Memory Resource Management in VMware ESX Server

Three kinds of memory addresses involved in virtual machines
  * Guest virtual addresses
  * Guest physical addresses ["physical" addresses in paper]
      Would be plain physical addresses if not in VM
      Only have meaning in the context of a specific VM
      Can be paged out to disk and not exist in RAM
      Can be mapped to any host physical address by VMM
      Can even map multiple guest physical pages to same host physical page
  * Host physical addresses [machine addresses in paper]
      Global across all VMs
      Never seen by guest OSes
      Correspond to actual bits in DRAM chips

These are defined by several data structures
  A guest operating systems running in a VM maintains a *primary* page table
    Maps from *guest virtual* addresses to *guest physical* addresses
  The VMM has per-VM data structure called pmap in this paper
    Maps from *guest physical* to *host physical*
    (Not to be confused with pmap machine-dependent layer in the BSD kernel.)
  In addition, the VMM maintains a *shadow* page table
    Maps directly from *guest virtual* addresses to *host physical*
    Hence is a function of guest PT and pmap
    Shadow PT is the only PT seen by hardware (assuming no nested paging)
    Accessed/Dirty bits authoritative (VMM must copy back to primary PT)
    Can be computed lazily on the fly, so use shadow PT as a cache
    Wednesday lecture will deal with keeping shadow PT in sync w. primary

Unfortunately, today's paper uses an older terminology:
  Ed Bugnion, who coined older terms, says he now prefers newer terminology
  Let's translate the following terms as we read the paper:
          Virtual address -> guest virtual address
       "Physical" address -> guest physical address
          Machine address -> host physical address
  Similar, we have VPN, PPN, MPN for virtual, physical, machine page number

Note that VMWare workstation (or kvm, virtualbox, etc.) has 4th address type:
  Host virtual memory is memory in processes running in host OS
  That's because VMWare workstation runs as a process on an existing OS
    Re-uses the host OSes device drivers, networking stack, etc.
    E.g., can run emacs and vmplayer side-by-side, where emacs uses host VA
VMWare ESX is different in that it replaces host OS
  Directly accesses NICs and other hardware
  No host networking stack to worry about

In ESX, what are three basic memory parameters for a VM? min, max, shares
  min - VMM always guarantees this much machine memory, or won't run VM
    Actually, need min + ~32MB overhead
  max - this is amount of guest physical memory VM OS thinks machine has
    Obviously VM can never consume more than this much memory
  share - how much host phys. memory this VM should have relative to other VMs
  Big question addressed by this paper: Memory management when over-committed

Straw man: Just page host physical mem to disk with LRU. Why is this bad?
  OS probably already uses LRU, which leads to "double paging" problem
    OS will free and re-use whatever guest physical page VMM just paged out
  Also, performance concerns limit how much you can over-commit
    [Can't modify OS bcopy to use many of Disco's tricks]
  Goal: Minimize memory usage and maximize performance with cool tricks

