Virtuozzo

Towards asynchronous revert in QEMU

Denis V. Lunev

den@openvz.org

Contents

- Background snapshot
- Asynchronous revert to sna
- Performance results
- Future

Background snapshot - asynchronous beast

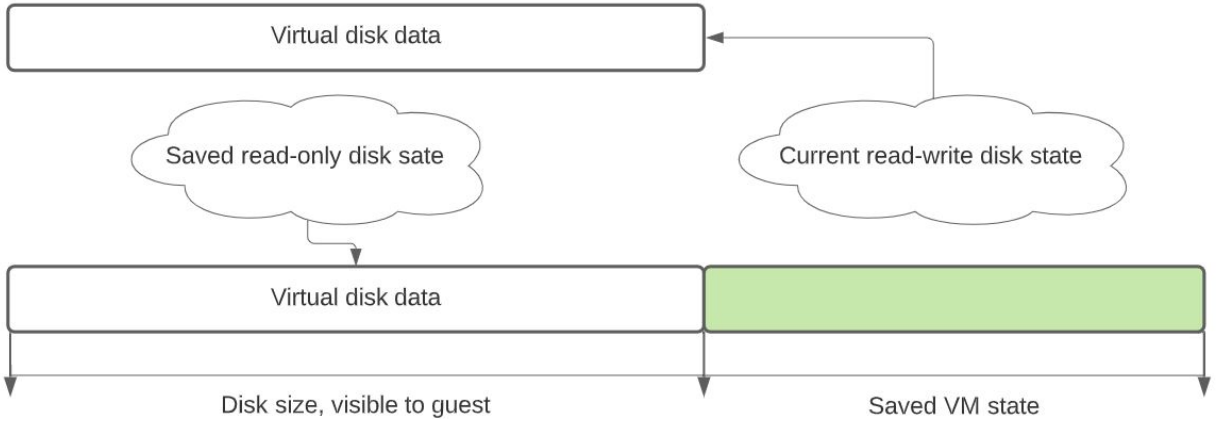
Something very straightforward

Create snapshot

- Stop VM CPUs
- Commit all pending IO
- Save CPU/devices state
- Save RAM
- Make virtual disk snapshot
- Start VM CPUs

VM state storage: by default inside QCOW2

- There is no infrastructure in QEMU to maintain writes into two different disk states

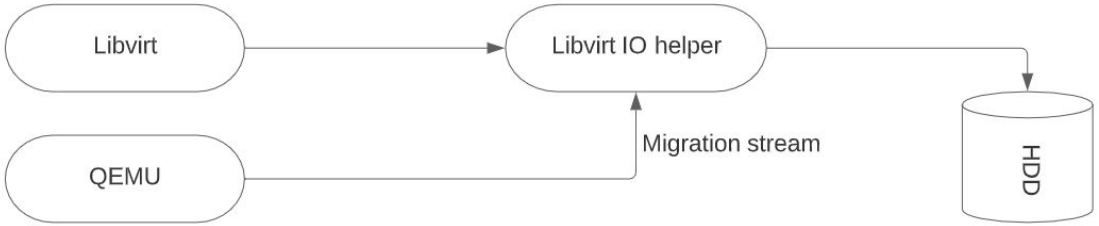


Create background snapshot

- Stop VM CPUs
- Commit all pending IO
- Save CPU/devices state
- Protect VM memory for write
- Make disk snapshot
- Start VM CPUs
- Store VM memory in background
- Save memory pages written by guest out of order

VM state storage

- Writing into 2 different states of the same disk in QEMU is not allowed
- Generic migration approach
  - Send migration stream via socket outside
  - Save into external file
- Spoiler: would be useful on restore



Implementation: interface

- Special migration mode:

  { 'execute': 'migrate-set-capabilities',
  'arguments': { 'capabilities': [{'capability':
  'background-snapshot', 'state': 'true'}] } }

- Start migration normally:

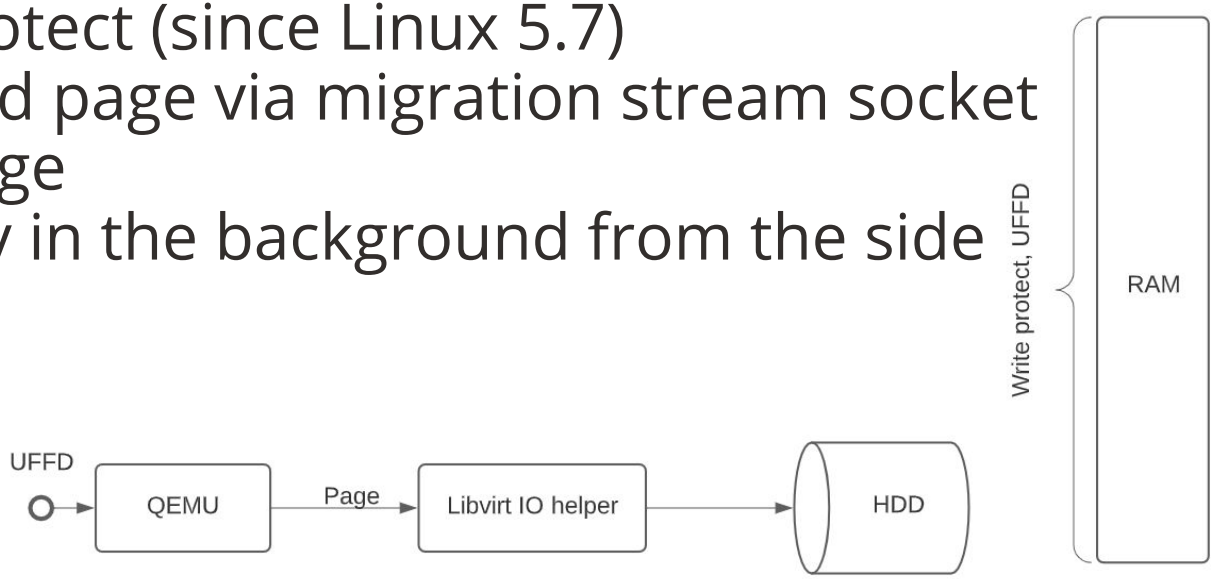
  virsh save VMname state-filename

- Easy testing

  virsh restore state-filename

Dirty tracking

- Protect VM memory for WRITE using user-fault-fd with write-protect (since Linux 5.7)
- Send modified page via migration stream socket
- Unprotect page
- Scan memory in the background from the side thread



Asynchronous revert to snapshot

A bit more complicated stuff...

Revert to snapshot: standard

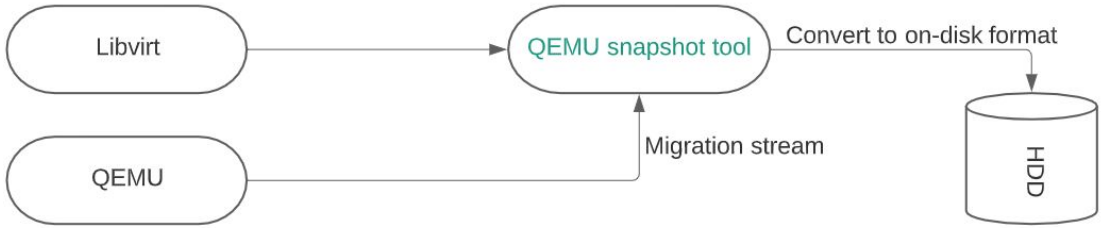
- Start QEMU
- Load the whole migration stream
- Start the guest
- The time of stream loading is increased with VM size. Knowingly weird!

Asynchronous revert to snapshot

- Start QEMU
- Load devices state (small, fixed size)
- Start the guest
- Load VM memory in the background
- Load memory pages accessed the by guest out of order
- No page addressing in the migration stream!

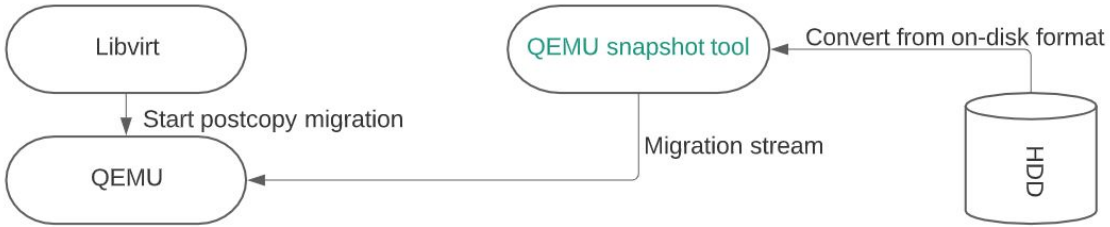
VM state storage

- Any public format change is expensive
- Public protocols are very costly to change too
- The idea:
  - replace libvirt IO helper with a new tool
  - convert migration stream at save
  - read new data in the same tool on revert



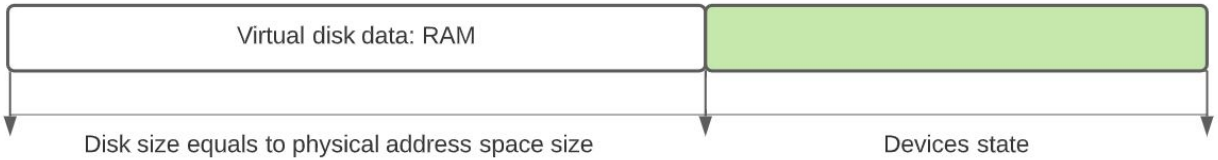
QEMU snapshot tool

- Start as a migration source in pre-copy mode
- Transfer devices state
- Transfer some memory (optional)
- Switch migration into post-copy mode
  - start the guest
- No need for separate control channel



Storing format: QCOW2!

- QCOW2 stores data addressed from 0 to something called “virtual size”
- Store RAM as data
- Store devices state as usual



QEMU snapshot tool (continued)

- Special migration mode:

  '{"execute": "migrate-set-capabilities", "arguments":
  {"capabilities": [{"capability": "postcopy-ram", "state": true}]}}'

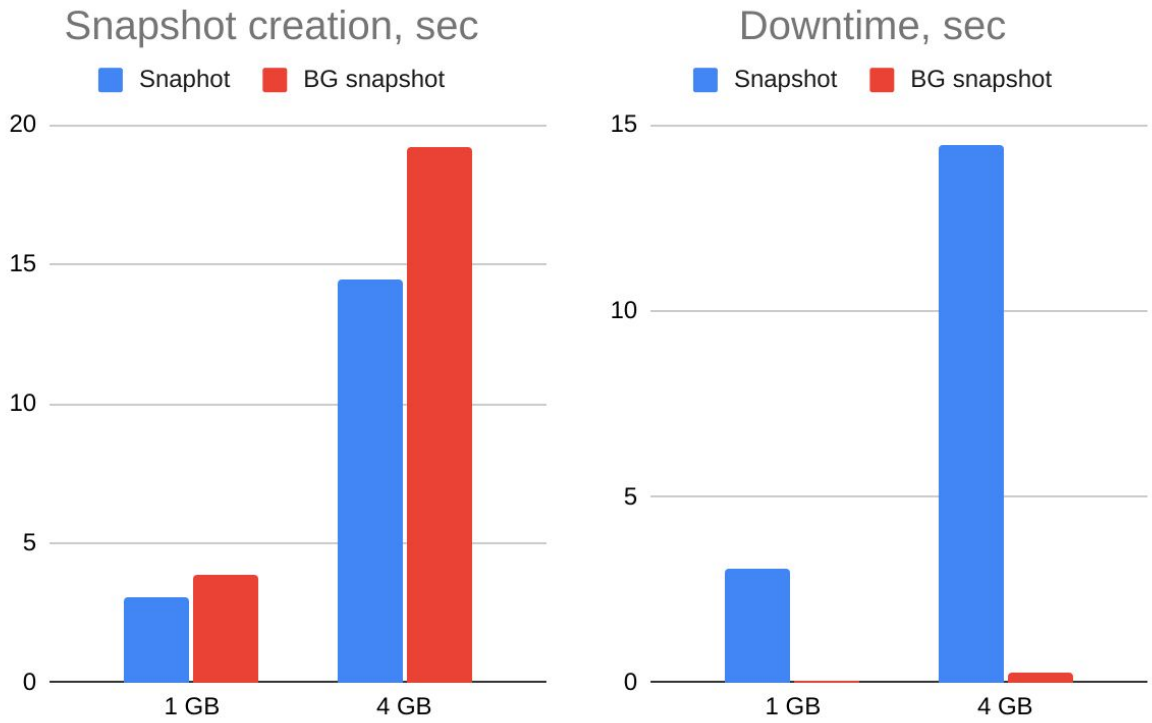
- Start incoming migration:

  '{"execute":"migrate-incoming", "arguments":
  { "uri":"exec:qemu-snapshot --revert --postcopy=0 state.qcow2"}}'

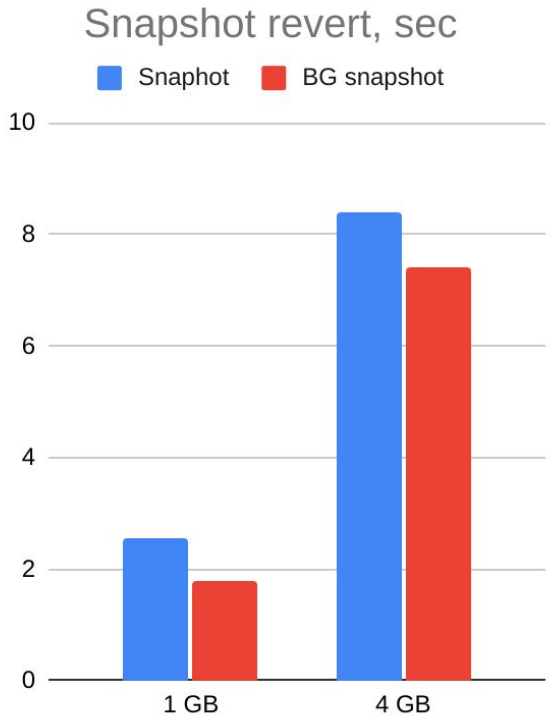
Performance results

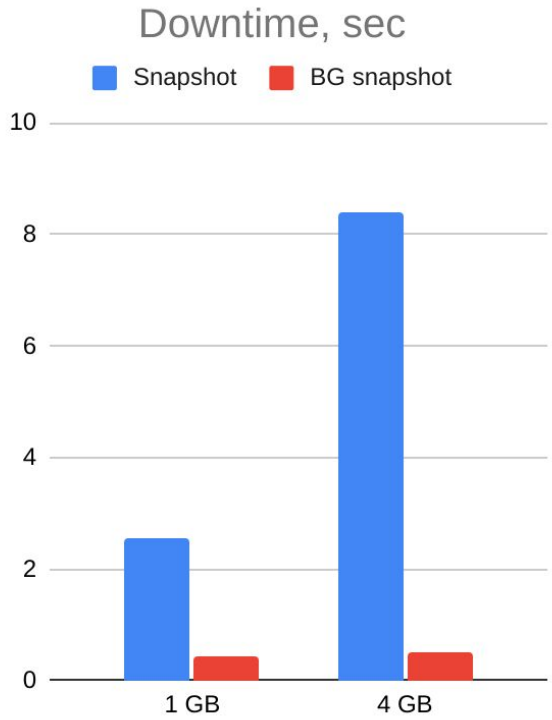
Something we are fighting for...

Performance: snapshot creation



Performance: revert to snapshot





Future work

Cool and shiny tomorrow

Current state

- UFFD merged into Linux 5.7
- Background snapshot is merged into QEMU 6.0
- qemu-snapshot-tool sent as RFC at May 12 2021

  https://lists.gnu.org/archive/html/qemu-devel/2021-05/msg03587.html

Performance bottlenecks

- Single threaded UFFD
  - create several UFFDs by address ranges
  - true multithreaded UFFD
- No pre-populated memory in guest
  - track accessed memory on snapshot

Questions?

www.virtuozzo.com

@VirtuozzoInc

www.linkedin.com/company/virtuozzo