Transparent Hugepage

Red Hat Inc.

Andrea Arcangeli
aarcange at redhat.com

11 Nov 2009

redhat.

Agenda

- Benefit of hugepages
- Hugetlbfs troubles
- Transparent Hugepage objectives
- Transparent Hugepage possible implementations
- Transparent Hugepage design

redhat.

Benefit of hugepage

- Enlarge tlb size
  - TLB is sparate for 4k and 2m pages thoguh
- Speedup tlb miss
  - Need 3 accesses to memory instead of 4 to refill the TLB
- Faster to allocate
  - Initial page fault huge speedup (like 50% faster)
  - clear_page/copy_page less cache friendly though

Benefit of hugepage

- NPT/EPT tlb miss cost (not counting data access)
  - Guest not using hugepages, KVM not using hugepages
    - 4 guest levels x 5 NTP accesses + 4 NTP accesses for final gpa->hpa translation = 24 memory accesses
  - Guest not using hugepages, KVM using hugepages
    - 4 guest levels x 4 NTP accesses, 3 accesses final gpa->hpa = 19 accesses
  - Guest using hugepages, KVM using hugepages
    - 3 x 4 + 3 = 15 accesses

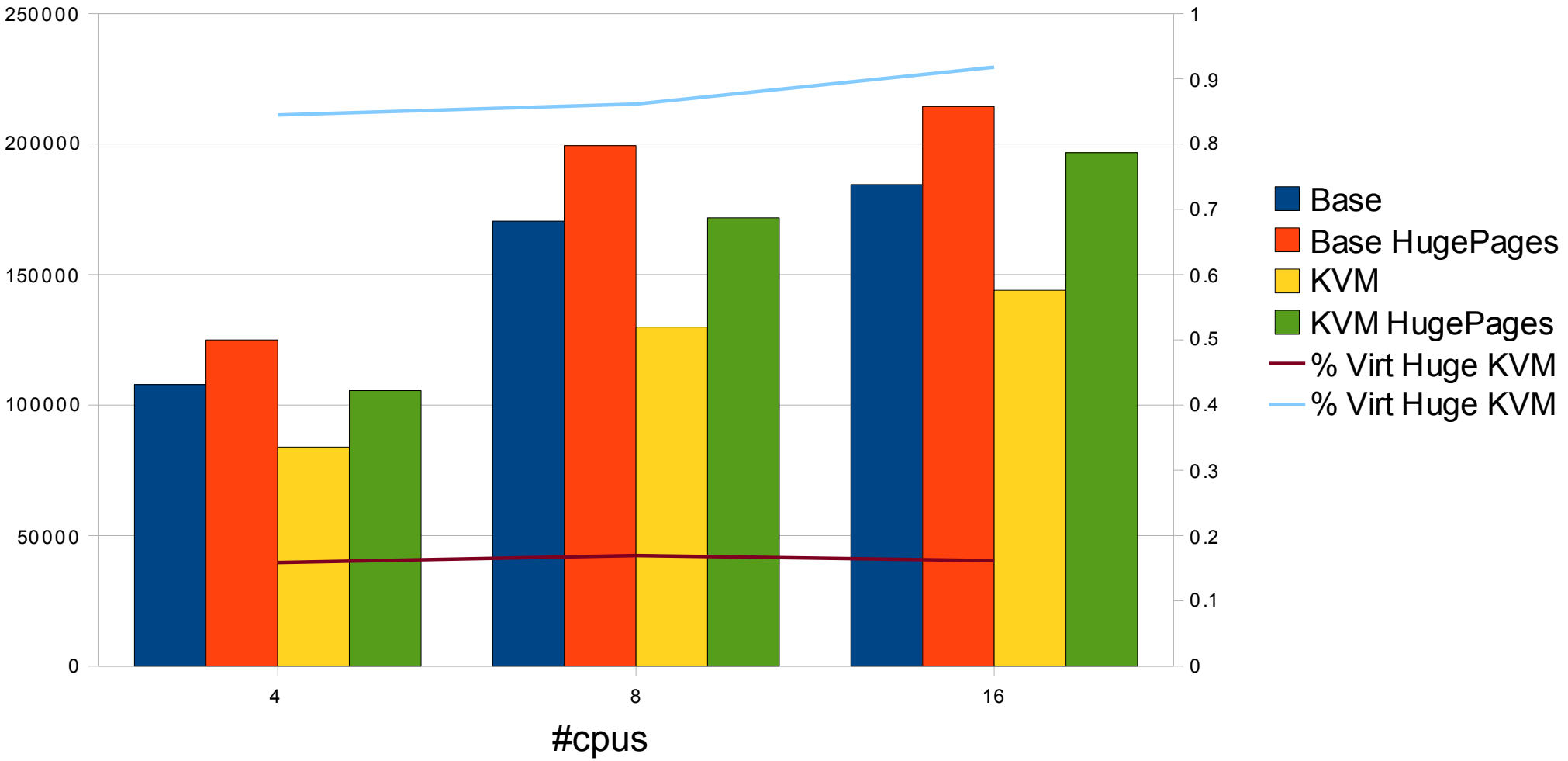
Benefit of hugepage

- NPT/EPT tlb miss cost is higher than with regular shadow paging
- Using 2M pages more valuable with the slower tlb miss of NPT/EPT shadow paging even if tlb miss itself wouldn't be improved
- A 16G process requires 32M of ptes (or 64M with NPT/EPT) without hugepage, much larger than the CPU dcache (pmd/pud/pgd of guest and npt fits)
  - A random memory access triggering a tlb miss in turn triggering two dcache misses in guest pte and npt pte might be ~3 times slower than with hugepages in both guest and host (no ptes and in turn no dcache misses in ptes)

