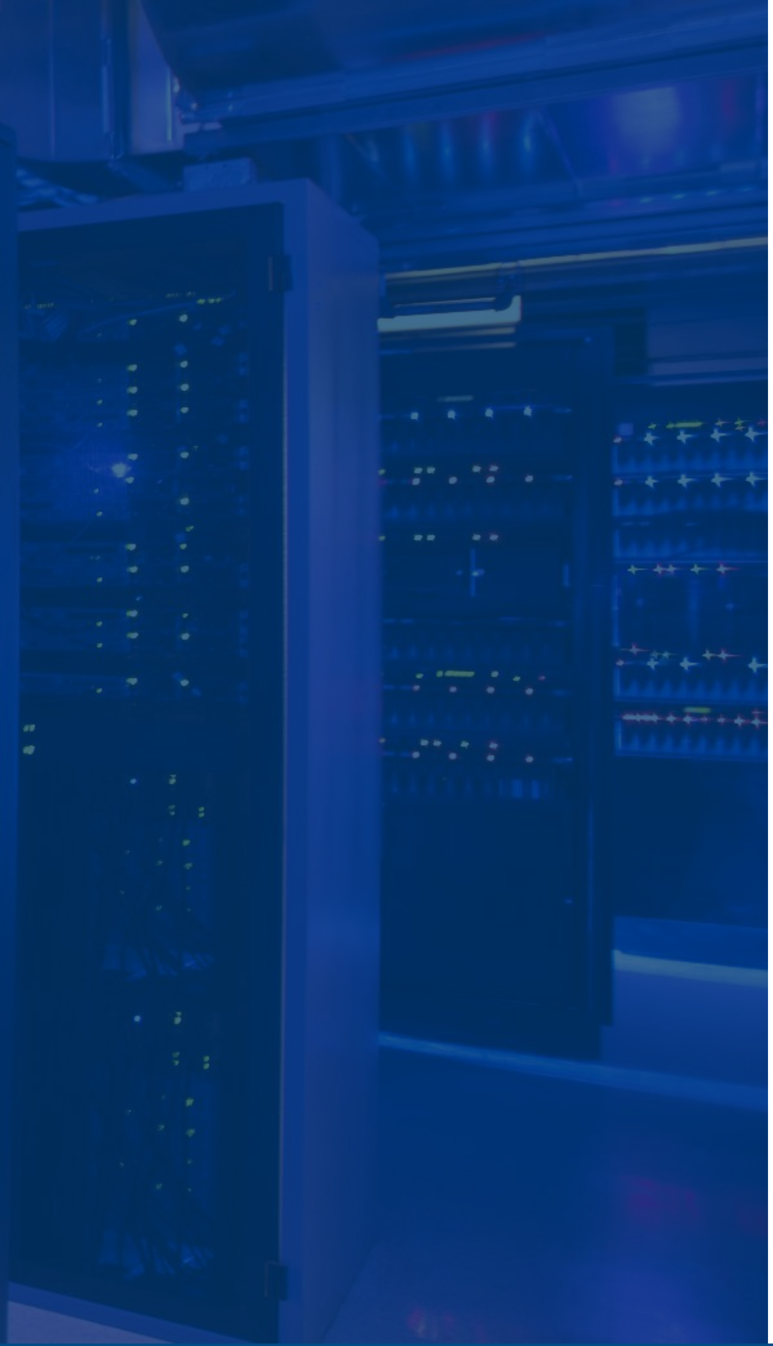
Virtio Devices Emulation in SPDK Based On VFIO-USER Protocol

