STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers

Virtual Conference
September 28-29, 2021

A SNIA Event

High Performance NVMe Virtualization with SPDK and vfio-user

Ben Walker
Core Maintainer, SPDK
Intel Corporation

Agenda

- Standardization
- Emulating NVMe Devices
- NVMe Client Library
- Performance

Standardization

Brief Background

Need to emulate device outside VMM

- Performance
- Security
- Stability/resilience
- Device can even run in separate VM

Initially conceived for SPDK

- NVMe device emulation
- But much broader than this use case now!

What is Virtual Function I/O (vfio)?

"The VFIO driver is an IOMMU/device agnostic framework for exposing direct device access to userspace…"

In other words, an interface for writing user space device drivers

Originally to be used by virtual machines for PCI passthrough

This happens to be how SPDK's NVMe, CBDMA, and DSA drivers are built

Introducing The VFIO-USER Protocol

Modelled after the VFIO ioctls

- VFIO commands/structs do exactly what we need

vhost-user is to vhost as vfio-user is to vfio

Commands/messages passed over UNIX domain socket

STORAGE DEVELOPER CONFERENCE SDC 21

Emulating NVMe Devices

Approach

NVMe-oF already requires nearly full emulation of an NVMe device

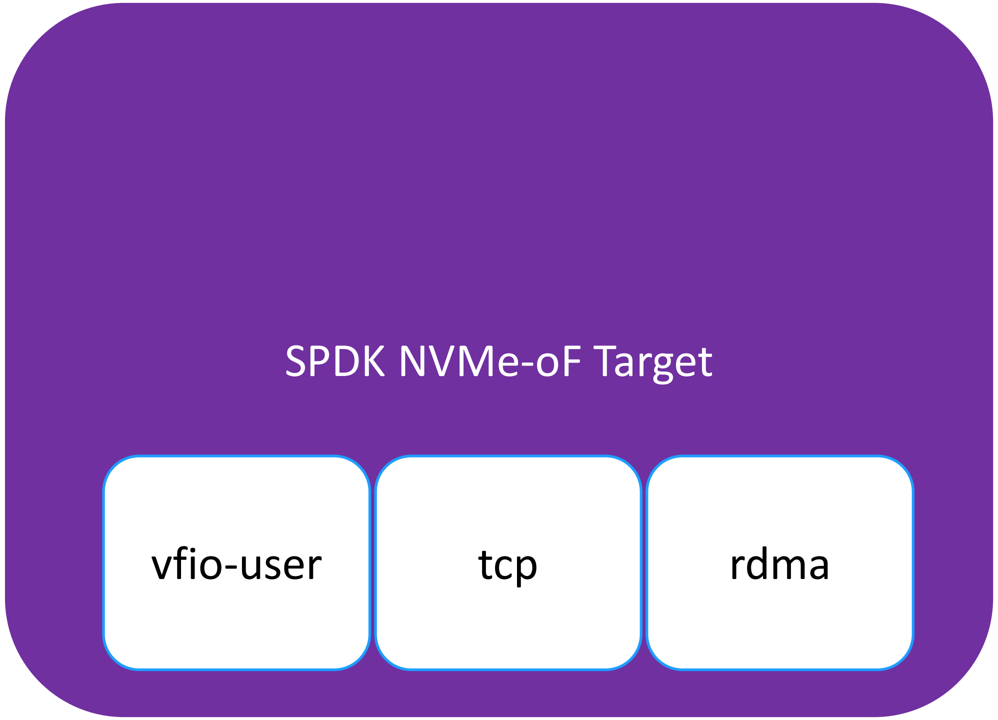
SPDK NVMe-oF already has a pluggable transport layer

Let's use the NVMe-oF Target!

