SPDK based user space NVMe over TCP transport solution

Ziye Yang
ziye.yang@intel.com
DCG, Intel
Notices and Disclaimers

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Performance results are based on testing as of February 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product or computer system can be absolutely secure. For more complete information visit http://www.intel.com/benchmarks.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.
© 2019 Intel Corporation.
Agenda

- Why NVMe-oF & SPDK solution?
- SPDK NVMe-oF development status
- SPDK NVMe-oF TCP transport introduction
- Conclusion
Why NVMe-oF & SPDK solution?
Why NVMe-oF fabrics

- NVMe over Fabrics project Goals: **Simplicity**, **Efficiency**, and **End-to-End** NVMe model
  - Scale NVMe remotely, beyond limitation of local PCIe
  - Provide remote NVMe performance equivalent to local PCIe
    - Take advantage of inherent parallelism of deep multi-Q model (64 commands/queue, up to 64K queues)
    - Avoid translation to or from other protocols (e.g., SCSI)
    - NVMe commands and structures are transferred end-to-end
  - Maintain architecture and software consistency between fabric types by standardizing a **common abstraction** and **encapsulation definition**.
- Storage computing disaggregation VS Storage computing aggregation
Why SPDK based solution?

What is SPDK (storage performance development kit: http://spdk.io)

- User Space, High Performance, Scalable Library
- End-to-End storage accelerated Solution
- Extensible Framework and Component

SPDK can explore the ability of CPU/NVMe SSDs/NICs to improve the performance of applications with low cost
# Why SPDK based user space NVMe-oF solution?

<table>
<thead>
<tr>
<th>Ingredients</th>
<th>Linux Kernel</th>
<th>SPDK solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment</strong></td>
<td>Need to upgrade the kernel or backporting (Most cloud storage providers' kernel is old)</td>
<td>Easy to deploy (Because it is in user space)</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Currently, worse than SPDK NVMe-oF solution in per CPU core aspect</td>
<td>Currently Better than Linux kernel solution in IOPS and latency in per CPU core aspect</td>
</tr>
<tr>
<td><strong>Spec compliance and functionality</strong></td>
<td>Follow NVMe-oF spec</td>
<td>Follow NVMe-oF spec</td>
</tr>
<tr>
<td><strong>Service recovery</strong></td>
<td>Reload the kernel module or reboot kernel</td>
<td>Restart SPDK NVMe-oF target application</td>
</tr>
<tr>
<td><strong>Further development challenge for programmers</strong></td>
<td>Need to development the kernel module, difficulty (****)</td>
<td>Need to develop based on SPDK framework, difficulty (***)</td>
</tr>
</tbody>
</table>
SPDK NVMe-oF development status?
SPDK NVMe-oF target timeline

- **July 2016:** Released with RDMA transport support
- **17.03 – 17.07:** Functional hardening
- **17.11 – 18.11:** RDMA transport improvements
- **19.01:** TCP transport released
- **19.04 – now:** Continuing improvement

© Intel Corporation. All Rights Reserved.
SPDK NVMe-oF host timeline

19.04-NOW: CONTINUING IMPROVEMENT

19.01: TCP TRANSPORT RELEASED

17.11-18.11: RDMA TRANSPORT IMPROVEMENTS

17.03-17.07: FUNCTIONAL HARDENING
(E.G., INTEROPERABILITY TEST WITH KERNEL TARGET)

DEC 2016: RELEASED WITH RDMA TRANSPORT SUPPORT
## SPDK NVMe-oF target design highlights

<table>
<thead>
<tr>
<th>NVMe* over Fabrics Target Features</th>
<th>Performance Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilizes user space NVM Express* (NVMe) Polled Mode Driver</td>
<td>Reduced overhead per NVMe I/O</td>
</tr>
<tr>
<td>Group polling on each SPDK thread (binding on CPU core) for multiple transports</td>
<td>Efficient Parallelism among different CPU cores</td>
</tr>
<tr>
<td>Each connection pinned to dedicated SPDK thread</td>
<td>No synchronization overhead among connections.</td>
</tr>
<tr>
<td>Asynchronous NVMe CMD handling in whole life cycle</td>
<td>No locks in NVMe CMD data handling path</td>
</tr>
</tbody>
</table>
General design and implementation

Transport Abstraction

- FC
- RDMA
- TCP

POSIX
VPP
Seastar

Merged, code is available after 19.07 release
Already released
Integrated, but performance tuning is still needed
In investigate phase
SPDK NVMe-oF TCP transport introduction
## Comparison between TCP and RDMA Transport?