Changpeng Liu & Xiaodong Liu

KVM FORUM 2022

intel®



Agenda

- Introduction
- Virtio BLK|SCSI Emulation
- Performance
- Summary

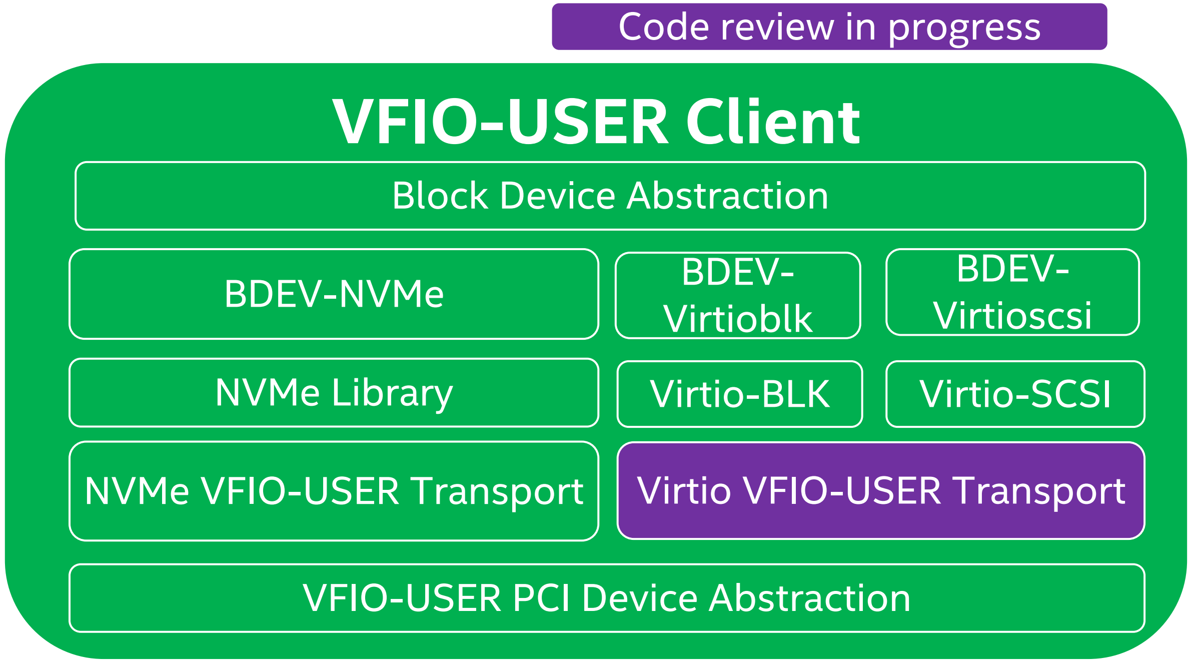
Introduction

VFIO-USER Introduction

- VFIO-USER is a protocol that allows a device to be emulated in a separate process outside of a Virtual Machine Monitor (VMM).
- The VFIO-USER specification is largely based on the Linux VFIO ioctl interface to implement them as messages to be sent over a UNIX domain socket.
- There are two parts of VFIO-USER:
  - VFIO-USER client runs in VMM or application.
  - VFIO-USER server for devices emulation in separate process.

