalties associated wl specific performance metrics.

When the performance metric is R: (R_0, R_1, R_2)



→ The largest reward can be obtained when $R \leq R_0$; a slightly lower reward corresponds to $R_0 < R \leq R_1$.

→ when $R_1 < R < R_2$, instead of gaining a reward, the provider of the 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 utility function is sometimes approximated by a quadratic curve.

PRICING & ALLOCATION ALGORITH

This algo partitions the set of users into 2 disjoint sets — winners (W) & losers (L).

The algo should:

- ★ Be computationally tractable.
- ★ Scale well
- ★ Be objective. Partitioning should only be based on the price Π_u of a user's bid.
- ★ Be fair. Make sure that the prices are uniform. All winners wlin a given resource pool pay the same price.
- ★ Indicate clearly at the end of the auction the unit prices for each resource pool.
- ★ Indicate clearly to all participants the relationship b/w the supply & the demand in the system

ACSA COMBINATORIAL AUCTION ALGORITHM

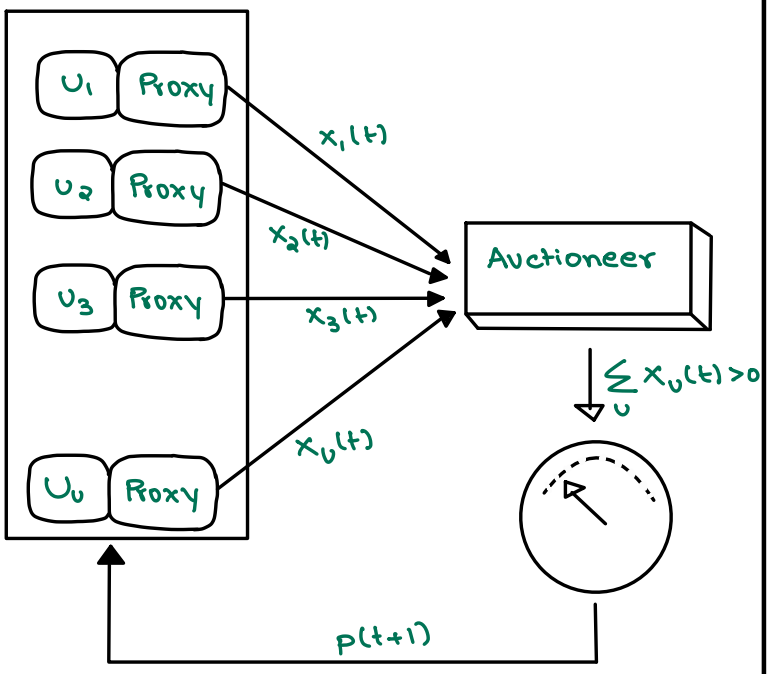
→ Here, the participants at the auction specify the resource & the quantity of that resource offered or desired at the price listed for that time slot.
→ Then the excess vector is calculated. If all its components are -ve, the auction stops. (Demand doesn't exceed the offer).
→ If Demand > Offer, the auctioneer increases the price for items & solicits bids at the new price.

Schematics:

$x_u(t) \rightarrow$ bids

$\sum_u x_u(t) \rightarrow$ excess demands

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, & in that case, raises the price of those resources.



FAIR QUEUEING

→ It introduces a bit-by-bit round-robin (BR) strategy; a single bit from each queue is transmitted & the queues are visited in a round-robin fashion.

$R(t)$ → No. of rounds of the BR algo up to time t

$N_{active}(t)$ → No. of active flows through the switch

t_i^a → Time when packet i of flow a, of size P_i^a arrives

S_i^a → Value of $R(t)$ when the 1st bit of packet i of flow a is transmitted

F_i^a → Value of $R(t)$ when the last bit of packet i of flow a is transmitted

then,

$$F_i^a = S_i^a + P_i^a$$
$$S_i^a = \max[F_{i-1}^a, R(t_i^a)]$$

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

$$R(t) \leq F_i^a,$$
$$\text{when } i = \max(j \mid t_j^a \leq t)$$

→ The next packet to be transmitted is the one w/ the smallest F_i^a.

