End-to-End Data Protection with SPDK

04/16/2019

Shuhei Matsumoto
SPDK Core Maintainer
IT Platform Products Management Division
Hitachi, Ltd.
Contents

1. Introduction
2. DIF Support in SPDK
3. Performance Evaluation
4. Summary and Next Steps
1. Introduction
End-to-End Data Protection

- Data corruption can occur anywhere and silent data corruption must be avoided.
- Data Integrity Field (DIF) provides a standardized end-to-end data protection mechanism that spans transport and protocol boundaries.

Create DIF with data creation  ➔  Check DIF through data path  ➔  Store data and DIF into storage media

- 8 byte DIF is associated with each data block.
  - Guard (GRD) tag contains a CRC of the data block.
  - Application (APP) tag is up to application.
  - Reference (REF) tag normally contains the lower 4 bytes of the associated Logical Block Address.

<table>
<thead>
<tr>
<th>Data Block</th>
<th>Metadata</th>
<th>GRD</th>
<th>APP</th>
<th>REF</th>
</tr>
</thead>
</table>

- There are many variables about DIF, metadata format, DIF settings, and DIF check types.
End-to-End Data Protection with Storage Arrays

- Dealing with data corruption is important for storage arrays and storage arrays have used DIF extensively.
- For some hosts that are not aware of DIF, iSCSI/FC HBAs have inserted/stripped DIF for write/read I/O.
- For some hosts that are aware of DIF, iSCSI/FC HBAs pass data with DIF for read/write I/O.
Use Cases of End-to-End Data Protection with SPDK

• It will be beneficial for storage applications if SPDK support DIF with compelling performance and efficiency without specialized hardware.

• DIF insert/strip feature in SPDK iSCSI target.

• DIF passthrough feature in SPDK NVMe-oF target
End-to-End Data Protection with SPDK 19.04

Storage Protocols
- iSCSI Target
- SCSI
- NVMe-oF Target

Storage Services
- Bdev
- NVMe
- Customized

Drivers
- NVMe Devices
  - NVMe-oF Initiator
  - NVMe PCIe Driver

Integration
- Utilities
  - DIF

Tool
- Intel® ISA-L
- Perf and FIO

DIF Support in SPDK 18.10
- DIF Support since SPDK 18.10
End-to-End Data Protection with SPDK 19.04 (Continued)

<table>
<thead>
<tr>
<th>Component</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPDK DIF Library</td>
<td>Provide routines to simplify integration of DIF operations into other SPDK libraries.</td>
</tr>
<tr>
<td>Intel ® Intelligent Storage Acceleration Library (ISA-L)</td>
<td>ISA-L is integrated with SPDK.</td>
</tr>
<tr>
<td>SPDK Bdev Layer</td>
<td>Expose DIF setting to upper layers.</td>
</tr>
<tr>
<td>SPDK NVMe Bdev Module</td>
<td>Find the location of DIF errors.</td>
</tr>
<tr>
<td>SPDK SCSI Layer</td>
<td>Return DIF context for SCSI read/write commands.</td>
</tr>
<tr>
<td>SPDK iSCSI Target</td>
<td>Support DIF insert and strip feature.</td>
</tr>
<tr>
<td>SPDK NVMe-oF Target</td>
<td>Support DIF passthrough feature.</td>
</tr>
<tr>
<td>SPDK NVMe Perf Tool and FIO Plugin</td>
<td>Support all DIF and DIX features.</td>
</tr>
</tbody>
</table>
2. DIF Support in SPDK
Byte Alignment and Granularity of Logical Blocks

- SPDK DIF library supports byte alignment and granularity for data payload.

![Logical Blocks Diagram]

- To ensure quality by unit tests, the following are used:
  - Fault Injection which can inject bit flip error into any field and offset.
  - Use cyclic values to set in the test data buffer.
Find the Data Block at which DIF Error was Detected

- SPDK NVMe bdev modules leaves DIF checks to NVMe controller but NVMe controller doesn’t report the location of DIF error. Hence SPDK NVMe bdev module finds the location of DIF error instead.

Read I/O with DIF checks.

DIF error?

- Yes: Read I/O without DIF checks
  - Find the corrupted data block from read data buffer
    - End

- No: Read I/O without DIF checks

Write I/O with DIF checks

DIF error?

