# VMWare

CS380L: Emmett Witchel

> *"A new ballooning technique reclaims memory from a VM by implicitly causing the guest OS to invoke its own memory management routines. An idle memory tax was introduced to solve an open problem in share-based management of space-shared resources"*
> Carl A. Waldspurger **Memory Resource Management in VMware ESX Server**

## 1 Overview

- What a corporate development budget buys you. (And elegant solutions to two long standing research problems)
- Background: VMWare ESX runs on bare hardware (sometimes called a bare metal hypervisor). Other VMWare products use the "hosted" approach where a "control OS" actually talks to hardware. In a hosted VM, an application does a read from a file, the OS translates that into a read from a block device, and the VMM translates that to a read from a file from the host OS (which is translated to a read from the machine's block device).

## 2 Resource management from the device level

Like they mentioned in the Disco paper, managing memory from the virtual machine level has many disadvantages. Xen statically partitions memory to ease its management.

- The VM page replacement algorithm can pick a page important to the guest OS. Causes performance anomalies.
- *D*ouble paging problem: If VM pages out first, OS page out will cause VM reclaim.
- OSes do not have a facility for changing amount of physical memory at runtime.

## 3 Ballooning

Force guest OS to use its own page replacement algorithm & communicate choice (implicitly) to VM monitor.

- Download VMWare balloon module into guest OS
- Inflate balloon to get OS to free (page out) memory
- Deflate balloon to get OS to use more memory
- Pages allocated to balloon have their entry in the `pmap` marked, and can be reclaimed by VM
- If guest touches a balloon page, allocate a new page and all bets are off.

Problems

- Might not be able to reclaim memory fast enough (VMWare rate-limits allocation)
- Guest OS might refuse memory allocation request or limit the driver's ability to allocate memory.
- Can always resort to paging. Use randomize algorithm to avoid pathologically bad cases of paging out exactly what guest OS needs.

VMWare PFN to MPN mapping preserves page coloring. Review page coloring. 1MB cache with 4KB pages has 256 colors.

See Figure 2 for results.

# 4 Content-based sharing

Disco's sharing mechanism required modification to the OS. Share by content avoids that restrictions. Sharing by content is a trick used in other systems (like Venti, which tracks version histories for file systems, and peer-to-peer systems).

Disco shared memory with paravirtualization, they changed the OS. The modified `bcopy`, made a network with a large maximum packet size, and interposed on disk accesses.

- Hash every page.
- Store hashes in a hash table.
- On collision, check if pages are identical. If they are, share copy-on-write.
- With no collision, store hash as hint. On future collision check if hint is still valid (page contents have not changed). If it is, share page.

Data structure details

- 16 byte records (0.5% of system memory)
- Share record: hash value, MPN, ref count, chain link
- Hint record: truncated hash value, guest page reference (VM id and PPN)
- Don't store backmap, store ref count for shared pages with overflow table for widely shared pages.
- Scan randomly, don't take too much CPU time (CPU overhead measured negligible)
- **How is memory reduced for the 1 VM case?**

*W*hy can VMWare use a reference count while Disco required a backmap?

Measurements in Figure 4.

# 5 Managing memory

The problem with proportional share allocators is that they let rich clients hoard resources. VMWare adds a tax to idle memory.

- Inflate cost of idle memory by tax rate
- Allow 25% idle memory to provide a buffer for a fast-growing working set increase.

- Only need percentage of idle memory, so measure it by random sampling (great low-overhead technique).
- Experiment to show it works (in one typical case).
- Multiple exponentially-weighted averages ($ewa_n$ = gain * new_value + (1 - gain) * $ewa_{n-1}$). Experiments show it tracks memory use well. Rapid response to increasing memory use, slower response to decreasing use.

VMs requires 32MB of overhead memory for VMs up to 1GB. *W*here does it store physical to machine mappings?

Page in high physical memory (above 4GB) which are the target of I/O are transparently remapped to low memory.

Four states of memory use, with hysteresis.