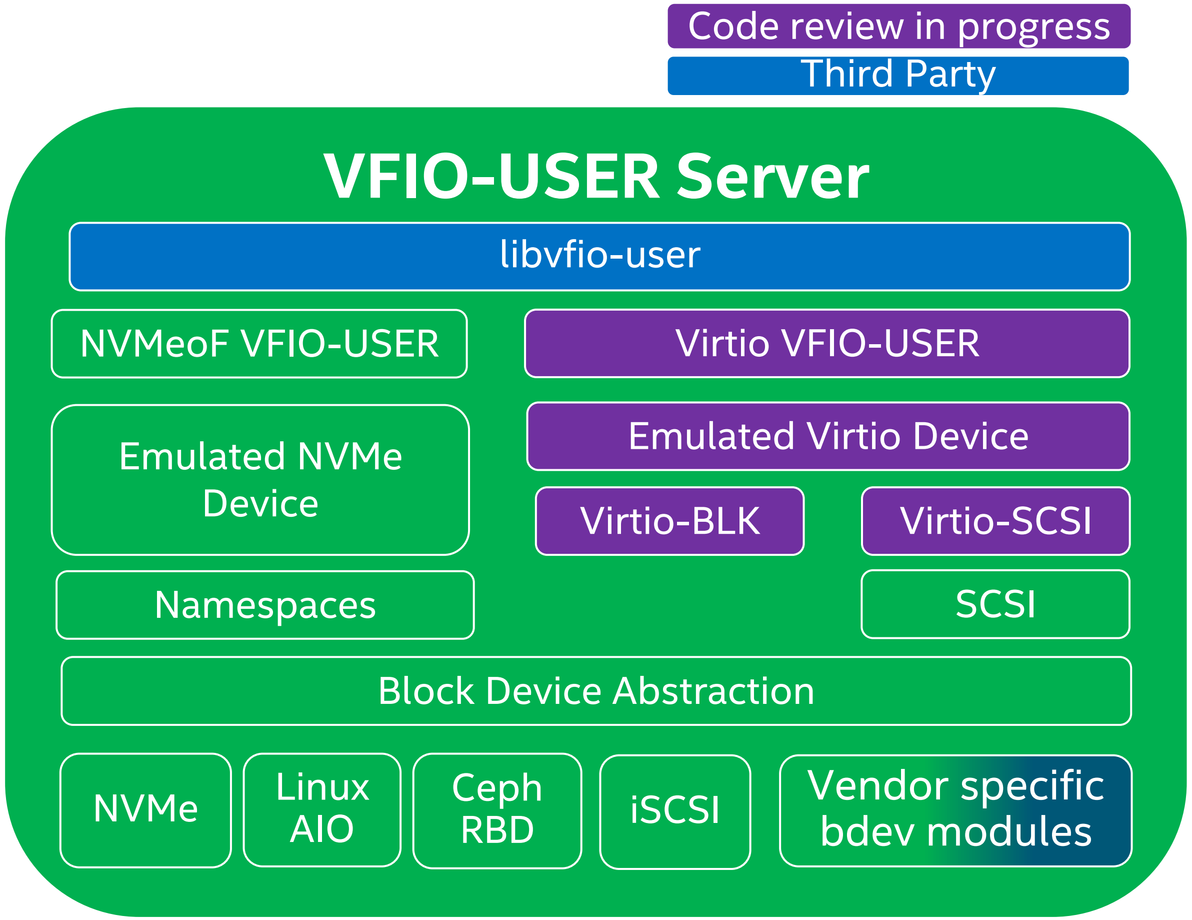
VFIO-USER Client in SPDK

- VFIO-USER client is used to provide PCI device abstraction access APIs
- Virtio and NVMe client library provide transport independent abstractions
- Virtio VFIO-USER transport is added to Virtio client library to use VFIO-USER protocol as communication channel to server process
- SPDK provides common block device abstraction based on different devices, such as NVMe, Virtio BLK and Virtio SCSI as a library



VFIO-USER Server in SPDK

- VFIO-USER Server is used to provide PCI devices emulation based on libvfio-user
- Emulated Virtio device library follows virtio protocol to response PCI BAR accesses from client and processes vrings
- Virtio BLK|SCSI processes detailed BLK|SCSI commands, SCSI commands are covered by the SCSI library in SPDK
- Block device layer provides abstraction of different block device types



VFIO-USER VS. VHOST-USER

- Client side:
  - SPDK virtio client library can support vhost-user, vfio-user and PCI as the transport channel to communicate with vhost-user, vfio-user server process, they are same in client side.
- Server side:

	SPDK VHOST-USER	SPDK VFIO-USER
Thread Model	One thread for one controller with multiple vrings	same
Virtio Feature Bits Support	Both packed and split vring are supported for virtio-blk and only split vring is supported for virtio-scsi	Both packed and split vring are supported for virtio-blk|scsi
Live Migration	Yes	Not implemented now
Multiple Sessions	Yes	Not implemented now
Interrupt Mode	Yes	Not implemented now

