Programming Emerging Storage Interfaces

Simon A. F. Lund
Samsung / SSDR
Programming Emerging Storage Interfaces

- Block Storage
- Zoned Block Storage
- Object Storage
- Computational Storage
Programming Emerging Storage Interfaces

- Block Storage
- Zoned Block Storage
- Object Storage
- Computational Storage
Programming Emerging Storage Interfaces

- Block Storage
- Zoned Block Storage
- Object Storage
- Computational Storage
 Programming Emerging Storage Interfaces

- Block Storage
- Zoned Block Storage
- Object Storage
- Computational Storage
Host responsibilities for Zoned Block Storage
Programming Emerging Storage Interfaces

- Host responsibilities for Zoned Block Storage
- Setup virtual NVMe devices
Programming Emerging Storage Interfaces

- Host responsibilities for Zoned Block Storage
- Setup virtual NVMe devices
- User Space tools and libraries
Programming Emerging Storage Interfaces

- Host responsibilities for Zoned Block Storage
- Setup virtual NVMe devices
- User Space tools and libraries
- Example library and tool usage
Host Responsibilities
From Open-Channel to ZNS
Host Responsibilities

- Device media represented as
  - Physical blocks (OCSSD 1.2)
  - Chunks (OCSSD 2.0 + Denali)
  - Zones (Zoned Namespaces)
Host Responsibilities

- Device media represented as
  - Physical blocks (OCSSD 1.2)
  - Chunks (OCSSD 2.0 + Denali)
  - Zones (Zoned Namespaces)
- Nomenclature: zone
Host Responsibilities: Zones

- Zone Layout
Host Responsibilities: Zones

- Zone Layout
- Zone Attributes and Condition
Host Responsibilities: Zones

- Zone Layout
- Zone Attributes and Condition
- Zone Constraints
  1. Write contiguously (within a zone)
  2. Reset before write (again)
Host Responsibilities: Zone Layout

- Block Storage
  - How many LBAs
  - Size of an LBA
Host Responsibilities: Zone Layout

- Block Storage
  - How many LBAs
  - Size of an LBA

- Zoned Block Storage
  - How many Zones
  - Attributes and Condition
Host Responsibilities: Zone Attributes

- Write Pointer
- Zone Condition
- Zone Start LBA
- Zone Capacity
- Zone Size
Host Responsibilities: Zone Attributes

WP: zslba + 0

ZC: Empty

Zone Start LBA: Zone Size: 16

Zone Capacity: 16
Host Responsibilities: Zone Write

WP: zslba + 0

ZC: Empty

Zone Start LBA

Zone Capacity: 16

Zone Size: 16
Host Responsibilities: Zone Write

After:
Write(zslba + 0, nlb=3, X)

WP: zslba + 4
ZC: Open
Zone Capacity: 16
Zone Size: 16

Q-Pair
Submission Q
Write(zslba + 0, nlb=3, X)
Completion Q

NVMe Controller
Host Responsibilities: Zone Write

After:
Write(zslba + 0, nlb=3, X)

WP: zslba + 4

ZC: Open

Zone Start
LBA

Zone Capacity: 16

Zone Size: 16

Q-Pair

Submission Q

Completion Q

Write(zslba + 4, nlb=7, Y)

NVMe Controller
Host Responsibilities: Zone Write

After:
Write(zslba + 0, nlb=3, X)
Write(zslba + 4, nlb=7, Y)

WP: zslba + 4
ZC: Open

Zone Start
LBA
Zone Capacity: 16
Zone Size: 16

2019 Storage Developer Conference. © Samsung / SSDR. All Rights Reserved.
Host Responsibilities: Zone Write

After:
Write(zslba + 0, nlb=3, X)
Write(zslba + 4, nlb=7, Y)

WP: zslba + 4
ZC: Open

Zone Start
LBA

Zone Capacity: 16
Zone Size: 16
Host Responsibilities: Zone Write

After:
Write(zslba + 0, nlb=3, X)
Write(zslba + 4, nlb=7, Y)
Write(zslba + 12, nlb=3, Z)

WP: nil

ZC: Full

Zone Start LBA
Zone Capacity: 16
Zone Size: 16
Host Responsibilities: Zone Write QD > 1

WP: zslba + 0

ZC: Empty

Zone Start LBA
Zone Capacity: 16
Zone Size: 16
Host Responsibilities: Zone Write QD > 1