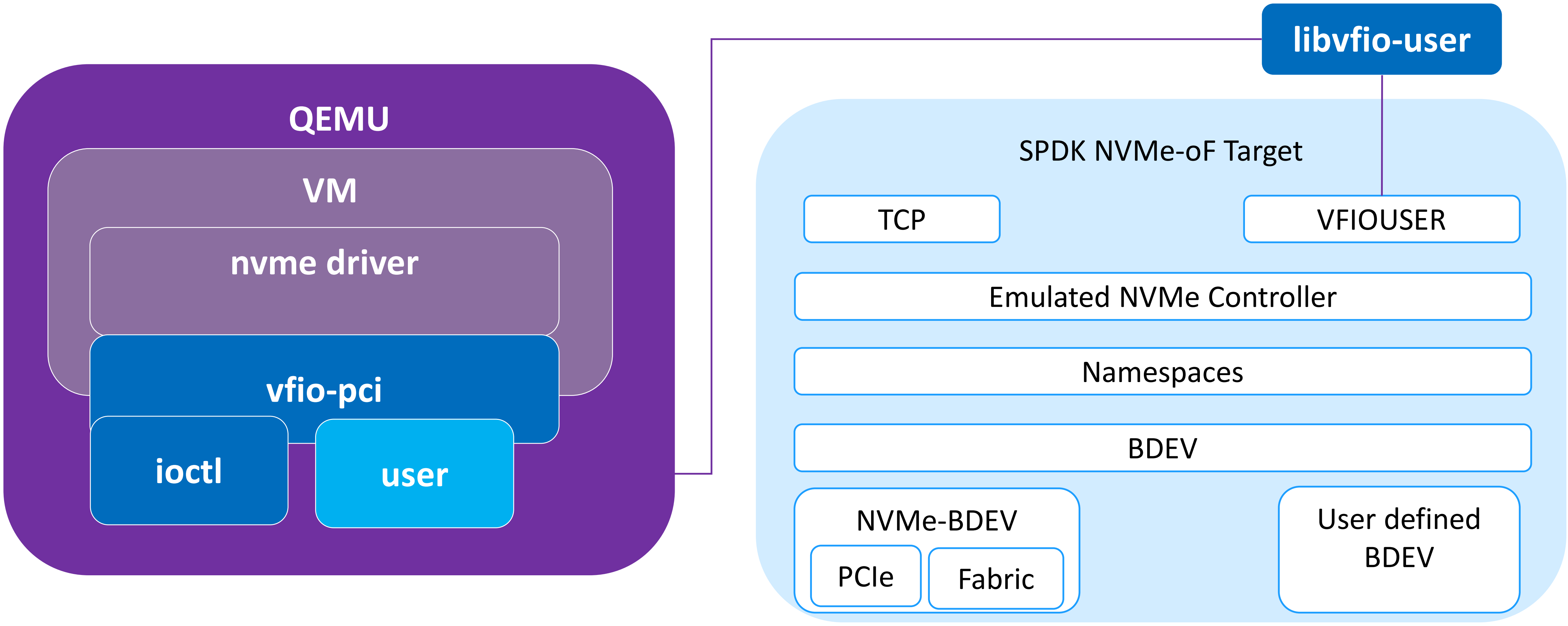
Let's make a new transport for NVMe-oF

A "shared memory" or "virtualization" transport

But fabrics *is* slightly different than PCIe. Some of the initialization flow is reversed.



Emulating an NVMe device



Challenges

The “listener” concept is different for vfio-user

- Need to “listen” on a Unix domain socket
- Only a single “host” can connect to the subsystem, rather than many
- No need to have an accept poller

Need to generalize concept of listener to “endpoint” in SPDK

- Push accept poller down into the transports. The vfio-user transport just won’t make one.

Challenges

Register reads and writes are very different for PCIe than fabrics	Libvfio-user provides a file descriptor that is signaled when an MMIO operation has occurred	Expand set of allowed Fabrics Property Get/Set commands
• MMIO rather than commands with requests and responses • The set of allowed registers is different	• Create a background thread blocked on that fd • Generate a fake fabrics property get/set command and send to target. For MMIO read, block until response.	• Wider range of registers allowed for PCIe

Challenges

Admin queue creation happens in reversed order compared to real fabrics devices

Real fabrics devices first create an admin queue, then read registers

PCIe devices first read registers, then create an admin queue

Need to create an admin queue as soon as “endpoint” is created so registers can be read

Generate fake admin queue creation command in transport, send to target

Success!

