Squeezing Compression into SPDK

Paul Luse, Jim Harris
Intel
Agenda

- High Level Architecture
- DPDK Library Overview
- Crypto Bdev Module
- Compression Bdev Module
- Introducing “reduce”
High Level Architecture

NVMeOF
Target

NVMe

Block Device Abstraction Layer (bdev)

Virtual bdevs

DPDK Encryption

DPDK Compression

PMDK

reducelib

NVME

env

DPDK

In Progress
# DPDK Libraries

## Core and feature libs
- **Core libraries**
  - Core functions such as memory management, software rings, timers, bus/device mgmt, etc.

## Device APIs
- **ETHDEV**
- **CRYPTODEV**
- **EVENTDEV**
- **SECURITY**
- **COMPRESSDEV**
- **BBDEV**

## Device PMDs
- **PMDs for physical and virtual Ethernet devices**
- **PMDs for HW and SW crypto accelerators**
- **Event-driven PMDs**
- **Hardware acceleration APIs for security protocols**
- **PMDs for HW and SW compression accelerators**
- **PMDs for HW and SW wireless accelerators**

## Packet classification
- Software libraries for hash/exact match, LPM, ACL etc.

## Accelerated SW libraries
- Common functions such as IP fragmentation, reassembly, reordering etc.

## Stats
- Libraries for collecting and reporting statistics.

## QoS
- Libraries for QoS scheduling and metering/policing.

## Packet Framework
- Libraries for creating complex pipelines in software.
Crypto Bdev Overview

1. Application
   <SPDK Defined Block API>

2. Block Device Layer
   VBDEV
   VBDEV
   BDEV
   Crypto
   BDEV

3. <Driver Specific API>

4. DPDK
   CryptoDev API
   AESNI PMD
   QAT PMD
   QAT HW

Drivers

I/O

Crypto Operations

2019 Storage Developer Conference. © Intel Corporation. All Rights Reserved.
Compression Bdev Overview

1. Application
   - SDDK Defined Block API
   - User I/O

2. Block Device Layer
   - VBDEV
   - libReduce
   - PMDK
   - Compression
   - BDEV

3. DPDK
   - compressdev API
   - ISA-L PMD
   - QAT PMD

4. Drivers
   - QAT HW

Metadata
Compression Operations
Compressed Data

2019 Storage Developer Conference. © Intel Corporation. All Rights Reserved.
Libreduce
Libreduce Overview

Block device for backing I/O units
- Typically thin-provisioned SPDK logical volume

Persistent memory file for mapping metadata
- Uses PMDK directly for persistent memory access

Metadata on block device
- Libreduce parameters
- Path to persistent memory file

Metadata algorithm only!
- Uses standard compression algorithms
Integration

**libreduce**

- `spdk_reduce_vol_init()`
- `spdk_reduce_vol_load()`
- `spdk_reduce_vol_readv()`
- `spdk_reduce_vol_writev()`

**PMDK**
- `pmem_persist()`

**Interaction:**
- `readv/writev/unmap` (to backing device)
- `compress/decompress` (to SW/accelerator)

**Independent from SPDK framework and bdev layer**

**Caller ensures I/Os do not cross chunk boundary**

**Single-threaded** (per compression volume)
### Backing Device
- Split into I/O units

### Persistent Memory File
- Metadata based on chunks
- Chunk map contains chunk entries
- Chunk entry maps a logical chunk to its I/O units on disk
- Logical map contains logical map entries
- Logical map entries map a logical offset to its chunk entry
Write Offset 4KB at Offset 0KB

Logical Map: Lookup 0KB => empty
Allocate chunk entry in chunk map => 0
Compress chunk data
  ▪ 4KB user data + 12KB zeroes
  ▪ Compresses to 2500 bytes
Allocate 1 backing I/O unit => 0
Write compressed data to SSD
Write and persist chunk entry
Write and persist logical map entry
Write Offset 16KB at Offset 64KB

Logical Map: Lookup 16KB => empty
Allocate chunk entry in chunk map => 1
Compress chunk data
  - 16KB user data
  - Compresses to 14000 bytes
Allocate 4 backing I/O unit => 1, 2, 3, 4
Write uncompressed data to SSD
  - 14000 bytes requires 4 4KB I/O units
Write and persist chunk entry
Write and persist logical map entry
Write Offset 4KB at Offset 4KB

Logical Map: Lookup 0KB => 0
Read I/O unit 0
Decompress 2500B => 16KB
Merge incoming 4KB
Allocate chunk entry in chunk map => 2
Compress chunk data => 5000B
Allocate 2 backing I/O unit => 5, 6
Write compressed data to SSD
Write and persist chunk entry
Write and persist logical map entry
Release old chunk entry and I/O units
Trim 16KB at Offset 64KB

Logical Map: Lookup 64KB => 1

Clear and persist logical map entry

Release old chunk entry and I/O units
Read 4KB at Offset 4KB

Logical Map: Lookup 0KB => 2
Read I/O units 5 and 6
Decompress 5000B => 16KB
  • Target user buffer for 4KB
  • Bit bucket for remaining 12KB

<table>
<thead>
<tr>
<th>Logical Map</th>
<th>Chunk Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>0KB 16KB 32KB 48KB 64KB 80KB 96KB 112KB</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>{ 5, 6 }, 2500</td>
</tr>
</tbody>
</table>

Persistent Memory
SSD

Backing I/O Units
Next Steps

spdk\_reduce\_vol\_unmap

Sub-chunk allocation masks

Performance data with persistent memory!

Additional on-disk metadata parameters (i.e. compression algorithm)

Method for reduced metadata file size

32-bit io\_unit/chunk indices (instead of 64-bit)
For More Information

- Main Website: https://spdk.io/
- Crypto & Compression vbdev Module Documentation: https://spdk.io/doc/bdev.html
- Libreduce Documentation: https://spdk.io/doc/reduce.html

Upcoming SPDK Developer Meetup: