Lecture 13 — CS695

- A2 due today
- Midsem exam. 1st March
- 2.30 pm
- LA ???
- all content till today (and some more)

(*) [memmgmt] — Memory Resource Management in VMware ESX Server
Carl Waldspurger (VMware) — OSDI 2002

[livemigration]
- snapshots
- record & replay

operating system design & implementation

Context:
- overcommit & server consolidation ~ pack multiple VMs on a single server / PM
  - committed resources greater than physical resources.

$\forall$ p m $\quad \sum_{\forall VMs} P_{ranges} > m_{range}$

~ vmware esx server → fully virtualized hypervisor

- "management" actions:
  - allocate memory / generate mapping
  - (*) swap in / swap out (revoke / reclaim)
  - (*) vmap / deallocate.
  - handling faults — revocation
  - free page state
  - permissions handling
  - sharing / cow

- 2 page tables
  - v2p
  - v2m

(*) pmap — p2m mappings per VM.

overcommit ⇒ increase in (simultaneous) demand ⇒ memory pressure

(*) statistical multiplexing

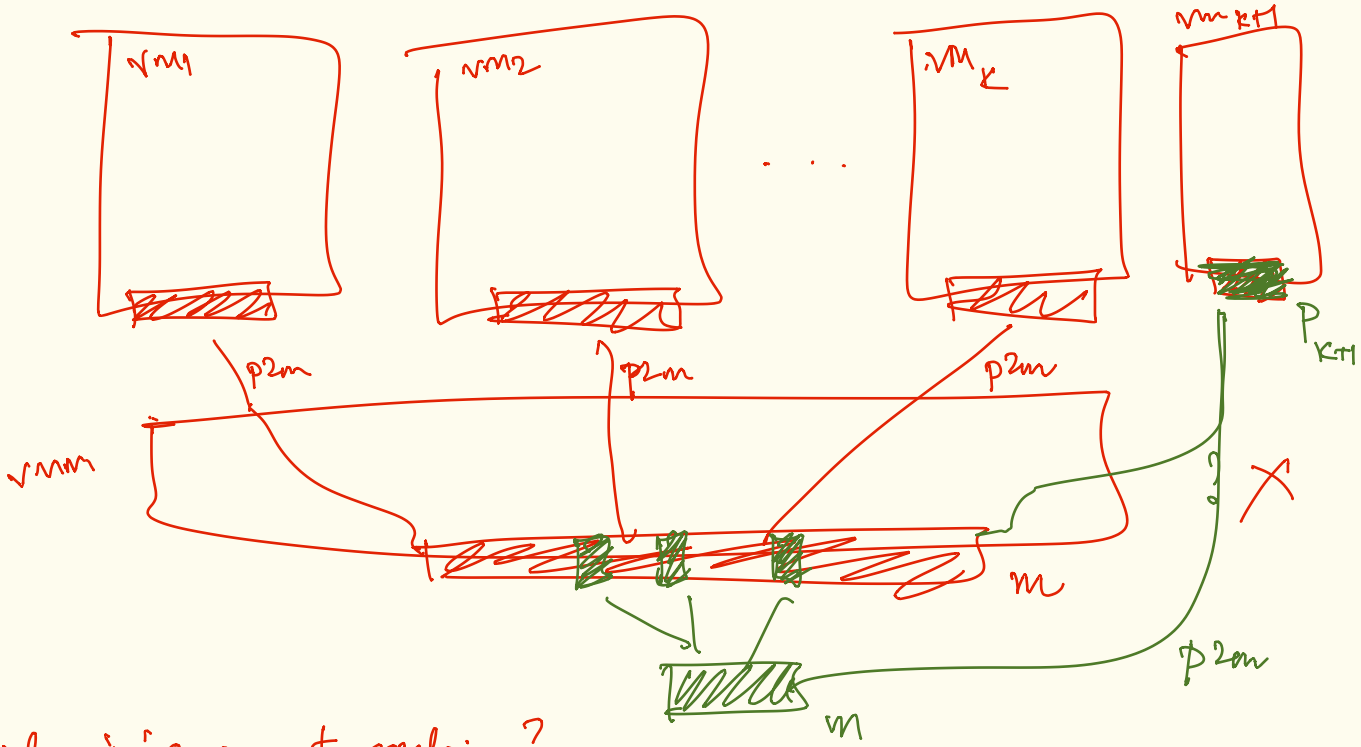
execution entities will have ~~static~~ statistically different req.

temporal / spatial manner

soln space

- avoid memory pressure ~ increasing memory utility / efficiency
  - CBPS ~ content based page sharing. (*)
- handle memory pressure
  - swapping / paging (demand)
  - ballooning. (*)

(i) ballooning



- which 'm' pages to reclaim?
- how many 'm' pages per vm?

