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 1\textsuperscript{st} iteration (R1) will migrate all guest RAM pages
  - Sync guest dirty bitmap at the start of each iteration (KVM\_GET\_DIRTY\_LOG)
  - The 2\textsuperscript{nd} (R2), 3\textsuperscript{rd} (R3), \ldots iterations only migrate dirty pages
  - The N\textsuperscript{th} (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(KERN_GET_DIRTY_LOG)
    • **Step 1**: Collect dirty bitmap only, pages kept writable
  • ioctl(KERN_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

(1) KERNEL/KVM: dirty page, insert PFN
(2) USER: collect dirty PFN
(3) KERNEL/KVM: reset/re-protect page
KVM Dirty Ring – A Closer Look

Dirty Ring (Empty)

Dirty Ring (Running)

Dirty Ring (Running)

Dirty Ring (Running)

Dirty Ring (Running)

Dirty Index

Collect Index

Reset Index
## Dirty Ring vs Dirty Logging

<table>
<thead>
<tr>
<th></th>
<th>Dirty Logging</th>
<th>Dirty Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Structure</strong></td>
<td>Bitmap</td>
<td>Ring / Array</td>
</tr>
<tr>
<td><strong>Data Element</strong></td>
<td>Bit</td>
<td>Page Frame Number</td>
</tr>
<tr>
<td><strong>Global Data Structure</strong></td>
<td>Yes</td>
<td>No (thread-local)</td>
</tr>
<tr>
<td><strong>Data Copy</strong></td>
<td>Yes</td>
<td>No (mmap)</td>
</tr>
<tr>
<td><strong>Data Collection</strong></td>
<td>KVM_GET_DIRTY_LOG</td>
<td>Memory Reads (from the rings!)</td>
</tr>
<tr>
<td><strong>Page Reset</strong></td>
<td>KVM_CLEAR_DIRTY_LOG</td>
<td>KVM_RESET_DIRTY_RINGS</td>
</tr>
<tr>
<td><strong>Collect Granularity</strong></td>
<td>Per-Slot</td>
<td>Per-VCPU, Per-PFN</td>
</tr>
</tbody>
</table>
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 \textbf{not} Exist)

- Migration thread (or using one migration thread per vcpu?):
  - One pointer \texttt{CURRENT} points to physical page index 0 (emulate 1\textsuperscript{st} round bulk mode)
  - Multiple per-vcpu \texttt{QUEUE}s that vcpu will push dirty pages in
  - If all the \texttt{QUEUE}s are...
    - Empty: Migrate the page on \texttt{CURRENT}, then increase \texttt{CURRENT}. If \texttt{CURRENT} points to the last page, stop vm and finish migrate (or switch to postcopy)
    - Some not empty: Choose one \texttt{QUEUE} and migrate the first page on this \texttt{QUEUE}, then kick the vcpu if the vcpu is blocked due to ring full. We should pick \texttt{QUEUE} in a round-robin fashion so that we’ll kick vcpus in order.

- VCPU threads:
  - Collect dirty rings, if...
    - Page index > \texttt{CURRENT}, drop it since we haven’t migrated the page.
    - Page index \(\leq\) \texttt{CURRENT}, queue into \texttt{QUEUE}. If there’re more than N pages in \texttt{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
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHat