How to Mitigate Latency Problems during KVM/QEMU Live Migration

June 7th, 2012

NTT Open Source Software Center
Takuya Yoshikawa

NTT

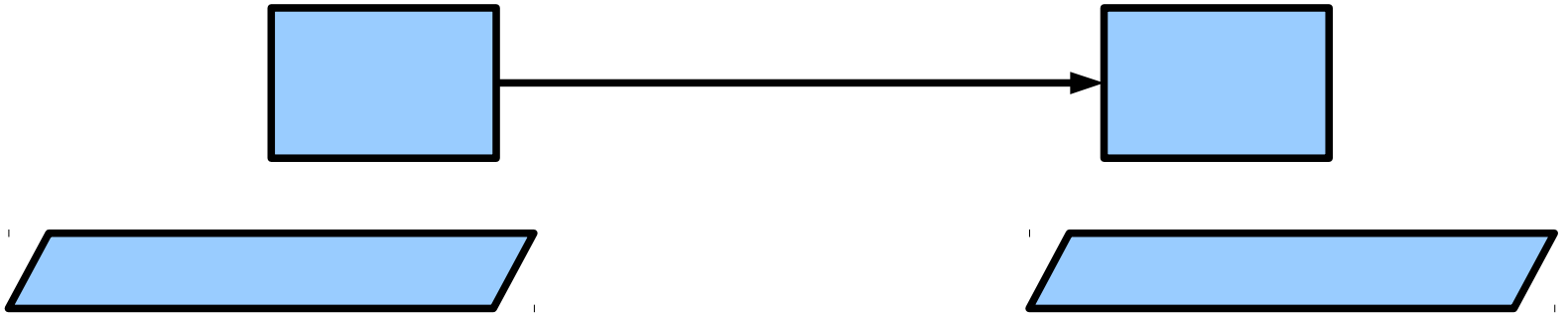
Table of Contents

- Brief introduction of KVM/QEMU live migration
  - How to transfer guest's memory during migration
  - What is needed to track guest's memory changes: GET_DIRTY_LOG
- Explanation of its latency problem
  - Why memory write can take ms during migration
  - What is causing the delay
- How to solve the problem
  - What's already done
  - Further development ideas being discussed

NTT

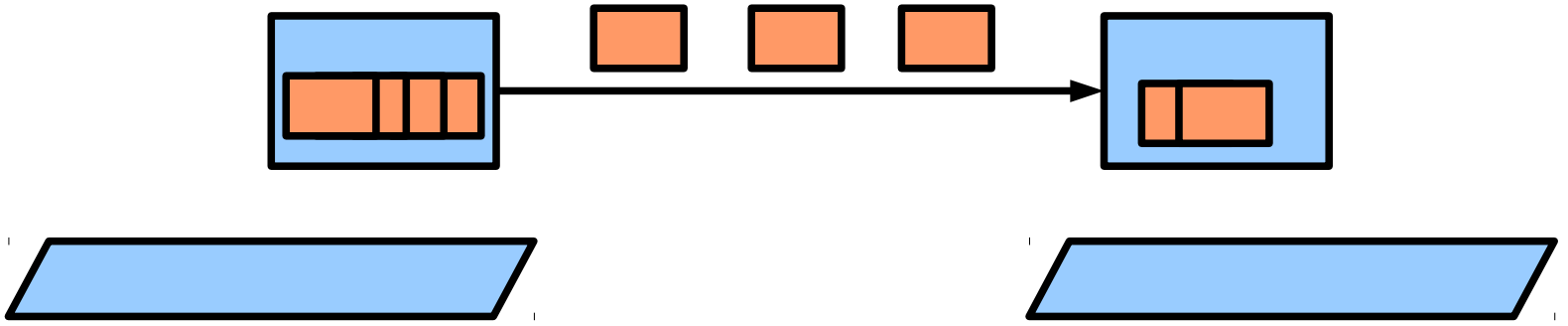
Live Migration

- Transfer guest to other node without stopping it
  - Can be used for load balancing, server maintenance, etc.
  - Need to send guest machine state
    - RAM takes a long time to send



Precopy Live Migration

- QEMU transfers guest's memory pages as follows:
  - Sends dirty pages, updated pages, iteratively
    - KVM has an API called GET_DIRTY_LOG for this
    - Continues this until the number of dirty pages decreases enough
  - Stops the guest and sends the remaining dirty pages
    - Needed to take a complete snapshot
    - Must be reasonably fast to be called seamless



