xNVMe and io_uring
NVMe passthrough

What does it mean for the SPDK NVMe driver?

Simon A. F. Lund (Samsung)
Agenda

How (and why) did SPDK start?

SPDK’s Motivation

Linux Storage Abstractions

xNVMe Overview

Performance Comparisons

Next Steps
How (and why) did SPDK start?

<table>
<thead>
<tr>
<th>Event</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting with enterprise storage company</td>
<td>“We have all of these SAS SSDs in this system, but can’t get all of the performance out of them.”</td>
</tr>
<tr>
<td>NVMe ratified but not yet commercially available</td>
<td>The performance problem was only going to get worse!</td>
</tr>
<tr>
<td>OS support for NVMe ramping quickly</td>
<td>Including BSD-licensed FreeBSD drivers</td>
</tr>
<tr>
<td>Intel® Storage Group merged with division responsible for DPDK</td>
<td>DPDK already tackling this same problem for network packet processing</td>
</tr>
</tbody>
</table>

Timeline: 2013
SPDK’s Motivation

- Break the software bottleneck for high-performance storage workloads
- Build an open-source community to innovate and collaborate
- Balance between ”develop new” and “optimize existing”
- Broad set of abstractions and implementations
SPDK and NVMe

- Break the software bottleneck
  - Performant and efficient NVMe access is priority #1!

- Build an open-source community
  - Collaboration with xNVMe and Linux kernel

- Balance between “develop new” and “optimize existing”
  - Improve SPDK’s ability to leverage Linux NVMe

- Broad set of abstractions and implementations
  - Enable multiple ways of accessing NVMe with SPDK
Outline

- Why
  - What do you do, when the OS storage abstractions fail?
  - What do you do, when the deployment environments fail?
- What
  - Device handles via generic and anonymous namespaces (e.g. /dev/ng0n1)
  - Device communication via io_uring command (with NVMe Passthrough)
  - SPDK Integration: xNVMe and bdev_xnvm
Why? 1/2

General storage abstractions
Why: storage abstractions

- Generic abstractions
  - Supporting a variety of devices in the same fashion

- Long-lived and well-known abstractions of blocks and files

- When/how/why do abstractions fail for NVMe?
Why: storage abstractions “speaking NVMe”

- **Speaking NVMe**
  - Read/write using extended LBA formats
  - Ext: directives / write_zeroes / copy
  - ZNS: mgmt. send/receive, append
  - Key-Value: 
    store(k,v) / retrieve(v), list, delete, exists
  - New command-sets: Computational Storage

---

Userland

Speak File

Speak NVMe

Syscall

FS Abstraction

Speak Block

Sync

loctl()

Kernel

IO Stack

Block Abstraction

Device

Speak NVMe

/fsdev/nvme0n1

NVMe Driver

Speak NVMe
Why: storage abstractions “speaking NVMe”

- **Speaking NVMe**
  - Read/write using extended LBA formats
  - Ext: directives / write_zeroes / copy
  - ZNS: mgmt. send/receive, append
  - Key-Value: 
    store(k,v) / retrieve(v), list, delete, exists
  - New command-sets: 
    Computational Storage

- **Abstraction failure;** must bypass OS abstractions to utilize devices
Why: device handles

- Everything is a file with NVMe represented as
  - NVMe Controllers as char devices (e.g. /dev/nvme0)
  - NVMe Namespaces as block devices (e.g. /dev/nvme0n1)
    - Caveat: only for NVM and ZNS Command-Sets
Why: device handles

- Everything is a file with NVMe represented as
  - NVMe Controllers as char devices (e.g. /dev/nvme0)
  - NVMe Namespaces as block devices (e.g. /dev/nvme0n1)
    - Caveat: only for NVM and ZNS Command-Sets

- Plug in a device with a command-set other than NVM/ZNS
  - Only the controller handle appears (e.g. /dev/nvme0)
  - Device does not fit, or match assumptions of, the Linux Block Device model
  - No representation of / FS entry to get a handle to the namespace
Why: device handles

- Everything is a file with NVMe represented as
  - NVMe Controllers as char devices (e.g. /dev/nvme0)
  - NVMe Namespaces as block devices (e.g. /dev/nvme0n1)
    - Caveat: only for NVM and ZNS Command-Sets

- Plug in a device with a command-set other than NVM/ZNS
  - Only the controller handle appears (e.g. /dev/nvme0)
  - Device does not fit, or match assumptions of, the Linux Block Device model
  - No representation of / FS entry to get a handle to the namespace

⇒ Abstraction failure; no means to get a handle to the namespace
Why: device communication

- **Efficiency** via `io_uring`
  - reducing the cost of crossing the border between userland and kernel
- Shared memory (rings)
  - Instead of memory-transfers
- Resource registration
  - Reduce lookup-cost
- Polling (`IOPOLL` | `SQPOLL`)
- Batching
  - One syscall => multiple commands
Why: device communication

- **Efficiency** via io_uring
  - reducing the cost of crossing the border between userland and kernel
- **Shared memory (rings)**
  - Instead of memory-transfers
- **Resource registration**
  - Reduce lookup-cost
- **Polling** (IOPOLL | SQPOLL)
- **Batching**
  - One syscall ➞ multiple commands
Why: device communication

- Speaking NVMe
  - Read/write using extended LBA formats
  - Ext: directives / write_zeroes / copy
  - ZNS: mgmt. send/receive, append
  - Key-Value: store(k,v) / retrieve(v), list, delete, exists
  - New command-sets: Computational Storage

- Facility: NVMe driver ioctl()
Why: device communication

- Speaking NVMe
  - Read/write using extended LBA formats
  - Ext: directives / write_zeroes / copy
  - ZNS: mgmt. send/receive, append
  - Key-Value: store(k,v) / retrieve(v), list, delete, exists
  - New command-sets: Computational Storage

- **Facility**: NVMe driver ioctl()
- **Abstraction failure**: no kernel facility to “Speak NVMe” efficiently
Existing solutions

- Move the storage abstraction out of the kernel and into userland

⇒ The SPDK Block Device abstraction (**bdev**)  
⇒ The SPDK NVMe driver

So, when does this fail?
Why? 2/2

Deployment Environments
Why: deployment environments

- Deployment of SPDK Apps using the SPDK NVMe driver
  - Requirement: detach the Kernel NVMe driver ➔ bind to vfio-pci/uio_generic
Why: deployment environments

- Deployment of SPDK Apps using the SPDK NVMe driver
  - **Requirement**: detach the Kernel NVMe driver ➔ bind to vfio-pci/uio_generic

- HW Failure
  - Other devices in the same iommu-group ➔ No detachment
  - Unsupported IOMMU / PCIe bar address-space ➔ binding failure
Why: deployment environments

- Deployment of SPDK Apps using the SPDK NVMe driver
  - **Requirement**: detach the Kernel NVMe driver \(\Rightarrow\) bind to vfio-pci/uio_generic

- HW Failure
  - Other devices in the same iommu-group \(\Rightarrow\) No detachment
  - Unsupported IOMMU / PCIe bar address-space \(\Rightarrow\) binding failure

- Cloud failure
  - Sheer lack of NVMe devices \(\Rightarrow\) Encapsulated storage-device-services
  - Restrictive environments
Why: io_uring command for SPDK?

- What do you do, when the deployment environment fails?
  ➔ **Fallback**: operating system managed (bdev_aio / bdev_uring)
Why: io_uring command for SPDK?

- What do you do, when the deployment environment fails?
  ➤ **Fallback**: operating system managed (bdev_aio / bdev_uring)

- Enable deployment of **SPDK** in environments otherwise unavailable
- Enable deployment of **SPDK** with minimal performance hit
  ➤ **Goals** of Linux and **SPDK** are aligned
Why: goals for Linux

An **open-ended** representation of NVMe devices for existing and new NVMe Command-Sets with a **fast-path** for communication

**Handles**

- Bring up devices regardless of Linux device model match

**Communication**

- Speak NVMe “natively”
- Scale as efficiently as **io_uring**
- Scale as efficiently as the SPDK NVMe Driver
What? 1/3

