SPDK NVMe-oF RDMA (Target & Initiator) Performance Report Release 24.05

Intel E810-CQDA2 RoCEv2 version

Testing Date: July 2024

Performed by:

Karol Latecki (karol.latecki@intel.com)

Jaroslaw Chachulski (jaroslawx.chachulski@intel.com)

Acknowledgments:

Krzysztof Karas (krzysztof.karas@intel.com)

Contents

Contents 2
Audience and Purpose 3
Test setup 4
Target Configuration 4
Initiator 1 Configuration 5
Initiator 2 Configuration 5
BIOS settings 6
SPDK Build Options 6
Introduction to SPDK NVMe-oF (Target & Initiator) 7
Test Case 1: SPDK NVMe-oF RDMA Target I/O core scaling 9
4KiB Random Read Results 12
4KiB Random Write Results 13
4KiB Random Read-Write Results 14
Large Sequential I/O Performance 15
Conclusions 16
Test Case 2: SPDK NVMe-oF RDMA Initiator I/O core scaling 17
4KiB Random Read Results 19
4KiB Random Write Results 20
4KiB Random 70/30 Read/Write Results 21
Conclusions 22
Test Case 3: Linux Kernel vs. SPDK NVMe-oF RDMA Latency 23
SPDK vs Kernel NVMe-oF RDMA Target Results 26
SPDK vs Kernel NVMe-oF RDMA Initiator Results 27
SPDK vs Kernel NVMe-oF RDMA Kernel + Initiator Results 28
Conclusions 29
Test Case 4: NVMe-oF RDMA Performance with increasing # of connections 30
4KiB Random Read Results 32
4KiB Random Write results 33
4KiB Random Read-Write Results 34
Conclusions 35
Summary 36
List of tables 37
List of figures 39
Appendix A – Test Case 1 & 2 SPDK NVMe-oF Initiator bdev configuration 40
Appendix B – Test Case 3 SPDK NVMe-oF Initiator bdev configuration 44
Appendix C - Kernel NVMe-oF RDMA Target configuration 44

Audience and Purpose

This report is intended for people who are interested in evaluating SPDK NVMe-oF (Target & Initiator) performance as compared to the Linux Kernel NVMe-oF (Target & Initiator). This report contains performance and efficiency of the SPDK vs. Linux Kernel NVMe-oF Target and Initiator for the RDMA transport only.

The purpose of report is not to imply a single "correct" approach, but rather to provide a baseline of well-tested configurations and procedures that produce repeatable results. This report can also be viewed as information regarding best known method/practice when performance testing SPDK NVMe-oF Target and Initiator components.

Test setup

Target Configuration

Table 1: Hardware setup configuration – Target system

Item	Description
Server Platform	SuperMicro® Ultra SuperServer SYS-220U-TNR 
Motherboard	Server board **X12DPU-6**
CPU	2 CPU sockets, Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz Number of cores: 28 per socket, number of threads: 56 per socket Both sockets populated Microcode: 0xd0003d1
Memory	16 x 32GB SK Hynix HMA84GR7DJR4N-XN, DDR4, 3200MHz Total of 512GB
Operating System	Fedora 37
BIOS	1.8
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
SPDK version	SPDK 24.05
Storage	**OS:** 1x 250GB Crucial CT250MX500SSD1 **Storage Target**: 14x Kioxia® KCM61VUL3T20 3.2TBs (FW: 0105) (6 on CPU NUMA Node 0, 8 on CPU NUMA Node 1)
NIC	4x 100GbE Intel(R) Ethernet Network Adapter E810-CQDA2. Single port connected on each NIC. ice driver version: 1.11.14 Irdma driver version: 1.11.58 NVM FW version: v4.20 2 NICs per CPU socket. Protocol: RoCEv2 *Note: The provided links are directly downloadable.*

Initiator 1 Configuration

Table 2: Hardware setup configuration – Initiator system 1

Item	Description
Server Platform	Intel® Server System M50CYP2UR208
CPU	Intel® Xeon® Gold 6348 Processor @ 2.60GHz (42MB Cache) Number of cores: 28 per socket, number of threads: 56 per socket (Both sockets populated) Microcode: 0xd000389
Memory	16 x 32GB Micron 36ASF4G72PZ-3G2J3, DDR4, 3200MHz Total 512GBs
Operating System	Fedora 37
BIOS	BIOS version: R01.01.0009
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
SPDK version	SPDK 24.05
Storage	**OS:** 1x 250GB Crucial CT250MX500SSD1
NIC	2x 100GbE Intel(R) Ethernet Network Adapter E810-CQDA2. Single port connected on each NIC. ice driver version: 1.11.14 Irdma driver version: 1.11.58 NVM FW version: v4.20 Protocol: RoCEv2 *Note: The provided links are directly downloadable.*

Initiator 2 Configuration

Table 3: Hardware setup configuration – Initiator system 2

Item	Description
Server Platform	Intel® Server System M50CYP2UR208
CPU	Intel® Xeon® Gold 6348 Processor @ 2.60GHz (42MB Cache) Number of cores: 28 per socket, number of threads: 56 per socket (Both sockets populated) Microcode: 0xd000389
Memory	16 x 32GB Micron 36ASF4G72PZ-3G2J3, DDR4, 3200MHz Total 512GBs
Operating System	Fedora 37
BIOS	BIOS version: R01.01.0009
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
SPDK version	SPDK 24.05
Storage	**OS:** 1x 250GB Crucial CT250MX500SSD1
NIC	2x 100GbE Intel(R) Ethernet Network Adapter E810-CQDA2. Single port connected on each NIC.

	ice driver version: 1.11.14 Irdma driver version: 1.11.58 NVM FW version: v4.20 Protocol: RoCEv2 *Note: The provided links are directly downloadable.*

BIOS settings

Table 4: Test systems BIOS settings

Item	Description
BIOS ***(Applied to all 3 systems)***	Hyper threading Enabled CPU Power and Performance Policy: • "Extreme Performance" for Target • "Performance" for Initiators CPU C-state No Limit CPU P-state Enabled Enhanced Intel® SpeedStep® Tech Enabled Turbo Boost Enabled

SPDK Build Options

All measurements included in this report document were done with SPDK build with "—enable-lto" option enabled. Link time optimization allows better SPDK performance thanks to code optimization done by inlining functions across compilation units, which in turn results in reduced function call overhead.

Introduction to SPDK NVMe-oF (Target & Initiator)

The NVMe over Fabrics (NVMe-oF) protocol extends the parallelism and efficiencies of the NVM Express* (NVMe) block protocol over network fabrics such as RDMA (iWARP, RoCE, InfiniBand™), Fiber Channel and TCP. SPDK provides both a user-space NVMe-oF target and initiator that extends the software efficiencies of the rest of the SPDK stack over the network. The SPDK NVMe-oF target uses the SPDK user-space, polled-mode NVMe driver to submit and complete I/O requests to NVMe devices which reduces the software processing overhead. Likewise, it pins connections to CPU cores to avoid synchronization and cache thrashing so that the data for those connections is kept as close to the CPU cache as possible.

The SPDK NVMe-oF target and initiator use the Infiniband/RDMA verbs API to access an RDMA-capable NIC. These should work on all flavors of RDMA transports but for the purpose of this report document are tested against RoCEv2. Similar to the SPDK NVMe driver, SPDK provides a user-space, lockless, polled-mode NVMe-oF initiator. The host system uses the initiator to establish a connection and submit I/O requests to an NVMe subsystem within an NVMe-oF target. NVMe subsystems contain namespaces, each of which maps to a single block device exposed via SPDK's bdev layer. SPDK's bdev layer is a block device abstraction layer and general-purpose block storage stack akin to what is found in many operating systems. Using the bdev interface completely decouples the storage media from the front-end protocol used to access storage. Users can build their own virtual bdevs that provide complex storage services and integrate them with the SPDK NVMe-oF target with no additional code changes. There can be many subsystems within an NVMe-oF target and each subsystem may hold many namespaces. Subsystems and namespaces can be configured dynamically via a JSON-RPC interface.

Figure 1 shows a high-level schematic of the systems used for testing in the rest of this report. The set up consists of three individual systems (two used as initiators and one used as the target). The NVMe-oF target is connected to both initiator systems point-to-point using QSFP28 cables without any switches. The target system has fourteen Kioxia® KCM61VUL3T20 SSDs which were used as block devices for NVMe-oF subsystems 100GbE Intel® E810-CQDA2 NICs connected to provide up to 200GbE of network bandwidth. Each Initiator system has two Intel® E810-CQDA2 100GbE NICs connected directly to the target without any switch.

One goal of this report was to make clear the advantages and disadvantages inherent to the design of the SPDK NVMe-oF components. These components are written using techniques such as run-to completion, polling, and asynchronous I/O. The report covers four real-world use cases.

For performance benchmarking the fio tool is used with two storage engines:

1) Linux Kernel libaio engine
2) SPDK bdev engine

Performance numbers reported are aggregate I/O per second, average latency, and CPU utilization as a percentage for various scenarios. Aggregate I/O per second and average latency data is reported from fio and CPU utilization was collected using sar (sysstat).

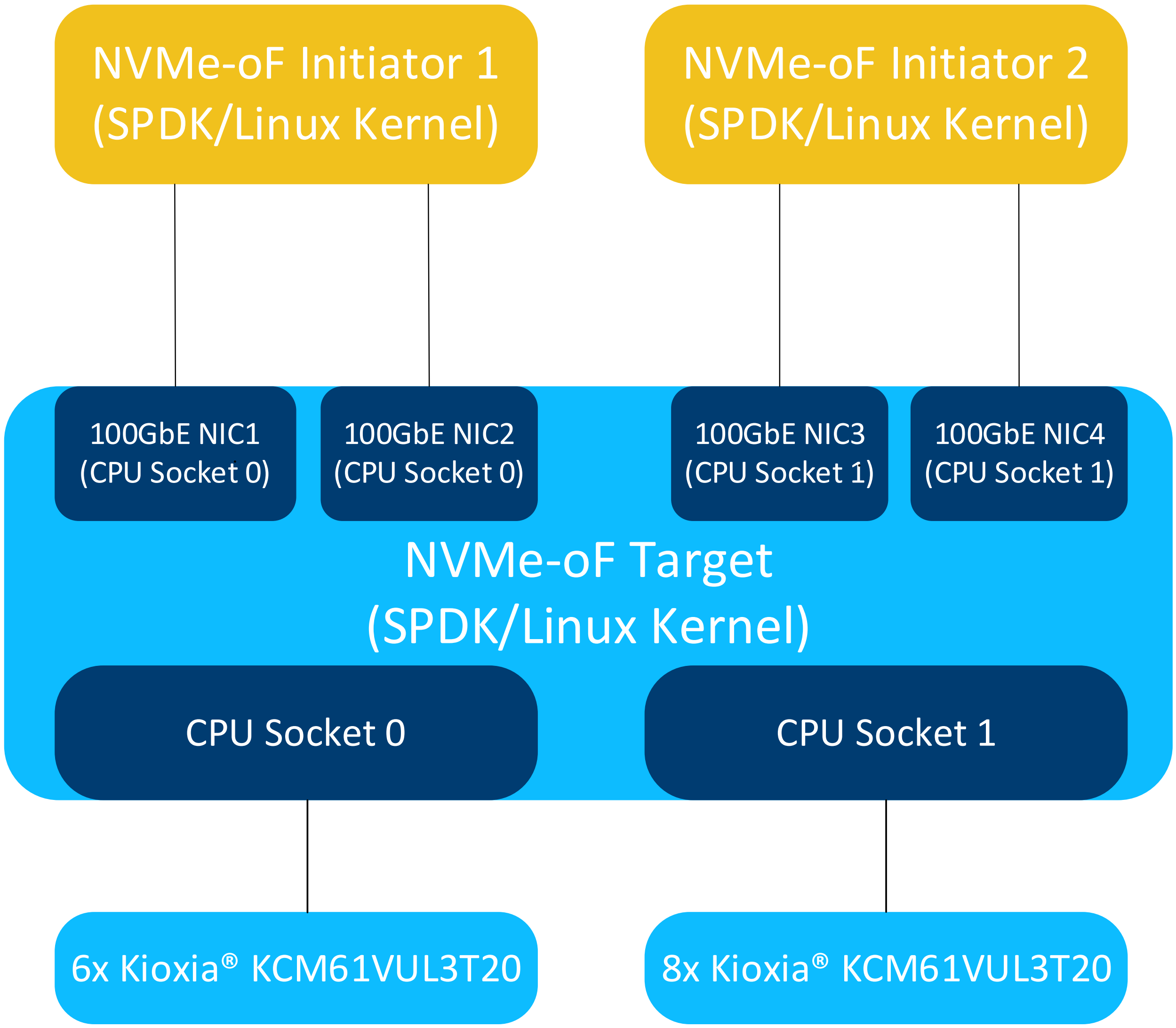

Figure 1: High-Level NVMe-oF RDMA performance testing setup

Test Case 1: SPDK NVMe-oF RDMA Target I/O core scaling

This test case was designed to demonstrate how the SPDK NVMe-oF target throughput in IOPS (I/O per second) scales when additional CPU cores are added to the SPDK NVMe-oF target application.

The SPDK NVMe-oF RDMA target was configured to run with 14 NVMe-oF subsystems. Each NVMe-oF subsystem ran on top of an individual NVMe bdev backed by a single Kioxia KCM61VUL3T20 NVMe drive. Each of the 2 host systems was connected to 7 NVMe-oF subsystems which were exported by the SPDK NVMe-oF Target over 2 x 100GbE link. The SPDK bdev fio plugin was used to target 7 NVMe-oF bdevs on each of the host. The SPDK Target was configured to use up to 10 CPU cores. We ran the following workloads on each initiator:

- 4KiB 100% Random Read
- 4KiB 100% Random Write
- 4KiB Random 70% Read 30% Write

Below table contains information about the test configuration in form of a sequence of commands used by spdk/scripts/rpc.py script to configure the SPDK NVMe-oF Target. The SPDK NVMe-oF Initiator (bdev fio_plugin) still uses plain configuration files.

Each workload was run three times at each CPU count and the reported results are the average of 3 runs. We preconditioned SSDs once before running the 4KiB Random Read and 4KiB Random 70/30 Read/Write workloads to ensure that the SSDs reached their steady state where we get repeatable results. However, for the 4KiB Rand Write workload we didn't precondition the NVMe devices to ensure workload saturated the network rather than being limited to the steady state performance of the SSDs, which is much lower than the available network bandwidth.

Table 5: SPDK NVMe-oF RDMA Target Core Scaling test configuration

Item	Description
Test Case	SPDK NVMe-oF Target I/O core scaling
SPDK NVMe-oF Target configuration	All the commands below were executed with spdk/scripts/rpc.py script. **Construct NVMe bdevs:** bdev_nvme_attach_controller -t PCIe -b Nvme0 -a 0000:17:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme1 -a 0000:18:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme2 -a 0000:65:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme3 -a 0000:66:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme4 -a 0000:67:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme5 -a 0000:68:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme6 -a 0000:98:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme7 -a 0000:99:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme8 -a 0000:9a:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme9 -a 0000:9b:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme10 -a 0000:e3:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme11 -a 0000:e4:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme12 -a 0000:e5:00.0 bdev_nvme_attach_controller -t PCIe -b Nvme13 -a 0000:e6:00.0

<table>
<tr><td></td><td>
Create RDMA transport layer:
nvmf_create_transport -t RDMA -n 8192
{
trtype: "RDMA"
max_queue_depth: 128
max_qpairs_per_ctrlr: 64
in_capsule_data_size: 4096
max_io_size: 131072
io_unit_size: 8192
max_aq_depth: 128
num_shared_buffers: 8192
buf_cache_size: 32
}

Create NVMe-oF subsystems and add NVMe bdevs as namespaces:
for i in $(seq 1 16); do
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode${i} -s SPDK00${i} -a -m 8
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode${i} Nvme$((i-1))n1
done

Add listeners to NVMe-oF Subsystems:
i=1
ips=(20.0.0.1 20.0.1.1 10.0.0.1 10.0.1.1)
for ip in ${ips[@]}; do
for j in $(seq 1 4); do
nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode${i} -t rdma \
-f ipv4 -s 4420 -a ${ip}
((i++))
done
done
</td></tr>
<tr><td>SPDK NVMe-oF Initiator - fio plugin configuration</td><td>
BDEV.conf
See Appendix A

fio.conf
[global]
ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin
spdk_json_conf=/tmp/spdk/bdev.conf
thread=1
group_reporting=1
direct=1
norandommap=1
rw=randrw
rwmixread={100, 70, 0}
bs=4k
iodepth={1, 64, 128, 192, 256}
time_based=1
ramp_time=60
runtime=300

[filename0]
filename=Nvme0n1
[filename1]
filename=Nvme1n1
[filename2]
filename=Nvme2n1
[filename3]
filename=Nvme3n1
[filename4]
filename=Nvme4n1
[filename5]
</td></tr>
</table>

	filename=Nvme5n1 [filename6] filename=Nvme6n1

4KiB Random Read Results

Table 6: SPDK NVMe-oF RDMA Target Core Scaling results, Random Read IOPS, QD=128

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	6214.54	1590.9	1125.9
2 cores	14395.49	3685.2	485.9
3 cores	23100.56	5913.7	306.2
4 cores	31152.24	7975.0	225.0
5 cores	37644.38	9637.0	185.7
6 cores	40651.60	10406.8	171.8
8 cores	42721.93	10936.8	163.4
10 cores	42670.92	10923.7	163.6

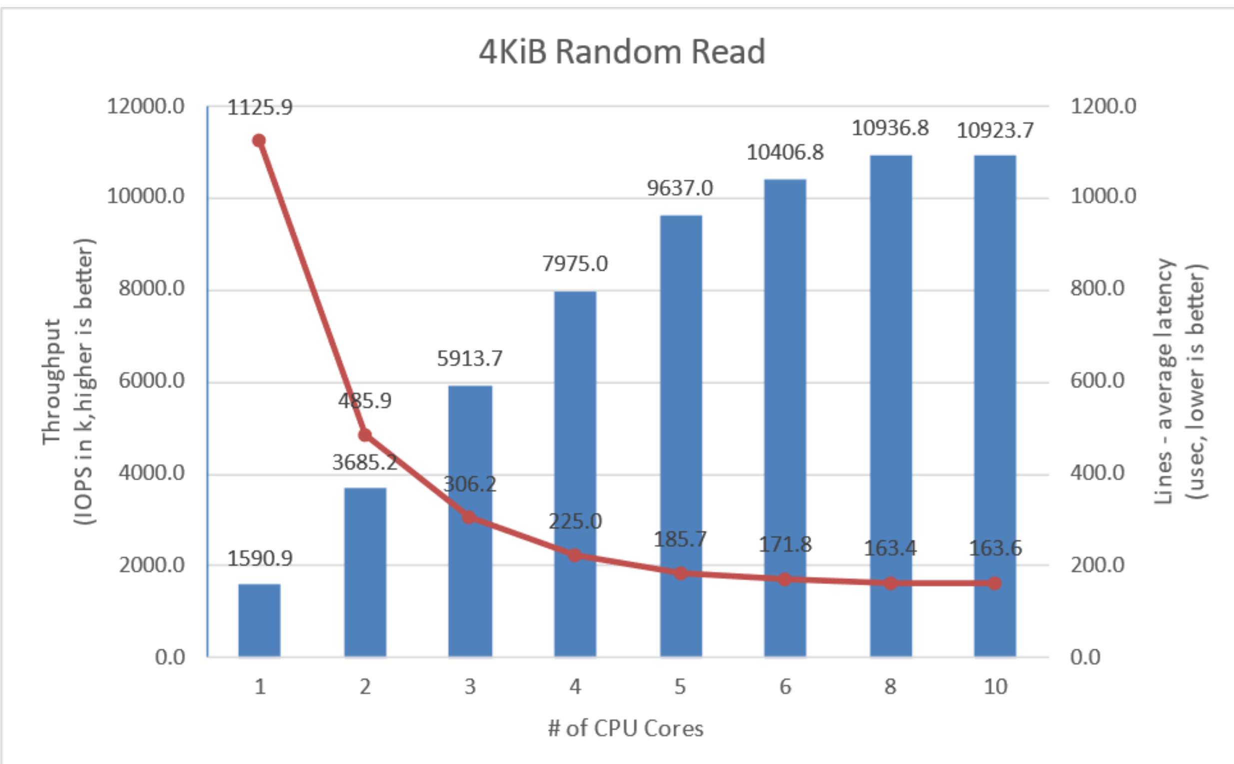

Figure 2: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read workload at QD=128

4KiB Random Write Results

Table 7: SPDK NVMe-oF RDMA Target Core Scaling results, Random Write IOPS, QD=64

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	8715.37	2231.1	400.0
2 cores	19977.12	5114.1	173.0
3 cores	30633.68	7842.2	114.8
4 cores	38635.56	9890.7	88.5
5 cores	40800.67	10445.0	84.5
6 cores	40962.52	10486.4	84.4
8 cores	41168.65	10539.2	84.3
10 cores	41218.05	10551.8	84.3

Note that the SSDs were not preconditioned for the 4K random write workload because that would limit the workload performance to the SSDs steady state performance.