NTT

What's GET_DIRTY_LOG doing

- Copy a snapshot of in-kernel dirty bitmap to userspace
  - Each bit represents one dirty/clean guest page
  - Things need to be done at the same time:
    - Clear the in-kernel dirty bitmap for the next logging: must be atomic to let the guest work concurrently
    - Write protect guest pages, by modifying sptes, to make KVM track successive guest writes: need to take mmu_lock now

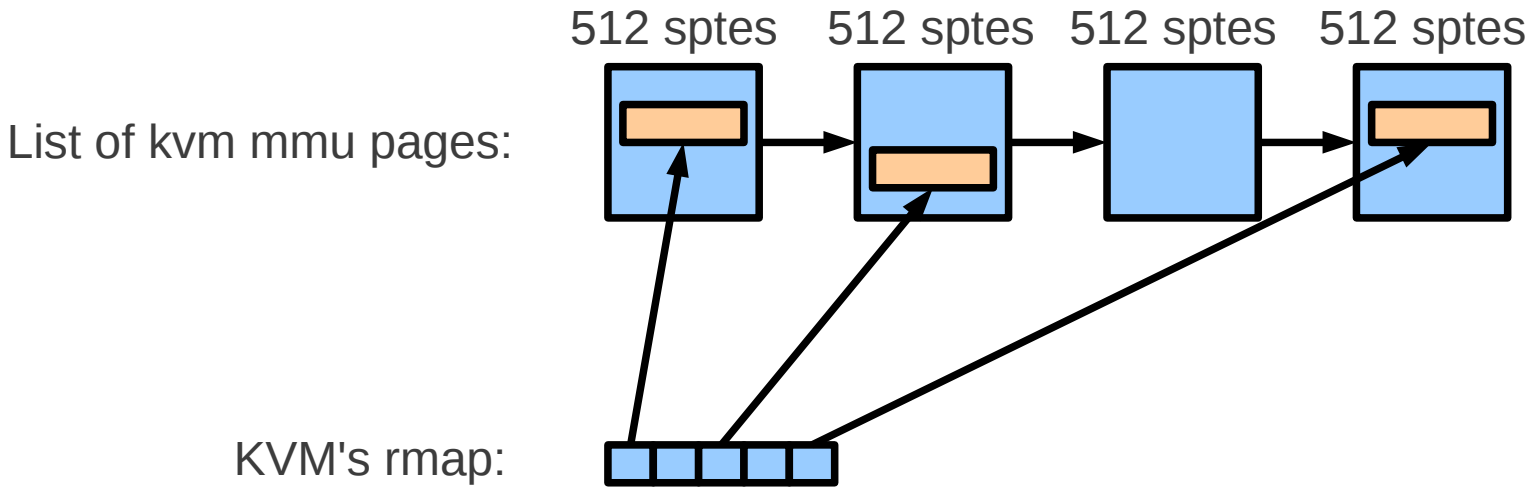
Latency Problem

- GET_DIRTY_LOG can hold mmu_lock for a long time
  - mmu_lock is widely used for KVM mmu work
    - page fault handling caused by dirty logging itself is included
    - VCPU threads can be forced to wait for this lock
  - Held during protecting all pages
    - Now being improved
  - The more pages to protect, the longer the lock hold time
- How long: simple test result
  - ms order of worst case latency in a guest
    - Could be easily seen with 4GB of memory before
    - Much improved now, but still can be seen with more than 10GB

NTT

What's Done: from my work

- Eliminate walking through entire kvm mmu pages
  - Use KVM's rmap to find sptes corresponding to guest's dirty pages
  - Use atomic operation in a loop to update dirty_bitmap instead of switching it at once by updating memslot with SRCU
    - SRCU update sometimes takes a long time
  - Much faster unless dirty pages are not too many
    - Some times faster even for tens of thousands of pages



NTT

Further Development Ideas: from KVM ML

- Release mmu_lock periodically
  - By mixing other work than write protections in between
    - Succeeded in avoiding ms worst case latency with 10GB of memory
- Lazy write protection
  - GET_DIRTY_LOG only protects top level entries: O(1)
  - Other protections are done at the time of page faults
    - Distribute protections to VCPU threads
- Make use of EPT's A/D bits
  - Latest processors only
  - No write protections
    - Guest will be freed from page fault overheads
  - Need to sync this info with dirty bitmap for GET_DIRTY_LOG
    - Some work for every guest page: not just dirty ones

NTT

Thank You!