SPDK NVMe-oF TCP (Target & Initiator) Performance Report Release 23.09

Mellanox ConnectX-5 version

Testing Date: November 2023

Performed by:

Jaroslaw Chachulski (jaroslawx.chachulski@intel.com)

Karol Latecki (karol.latecki@intel.com)

Acknowledgments:

Krzysztof Karaś (krzysztof.karas@intel.com)

Contents

Audience and Purpose

This report is intended for people who are interested in evaluating SPDK NVMe-oF (Target & Initiator) performance. This report contains SPDK NVMe-oF Target and Initiator performance characteristics and provides comparison data between SPDK and its Kernel NVMe-oF Target and Initiator counterparts. This report covers the TCP transport only.

The purpose of reporting these tests 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 & Initiator).

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: 0xd000389
Memory	16 x 32GB SK Hynix HMA84GR7DJR4N-XN, DDR4, 3200MHz Total of 512GB
Operating System	Fedora 37
BIOS	1.4b
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
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 Mellanox ConnectX-5 NICs. Both ports connected. 2 NICs per CPU socket.

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: 0xd00037b
Memory	16 x 32GB Micron 36ASF4G72PZ-3G2J3, DDR4, 3200MHz Total 512GBs
Operating System	Fedora 37
BIOS	SE5C620.86B.01.01.0007.2210270543
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
Storage	**OS:** 1x 250GB Crucial CT250MX500SSD1
NIC	2x 100GbE Mellanox ConnectX-5 Ex NIC. Single port on each NIC connected to Target server. 1 NIC per CPU socket.

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: 0xd00037b
Memory	16 x 32GB Micron 36ASF4G72PZ-3G2J3, DDR4, 3200MHz Total 512GBs
Operating System	Fedora 37
BIOS	SE5C620.86B.01.01.0006.2207150335
Linux kernel version	6.0.18-300.fc37.x86_64 Spectre-meltdown mitigations enabled
Storage	**OS:** 1x 250GB Crucial CT250MX500SSD1
NIC	2x 100GbE Mellanox ConnectX-5 Ex NIC. Single port on each NIC connected to Target server. 1 NIC per CPU socket.

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

Software

The tests and evaluations conducted in this report are based on the Storage Performance Development Kit (SPDK) version 23.09.

For I/O benchmarking tests, fio version 3.28 was utilized.

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.

TCP configuration

Note that the SPDK NVMe-oF target and initiator use the Linux Kernel TCP stack. We tuned the Linux Kernel TCP stack for storage workloads over 100 Gbps NIC by settings the following parameters using sysctl:

```
net.core.busy_poll = 0
net.core.busy_read = 0
net.core.somaxconn = 4096
net.core.netdev_max_backlog = 8192
net.ipv4.tcp_max_syn_backlog = 16384
net.core.rmem_max = 268435456
net.core.wmem_max= 268435456
net.ipv4.tcp_mem = "268435456 268435456 268435456"
net.ipv4.tcp_rmem = "8192 1048576 33554432"
net.ipv4.tcp_wmem = "8192 1048576 33554432"
net.ipv4.route.flush = 1
vm.overcommit_memory = 1
```

SPDK iobufs

SPDK introduced a common pool of buffers to be used across libraries in SPDK called "iobuf". Over time more components are being converted to share the "iobufs". The default counts of elements are defined at values common for some use cases. Depending on the test scenario those values might need to be increased via "iobuf_set_options" RPC. Please see "spdk/scripts/calc-iobuf.py" for guidance on minimum “iobuf” pool sizes.

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™), Fibre 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 close to the CPU.

The SPDK NVMe-oF target and initiator uses the underlying transport layer API which in case of TCP are POSIX sockets. 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 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 and four 100GbE Mellanox ConnectX®-5 NICs connected to provide up to 400GbE of network bandwidth. Each Initiator system has two Mellanox ConnectX®-5 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 io_uring 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 (systat).

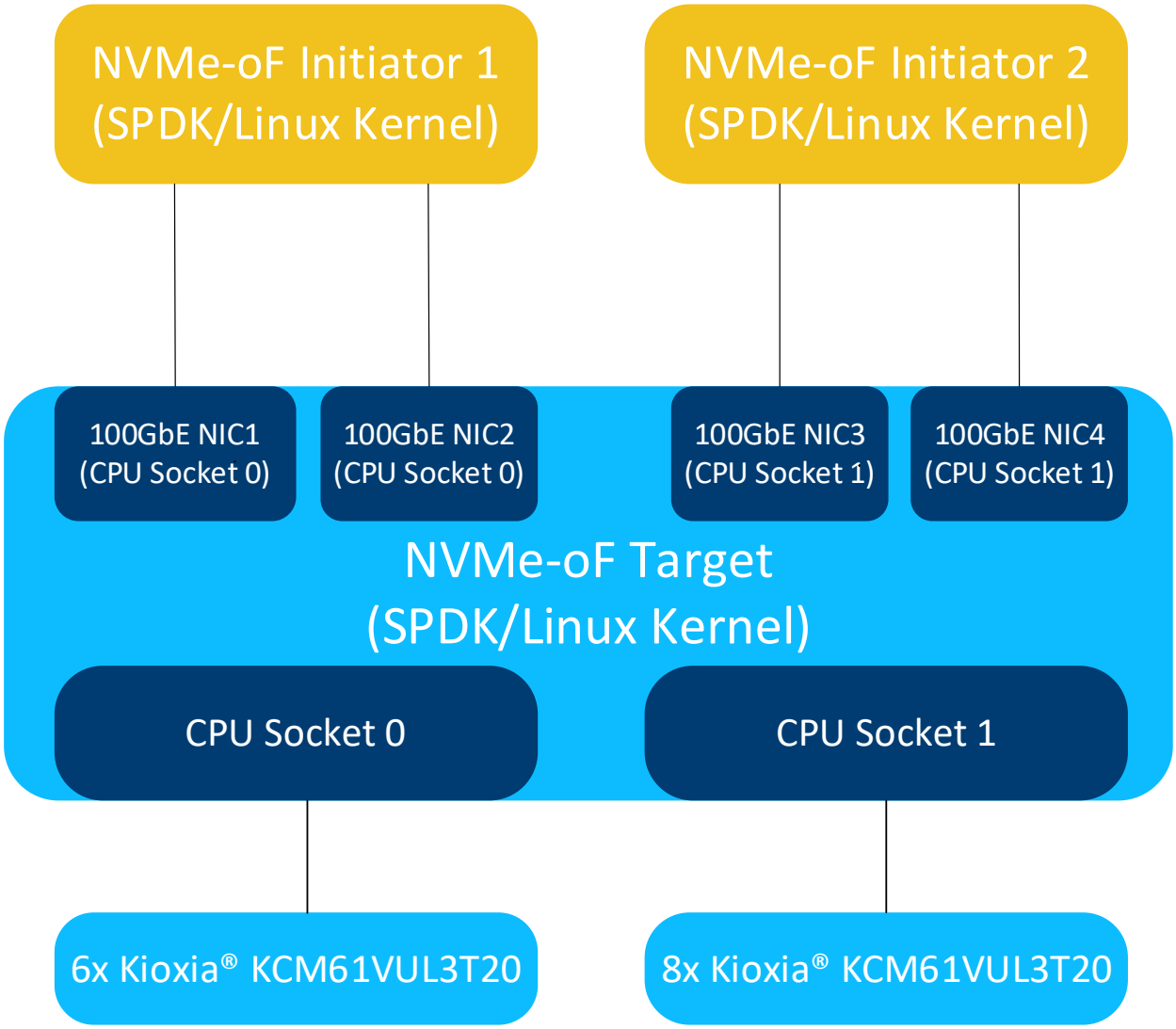

Figure 1: High-Level NVMe-oF TCP performance testing setup

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

This test case was performed in order to understand the performance of SPDK TCP NVMe-oF target with I/O core scaling.

The SPDK NVMe-oF TCP 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 NIC. 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 1, 4, 8, 12, 16, 24, 32, 40 and 48 CPU cores. We ran the following workloads on each initiator:

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

We scaled the fio jobs using fio parameter numjob=4 in order to generate more I/O requests. When using the SPDK fio plugin it is important to note the difference between the fio I/O depth parameter and the NVMe device I/O depth because we can configure an fio job to send I/Os to more than one NVMe device and we can also scale the number of fio jobs using the numjobs parameter. The parameter values presented in the table below are actual queue depths used for each of the NVMe devices specified by the filename. 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.

For detailed configuration please refer to the table below. The actual SPDK NVMe-oF configuration was done using JSON-RPC and the table contains the sequence of commands used by spdk/scripts/rpc.py script rather than a configuration file. 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 the 3 runs. For workloads which need preconditioning, 4KiB Random Read and 4KiB Random 70%/30% Read /Write we ran preconditioning once before running all of the workload to force the NVMe devices into a steady state so that we get consistent results.

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

Item	Description
Test Case	Test 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. **Set iobuf buffer pool options:** iobuf_set_options --small-pool-count 32767 --large-pool-count 16383 **Enable zero-copy send on Target side before initializing all other subsystems.** sock_impl_set_options –impl=posix –enable-zerocopy-send-server (note: zerocopy for Client side is disabled by default)

	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
	
	Create a TCP transport:
	nvmf_create_transport -t TCP
	{
	"trtype": "TCP",
	"max_queue_depth": 128,
	"max_io_qpairs_per_ctrlr": 127,
	"in_capsule_data_size": 4096,
	"max_io_size": 131072,
	"io_unit_size": 131072,
	"max_aq_depth": 128,
	"num_shared_buffers": 8192,
	"buf_cache_size": 32,
	"dif_insert_or_strip": false,
	"c2h_success": true,
	"sock_priority": 0,
	"abort_timeout_sec": 1
	}
	
	Create NVMe-oF subsystems and add NVMe bdevs as namespaces:
	for i in $(seq 1 14); do
	nvmf_subsystem_create nqn.2018-09.io.spdk:cnode${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 tcp \
	-f ipv4 -s 4420 -a ${ip}
	((i++))
	done
	done

SPDK NVMe-oF Initiator - FIO plugin configuration	**BDEV.conf:** See appendix B. **FIO.conf** [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={192, 256, 384} time_based=1 numjobs=4 ramp_time=60 runtime=300 [filename0] filename=Nvme0n1 [filename1] filename=Nvme1n1 [filename2] filename=Nvme2n1 [filename3] filename=Nvme3n1 [filename4] filename=Nvme4n1 [filename5] filename=Nvme5n1 [filename6] filename=Nvme6n1

4KiB Random Read Results

Table 6: SPDK NVMe-oF TCP Target Core Scaling results, Random Read IOPS, QD=384

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	653.6	21.42	8216.1
4 cores	2495.4	81.77	2172.0
8 cores	5722.1	187.50	932.6
12 cores	8233.8	269.81	644.2
16 cores	9978.0	326.96	529.8
24 cores	10728.3	351.55	490.9
32 cores	10959.9	359.13	480.5
40 cores	11013.0	360.87	478.5
48 cores	11062.9	362.51	476.2

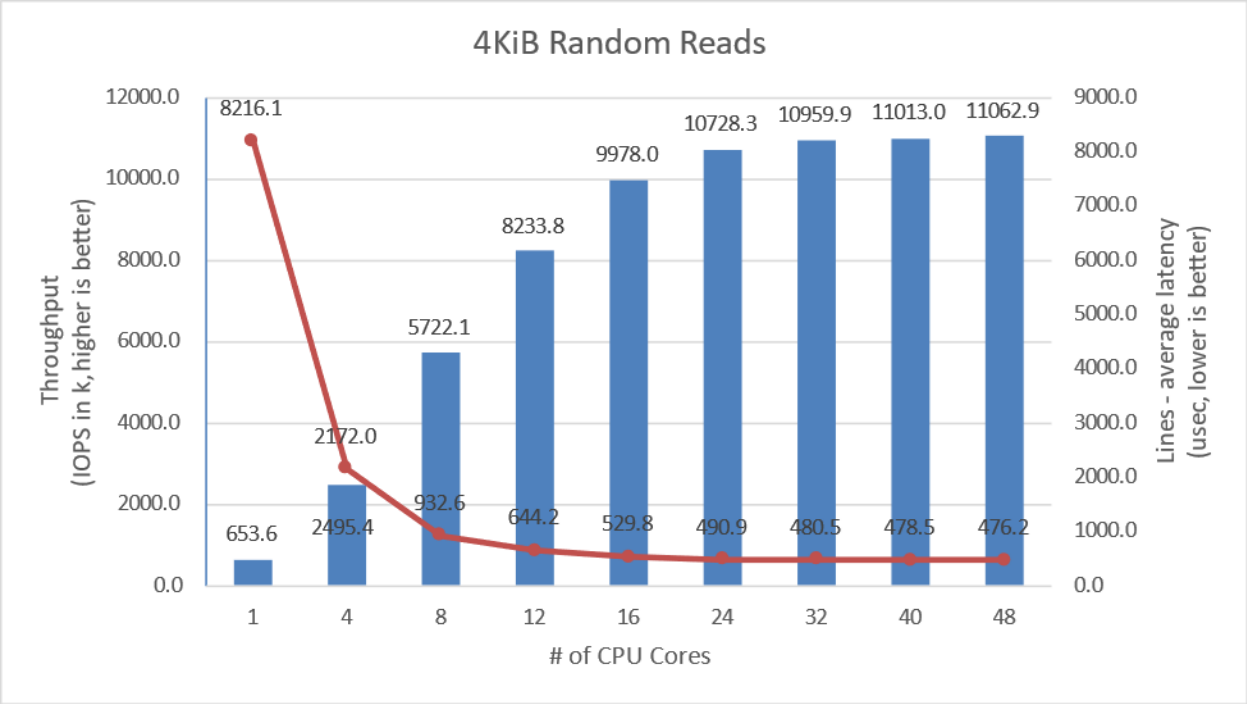

Figure 2: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read workload at QD = 384

4KiB Random Write Results

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.