START-TIME FAIR QUEUEING

* The basic idea is to organize the consumers of the CPU bandwidth in a tree-structure; the root node is the processor & the leaves are the threads of each application.
* A schedular acts at each level of the hierarchy. The fraction of the processor bandwidth, B, allocated to the intermediate node i is:

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

(w_j: weight of the n children of the node i)

Rules an SFQ schedular follows:

R1: The threads are serviced in the order of their virtual start-up time

R2: The virtual start-up time of the i-th activation of thread x is:

$S_x^i(t) = \max[v(t^i), F_x^{(i-1)}(t)]$ & $S_x^0 = 0$

The condition for thread i to be started is that thread (i-1) has finished & that the scheduler is active.

R3: The virtual finish time of the ith activation of thread x is

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

(q: Time quantum of the thread)

A thread is stopped when its time quantum has expired.

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:

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

SUM - SFQ

Q. There are 2 threads w/ weights $w_a = 1$ & $w_b = 4$. Time quantum $q = 12$. (Consider scheduling decisions at t=0 & t=3)

(pg 187-189 in the tb for more values of t)

Initially, $S_a^0 = 0$, $S_b^0 = 0$

$v_a(0) = 0$, $v_b(0) = 0$

① t = 0

$S_a^0 = S_b^0$ ⇒ tie

Thread b is arbitrarily chosen to run first.

The virtual finish time of thread b is:

$$F_b^0 = S_b^0 + \frac{q}{w_b}$$

$$= 0 + \frac{12}{4} = 3$$

② t = 3

Both threads are runnable & thread b was in service; thus, $v(3) = S_b^0 = 0$; then

$S_b^1 = \max[v(3), F_b^0]$

$= \max(0, 3)$

$= 3$

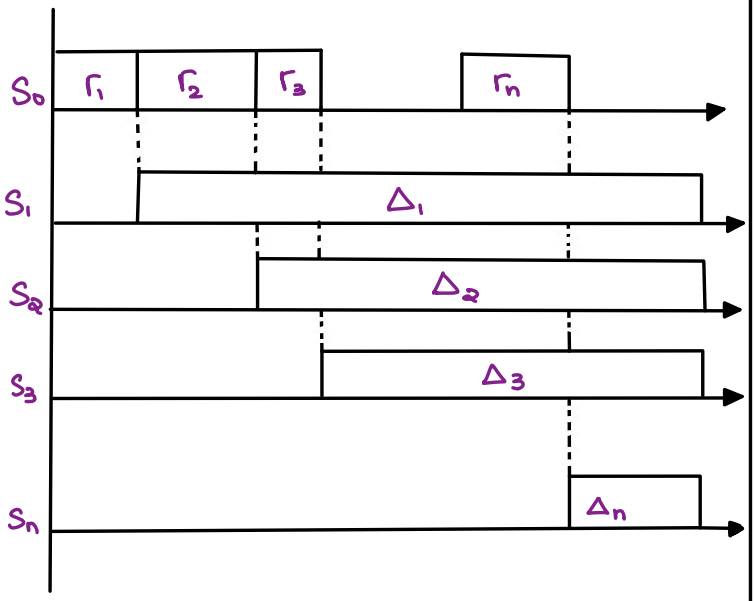
But $S_a^0 < S_b^1$, thus thread a is selected to run. Its virtual finish time:

$$F_a^0 = S_a^0 + \frac{q}{w_a}$$

$$= 0 + \frac{12}{1} = 12$$

OPTIMAL PARTITIONING RULE

- The workload is partitioned to ensure the earliest possible completion time
- The algorithm requires worker nodes to complete to complete execution at the same time.
- The head node, S_0, distributes sequentially the data to individual worker nodes.



EQUAL PARTITIONING RULE

- EPR assigns an equal workload to individual worker nodes.
- The head node, S_0, distributes sequentially the data to individual worker nodes.