Generic device handles
What: a solution to handles

Handles

- NVMe generic char interface e.g. /dev/ng0n1
What: a solution to handles

Handles

- NVMe generic char interface e.g. /dev/ng0n1
- Initial support: Linux 5.13 (June 2021)
  - Brings up handles for namespaces with NVM and ZNS command-sets
- Command-set independence: Linux 6.0
  - Brings up handles for namespaces with any command-set
What: a solution to handles

Handles

- NVMe generic char interface e.g. /dev/ng0n1
- Initial support: Linux 5.13 (June 2021)
  - Brings up handles for namespaces with NVM and ZNS command-sets
- Command-set independence: Linux 6.0
  - Brings up handles for namespaces with any command-set

Device files are provided **regardless** of a matching device model, thereby enabling handles for existing and future NVMe command-sets.
What? 2/3

Communication via io_uring **command** (io_uring_cmd)
What: **io_uring command**

- **Generic facility** to attach **io_uring** capabilities to a command **provider**
- Larger ring-entries embedding **commands** and their **completions**
- Command **Provider** (driver, file-system, etc.)
What: io_uring command

- Generic facility to attach io_uring capabilities to a command provider
- Larger ring-entries embedding commands and their completions
- Command Provider (driver, file-system, etc.)
- One such command Provider is the NVMe driver
  - Providing NVMe passthrough commands
  - Commands defined equivalent to NVMe driver IOCTLs
  - NVMe driver IOCTL extended with iovec support

note: this was a requirement enabling non-bounce-buffer utilization by the SPDK bdev abstraction
What: io_uring command

Handles

- Bring up devices regardless of Linux device model match

Communication

- Speak NVMe “natively”
- Scale as efficiently as io_uring?
- Scale as efficiently as the SPDK NVMe Driver?

For more: see Kanchan Joshi’s Linux Plumbers Conference slides

What 3/3

SPDK Integration via xNVMe (bdev_xnvme)
- Core API
  - Commands and Buffers
  - Queues & Callbacks
- Command-Set Helpers
  - NVM read / write / write_zeroes / copy
  - ZNS mgmt. send / receive / append
  - KV store / retrieve / list / exists / delete
- Command-Line Tools
  - xnvme, lblk, zoned, kvs
- xNVMe is used for
  - I/O interface independence
  - Minimal abstraction cost
  - Convenient command-line tools
  - Rapid experimentation via Python

- Further details
  SYSTOR22 Presentation and Paper
  https://www.youtube.com/watch?v=YoA6FVnc_pU
  https://dl.acm.org/doi/abs/10.1145/3534056.3534936
  Web: https://xnvme.io/
SPDK Integration: bdev_xnvme

- With SPDK v22.09 a new bdev module is introduced: bdev_xnvme
- The xNVMe bdev module calls into the core xNVMe API
- A single bdev implementation for:
  - libaio, io_uring, and io_uring_cmd
  - Device-specific handling (zone mgmt.)
- Further details, Krishna K. Reddy
  - SDC Presentation
    https://www.youtube.com/watch?v=WbdCht6f_tU
Comparison: peak IOPS for saturated CPU

io_uring_cmd vs io_uring
io_uring_cmd vs SPDK NVMe Driver

SPDK Bdev implementations (aio, uring, xNVMe)
Comparison: system and software

- Core i5-12600, SMT enabled, Turbo-Boost disabled
  - 4x Samsung 980 Pro 1TB (512 RR ~1.0M IOPS / 4K RR 1.0M IOPS)
  - 4x Samsung 980 Pro 2TB (512 RR ~0.8M IOPS / 4K RR 0.8M IOPS)
- Device roofline \(~8M\) IOPS (according to spec. Sheet)
- Software
  - Linux 6.5
  - fio 3.34
  - xNVMe v0.7.1
  - SPDK v23.04 + patches for xNVMe submodule updated to v0.7.1
Comparison: system and software

