25-Feb-2022

CS565 Midsem Assignment

Name: M. Maheeth Reddy Roll No. 1801CS31

Objectives

1. (d) All of the above
2. (a) Distributed System; Message Passing
3. (b) False
4. (a) Utility Computing
5. (d) Public
6. (d) All of the above
7. (d) All of the above
8. (d) All of the above
9. (d) All of the above
10. (d) All of the above

11. Problems with Trap-and-Emulate process virtualization

(*) Guest OS may realize it is running at lower privilege level.
- → Some registers in X86 reflect CPU privilege level
- → Guest OS can read these values and get offended

(*) Some X86 instructions which change hardware state run in both privileged and unprivileged modes
- → They will behave differently when guest OS in ring 0 vs ring 1, which is less privileged.
- → OS behaves incorrectly in ring 1 will not trap to VMM
- → Instruction set of X86 is not easily visualizable

(*) E Flags register is a set of CPU flags. Consider the popf instruction in X86
- → Executed in ring 0, all flags set normally.
- → Executed in ring 1, only some flags set.

So, popf is a sensitive instruction, not privileged, doesn't trap, behaves differently when executed in different privilege levels, Guest OS is buggy in ring 1.

→ V-rings, no root / non-root modes yet

→ Hypervisor in ring 0, guest OS in ring 1

→ HV doesn't know, so it doesn't try to change setting

→ OS doesn't know, so assumes change was successful.

Binary Translation

In binary translation, we rewrite VM binary to never issue these 17 instructions which causes problems in Trap-and-Emulate.

Goal: Full Virtualization = Guest OS not modified.

Approach: Dynamic Binary Translation

(P.T.O)

1) Inspect code blocks to be executed

2) If needed, translate to alternate instruction sequence

Eg: To emulate desired behaviour, possibly even avoiding trap

3) Otherwise, run at hardware speeds and cache translated blocks to amortize translation costs.

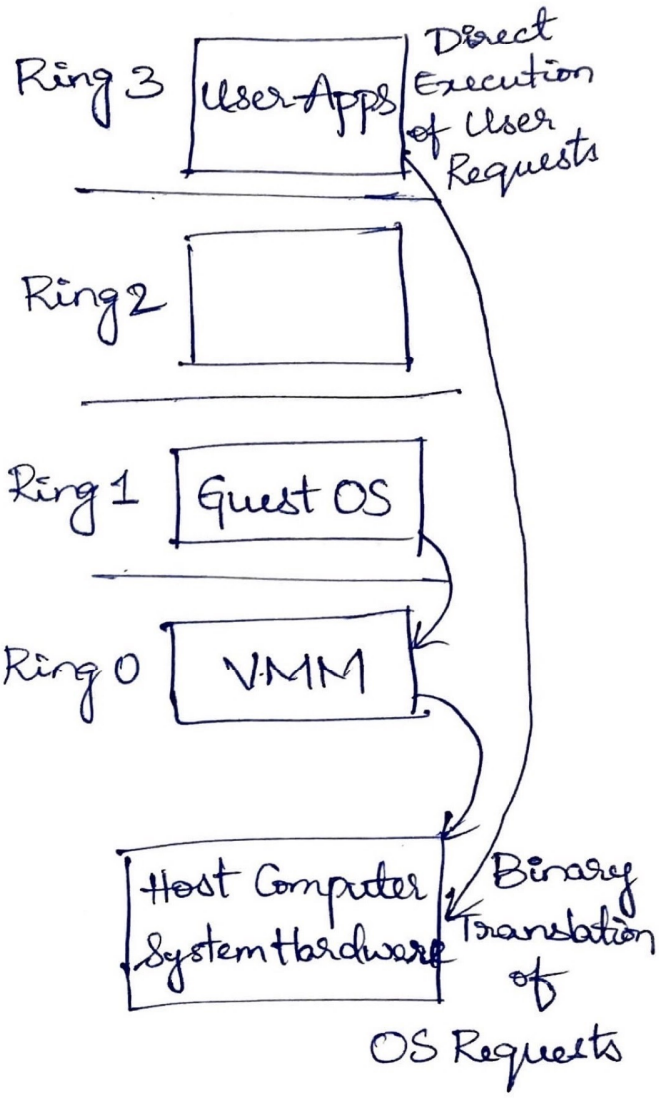


Fig: Binary Translation approach to X86 virtualization.

