Memory Overcommit… without the Commitment

Speaker: Dan Magenheimer
Oracle Corporation
Overview

- What is overcommitment?
  - aka oversubscription or overbooking
- Why (and why not) overcommit memory?
- Known techniques for memory overcommit
- Feedback-directed ballooning
CPU overcommitment

Four *underutilized* 2-cpu virtual servers

One 4-CPU physical server

Xen supports CPU overcommitment (aka “consolidation”)
I/O overcommitment

Four *underutilized* 2-cpu virtual servers each with a 1Gb NIC

One 4-CPU physical server with a 1Gb NIC

Xen supports I/O overcommitment
Memory overcommitment???

Four *underutilized* 2-cpu virtual servers each with 1GB RAM

One 4-CPU physical server w/4GB RAM

SORRY! Xen *doesn’t* support memory overcommitment!
Why doesn’t Xen overcommit memory?

- Memory is cheap – buy more
- “When you overbook memory excessively, performance takes a hit”
- Most consolidated workloads don’t benefit much
- Overcommit requires swapping – and Xen doesn’t do I/O in the hypervisor
- Overcommit adds lots of complexity and latency to important features like save/restore/migration
- Operating systems know what they are doing and can use all the memory they can get
Why doesn’t Xen overcommit memory?

- Memory is cheap – buy more… except when you’re out of slots or need BIG dimms.
- “When you overbook memory excessively, performance takes a hit”… yes, but true of overbooking CPU and I/O too. Sometimes tradeoffs have to be made.
- Most consolidated workloads don’t benefit much… but some workloads do!
- Overcommit requires swapping – and Xen doesn’t do I/O in the hypervisor… only if black-box swapping is required
- Overcommit adds lots of complexity and latency to important features like save/restore/migration… some techniques maybe… we’ll see…
- Operating systems know what they are doing and can use all the memory they can get… but an idle or lightly-loaded OS may not!
Why should Xen support memory overcommitment?

- Competitive reasons
  
  "VMware Infrastructure’s exclusive ability to overcommit memory gives it an advantage in cost per VM that others can’t match" *

- High(er) density consolidation can save money

- Sum of guest working sets is often smaller than available physical memory

- Inefficient guest OS utilization of physical memory (cacheing vs “hoarding”)

Problem statement

Oracle OnDemand businesses (both internal/external):
• would like to use Oracle VM (Xen-based)
• but uses memory overcommit extensively

The Oracle VM team was asked… can we:
• implement memory overcommit on Xen?
• get it accepted upstream?
Memory Overcommit Investigation

• Technology survey
  • understand known techniques and implementations
  • understand what Xen has today and its limitations

• Propose a solution
  • OK to place requirements on guest
    • e.g. black-box solution unnecessary
  • soon and good is better than late and great
    • phased delivery OK if necessary
      • e.g. Oracle Enterprise Linux now, Windows later
  • preferably high bang for the buck
    • e.g. 80% of value with 20% of cost
Techniques for memory overcommit

- Ballooning
- Content-based page sharing
- VMM-driven demand paging
- Hot-plug memory add/delete
- Ticketed ballooning
- Swapping entire guests

Black-box or gray-box* … or white-box?

WHAT IF.....?

Operating systems were able to:

• recognize when physical memory is not being used efficiently and communicate relevant statistics
• surrender memory when it is underutilized
• reclaim memory when it is needed

And Xen/domain0 could balance the allocation of physical memory, just as it does for CPU/devices?

.... Maybe this is already possible?!?
Ballooning (gray-box)

- In-guest device driver
  - “steals” / reclaims memory via guest in-kernel APIs
    - e.g. `get_free_page()` and `MmAllocatPagesforMdl()`
- Balloon inflation increases guest memory pressure
  - leverages guest native memory management algorithms
- Xen has ballooning today
  - mostly used for domain0 autoballooning
  - has problems, but recent patch avoids worst*
- Vmware and KVM have it today too

Issues:
- driver must be installed
- not available during boot
- reclaim may not be fast enough; potential out-of-memory conditions

Content-based page sharing (black-box)

- One physical page frame used for multiple identical pages
  - sharing works both intra-guest and inter-guest
  - hypervisor periodically scans for copies and “merges”
  - copy-on-write breaks share
- Investigated on Xen, but never in-tree* **
  - measured savings of 4-12%
- Vmware*** has had for a long time, KVM soon ****

Issues:
- Performance cost of discovery scans, frequent share set-up/tear-down
- High complexity for relatively low gain

Demand paging (black-box)

- VMM reclaims memory and swaps to disk
- VMware has today
  - used as last resort
  - randomized page selection
- Could potentially be done on Xen via domain0

Issues:
- "Hypervisor" must have disk/net drivers
- "Semantic gap"* → Double paging

Hotplug memory add/delete (white-box)

- Essentially just ballooning with:
  - larger granularity
  - less fragmentation
  - potentially unlimited maximum memory
  - no kernel data overhead for unused pages

Issues:
- Not widely available yet (for x86)
- Larger granularity
- Hotplug delete requires defragmentation