Hugepages

RHEL5.4 KVM Java Performance

Intel Nahalem 2.4 Ghz, 24 Gb mem



redhat

Limit of hugetlbfs

- Hugepages can be used with hugetlbfs
  - They can't be swapped out
  - They must be reserved at boot
  - Hugepages and regular pages can't be mixed in the same vma
  - If reservation is not used and dynamic allocation fails things go bad in kvm
  - Admin mount fs, generally requires privilege
  - On the code side hugetlbfs is growing like a second but inferior Linux VM with its own paths, as people adds more features to hugetlbfs to beahve more like tmpfs

redhat.

Limit of hugetlbfs

- Reservation at boot time may not be big deal with database
  - 1 database
  - 1 machine
  - 1 database cache
  - 1 database cache size set in config file or GUI
  - 1 reservation of hugepages with known size
  - Swapping may still be a problem for SAP though

redhat

Limit of hugetlbfs

- Reservation at boot time huge problem for a virtualization hypervisor like RHEV-H
  - Unknown number of virtual machines
  - Unknown amount of memory used by virtual machines
  - We want to use as many hugepages as available to back guest physical memory (especially with NPT/EPT)
  - Virtual machines are started, shutdown, migrated on demand by user or RHEV-M
  - We don't want to alter behavior of speeded-up virtual machines and we want swap as usual

Transparent Hugepages

- Prefault
  - no tlb benefit
  - Less CPU cache waste (allocation/COW)
  - Less memory waste
  - no cpu speedup if using prefault in 4k chunks
  - Page fault speedup if using pefault with sizes like 8k/16k/32k (but then they destroy more cache) but with >4k prefault COW is almost as slow as 2M prefault...
  - No requirement for glibc to enlarge mappings on 2m boundaries

redhat

Transparent Hugepages

- Decided that prefault is not worth it
  - Complexity of code
  - Large systems will not care about the minor memory waste
  - Future CPU caches will be larger
  - On large systems prefault doesn't optimize the page faults good enough, so it's not a good tradeoff to waste lots of CPU in executing 511 more page faults for each 2M virtual page, just to save CPU cache after the page fault returns
  - Non temporal stores can avoid cache trashing during COW (todo)

redhat.

Transparent Hugepages

- any linux process will receive 2M pages instead of 4k pages if the mmap region is 2M naturally aligned (glibc should map and unmap in 2M aligned chunks)
- When memory pressure triggers the hugepage are splitted and they can be swapped out transparent
- Tries to modify as little code as possible
- Entirely transparent to userland
- Already working with KVM with a small patch incremental with the hugetlbfs support
- Huge boost for page faults too and later the CPU accesses memory faster

redhat.

Transparent Hugepages

- Current implementation only covers anonymous memory
  - KVM guest physical memory is incidentally backed by anonymous memory ;)
  - In the future database may require tmpfs to use transparent hugepages too if they want to swap (database main painful limit of hugetlbs is the lack of swapping and SAP swaps its caches)

Transparent Hugepages

- Embedded systems that have very small cpu caches, and very small memory size may want to disable transparent behavior of hugepages
- Even when the transparent behavior is disabled, hugepages are still allocated fine on madvise(MADV_HUGEPAGE) regions (those apps using MADV_HUGEPAGE like KVM explicitly will not waste memory or CPU caches even on embedded)
- madvise(MADV_HUGEPAGE) still more flexibile than libhugetlbfs (swaps, no privilege, no hacks)
- Kernel daemon can be enabled to collapse hugepages on MADV_HUGEPAGE regions

Transparent Hugepages

- The low impact on the VM is achieved thanks to split_huge_page* that will not fail (returns void) even if the hugepage is under O_DIRECT
- For most VM files this translates in 1 liner change, instead of >100 lines or more of non trivial changes
- A kernel daemon will be enabled to run collapse_huge_page on MADV_HUGEPAGE regions in background
- collapse_huge_page much easier because it can fail for example if the regular page is under O_DIRECT

redhat.

Transparent Hugepages

- O_DIRECT already isn't calling split_huge_page (so KVM cache=off runs on hugepages natively)
- mremap/mprotect in the future can stop calling split_huge_page too by just teaching them to handle hugepages natively (not critical right now)

redhat

Q/A

- You're very welcome!