Benefits of VFIO-USER

- Unified client driver to enable different PCI device types.
  - VFIO-USER client driver support in QEMU.
  - VFIO-USER client driver support in Cloud Hypervisor.
  - SPDK VFIO-USER client support for NVMe and Virtio devices.
  - Live migration support to cover different PCI device types.
- VHOST-USER is designed for virtio devices
- Much thinner than VHOST-USER in VMM
  - PCI emulation is in remote process for VFIO-USER

Virtio BLK|SCSI Emulation

Virtio device layout

- VFIO Region 1: MSI-X Table, 0x0000 - 0x1000
- VFIO Region 2: MSI-X Pending Bit Array, 0x0000 - 0x1000
- VFIO Region 4:
  - Common configuration 0x0000 - 0x1000
  - ISR access 0x1000 - 0x2000
  - Device specific configuration 0x2000 - 0x3000
  - Notifications 0x3000 - 0x4000
  - It's up to user's configuration to set area `Notifications` as memory mappable or not(Interrupt or Polling mode in target).

VFIO Region 4 Callback

- Common configuration access responses based on virtio specification via (offset, length, R|W flag)

```
access_bar4-->
              -->common_cfg-->
                             -->features negotiation
                             --> queues setup
                             --> start|stop device
```

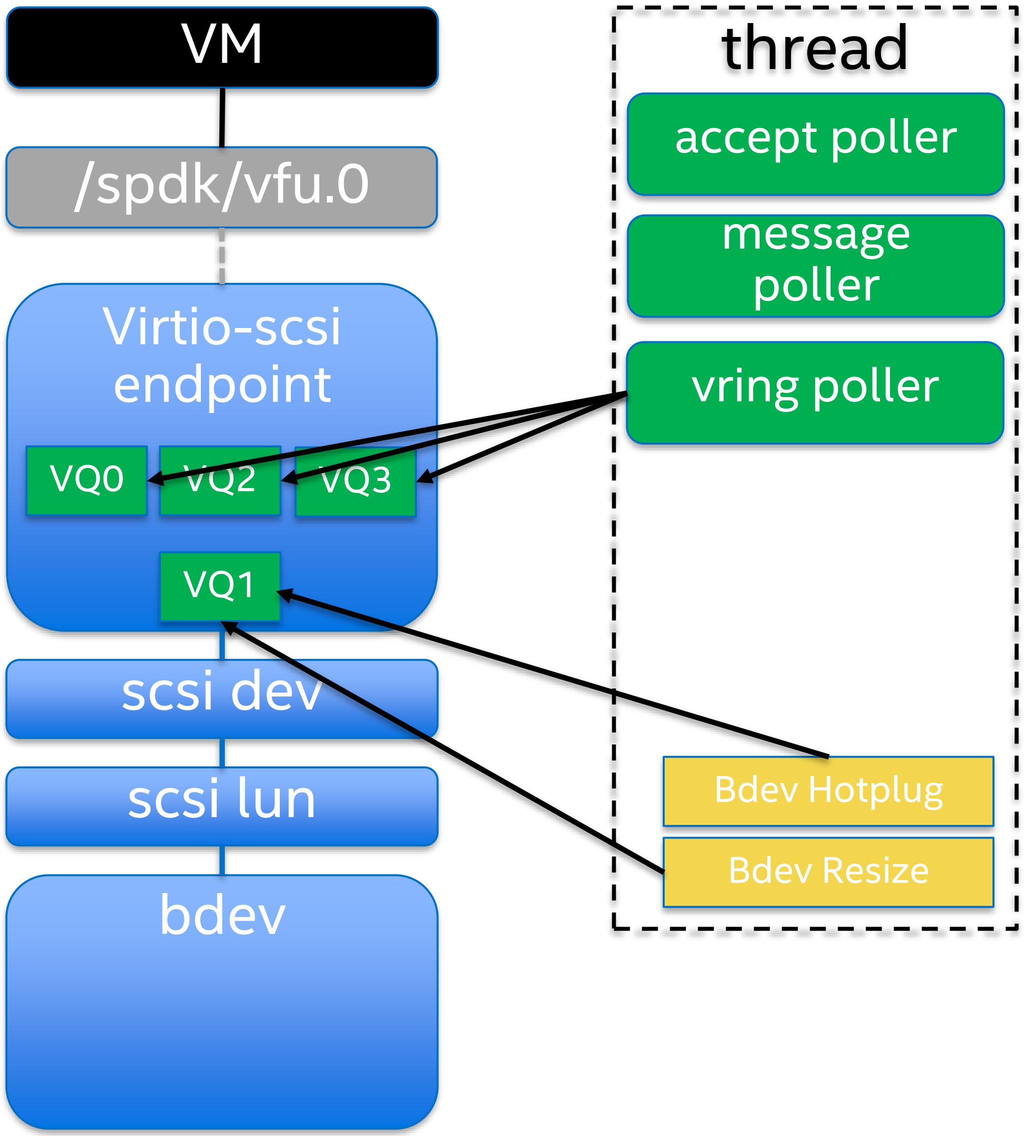
- Currently virtio-blk|scsi devices are supported and users can add other virtio device type support based on emulated virtio device library.
- Device specific configuration access will be redirect to device layer