WP: zslba + 0

ZC: Empty

Zone Start LBA

Zone Capacity: 16

Zone Size: 16

Q-Pair

Write(zslba + 12, nlb=3, Z)
Write(zslba + 4, nlb=7, Y)
Write(zslba + 0, nlb=3, X)

Submission Q

Completion Q

NVMe Controller

Write(zslba + 4, nlb=7, Y)
Host Responsibilities: Zone Write QD > 1

After:
Write(zslba + 4, nlb=7, Y)

WP: zslba + 0

ZC: Empty

Zone Capacity: 16
Zone Size: 16

Q-Pair
- Write(zslba + 12, nlb=3, Z)
- Write(zslba + 0, nlb=3, X)
- Write(zslba + 4, nlb=7, Y)
- Submission Q
- Completion Q

NVMe Controller
- Write(zslba + 12, nlb=3, Z)
Host Responsibilities: Zone Write QD > 1

WP: zslba + 0

After:
Write(zslba + 4, nlb=7, Y)
Write(zslba + 12, nlb=3, Z)

ZC: Empty

Zone Start
LBA

Zone Capacity: 16
Zone Size: 16

Q-Pair

Submission Q
Write(zslba + 0, nlb=3, X)
Completion Q
Write(zslba + 12, nlb=5, Z)
Write(zslba + 4, nlb=7, Y)

NVMe Controller
Write(zslba + 0, nlb=3, X)
Host Responsibilities: Zone Write QD > 1

WP: zslba + 4

After:
Write(zslba + 4, nlb=7, Y)
Write(zslba + 12, nlb=3, Z)
Write(zslba + 0, nlb=0, X)

ZC: Open

Zone Start

Zone Capacity: 16

Zone Size: 16

2019 Storage Developer Conference. © Samsung / SSDR. All Rights Reserved.
Host Responsibilities: Zone Write QD $> 1$

WP: ?

ZC: ?

Zone Start LBA

Zone Capacity: 16

Zone Size: 16

Q-Pair

Write(nilba+12, nilb=3, Y)
Write(nilba+4, nilb=7, Y)
Write(nilba+0, nilb=3, X)

Submission Q

Completion Q

2019 Storage Developer Conference. © Samsung / SSDR. All Rights Reserved.
Host Responsibilities: Zone Write QD > 1

Submission Order != Execution / Completion Order

Synchronize:
Wait for a write to WP finish before writing to WP + k
Host Responsibilities: Zone Append

- WP: zslba + 0
- ZC: Empty

Diagram:
- Q-Pair: Submission Q, Completion Q
- NVMe Controller
- Zone Start LBA
- Zone Capacity: 16
- Zone Size: 16
Host Responsibilities: Zone Append

- WP: zslba + 0
- ZC: Empty
- Zone Start LBA
- Zone Capacity: 16
- Zone Size: 16
Host Responsibilities: Zone Append

WP: zslba + 4

ZC: Open

Zone Start LBA

Zone Capacity: 16

Zone Size: 16

After: Append(zslba, nlb=3, X)

Q-Pair

Submission Q

Completion Q

Append(zslba, nlb=3, X)

NVMe Controller
Host Responsibilities: Zone Append

After:
Append(zslba, nlb=3, X)

WP: zslba + 4

ZC: Open

Zone Start
LBA

Zone Capacity: 16

Zone Size: 16

Q-Pair

Append(zslba, nlb=7, Y)
Submission Q
Completion Q

NVMe Controller

2019 Storage Developer Conference. © Samsung / SSDR. All Rights Reserved.
Host Responsibilities: Zone Append

After:
- Append(zslba, nlb=3, X)
- Append(zslba, nlb=7, Y)

WP: zslba + 4
ZC: Open

Zone Start LBA
Zone Capacity: 16
Zone Size: 16

Q-Pair
- Submission Q
- Append(zslba, nlb=7, Y)
- Completion Q

NVMe Controller
Host Responsibilities: Zone Append

After:
Append(zslba, n1b=3, X)
Append(zslba, n1b=7, Y)

WP: zslba + 4
ZC: Open

Zone Start LBA
Zone Capacity: 16
Zone Size: 16

Q-Pair
Append(zslba, n1b=3, Z)
Submission Q
Completion Q

NVMe Controller
Host Responsibilities: Zone Append

After:
Append(zs|lba, nlb=3, X)
Append(zs|lba, nlb=7, Y)
Append(zs|lba, nlb=3, Z)

WP: nil

ZC: Full

Zone Capacity: 16
Zone Size: 16
Host Responsibilities: Append QD > 1

WP: nil

ZC: Full

Zone Start LBA
Zone Capacity: 16
Zone Size: 16
Host Responsibilities: Append QD > 1

Where is my data?
Host Responsibilities: Append QD > 1

Where is my data?
Read the completion entry for location

Response (cq-entry)
Host Responsibilities: Zone Reset

After:
Append/Write(nlb=3, X)
Append/Write(nlb=7, Y)
Append/Write(nlb=3, z)

WP: nil

ZC: Full

Zone Start
LBA

Zone Capacity: 16

Zone Size: 16
Host Responsibilities: Zone Reset

WP: zslba + 0

After:
Append/Write(nlb=3, X)
Append/Write(nlb=7, Y)
Append/Write(nlb=3, z)
Reset(zslba)

ZC: Empty

Zone Start
LBA

Zone Capacity: 12

Zone Size: 16
Host Responsibilities

- How to manage this as storage developer?
- You need to get retrieve information on
  - Zone Layout
  - Zone Attributes and Condition
- Read, Write, Append, and Reset commands
User Space tools and libraries
Open-Source Ecosystem: OS Support

- UNIX-like OS: everything is a file
Open-Source Ecosystem: OS Support

- UNIX-like OS: everything is a file
- System Calls
  - `ioctl()`, `read()`, `write()`, `pread()`, `pwrite()`, etc.
  - `aio_read()`, `aio_write()`
  - `io_uring_prep_readv` / `io_uring_submit`
Open-Source Ecosystem: OS Support

- UNIX-like OS: everything is a file
- System Calls
  - ioctl(), read(), write(), pread(), pwrite(), etc.
  - aio_read(), aio_write()
  - io_uring_prep_readv / io_uring_submit
- Wrapped in libraries libc, libaio, liburing
Open-Source Ecosystem: OS Support

- **Pros:**
  - General block storage infrastructure
  - Efficient async R/W with `io_uring` / `liburing`

- **Cons:**
  - Synchronous `ioctl()` interface
  - Limited control over command construction
Open-Source Ecosystem: OS Bypass

- API: Everything is a function call to opaque*
Open-Source Ecosystem: OS Bypass

- API: Everything is a function call to opaque*
- Driver in User Space
  - Intel SPDK
  - libnvme (SPDK without DPDK)
Open-Source Ecosystem: OS Bypass

- API: Everything is a function call to opaque*
- Driver in User Space
  - Intel SPDK
  - libnvme (SPDK without DPDK)
- Driver in Kernel access from User Space
  - NVMe-Direct
Open-Source Ecosystem: OS Bypass

- **Pros:**
  - Full control over command construction
  - Efficient async interface for ANY command

- **Cons:**
  - Controller detachment from kernel
  - Non-trivial controller sharing

[Demo detach]
Open-Source Ecosystem: tools

- nvme-cli
  - Built on Linux ioctl()
  - Limited port for FreeBSD ioctl()
  - Port built on SPDK
- nvmecontrol
  - FreeBSD base system, built on ioctl()
Virtual NVMe Devices
Setup, usage, and modification with QEMU
QEMU: The Quick Emulator

- A generic machine emulator and virtualizer
  - ia32, x86_64, mips, sparc, arm, risc-v
  - KVM-client e.g. using Intel VT-x
- Includes a huge collection of emulated devices
QEMU: The Quick Emulator

- A generic machine emulator and virtualizer
  - ia32, x86_64, mips, sparc, arm, risc-v
  - KVM-client e.g. using Intel VT-x
- Includes a huge collection of emulated devices
- Active community
  - ~130 subsys maintainers
  - ~1500 individual contributors)
QEMU: Contributions

- Upstream contributions (Klaus A. B. Jensen)
  - Full NVMe 1.3 support
  - Ongoing NVMe 1.4 support
  - Full ZNS support
  - Upcoming TPs
QEMU: Contributions

- Upstream contributions (Klaus A. B. Jensen)
  - Full NVMe 1.3 support
  - Ongoing NVMe 1.4 support
  - Full ZNS support
  - Upcoming TPs
- Extending the QEMU NVMe drive model
QEMU: NVMe Device Model

- QEMU/hw/block/nvme.c
QEMU: NVMe Device Model Usage

- NVMe Controller 0
  - NVMe Namespace Conventional (CNS) nvme0n1
  - NVMe Namespace Zoned (ZNS) nvme0n2
  - NVMe Namespace Your Thing (??) nvme0n3

- NVMe Controller 1
  - NVMe Namespace Conventional (CNS) nvme1n1
QEMU: NVMe Usage Hands On!