- Yes: Find the corrupted data block from write data buffer
  - End

- No: End
iSCSI Read and Write Sequence

**iSCSI Read**

- **iSCSI Initiator**
  - SCSI Read Cmd
  - Recv Data
  - Recv Data
  - Recv Data
  - Cmd. Complete

- **iSCSI Target**
  - Data In PDU
  - Data In PDU
  - Data In PDU
  - Send Status

**iSCSI Write**

- **iSCSI Initiator**
  - SCSI Write Cmd
  - Data Out PDU
  - R2T
  - R2T

- **iSCSI Target**
  - Data Out PDU
  - Recv Data
  - Send Status
  - Send Status

- **iSCSI Target**
  - R2T
  - Recv Data
  - Send Status
  - Send Status
**Query DIF Context to Insert/Strip DIF to/from iSCSI PDU**

**SPDK iSCSI Target**
- Filter out only PDUs which has data segment
- Get CDB and LUN ID in PDU
- Set buffer offset to 0
- Get buffer offset in PDU
- Query DIF context (LUN ID, CDB, offset)

**SPDK SCSI Layer**
- Get start LBA from CDB and buffer offset
- Get other DIF context from the backing bdev

**Steps**
1. Filter out only PDUs which has data segment
2. Get CDB and LUN ID in PDU
3. Set buffer offset to 0
4. Get buffer offset in PDU
5. Get start LBA from CDB and buffer offset
6. Get other DIF context from the backing bdev
7. Use DIF context

**Notes**
- SPDK iSCSI Target
- SPDK SCSI Layer
SPDK iSCSI target avoids extra data copy and any bounce buffer by using the special scatter-gather list.
Read the Incoming Write Data with DIF Insertion

1. Read iSCSI Request PDU header by TCP/IP stack
2. Allocate write data buffer
3. All write data is read?
   - Yes
     - Process iSCSI Request PDU
   - No
     - Create scatter list to leave DIF spaces
     - Read the incoming write data by TCP/IP stack
     - Generate and insert DIF into DIF spaces
     - Process iSCSI Request PDU
• Incoming write data may be split into multiple TCP packets.
• iSCSI target adjusts scatter-list before every read.
• If the newly read starts from the unaligned offset, DIF insertion steps back to the aligned offset and generates and inserts DIF to full data blocks.
Write the Outgoing Read Data with DIF Strip

1. Create iSCSI Response PDU
2. Verify DIF in read data?
   - Yes: All read data is written?
     - Yes: Process data error
     - No: Create scatter list to ignore DIF space
3. Write the read data by TCP/IP stack
4. Complete iSCSI Response PDU
3. Performance Evaluation
System Configuration

- Use two Xeon processor servers
- SPDK iSCSI target used
  - a single CPU core.
  - Linux kernel TCP/IP stack.
  - a single NVMe SSD which supports SGL.
- iSCSI initiator used
  - Linux kernel iSCSI initiator.
- Use FIO 3.3 and create 8 iSCSI sessions on a single 10G link.
- Both 10G NICs set MTU to 9000.
4KB Random Write

**Configuration** | **Throughput (KIOPS)** | **Overhead**
--- | --- | ---
SPDK 19.01 + (NVMe format = 512) | 153 | -
SPDK 19.04 pre + (NVMe format = 512) | 152 | 0%
SPDK 19.04 pre + (NVMe format = 512 + 8) + (DIF check = disable) | 140 | 9.2%
SPDK 19.04 pre + (NVMe format = 512 + 8) + (DIF check = enable) | 136 | 12.5%
### 4KB Random Read

#### Throughput (KIOPS)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Throughput (KIOPS)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPDK 19.01 + (NVMe format = 512)</td>
<td>208</td>
<td>-</td>
</tr>
<tr>
<td>SPDK 19.04 pre + (NVMe format = 512)</td>
<td>209</td>
<td>0%</td>
</tr>
<tr>
<td>SPDK 19.04 pre + (NVMe format = 512 + 8) + (DIF check = disable)</td>
<td>194</td>
<td>7.2%</td>
</tr>
<tr>
<td>SPDK 19.04 pre + (NVMe format = 512 + 8) + (DIF check = enable)</td>
<td>185</td>
<td>12.4%</td>
</tr>
</tbody>
</table>
4. Summary and Next Steps
Summary and Next Steps

Summary

• SPDK iSCSI target provides DIF insert and strip feature without specialized hardware in SPDK 19.04.
• Performance evaluation showed that the overhead were a little more than 10% both for 4KB random read and write.

Next Steps

• Performance Evaluation with Vector Packet Processing (VPP)
• Support DIF in bdev modules (e.g. crypto, compress, and RAID)
• Support DIX (separate metadata) in bdev modules
• Support DIF insert and strip feature in NVMe-TCP target.
End-to-End Data Protection with SPDK

04/16/2019

Shuhei Matsumoto
SPDK Core Maintainer
IT Platform Products Management Division
Hitachi, Ltd.
Inspire the Next