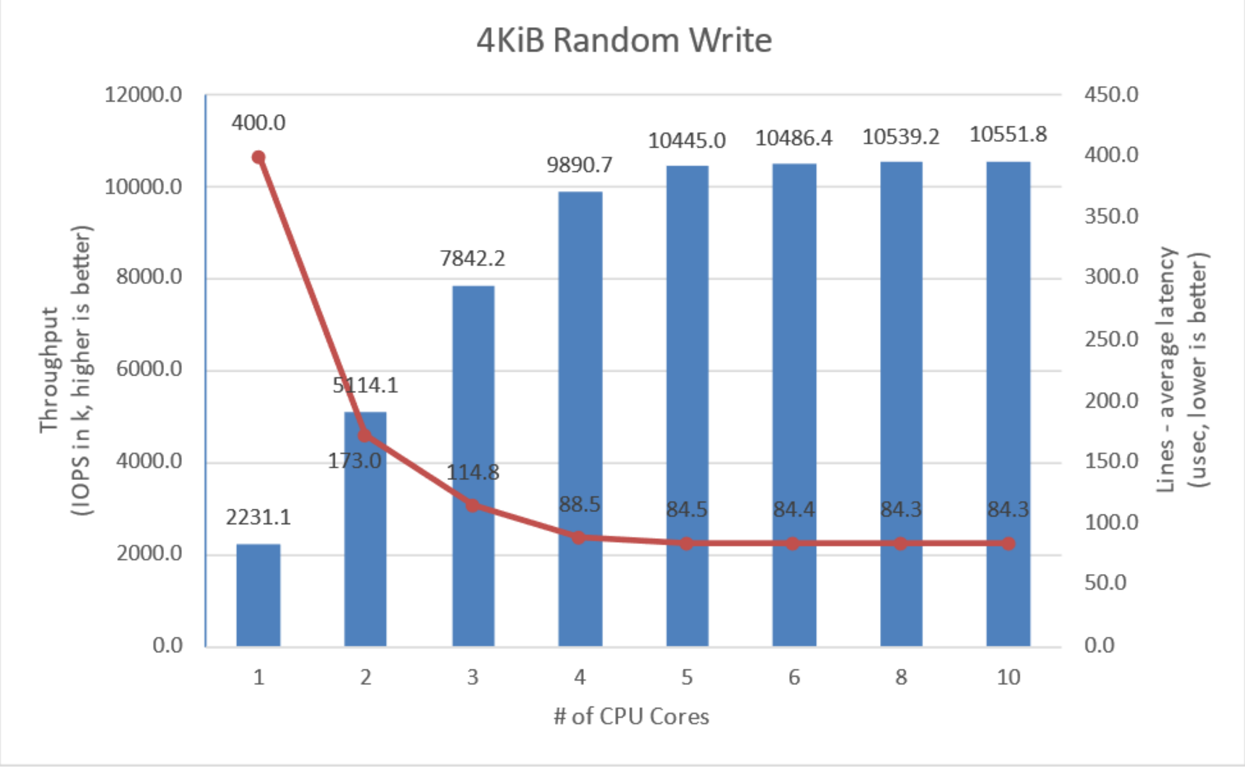

Figure 3: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=64

4KiB Random Read-Write Results

Table 8: SPDK NVMe-oF RDMA Target Core Scaling results, Random Read/Write 70%/30% IOPS, QD=128

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	6433.19	1646.9	1087.5
2 cores	15134.40	3874.4	461.5
3 cores	23515.09	6019.9	301.1
4 cores	31231.14	7995.2	223.4
5 cores	37762.87	9667.3	184.5
6 cores	40876.61	10464.4	170.4
8 cores	45321.86	11602.4	153.7
10 cores	46089.12	11798.8	151.1

Figure 4: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB Random 70/30 Read/Write workload at QD=128

Large Sequential I/O Performance

128KiB block size I/O tests were performed with sequential I/O workloads at queue depth 4 and 8 . The rest of the fio configuration is similar to the 4KiB test case in the previous part of this document. We used iodepth=4 and iodepth=8, because higher queue depth resulted in negligible bandwidth gain and a significant increase in latency.

Table 9: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential Read IOPS, QD=4

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	42340.58	338.7	165.2
2 cores	42530.52	340.2	164.4
3 cores	42530.89	340.2	164.4
4 cores	42568.69	340.5	164.3

Table 10: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential Write IOPS, QD=4

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	43961.13	351.7	159.1
2 cores	43982.00	351.9	159.0
3 cores	43982.68	351.9	159.0
4 cores	43988.23	351.9	159.0

Table 11: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential 70% Read 30% Write IOPS, QD=8

# of Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	47916.12	383.3	291.8
2 cores	48050.63	384.4	291.0
3 cores	48028.95	384.2	291.2
4 cores	48077.86	384.6	290.9

Conclusions

1. 100% 4KiB Random Read workload throughput scales up almost linearly with the addition of I/O cores up to 4 cores. Peak performance is reached with 8 CPU cores. Latency decreases significantly with the addition of 2 core. As more cores are added the rate of reduction slows down and stabilizes after 6 cores.

2. 100% 4KiB Random Write workload throughput scales up with the addition of I/O cores up to 3 cores reaching 7.8 million IOPS. Further increasing the number of CPU cores results in non-linear performance improvement. Peak performance is reached with 10 CPU cores. Latency decreases significantly with the addition of 2 core. After addition of 4 cores the improvements become insignificant.

3. 70/30% 4KiB Random Read/Write throughput scales up with the addition of I/O cores up to 4 cores reaching almost 8 million IOPS. Further increasing the number of CPU cores results in non-linear performance improvement. Peak performance is reached with 10 CPU cores saturating network link. Latency decreases significantly with the addition of 2 core. As more cores are added the rate of reduction slows down and stabilizes after 6 cores.

4. For large sequential I/Os, a single CPU core Target saturated the network bandwidth. Therefore, adding more CPU cores did not result in increased performance for these workloads, because the network was saturated.

Test Case 2: SPDK NVMe-oF RDMA Initiator I/O core scaling

This test case was designed to demonstrate how the SPDK NVMe-oF initiator throughput in IOPS (I/O per second) scales when additional CPU cores are added to the SPDK NVMe-oF initiator.

The test setup for this test case is slightly different than the set up described in introduction chapter, as we used just a single SPDK NVMe-oF RDMA Initiator. The Initiator was connected to Target server with two 100 Gbps network links, using single port from two separate network interface cards.

The SPDK NVMe-oF RDMA Target was configured using 6 cores; all the other configurations are similar to test case 1. The SPDK bdev fio plugin was used to target 14 individual NVMe-oF subsystems exported by the Target. The number of CPU threads used by the fio process was managed by setting the fio job sections and numjobs parameter and ranged from 1 to 8 CPUs. For detailed fio job configuration see table below.

- 4KiB 100% Random Read
- 4KiB 100% Random Write
- 4KiB Random 70% Read 30% Write

It is important to note that fio iodepth parameter values presented in the table below are actual queue depths used for each of the connected subsystem. These values were calculated in test based on number of fio job sections, numjobs parameter and the number of "filename" targets grouped in each of the fio job sections.

Table 12: SPDK NVMe-oF RDMA Initiator Core Scaling test configuration

Item	Description
Test Case	SPDK NVMe-oF RDMA Initiator I/O core scaling
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 6 CPU cores.
SPDK NVMe-oF Initiator 1 - fio plugin configuration	**BDEV.conf** See Appendix B. **fio.conf** **For 1 CPU initiator configuration:** [global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={1 ,32, 64, 128, 192, 256} time_based=1

	ramp_time=60 runtime=300 numjobs=1 [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 filename=Nvme8n1 filename=Nvme9n1 filename=Nvme10n1 filename=Nvme11n1 filename=Nvme12n1 filename=Nvme13n1
	fio.conf **For CPU > 1 (up to N=8) initiator configuration “filename=NvmeXn1” are evenly spread across fio job threads:** [global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={1,32, 64, 128, 192, 256} time_based=1 ramp_time=60 runtime=300 numjobs=X [filename0] filename=Nvme0n1 filename=Nvme1n1 [filename1] filename=Nvme2n1 filename=Nvme3n1 […] [filename N-1] filename=Nvme10n1 filename=Nvme11n1 [filename N] filename=Nvme12n1 filename=Nvme13n1

4KiB Random Read Results

Table 13: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random Read IOPS, QD=64, SPDK Target 6 CPU Cores

# of Initiator CPU Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	5369.50	1374.6	537.5
2 cores	14016.99	3588.3	216.0
3 cores	19320.85	4946.1	164.4
4 cores	21342.24	5463.6	161.0
5 cores	21300.78	5453.0	163.0
6 cores	21283.18	5448.5	163.9
7 cores	21283.92	5448.7	164.1
8 cores	21286.21	5449.3	164.1