  --virtio-blk contains `capacity`, `blk_size` etc.

  --virtio-scsi contains `num_queues` etc.

Virtio-SCSI Thread Model

- Listen to the UNIX domain socket on specified thread and start the accept poller
- QEMU connects to UNIX domain socket
- Accept poller starts a connection poller which will deliver socket messages to emulated virtio device library
- VFIO Region access callbacks are called
- Starts vring poller when VM asks to start device



Virtio-BLK|SCSI Commands Processing

- Virtio-BLK
  - READ/WRITE/WRITE ZEROES/DISCARD/FLUSH commands are supported, after parsing these commands from vrings, they will be mapped to block layer APIs such as spdk_bdev_readv/writev/write_zeroes/unmap/flush directly.
- Virtio-SCSI
  - SPDK SCSI library with device abstraction and mandatory SPC|SBC commands support, and we've used this library in iSCSI target.
  - We can use SPDK SCSI library to process SCSI commands.

Performance

VM Performance tests Configuration

QEMU command line

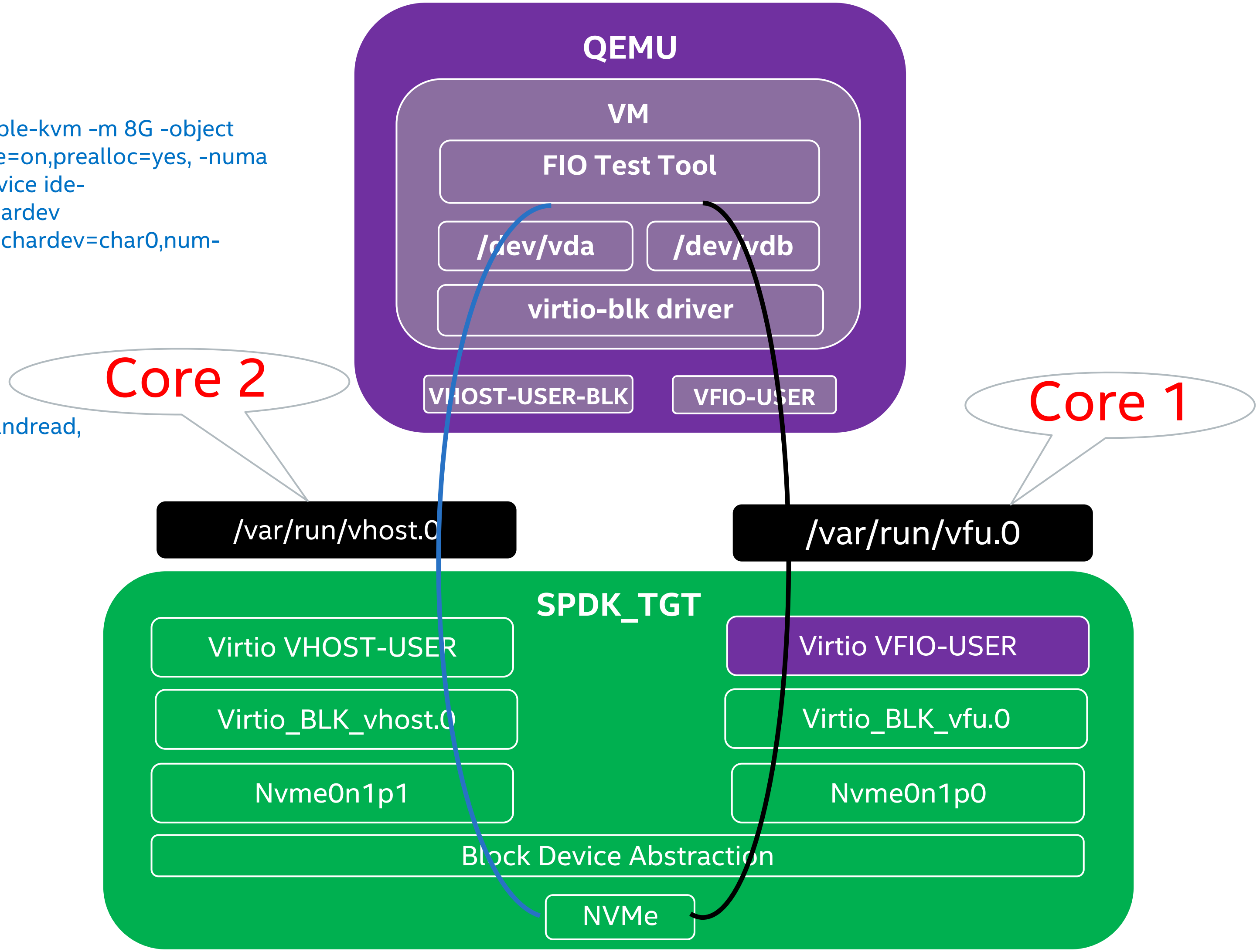
