How to Use KVM's Reverse Mappings to Improve Scalability

November 7th, 2012

NTT Open Source Software Center
Takuya Yoshikawa

NTT

Table of Contents

NTT

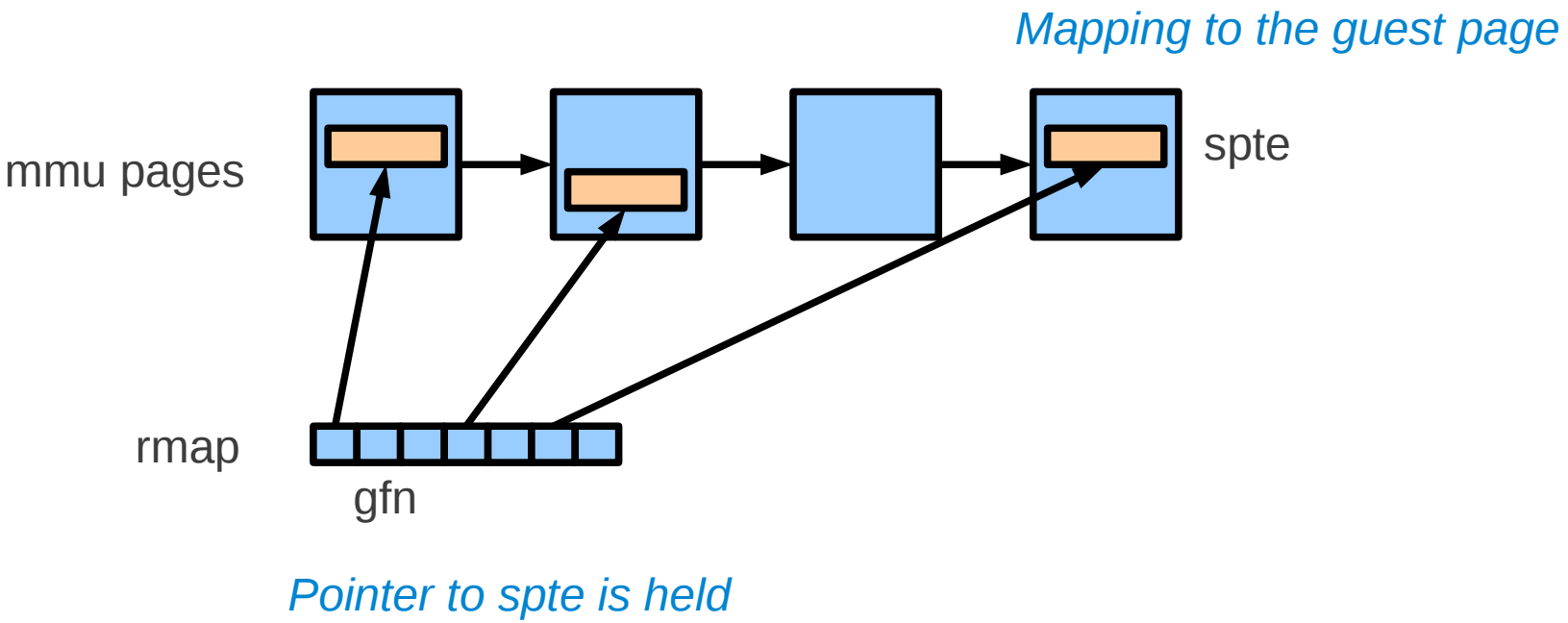
Introduction

- Why rmap
  - My work was mainly done around it last year
    - Not intentional
  - May be able to get more from it
    - Improvement on my previous work
    - More use cases
- What's rmap
  - Tells us which sptes have mappings to a given guest page
    - At least one ulong for each guest page: >= 0.2% overhead
    - Also exists for huge page levels: called rmap_pde before
  - Used for many mmu works, e.g.,
    - Write protecting a guest page
    - Unmapping a guest page

NTT

How rmap can be visualized

- Just for two dimensional paging
  - Assuming EPT or NPT
  - Otherwise lists of sptes need to be drawed



What's achieved

- Fast dirty page logging
  - Originally called SRCU-less dirty logging
  - Good for live migration and VGA emulation
- Efficient THP page invalidation
  - Optimized mmu_notifier's unmapping
  - Good example of rmap handling

