redhat.

KVM Dirty Ring Interface

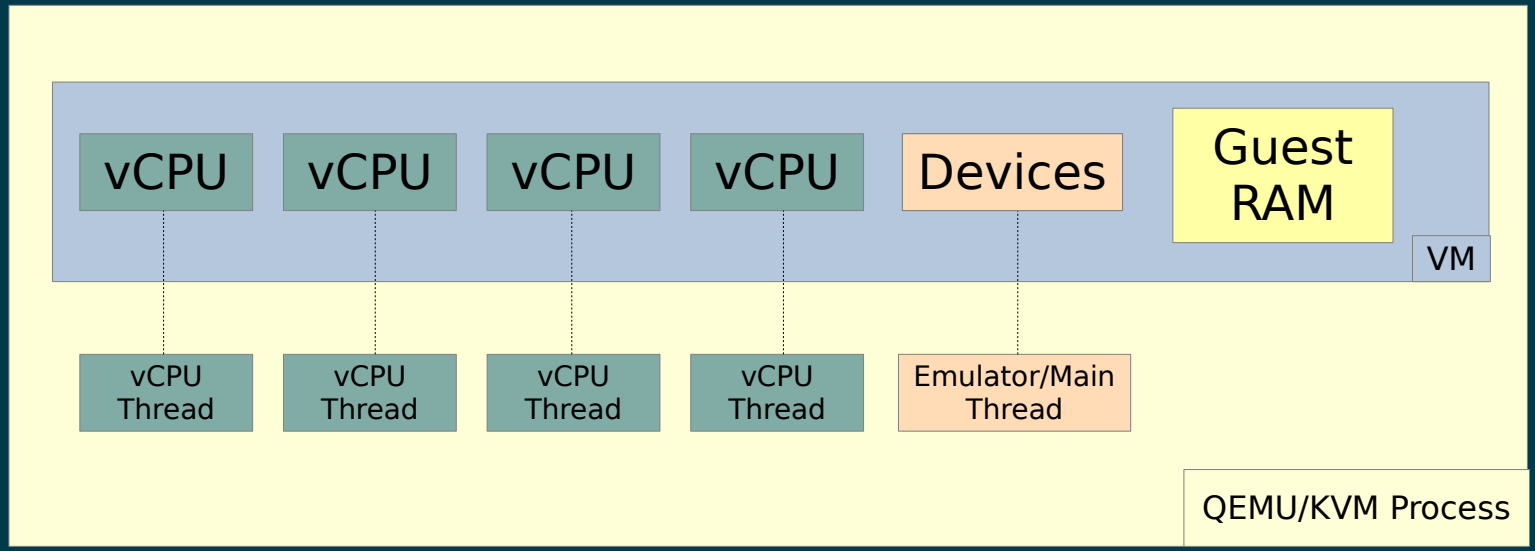
Peter Xu
Senior Software Engineer, Red Hat
October 30th, 2020

Outlines

- Background
- Design & Implementation
- Conclusions & Future work

Background

VM: The Bird View (QEMU/KVM)

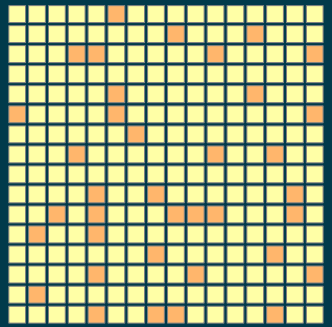


VM Live Migration

- VM migration
  - Move one VM instance from one host to another
  - Data to migrate: RAM, device states, internal states, etc.
  - Stop source VM → Copy data → Start destination VM
- VM live migration
  - Migrate without stopping the VM (or, very low downtime)
  - Data to migrate is changing! Especially, the guest RAM
  - Copy data → Stop source VM → Start destination VM
- VM dirty page tracking needed for tracking guest RAM changes

KVM Dirty Log (KVM_GET_DIRTY_LOG)