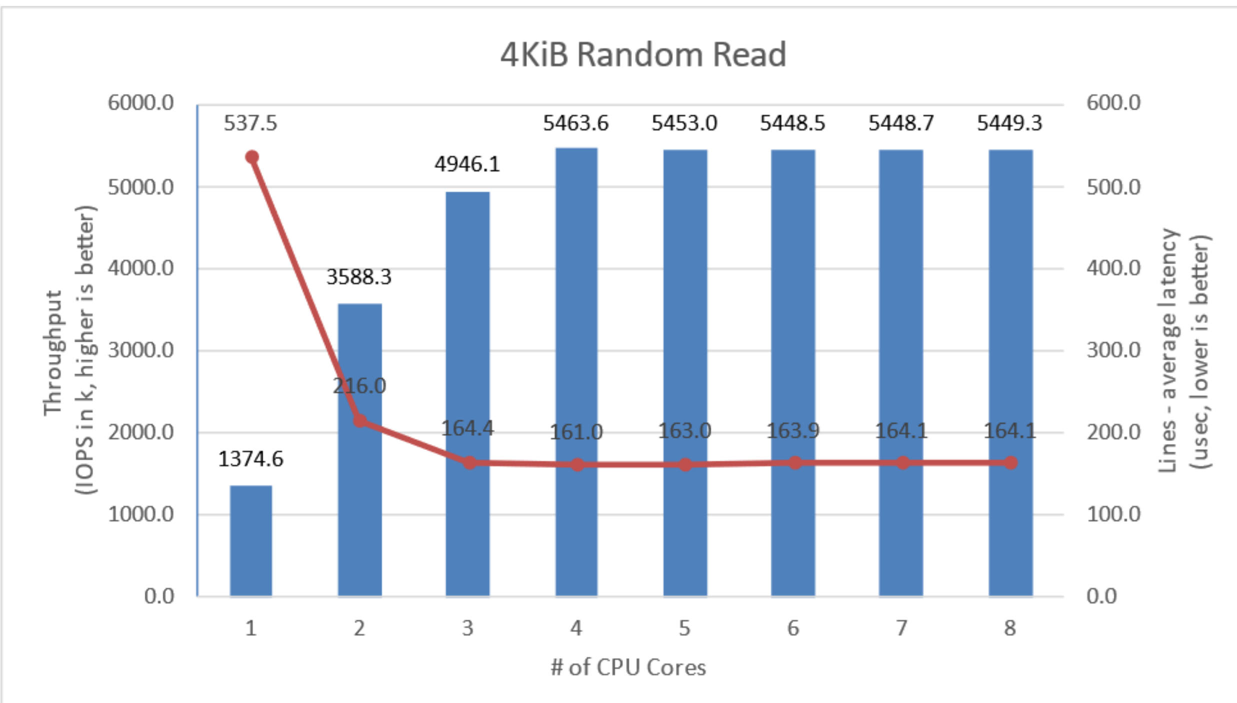

Figure 5: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read QD=64 workload

4KiB Random Write Results

Note: The SSDs were not pre-conditioned before running the 100% Random Write test cases. This allowed the throughput to scale to the 2x 100GbE network bandwidth rather than limiting the workload performance to the storage bottleneck (which is approx. 3.2M IOPS).

Table 14: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random Write IOPS, QD=64, SPDK Target 6 CPU Cores

# of Initiator CPU Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	5288.66	1353.9	540.5
2 cores	11597.50	2969.0	248.2
3 cores	15963.79	4086.7	200.9
4 cores	18158.20	4648.5	191.6
5 cores	18870.00	4830.7	186.2
6 cores	20359.07	5211.9	171.3
7 cores	20963.12	5366.6	166.5
8 cores	21149.28	5414.2	165.1

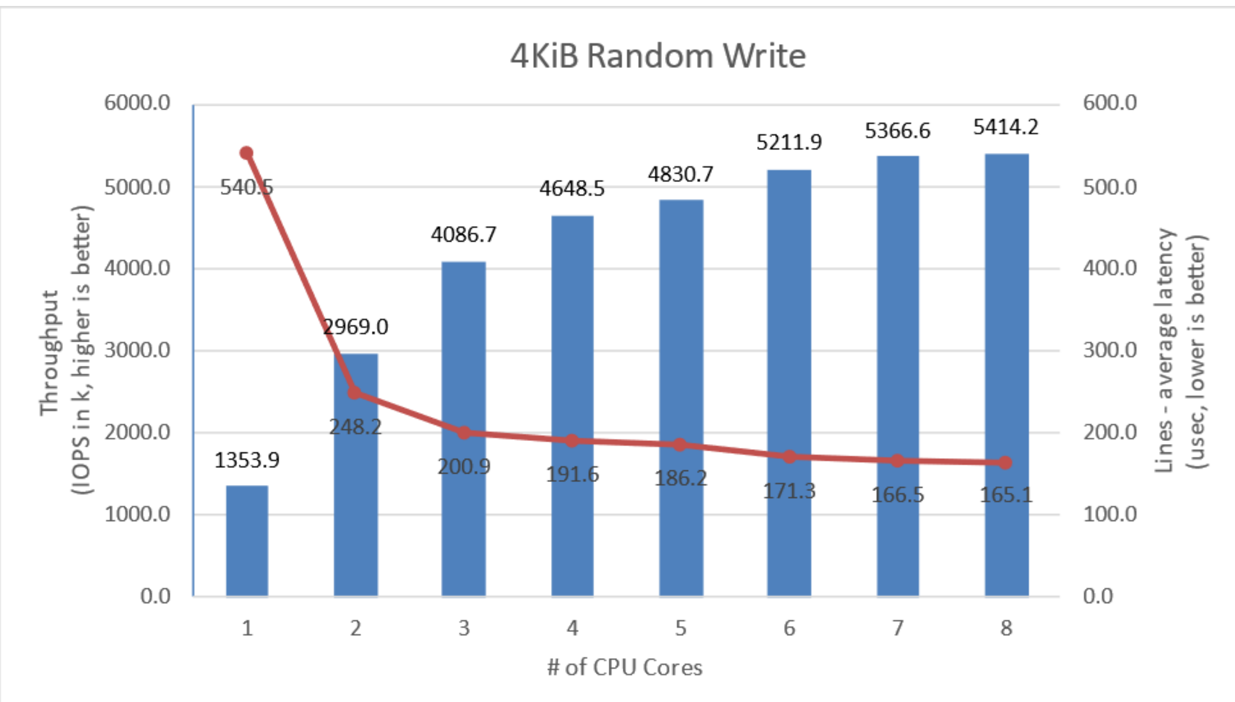

Figure 6: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=64

4KiB Random 70/30 Read/Write Results

Table 15: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random 70%/30% Read/Write IOPS, QD=64, SPDK Target 6 CPU Cores

# of Initiator CPU Cores	Bandwidth (MiBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	5242.73	1342.1	538.0
2 cores	13449.76	3443.1	219.3
3 cores	18407.31	4712.3	166.7
4 cores	24497.92	6271.5	135.4
5 cores	26042.29	6666.8	130.5
6 cores	27041.83	6922.7	127.7
7 cores	27602.42	7066.2	125.9
8 cores	28005.68	7169.5	124.1

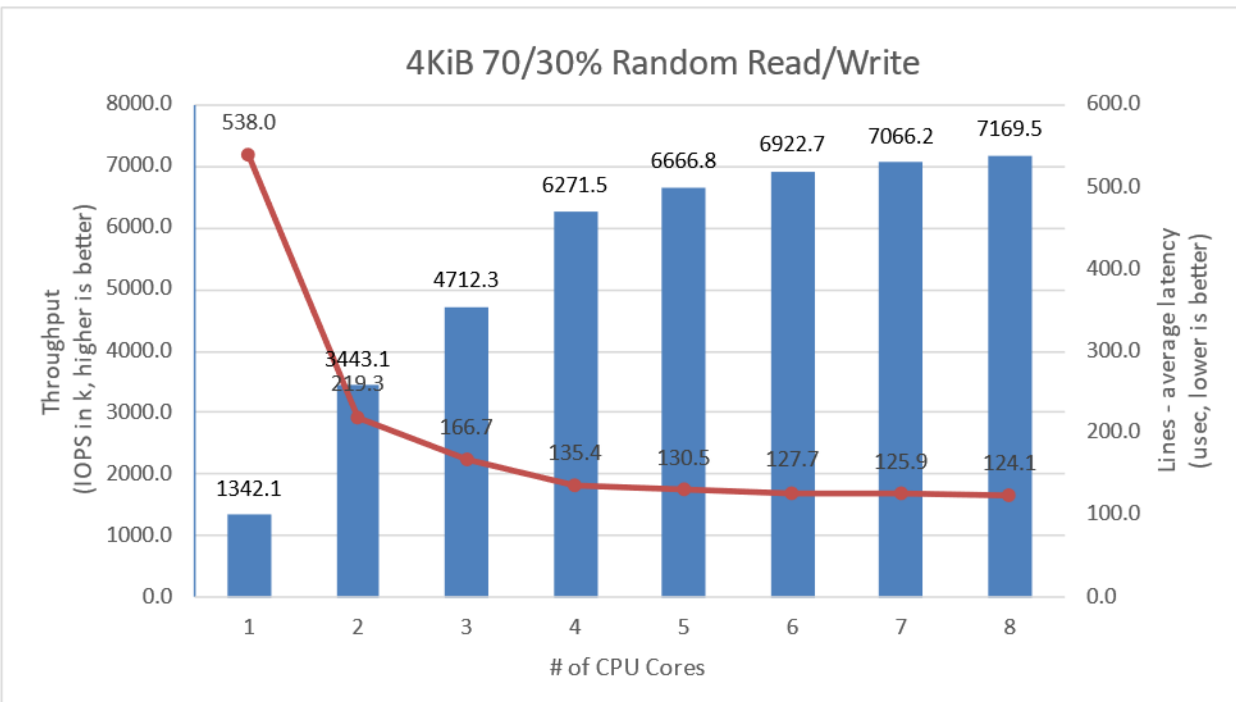

Figure 7: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB Random 70% Read 30% Write Workload at QD=64

Conclusions

1. Random Read throughput scaling was linear when increasing the number of initiator cores up to 3 CPU cores. With 4 CPU cores, a performance of approximately 5.46 million IOPS was achieved using the SPDK NVMe-oF RDMA Initiator, thereby saturating the 200 GbE link between the Target and the Initiator.

2. The performance scaling of the Random Write workload was not linear with the addition of initiator CPU cores. Initially, there was a significant increase from 1.3 million IOPS with one core to 3 million IOPS with two cores. Subsequent additions of cores led to smaller increments in performance. IOPS growth decelerated with additional cores, peaking at 5.4 million with eight cores, at which point the NVMe Target drives reached their maximum 4KiB random write processing capability.

3. The Random Read/Write performance scaled sub-linearly, reaching 6.2 million IOPS with four I/O CPUs. Beyond this, the increase in Initiator CPU cores led to a non-linear performance gain, peaking at 7.1 million IOPS with eight cores using the SPDK NVMe-oF RDMA initiator. This throughput surpassed the 200 GbE link capacity between the Target and Initiator, due to the bi-directional nature of the mixed read-write workload, which isn't strictly bound by the 200 GbE one-way network limit.

Test Case 3: Linux Kernel vs. SPDK NVMe-oF RDMA Latency

This test case was designed to understand latency characteristics of SPDK NVMe-oF RDMA Target and Initiator vs. the Linux Kernel NVMe-oF RDMA Target and Initiator implementations on a single NVMe-oF subsystem. The average I/O latency and p99 latency was compared between SPDK NVMe-oF (Target/Initiator) vs. Linux Kernel (Target/Initiator). Both SPDK and Kernel NVMe-oF Targets were configured to run on a single core, with a single NVMe-oF subsystem backed by a Null Block Device. The null block device (bdev) was chosen as the backend block device to eliminate the media latency during these tests.

Kernel NVMe-oF Initiator disclaimer:

For establishing Kernel NVMe-oF RDMA Initiator connections "nvme-cli" tool was used. While performing benchmark tests two issues were encountered:

- It was not possible to establish connection and create a NVMe block device on Initiator side with poll queues enabled (link). Using "nvme-cli" with "—nr-poll-queues" parameter present resulted in "Kernel Oops" to be generated. Because of this issue the fio workload for Kernel Initiator connection was configured to use "libaio" engine.
- Attempts to establish connection with default number of IO queues (which is equal to number of CPU cores on Initiator system) resulted in connection timeouts. To work around this issue "—nr-io-queues=32" was added to nvme-cli command. This does not affect the results in this test case as only a single connection with very small queue depth is tested.

Table 16: Linux Kernel vs. SPDK NVMe-oF RDMA Latency test configuration

Item	Description
Test Case	Linux Kernel vs. SPDK NVMe-oF RDMA Latency
	Test configuration
SPDK NVMe-oF Target configuration	The following commands are executed with spdk/scripts/rpc.py script to configure the SPDK NVMe-oF target. nvmf_create_transport -t RDMA (creates RDMA transport layer with default values: trtype: "RDMA" max_queue_depth: 128 max_qpairs_per_ctrlr: 64 in_capsule_data_size: 4096 max_io_size: 131072 io_unit_size: 8192 max_aq_depth: 128 num_shared_buffers: 8192 buf_cache_size: 32) bdev_null_create Nvme0n1 10240 4096 nvmf_subsystem_create nqn.2018-09.io.spdk:cnode1 -s SPDK001 -a -m 8 nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode1 Nvme0n1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t rdma -f ipv4 -s 4420 -a 20.0.0.1

<table>
<tr><td>Kernel NVMe-oF Target configuration</td><td>The following target configuration file loaded using nvmet-cli tool.
<pre>
{
 "ports": [
  {
   "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.0.1",
    "trsvcid": "4420",
    "trtype": "rdma"
   },
   "portid": 1,
   "referrals": [],
   "subsystems": [
    "nqn.2018-09.io.spdk:cnode1"
   ]
  }
 ],
 "hosts": [],
 "subsystems": [
  {
   "allowed_hosts": [],
   "attr": {
    "allow_any_host": "1",
    "version": "1.3"
   },
   "namespaces": [
    {
     "device": {
      "path": "/dev/nullb0",
      "uuid": "621e25d2-8334-4c1a-8532-b6454390b8f9"
     },
     "enable": 1,
     "nsid": 1
    }
   ],
   "nqn": "nqn.2018-09.io.spdk:cnode1"
  }
 ]
}
</pre></td></tr>
<tr><td colspan="2">fio configuration</td></tr>
<tr><td>SPDK NVMe-oF Initiator fio plugin configuration</td><td>BDEV.conf
See Appendix B.

fio.conf
<pre>
[global]
ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin
spdk_json_conf=/tmp/spdk/bdev.conf
thread=1
group_reporting=1
direct=1

norandommap=1
rw=randrw
rwmixread={100, 70, 0}
bs=4k
iodepth=1
time_based=1
ramp_time=60
runtime=300

[filename0]
</pre></td></tr>
</table>

	filename=Nvme0n1
Kernel initiator configuration	**Device config** The following configuration was performed using nvme-cli tool. modprobe nvme-fabrics nvme connect –n nqn.2018-09.io.spdk:cnode1 –t rdma –a 20.0.0.1 –s 4420 **fio.conf** [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth=1 time_based=1 ramp_time=60 runtime=300 [filename0] filename=/dev/nvme0n1

SPDK vs Kernel NVMe-oF RDMA Target Results

This following data was collected using the Linux Kernel initiator against both SPDK and Linux Kernel NVMe-oF RDMA target.

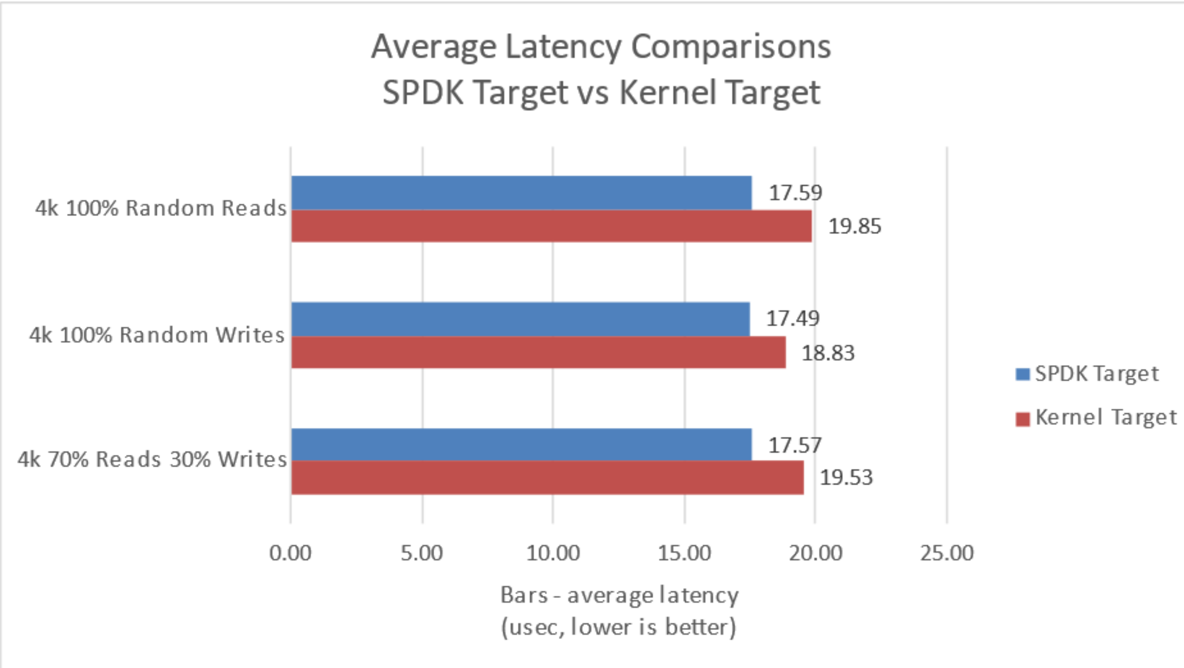

Figure 8: SPDK vs. Kernel NVMe-oF RDMA Target average I/O latency for various workloads run using the Kernel Initiator

Table 17: SPDK NVMe-oF RDMA Target Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	17.59	56018	17.0	19.0	42.8	370.7
4KiB 100% Random Write	17.49	56323	16.8	18.6	42.6	369.3
4KiB 70/30% Random Read/Write	17.57	55959	17.3	19.4	44.0	368.0

Table 18: Linux Kernel NVMe-oF RDMA Target Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	19.85	49713	19.2	21.3	45.1	254.3
4KiB 100% Random Write	18.83	52360	18.6	20.5	46.3	261.8
4KiB 70/30% Random Read/Write	19.53	50442	19.1	21.1	26.8	253.3

SPDK vs Kernel NVMe-oF RDMA Initiator Results

This following data was collected using the Linux Kernel and SPDK NVMe-oF RDMA initiator against an SPDK NVMe-oF RDMA target.

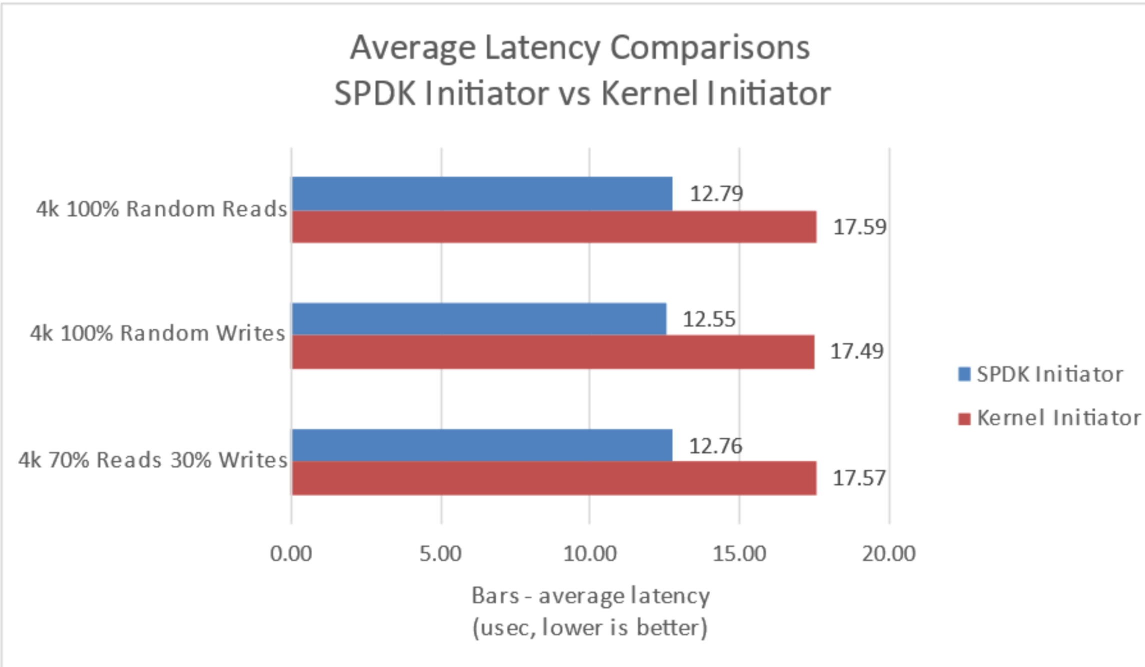

Figure 9: SPDK vs. Kernel NVMe-oF RDMA Initiator average I/O latency for various workloads against SPDK Target

Table 19: SPDK NVMe-oF RDMA Initiator Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	12.79	77225	14.0	14.8	36.6	355.7
4KiB 100% Random Write	12.55	78661	13.5	14.6	36.6	358.4
4KiB 70/30% Random Read/Write	12.76	77323	13.9	14.7	36.3	354.3

Table 20: Linux Kernel NVMe-oF RDMA Initiator Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	17.59	56018	17.0	19.0	42.8	370.7
4KiB 100% Random Write	17.49	56323	16.8	18.6	42.6	369.3
4KiB 70/30% Random Read/Write	17.57	55959	17.3	19.4	44.0	368.0

SPDK vs Kernel NVMe-oF RDMA Kernel + Initiator Results

Following data was collected using SPDK Target with SPDK Initiator and Linux Target with Linux Initiator.