What happens under memory pressure (Sec 6.3)?
  System can be in one of four states: high, soft, hard, low
    high - (6% free) plenty of memory

```
    soft - (4% free) try getting OSes to give back memory (page as last resort)
    hard - (2% free) use random eviction to page stuff out to disk
    low - (1% free) block execution of VMs above their mem usage targets

How to convince an OS to give back memory?  Ballooning
    Implement special psudo device driver that allocates pinned physical memory
    VMM asks baloon driver to allocate memory
    Baloon driver tells VMM about pages that guest OS will not touch
  How does balloon driver communicate with VMM?
    Section 3.6 says polls once per second to get target balloon size
    Could use any IO mechanism to communicate
      Access special guest physical address (handle via tracing), inb/outb
    Today would use "hypercall" (e.g., vmcall) instruction
    Could also conceivably use interrupts instead of polling
  What happens if balloon memory accessed?
    OS shouldn't touch private balloon driver memory, so VM probably rebooted
    Handle as hidden page fault that VMM satisfies with zero-filled page
    After this, assuming reboot, VMM needs to resynchronize balloon state
  How well does ballooning work (Figure 2)?
    Looks like only small penalty compared to limiting physical memory at boot
    Why the penalty?  OS sizes data structures based on physical memory size
      E.g., might have one "ppage" structure per physical page, will use memory
  Does Figure 2 show that ballooning is effective?
    Would be nice to have third bar with just random eviction paging
    As it is, don't know how much cleverness of technique is buying us

How to share pages across OSes?
  Use hashing to find pages with identical contents
  Big hash table maps hash values onto host physical pages:
    If only mapped once, page may be writable, so hash is only *hint*
      Hash table has pointer back to guest physical page
      Must do full compare w. other page before combining into shared CoW page
    If mapped multiple times, just keep 16-bit reference count
      If counter overflows (e.g., on zero page) use overflow table
  Scan OS pages randomly to stick hints in hash table
  Note:  Always try sharing a page before paging it out to disk
  How well does this work?
    Figures 4, 5 show significant memory savings
  Does it matter for performance?
    p. 7 - tiny 0.5% average speed-up (when no memory contention)
    So this probably means sharing is rarely harmful despite extra hashing
    But would be nice to see actual speedups under limited memory conditions

How does proportional share typically work (no idle tax)?
  Each VM has been assigned number of "shares" S by administrator
    and has been given some number of pages P by VMM
  Reclaim a page from OS with lowest ratio of "shares-to-pages" S/P
    E.g., if A and B both have S=1, reclaim from larger of the two
          if A has twice B's share, then A can use twice as much memory
    Can view S/P as "price" guest OS can afford to pay for a page
  Why is this not good enough?
    May have high-priority VM wasting tons of memory
    (This is reasonable:  high priority VM might sometimes not use memory)
  Digression:  Simple tax arithmetic
    [Ignore payroll, state, and local taxes.]
    Suppose my income tax rate is T (e.g., might be 28% in U.S.)
      For each $1 gross I earn, I pay T*$1 in taxes
      So $1 gross = $(1-T) take home
      And $1 take home requires $1/(1-T) gross pay
    Call k = 1/(1-T) the cost of a take home dollar (e.g., ~$1.39 for 28% tax)
  Idea: idle memory tax.  substituting memory pages for dollars...
    Any page you are using is fully "tax deductible" - just costs one page
    If you aren't using a page, must pay fraction T of it back to the system
      So each idle page actually costs you k times the price of a non-idle page
    Now how much can a VM afford to pay for each "take home" page?
```

$$\text{rho} = \frac{S}{(\#\text{ used pages}) + k*(\#\text{ idle pages})}$$

$$\text{rho} = \frac{S}{P * (f + k(1-f))}$$

  - where f is fraction of active (non-idle) pages, and
    - k is "idle page cost", $k = 1/(1-T)$ for tax rate $0 <= T < 1$
      - at 75% tax rate, k = 4
- So reclaim from VM with lowest rho, instead of lowest S/P

How to determine how much idle memory?
- Statistical sampling: Pick n pages at random, invalidate, see if accessed
  - If t pages touched out of n at end of period, estimate usage as t/n
  - How expensive is this? <= 100 page faults over 30 seconds negligible
- Actually keep three estimates:
  - Slow exponentially weighted moving average of t/n over many samples
  - Faster weighted average that adapts more quickly
  - Version of faster average that incorporates samples in current period
- Use max of 3. Why?
  - Basically, when in doubt, want to respect priorities
    - Spike in usage likely means VM has "woken up"
    - Small pause in usage doesn't necessarily mean it will last longer
  - Anecdote: Behavior when X server paged out by low-priority simulation
- How well does this do?
  - Fig. 7 (p. 9) looks good in terms of memory utilization
  - Would be nice to see some end-to-end throughput numbers, too, though

What is issue with I/O pages and bounce buffers?
- Better to re-locate pages that are often used for I/O