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 Virtualization Overhead

IOPS

SATA (HDD)
Latency reducing

- KVM optimizations
  - `kvm_halt_poll` by Paolo Bonzini
  - QEMU IOThread poll by Stefan Hajnoczi

- Kernel optimizations
  - `/sys/block/nvme0n1/queue/io_poll` by Jens Axboe

- 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
- BlockBackend
- Block layer
- QCOW2

QEMU

Host kernel

- VFIO
- vfio-pci.ko
Implementation

- $QEMU_SRC/util/vfio_helpers.c
  - A generic helper library for userspace drivers
  - Manages per device DMAR address 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 IOTThread 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..

1) **Guest** queues virtio req incl. I/O vectors (GPA, or vIOVA if vIOMMU enabled)

2) **VirtIO frontend** parses req and maps I/O addresses to HVA, preparing for DMA from/to guest

3) Frontend calls into **block backend** for I/O:
   - blk_aio_{preadv,pwritev}, with HVA

4) **Block layer** hands over the req to block driver:
   - bdrv_nvme.nvme_co_{preadv,pwritev}

5) **Block driver** finishes the I/O and calls back to frontend

6) virtqueue_push() and virtio_notify()
Step 5) Driver operations

(1) Check that the HVAs (offsets and lengths) are aligned
   If not, allocate an aligned bounce buffer to do next steps
(2) Translate host HVAs to the IOVAs in the underlying host IOMMU context,
    in page unit
(3) Prepare an NVMe Request structure that contains the page list 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
Addr translations

Guest app buffer

Guest physical addr

Host virtual address (no vIOMMU)

IOVA

PRP list

submission queue

NVMe
IOVA mapping

```
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);
```
Addr translations

Guest app buffer

Guest physical addr

Host virtual address (no vIOMMU)

IOVA addr space

PRP list

I/O queue

iova
How about host buffers?

- VFIO_IOMMU_MAP_DMA each new buffer as it comes
- Keep record of mapped buffers for later use if advisable
  - Distinguish throwaway / fixed mappings
- Use a pair of self-incrementing counter to track available IOVAs
- If free IOVAs run out, discard all temporary mappings and reset counter
- BlockDriver bdrv_register_buf bdrv_unregister_buf to optimize buffer I/O
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...
Comparison

![Comparison Diagram]

- vfio-pci passthrough
- SPDK vhost-user-blk
- nvme://
- POSIX
## Functional 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>
<tr>
<td>SPDK vhost-user-blk</td>
<td>* Requires hugepages</td>
</tr>
<tr>
<td></td>
<td>* No block jobs, snapshots or qcow2</td>
</tr>
<tr>
<td></td>
<td>* Fixed device type</td>
</tr>
<tr>
<td>Device assignment</td>
<td>All above except hugepages, plus no migration.</td>
</tr>
</tbody>
</table>

* Requires hugepages
* No block jobs, snapshots or qcow2
* Fixed device type
NVMe Performance Comparison

IOPS

randread-1-req, randread-4-req, randwrite-1-req, randwrite-4-req, randrw-1-req, randrw-4-req

- **linux-aio**
- **vfio-pci**
- **nvme://**
- **host**
IOPS compared to POSIX

<table>
<thead>
<tr>
<th>(IOPS)</th>
<th>POSIX</th>
<th>nvme://</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand-read-1-req</td>
<td>18212</td>
<td>20517</td>
<td>+12%</td>
</tr>
<tr>
<td>rand-read-4-req</td>
<td>63400</td>
<td>76503</td>
<td>+20%</td>
</tr>
<tr>
<td>rand-write-1-req</td>
<td>29096</td>
<td>35512</td>
<td>+22%</td>
</tr>
<tr>
<td>rand-write-4-req</td>
<td>76466</td>
<td>86329</td>
<td>+12%</td>
</tr>
<tr>
<td>rand-rw-1-req</td>
<td>15727</td>
<td>16283</td>
<td>+3%</td>
</tr>
<tr>
<td>rand-rw-4-req</td>
<td>42762</td>
<td>52222</td>
<td>+22%</td>
</tr>
</tbody>
</table>
Status and future

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

- 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
  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