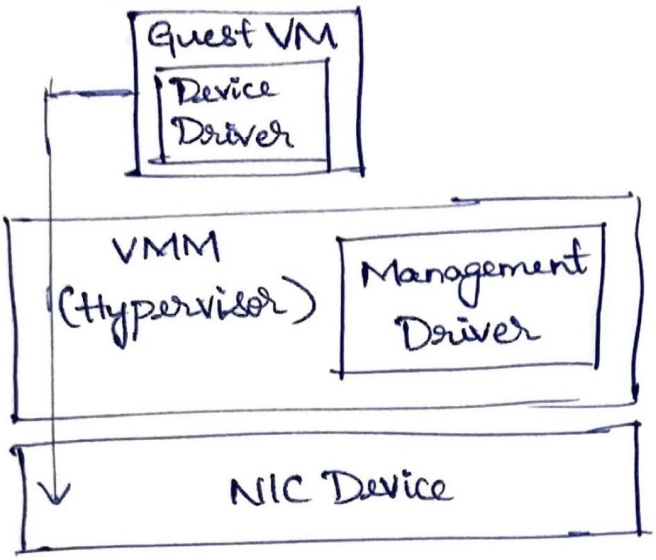
12. There are three key models for device virtualization:

a) Pass through model: VMM level drives configures device access permissions.

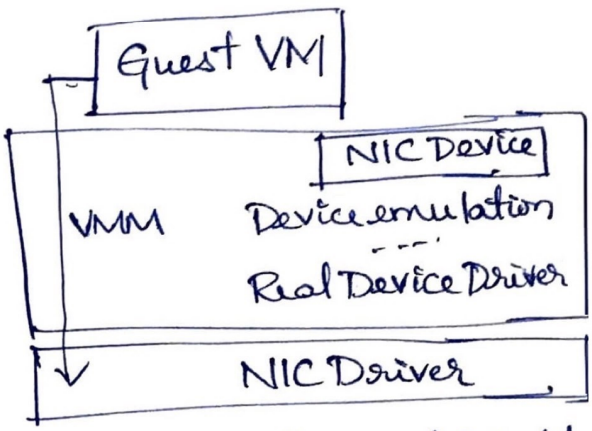
- (*) VM provided with exclusive access to the drive
- (*) VM can directly access the device
- (*) VMM must have exact type of device as what VM expects

b) Hypervisor Direct Model: VMM intercepts all device accesses.

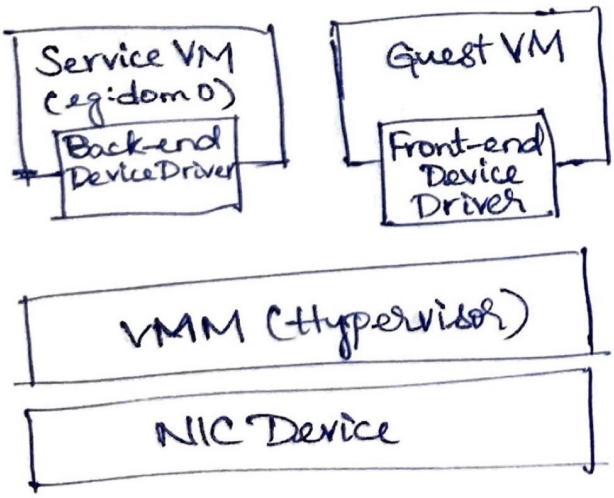
- (*) Emulates device operation.
  - → Translate to generic I/O operation
  - → Transverse VMM-resident I/O stack
  - → Invoke VMM-resident drivers.



Pass-through Model



Hypervisor Direct Model



Split-Device Driver Model

c) Split-Device Driver Model:

Device access control are split between:

(*) Front End driver in guest VM

and

(*) Backend End driver in service VM

13. Linux Containers:

A linux container is a set of one or more processes that are isolated from the rest of the system. Containers run in host system kernel, seperated with policies and can use apps in the host system.

Linux Containers are portable between OS variants supporting linux containers and there is no overhead with hypervisor and guest OS kernel.

Advantages of using containerized apps over Vm-based apps:

① Containers are independent hosts for applications that use a single, stripped-down version of OS to run whereas VM's use full version of an OS.

(2) Containers run a virtualized workload, processed by an application broken up into microservices, making them more lightweight and flexible than a VM.

(3) Containers can scale up and down applications very easily.

(4) Containers are highly suitable for a microservices architecture, in which applications are broken into small, self-sufficient components which can be deployed and scaled individually. Containers are an attractive option for deploying and scaling each of these microservices.

(5) Containers provide for more flexibility and portability than VMs in multi-cloud environments. When software components are deployed in containers, it is possible to easily "lift & shift" those containers from onpremise bare metal sovers to virtualized & public-cloud environments.

(6) Containers are easily controlled by API and thus are also ideal for automation and CI/CD pipelines.

14. Hotspot Detection:

Sandpiper implements a hotspot detection algorithm that determines when to migrate virtual machines.
The hotspot detection component employs a monitoring & profiling engine that gathers usage statistics on various virtual & physical servers and constructs profiles of resource usage. These profiles are used in conjunction with prediction techniques to detect hotspots in the system. When there is increased workload on a server, it should be simply migrated to a less loaded server. Manually initiated migration lacks the agility to respond to sudden workload changes. So, detecting hotspots is an important step to flexibly remap physical resources to virtual servers.

Hotspot Mitigation:

Upon detecting hotspots, sandpiper's migration manager is invoked for hotspot mitigation. This hotspot mitigation algorithm resorts to a heuristic to determine which overloaded VMs to migrate and where, such that migration overhead is minimized.

15. Given,

(i) Desired mean response time, $d = 1.5s$

(ii) The 10 most recent service times:

1.1s, 0.9s, 1.2s, 0.8s, 1.3s, 0.7s, 1.4s, 0.6s, 1.2s, 0.8s

(iii) Inter-arrival time follows Gamma Distribution,

$$\text{Gamma}(a,b) = \text{Gamma}(2,1)$$

(iv) Peak Request Arrival Time, $\lambda_{peak} = 1$ request per second.

As λ_{peak} is given, we need to find λ_{cap}, the request arrival rate. λ_{cap} follows the inequality below,

(P.T.O)

$$d_{cap} \geq \frac{1}{\left(S + \frac{\sigma_a^2 + \sigma_b^2}{2(d-S)}\right)} \quad \text{---(1)}$$

where, $S \longrightarrow$ mean service time

$\sigma_a^2 \longrightarrow$ variance of inter arrival time

$\sigma_b^2 \longrightarrow$ variance of service time

Mean Service Time, S = Mean of the 10 recent service times

$$= \frac{1.1+0.9+1.2+0.8+1.3+0.7+1.4+0.6+1.2+0.8}{10}$$

$$= 1 \text{ second}$$

Variance of inter arrival time, $\sigma_a^2 = ab^2 = 2 \times 1^2 = 2$

[$\because$ Variance of Gamma$(\alpha, \beta) = \alpha\beta^2$]

Variance of Service time, σ_b^2

$$= \left\{\begin{array}{l}\text{Mean of squares} \\ \text{of service time}\end{array}\right\} - \left\{\begin{array}{l}\text{Square of mean} \\ \text{of service time}\end{array}\right.$$

$$= \begin{array}{l}1.068 \\ \text{(after calculation)}\end{array} - 1.$$

$$\sigma_b^2 = 0.068$$

We have, $d = 1.5$ $s = 1$ $\sigma_a^2 = 2$ $\sigma_b^2 = 0.068$

From (1)

$$\Rightarrow d_{cap} \geq \frac{1}{1 + \frac{2 + 0.068}{2(1.5-1)}} = \frac{1}{1 + \frac{2.068}{1}} = \frac{1}{3.068}$$

$$\Rightarrow \boxed{d_{cap} \geq \frac{1}{3.068}} \quad \text{---(2)}$$

The factor by which CPU capacity must be increased is given by $\frac{d_{peak}}{d_{cap}} = \frac{1}{d_{cap}}$

From (2) $\Rightarrow d_{cap} \geq \frac{1}{3.068} \Rightarrow \boxed{\frac{1}{d_{cap}} \leq 3.068}$

∴ The current CPU capacity of the server should be scaled by a maximum factor of 3.068 for servicing peak request arrival time of 1 request per second.