Table 7: SPDK NVMe-oF TCP Target Core Scaling results, Random Write IOPS, QD=128

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	398.4	13.06	4518.8
4 cores	1632.7	53.50	1092.8
8 cores	3080.1	100.93	577.0
12 cores	4096.9	134.25	433.6
16 cores	4902.5	160.64	361.1
24 cores	5484.4	179.71	322.7
32 cores	5689.2	186.42	311.1
40 cores	5672.3	185.87	312.8
48 cores	5526.8	181.10	320.8

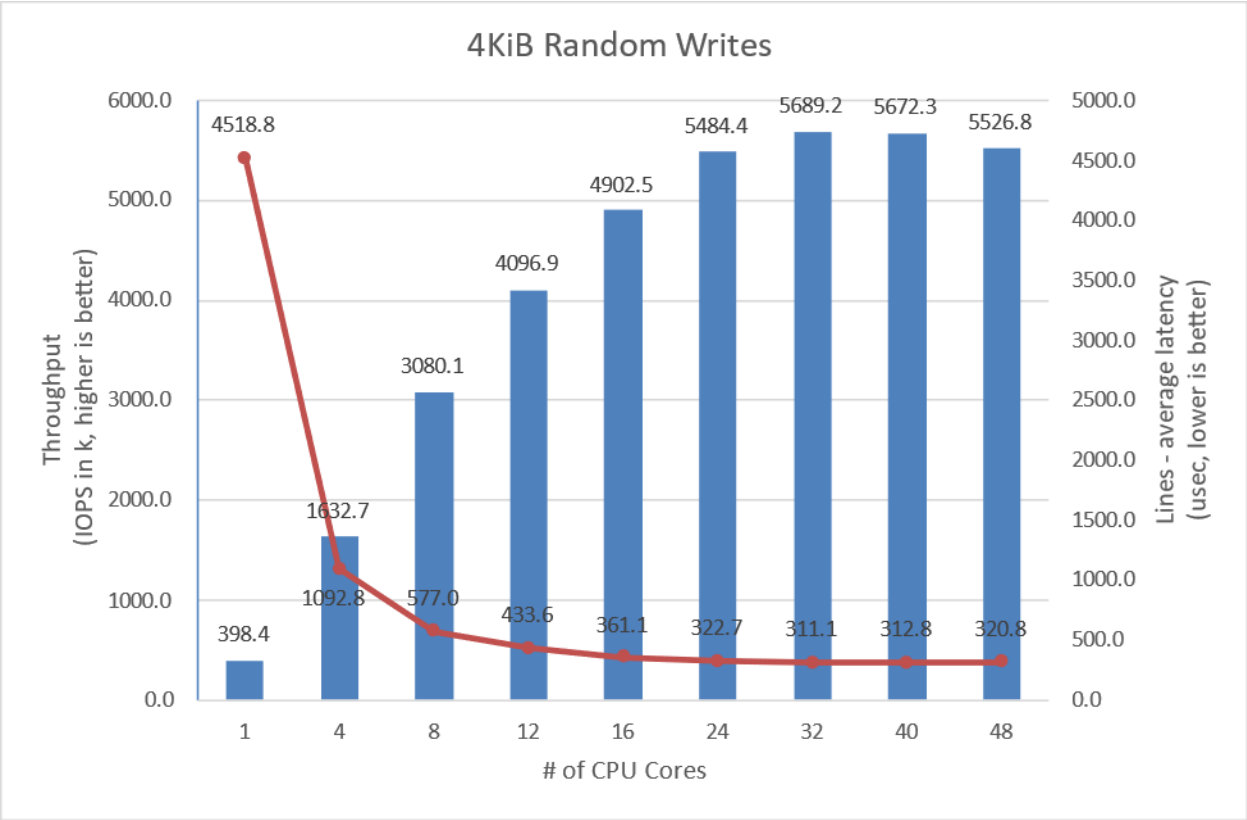

Figure 3: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=384

4KiB Random Read-Write Results

Table 8: SPDK NVMe-oF TCP Target Core Scaling results, Random Read/Write 70%/30% IOPS, QD=384

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	474.6	15.55	11323.5
4 cores	1970.8	64.58	2724.3
8 cores	4066.5	133.25	1318.4
12 cores	5983.3	196.06	895.3
16 cores	7480.0	245.11	715.3
24 cores	9341.4	306.10	571.7
32 cores	9765.2	319.99	546.5
40 cores	10032.1	328.73	532.1
48 cores	10081.4	330.35	529.4

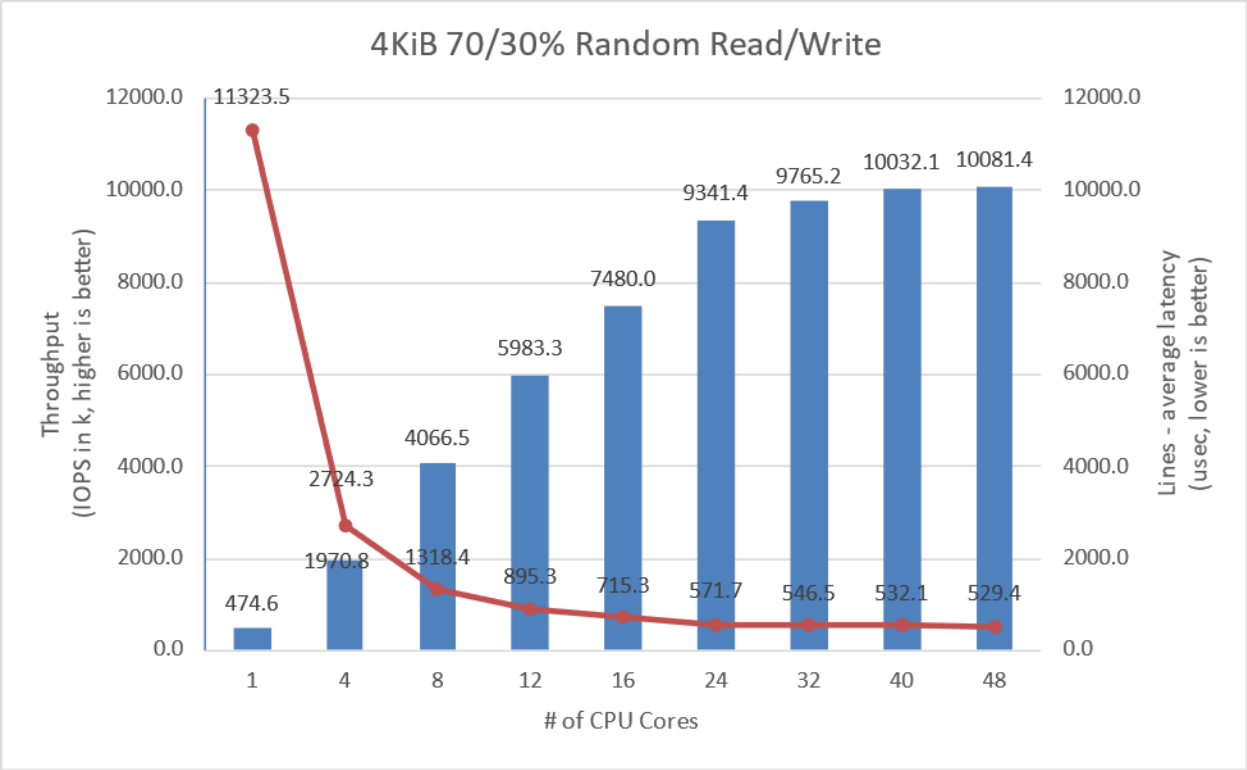

Figure 4: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 4KiB Random 70/30 Read/Write workload at QD=384

Large Sequential I/O Performance

We measured the performance of large block I/O workloads by performing sequential I/Os of size 128KiBs at queue depth 32. We used iodepth=32 because higher queue depth resulted in negligible bandwidth gain and a significant increase in the latency. The rest of the fio configuration is similar to the 4KiB test case in the previous part of this document.

