Userspace NVMe Driver in QEMU

Fam Zheng
Senior Software Engineer

KVM Form 2017, Prague
About NVMe

- Non-Volatile Memory Express
- A scalable host interface specification like SCSI and virtio
  - Up to 64k I/O queues, 64k commands per queue
  - Efficient command issuing and completion handling
- Extensible command sets
- Attached over PCIe, M.2 and fabrics (FC, RDMA)
Why?
Overhead
SATA HDD Baremetal Performance

<table>
<thead>
<tr>
<th>SATA (HDD)</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>randread-1-req</td>
<td>~100</td>
</tr>
<tr>
<td>randread-4-req</td>
<td>~150</td>
</tr>
<tr>
<td>randwrite-1-req</td>
<td>~250</td>
</tr>
<tr>
<td>randwrite-4-req</td>
<td>~250</td>
</tr>
<tr>
<td>randw-1-req</td>
<td>~150</td>
</tr>
<tr>
<td>randw-4-req</td>
<td>~200</td>
</tr>
</tbody>
</table>
SATA HDD and SSD Performance Comparison

IOPS

SATA (HDD)  SATA (SSD)

randread-1-reg  randwrite-1-reg  randrw-1-reg  randread-4-reg  randwrite-4-reg  randrw-4-reg
Faster device → more visible overhead!
FusionIO is an old model so may not represent its state-of-art.

SATA (SSD) test is done on a different host so the relativity doesn’t matter much.
FusionIO is an old model so may not represent its state-of-art

SATA (SSD) test is done on a different host so the relativity doesn’t matter much
Latency Reducing

- KVM optimizations
  - `kvm_halt_poll` by Paolo Bonzini
  - QEMU AioContext polling by Stefan Hajnoczi

- Kernel optimizations
  - `/sys/block/nvme0n1/queue/io_poll` by Jens Axboe
    (improves aio=threads case)

- Device assignment
  - QEMU: `-device vfio-pci`

- Userspace device driver based on VFIO
  - DPDK/SPDK: `vhost-user-blk`
  - QEMU: VFIO driver in this talk
Architecture
From QEMU PoV

Guest kernel
- VirtIO driver
- VirtIO device
- BlockBackend
- Block layer
- QCOW2
- VFIO NVMe driver

QEMU
- VFS
- vfio-pci.ko
- nvme.ko

Host kernel
Implementation

- `$QEMU_SRC/util/vfio-helpers.c`
  - A generic helper library for userspace drivers
  - Manages per device IO virtual address (IOVA) space
  - Optimized for I/O operations:
    - Pre-allocate IOVA for all guest ram
    - Efficient oneshot IOVA allocation for bounce buffer I/O

- `$QEMU_SRC/block/nvme.c`
  - Registers a new BlockDriver (nvme://)
  - Handles NVMe logic
  - Integrates with AioContext polling
  - Prepared for QEMU multiqueue block layer
Characteristics

- Commands: READ, WRITE (with FUA), FLUSH
- IOV based (zero-copy)
- One IO queue pair for now
- More efficient for guest I/O
- Less efficient for bounce buffered I/O and utility
  - More on this later...
- Device is exclusively used by one VM similar to device assignment
I/O Request Lifecycle

virtio-*.ko  Queue virtio request (GPA/vIOVA)
↓
virtio  Map I/O address to host address (HVA)
↓
virtio-blk  Parse request, call blk_aio_preadv/pwritev
↓
block layer  Call NVMe driver
↓
NVMe driver  Send request to device
NVMe Driver Operations

(1) Check that the addresses and lengths are aligned
   If not, allocate an aligned bounce buffer to do next steps
(2) Map host addresses to IOVAs
(3) Prepare an NVMe Request structure using IOVAs and put it on the NVMe I/O queue
(4) Kick device by writing to doorbell
(5) Poll for completions of earlier requests
(6) Yield until irq eventfd is readable
Address Translations

Guest app buffer

Guest physical addr

Host virtual address (no vIOMMU)

IOVA

page list

submission queue

page list is pre-allocated!