- KVM dirty log is the initial solution for KVM dirty tracking
  - Data structure: a huge bitmap
  - Each bit corresponds to one guest page of PAGE_SIZE
  - Userspace periodically collects dirty pages from KVM using ioctl(KVM_GET_DIRTY_LOG)

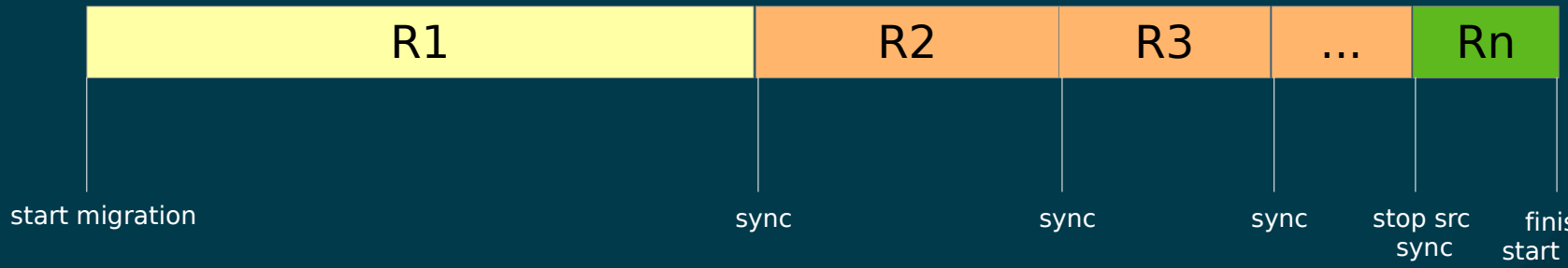


Orange: bit set, page dirty

Yellow: bit clear, page clean

VM Live Migration

- Live Migration work flow
  - The 1st iteration (R1) will migrate all guest RAM pages
  - Sync guest dirty bitmap at the start of each iteration (KVM_GET_DIRTY_LOG)
  - The 2nd (R2), 3rd(R3), … iterations only migrate dirty pages
  - The Nth (Rn) last iteration will migrate with VM stopped



KVM Dirty Log Is Not Ideal...

- ioctl(KVM_GET_DIRTY_LOG) is slow!
  - Step 1: Copy dirty bitmap
  - Step 2: Reset page protections
  - Both steps are linear to guest memory size
- Step 2 scans over the whole bitmap with mmu_lock held
  - Guest may hang death for every page fault
- For big systems, can take (hang) a few seconds or more!

KVM Dirty Log - Variance 1

- New capability: KVM_CAP_MANUAL_DIRTY_LOG_PROTECT2
  - Requires Linux 5.0+ kernels
  - Split the two steps to improve responsiveness
- New steps to collect dirty bitmap
  - ioctl(KVM_GET_DIRTY_LOG)
    - Step 1: Collect dirty bitmap only, pages kept writable
  - ioctl(KVM_CLEAR_DIRTY_LOG) (multiple of)
    - Step 2: Reset page protections
    - Allow to specify a subset of guest pages
- Huge guest VM responsiveness greatly improved!

KVM Dirty Log - Variance 2

- Enabling of KVM dirty logging is slow too!
  - Which applies KVM_MEM_LOG_DIRTY_PAGES to memslots
  - Requires initial reset of page protections of all guest ram
- New bit: KVM_DIRTY_LOG_INITIALLY_SET
  - Requires Linux 5.7+ kernels
  - Initialize the bitmap with all ones
  - Skip page protections in the 1st iteration
- Migration starts tens times faster than before for a guest with 128GB memory!

Dirty Bitmaps: The Good and the Evil

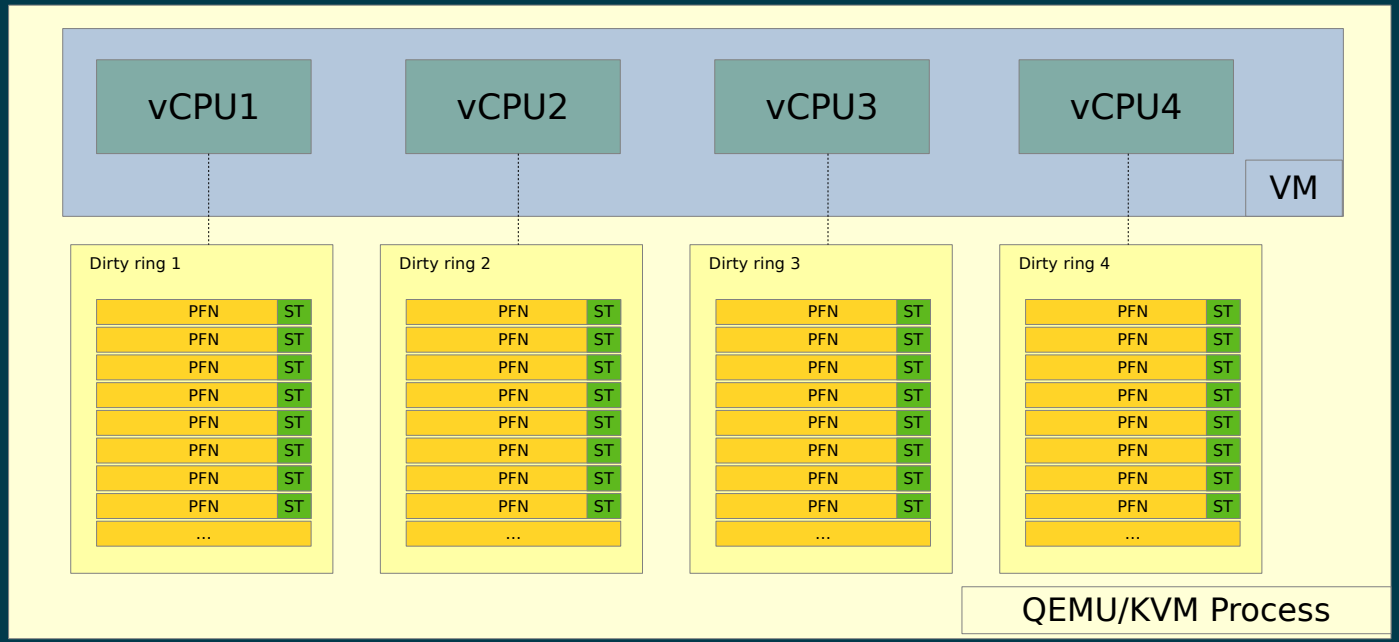
- Dirty bitmap is an ideal structure for many reasons...
  - Data efficient, each bit represents one dirty page
  - Easy serialization using atomic operations
- However VM is getting bigger... so are the dirty bitmaps
  - Collecting dirty bitmap will always be slower
  - Sometimes, we need to sync dirty bitmap between src/dst
  - Hard to scale

Design & Implementation

KVM Dirty Ring

- Original authors
  - 2017: Lei Cao <lei.cao@stratus.com>
  - 2018: Paolo Bonzini <pbonzini@redhat.com>
  - 2019+: me
- Design
  - Use per-vcpu rings to store dirty PFNs (Page Frame Num)
  - Separated collection (step1) and page re-protection (step2)
  - Shared data structure (mmap(), no kernel/user copy)
  - Thread-local buffers

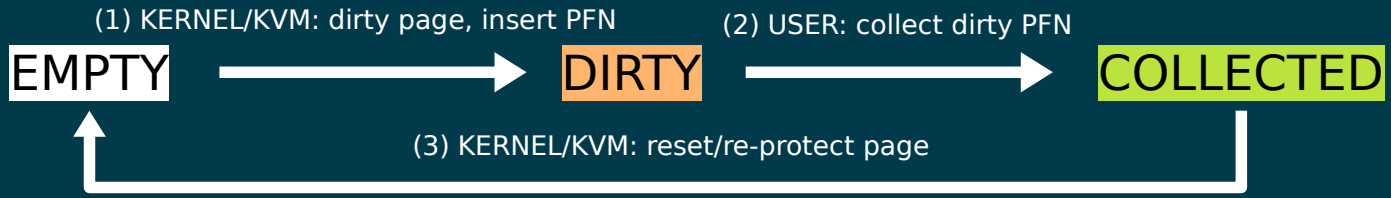
KVM Dirty Ring (cont.)



