Memory Management in Virtualized Settings

Questions on projects?

Comment on reviews;

- Remember to include what confused you; not "problems" you find in the work (that is too easy!)
-

Notes from reviews:

- What about NUMA? Targeting much smaller machines
- How hard is memory hotplug?
- How is VMware pmap different than Disc PMAP
  - They are similar: disco PMAP is also translates GPA to HPA via pre-filled in TLB entry; also has GVA backpointer
  - Sharing in VMware handled via separate hash table
- How are shares allocated/reallocated?
  - By an admin
- What about communication?
  - Not needed for this scenario – separate VMs.
- How are multiple moving averages used?
- Still need to remap I/O memory?
- Randomized page replacement?
- Hash collisions – chaining?
  - # of buckets is smaller than # of hash values; chaining is used within a bucket but hash values are assumed unique
- Random sampling: what workloads are bad?
  - Phased workloads with ~30-second phases
  - Sparse random access, where lots of data is needed at low latency but popularity varies widely (assume big memory)
- Private channel to VM server: a hypercall – special trap the hypervisor handles, OR a shared memory page
- How are VMs charged for shared pages? Fractionally : 1/n for n sharers
- What if balloon driver stops working?
  -

VMWARE PROBLEM:

- Want to statistically multiplex OS, so can have efficient memory usage (avoid cost of buying lots of memory)
  - Example: have 10 guest VMs, tell each one they have 512 MB of ram, but only really have 2 gb of ram.
  - QUESTION: Why overcommit?
    - ANSWER: more efficient use of memory; most guest OS don't use all the physical memory. Can shift memory between VM as needed
    - ANSWER: what if you don't have enough memory? Life still sucks - can't fix it.

- QUESTION: why is this different from virtual address in an OS?
  - ANSWER:
    - OS makes no guarantee about what pages available to app,
    - app doesn't do paging; OS doing it on its own
- Can't control how guest OS manages memory directly; must tell it that it has a fixed amount of memory (e.g. no hotplug / hotremove)

GOAL:

- Vary machine memory allocated to a VMM
- Separate policy (how much memory, what memory) from mechanism (how do you get pages).
- Let guest OS provide policy of what to remove

Mechanisms: for removing / adding pages

Policies: for determining how much memory a VMM should have, when to reclaim, how much to reclaim (or grant), for determining which pages to remove

QUESTION: What is the big tradeoff here?

ANSWER: cpu cycles to do extra computation vs. memory & cost of swapping

---

Mechanisms:

---

1. Worst case: paging. Can select pages from a VMM and swap them to disk

QUESTION: why so bad? used by OS

ANSWER: incomplete knowledge of what pages should be taken. E.g. might interfere with working set policy of OS—page not used, but because process not scheduled but will be soon, or page is pinned in OS for good reasons.

ANSWER: double-paging. VMM may swap a LRU page, then OS swaps same page—must bring page back in first, so double cost, single benefit

QUESTION: VMWARE uses random paging. WHY? Is it a good or bad thing?

2. Better case: ask OS to give back memory. Ballooning

Separate allocation (# of pages, handled by VMM) from replacement (which pages, handled by OS)

Key idea: allocate physical pages in OS, give them to VMM

IMPLEMENTATION: write a kernel driver (or could be usermode) that allocates pages and pins them in memory. Then tell VMM about the pages via hypercall

WHY WORKS: OS cannot modify/take pinned pages

QUESTION: does VMM have to swap them?

A: no, balloon driver owns pages, doesn't care about contents

NOTE: ballooning is a bit like Irix telling Disco about free pages.

QUESTION: why does this work?

A: OS will either give memory off free list to driver, or will remove other things from memory to give to driver. E.g. driver has higher priority on memory than existing uses

KEY OBSERVATION: pages are explicitly reclaimed from a VM/OS. Not chosen at random. Unlike global page replacement policies, e.g. clock. VMM has to ask an OS for a page, or swap a page from an OS.

QUESTION: Why not add a new interface to OS for this?

Answer: is being added. Question – what should it do?

Example: transcendental memory. Data that can be reclaimed at any time. Good for cached data, cold pages

3. Best case: share pages

Idea: if two pages have same contents, only need to store one copy.

Example: all the pages that are zero filled

Example: two VMs run same kernel, same binaries --> probably text

segments have same bits

BIG PROBLEM: how do you find duplicate pages? compare all pages?

ANSWER: build an index

IMPLEMENTATION:

1. pick candidate pages
2. hash contents, search hash table
3. On hit: compare full pages, set up COW

   QUESTION: why bother comparing if hashes match?

   ANSWER: hash is small enough that collisions could exist

   If they exist: pick only 1 page to deduplicate

   ANSWER: hash becomes a hint that the page may match, not a guarantee that

   it does

   QUESTION: why not make hash bigger?

   ANSWER: tradeoff memory overhead for comparison cost on false match
4. On miss: add page as a "hint" - something that could be shared

Questions:

1. what pages do you pick? A: random
2. How many, how often? A: some rate mechanism. Trade overhead for earlier detection of sharing. E.g. 100 pages / 30 seconds.

3. What about shared count vs backmaps? Share counts use less memory (16 bits for all sharers) vs backmaps that take a pointer (32 bits) per sharer; for zero pages there could be thousands. Note: need to make sure never need to map from frame to GPA, as information not present

REMEMBER: need more memory to improve performance, so not worth spending too much time finding memory if it overrides the performance gains

NOTE: HOW IMPORTANT IS THIS? 7-18%? Why not just buy more RAM?

QUESTION: Why wouldn't you want to share duplicate pages?

- Security issues? Can detect if shared by testing for CoW – suppose a page has a password in it, can try different values
- Performance isolation? One machine can affect performance of another

QUESTION: How do you account/charge for shared pages – which VM?

NOTE: page sharing now disabled by default:

- High overhead on big memory machines
- Security issues (as noted above)

POLICY

Big question: how do you allocate pages between virtual machines?

- global vs. local policy?

PROBLEM: Working set for an OS not quite like for an application; applications don't adapt to changing memory sizes but OS does

PROBLEM: often have a desired performance goal for an OS, or priorities for OS, want some minimum performance.

- > drives towards local policy

VMWARE POLICY: proportional share (we'll see this later)

Key idea:

- some pool of resources R

- Want to allocate fractions of it to different users
- would like a minimum guarantee, but efficient use of excess capacity

Solution:

- give each user a set of shares, like stock shares in a company
- value of a share is #shares / total # shares—this is minimum guarantee
- At any time, amount of resource is # shares / total # shares **demanded**
- **Shares represent *relative resource rights that depend on the total number of shares contending for the resource***

Idea: under heavy use, get strict proportion. Under light use, can get more in proportion to others who want more and their shares/

Way to think about it: everybody who wants a resource buys lottery tickets with shares. Winner picked at random from all shares bid. If not need, don't buy tickets

So: under full demand by everyone, all pay same price per page: shares / pages granted. When not everybody has full demand, some with fewer shares will get more pages

RECLAMATION: when pages needed, search for VM that is paying the **least** for its memory (e.g. got some memory when others didn't want it.)

Algorithm: dynamic min-funding revocation.

Example

VM 1: 100 shares
VM 2: 100 shares

Total memory: 400 mb

VM 1 starts running, acquire 256 mb for 100 shares

price = 100/256 = 0.4

VM2 starts running, gets remainder: 144 MB for 100 shares

price = 100/144 = 0.69

When VM2 wants more memory, it comes from VM1

VM2 needs more pages, asks for 56

VM2 price = 100/200 =0.5

VM1 price = 100/200 = 0.5

Now VM1 has 200 MB, VM1 has 200 MB, both pay same price - in equilibrium

NOTE: reclamation is kind of expensive; need to activate balloon or swap pages.

QUESTION: is this the right policy? It doesn't guarantee timeliness, just a minimum.

NOTE: Real problem is not minimum guarantee, but how to efficiently use memory above that.

PREVENTING UNDERUSE OF MEMORY:

Problem: OS may have memory it is not using, e.g. free list or pages not being referenced (e.g. non pageable kernel pages that aren't referenced). Could be better used in another VM.

QUESTION: does this problem arise in a normal OS?

ANSWER: yes, but handled by normal working set or clock algorithm - unreferenced pages get replaced

QUESTION: why different in a VMM?

ANSWER: don't want to take a specific page (leave that to OS), but want to measure memory usage and reclaim any page (with balloon)

SOLUTION:

Tax on idle memory

Assume you know what fraction of client memory is idle – not being used well.

Concept: charge more for unused memory than used memory—represents a lost opportunity

Tax rate is fraction of idle pages that will be reclaimed.