(*) collect meta-information about accesses to estimate per-page utility.

high overhead

incomplete! / wrong decisions.

(*) double-paging problem ~ (i) VMM swaps out a m-page invalidates p2m mapping

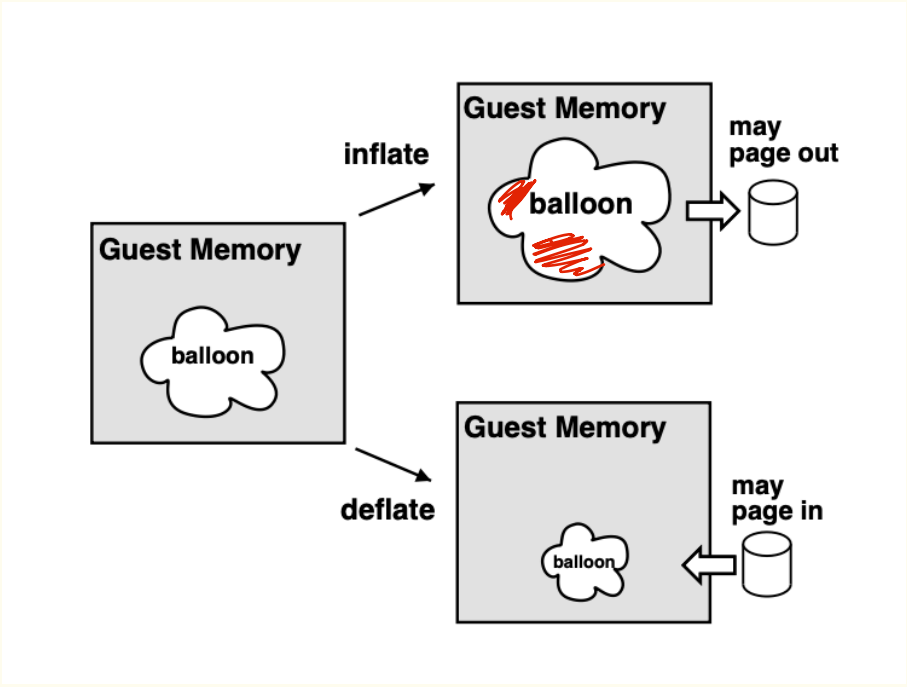
(ii) VM ~ decides to swap out the same p page
⇒ rebuild p2m
⇒ swap back p to virtual disk.

(*) Soln. memory ballooning.

idea: guest knows state of its memory utility.

approach ~ co-operative guest OS + balloon inflate/deflate.
induce/reduce memory pressure in the guest.

~ balloon driver (LKM) in the guest
~ interacts w/ a virtual/pseudo device
~ device IO is a pvt. channel for the VMM ↔ balloon driver.
- communicate actions
- transfer 'p' into

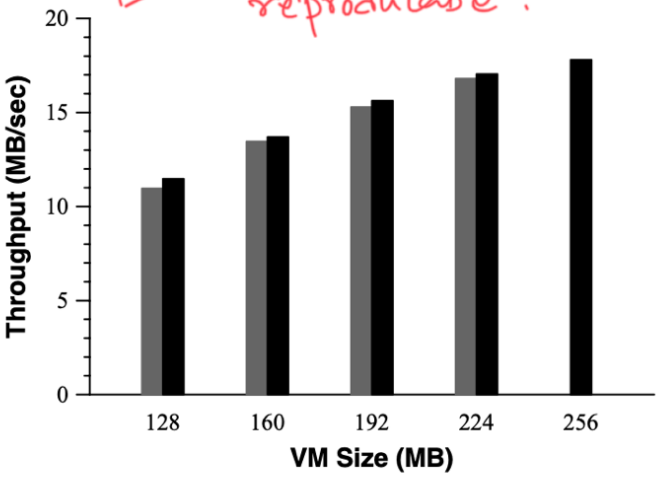


balloon driver

inflate
- alloc memory for balloon (Kernel) guest OS
- "pin" memory of balloon driver
- convey p-range of balloon to VMM
- invalidate p2m mappings at VMM
~ move free pages in the pmap

Q ~ how much to balloon per VM?
↳ ⇒ effective memory size of a VM.