- Final patch that went into SPDK contained *only* a new transport.
  - No other code changes!
- Generalization is useful for future additional transports we expect to see
  - Running the NVMe-oF target as firmware?
  - QUIC?
- SPDK is a great NVMe emulator
  - Can leverage this to prototype new NVMe features and test from QEMU

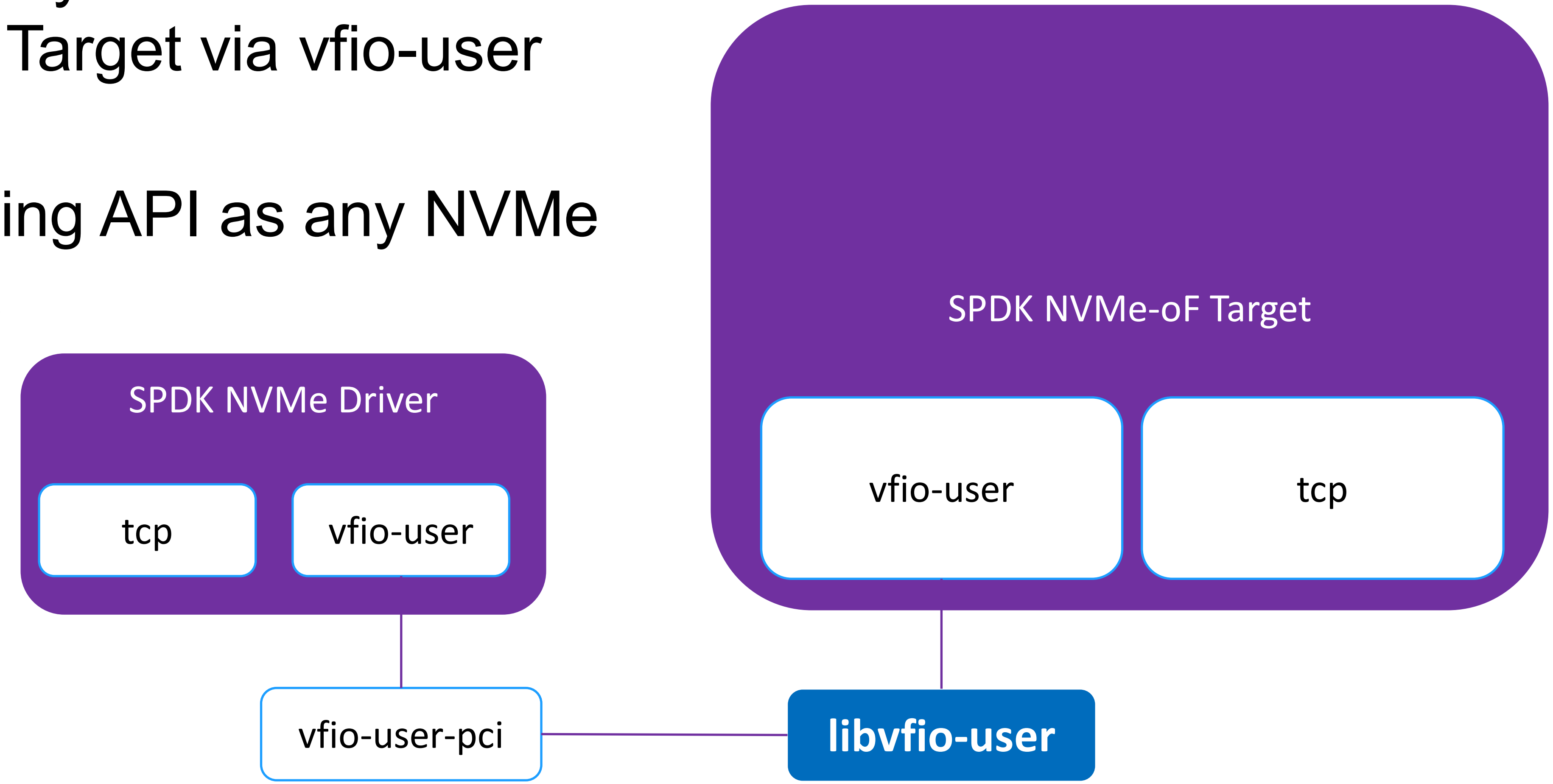
NVMe Client Library

We need a way to test the vfio-user transport

- Vfio-user is just a protocol spoken over a UNIX domain socket between two processes. The “client” does not need to be a VMM.
- SPDK’s nvme library supports a pluggable transport system
- Let’s implement a transport on the client side!