Fast dirty page logging

- Problem
  - GET_DIRTY_LOG sometimes took a long time
- Cause
  - Write protection by traversing mmu pages was slow
    - Unnecessarily heavy for relatively small numbers of dirty pages
    - Serious cache pollution
  - dirty_bitmap update by SRCU sometimes got slow
    - Due to the nature of SRCU
- Solution
  - Write protection based on dirty_bitmap and rmap
    - Scans dirty_bitmap to find pages to protect and then uses rmap to find sptes
    - Updates dirty_bitmap by atomic bitops: word-by-word xchg
- Result
  - Stable GET_DIRTY_LOG time proportional to the number of dirty pages

NTT

Get dirty log change in detail

- Before:

  For each mmu page sp *Start from kvm->arch.active_mmu_pages global list*

  If sp has mappings to memslot *Check sp->slot_bitmap*

  For each spte in sp

  Write protect the mapping if needed

  SRCU-update dirty_bitmap

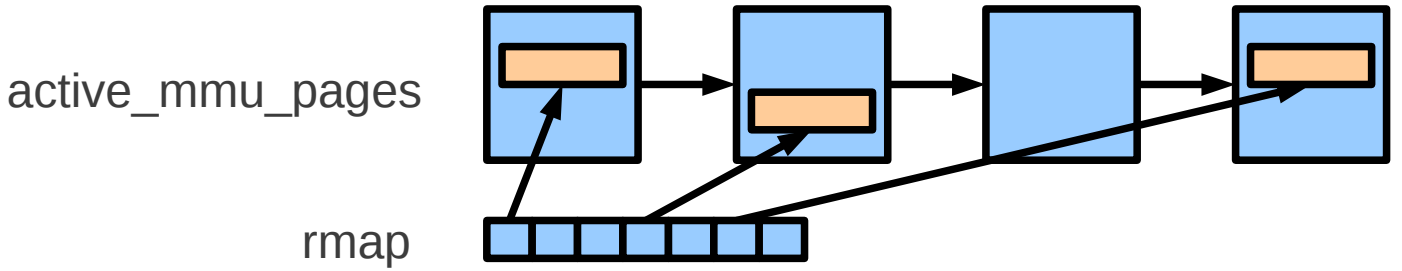
- After:

  For each long size word in dirty_bitmap

  If word is not zero

  Update that word using xchg *SRCU-less word-by-word update*

  Write protect the dirty pages reached from xchged-word and rmap



Efficient THP page invalidation

- Problem
  - Swapping out guest memory backed by THP pages took a long time
- Cause
  - Invalidating a THP page was slow
    - Unmapping every 4K page in it by kvm_unmap_hva()
    - 20~40us
- Solution
  - Introduced kvm_unmap_hva_range()
- Result
  - More than 5 times faster
    - 3~4us
- Related info
  - Eric Northup once reported 30 sec delay when unmapping 128GB of memory
    - Should be mitigated to some extent by this work

NTT

What's changed by kvm_unmap_hva_range()

- Before:

  For each page in [hva_start, hva_end)

  For each memslot *Touches unrelated memslots 512 times*

  Unmap page if in memslot

- After 1: *Skips unrelated memslots first*

  For each memslot that intersects with [hva_start, hva_end)

  For each page in that intersection *Loop over rmap*

  Unmap page

Additional improvement for huge page mappings

- Before(After 1):

  For each page in the intersection

  For each level *For huge level handle the same rmap 512 times*

  Unmap using rmap_level[gfn_level(page)]

- After 2:

  For each level

  For each page_level in the intersection *Loop over gfn_level range*

  Unmap using rmap_level[gfn_level(page_level)]

rmap structure change

- During the work rmap structure was changed a bit
  - rmap_pde was split out from lpage_info
    - Integrated with rmap as rmap[level][gfn]
    - Cleaned up the code a bit for easily accessing a range of rmaps

What we can do next

- Fine grained control of live migration
  - Make initial write protection rmap based
    - Can drop sp->slot_bitmap
    - Fine-grained mmu locking

