Lecture 14 CS695

[memmgmt] [livemigration]

→ building on the VM primitive

→ ballooning

content-based sharing

downtime

migration time

save-and-restore copy

iterative pre-copy ⊛

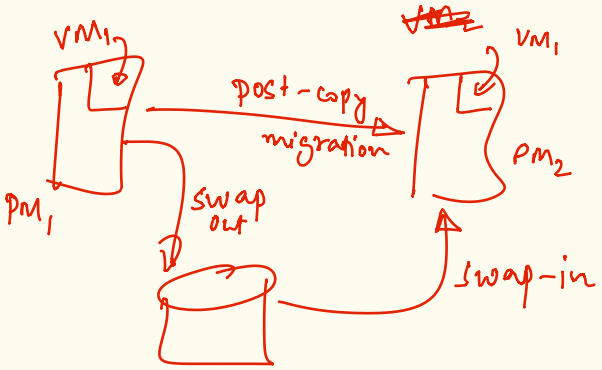
post copy migration

coverage:

- A3
- containers
- virtualization / service paradigms
  - FaaS / serverless
  - API virtualization
- research papers

paper — ~20 yrs old.

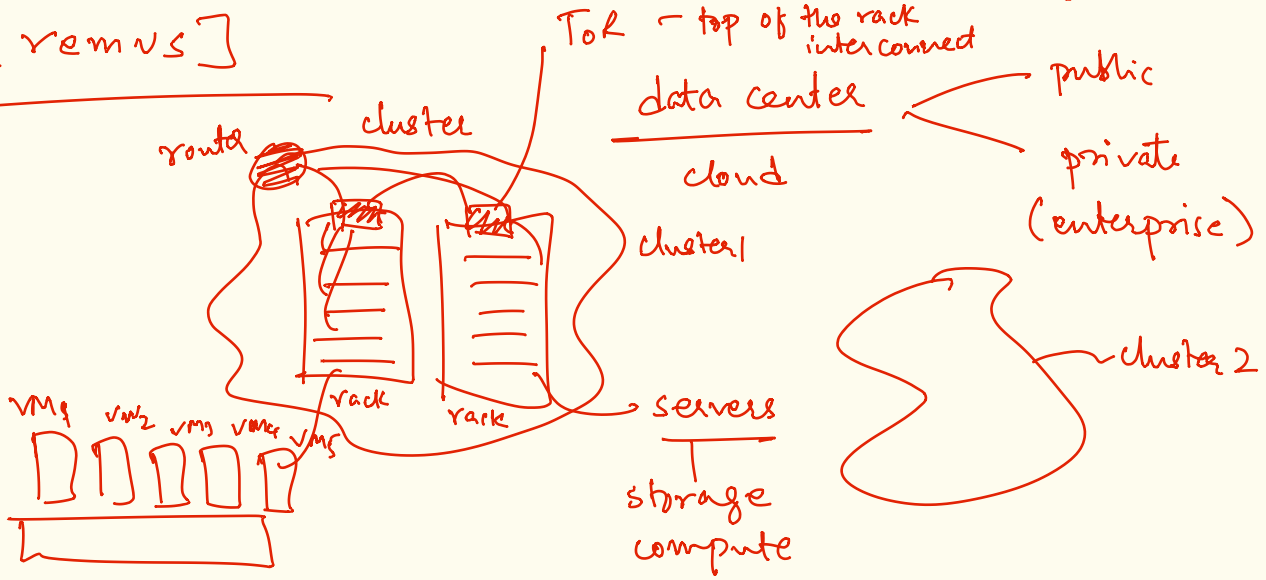
DT, MT = f (memory size, writeable working set size, dirty rate, network b/w → migration rate)



[sandpiper]

[remus]

VM-based provisioning & management.