Tax = t
Normal cost = 1
Taxed cost = 1/(1-t)

t = 0 --> taxed cost = 1, normal cost (counts as one page)

t = 0.5 -> taxed cost = 2, one idle page counts as two used pages

t= 0.75 -> taxed cost = 4

t = 1 --> taxed cost = infinite (counts as infinite pages)

Shares per page = shares / pages * (frac-used + taxed-cost*frac-idle)

Example: tax rate - 0.5, 50% of pages idle, 100 shares, 100 pages

t = 0 rate = 100/100*(0.5 + 0.5) == 1
t = 0.5 rate = 100/100*(0.5 + 1) = 0.66 – you are paying less, so your pages will be taken away
t = 1 rate = 100/100*(0.5+inf) = 0 – if you have any idle pages, your memory will be taken away first until you don't have any idle pages

Result: when pages needed, rate will look lower for those needing pages

VMware choice: 0.75 ->

QUESTION: Does this apply to buffer cache in guest OS?

ANSWER: yes - just normal memory. If not access

QUESTION: what would happen to Windows, that uses extra memory as a cache?

ANSWER: it would work - contents of memory aren't relevant; if that memory is not accessed, it will be reclaimed through balloon.

Detecting idle fraction/memory demand:

Really: How do you calculate the working set for a program?

QUESTION: how different than determining idle pages in an OS?

ANSWER: want to avoid bad interaction with OS page management

SOLUTION: randomly scan 100 pages every 30 seconds per vm, use as a **statistical estimate**. Can do by making page invalid, catching trap in VMM

QUESTION: Why does this work? altain@earthlink.net

- Treat pages not referenced in 30 seconds as old enough to swap out; most in use pages should be referenced within 30 seconds so gives a sample of how much memory is needed
- Works better for small VMS (< 1GB) than huge ones as sampling a higher fraction of memory
- **QUESTION: What workloads are bad for sampling?**

QUESTION: from scan rate, how do you compute idleness?

QUESTION: how do you handle fluctuation?

ANSWER: Exponentially Weighted Moving Average: new value = x*last sample + (1-x) old value

How choose x?

- high x weights towards recent values, responds quickly
- low x weights history more, takes a while to respond

example: x = 0.2e

QUESTION: what do you want in this situation?

ANSWER: quickly respond to needs for more memory, slowly handle decrease.

HOW DONE: pick max of slow + fast average (plus one for current period)

QUESTION: What happens when there is a sudden increase in use?

ANSWER: use max of slow, fast, and fast+current, which incorporates page use from current 30 second window. Lots of page faults raise fast+current

REASON: Current rapidly increases under memory pressure, so allocate memory quickly. Slow decays slowly when memory use drops, so keeps memory around for a while

---

Other policies

- Admission control: ensure that only run if you have enough memory (memory for all min, memory+swap for all max)
- Usage levels to trigger behavior:
  - ballooning
  - swapping pages
  - suspend a VM

Allocation Policies:

Set target percent memory free to drive what policy to use and how aggressively (6-1% free)

High: ample free memory, no reclamation

Soft: reclaims memory with ballooning, paging only if ballooning not possible

Hard: paging (cannot wait for ballooning, too much need)

low: paging + suspend VMs below their allocation (so not fault in more pages – like workingset)

Evaluation

How is the system evaluated?

- Total system performance?
- Each feature separately?

What is measured?

- Contribution of a feature
  - e.g. how much memory is shared
  - e.g. what is overhead of a single feature compared to not having feature
- OS/Application behavior
  - e.g. amount of sharing by OS/workload
  - Motivates adding a feature

Demonstration vs evaluation/proof

- demonstration shows a feature works (e.g. you can reclaim shared memory)
  - May show how well it works for small # of workloads
- Evaluation tries to predict how well it will work in other cases
  - generally use standard benchmarks that are representative of broad sets of workloads
- Time series for responsive systems often used

    - show behavior when events happen: machine failures, reconfiguration, program phases
-

Parameter sensitivity

- Many systems have parameters that govern their behavior
    - E.g. tax of 0.75
- Generally good to evaluate range of values to show why value was picked
- Many systems depend on system characteristics
    - Amount of overcommit
    - Total memory size for scanning for sharing
    - CPU power for hashing
    - I/O bandwidth for paging
- Good to show behavior across range of sizes
-