- **high**: No reclamation
- **soft**: Balloon, page when necessary (share before swap)
- **hard**: Page
- **low**: Suspend VM

## 6 Evaluation notes

- Each data point in Figure 2 is the average of 3 runs.
- Data on memory sharing nicely blends controlled experiments with real data.

## 7 Discussion

From a VMWare blog post, 10/08/2008. http://blogs.vmware.com/virtualreality/2008/10/memory-overcomm.html

I bet MSFT will no longer question the value of overcommit once they are finally able to list it as an upcoming Hyper-V feature.

We conducted an online survey of 110 VMware customers essentially asking them three questions:

- Do you use memory overcommit?
  - 57% answered they are using memory overcommit
- Do you use memory overcommit in test/dev, production or both?
  - Of the 57% who answered yes, 87% said they use it in production and test and development, 2% only in production, 11% only in test/dev.
- What is your virtual-to-physical memory ratio per ESX host (i.e., overcommit ratio)?
  - Virtual-to-physical memory ratios ranged from 1.14 to 3 (average 1.8, median 1.75). 75% of the respondents use memory overcommit ratios of 1.5 or higher and 37% utilize a ratio of 2 or higher.

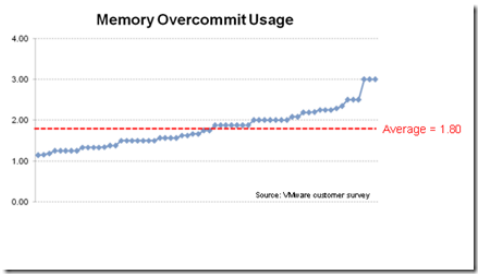


Figure 1: Memory overcommit usage.

# 8 Compatability is not transparency

*"We conclude that building a transparent VMM is fundamentally infeasible, as well as impractical from a performance and engineering standpoint."*

*Soon, malware that limits itself to non-virtualized platforms will be passing up a large percentage of commercial, military and institutional targets.*

*Perhaps the most concise argument against the utility of [virtual machine-based rootkits] is: "Why bother?" VMBRs change the malware defender's problem from a very difficult one (discovering whether the trusted computing base of a system has been compromised) to the much easier problem of detecting a VMM.*

Garfinkel, Adams, Warfield, Franklin **Compatibility is Not Transparency: VMM Detection Myths and Realities**

## 8.1 Differences between real and virtual

- **Instruction semantics.** Some non-virtualizable instructions (e.g., SIDT, SGDT, SSL) can examine privileged state, and they are performance and semantically unimportant for the VMM, so they are not virtualized. Hardware support is unlikely to completely close this gap.
- **Chipset features.** Chipsets are difficult to model in detail, VMWare uses a very old motherboard model.
- **Paravirtualized devices.** I/O paravirtualization creates odd devices (e.g., network cards with huge maximum packet sizes), and VMWare uses their own identification strings.
- IOMMU and device pass-through support may erode this point (restartable semantics for DMA faults?).
- **Size of shared hardware structures.** Virtual TLB, instruction and data caches are effectively smaller. Modify page table, don't sync TLB and then access newly mapped memory.
- **Local timing.** Some operations run more slowly (e.g., syscall), some faster (e.g., device register read). The ratio of latencies is very hard to get right (nop vs. cpuid). Any timing channel makes VMM detectable. "While [hardware virtualization] changes the timing fingerprint of the virtual environment relative to a software-only approach, the fingerprint still differs from that of native execution."
  - **Noise.** Noise can help, but it eventually reduces throughput.
  - **Time dilation insufficient.** It must dialate all latencies equally. Does this require machine simulation?

- **External timing.** External timing sources are pretty much unstoppable. If the machine has a network, timing attacks to detect the VMM are inevitable.
- **Optimize the common case.** Optimizing the common (performance) case for VMs often means creating a detectable performance cliff, e.g., caching shadow page tables.

### 8.2 Virtualization doesn't need to be transparent

- Many platorms, from desktop to server run under a VM. Reasons for running a VM: boot a different OS, overbook hardware resources, increase security.
- VM-based rootkits are easier to detect than OS-based rootkits, and so unlikely to gain popularity. A minimal VMM that refused to load signed VMMs works.