Guest Memory Overcommit

Page hinting, resizing & more

Rik van Riel, Red Hat

KVM Forum 2011

Guest Memory Overcommit

- Why overcommit memory?
- Problems with memory overcommit
- Async pagefault
- Free page hinting
- Memory resizing vs. Transparent hugepages
- Conclusions

Why Overcommit Memory?

- Users want CHEAP virtual machines
  - Prices continuously going down
  - Migrate providers for a dollar/month/VM savings?
- However, they do want it all
  - Always see all the memory they paid for
  - Enough CPU available when they need it
- Overcommit is the way to cheap
  - Share power/hardware/... with more users
  - Hardware is getting cheaper, electricity is not
- Our challenge: make it fast

Problems With Memory Overcommit

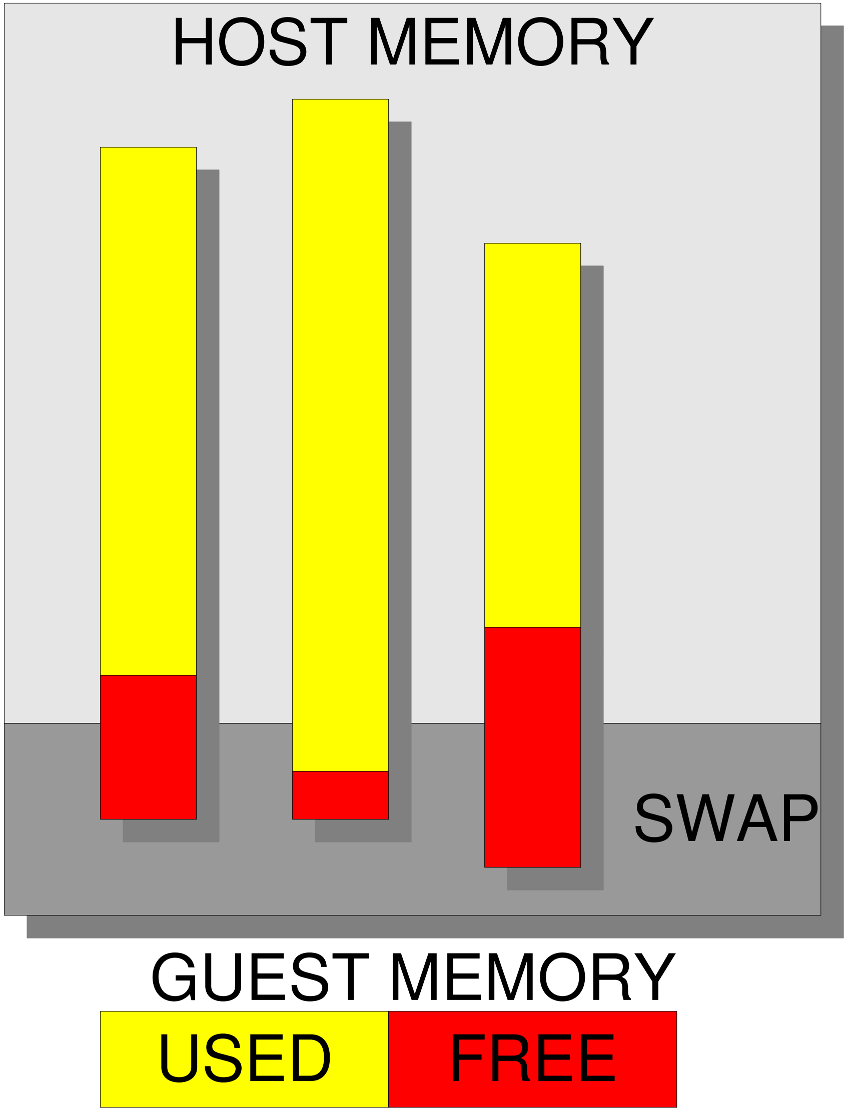
- Memory is a non-renewable resource
- Secondary. Storage. Is. Really. Slow.
  - Many millions of CPU cycles in one disk seek
- Overcommit is easy
  - KVM guest is just like a process
  - Host handles swapping and page faults
- Process in guest accesses non-resident memory
  - Entire VCPU stalls until swapin disk IO is done!
- Host swaps guest page cache and free pages

Async Pagefault

- Host paging blocks the entire VCPU on swapin
- Most swapins are guest processes in sleepable context
  - Anonymous memory
  - Page cache
  - copy_to/from_user
- Guest can suspend the faulting process instead
  - VCPU can run other processes, interrupts, etc
- Implemented by Gleb last year and upstream
- Reduces the impact of host swapins of guest memory
  - Still generates disk IO that slows down others
  - How to reduce the number of host swapins&outs?