NVMe client library with vfio-user transport

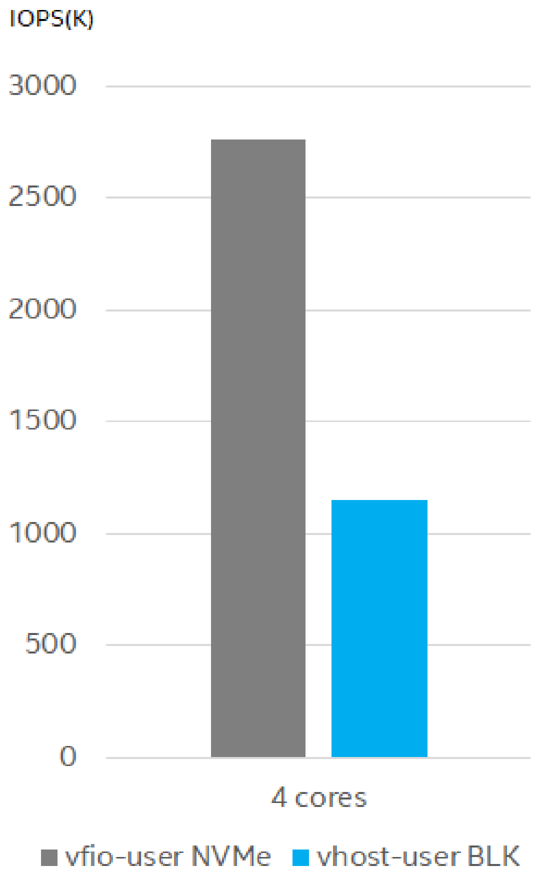
- SPDK NVMe library can connect with SPDK NVMe-oF Target via vfio-user transport.
- Same programming API as any NVMe device via SPDK



Performance

Benchmark: Threading Model

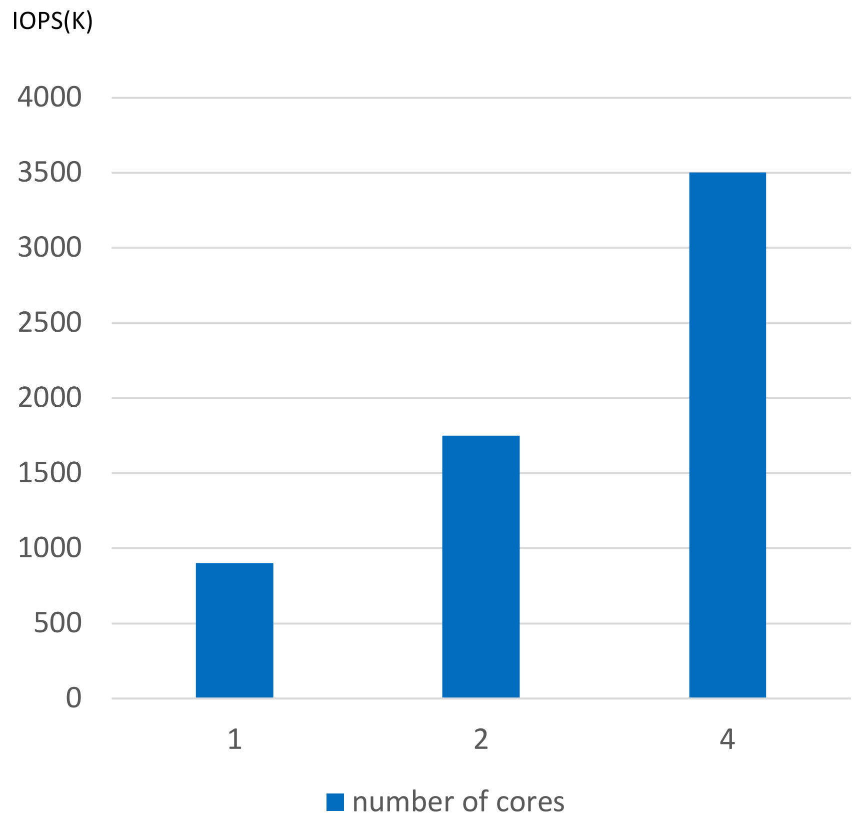
- Vhost-user forces the virtio-scsi or virtio-blk protocols
  - Virtio-scsi is heavily stateful. Requires locking to support multiple connections.
  - SPDK does virtio-scsi using just a single thread – it's faster than locking!
- Vfio-user lets us pick any device interface, so we pick NVMe!
  - NVMe can handle parallel submission and command processing



