Memory overcommit for overcommitted admins

Presented by Jonathan Davies
Based on the work of David Vrabel
Outline

1. Memory overcommit: Why? How?
2. Design approach
3. Rapid prototyping by simulation
4. Conclusion
Resource overcommit

In many virtualization scenarios, resource overcommit is welcome.

Users understand that performance will be impacted when over-committing.

CPUs  GPUs  Storage capacity  Network bandwidth
Resource overcommit

In many virtualization scenarios, resource overcommit is welcome.

Users understand that performance will be impacted when over-committing.
Why memory overcommit?

This host is “full”.

But in some cases the VMs may not need all their memory all the time.

➢ e.g. bursty workloads where VMs are often idle
➢ e.g. the user who demands a 16 GB VM but will only ever use 4 GB

If we need to, we should be able to find enough room in which to start another VM
➢ e.g. to cope with unforeseen short-term capacity requirements
Memory overcommit techniques

- Memory hotplug/unplug
- Memory ballooning
- Hypervisor paging
Memory hotplug/unplug

Add and remove virtual DIMMs

Issues

➢ Removing memory is hard, requiring OS support, so will not work for all VMs
➢ No guarantee over timely release
➢ Administrator needs to indicate how much memory it is “safe” to remove
Memory ballooning

Reclaim unused pages from running VMs

Issues

➢ Requires an in-guest driver, so VMs must be trusted to co-operate
➢ VM reboot resets balloon, so bulk reboots could cause OOM
➢ No guarantee over timely release
➢ Administrator needs to indicate how much memory it is “safe” to remove
Hypervisor paging

Shift some of a VM’s memory into backing store

Issues

➢ Slow to swap pages back in when needed
➢ Requires paging infrastructure in hypervisor
Outline

1. Memory overcommit: Why? How?
2. Design approach
3. Rapid prototyping by simulation
4. Conclusion
Our design goals

- Automatic sizing
- OS agnostic
- Agentless
- Fair
- Disable-able
Chosen approach

Use hypervisor paging

✓ Hypervisor paging is straightforward with Linux
✓ No OS co-operation required
✓ No in-guest agent required
✗ Won’t work with VMs with PCI-passthrough
✗ Cannot use memory allocated from hugetlbfs

Use feedback from the system to automatically size VMs

✓ No guesswork or manual configuration of memory sizes
✓ Minimise amount of swapping needed
Control paging using cgroups

Each VM sits in two cgroups:

1. A per-VM cgroup. Its memory size is controlled by adjusting `memory.limit_in_bytes`.

2. For VMs with overcommit enabled, a host-wide cgroup with swap enabled.
   For VMs with overcommit disabled, a host-wide cgroup with swap disabled.

Use a per-host swap device.
Automatic VM sizing

A “squeezer” daemon maintains an estimate of each VM’s working set.

The squeezer estimates the VM’s working set size from qemu major fault rates:

- Low fault rate → reduce estimate
- High fault rate → increase estimate

The squeezer sets the VM’s cgroup memory limit to this estimate.

1 by observing majflt + cmajflt from qemu process’s /proc/<pid>/stat file
Automatic VM sizing

- Configured VM memory size
- Working set estimate
- Actual working set

High fault rate so increase estimate
Low fault rate so reduce estimate
Problem: excessive paging

- Configured VM memory size
- Working set estimate
- Actual working set

Bad decision to squeeze 😞

Bad decision to squeeze 😞
What if several VMs on a host increase their working set at around the same time?

If the sum of VMs’ working sets exceeds host memory, the host is overcommitted.

- VM performance will be heavily impacted.
- Computation of working set estimates cannot occur.

We may be able to eventually recover by migrating VMs to other hosts.

But we want to avoid this scenario whenever possible.
Solution: track long-term trends

Configured VM memory size

Working set estimate

Actual working set
Outline

1. Memory overcommit: Why? How?
2. Design approach
3. Rapid prototyping by simulation
4. Conclusion
How to evaluate a squeezing algorithm?

Goal: maximise squeezing while minimising performance degradation

There are several important parameters for the squeezer, including

1. How quickly should the squeezer react to an increase in fault rate?
2. When the fault rate is low, when should the squeezer try decreasing the cgroup limit?
3. By what amount should the squeezer increase or decrease the cgroup limit?
4. What length of time window should be used for the long-term estimate?