“Ticketed” ballooning *(white-box)*

- Proposed by Ian Pratt*
- A ticket is obtained when a page is surrendered to the balloon driver
- Original page can be retrieved if Xen hasn’t given the page to another domain
- Similar to a system-wide second-chance buffer cache or unreliable swap device
- Never implemented (afaik)

Whole-guest swapping (black-box)

- Proposed by Keir Fraser*
- Forced save/restore of idle/low-priority guests
- Wake-on-LAN-like mechanism causes restore
- Never implemented (afaik)

Issues:
- Very long latency for guest resume
- Very high system I/O overhead when densely overcommitted

Observations

• Xen balloon driver works well
  • recent patch avoids O-O-M problems
  • works on hvm if pv-on-hvm drivers present
  • ballooning up from “memory=xxx” to “maxmem=yyy” works (on pvm domains)
  • ballooned-down domain doesn’t restrict creation of new domains
• Linux provides lots of memory-status information
  • /proc/meminfo and /proc/vmstat
  • Committed_AS is a decent estimator of current memory need
• Linux does OK when put under memory pressure
  • rapid/frequent balloon inflation/deflation just works… as long as remaining available Linux memory is not too small
  • properly configured Linux swap disk works when necessary; obviates need for “system-wide” demand paging
• Xenstore tools work for two-way communication even in hvm
Proposed Solution: Feedback-directed ballooning (gray-box)

Use relevant Linux memory statistics to control balloon size

• Selfballooning:
  • Local feedback loop; immediate balloon changes
  • Eagerly inflates balloon to create memory pressure
  • No management or domain0 involvement

• Directed ballooning:
  • Memory stats fed from each domainU to domain0
  • Policy module in domain0 determines balloon size, controls memory pressure for each domain (not yet implemented)
Implementation: Feedback-directed ballooning

- No changes to Xen or domain0 kernel or drivers!
- Entirely implemented with user-land bash scripts
  - Self-ballooning and stat reporting/monitoring only (for now)
- $Committed_{AS}$ used (for now) as memory estimator
- Hysteresis parameters -- settable to rate-limit balloon changes
- Minimum memory floor enforced to avoid O-O-M conditions
  - same maxmem-dependent algorithm as recent balloon driver bugfix
- Other guest requirements:
  - Properly sized and configured swap (virtual) disk for each guest
  - HVM: pv-on-hvm drivers present
  - Xenstore tools present (but not for selfballooning)
Feedback-directed Ballooning Results

- Overcommit ratio
  - 7:4 w/default configuration (7 512MB loaded guests, 2GB phys memory)
  - 15:4 w/aggressive config (15 512MB idle guests, 2GB phys memory)
  - for pvm guests, arbitrarily higher due to “maxmem=”

- Preliminary performance
  (Linux kernel make after make clean, 5 runs, mean of middle 3)

<table>
<thead>
<tr>
<th>ballooning</th>
<th>Memory (MB)</th>
<th>Min (MB)</th>
<th>User (sec)</th>
<th>Sys (sec)</th>
<th>Elapsed (sec)</th>
<th>Major page faults</th>
<th>Down Hysteresis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>2048</td>
<td>785</td>
<td>121</td>
<td>954</td>
<td>0</td>
<td>8940</td>
<td>10</td>
</tr>
<tr>
<td>Self</td>
<td></td>
<td>368</td>
<td>778</td>
<td>104</td>
<td>1209</td>
<td>8520</td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td>1024</td>
<td>775</td>
<td>95</td>
<td>1120</td>
<td>4650</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self</td>
<td></td>
<td>238</td>
<td>775</td>
<td>93</td>
<td>1201</td>
<td>9490</td>
<td>10</td>
</tr>
<tr>
<td>Off</td>
<td>512</td>
<td>775</td>
<td>79</td>
<td>1167</td>
<td>8520</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self</td>
<td></td>
<td>172</td>
<td>775</td>
<td>80</td>
<td>1202</td>
<td>9650</td>
<td>10</td>
</tr>
<tr>
<td>Off</td>
<td>256</td>
<td>773</td>
<td>79</td>
<td>1186</td>
<td>9150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self</td>
<td></td>
<td>106</td>
<td>775</td>
<td>80</td>
<td>1202</td>
<td>9650</td>
<td>10</td>
</tr>
</tbody>
</table>

⇒ Selfballooning costly for large-memory domains but barely noticeable for smaller-memory domains
Domain0 screenshot with monitoring tool and xentop showing memory overcommitment
Future Work

• Domain0 policy module for directed ballooning
  • some combination of directed and self-ballooning??

• Improved feedback / heuristics
  • Combine multiple memory statistics, check idle time
  • Prototype kernel changes (“white-box” feedback)
    • Better “idle memory” metrics

• Benchmarking in real world

• More aggressive minimum memory experiments

• Windows support
Conclusions

• Xen *does* do memory overcommit *today*!
• Memory overcommit has some performance impact
  • but still useful in environments where high VM density is more important than max performance
• Lots of cool research directions possible for “virtualization-aware” OS memory management
Memory Overcommit... without the Commitment

Speaker: Dan Magenheimer
Oracle Corporation

2008