Nested LRU Problem

- Three guests
- Host is swapping
- Host swaps out oldest guest pages (often free)
- Guest re-uses free pages for new content
- Swap IO due to free memory
- Content of free pages could be discarded



Free Page Hinting

- Free pages contain no useful information
  - The host could throw away free guest pages!
    - Swapout, KSM, etc
    - Avoid disk IO on swapout
  - Give the guest a fresh page on use
  - Avoid disk IO on swapin
- Guest needs to inform host what memory is free
  - Can use a big bitmap
  - Be careful at state transitions (free->used)

Free Page Hinting Details

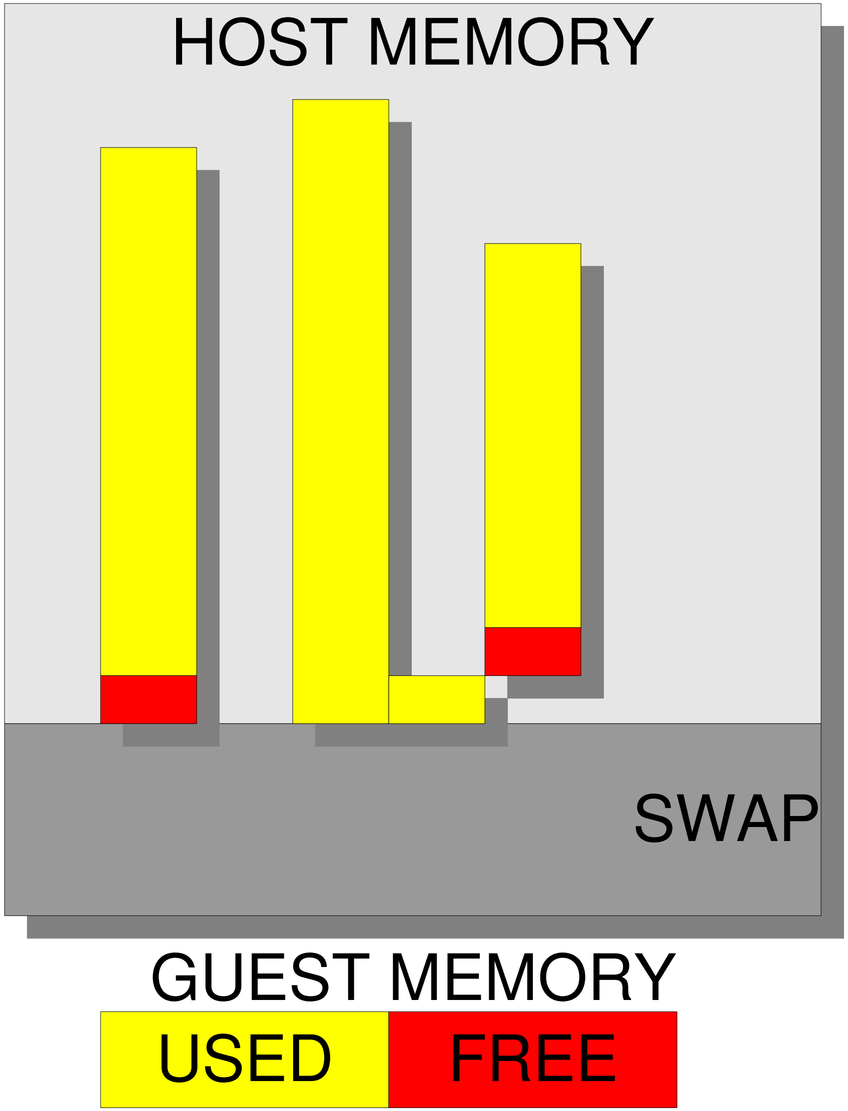
- Keep a large bitmap per guest (or per pgdat)
  - One bit per (4kB) page
  - Use arch_free_page & arch_alloc_page hooks
    - Set bit at free time, clear bit at alloc time
    - Overhead in the guest: touch a bitmap in alloc & free
- On the host side, check bitmap
  - In ksmd, discard unused guest pages
  - At swapout time, discard unused pages
  - At swapin time
    - Give process a fresh page
    - Free swap space
    - Skip swap IO
- Host side is more overhead, but only when memory is tight
- Interface may also be usable by eg. a JVM

Free Page Hinting Race Conditions

- Swapout & ksmd discard vs. unused->used transition
  - Check page unused bit in bitmap
  - Unmap page from guest
  - Re-check bit in bitmap
    - If still set, discard page
    - If now clear, remap the same page into the guest
  - Hold the right lock in the host to block page faults by the process, while doing the re-check
- Swapin IO avoidance
  - Can avoid IO if page is touched while the “unused” bit for the page is still set
  - Page allocator in guest kernel touches the page, before clearing the bit

Nested LRU With Free Page Hinting

- Many free pages eliminated
    - Swapout & ksmd
- Guests now fit in RAM
    - The used memory...
- Swap IO greatly reduced
    - We'll never catch them all
- But what if the sum of used memory exceeds RAM?