- QEMU configure and build:
  - QENV setup
    - environment config
    - machine config
    - device config
    - run and access

QEMU build demo link

```bash
mkdir build
cd build
../configure
   --target-list=x86_64-softmmu
make -j $(nproc)
```

QENV setup demo link
User Space tools and libraries
Open-Source Ecosystem: an overview

Linux Kernel NVMe Driver

SPDK NVMe Driver

FreeBSD Kernel NVMe Driver
Open-Source Ecosystem: an overview

File-systems

Block Device

syscalls()

Linux Kernel NVMe Driver

SPDK NVMe Driver

FreeBSD Kernel NVMe Driver

API
Open-Source Ecosystem: an overview
Open-Source Ecosystem: a contribution

Diagram showing the xNVM backend and its interactions with various drivers and APIs:
- xNVM API
- xnvme_buf
- xnvme_cmd
- xnvme_async
- xnvme_dev

File-systems:
- Block Device
  - syscall(s)
- API
- File-systems
  - Block Device
  - syscall(s)

Drivers:
- Linux Kernel NVMe Driver
- SPDK NVMe Driver
- FreeBSD Kernel NVMe Driver
xNVMe
Cross-platform libraries and tools for NVMe devices
xNVMe: API: xnvme_dev

Abstract handle to your controller / namespace

- **Linux Backend**
  - Open FDs for NVMe controller and namespace
  - `io_uring_register(., IORING_REGISTER_REGISTER_FILES, )`
  - Reduce overhead of kernel retrieving handles for each IO

- **SPDK Backend**
  - Initialize and attach to controller
  - `spdk_env_opts_init() / spdk_env_init() / spdk_nvme_probe()`
Allocate and free memory for use by the `xnvme_cmd` interface

- **Linux Backend**
  - Pagesize aligned for `ioctl()`
  - `posix_memalign()` / `free()`
  - `io_uring_register(IORING_REGISTER_BUFFERS, ...)`

- **SPDK Backend**
  - Allocate physical memory / DMA transferable
  - `spdk_dma_{malloc,realloc,free}()`
xNVMe: API: xnvme_async

Context for asynchronous / non-blocking xnvme_cmd interface

- Linux Backend
  - Threadpool allocation for pseudo-async behavior via ioctl()
  - SQ / CQ setup for io_uring
- SPDK Backend
  - NVMe QP setup
  - spdk_nvme_ctrlr_{alloc,free}_io_qpair()
xNVMe: API: xnvme_cmd

Synchronous and asynchronous command interface

- Linux Backend
  - io_uring_{submit, peek, wait}
  - Jobs to ioctl() threadpool
- SPDK Backend
  - spdk_nvme_ctrlr_cmd_{admin_raw, io_raw_with_md}()
  - spdk_nvme_qpair_process_completions()
Open-Source Ecosystem: a contribution
Open-Source Ecosystem: a contribution

![Diagram of the Open-Source Ecosystem for NVMe]

- xNVMe API
- xnvme_buf
- xnvme_cmd
- xnvme_async
- xnvme_dev

xNVMe backend

- File-systems
- Block Device
- syscalls()

- Linux Kernel NVMe Driver
- SPDK NVMe Driver
- FreeBSD Kernel NVMe Driver
Open-Source Ecosystem: a contribution
Open-Source Ecosystem: a contribution
Usage of `zoned` Demo link
Open-Source Ecosystem: a contribution
Open-Source Ecosystem: a contribution
Usage of `ZROFS`

Demo link
Open-Source Ecosystem: a contribution
Open-Source Ecosystem: a contribution
Usage of `xnvme`

Demo link
Open-Source Ecosystem: a contribution
Building Open-Source ecosystem
- Building Open-Source ecosystem
- Cross-platform for existing and emerging storage interfaces
- Building Open-Source ecosystem
- Cross-platform for existing and emerging storage interfaces
- Tools
- Building Open-Source ecosystem
- Cross-platform for existing and emerging storage interfaces
- Tools
- Libraries
- Building Open-Source ecosystem
- Cross-platform for existing and emerging storage interfaces
- Tools
- Libraries
- VALUE
Thanks

WWW https://xnvme.io
MAIL simon.lund@samsung.com

www.linkedin.com/in/simonlund