KVM Memory Optimizations

Increasing memory density with KSM, and more

Prateek
Advisor: Puru

March 22, 2011

High-Level Goals

- Increase number of VMs without degrading performance.
- Improve how guests (and hosts) utilize physical memory.
- Why Memory : Most constrained and non-renewable resource.
- Catastrophic costs when used wrong : Page-fault vs performance extremely non-linear.
- Memory overcommit is one of the key drivers of virtualized hosting
- *Everyone wants 8GB, even if they are using a few MB*

Page-Sharing

KSM & Page-Sharing

- KSM implements *Content Based Page Sharing* in linux.
- Multiple pages with the same content are merged into one.
- How to detect similarity among ever-changing objects(pages) ?
- Brute-force search at regular intervals : scanning.
- Different implementations : VMWare ESX, Difference Engine, Satori, KSM.

This talk: Describe 3 modifications to KSM

KSM operation

In this figure, we see how KSM operates.

KSM

The Good:

- Very general and non-disruptive solution - meant for page sharing in arbitrary VM areas.
- Any (anonymous) memory area can be made shareable (by any process, not just KVM/QEMU)
- Very few heuristics used.

The Bad:

- Significant overhead
- 10-20 % CPU utilization in most cases.

The Ugly :

- Design constrained by a patent minefield.

Page Sharing by Flags

- Hypothesis: Sharing page-cache pages is bad.
- Page-cache pages are overwritten frequently anyway.
- Page-cache size is 50% of available memory , so significant KSM savings.
- This hypothesis turns out to be *wrong*

Implementation

- Guest writes its page-flags into a memory hole.
- Host (KSM) needs to access statically defined address in the guest. HOW?
- Kernel doesnt seem to have a mechanism to provide memory by physical addresses
- Currently : Create memory-hole at boot-time and write to it (using `ioremap`)

Lookahead

- Lots of shared pages are file-backed.
- Peek at next page before doing the tree-search
- Assuming consecutive shared pages occour with probability p , Reduce search costs from log(u) to (1-p)log(u) + (p)1.

Lookahead results

Lookahead doesnt work in all cases.Figure: lookahead success and KSM % cpu difference for 3 workloads: static, desktop, kernelcompile

Lookahead conclusion

- Lookahead optimization has no overhead in the worst case.
- Shared pages **increase** because KSM can scan lot more pages per CPU cycle.
- Ideal situations: sharing of large files.
- Desktop environments are best. (X11 fonts, programs, etc).
- Sub-optimal situations: Lots of sharing, but fragmented. (Kernel compile)

Exclusive Caching

- All guest disk IO goes through host page-cache.
- Double-caching is a problem. Same block is present in both host and guest caches.
- Clearly this is a waste
- Exclusive caches are known to provide better cache utilization. [See: Geiger, gill, mycacheyours].

- Mounting virtual disks as $O_D IRECTaddstoomuchpenalty$

Double caching problem

Fig shows percentage of double cached pages in host cache for various workloads

Exclusive Caching

- Wasting memory by caching 2 copies clearly wasteful
- No existing solution for virtual setups
- Use KSM!
- Drop a page from host page-cache if it's already present in guest.
- Luckily for us, KSM builds a nice search tree of all guest pages!
- Scan at the end of every KSM pass, comparing host page-cache pages with unstable,stable tree.
- If match found, drop page from host cache

Benefits

TODO

- More experiments, with different workloads.
- Need low-intensity, high-sharing benchmarks.
- KSM designed for 'idle' loads. How to evaluate idleness?