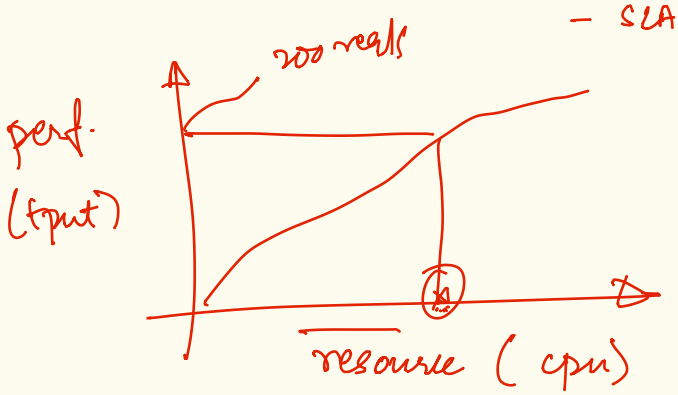
service provider: provide execution envs. for applications.

DC — VMs

cloud / service provider — cost:
- utilization
- SLA violations

functional:
- isolation

perf.:
- SLAs, SLOs
- throughput
- latency



(i) use perf. to resources correlation to determine resources needed.

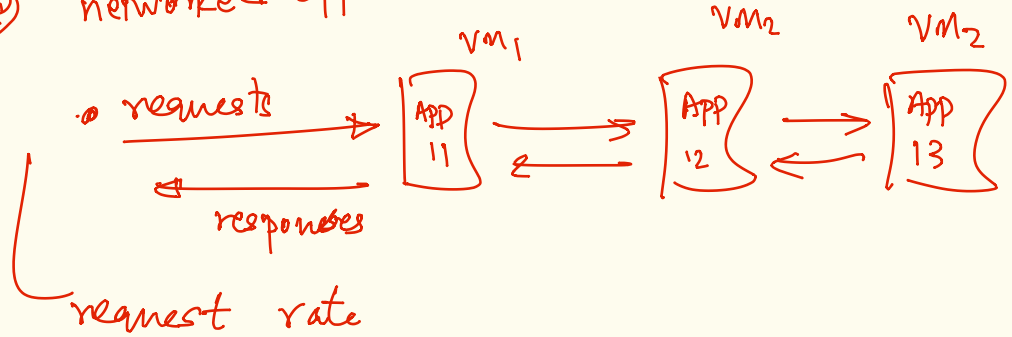
(ii) specify SLA & provider assures all resources to meet SLA.

e.g.s. - maintain 200 rq/sec
- 99th percentile response time < 50ms.

SLA ~ service level agreement/objective.

[sandpiper]

* networked applications



* dynamic workloads

* hotspot: enough resources allocated to a VM. → SLA violations

if aggregate resource demand > capacity of physical machine.

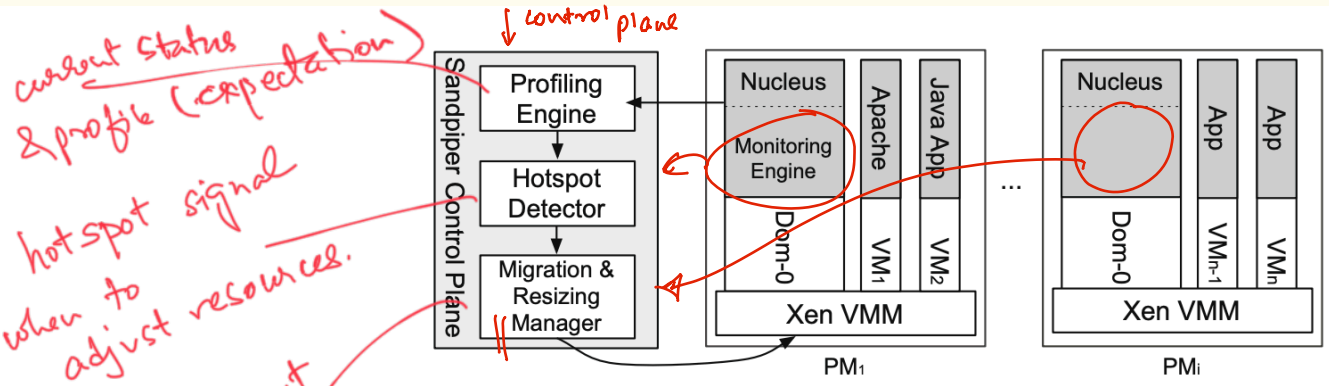


Fig. 1. The Sandpiper architecture.

(i) monitoring & profiling

(ii) hotspot detection

(iii) resizing & migration.

black box vs. gray box

black box

gray box.

cpu: xenmon ~ tapped into vm scheduling events & est. CPU util. per vm.