Table 9: SPDK NVMe-oF TCP Target Core Scaling results, 128KiB Sequential Read IOPS, QD=16

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	153.5	160.97	1466.0
4 cores	358.3	375.72	624.4
8 cores	358.8	376.27	623.5
12 cores	358.8	376.28	623.5

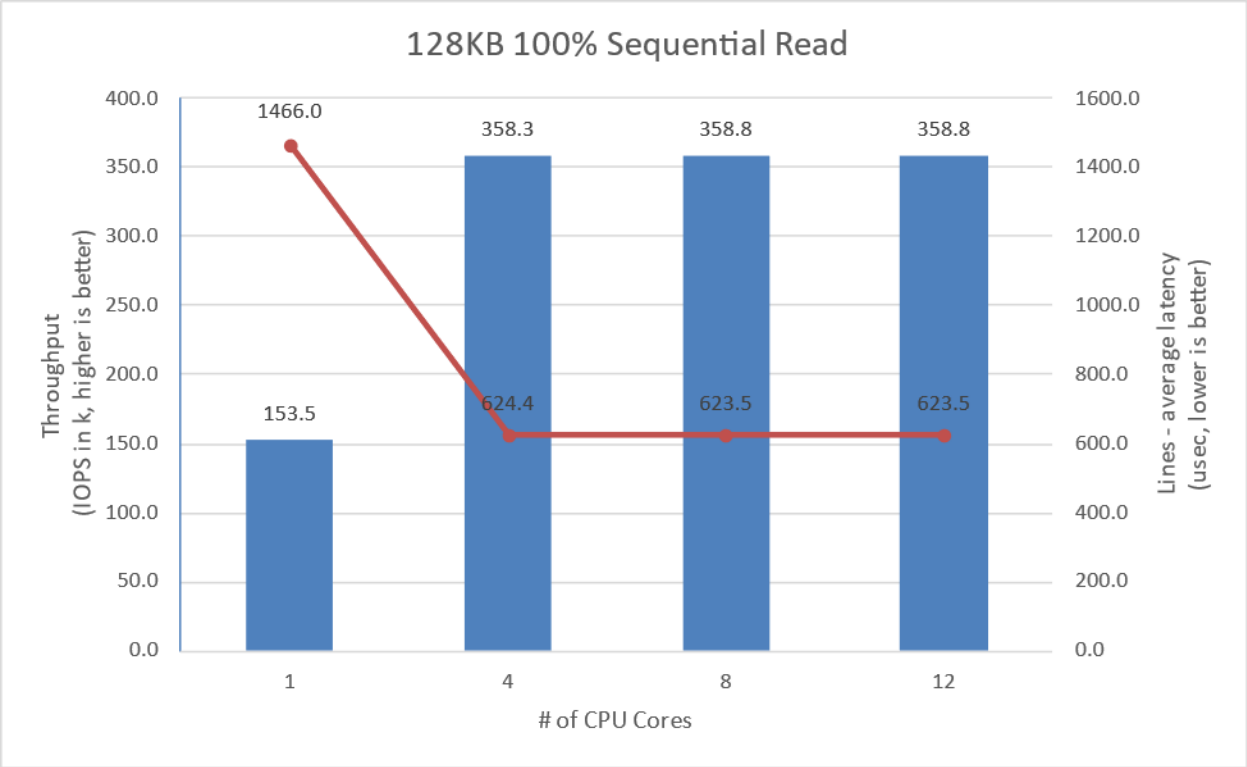

Figure 5: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KiB 100% Sequential Read Workload at QD=16 and initiator FIO numjobs=4

Table 10: SPDK NVMe-oF TCP Target Core Scaling results, 128KiB Sequential Write IOPS, QD=16

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	24.0	25.20	9334.9
4 cores	88.2	92.43	2541.3
8 cores	164.0	171.96	1366.7
12 cores	204.2	214.11	1098.3
16 cores	250.7	262.91	894.1
24 cores	284.7	298.52	787.3
32 cores	316.4	331.73	707.8
40 cores	333.3	349.44	672.2
48 cores	302.8	317.48	743.2

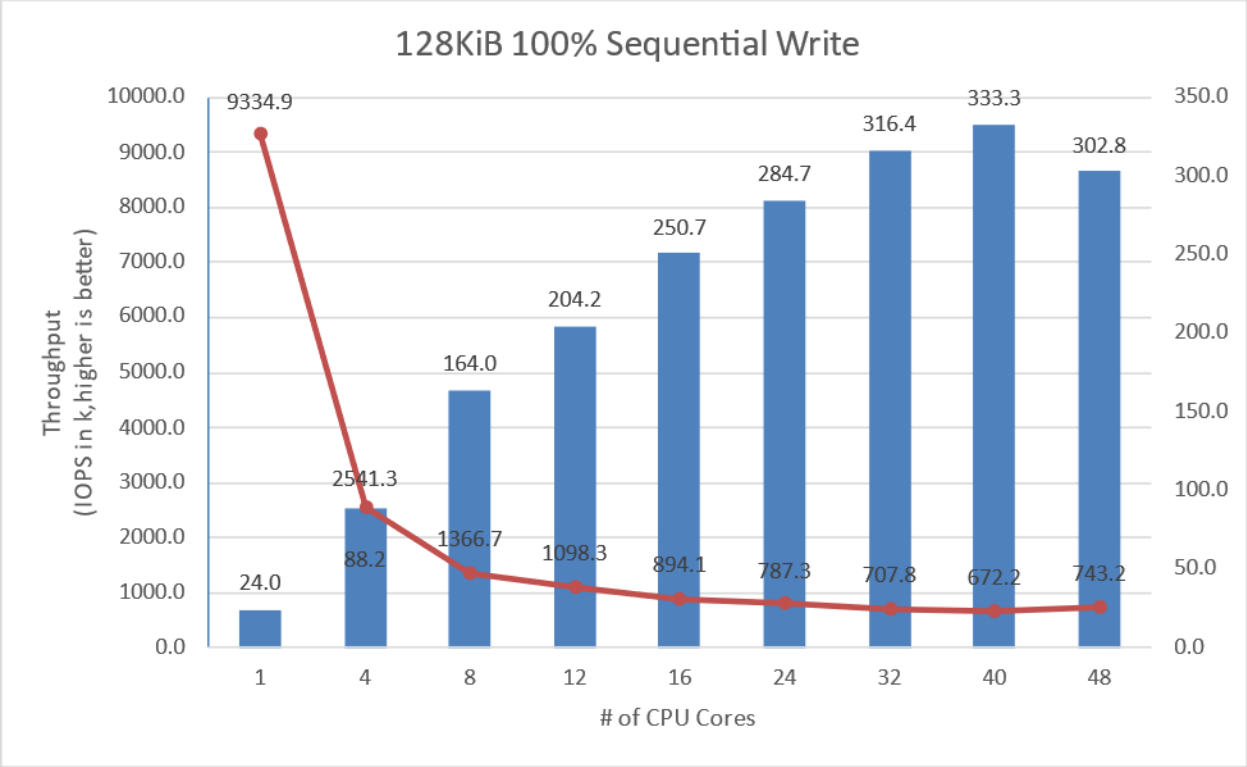

Figure 6: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KiB 100% Sequential Write Workload at QD=16 and Initiator FIO numjobs=4

Table 11: SPDK NVMe-oF TCP Target Core Scaling results, 128KiB Sequential 70% Read 30% Write IOPS, QD=16