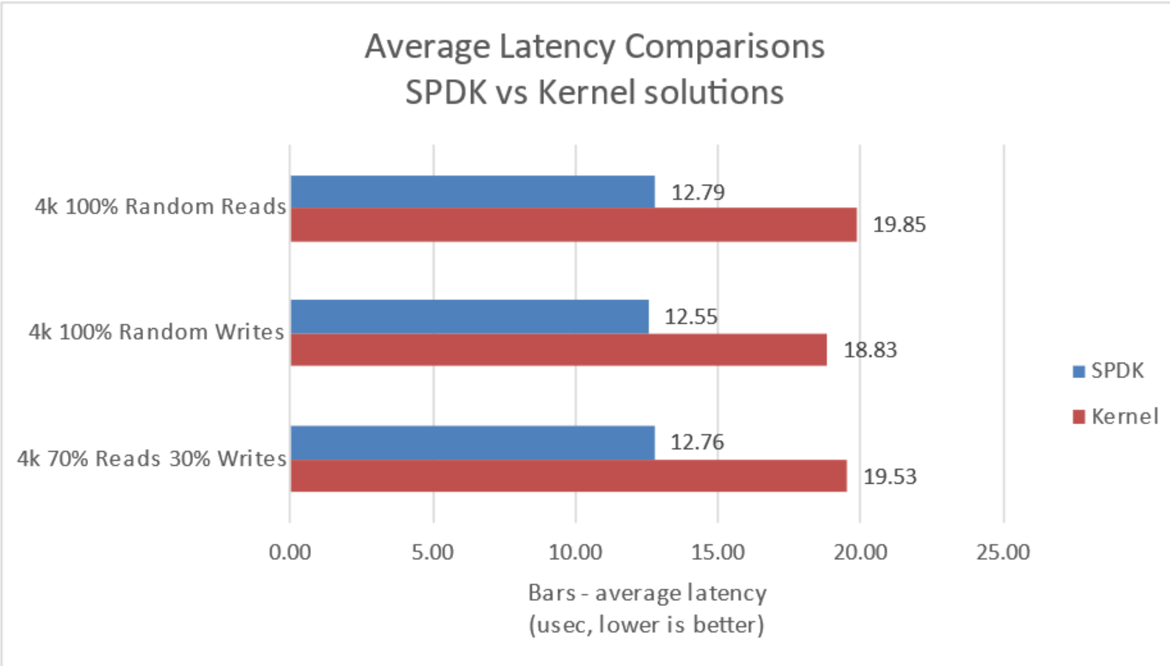

Figure 10: SPDK vs. Kernel NVMe-oF RDMA solutions average I/O Latency for various workloads against SPDK Target

Table 21: SPDK NVMe-oF RDMA Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	12.79	77225	14.0	14.8	36.6	355.7
4KiB 100% Random Write	12.55	78661	13.5	14.6	36.6	358.4
4KiB 70/30% Random Read/Write	12.76	77323	13.9	14.7	36.3	354.3

Table 22: Linux Kernel NVMe-oF RDMA Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Read	19.85	49713	19.2	21.3	45.1	254.3
4KiB 100% Random Write	18.83	52360	18.6	20.5	46.3	261.8
4KiB 70/30% Random Read/Write	19.53	50442	19.1	21.1	26.8	253.3

Conclusions

1. For the RDMA transport, the SPDK NVMe-oF Target reduces the NVMe-oF average round trip I/O latency by up to 2.26 usec vs. the Linux Kernel NVMe-oF target used in Fedora 37 6.0.18 setup. This is entirely software overhead, therefore, using the SPDK NVMe-oF target reduces the NVMe-oF software overhead by approximately 11.4% vs. the Linux Kernel NVMe-oF target.
2. The SPDK NVMe-oF Initiator reduces the NVMe-oF software overhead by up to 4.94 usec vs. the Linux Kernel NVMe-oF Initiator for the RDMA transport, which is approximately 28.25% of Linux Kernel NVMe-oF Initiator overhead.
3. The SPDK NVMe-oF TCP target and initiator reduced the average latency by up to 7.06 usec vs. the Linux Kernel NVMe-oF target and initiator, which eliminates up to 35.58% NVMe-oF software overhead.

4.

Test Case 4: NVMe-oF RDMA Performance with increasing # of connections

This test case was designed to demonstrate the throughput and latency of the SPDK NVMe-oF RDMA Target vs. Linux Kernel NVMe-oF RDMA Target under increasing number of connections per subsystem. The number of active connections (or I/O queue pairs) per NVMe-oF subsystem was varied, we measured the aggregated IOPS and number of CPU cores used by each target. The number of CPU cores metric was calculated from %CPU utilization measured using sar (sysstat package in Linux). The SPDK NVMe-oF RDMA Target was configured to run on 8 CPU cores, export 14 NVMe-oF subsystems (1 per Kioxia NVMe SSD) and 2 initiators were used both running the I/O workloads below to 7 separate subsystems using Kernel NVMe-oF RDMA initiator.

- 4KiB 100% Random Read
- 4KiB 100% Random Write
- 4KiB Random 70% Read 30% Write

Kernel NVMe-oF Initiator disclaimer:

For establishing Kernel NVMe-oF RDMA Initiator connections "nvme-cli" tool was used. While performing benchmark tests two issues were encountered:

- It was not possible to establish connection and create a NVMe block device on Initiator side with poll queues enabled (link). Using "nvme-cli" with "—nr-poll-queues" parameter present resulted in "Kernel Oops" to be generated. Because of this issue the fio workload for Kernel Initiator connection was configured to use "libaio" engine.
- Attempts to establish connection with default number of IO queues (which is equal to number of CPU cores on Initiator system) resulted in connection timeouts. To work around this issue "—nr-io-queues=32" was added to nvme-cli command. This does not affect the results in this test case as connections are not scaled beyond 16 per NVMe-oF subsystem, thus maximum number of used IO queues per NVMe-oF subsystem is 16.

Table 23: NVMe-oF RDMA Performance with increasing number of connections test configuration

Item	Description
Test Case	NVMe-oF RDMA Target performance with increasing # of connections
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 8 CPU cores.
Kernel NVMe-oF Target configuration	Target configuration file loaded using nvmet-cli tool. For detailed configuration file contents please see Appendix C.
Kernel NVMe-oF Initiator #1	**Device config** Performed using nvme-cli tool.

	modprobe nvme-fabrics nvme connect –n nqn.2018-09.io.spdk:cnode1 –t rdma –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode2 –t rdma –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode3 –t rdma –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode4 –t rdma –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode5 –t rdma –a 20.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode6 –t rdma –a 20.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode7 –t rdma –a 20.0.1.1 –s 4420
Kernel NVMe-oF Initiator #2	**Device config** Performed using nvme-cli tool. modprobe nvme-fabrics nvme connect –n nqn.2018-09.io.spdk:cnode8 –t rdma –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode9 –t rdma –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode10 –t rdma –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode11 –t rdma –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode12 –t rdma –a 10.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode13 –t rdma –a 10.0.1.1 –s 4420
fio configuration (used on both initiators)	**fio.conf** [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={32, 64, 128, 192} time_based=1 ramp_time=60 runtime=300 numjobs={1, 4, 8, 12, 16} [filename1] filename=/dev/nvme0n1 [filename2] filename=/dev/nvme1n1 [filename3] filename=/dev/nvme2n1 [filename4] filename=/dev/nvme3n1 [filename5] filename=/dev/nvme4n1 [filename6] filename=/dev/nvme5n1 [filename7] filename=/dev/nvme6n1

The SPDK NVMe-oF Target was configured to use 8 CPU cores for Random Read and Random Read/Write workloads and 4 CPU cores for Random Write workloads. We did not limit the number of CPU cores for the Linux Kernel NVMe-oF target. The graph below shows the relative efficiency in terms of IOPS/core which was calculated by dividing the total aggregate IOPS by the total CPU cores used while running that specific workload.

4KiB Random Read Results

Figure 11: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB 100% Random Reads QD=192, using the Kernel Initiator

Table 24: Linux Kernel NVMe-oF RDMA Target: 4KiB 100% Random Reads, QD=192

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	18987.80	4860.9	555.1	10.8
4	39174.02	10028.5	267.9	28.8
8	38978.48	9978.5	269.2	34.0
12	38582.16	9877.0	271.8	34.9
16	38665.42	9898.3	271.2	35.2

Table 25: SPDK NVMe-oF RDMA Target: 4KiB 100% Random Reads, QD=192

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	19498.75	4991.7	542.5	8.1
4	42618.37	10910.3	246.2	8.1
8	42605.17	10906.9	246.2	8.1
12	42316.48	10833.0	247.8	8.1
16	41952.85	10739.9	249.9	8.1

