Hardware Accelerated Data Integrity Check on a CSD

Vinit Vyas, Solidigm
Michael Mesnier, Intel Labs
Disclaimers

All product plans, roadmaps, specifications, and product descriptions are subject to change without notice.

Nothing herein is intended to create any express or implied warranty, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, or any warranty arising from course of performance, course of dealing, or usage in trade.

Contact your Solidigm representative or your distributor to obtain the latest specifications before placing your product order.

For copies of this document, documents that are referenced within, or other Solidigm literature, please contact your Solidigm representative.

All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

© Solidigm. “Solidigm” is a trademark of SK hynix NAND Product Solutions Corp (d/b/a Solidigm). “Intel” is a registered trademark of Intel Corporation. Other names and brands may be claimed as the property of others.

Some results have been estimated or simulated using internal Solidigm analysis or architecture simulation or modeling, and provided to you for information purposes only. Any differences in your system hardware, software or configuration may affect your actual performance.
Outline

- Use case description
- Why use CS?
- Software stack
- CS implementation
- Distributed processing and scalability
- Future work
- Conclusion
Use Case and Problem Statement

Using computational storage for expensive data integrity checks

Data integrity check is:
- Compute intensive
- Read intensive (SSD & PCIe bus)
- Memory intensive (host)
- Not scalable
Why Use CS For This Use Case?

- Off-load the host
  - The host is only interested in the data integrity check results
- Reduces PCIe traffic
  - No need to consume bandwidth and power to move the raw data to the host
- Reduces host memory footprint
  - All data required for processing is contained in the drive
- Scalable with storage
  - Performance increases as drives are added
Software Stack

- Application Layer
- Object Storage (MINIO, CEPH, SWIFT, ...)
- Filesystem Layer (ext3, ext4, xfs, ...)
- Device Drivers

File: "vid.mpg"

Shard 1 | Shard 2 | Shard 3
---|---|---
0 | 1 | 2 | 3
4 | 5 | 6 | 7
8 | 9 | 10 | 11
12 | 13 | 14 | 15

LBA [list]

Vendor Unique (VU)
Read Cmd with LBA [list]

NVMe TCP/IP

CSD

• Data integrity validation of the object shard is decoupled from data transfer.
• Data integrity hash calculation is done by the CSD.
• The object storage node validates the result.
CSD Implementation

1: VU Read Cmd with LBA [list]

2: NAND Read Cmd: LBA [list]

3: Read from NAND to buffer

4: Data integrity acc. on NAND data cmd

5: Data integrity result transfer

6a: Data integrity result added to VU cmd completion

6b: Deallocate buffer

File: vid.mpg

LBA [list]

Size 10 MB

Application Layer

Object Storage (MINIO, CEPH, SWIFT, ...)

Filesystem Layer (ext3, ext4, xfs, ...)

Device Drivers

NVMe plane

NVMe TCP/IP

Vendor Unique (VU) Read Cmd with LBA [list]

Filesystem Layer (ext3, ext4, xfs, ...)

Device Drivers

NVMe TCP/IP

Application Layer

Object Storage (MINIO, CEPH, SWIFT, ...)

File: vid.mpg

LBA [list]

Size 10 MB

NVMe plane

NVMe TCP/IP

Vendor Unique (VU) Read Cmd with LBA [list]
CSD Implementation

- Gen5 off-the-shelf product
- Single ASIC controller
  - Low cost
  - High energy efficiency
  - High performance
- Off-the-shelf NVMe driver
- Ready to support TP4091 & TP4131
Performance and Scalability (1)

Presented by Intel Labs at FMS 2022
Performance and Scalability (2)

Modeling results indicate a high degree of scalability ideal for CPU offload*

*Based on Solidigm's internal analysis. CSD theoretical performance based upon modeling.
Future Work

- **Align with NVMe TP4091**
  - Enhance implementation to leverage the Computational Programs Command Set
  - TP4091 commands can activate and execute the data integrity check

- **Align with SNIA CS API**
  - Leverage the SNIA CS API to standardize the user library

- **Introduce dynamic resource allocation**
  - Leverage existing FW architecture and CSD programming model
Dynamic Compute Resource Allocation

The diagram illustrates the process of dynamic compute resource allocation. It shows the flow of data from the host, through NVMe Admin Cmd (host hint), to the Resource Allocation module. This module distributes resources to CS Resource Factor and SSD Resource Factor. The Resource Adjudicator (FW/OS) then decides on resource allocation.

SSD FW is connected to the SSD Resource Factor, and CSD Compute Resources Throttling is shown with a gauge indicating resource allocation. The buffer is involved in data integrity checking and result transfer. SSD FW includes HW acceleration for buffer and compute operations.

The process includes:
1. VU Read Cmd
2. NAND Read Cmd: LBA [list]
3. Read from NAND to buffer
4. Data integrity acc. on NAND data cmd
5. Data integrity result transfer
6a. Data integrity result added to VU cmd completion
6b. Deallocate buffer

The diagram also highlights the NVMe plane, which integrates with the various components to facilitate efficient resource allocation and management.
Conclusion

• Computational Storage is ideal for processing meta-data tasks on-drive
  ▪ Utilizes existing HW accelerators and SW solutions, no ‘new’ work required
  ▪ Operates on SHARDED data
    ▪ **Major value add to customer’s concerns of data locality**
  ▪ Scales across multiple CSDs
    ▪ **Works independently, but brings overall increased performance to system**
    ▪ Our PoC demonstrates linear scaling performance with additional drives

• This use-case is adaptable to the latest Computational Storage standards
  ▪ Can become fully compliant with TP4091/4131 and Architectural and API specs
  ▪ Does not restrict Host from using resources for other Computational work on drive
Please take a moment to rate this session.

Your feedback is important to us.