16. The microservice software architecture allows a system to be divided into a number of smaller, individual and independent services. Each service is flexible, robust and complete. They run as autonomous processes and communicate with one another through APIs. Each microservice is composed of different codes in different programming languages, databases and software ecosystems. Through microservices, applications are easier to build, optimize and maintain when they are divided into a set of smaller parts. Some of the key advantages of microservices are:

① Independence: Microservices grant the developers more independence to work autonomously and make technical decisions quickly in smaller groups.

② Enhanced speed & productivity: The microservices architecture handles the problem of speed and

productivity by splitting applications into manageable services that are fast enough to develop.

③ Resilience: An application will still function if part of it goes down because microservices enable for spinning up a replacement.

④ Improve fault isolation: Larger applications can remain mostly unaffected by the failure of a single module.

⑤ Scalability: Microservices require fewer resources to scale the necessary components.

⑥ Lifecycle Automation: The individual components of microservices can more easily fit into continuous delivery pipelines when monoliths increase complexities.

⑦ Ease of understanding: With added simplicity of microservices, developers can better understand the functionality of a service.

17. Factors that should be considered while choosing a right cloud service provider :

① Regular Compliance : Your business processes running on cloud based system are compliant with the standards which your customers require.

② Adaptable to Infrastructure Requirements : Provider should understand your infrastructure requirements and offer solutions corresponding to that without compromising on business continuity.

③ Favourable service level agreements : Establishes a relationship between a user & a cloud service provider.

④ Different types of Cloud services : Selecting the service that is compatible with business requirement.

⑤ Security : Cloud service should ensure that your data is safe and can traverse across different platform, resilient with AUTHENTICATION and ease of availability

⑥ Should provide good Customer Service, and.

⑦ Flexibility in Pricing Plans

18. Black-Box Monitoring:

It is entirely OS and application agnostic. Therefore, it uses the profile and the information from outside the VM. It depends on the Xen hypervisor.

It monitors servers with a focus on areas such as disk space, CPU usage, memory usage, load averages etc. Testing externally visible behaviour as a user would see it. These are deemed as the standard system metrics to monitor, in the industry.

Gray-Box Monitoring:

It can be supported, when feasible using a light-weight monitoring daemon that is installed inside each virtual server. In linux, the monitoring daemon user the /proc interface to gather OS level statistics of CPU, network, and memory usage. The memory usage monitoring, in particular, enables proactive detection and mitigation of memory hotspots. The monitoring daemon also can process logs of applications such as web & database servers to derive statistics such as request rate, request drops and service times.

19. Two approaches to address the problems of networking VMs. are:
a) Using Specialized Hardware (Hardware based approach)
b) Software based Approach.

Comparision

	Hardware Based Approach	Software-Based Approach
①	SR-IOV, single root I/O virtualization	Open vSwitch is used.
②	Sacrifices flexibility & forwarding logic	Focuses on forwarding flexibility
③	Hits native performance	The compromise made is to avoid targeting worst case performance
④	The main idea behind this is CPUs are not designed to forward packets but the NIC is	Smarts of this approach is the switch routing decisions lie in user space.
⑤	SR-IOV, the physical link itself supports virtualiz-ation in hardware.	This is where one decides what rules or filters apply to packets of a certain type.

(P.T.O)

⑥	The NIC provides a physical function (a standard ethernet port). Several virtual functions are also provided.	Perhaps based on network updates from other, possibly virtual, such as in network.
⑦	Each VM is mapped to one of these functions. So, the VMs themselves get	Programmed using open flow.