4KiB Random Write results

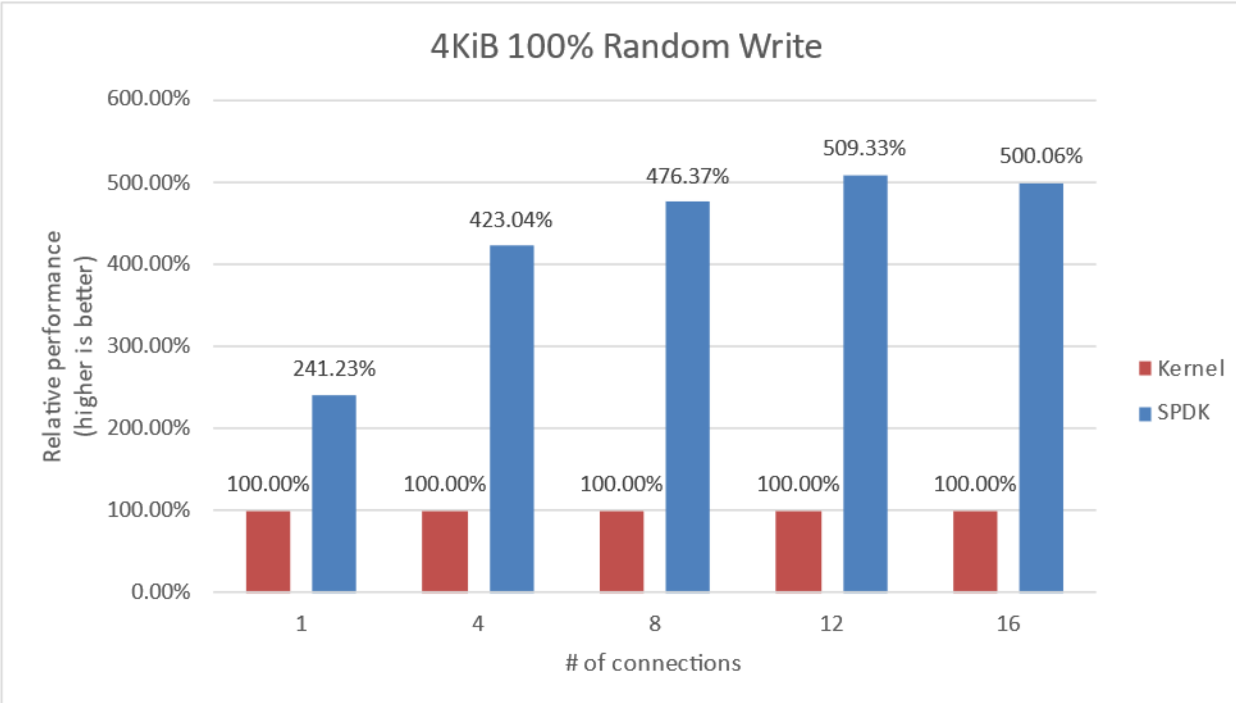

Figure 12: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB 100% Random Writes QD=128 workload, using Kernel Initiators

Note: The SSDs were not pre-conditioned before running 100% Random Write I/O te st.

Table 26: Linux Kernel NVMe-oF RDMA Target: 4KiB 100% Random Writes, QD=128

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	19781.39	5064.0	358.2	9.3
4	26803.11	6861.6	268.6	17.4
8	27395.39	7013.2	263.8	20.2
12	29112.17	7452.7	256.5	22.8
16	30116.98	7709.9	240.1	23.4

Table 27: SPDK NVMe-oF RDMA Target: 4KiB 100% Random Writes, QD=128

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	20775.32	5318.5	341.3	4.1
4	26504.01	6785.0	268.0	4.1
8	26300.68	6733.0	270.9	4.1
12	26476.38	6777.9	275.8	4.1
16	26180.79	6702.3	269.7	4.1

4KiB Random Read-Write Results

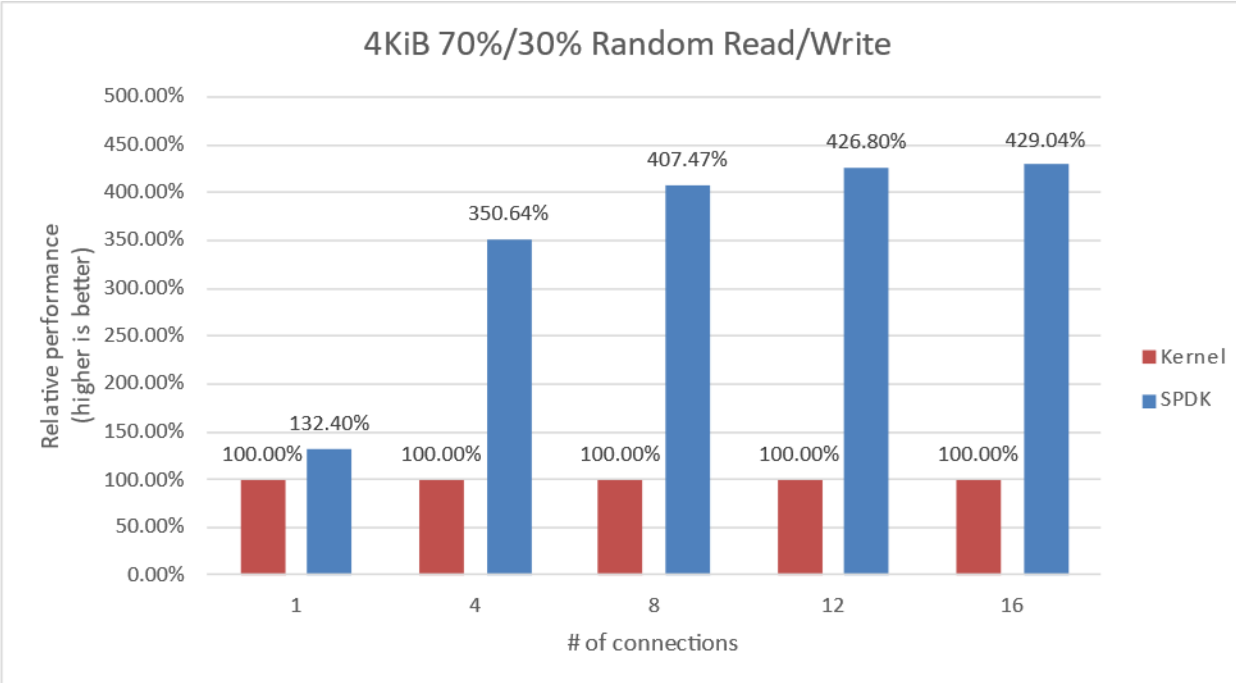

Figure 13: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB Random 70% Reads 30% Writes QD=192 Workload, using Kernel Initiators

Table 28: Linux Kernel NVMe-oF RDMA Target: 4KiB 70% Random Read 30% Random Write, QD=192

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	19362.85	4956.9	544.3	10.4
4	41269.27	10564.9	254.3	28.2
8	40859.01	10459.9	256.7	33.4
12	40095.28	10264.4	261.5	34.4
16	40293.67	10315.2	260.2	34.7

Table 29: SPDK NVMe-oF RDMA Target: 4KiB 70% Random Read 30% Random Write, QD=192

Connections per subsystem	Bandwidth (MiBps)	IOPS (k)	Avg. Latency (usec)	# CPU Cores
1	19799.82	5068.7	533.9	8.1
4	41455.17	10612.5	253.1	8.1
8	40233.56	10299.8	260.7	8.1
12	40121.85	10271.2	261.3	8.1
16	40245.41	10302.8	260.5	8.1

Conclusions

1. SPDK NVMe-oF Target performance peaked at 4 connections for all workloads. Increasing the number of connections per subsystem beyond these values did not result in significant changes to the IOPS or latency.

2. The performance of the Linux Kernel NVMe-oF Target exhibited variations across different workloads and connection counts. For Random Read operations, optimal performance was not consistently achieved at 4 connections per subsystem. Instead, performance varied with QD and number of connections, peaking at 192 QD with 4 connections. For Random Write workloads, the highest performance was found at 192 QD with 16 connections. In contrast, the 70/30 Random Read/Write workload showed optimal performance at a 192 QD with 4 connections. Thus, while 4 connections per subsystem were generally effective, the highest performance metrics were observed at different points depending on the specific workload and configuration.

3. The SPDK NVMe-oF target shows up to 5.09x more IOPS/Core relative to the Linux Kernel NVMe-oF target as the number of connections per subsystem increased.

Summary

This report showcased performance results with SPDK NVMe-oF RDMA Target and Initiator under various test cases, including:

- I/O core scaling
- Average I/O latency
- Performance with increasing number of connections per subsystems

It compared performance results while running the Linux Kernel NVMe-oF RDMA (Target/Initiator) against the accelerated polled-mode driven SPDK NVMe-oF RDMA (Target/Initiator) implementation. Like in the previous reports, throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF Target cores.

It was also observed that the SPDK NVMe-oF Target average latency is up to 2.26 usec lower than Kernel when testing against null bdev based backend. The advantage of SPDK is even greater when comparing NVMe-oF Initiators: the SPDK NVMe-oF RDMA average latency is lower by up to 35.58% lower than Kernel initiator.

The SPDK NVMe-oF Target performed up to 5.09 times better w.r.t IOPS/core than Linux Kernel NVMe-oF target while running 4KiB Random Write workload with increasing number of active connections per NVMe-oF subsystem.

This report provides information regarding methodologies and practices while benchmarking NVMe-oF using SPDK, as well as the Linux Kernel. It should be noted that the performance data showcased in this report is based on specific hardware and software configurations and that performance results may vary depending on different hardware and software configurations.

List of tables

Table 1: Hardware setup configuration – Target system 4
Table 2: Hardware setup configuration – Initiator system 1 5
Table 3: Hardware setup configuration – Initiator system 2 5
Table 4: Test systems BIOS settings 6
Table 5: SPDK NVMe-oF RDMA Target Core Scaling test configuration 9
Table 6: SPDK NVMe-oF RDMA Target Core Scaling results, Random Read IOPS, QD=128 12
Table 7: SPDK NVMe-oF RDMA Target Core Scaling results, Random Write IOPS, QD=64 13
Table 8: SPDK NVMe-oF RDMA Target Core Scaling results, Random Read/Write 70%/30% IOPS, QD=128 14
Table 9: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential Read IOPS, QD=4 15
Table 10: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential Write IOPS, QD=4 15
Table 11: SPDK NVMe-oF RDMA Target Core Scaling results, 128KiB Sequential 70% Read 30% Write IOPS, QD=8 15
Table 12: SPDK NVMe-oF RDMA Initiator Core Scaling test configuration 17
Table 13: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random Read IOPS, QD=64, SPDK Target 6 CPU Cores 19
Table 14: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random Write IOPS, QD=64, SPDK Target 6 CPU Cores 20
Table 15: SPDK NVMe-oF RDMA Initiator Core Scaling results, 4KiB Random 70%/30% Read/Write IOPS, QD=64, SPDK Target 6 CPU Cores 21
Table 16: Linux Kernel vs. SPDK NVMe-oF RDMA Latency test configuration 23
Table 17: SPDK NVMe-oF RDMA Target Latency and IOPS at QD=1, Null Block Device 26
Table 18: Linux Kernel NVMe-oF RDMA Target Latency and IOPS at QD=1, Null Block Device 26
Table 19: SPDK NVMe-oF RDMA Initiator Latency and IOPS at QD=1, Null Block Device 27
Table 20: Linux Kernel NVMe-oF RDMA Initiator Latency and IOPS at QD=1, Null Block Device 27
Table 21: SPDK NVMe-oF RDMA Latency and IOPS at QD=1, Null Block Device 28
Table 22: Linux Kernel NVMe-oF RDMA Latency and IOPS at QD=1, Null Block Device 28
Table 23: NVMe-oF RDMA Performance with increasing number of connections test configuration 30
Table 24: Linux Kernel NVMe-oF RDMA Target: 4KiB 100% Random Reads, QD=192 32
Table 25: SPDK NVMe-oF RDMA Target: 4KiB 100% Random Reads, QD=192 32
Table 26: Linux Kernel NVMe-oF RDMA Target: 4KiB 100% Random Writes, QD=128 33
Table 27: SPDK NVMe-oF RDMA Target: 4KiB 100% Random Writes, QD=128 33