Q. is this experiment reproducable ?



(*) what was workload pattern of each of the 40 clients.

Figure 2: **Balloon Performance.** Throughput of single Linux VM running `dbench` with 40 clients. The black bars plot the performance when the VM is configured with main memory sizes ranging from 128 MB to 256 MB. The gray bars plot the performance of the same VM configured with 256 MB, ballooned down to the specified size.

---

or removed from a VM in order to rapidly adjust its

(2) Sharing memory (avoid pressure)

~ VMs — same/similar guest OS, applications, libraries, data.

~ explicit tagging of shareable content.

~ bcopy — block IO ~ hook in VM. ~ falls into an unified disk cache at the VMM. block read/write

- vdisk1, block # 23 ⟺ vdisk2., block #46
(VM1) (VM2)
VMM b23, vd1 memory
b46, vd2

CBPS: content based page sharing

~ page identity ⟺ content.

~ scan all pages
share identical pages
} out-of-band

- nxn ~ $O(n^2)$ — ✗

- hash based technique

per page scan
(i) generate hash
(ii) lookup hash for entry
(iii) if entry exists
- rehash the entry
- byte-by-byte comparison to confirm match
- if match
  - share cow

(iv) if entry does not exist
add (hint) entry to the hash

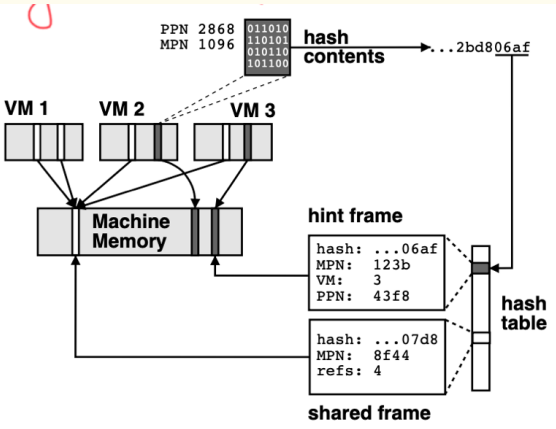


Figure 3: **Content-Based Page Sharing.** ESX Server scans for sharing opportunities, hashing the contents of candidate PPN 0x2868 in VM 2. The hash is used to index into a table containing other scanned pages, where a match is found with a hint frame associated with PPN 0x43f8 in VM 3. If a full comparison confirms the pages are identical, the PPN-to-MPN mapping for PPN 0x2868 in VM 2 is changed from MPN 0x1096 to MPN 0x123b, both PPNs are marked COW, and the redundant MPN is reclaimed.

Q ~ scan rate? MADVISE

~ where to scan?