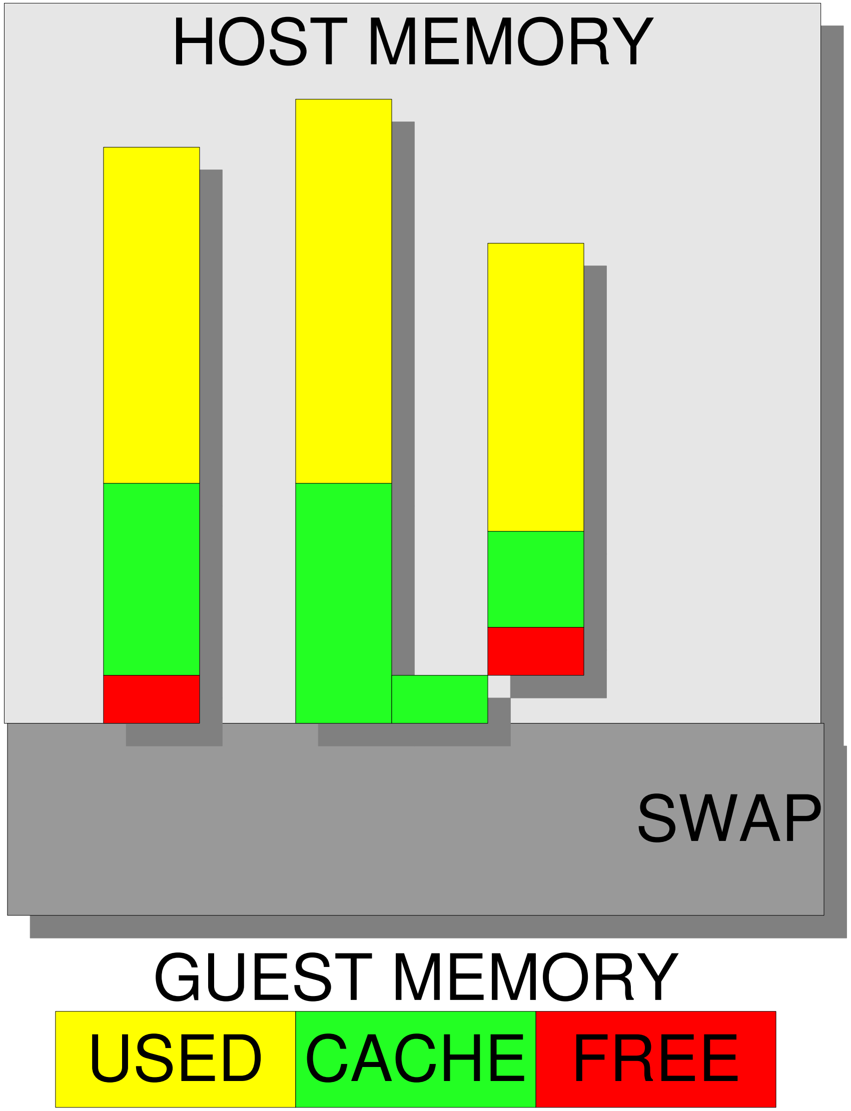


Dynamic Memory Resizing

- We can do more than eliminate free memory
  - Guest has working set, page cache and free
- When host is near swapout
  - Ask each guest to free some memory
  - Do not shrink guest below minimum size
- When a guest is near swapout
  - Do not shrink memory when asked (or not much)
  - Ask the host for some memory back
  - Do not grow guest above maximum size

Dynamic Memory Resizing Illustrated

- Ask guests to keep extra memory free when host needs it
- Obey guest min/max size
- Avoid even more swap IO
- Combine with cgroups for guest prioritization
- Fit more guests per host



To Balloon or not to balloon?

- Guest resizing traditionally done through a balloon
  - Hypercall per freed page (fixable with batching)
  - Hopeless memory fragmentation
    - Big problem for Transparent Huge Pages (THP)
    - Defragmentation would touch more memory
      - While the host is already under memory pressure
- May be better off adjusting the guest free memory targets
  - Automatically helps defragment memory inside a guest
    - Good for THP and slab/slub
  - Free page hinting can be used to physically free unused guest pages on the host
  - A lot fewer hypercalls than any balloon implementation

Conclusions

- Users want faster, cheaper & more
  - Whoever can provide that will be the industry standard
  - Needs modifications to both guest and host to work best
- KVM already provides cheaper & more
  - Host swapping
  - Async pagefault reduces impact of host swapping
- To go faster, we must reduce the IO
  - Skip IO on free pages
  - Resize the amount of used memory in a guest
    - Reduce if lightly used, increase if heavily used
    - Depending on memory pressure in host
- Your ideas here...