- "taskset -c 4-8 /qemu-devel/qemu-system-x86_64 -cpu host -smp 4 -enable-kvm -m 8G -object memory-backend-file,id=mem0,size=8G,mem-path=/dev/hugepages,share=on,prealloc=yes, -numa node,memdev=mem0 -drive file=/root/fedora_33.img,if=none,id=disk -device ide-hd,drive=disk,bootindex=0 -device vfio-user-pci,socket=/var/run/vfu.0 -chardev socket,id=char0,path=/var/run/vhost.0 -device vhost-user-blk-pci,id=blk0,chardev=char0,num-queues=4,packed=on"

FIO parameters in VM

- "filename=[/dev/vda,/dev/vdb],bs=4k,numjobs=4,iodepth=[1,4,8,16],rw=randread, ramp_time=60,runtime=1200"

SPDK Configuration

- P5800X Optane 1.5 TB split into two logical parts
- Unix Domain socket vfu.0 running in core 1
- Unit Domain socket vhost.0 running in core 2
- 4 IO queues with qsize=128



VM Performance

- Test case 1:

From iodepth=1 to 16, the performance number is almost same for the two controllers.

VM Test Workloads using logical volumes of one physical NVMe SSD	VFIO-USER BLK	VHOST-USER BLK
Random Read Iodepth 16, 4 Jobs	705K	689K
Random Read Iodepth 8, 4 Jobs	712K	699K
Random Read Iodepth 4, 4 Jobs	632K	602K
Random Read Iodepth 1, 4 Jobs	233K	239K

- Test case 2:

Compared with Test case 1, we replace NVMe SSD with NULL loopback device in this case and test random write workload with iodepth=64,128.