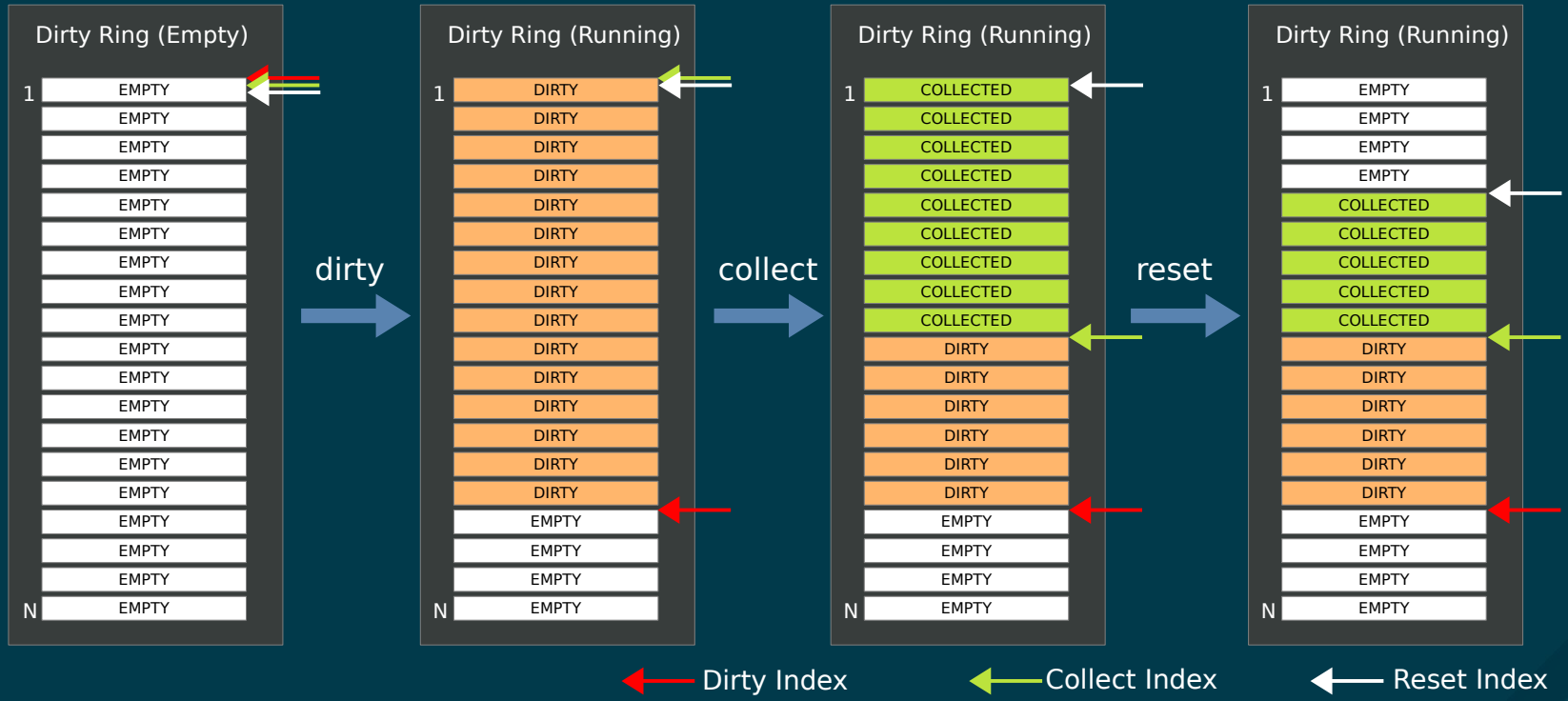
PFN: Page frame number of the dirty page
ST: State of the dirty ring entry