Table 28: Linux Kernel NVMe-oF RDMA Target: 4KiB 70% Random Read 30% Random Write, QD=192 34

Table 29: SPDK NVMe-oF RDMA Target: 4KiB 70% Random Read 30% Random Write, QD=192 34

List of figures

Figure 1: High-Level NVMe-oF RDMA performance testing setup 8

Figure 2: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read workload at QD=128..12

Figure 3: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=64 13

Figure 4: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KiB Random 70/30 Read/Write workload at QD=128 14

Figure 5: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read QD=64 workload 19

Figure 6: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=64 20

Figure 7: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KiB Random 70% Read 30% Write Workload at QD=64..21

Figure 8: SPDK vs. Kernel NVMe-oF RDMA Target average I/O latency for various workloads run using the Kernel Initiator..26

Figure 9: SPDK vs. Kernel NVMe-oF RDMA Initiator average I/O latency for various workloads against SPDK Target..27

Figure 10: SPDK vs. Kernel NVMe-oF RDMA solutions average I/O Latency for various workloads against SPDK Target..28

Figure 11: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB 100% Random Reads QD=192, using the Kernel Initiator 32

Figure 12: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB 100% Random Writes QD=128 workload, using Kernel Initiators 33

Figure 13: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target IOPS/Core for 4KiB Random 70% Reads 30% Writes QD=192 Workload, using Kernel Initiators 34

Appendix A – Test Case 1 & 2 SPDK NVMe-oF Initiator bdev configuration

Initiator system 1


```
{
  "subsystems": [
    {
      "subsystem": "bdev",
      "config": [
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme0",
            “trtype”: “rdma”,
            "traddr": "20.0.0.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode0",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme1",
            “trtype”: “rdma”,
            "traddr": "20.0.0.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode1",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme2",
            “trtype”: “rdma”,
            "traddr": "20.0.0.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode2",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme3",
            “trtype”: “rdma”,
            "traddr": "20.0.0.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode3",
```

```
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme4",
            “trtype”: “rdma”,
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode4",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme5",
            “trtype”: “rdma”,
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode5",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme6",
            “trtype”: “rdma”,
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode6",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme7",
            “trtype”: “rdma”,
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode7",
            "adrfam": "IPv4"
          }
        }
      ]
    }
  ]
```

Initiator system 2

```
{
  "subsystems": [
    {
      "subsystem": "bdev",
```

```
"config": [
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme0",
      “trtype”: “rdma”,
      "traddr": "10.0.0.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode0",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme1",
      “trtype”: “rdma”,
      "traddr": "10.0.0.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode1",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme2",
      “trtype”: “rdma”,
      "traddr": "10.0.0.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode2",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme3",
      “trtype”: “rdma”,
      "traddr": "10.0.0.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode3",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme4",
      “trtype”: “rdma”,
      "traddr": "10.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode4",
      "adrfam": "IPv4"
    }
  },
  {
```

```
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme5",
            “trtype”: “rdma”,
            "traddr": "10.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode5",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme6",
            “trtype”: “rdma”,
            "traddr": "10.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode6",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme7",
            “trtype”: “rdma”,
            "traddr": "10.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode7",
            "adrfam": "IPv4"
          }
        }
      ]
    }
  ]
}
```

Appendix B – Test Case 3 SPDK NVMe-oF Initiator bdev configuration

```
{
  "subsystems": [
    {
      "subsystem": "bdev",
      "config": [
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme0",
            "trtype": "tcp",
            "traddr": "20.0.0.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode0",
            "adrfam": "IPv4"
          }
        }
      ]
    }
  ]
}
```

Appendix C - Kernel NVMe-oF RDMA Target configuration

Example Linux Kernel NVMe-oF RDMA Target configuration for Test Case 4.

```
{
  "ports": [
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "trtype": "rdma"
      },
      "portid": 1,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode1"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4421",
        "trtype": "rdma"
      },
      "portid": 2,
      "referrals": [],
```

```
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode2"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4422",
        "trtype": "rdma"
      },
      "portid": 3,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode3"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4423",
        "trtype": "rdma"
      },
      "portid": 4,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode4"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4424",
        "trtype": "rdma"
      },
      "portid": 5,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode5"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4425",
        "trtype": "rdma"
      },
      "portid": 6,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode6"
      ]
    },
    {
```

```
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4426",
        "trtype": "rdma"
      },
      "portid": 7,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode7"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4427",
        "trtype": "rdma"
      },
      "portid": 8,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode8"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvcid": "4428",
        "trtype": "rdma"
      },
      "portid": 9,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode9"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvcid": "4429",
        "trtype": "rdma"
      },
      "portid": 10,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode10"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvcid": "4430",
        "trtype": "rdma"
```

```
      },
      "portid": 11,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode11"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvcid": "4431",
        "trtype": "rdma"
      },
      "portid": 12,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode12"
      ]
    },
{
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4432",
        "trtype": "rdma"
      },
      "portid": 13,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode13"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4433",
        "trtype": "rdma"
      },
      "portid": 14,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode14"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4434",
        "trtype": "rdma"
      },
      "portid": 15,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode15"
```

```
        ]
      },
      {
        "addr": {
          "adrfam": "ipv4",
          "traddr": "10.0.1.1",
          "trsvcid": "4435",
          "trtype": "rdma"
        },
        "portid": 16,
        "referrals": [],
        "subsystems": [
          "nqn.2018-09.io.spdk:cnode16"
        ]
      }
    ],
  "hosts": [],
   "subsystems": [
      {
        "allowed_hosts": [],
        "attr": {
          "allow_any_host": "1",
          "version": "1.3"
        },
        "namespaces": [
          {
            "device": {
              "path": "/dev/nvme0n1",
              "uuid": "b53be81d-6f5c-4768-b3bd-203614d8cf20"
            },
            "enable": 1,
            "nsid": 1
          }
        ],
        "nqn": "nqn.2018-09.io.spdk:cnode1"
      },
      {
        "allowed_hosts": [],
        "attr": {
          "allow_any_host": "1",
          "version": "1.3"
        },
        "namespaces": [
          {
            "device": {
              "path": "/dev/nvme1n1",
              "uuid": "12fcf584-9c45-4b6b-abc9-63a763455cf7"
            },
            "enable": 1,
            "nsid": 2
          }
        ],
        "nqn": "nqn.2018-09.io.spdk:cnode2"
      },
      {
        "allowed_hosts": [],
        "attr": {
```

```
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme2n1",
            "uuid": "ceae8569-69e9-4831-8661-90725bdf768d"
          },
          "enable": 1,
          "nsid": 3
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode3"
    },
   {
     "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme3n1",
            "uuid": "39f36db4-2cd5-4f69-b37d-1192111d52a6"
          },
          "enable": 1,
          "nsid": 4
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode4"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme4n1",
            "uuid": "984aed55-90ed-4517-ae36-d3afb92dd41f"
          },
          "enable": 1,
          "nsid": 5
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode5"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
```

```
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme5n1",
            "uuid": "d6d16e74-378d-40ad-83e7-b8d8af3d06a6"
          },
          "enable": 1,
          "nsid": 6
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode6"
    },
   {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme6n1",
            "uuid": "a65dc00e-d35c-4647-9db6-c2a8d90db5e8"
          },
          "enable": 1,
          "nsid": 7
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode7"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme7n1",
            "uuid": "1b242cb7-8e47-4079-a233-83e2cd47c86c"
          },
          "enable": 1,
          "nsid": 8
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode8"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
```

```
          "path": "/dev/nvme8n1",
          "uuid": "f12bb0c9-a2c6-4eef-a94f-5c4887bbf77f"
        },
        "enable": 1,
        "nsid": 9
      }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode9"
  },
  {
    "allowed_hosts": [],
    "attr": {
      "allow_any_host": "1",
      "version": "1.3"
    },
    "namespaces": [
      {
        "device": {
          "path": "/dev/nvme9n1",
          "uuid": "40fae536-227b-47d2-bd74-8ab76ec7603b"
        },
        "enable": 1,
        "nsid": 10
      }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode10"
  },
  {
    "allowed_hosts": [],
    "attr": {
      "allow_any_host": "1",
      "version": "1.3"
    },
    "namespaces": [
      {
        "device": {
          "path": "/dev/nvme10n1",
          "uuid": "b9756b86-263a-41cf-a68c-5cfb23c7a6eb"
        },
        "enable": 1,
        "nsid": 11
      }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode11"
  },
  {
    "allowed_hosts": [],
    "attr": {
      "allow_any_host": "1",
      "version": "1.3"
    },
    "namespaces": [
      {
        "device": {
          "path": "/dev/nvme11n1",
          "uuid": "9d7e74cc-97f3-40fb-8e90-f2d02b5fff4c"
        },
```

```
          "enable": 1,
          "nsid": 12
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode12"
    },
{
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme12n1",
            "uuid": "d3f4017b-4f7d-454d-94a9-ea75ffc7436d"
          },
          "enable": 1,
          "nsid": 13
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode13"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme13n1",
            "uuid": "6b9a65a3-6557-4713-8bad-57d9c5cb17a9"
          },
          "enable": 1,
          "nsid": 14
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode14"
    },
    {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme14n1",
            "uuid": "ed69ba4d-8727-43bd-894a-7b08ade4f1b1"
          },
          "enable": 1,
          "nsid": 15
        }
```

```
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode15"
    },
  {
      "allowed_hosts": [],
      "attr": {
        "allow_any_host": "1",
        "version": "1.3"
      },
      "namespaces": [
        {
          "device": {
            "path": "/dev/nvme15n1",
            "uuid": "5b8e9af4-0ab4-47fb-968f-b13e4b607f4e"
          },
          "enable": 1,
          "nsid": 16
        }
      ],
      "nqn": "nqn.2018-09.io.spdk:cnode16"
    }
  ]
}
```

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates.

Your costs and results may vary.

No product or component can be absolutely secure.

Intel technologies may require enabled hardware, software or service activation.

§