<table>
<thead>
<tr>
<th>Ingredients</th>
<th>RDMA transport</th>
<th>TCP transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>High performance</td>
<td>Low performance</td>
</tr>
<tr>
<td>Hardware dependency</td>
<td>RDMA capable NICs (supporting RoceV2 or iwarp)</td>
<td>No dependency on NICs (It can reuse the existing NICs in data center)</td>
</tr>
<tr>
<td>Physical distance restriction</td>
<td>May only be suitable in a small data center</td>
<td>No distance requirements (suitable for general cloud storage interface)</td>
</tr>
<tr>
<td>Programming skills requirements</td>
<td>Must be familiar with RDMA related primitives. (High requirements for developers for debugging)</td>
<td>Relatively easy to debug (Since most programmers are familiar with TCP)</td>
</tr>
<tr>
<td>Usage scope</td>
<td>Especially for high performance usage</td>
<td>Can be applied into any usage scenarios due to its flexibility</td>
</tr>
</tbody>
</table>
## Performance design consideration for TCP transport on target side

<table>
<thead>
<tr>
<th>Ingredients</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design framework</td>
<td>Follow the general SPDK NVMe-oF framework (e.g., polling group)</td>
</tr>
<tr>
<td>TCP connection optimization</td>
<td>Use the SPDK encapsulated Socket API (preparing for integrating other stack, e.g., VPP)</td>
</tr>
<tr>
<td>NVMe/TCP PDU handling</td>
<td>Use state machine to track</td>
</tr>
<tr>
<td>NVMe/TCP request life time cycle</td>
<td>Use state machine to track (Purpose: Easy to debug and good for further performance improvement)</td>
</tr>
</tbody>
</table>
TCP PDU Receiving handling for each connection

```c
enum nvme_tcp_pdu_recv_state {
    /* Ready to wait PDU */
    NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_READY,

    /* Active tpair waiting for any PDU common header */
    NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_CH,

    /* Active tpair waiting for any PDU specific header */
    NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_PSH,

    /* Active tpair waiting for payload */
    NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_PAYLOAD,

    /* Active tpair does not wait for payload */
    NVME_TCP_PDU_RECV_STATE_ERROR,
};
```
Error Path

New → Need buffer?
  - Not a valid cmd
  - Read cmd?

Ready to Execute in NVMe
  - Executing in NVMe driver

Transfer data from Host (get data for write cmd)
  - Executed in NVMe driver

Completed

Transfer Response PDU to host

Free

Recycling resource

Get data error

Write cmd (Sent R2T if needed)
SPDK TCP transport target side
I/O scaling

System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/andread/randwrite/randrw
SPDK TCP transport host side
I/O scaling

System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600x P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.01.1. (3): Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw
Latency comparison between SPDK and Kernel (NULL block dev is used)

System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rdwr/rdwrw
Latency comparison between SPDK and Kernel (NULL block dev is used)

System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) : Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw

2019 Storage Developer Conference. © Intel Corporation . All Rights Reserved.
IOPS/CPU core comparison between SPDK and kernel on target side

System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U-TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.01.1. (3) Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw
SPDK NVMe/TCP transport: Further development plan

- Continue enhancing the functionality
  - Including the compatible test with Linux kernel solution.

- Performance tuning
  - Integration with third party software
    - Deep integration with user space stack, e.g., VPP/Seastar + DPDK, need the stability and performance tuning.
  - Leveraging hardware features
    - Use existing hardware features of NICs for performance improvement, e.g., VMA from Mellanox’s NIC; ADQ from Intel’s 100Gbit NIC.
    - Figuring out offloading methods with hardware, e.g., FPGA, Smart NIC, and etc.
To integrate ADQ feature of Intel 800 E800 series NIC into SPDK

- Try to Leverage the ADQ (application device queue) feature of Intel’s NIC (i.e., E810 100Gbps NIC). Benefit: **High IOPS with improved (tail) latency**.

  - ADQ is an application specific queuing and steering technology that **dedicates and isolates** application specific hardware NIC queues.
  - These queues are then connected optimally to application specific threads of execution.

With ADQ
Application traffic to a dedicated set of queues

Without ADQ
Application traffic intermixed with other traffic types
ADQ integration requirements

- **Kernel & driver**: (a) Busy polling; (B) socket option for NAPI_ID (SO_INCOMING_NAPI_ID); (c) symmetric polling; ....
- **Application**: Epoll threads watch the socket with same NAPI_ID
- **Hardware**: (a) Application level filtering & traffic shaping; (b) Flow based queue steering and load balance.
Conclusion
Conclusion

• SPDK NVMe-oF solution is well adopted by the industry. In this presentation, followings are introduced, i.e.,
  - The development history of SPDK NVMe-oF solution.
  - SPDK TCP transport status introduction & some further development plan.

• Call for activity in spdk community
  - Welcome to bug submission, idea discussion and patch submission.
## Conclusion: Further development plan for NVMe-oF in SPDK

<table>
<thead>
<tr>
<th>Ingredients</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spec compliance</td>
<td>Continue following the NVMe-oF spec</td>
</tr>
<tr>
<td>Interoperability with kernel</td>
<td>Continue the interoperability test with Linux kernel solution.</td>
</tr>
<tr>
<td>Performance</td>
<td>Continue performance enhancements and integration with other solutions.</td>
</tr>
<tr>
<td>Advanced feature</td>
<td>Continuing extracting the common features from customers and put in the roadmap</td>
</tr>
</tbody>
</table>
Q & A