- 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 : 16 vcpus 16GB memory, 16 IO queues; VM OS: Fedora 33, kernel 5.10.8-200, blk-mq enabled; Software: QEMU with vfio-user-pci patch, IO distribution: SPDK, FIO 3.21, io depth=128, numjobs=16, direct=1, block size=4k,randread,total tested data size=400GiB

STORAGE DEVELOPER CONFERENCE SDC 21

Benchmark: Core Scaling



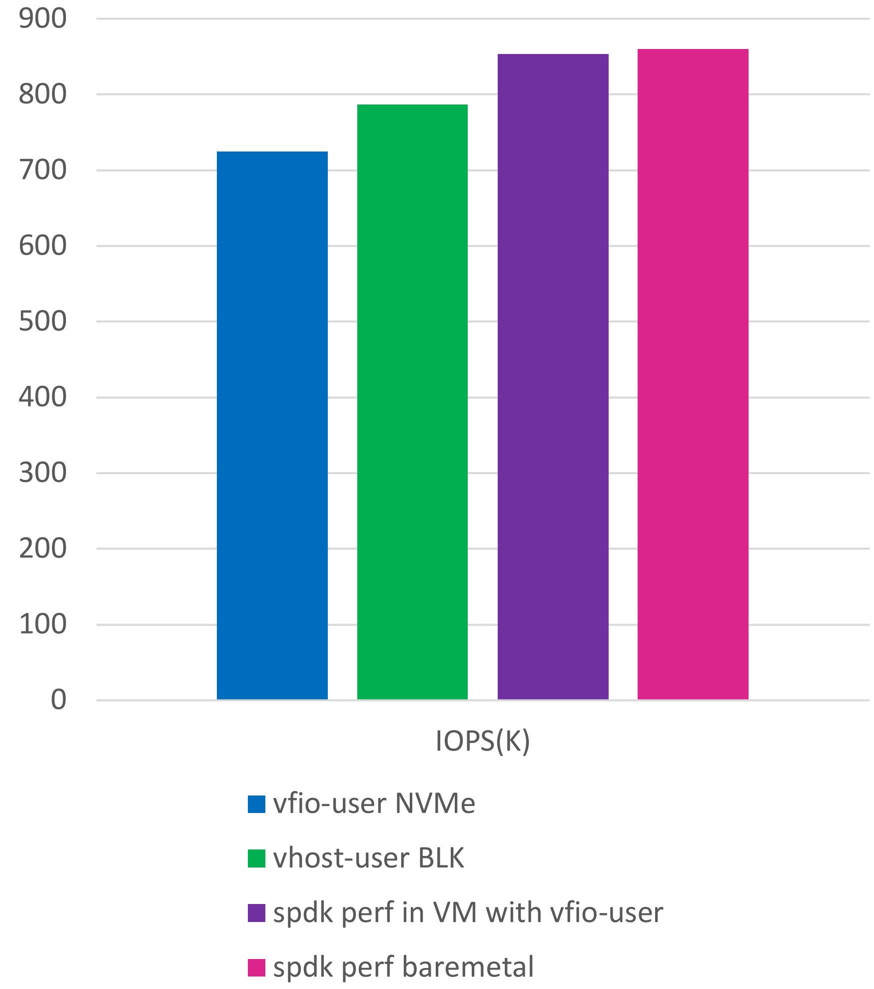
Vfio-user Core Scaling

- Scaling from 1 to 4 cores on target
- 4K Random Read, 128 Queue Depth from 4 fio jobs

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, blk-mq enabled; Software: QEMU with vfio-user-pci patch, IO distribution: SPDK, FIO 3.21, io depth=128, numjobs=4, direct=1, block size=4k,randread,total tested data size=400GiB

Benchmark: Single Thread

- P5800X SSD
- 4KiB Random Read at Queue Depth 128 on 4 queues from client
- Single core in NVMe-oF target



Please take a moment to rate this session.

Your feedback is important to us.