- Linux Kernel version 6.5
- Debian Bullseye kernel config with the following changes
  - CONFIG_BLK_CGROUP=N
  - CONFIG_BLK_WBT_MQ=N
  - CONFIG_HZ=250
  - CONFIG_RETPOLINE=N
  - CONFIG_PAGE_TABLE_ISOLATION=N
- NVMe driver loaded with as
  - modprobe -r nvme && modprobe nvme poll_queues=1
  - /sys/block/{device}/queue/iostats set to 0
  - /sys/block/{device}/queue/nomerges set to 2
  - /sys/block/{device}/queue/wbt_lat_usec set to 0
Comparison: system and software

- **Tools**
  - fio: t/io_uring via "one-core-peak.sh"
  - fio: t/io_uring manually invocation
  - bdevperf

- Logs of all runs are provided for inspection and reproducibility
  - [https://github.com/safl/sceb](https://github.com/safl/sceb)

- Also contains scripts, hw-info information, kernel-config etc.
## io_uring vs. io_uring_cmd

<table>
<thead>
<tr>
<th>#Devices</th>
<th>Millions of 512 byte IOPS via io_uring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-n=#Devices IOPOLL</td>
</tr>
<tr>
<td>1</td>
<td>1.17</td>
</tr>
<tr>
<td>2</td>
<td><strong>2.32</strong></td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>2.18</td>
</tr>
<tr>
<td>5</td>
<td>2.10</td>
</tr>
<tr>
<td>6</td>
<td>2.03</td>
</tr>
<tr>
<td>7</td>
<td>2.03</td>
</tr>
<tr>
<td>8</td>
<td>2.02</td>
</tr>
</tbody>
</table>
### io_uring vs. io_uring_cmd

<table>
<thead>
<tr>
<th>#Devices</th>
<th>Millions of 512 byte IOPS via io_uring</th>
<th>Millions of 512 byte IOPS via io_uring_cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-n=#Devices IOPOLL</td>
<td>-n=#Devices IOPOLL</td>
</tr>
<tr>
<td></td>
<td>-n2 -c16 --s16 IOPOLL</td>
<td>-n2 -c16 --s16 IOPOLL</td>
</tr>
<tr>
<td></td>
<td>-n1 NOPOLL NOBATCH</td>
<td>-n1 NOPOLL NOBATCH</td>
</tr>
<tr>
<td></td>
<td>-n1 SQPOLL</td>
<td>-n1 SQPOLL</td>
</tr>
<tr>
<td>1</td>
<td>1.17</td>
<td>1.16</td>
</tr>
<tr>
<td>2</td>
<td>2.32</td>
<td>2.32</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>3.18</td>
</tr>
<tr>
<td>4</td>
<td>2.18</td>
<td>4.16</td>
</tr>
<tr>
<td>5</td>
<td>2.10</td>
<td>4.12</td>
</tr>
<tr>
<td>6</td>
<td>2.03</td>
<td>3.97</td>
</tr>
<tr>
<td>7</td>
<td>2.03</td>
<td>3.82</td>
</tr>
<tr>
<td>8</td>
<td>2.02</td>
<td>3.97</td>
</tr>
</tbody>
</table>

43 | © 2023 SNIA. © Simon A. F. Lund / SSDR / Samsung / GOST. All Rights Reserved.
Eval: goals for Linux

An **open-ended** representation of NVMe devices for existing and new NVMe Command-Sets with a **fast-path** for communication

**Handles**

- Bring up devices regardless of Linux device model match

**Communication**

- Speak NVMe “natively”
- Scale as efficiently as io_uring
- Scale as efficiently as the SPDK NVMe Driver?

<table>
<thead>
<tr>
<th>Peak IOPS in Millions</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_uring</td>
</tr>
<tr>
<td>io_uring_cmd</td>
</tr>
</tbody>
</table>
Comparison: IOPS via SPDK

- I/O generator
  - `bdevperf -q 128 -o 512 -w randread -t10 <bdev_conf> -m <variations>

- Two variations
  - `-m[0]; using a single core and no thread-sibling`
  - `-m[0,1]; using a single core and its thread-sibling`
  - Equivalent comparison of SMT effect as is done by `t/io_uring`
Comparison: IOPS via SPDK