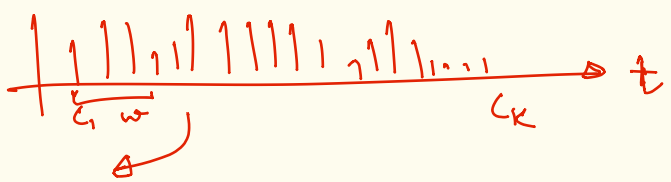
network / disk. / IO : through the virtual IO backend drivers.

memory : ~ sample + invalidate ⇒ working set estimate

~ swap rate /vm

(*) time series

cpu util



profile

pdf

current alloc

cpu util

effect of work-conserving scheduler.

if ($k+1^{th}$ util > threshold) & k > n greater than threshold ⇒ HOTSPOT

$\hat{\mu}_{k+1} = \mu + \phi(\mu_k - \mu)$

mean bias factor

$\lambda_{cap} = \frac{1}{S}$ ← service time

$\frac{\lambda_{peak}}{\lambda_{cap}} = \#\,cpus$

Remus → (for next class).

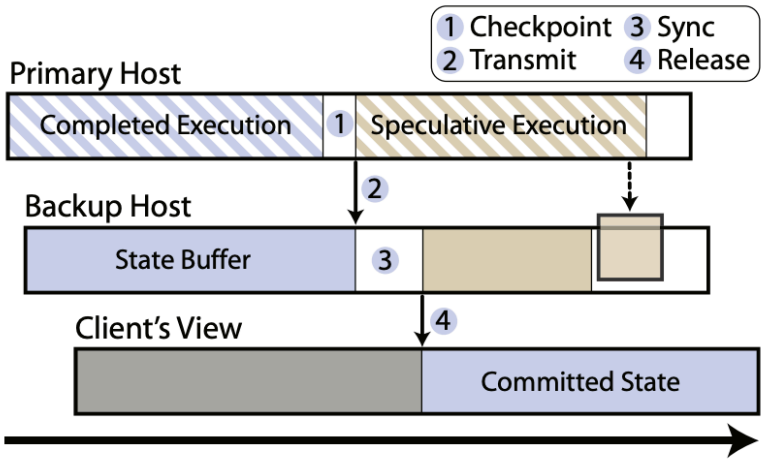


Figure 1: Speculative execution and asynchronous replication in Remus.

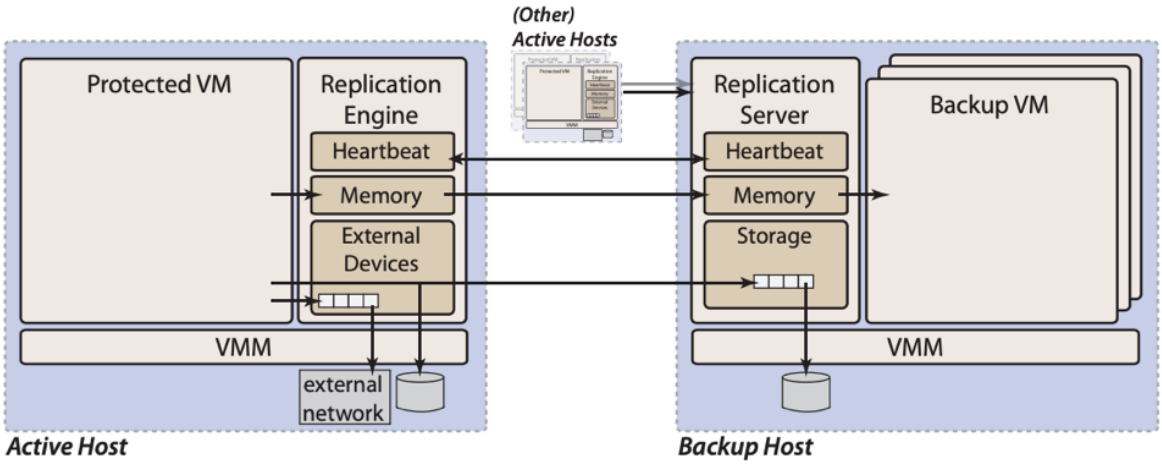


Figure 2: Remus: High-Level Architecture