IOMMU

NVMe
IOVA Mapping

```c
struct vfio_iommu_type1_dma_map dma_map = {
    .argsz = sizeof(dma_map),
    .flags = VFIO_DMA_MAP_FLAG_READ |
            VFIO_DMA_MAP_FLAG_WRITE,
    .vaddr = (uintptr_t)host,
    .size = size,
    .iova = iova,
};

ioctl(vfio_fd, VFIO_IOMMU_MAP_DMA, &dma_map);
```
Address Translations

Guest app buffer

Guest physical addr

Host virtual address (no vIOMMU)

IOVA addr space

PRP list

I/O queue
How About Host Buffers?

- The (slow) default:
  VFIO_IOMMU_MAP_DMA each new buffer to a new address as it comes

- Remedy for hot buffers:

  ```c
  void bdrv_register_buf(BlockDriverState *bs, void *host, size_t size);
  void bdrv_unregister_buf(BlockDriverState *bs, void *host);
  ```

  Map/unmap a buffer to IO virtual address in the same way as guest ram.
The IOVA Allocator

- Keep record of mapped buffers for later use, if advisable
- Distinguish throwaway / fixed mappings with a parameter

```c
int qemu_vfio_dma_map(QEMUVFIOState *s, void *host, size_t size,
  bool temporary, uint64_t *iova)
```

- Use a pair of self-incrementing counters to track available IOVAs
- When free IOVAs run out, discard all temporary mappings and reset counter (caller makes sure all old mappings are useless)
Usage

- Until patches are merged to mainline:
  git clone https://github.com/qemu/famz --branch nvme
- configure && make, as usual
- Bind device to vfio-pci, see also:
  https://www.kernel.org/doc/Documentation/vfio.txt
- ./x86_64-softmmu/qemu-system-x86_64
  -enable-kvm
  ...
  -drive file=nvme://0000:44:00.0/1,if=none,id=drive0
  -device virtio-blk,drive=drive0,id=virtio0

- Syntax:
  nvme://<domain:bus:dev.func>/<namespace>
  Or, use structured option
  -drive
  driver=nvme,device=<domain:bus:dev.func>,namespace=<N>,if=none...
IOPS Improvement over Linux-aio

<table>
<thead>
<tr>
<th>(IOPS)</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand-read-1-req</td>
<td>+12%</td>
</tr>
<tr>
<td>rand-read-4-req</td>
<td>+20%</td>
</tr>
<tr>
<td>rand-write-1-req</td>
<td>+22%</td>
</tr>
<tr>
<td>rand-write-4-req</td>
<td>+12%</td>
</tr>
<tr>
<td>rand-rw-1-req</td>
<td>+3%</td>
</tr>
<tr>
<td>rand-rw-4-req</td>
<td>+22%</td>
</tr>
</tbody>
</table>
# Configuration Limitations

<table>
<thead>
<tr>
<th>Approach</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX</td>
<td>None</td>
</tr>
<tr>
<td>nvme://</td>
<td>One NVMe, one VM</td>
</tr>
</tbody>
</table>
| SPDK vhost-user-blk       | * Host must use hugepages  
                           |   * Guest must use VirtIO       |
| Device assignment         | * One NVMe, one VM  
                           |   * Guest must use NVMe        |
## Feature Availability

<table>
<thead>
<tr>
<th>Approach</th>
<th>Host block features</th>
<th>QEMU block features</th>
<th>Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nvme://</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPDK vhost-user-blk</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Device assignment</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Overall comparison

![Diagram showing performance and functionality comparison between different technologies: vfio-pci passthrough, SPDK vhost-user-blk, nvme://, POSIX.]
Status and future

- Status
  - Patches v3 on qemu-devel@nongnu.org:
  - Also available at github: https://github.com/famz/qemu

- TODO
  - Get it merged!
  - Integrate with multi-queue block layer
Benchmark configuration

- Host 1: Fedora 26 / RHEL 7 (x86_64)
  Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz x2
  64GB ram
  Intel Corporation DC P3700 380G
  FusionIO ioDrive2 340G
  Western Digital WD RE4 WD5003ABYX 500GB 7200 RPM 64MB

- Host 2: Fedora 26
  Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz
  16GB ram
  Samsung SSD 840 PRO 128G

- Guest: Fedora 26 (x86_64), 1 vCPU, 1GB ram

- Tool: fio-2.18

- Job:
  ramp_time = 30
  runtime = 30
  bs=4k
  rw={randread, randwrite, randrw}
  iodepth={1, 4}
THANK YOU