20. Stages of live VM migration:

Stage 0: Pre-Migration –

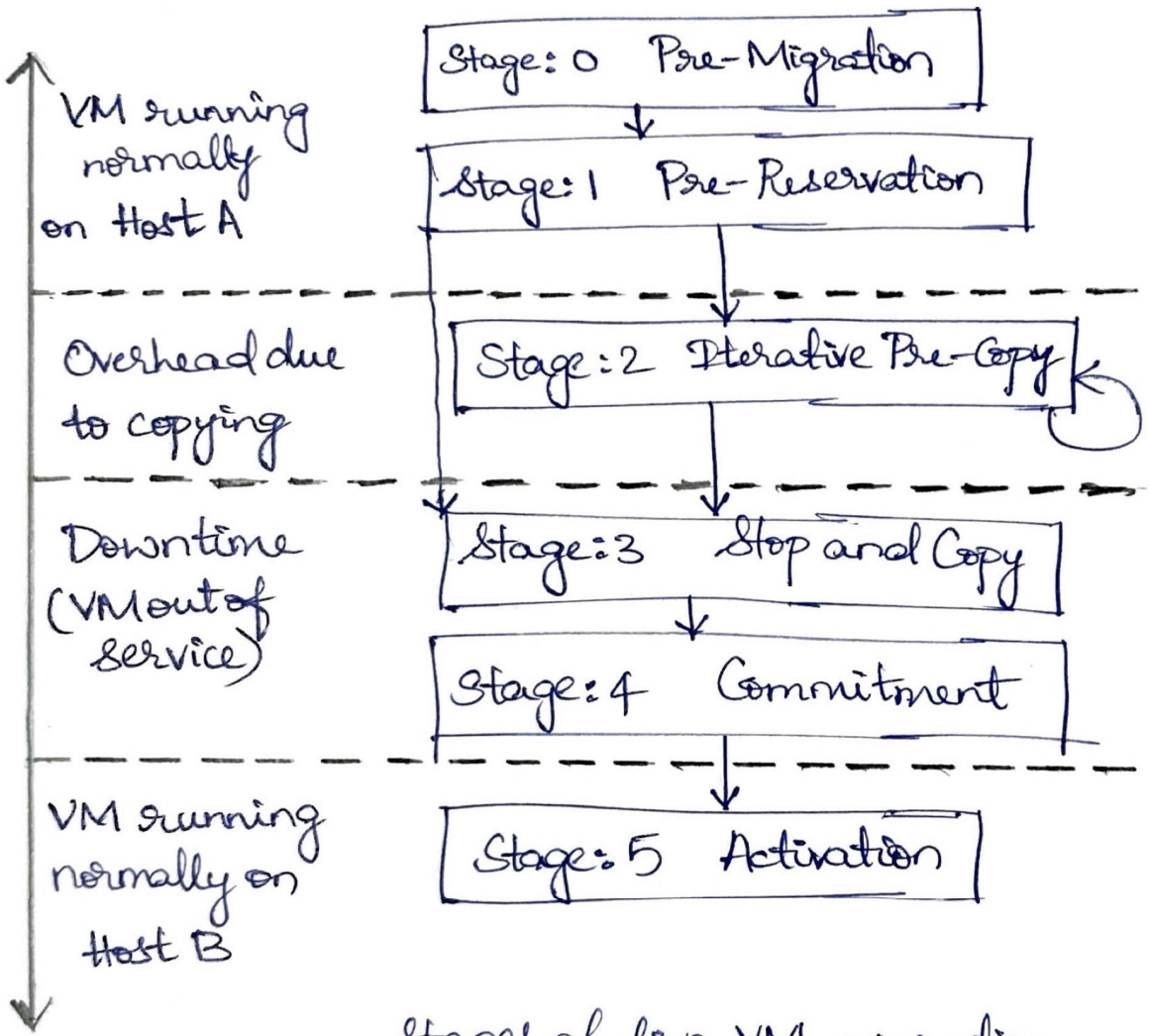
A VM will be running normally on a host A. Stage 0 will ensure which are active VMs on the host, what are the alternative physical hosts that maybe pre-selected for migration, and what are the block devices which are mirrored and free resources which are maintained.

Stage 1: Reservation –

It confirms that the required resources are available at host B and reserves a VM container in the target host.

Stage 2: Iterative Pre-Copy –

It enables shadow paging and then keeps on copying the dirty pages in successive rounds.



Stages of live VM migration

Stage 3: Stop and Copy –

In this stage, VM on host A will be suspended, and it will generate network traffic to redirect to host B. It will synchronize the remaining VM state to host B. This is called stop & copy on host B.