- Satures a single SMT thread

<table>
<thead>
<tr>
<th># Devices</th>
<th>Millions of 512 byte IOPS via the SPDK NVMe Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-m[0]</td>
</tr>
<tr>
<td>1</td>
<td>1.15</td>
</tr>
<tr>
<td>2</td>
<td>2.31</td>
</tr>
<tr>
<td>3</td>
<td>3.34</td>
</tr>
<tr>
<td>4</td>
<td>4.35</td>
</tr>
<tr>
<td>5</td>
<td>5.22</td>
</tr>
<tr>
<td>6</td>
<td>6.11</td>
</tr>
<tr>
<td>7</td>
<td>7.11</td>
</tr>
<tr>
<td>8</td>
<td><strong>7.24</strong></td>
</tr>
</tbody>
</table>
Comparison: IOPS via SPDK

### Why the gap?

- **Generic facility**
  - Does more than specialized user-space driver
  - Taps into generic kernel-infra
  - `io_uring_cmd` specific I/O path reduction

### Un-tapped optimizations

- Management of DMA Mapping

<table>
<thead>
<tr>
<th># Devices</th>
<th>Millions of 512 byte IOPS via the SPDK NVMe Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-m[0]</td>
</tr>
<tr>
<td>1</td>
<td>1.15</td>
</tr>
<tr>
<td>2</td>
<td>2.31</td>
</tr>
<tr>
<td>3</td>
<td>3.34</td>
</tr>
<tr>
<td>4</td>
<td>4.35</td>
</tr>
<tr>
<td>5</td>
<td>5.22</td>
</tr>
<tr>
<td>6</td>
<td>6.11</td>
</tr>
<tr>
<td>7</td>
<td>7.11</td>
</tr>
<tr>
<td>8</td>
<td>7.24</td>
</tr>
</tbody>
</table>
Eval: goals for Linux

An **open-ended** representation of NVMe devices for existing and new NVMe Command-Sets with a **fast-path** for communication

**Handles**
- Bring up devices regardless of Linux device model match

**Communication**
- Speak NVMe “natively”
- Scale as efficiently as io_uring
- Scale as efficiently as the SPDK NVMe Driver?

<table>
<thead>
<tr>
<th>Peak IOPS in Millions</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_uring</td>
</tr>
<tr>
<td>io_uring_cmd</td>
</tr>
<tr>
<td>SPDK</td>
</tr>
</tbody>
</table>
Comparison: bdev implementations

- Compare the following
  - bdev_xnvme vs bdev_uring
  - bdev_xnvme vs bdev_aio
  - bdev_xnvme with io-mechanisms: libaio / io_uring / io_uring_cmd

- Using bdevperf
  - Compare single-device qd=1 for a sense of overhead
  - Compare single-device qd=128 for a sense of scale

- Provide the data to motivating next steps for bdev_xnvme
Comparison:

SPDK bdevs using libaio

*bdev_xnvme* vs *bdev_aio*

*bdev_xnvme*: `{io_mechanism=libaio}`
bdev\_aio vs bdev\_xnvme

- bdev\_xnvme at scale with bdev\_aio
Comparison:

SPDK bdevs using io_uring

bdev_xnvme vs bdev_uring
bdev_xnvme: {io_mechanism=io_uring}
bdev_uring vs bdev_xnvme

- bdev_xnvme at scale with bdev_uring
bdev_uring vs bdev_xnvme

- bdev_xnvme at scale with bdev_uring
- bdev_xnvme “out-scales” bdev_uring with IOPOLL enabled
Comparison:

SPDK bdev using io_uring_cmd

bdev_xnvme vs bdev_uring

bdev_xnvme: {io_mechanism=io_uring_cmd}

Single device
bdev_uring vs bdev_xnvme

• bdev_xnvme (io_uring_cmd) > bdev_uring
bdev_uring vs bdev_xnvme

- bdev_xnvme (io_uring_cmd) > bdev_uring
- bdev_xnvme (io_uring_cmd) > bdev_xnvme (io_uring)
Comparison:

SPDK bdev using io_uring_cmd

bdev_xnvme vs bdev_uring
bdev_xnvme: \{io_mechanism=io_uring_cmd\}

Multiple device
bdev_uring vs bdev_xnvme

- bdev_xnvme (io_uring_cmd) > bdev_uring
  - Both with and without IOPOLL
- bdev_xnvme (io_uring_cmd) > bdev_xnvme (io_uring)
What are next steps?
Next Steps: io_uring_cmd

- **Handles / Encapsulation**
  - I/O access-control matching file-permissions on /dev/ng*n*
  - Disable CAP_SYS_ADMIN for identify-commands (ns,ns-cs,ctrlr,ctrlr-cs,etc.)
  - Enable non-root access to device information such as maximum-data-transfer-size (MDTS), device properties

- **Communication**
  - Investigate potentials for large-block-sizes / hugepages
  - Investigate DMA pre-mapping
Next Steps: io_uring_cmd

- Handles / Encapsulation [DONE]
  - I/O access-control matching file-permissions on /dev/ng*n*
  - Disable CAP_SYS_ADMIN for identify-commands (ns,ns-cs,ctrlr,ctrlr-cs,etc.)
  - Enable non-root access to device information such as maximum-data-transfer-size (MDTS), device properties

- Communication
  - Investigate potentials for large-block-sizes / hugepages
  - Investigate DMA pre-mapping
Next Steps: bdev_xnvme

- Efficiency; match the IOPS rate achieved by the other bdevs
  - Exploring opportunities to enable batching
  - Performance “policy” e.g. “conserve_cpu” to disable optimizations
  - Otherwise: auto-enable `io_uring` optimizations where applicable and gracefully degrade in case of lacking system support

- Functionality
  - NVM commands: Write Zeroes, Flush
  - ZNS commands: (Zone Management Send/Receive)

- Deployment on Windows (`IOCP` and `IORING`)
  ➔ Broaden SPDK deployment while matching interface efficiency
Next Steps: bdev_xnvme

- Efficiency: match the IOPS rate achieved by the other bdevs
  - Exploring opportunities to enable batching
  - Performance “policy” e.g. “conserve_cpu” to disable optimizations
  - Otherwise: auto-enable io_uring optimizations where applicable and gracefully degrade in case of lacking system support

- Functionality
  - NVM commands: Write Zeroes, Flush
  - ZNS commands: (Zone Management Send/Receive)

- Deployment on Windows (IOCP and IORING)
  ➔ Broaden SPDK deployment while matching interface efficiency
Next Steps: xNVMe

- **Currently supported ✓**
  - `IORING_SETUP_{IOPOLL|SQPOLL|SINGLE_ISSUER}`
  - Resource-registration (files)
  - **Batching**: done on-behalf of the user via delayed submission

- **Currently missing**
  - `IORING_SETUP_{COOP|DEFER}_TASKRUN`
  - Resource-registration (buffers, rings)

- **General optimizations**: sqe-reuse, alignment, command-construction
So, what does it mean for SPDK?

- The xNVMe **bdev** shows promise of encapsulating Linux kernel NVMe interface for the bdev abstraction
  - Single bdev to handle *libaio*, *io_uring*, and *io_uring_cmd*
  - Single bdev to handle *zone-management*
- A wider range of deployment of SPDK Applications
- Closer collaboration and integration of storage eco-systems
- What does it mean for the SPDK NVMe driver?
Thanks!

- **Collaboration**
  - Reproducing io_uring_cmd vs SPDK NVMe benchmarks
  - Linux Kernel io_uring_cmd optimizations
  - SPDK bdev_xnvme optimizations and functional expansion
  - xNVMe optimization and functional expansion
  - Link to previous presentation at SPDK Virtual Forum 2022
    - [https://youtu.be/aYALmcP6PDU?si=H-TC_CJWgERzrd8W](https://youtu.be/aYALmcP6PDU?si=H-TC_CJWgERzrd8W)

- **Contact**
  - SPDK Slack Channels: [https://spdk-team.slack.com/](https://spdk-team.slack.com/)
  - Samsung GOST / xNVMe @ Discord: [https://discord.gg/XCbBX9DmKf](https://discord.gg/XCbBX9DmKf)
Please take a moment to rate this session.

Your feedback is important to us.