Memory CoW in Xen

Talk overview

- Why is CoW need?
- Memory CoW basics
- CoW mechanism:
  - Establishing sharings
  - No-Copy-on-Write
  - Asynchronous memory reclamation
- Conclusions

Why is memory CoW needed?

- In VMMs there is a relatively large degree of memory duplication (Waldspurger claims 67% for synthetic, up to 42.9% on "real world" workloads)
- Intra-domain duplication is already exploited by the OSes (CoW on fork, shared libraries, global page cache), but Inter-domain duplication is out of the guests' reach

CoW basics

- Once two pages are detected to have the same content, only one copy kept, mappings turned RO
- On write fault create a private copy (unless there is only one mapping left)
- Memory is needed to satisfy the write faults. At the worst case we might need all the freed memory back

CoW mechanism

CoW issues

- How to design the low-level mechanism for sharing + fault handling?
- How to find pages to share?
- How to minimise the impact of CoW faults?
- Where to get the memory to satisfy CoW write faults?

CoW and shadow PTs

- Sharing difficult with direct page tables
  - Reverse mappings needed
  - Remapping (either to establish a sharing or to satisfy a fault) difficult
- Shadow page tables already managed by the hypervisor, relatively easy to add OS agnostic reverse mappings
- Write faults can be handled transparently
- However PV guests don't use shadow PTs by default

get_page => get_gfn

- Xen memory management assumes exclusive ownership of pages (i.e. single owner), grant tables + direct foreign mappings handled as special cases
- Synthetic cow_domain introduced (acts as an owner). All page gets/puts turned into GFN gets/puts (they take an extra domain argument)
- gfn_info structures created to store counts for shared pages

share_mfns hypercall

- Two extra hypercalls hypercalls added: mark_page_ro and share_mfns
- share_mfns(source_mfn, client_mfn) prompts Xen to remap all client_mfn's gfns to source_mfn
- share_mfns fails if either source_mfn or client_mfn weren't previously marked RO with mark_ro(mfn)

Detecting duplicate pgs

- Underlying principle: use cheap heuristics where possible, avoid content hashing
- PV interfaces treated as a source of hints
  - CoW disk: the CoW relationships can be effectively brought into memory
  - location of shared libs?
- CoW daemon in Dom0 aggregates hints, verifies duplication, administers share_mfns

No-Copy-on-Write

- Let's consider page allocation cycle:
  1. a free page (content logically zero) is allocated
  2. the page is freed
  3. page is scrubbed [possibly]
  4. goto 1.
- If write fault in 1. or in 3. there is no need to copy as the page is logically zero

Repayment FIFO

- Xen doesn't overcommit memory, paging is the guests' responsibility
- Guest might be unable/unwilling to give up the memory required to resolve faults, repayment guarantees needed ahead of time
- Guest maintained repayment FIFO containing volatile pages introduced, |FIFO| >= #bonus pages
- IBM's page hinting patch: volatile pages in Linux

Repayment FIFO in HVM

- HVM domains can take advantage of extra memory indirectly
- Service domains/Dom0 can usefully accommodate extra memory: e.g. a memory based swap file can be created
- Repayment FIFO will be maintained by the service domain

Future directions

- Filesystem level I/O (note JavaGuest project)
- Read Only memory
- Budy-like sharing: hashes for ^2 memory blocks, allows to efficiently share more then a single page
- Domain building: forking a RO “domain stub”
- Security consideration: CoW creates a new covert channel, need a way of protecting sensitive memory (e.g. ssh keys)

Questions?