Stage 4: Commitment –

In this stage, host B intimates to host A that it has successfully acquired a consistent OS image. Host A acknowledges this message as the commitment of the migration transaction. Host A may now dismiss the original VM, and host B becomes the primary host.

Stage 5: Activation :-

The migrated VM will now start running on host B. It will connect to the local devices and resume in local operations.

21 Given,

Start-up company that wishes to run a service with,

128 servers (1024 cores) and

524 TB storage.

Costs are given as follows:

Cloud service provider cost = $0.12/GB per month

= $0.10/CPU per hour

Owning Cost = $349K for storage

= $7.5K per month for CPU

Now, we need to compare the per month costs of these two options,

Ownership (Considering operations for M months)

Storage $\Rightarrow \frac{349}{M}$

Total $\Rightarrow \frac{1555}{M} + 7.5$

(Considering 0.5 : 0.4 : 0.15 split for hardware : power : network and 3 years lifetime of hardware)

Outsourcing:

Storage = $0.12 × 524 × 1000 ≈ $62K

Total = Storage + CPUs = $62K + $0.10 × 1024 × 24 × 30

= $62K + $74K

= $136K

Break-even Analysis

Now, we shall analyse the required number of months for ownership operations to breakeven with outsourcing operations.

$\frac{\$349K}{M} < \$62K \Rightarrow M > 5.55$ months (for storage)

$\$\frac{1555K}{M} + 7.5 < \$136K$

$\Rightarrow M > 12$ months (for total)

∴ Outsourcing will be preferred by me. As it takes more than a year for the breakeven point.

22. Given,

Service Period = 30 days

Maximum service hours per day = 15 hours

⇒ Total service uptime = 30 days × 15 hrs/day

= 450 hrs.

Cost = Rs. 2500 per day.

⇒ Total cost = Rs. 2500 per day × 30 days = Rs 75000

Total outage time = (5hrs + 30min + 1hr 30min + 5min + 2hrs 30min)

= 9 hrs 35 min

$$\text{Service Availability} = 1 - \frac{\text{outage time}}{\text{uptime(actual)}}$$

$$= 1 - \frac{9\text{hr } 35\text{min}}{450\text{ hr}} = 97.92\%$$

Service Credit Available = 25% of total cost = $\frac{25}{100} \times 75000$

= Rs. 18750

Effective Cost payable = Rs 75000 − 18750 = Rs. 56,250

So the effective cost payable for buying the cloud service is Rs. 56,250

23. a) **Cost / Effective-Hour**

In-house server :

Purchase Cost = Rs. 70000

Purchase Cost per hour = Rs. $\frac{70000}{3 \times 365 \times 24}$ [Considering a span of 3 years]

= Rs. 2.66

Given efficiency = 40% = 0.4

So, the cost per hour would be (effective) = $\frac{2.66}{0.4}$ = Rs 6.65

∴ Total Cost per effective hour = 6.65 + management cost + power & cooling cost

= Rs. 6.65 + 20 + 30 = Rs. 56.65

Cloud Server

Given cost per hour = Rs. 7

Efficiency = 80% = 0.8

Cost per effective hour = Rs. $\frac{7}{0.8}$ = Rs. 8.75

∴ Total Cost per effective hour = Rs. 8.75 + management cost

= Rs 8.75 + 1 = Rs. 9.75

∴ Total cost/effective-hour for cloud server is Rs. 9.75 and in-house server is Rs. 56.65

b) Cost Analysis for 3 years

In-house server:

Purchase cost = Rs 70,000

Power & Cooling cost for 3 years = 30 × 3 × 365 × 24
= Rs. 7,88,400

Management Cost = 20 × 3 × 365 × 24 = Rs. 5,25,600

Total Cost = Rs 70,000 + Rs 7,88,400 + Rs 5,25,600
= Rs 13,84,000

Cloud Server:

Cost per hour = Rs 7

cost for 3 years = 7 × 3 × 365 × 24 = Rs 183960

Management Cost (for 3 years) = 1 × 3 × 365 × 24 = Rs 26,280

Total Cost = Rs 183960 + Rs 26280 = Rs 2,10,240

Given, Revenue per day = Rs.3000

For 3 years = Rs 3000 × 3 × 365 = Rs 32,85,000

Expected Profits = Revenue generated - Cost incurred

For In-house, expected profit = Rs 32,85,000 - Rs 13,84,000

= Rs 19,01,000

For Cloud server, expected profit = Rs.32,85,000

= Rs 2,10,240

= Rs. 30,74,760

∴ Expected profit for in-house server is Rs 19,01,000 and for cloud server is Rs 30,74,760

c) Let 'x' be modified efficiency.