How can we find an optimal squeezing algorithm?
Use simulation to aid rapid prototyping

Approach:

1. Monitor
   Monitor real guest memory usage patterns under real-world workloads

2. Model
   Construct a model that can quantify the impact of a squeezer algorithm on a VM workload

3. Simulate
   Rapidly evaluate possible squeezer algorithms
We used idle page tracking\textsuperscript{1} to monitor the page-accessing activity of guests under real world workloads.

The file \texttt{/sys/kernel/mm/page_idle/bitmap} contains one bit per physical page on the host.

When a page is accessed, the corresponding bit is cleared.

We periodically sample the pages corresponding to a VM’s memory, resetting the bits after each sample.

\textsuperscript{1} available since Linux 4.3, see Documentation/mm/idle_page_tracking.txt
Example 1: git clone in Linux
Example 1: git clone in Linux
Example 1: git clone in Linux
Example 1: git clone in Linux
Example 2: kernel build in Linux
Example 2: kernel build in Linux
Example 2: kernel build in Linux
Example 2: kernel build in Linux
Example 3: Youtube video in Windows
Example 3: Youtube video in Windows
Example 3: Youtube video in Windows
Example 3: Youtube video in Windows
Converting into a model

A simple model that approximates these three data sets:

➢ Split PFNs into two sets: used and unused
➢ Used pages will be accessed according to a normal distribution
➢ Unused pages will not be accessed

We can model variation over time by varying the model’s parameters.
Linux memory subsystem simulator

We have created **psim** to simulate a VM performing a workload and measure the amount of work done in a given time

- [https://github.com/jjd27/psim](https://github.com/jjd27/psim)

For every simulator tick:

- Workload determines which page the vCPU accesses
  - Accessing a free or unmapped page incurs a minor fault, blocking the vCPU from doing work for a short time
  - Accessing a paged-out page incurs a major fault, blocking the vCPU from doing work for a prolonged time

- vCPU does a unit of work on the page

Simulate Linux memory subsystem behaviour:

- Periodically scan memory to update active/inactive lists according to page accesses since last scan
  - Page out inactive pages if the number of allocated pages exceeds cgroup memory limit

Simulated workload for rapid prototyping

An example workload, looping forever:

- Access 50% of pages according to a normal distribution (one page per tick for 100k ticks)
- Access 75% of pages according to a normal distribution (one page per tick for 100k ticks)

Run the squeezer algorithm every 1k ticks
Memory limit = 64 pages, i.e. 50% of static allocation
Total work done = 436k units of work, i.e. 87.1% efficiency
Simple automatic squeezing

Algorithm: Reduce/increase limit by 1 page if less/more than 2 major faults in period

Average memory limit = 68.9 pages, i.e. 53.8% of static allocation
Total work done = 460k units of work, i.e. 92.0% performance
Proportional squeezing

Algorithm: Reduce/increase limit by number of major faults in period

Average memory limit = 71.9 pages, i.e. 56.2% of static allocation
Total work done = 477k units of work, i.e. 95.3% performance
Smooth out recent fault rate

Algorithm: Reduce/increase limit by number proportional to weighted average of the number of faults since the start

Average memory limit = 68.4 pages, i.e. 53.4% of static allocation
Total work done = 468k units of work, i.e. 93.5% performance
Or, be more conservative

Algorithm: Hard squeeze until first page fault; soft squeeze if we get fewer than 2 faults in 100 intervals

Average memory limit = 102.7 pages, i.e. **80.2%** of static allocation
Total work done = 497k units of work, i.e. **99.4%** performance
Conclusions

➢ Memory overcommit is beneficial for some use-cases and workloads
➢ Hypervisor paging does not require guest co-operation
➢ Automatic VM memory sizing avoids administrative effort and guesswork
➢ A squeezer algorithm can be evaluated using simulation based on real-world data
➢ A good squeezer algorithm can maintain good levels of VM performance while squeezing VMs close to their working set
Memory overcommit for overcommitted admins

Presented by Jonathan Davies
Based on the work of David Vrabel

OCTOBER 2018 | KVM FORUM