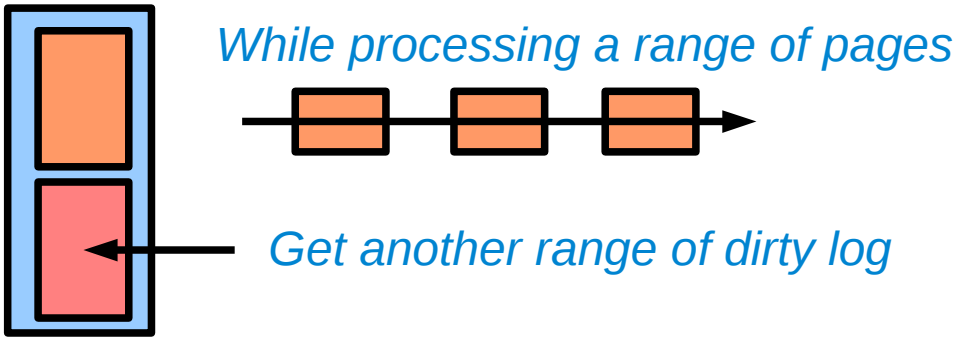
    Good for increasing the number of memslots
  - Make GET_DIRTY_LOG treat a range of addresses
    - Reduce mmu_lock contention naturally
    - Avoid getting too many dirty logs at once
      - QEMU cannot process so many pages at once
      - Dirty log gets stale while processing many pages
- Make use of EPT's A/D bits for dirty logging
  - Latest processors only
  - No write protection

    Use rmap for syncing with dirty_bitmap: see kvm-ppc
    - Guest will be freed from page fault overhead

NTT

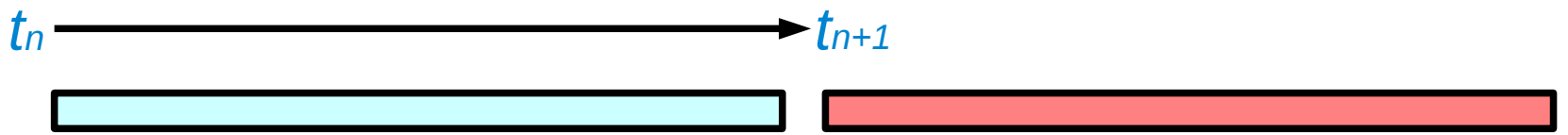
More about fine-grained get dirty log

- Problem of the current live migration scheme
  - Same pages become dirty again while processing many pages
    - Heavy QEMU's dirty_bitmap handling *Being improved a lot by Juan*
    - Cannot proceed while processing that many pages
- What's necessary *To process up-to-date dirty log info for each range*
  - New GET_DIRTY_LOG API
  - Make the current global dirty_bitmap handling treat a range of pages
    - Need a way to guess the remaining dirty pages without global sync
  - Integration with the latest QEMU's migration code
    - Separate migration thread may make it easy to use the new API
    - Multi-threaded processing may also be possible: locking issues will be there

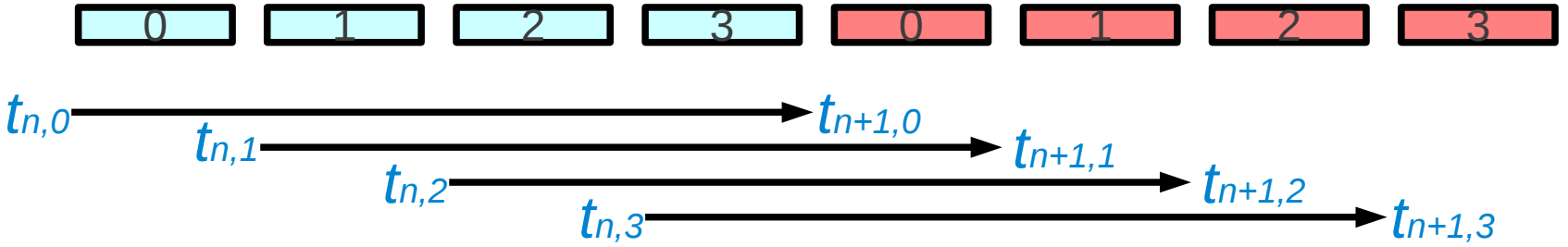


Example: 4 ranges with single threading

1 range only:



4 ranges:



Thank you!