We need to equate cost/effective-hour for in-house & cloud server.

Cost/effective-hour for cloud = Rs 8.75 (from (a))

for in-house = Rs. $\frac{70000}{3 \times 365 \times 24 \times x}$

$$\Rightarrow \frac{70000}{3 \times 365 \times 24 \times x} = 8.75$$

$$\Rightarrow x = \frac{70000}{3 \times 365 \times 24 \times 8.75}$$

$\Rightarrow \quad x = 0.304$

So, the modified efficiency of in-house server is 30.4%

24. a) Power consumption in a physical machine is given by,

$$\Delta P_j = \begin{cases} P_{j,idle} + (P_{j,peak} - P_{j,idle}) UT_j & UT_j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

where,

$P_{j,idle} \longrightarrow$ Half of peak power consumption, $P_{j,peak}$

$UT_j \longrightarrow$ Percentage of utilization in PM_j

Servers' total energy consumption ΔE_j is determined via the energy model represented as:

$$\Delta E_j = \int_t \Delta P(UT_j)\, dt$$

b) Proposed algorithm first distinguishes the PMs having hotspots, then selects some VMs from each hotspot PM to perform migration and determines destination PMs for the VMs.

Algorithm to predict hotspots:

Input: PM_{active} Output: Is PM having hotspot or not?

window length ← 20; $UT_{pm}(t) = \frac{CU_{pm}(t)}{Cap_{pm}(t)}$,

Set Th_{not} applying MAD = 1 − S * MAD

for each pm ∈ PM_{active} do

begin

if length. $UTHistory_{pm}(t)$ < window length, then

if $UT_{pm}(t) > Th_{static}$. then

return true

else return false

else predict $UT_{pm}(t+1)$ using CBS Bi-LSTM

if $UT_{pm}(t) > Th_{not}$ && $UT_{pm}(t+1) > Th_{not}$, then

return true

else return false

end.

Algorithm to predict cold spots

We follow the same algorithm as hotspot, but we compare UT_{pm} with Th_{cold} i.e.,

$$UT_{pm} > Th_{hot} \quad \text{to} \quad UT_{pm} \leq Th_{cold}.$$

(c) PM's energy consumption relies on the utilization of resources mainly memory, network bandwidth, CPU etc. However, existing researchers reveal that the CPU utilizes more power than other resources. Hence, we represent the PMs resource usage by its CPU usage. As compared to the energy consumption caused by different algorithms, the suggested approach uses a lesser amount of energy. The reason being that the PMs having their usage lower than the threshold limit are switched to power-saving mode. The energy consumption of PMs is further optimized via packing the VMs into the most loaded PMs.

d) A majority of energy is used in cooling system. On optimizing the cooling system, we can hence reduce energy usage. We can use Deep Reinforcement Learning to optimize the cooling system in a green data center. Traditional methods involve knowing mechanical cooling, electrical and thermal management but we can use large amount of monitoring data to optimize the control policy.

An end-to-end cooling algorithm can be created that is based on the actor-critic framework and the DDPG algorithm. The evaluation network can be trained to predict an energy cost counter penalized by the cooling status of the DC room, and a policy network can be trained to predict optimized control settings when given current load and weather information DDPG can be used. The objective function aims to strike a balance between minimizing the PUE and preventing overheating in the server.

The penalty function:

$$\alpha \ln\left(1+e^{T_{zi}-\phi}\right) + E_{pm}$$

where α is penalty pricing
T_{zi} is average outlet temperature of zone i
ϕ is overheating threshold

The cost function (reverse of reward) is minimized.
The training data can be collected from historical insights.

∴ Using Deep Reinforcement Learning will give us the optimal Q and μ network with the best weight parameters θ_{best} & μ_{best}.

∴ Such an algorithm, will give us the optimal control policy for cooling system & hence reduce the energy consumption by a lot.