KSM — Linux Kernel Same page Merging

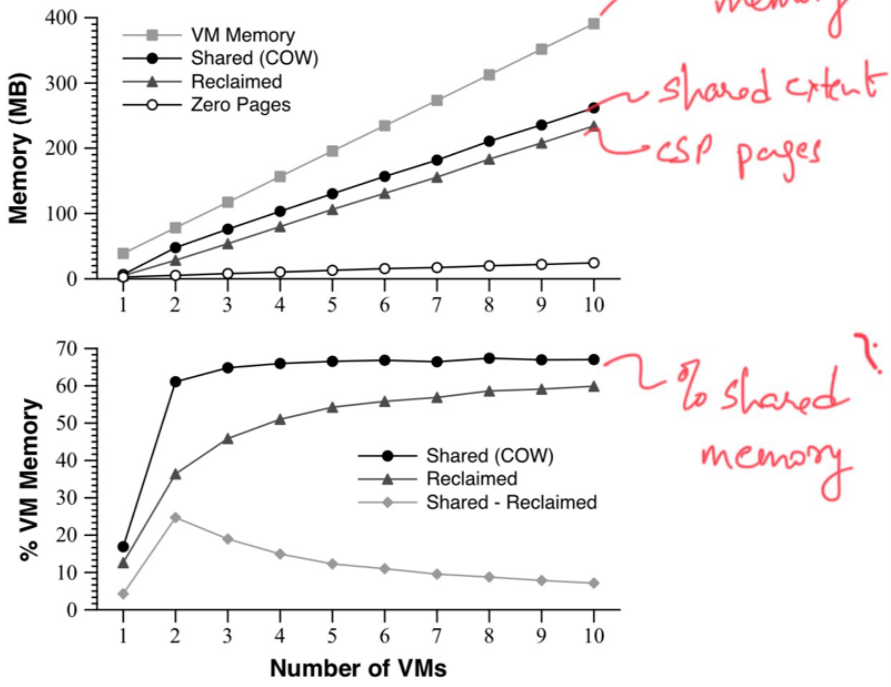


Figure 4: **Page Sharing Performance.** Sharing metrics for a series of experiments consisting of identical Linux VMs running SPEC95 benchmarks. The top graph indicates the absolute amounts of memory shared and saved increase smoothly with the number of concurrent VMs. The bottom graph plots these metrics as a percentage of aggregate VM memory. For large numbers of VMs, sharing approaches 67% and nearly 60% of all VM memory is reclaimed.

		Total	Shared		Reclaimed	
	Guest Types	MB	MB	%	MB	%
A	10 WinNT	2048	880	42.9	673	32.9
B	9 Linux	1846	539	29.2	345	18.7
C	5 Linux	1658	165	10.0	120	7.2

Figure 5: **Real-World Page Sharing.** Sharing metrics from production deployments of ESX Server. (a) Ten Windows NT VMs serving users at a Fortune 50 company, running a variety of database (Oracle, SQL Server), web (IIS, Websphere), development (Java, VB), and other applications. (b) Nine Linux VMs serving a large user community for a nonprofit organization, executing a mix of web (Apache), mail (Majordomo, Postfix, POP/IMAP, MailArmor), and other servers. (c) Five Linux VMs providing web proxy (Squid), mail (Postfix, RAV), and remote access (ssh) services to VMware employees.

memory management strategy proposed in this paper —

mechanism. | decisions

Step 1: sharing — avoid/delay memory pressure — content based sharing { scan rate, where to scan

Step 2: ballooning — handle memory pressure → ballooning — size of VM ⇒ size of balloon

Step 3: demand paging — too much pressure, VMM falls back to forced swapping. → swapping → when to move to swapping?

Q. what should the memory size of a VM? (by implication size of balloon).

- just enough to meet current memory usage
- demand on memory is dynamic/time-varying
- usage different from allocation (mapped memory).

(*) one common strategy is share-based or proportionate sharing/allocation of memory.

eg: of VM_1, VM_2 & VM_3
a 1:2:3 ratio yields an allocation of 1GB, 2GB & 3GB resp. from a 6GB total memory.

paper refers to this as shares-per-VM.

and memory can allocated to VMs that "pay" a higher price (shares/page) compared to a VM that pays a lower price.

BUT allocation is not same as usage!

so, we really want effective shares-per-page.

i.e: how allocated pages is the VM using.

$$\varphi = \frac{S}{P \cdot (f + K \cdot (1-f))}$$

- φ: effective / adjusted shares per page ratio
- S: shares of a VM.
- P: #allocated pages
- f: fraction of active pages
- K: idle page cost

referred as working set size in paper

$$K = \frac{1}{1-\tau}$$

$\tau = 0$

no cost $\Rightarrow \varphi = \frac{S}{P}$

$\tau = 1$

no shares. $\varphi = 0$

$\Rightarrow$ random allocation!

$\Rightarrow$ in practice $0 \leq \tau < 1$

(*) how to estimate 'f'?

(1) invalidate all p2m mappings.

on each access. → page fault
- add to active count
- reset mapping.
- redo access.

redo this loop periodically to get time varying ~~upd~~ estimate of 'f'.

(*) invalidate $\Rightarrow$ flip pte flag present bit to 0.
do not remove mapping.

costly → each page access is a fault.

(2) redo 1 with a sample of 'n' pages.

- fraction of n approximates all pages

n pages chosen at random.

eg: 100 pages every 30 second.

(*) estimate 'f' in every time window

- n=100 T=30 sec.

$f = \frac{a}{n}$ — # accessed pages out of n in time t.

- moving average across time windows to keep updating 'f'.

from 'f' estimate ξ ← effective shares per page.

for each VM.

- use ξ_1, ξ_2, ξ_3 etc. for proportionate sizing of memory of each VM.
- adjust memory sizes of each VM via: ballooning.

Questions answered via experiments —

- Does the sampling based heuristic track memory usage well/accurately?
- How does idle tax & active fraction change allocation levels of VM?