KVM Dirty Ring Entry – State Machine

- State EMPTY
  - The entry is free to use
- State DIRTY
  - The entry contains a valid dirty PFN to be collected
- State COLLECTED:
  - The entry contains a valid collected PFN, to be reset



KVM Dirty Ring – A Closer Look



Dirty Ring vs Dirty Logging

	Dirty Logging	Dirty Ring
Data Structure	Bitmap	Ring / Array
Data Element	Bit	Page Frame Number
Global Data Structure	Yes	No (thread-local)
Data Copy	Yes	No (mmap)
Data Collection	KVM_GET_DIRTY_LOG	Memory Reads (from the rings!)
Page Reset	KVM_CLEAR_DIRTY_LOG	KVM_RESET_DIRTY_RINGS
Collect Granularity	Per-Slot	Per-VCPU, Per-PFN

Dirty Ring Full

- Dirty rings can get full because ring size is limited
- When it happens...
  - Current (write) instruction is interrupted on vcpu
  - The vcpu exits with reason KVM_EXIT_DIRTY_RING_FULL
  - Userspace reaps the dirty ring to free some slots
  - Userspace sents ioctl(KVM_RESET_DIRTY_RINGS)
  - Userspace continue the vcpu with KVM_RUN
  - Re-run the interrupted (write) instruction
  - Vcpu continues execution

“Side Effect” - Auto Converge v2.0 (?)

- Unlike dirty logging, dirty ring tracking can block vcpus
  - Which... provides a natural way to throttle the source!
- With dirty ring, auto-converge may achieve...
  - Finer granularity (on “what to throttle”):
    - Global throttle → per-vcpu throttle (per-vcpu rings)
  - Better responsiveness (on “trap points”):
    - Every dirty sync → every ram write (triggers ring full)
- A better auto-converge is possible!

Conclusion

- Benefits
  - Reduced/Configurable memory footprint
  - Synchronization is cheaper, can run in the background
  - Much more friendly to VMs with huge memories
- Possible scenarios
  - COLO
  - Low/Medium dirty rate huge VM migrations
  - Enhanced auto-converge (maybe?)
  - More?

Future work?

- Have it merged... With more real world runs
- Non-x86 support? Per-vcpu ring reset? Etc.
- For QEMU...
  - Support the new interface in QEMU kvm-all.c ✓
  - Remove ramblock/migration dirty bitmap ❓
  - Let precopy to read dirty pages in queues❓❓❓❓
    - Precopy will look more like postcopy!
    - Auto-converge will be on by default since ring gets full (next slide...)

An Imaginary Precopy World...
(Where Dirty Bitmap Does not Exist)

- Migration thread (or using one migration thread per vcpu?):
  - One pointer CURRENT points to physical page index 0 (emulate 1st round bulk mode)
  - Multiple per-vcpu QUEUEs that vcpu will push dirty pages in
  - If all the QUEUEs are...
    - Empty: Migrate the page on CURRENT, then increase CURRENT. If CURRENT points to the last page, stop vm and finish migrate (or switch to postcopy)
    - Some not empty: Choose one QUEUE and migrate the first page on this QUEUE, then kick the vcpu if the vcpu is blocked due to ring full. We should pick QUEUE in a round-robin fashion so that we'll kick vcpus in order.
- VCPU threads:
  - Collect dirty rings, if...
    - Page index > CURRENT, drop it since we haven't migrated the page.
    - Page index <= CURRENT, queue into QUEUE. If there're more than N pages in QUEUE that this vcpu owns, stop vcpu until the migration thread kicks it.

References

- Latest version:
  - https://lore.kernel.org/kvm/20201007205342.295402-1-peterx@redhat.com/
- Repos for testing
  - Kernel: https://github.com/xzpeter/linux/tree/kvm-dirty-ring
  - QEMU: https://github.com/xzpeter/qemu/tree/kvm-dirty-ring

THANK YOU!!!

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos