Virtuozzo

QEMU snapshots are slow. Really?

Denis V. Lunev
den@openvz.org

Contents

- Internal snapshots
- Performance results
- Future
  - background snapshot
  - asynchronous revert to snapshot

Internal snapshots - ‘savevm’/’loadvm’

Important details

Motivation (a long time ago, in a galaxy far-far away...)

- Downtime for 8 GB VM: **300** seconds
- Storage capacity: 150 Mb/sec
- Really achieved speed: 27 Mb/sec

Create snapshot

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

QEMU IO pattern (not so bad?)

virsh qemu-monitor-command vm --hmp trace-event qcow2_writev_start_req on

- writes are sequential
- write size is not so bad

qcow2_writev_start_req co 0x55ee7792fa30 offset **0x109a2e7c3c** bytes **131328**
qcow2_writev_start_req co 0x55ee78176f40 offset **0x109a307d3c** bytes **131337**
qcow2_writev_start_req co 0x55ee77b1b710 offset 0x109a327e45 bytes 131328
qcow2_writev_start_req co 0x55ee7858e010 offset 0x109a347f45 bytes 53360

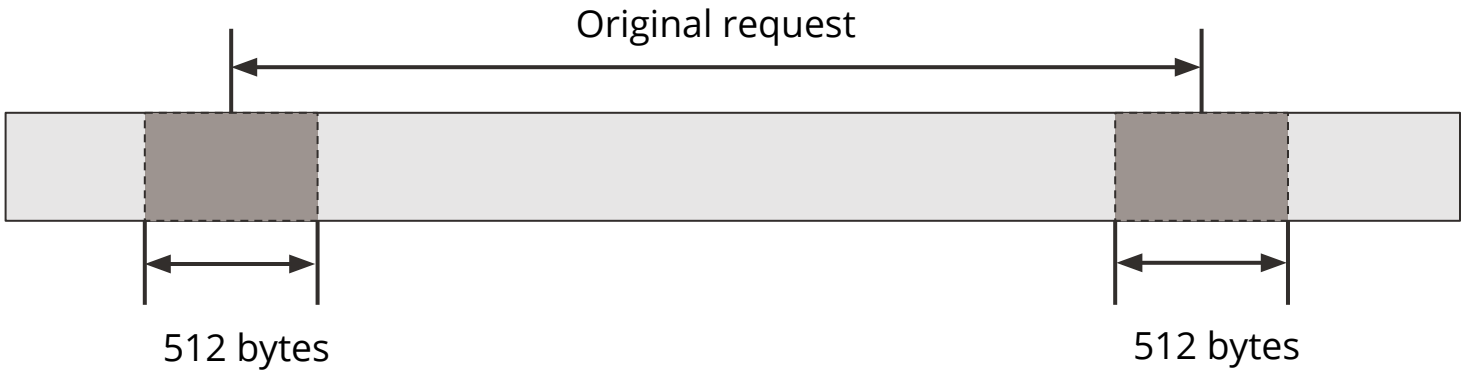
Linux real IO pattern

blktrace -d /dev/sda -o - | blkparse -i -

8,0	4	40	0.120622778	677708	D	R	**347181323** + 1 [qemu-kvm]
8,0	3	98	0.121070367	0	C	R	347181323 + 1 [0]
8,0	3	106	0.121181060	677708	D	R	**347181580** + 1 [qemu-kvm]
8,0	3	107	0.121230086	0	C	R	347181580 + 1 [0]
8,0	4	48	0.121512160	678123	D	WS	**347181323** + 258 [qemu-kvm]
8,0	3	108	0.121963520	0	C	WS	347181323 + 258 [0]
8,0	4	56	0.122192028	677708	D	R	**347181580** + 1 [qemu-kvm]
8,0	3	109	0.122592687	0	C	R	347181580 + 1 [0]
8,0	3	117	0.122700027	677708	D	R	**347181837** + 1 [qemu-kvm]
8,0	3	118	0.122980774	0	C	R	347181837 + 1 [0]
8,0	4	64	0.123417678	678123	D	WS	**347181580** + 258 [qemu-kvm]
8,0	3	119	0.123871902	0	C	WS	347181580 + 258 [0]

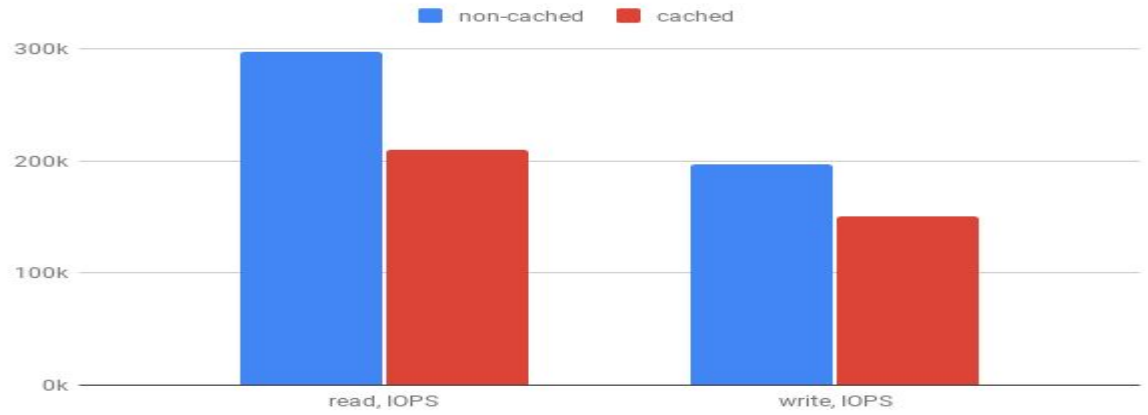
IO pattern analysis

- Synchronous non-aligned operations
- Read-modify-write IO pattern
- Non-cached IO is a problem?



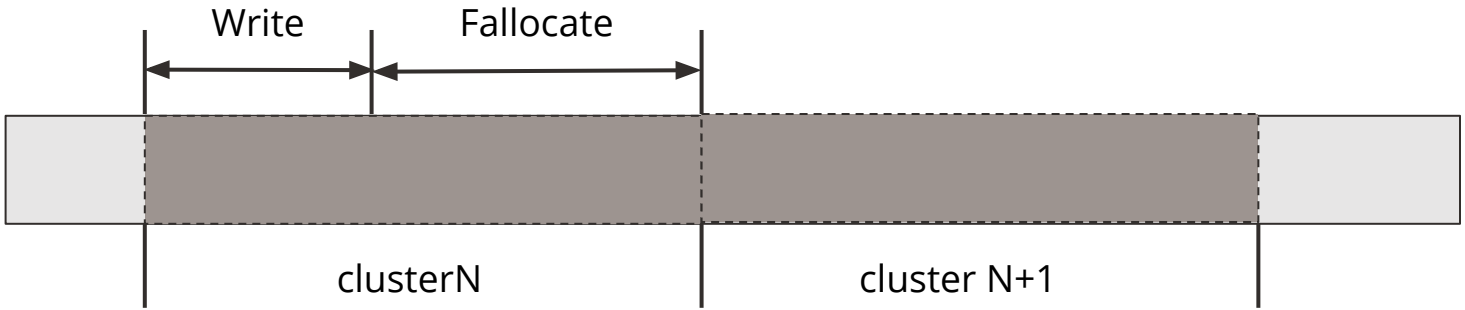
QEMU and non-cached IO

- ‘Native’ AIO worker, no additional threads
- Predictable latency
- Good behavior under memory overcommit



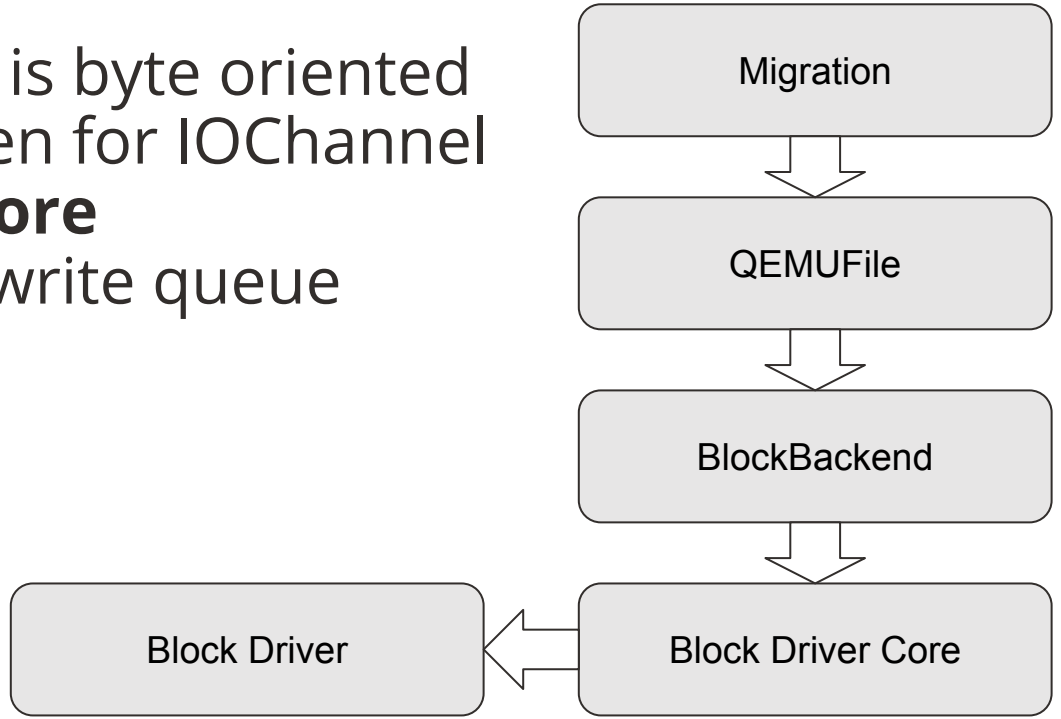
Faster snapshot

- Data is written sequentially allocating new clusters
- Longer but limited asynchronous queue
- Cluster aligned IO

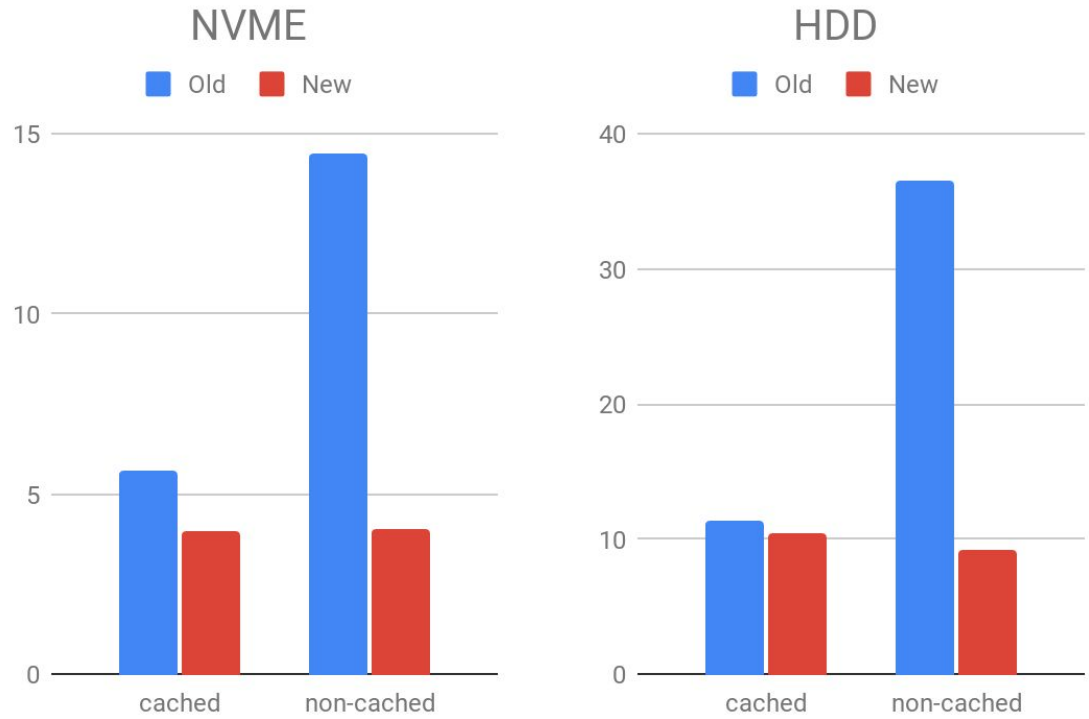


QEMU snapshot architecture

- Migration code is byte oriented
- Originally written for IOChannel
- **Block Driver Core**
- Asynchronous write queue



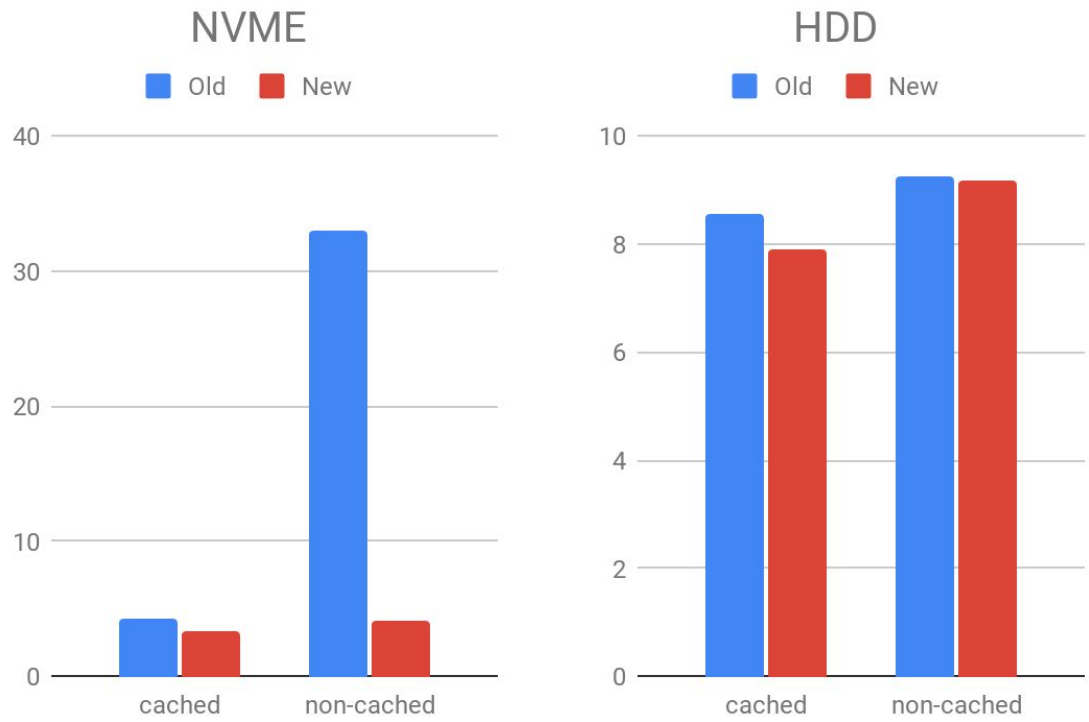
Snapshot creation time (lower is better)



Faster revert to snapshot

- Rotational disks itself are good with read-ahead
- Read sequentially with big enough data chunk
- Prefetch some additional chunks in background
- Issue read request
  - In read completion (if there is spare space in buffer)
  - Once some buffer space is freed
- No request queue!

Revert to snapshot



Conclusions

- Good queue for write is mandatory
- Cluster aligned IO operation is a must
- Rotational disks have very good read-ahead in HW
- Sequential IO for rotational media is more important than longer queue
- All these tricks are a bandaid! The guest is still stopped for the whole operation

Background snapshot

Brilliant future to come tomorrow or after tomorrow. May be later....

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

Implementation state

- Based on write-protect with userfaultfd
- Support is included into Linux kernel 5.7
- QEMU code as unit test for kernel
- QCOW2 driver disallow writing into to snapshots at the same time
- Migration stream is saved outside at the moment

Fast snapshot revert (postcopy revert)

- Load CPU/devices state
- Start VM
- Load each accessed guest page on page fault
- Fill memory from file in background

QCOW2 as storage

- Parse memory section of migration stream
- Drop it from migration stream
- Save content as data into **separate** QCOW2 image
- Save migration stream as usual

Questions?

www.virtuozzo.com

@VirtuozzoInc

www.linkedin.com/company/virtuozzo