The purpose for this test case is just compare the virtualization overhead for the two controllers, and the test result shows that the performance number is still almost same.

VM Test Workloads using NULL loopback device as backend	VFIO-USER BLK	VHOST-USER BLK
Random Write Iodepth 128, 4 Jobs	779K	773K
Random Write Iodepth 64, 4 Jobs	775K	768K

HOST System Configuration: 2 * Intel(R) Xeon(R) Platinum 8180M CPU @ 2.50GHz; 128GB, 2666 DDR4, 6 memory Channels; Bios: HT disabled, Turbo disabled; OS: Fedora 30, kernel 5.6.13-100. VM configuration : 4 vcpus 8GB memory, 4 IO queues; VM OS: Fedora 33, kernel 5.10.8-200; Software: QEMU with vfio-user-pci patch, SPDK with VFIO-USER patches; VM: FIO 3.21, io depth=[128,64,16,8,4,1] numjobs=4, direct=1, block size=4k,randread,randwrite,ramp_time=60,runtime=1200; NVMe SSD: Intel (R) Optane (TM) SSD P5800X Series, 1.5TB

intel.

SPDK VIRTIO Client Performance

- Using same configuration as Test case 2 in server side
  - build/bin/spdk_tgt –m 0x6
  - scripts/rpc.py bdev_null_create Null0 102400 512
  - scripts/rpc.py bdev_null_create Null1 102400 512
  - scripts/rpc.py vfu_construct_endpoint vfu.0 --cpumask 0x2 --model-name virtio_blk
  - scripts/rpc.py vfu_virtio_blk_add_bdev vfu.0 --bdev-name Null0 --num-queues=4 --qsize=128 --packed-ring
  - scripts/rpc.py vhost_create_blk_controller vhost.0 Null1 --cpumask 0x4
- Using the following commands to start SPDK client test tool
  - test/bdev/bdevperf/bdevperf -r /var/tmp/spdk.sock.1 -g -s 2048 -q 128 -o 4096 -w randread -t 1200 -m 0x8
  - scripts/rpc.py -s /var/tmp/spdk.sock.1 bdev_virtio_attach_controller --dev-type blk --trtype vfio-user --traddr vfu.0 VirtioBlk0
  - test/bdev/bdevperf/bdevperf.py -s /var/tmp/spdk.sock.1 perform_tests
  - Repeat step2 to replace “vfio-user” with “user” and “vfu.0” with “vhost.0” for vhost-user blk tests

- Test case 3:

  Polling mode driver is running both on client and server side

SPDK Client Test using NULL loopback device as backend	VFIO-USER BLK	VHOST-USER BLK
Randread Iodepth 128, 1 thread	5407K	5851K

Summary

Plans

- For VHOST-USER, QEMU emulates PCI device part and leave the vring processing in SPDK vhost library, but for vring dequeue|enqueue processing, they are almost same in VHOST-USER and VFIO-USER, we will abstract this part as a common library in future.
- Interrupt mode support.
  - Accept poller and socket message poller can switch to interrupt mode after starting device
  - Vring poller interrupt support with non mapped `Notification` section.
- Live migration support.

Source Code

- Patches under review
  - https://review.spdk.io/gerrit/c/spdk/spdk/+/12315, Virtio BLK emulation in server side
  - https://review.spdk.io/gerrit/c/spdk/spdk/+/12673, Virtio SCSI emulation in server side
  - https://review.spdk.io/gerrit/c/spdk/spdk/+/13896, Virtio BLK client block device support
  - https://review.spdk.io/gerrit/c/spdk/spdk/+/13897, Virtio SCSI client block device support
  - https://github.com/oracle/qemu.git, branch `vfio-user-irqmask2`, QEMU VFIO-USER Client support

- Done
  - https://github.com/cloud-hypervisor/cloud-hypervisor, Cloud-Hypervisor VFIO-USER Client support
  - https://github.com/spdk/spdk/vfio-user, SPDK VFIO-USER Client support