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	51.5	53.96	4386.4
4 cores	179.3	188.04	1250.2
8 cores	306.8	321.70	730.1
12 cores	369.3	387.24	605.6
16 cores	392.7	411.81	569.8
24 cores	418.7	439.00	534.1
32 cores	422.3	442.78	529.5

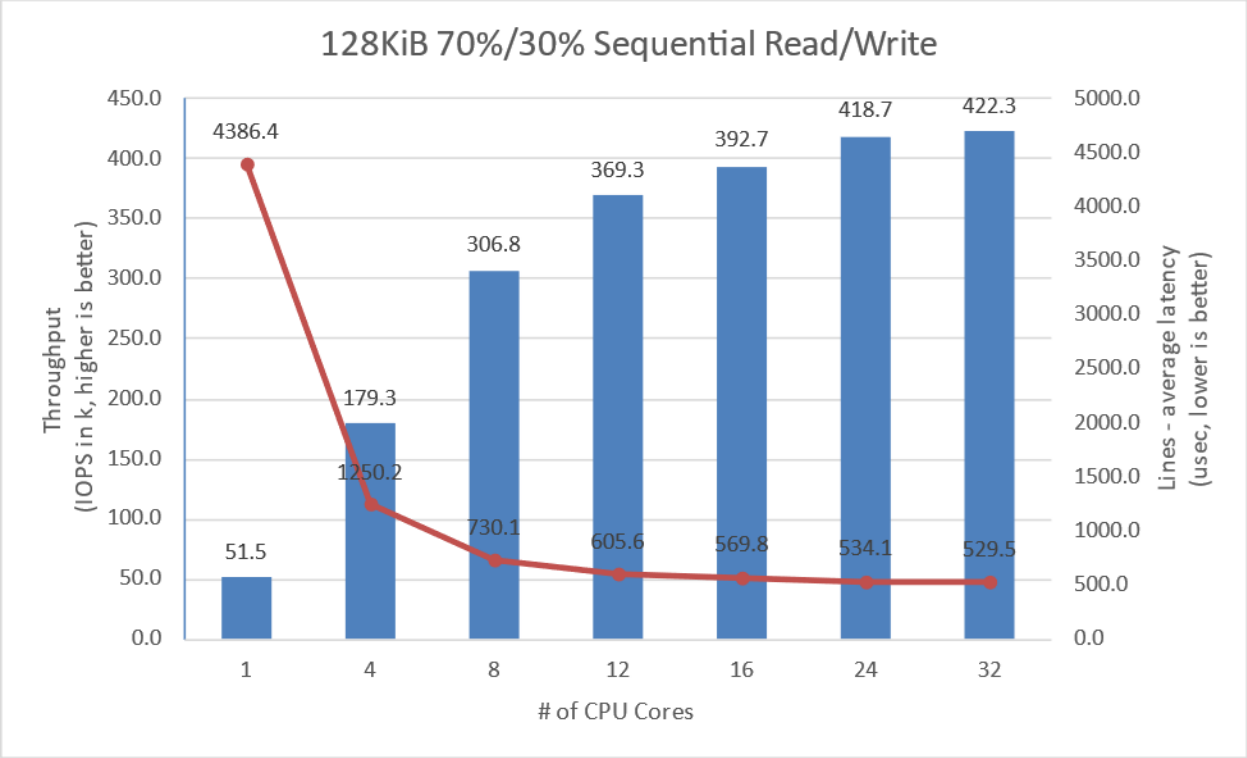

Figure 7: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KiB Sequential 70% Read 30% Write Workload at QD=16 and Initiator FIO numjobs=4

Conclusions

1. The SPDK NVMe-oF TCP Target IOPS throughput scaling is close to linear with addition of CPU cores for 4KiB Random Read workload up to 16 CPU cores, reaching 326 Gbps bandwidth and 9.9 million IOPS. Adding more CPUs to Target configuration results in non-linear performance gains peaking at about 11 million IOPS at 40 CPU cores, reaching 400GbE network link saturation.

2. For the 4KiB Random Write workload 100GbE link is saturated at 8 CPU cores. Performance scales up almost linearly to 12 CPU cores reaching 4.1 million IOPS. Adding more CPUs to Target configuration results in non-linear performance gains peaking at about 5.7 million IOPS at 32 CPU cores.

3. 4KiB 70/30% Random Read/Write workload throughput scales linearly up to 24 CPU cores reaching 9.3 million IOPS. Increasing the number of CPU cores beyond 24 results in non-linear performance improvement, peaking at about 10 million IOPS at 40 CPU cores.

4. The best trade-off between CPU efficiency and network saturation is when the Target is configured with 24CPU cores. The performance we achieved with these configurations allows for full (or nearly) saturation of a 400Gbps link between Target and Initiators for all Random Read and Random Read/Write workloads, and 200Gbps link saturation for Random Write workload.

5. For the 4KiB Random Write workload, we did not precondition the drives. If preconditioned, the NVMe drives would max out at about 4.9 million IOPS. Not preconditioning the drives allowed us to artificially increase their throughput and serve more IO requests than usual.

6. The throughput of large block workloads scaled up with addition of CPU cores reaching peak performance at different CPU core counts. For the 128K Sequential Reads workload, the peak throughput of 358.8 Gbps was observed at 8 CPU cores. For the 128K Sequential Writes, the throughput scaled linearly to 316.4 Gbps at 32 cores and reached peak performance of 333.3 Gbps at 40 CPU cores. For the 128K Sequential 70/30 Read/Write workload, the scaling was linear up to 8 CPU cores reached peak performance of 422.3 Gbps at 32 cores.

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

This test case was performed in order to understand the performance of SPDK NVMe-oF TCP Initiator as the number of CPU cores is scaled up.

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 TCP Initiator. The Initiator was connected to Target server with two 100 Gbps network links.

The SPDK NVMe-oF TCP Target was configured similarly as in test case 1, using 24 cores. We used 24 CPU cores based on results of the previous test case which show that the target can easily serve about 6 million IOPS for all workloads, which is enough IOPS to saturate 200 Gbps network connection.

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 48 CPUs. For detailed fio job configuration see table below. Fio was run with following workloads:

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

It is important to note that fio IO depth 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 TCP Initiator Core Scaling test configuration

Item	Description
Test Case	Test SPDK NVMe-oF TCP Initiator I/O core scaling
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 24 CPU cores.
SPDK NVMe-oF Initiator 1 - FIO plugin configuration	**FIO.conf** **For X*4 CPU (up to 48) 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={192, 256, 384}

	time_based=1 ramp_time=60 runtime=300 numjobs=X [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 [filename1] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 [filename2] filename=Nvme8n1 filename=Nvme9n1 filename=Nvme10n1 [filename3] filename=Nvme11n1 filename=Nvme12n1 filename=Nvme13n1

4KiB Random Read Results

Table 13: SPDK NVMe-oF TCP Initiator Core Scaling results, 4KiB Random Read IOPS, QD=256

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	444.6	14.57	8028.3
4 cores	2012.8	65.95	1755.4
8 cores	3215.2	105.36	1098.9
12 cores	4282.0	140.31	819.3
16 cores	5108.0	167.38	684.4
24 cores	5696.3	186.66	615.5
32 cores	5694.4	186.59	619.7
40 cores	5683.3	186.23	622.7
48 cores	5688.5	186.40	622.7

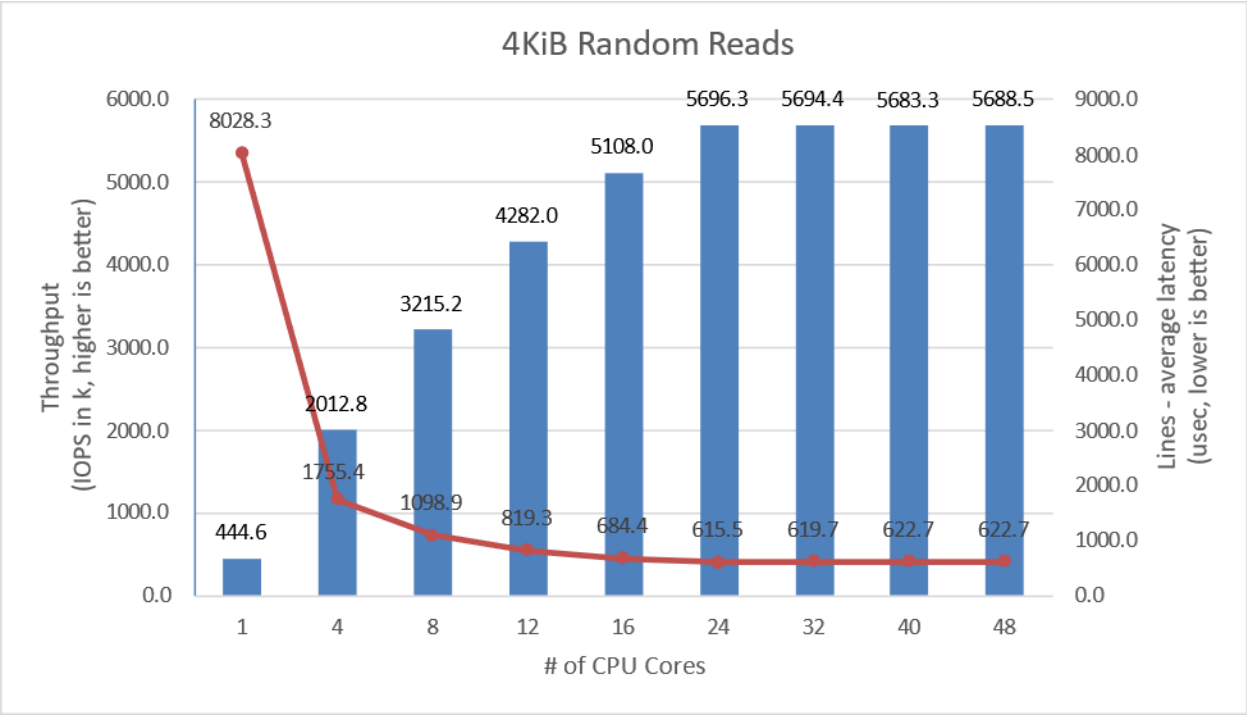

Figure 8: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Read QD=256 workload

4KiB Random Write Results

Table 14: SPDK NVMe-oF TCP Initiator Core Scaling results, 4KiB Random Write IOPS, QD=128

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	620.6	20.33	2026.8
4 cores	2809.8	92.07	554.9
8 cores	3551.3	116.37	484.1
12 cores	3588.3	117.58	489.1
16 cores	3406.4	111.62	518.6
24 cores	3392.5	111.17	522.6
32 cores	3290.8	107.83	541.6
40 cores	3279.7	107.47	540.4
48 cores	3230.5	105.86	555.2

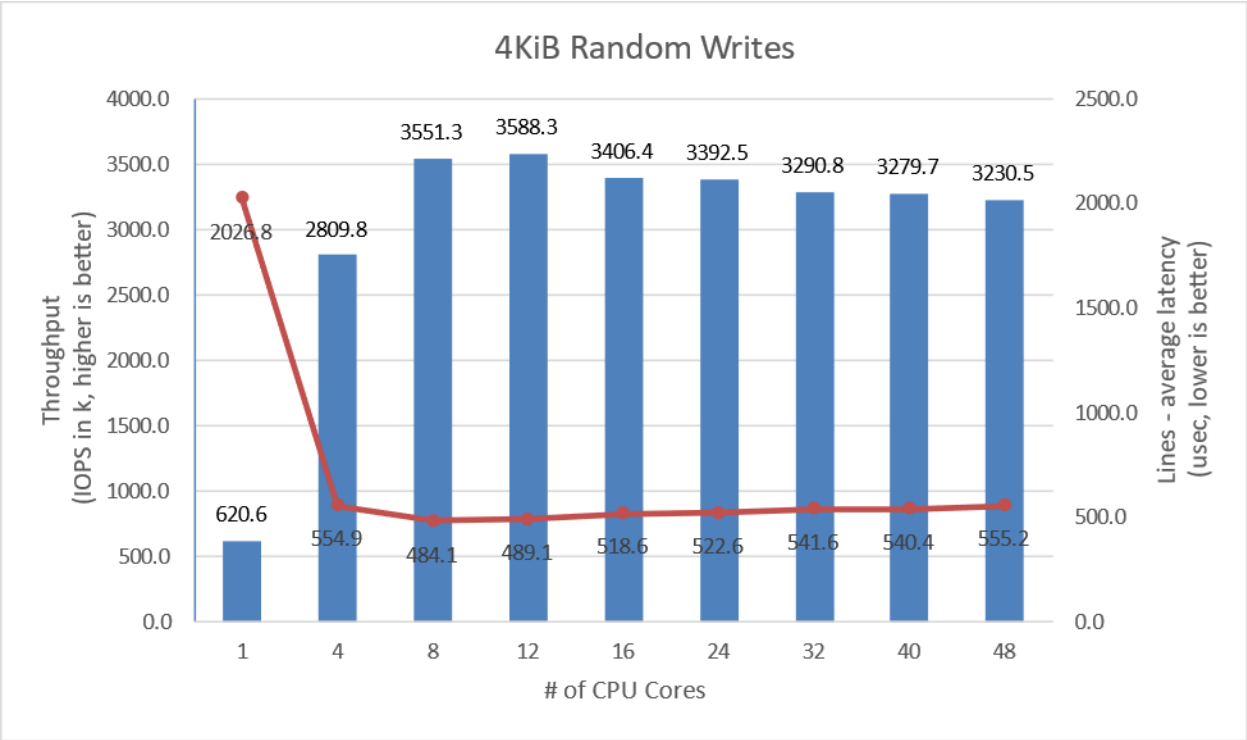

Figure 9: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KiB 100% Random Write Workload at QD=128

4KiB Random Read-Write Results

Table 15: SPDK NVMe-oF TCP Initiator Core Scaling results, 4KiB Random 70%/30% Read/Write IOPS, QD=256

# of Cores	Throughput (IOPS k)	Bandwidth (Gbps)	Avg. Latency (usec)
1 core	449.8	14.74	7927.6
4 cores	2143.9	70.25	1664.6
8 cores	3434.6	112.55	1029.7
12 cores	4659.0	152.67	754.4
16 cores	5559.5	182.17	630.2
24 cores	6320.0	207.09	556.2
32 cores	6384.6	209.21	553.8
40 cores	6303.8	206.56	561.8
48 cores	6124.8	200.70	579.3

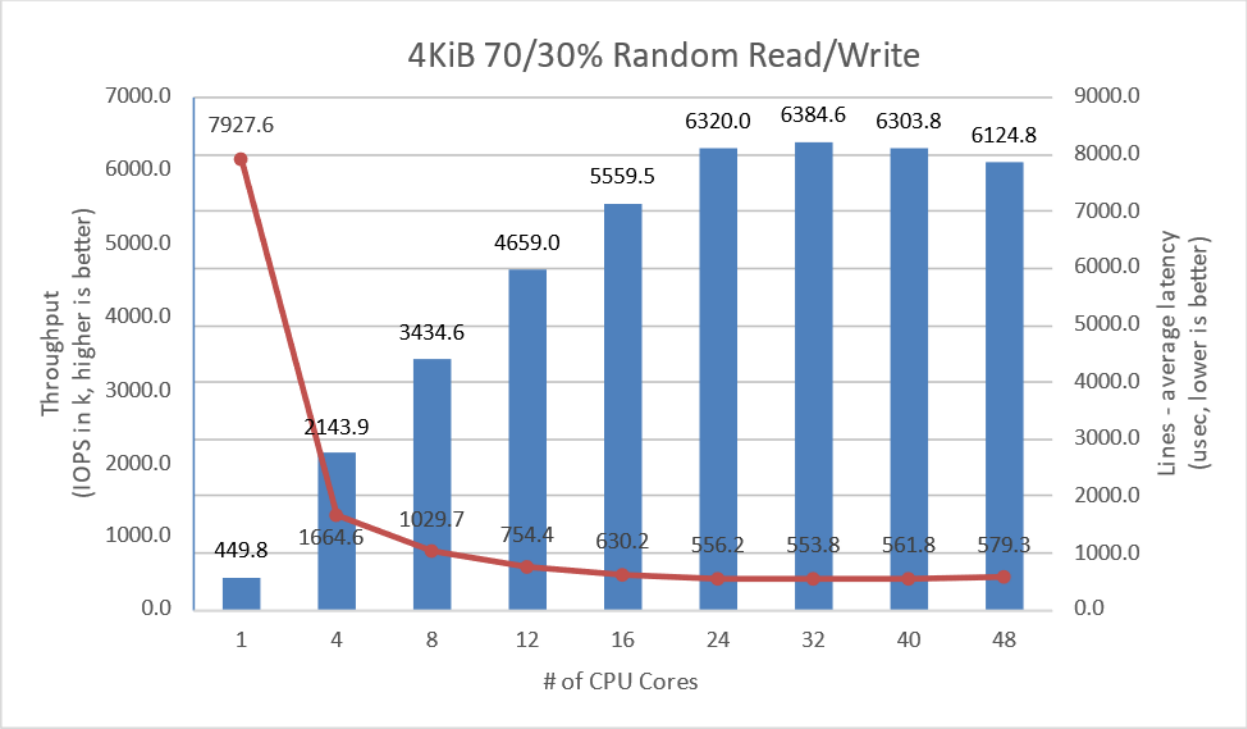

Figure 10: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KiB Random 70% Read 30% Write Workload at QD=256

Conclusions

1. For the 4KiB Random Read workload, the SPDK NVMe-oF TCP Initiator performance scales linearly up to 16 CPU cores. Increasing the number of CPU cores beyond 16 CPU results in non-linear performance improvement, which peaks at 24 CPU cores with 5.7 million IOPS, reaching 200 Gbps network link saturation.

2. In case of 4KiB Random Write workload, performance scaling is non-linear. It rapidly scales to 2.8 million IOPS when transitioning to 4 CPU cores and reaches 100 Gbps network link saturation. Further CPU scaling results in non-linear performance change, and peaks at 8 CPU cores at 3.55 million IOPS. 200 Gbps network link is not saturated.

3. Mixed Random Read-Write workload performance scales linearly up to 12 CPU cores, reaching 4.66 million IOPS. Increasing the number of cores further to 32 allows to reach peak performance of 6.38 million IOPS. Beyond this point increasing number of cores results in noticeable performance degradation.

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

This test case was designed to understand latency characteristics of SPDK NVMe-oF TCP Target and Initiator vs. the Linux Kernel NVMe-oF TCP 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 on top of a *Null Block Device*. The null block device (bdev) was chosen as the backend block device to eliminate the media latency during these tests.

For this test case a Linux Kernel feature called aRFS (Accelerated Receive Flow Steering) was used. With RFS, network packets are forwarded depending on the location of the application thread processing the socket that includes those packets. This forwarding is done entirely in software. But for the accelerated RFS – or aRFS – NICs with proper flow steering support can do this forwarding in hardware instead. The key benefit to aRFS is ensuring that Rx packets for a given TCP flow are directed to an Rx queue that is processed on the same CPU core as the thread processing that TCP flow. This helps ensure caching effects of the Rx path and the user space processing happen on the same CPU core. Both RFS and aRFS are available in most Linux distributions but need to be configured before using. Steps for enabling aRFS in this test are described in table below.

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

Item	Description
Test Case	Linux Kernel vs. SPDK NVMe-oF Latency
Test configuration	

SPDK NVMe-oF Target configuration	All below commands are executed with spdk/scripts/rpc.py script.
	
	Set iobuf buffer pool options:
	iobuf_set_options --small-pool-count 32767 --large-pool-count 16383
	
	nvmf_create_transport -t TCP
	(creates TCP transport layer with default values:
	{
	"trtype": "TCP",
	"max_queue_depth": 128,
	"max_io_qpairs_per_ctrlr": 127,
	"in_capsule_data_size": 4096,
	"max_io_size": 131072,
	"io_unit_size": 131072,
	"max_aq_depth": 128,
	"num_shared_buffers": 8192,
	"buf_cache_size": 32,
	"dif_insert_or_strip": false,
	"c2h_success": true,
	"sock_priority": 0,
	"abort_timeout_sec": 1
	}
	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 tcp -f ipv4 -s 4420 -a 20.0.0.1

Kernel NVMe-oF Target configuration	Target configuration file loaded using nvmet-cli tool.
	{
	"ports": [
	{
	"addr": {
	"adrfam": "ipv4",
	"traddr": "20.0.0.1",
	"trsvcid": "4420",
	"trtype": "tcp"
	},
	"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"
	}
	]
	}

fio configuration	
SPDK NVMe-oF Initiator FIO plugin configuration	**BDEV.conf** See appendix D **fio.conf** [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 time_based=1 ramp_time=60 runtime=300 [filename0] filename=Nvme0n1

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

aRFS Configuration	**Enable ntuple feature in the NIC driver and check its status:**
	$ ethtool -K eth3 ntuple on
	
	$ ethtool -k eth3| grep ntuple
	ntuple-filters: on
	
	Disable Linux Kernel IRQ balancer
	$ service irqbalance stop
	
	Ensure that NIC's IRQ affinity is spread across all cores:
	$ set_irq_affinity.sh eth3
	$ show_irq_affinity.sh eth3
	(Mellanox utility scripts available on Github.com)
	
	Configure the RFS global and per-queue flow table entries. This needs to be done for every NIC interface taking part in the test.
	echo 32768 > /proc/sys/net/core/rps_sock_flow_entries
	for r in /sys/class/net/eth3/queues/rx-*/rps_flow_cnt;
	do echo 512 > $r
	done

SPDK vs Kernel NVMe-oF Target Results

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

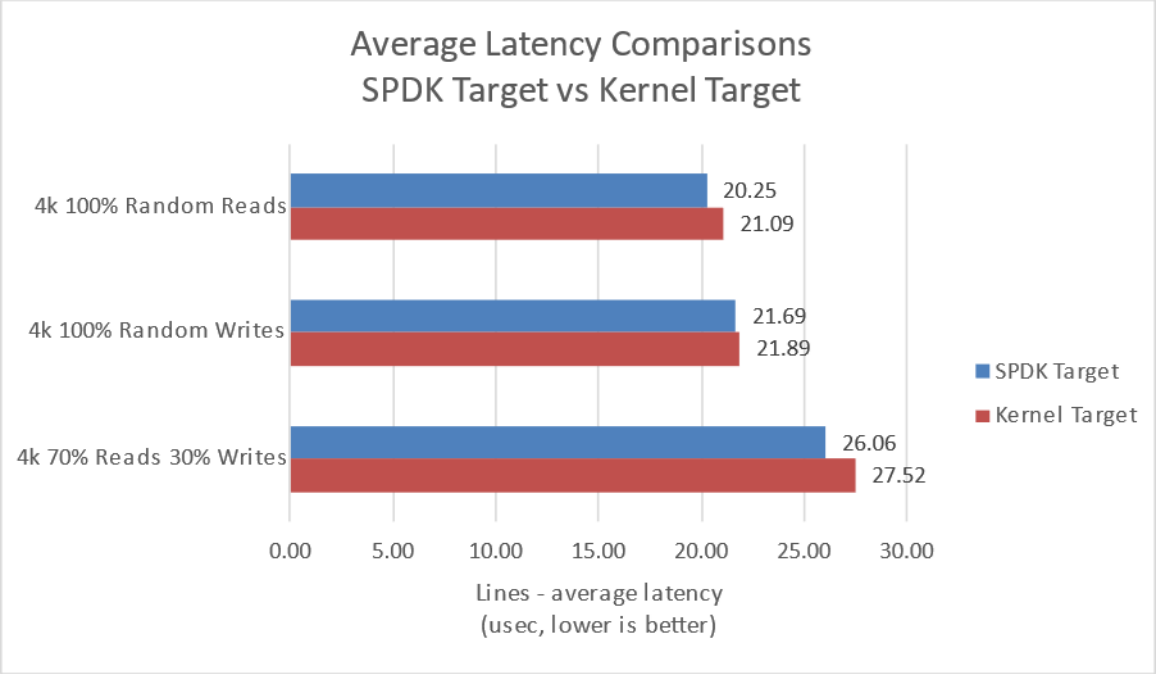

Figure 11: SPDK vs. Kernel NVMe-oF TCP Target Average I/O Latency for various workloads run using the Kernel Initiator

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	20.25	48579	26.8	41.4	92.3	212.7
4KiB 100% Random Writes IOPS	21.69	45385	35.2	55.0	107.3	229.0
4KiB 100% Random 70% Reads 30% Writes IOPS	26.06	37848	59.4	100.3	190.9	238.5

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	21.09	46672	27.3	30.4	58.3	164.2
4KiB 100% Random Writes IOPS	21.89	44985	29.0	31.5	59.5	162.1
4KiB 100% Random 70% Reads 30% Writes IOPS	27.52	35877	58.8	97.5	183.1	220.0

SPDK vs Kernel NVMe-oF TCP Initiator Results

This following data was collected using Kernel & SPDK initiator against an SPDK target.

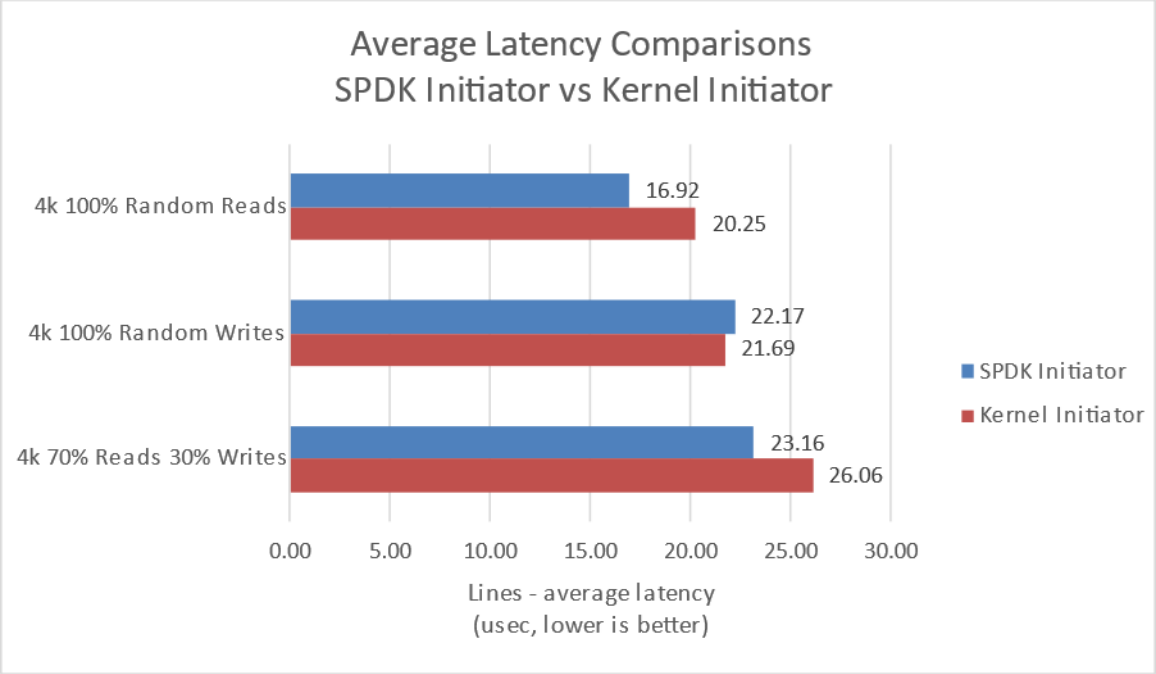

Figure 12: SPDK vs. Kernel NVMe-oF TCP Initiator Average I/O Latency for various workloads against SPDK Target

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	16.92	58337	25.7	34.0	61.9	196.3
4KiB 100% Random Writes IOPS	22.17	44664	35.9	62.0	101.2	227.0
4KiB 100% Random 70% Reads 30% Writes IOPS	23.16	42727	54.9	82.6	125.2	223.7

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	20.25	48579	26.8	41.4	92.3	212.7
4KiB 100% Random Writes IOPS	21.69	45385	35.2	55.0	107.3	229.0
4KiB 100% Random 70% Reads 30% Writes IOPS	26.06	37848	59.4	100.3	190.9	238.5

SPDK vs Kernel NVMe-oF Kernel + Initiator Results

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

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	16.92	58337	25.7	34.0	61.9	196.3
4KiB 100% Random Writes IOPS	22.17	44664	35.9	62.0	101.2	227.0
4KiB 100% Random 70% Reads 30% Writes IOPS	23.16	42727	54.9	82.6	125.2	223.7

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

Access Pattern	Avg. Latency (usec)	IOPS	p99 (usec)	p99.9 (usec)	p99.99 (usec)	p99.999 (usec)
4KiB 100% Random Reads IOPS	21.09	46672	27.35	30.42	58.28	164.18
4KiB 100% Random Writes IOPS	21.89	44985	28.97	31.53	59.48	162.13
4KiB 100% Random 70% Reads 30% Writes IOPS	27.52	35877	58.78	97.45	183.07	220.02

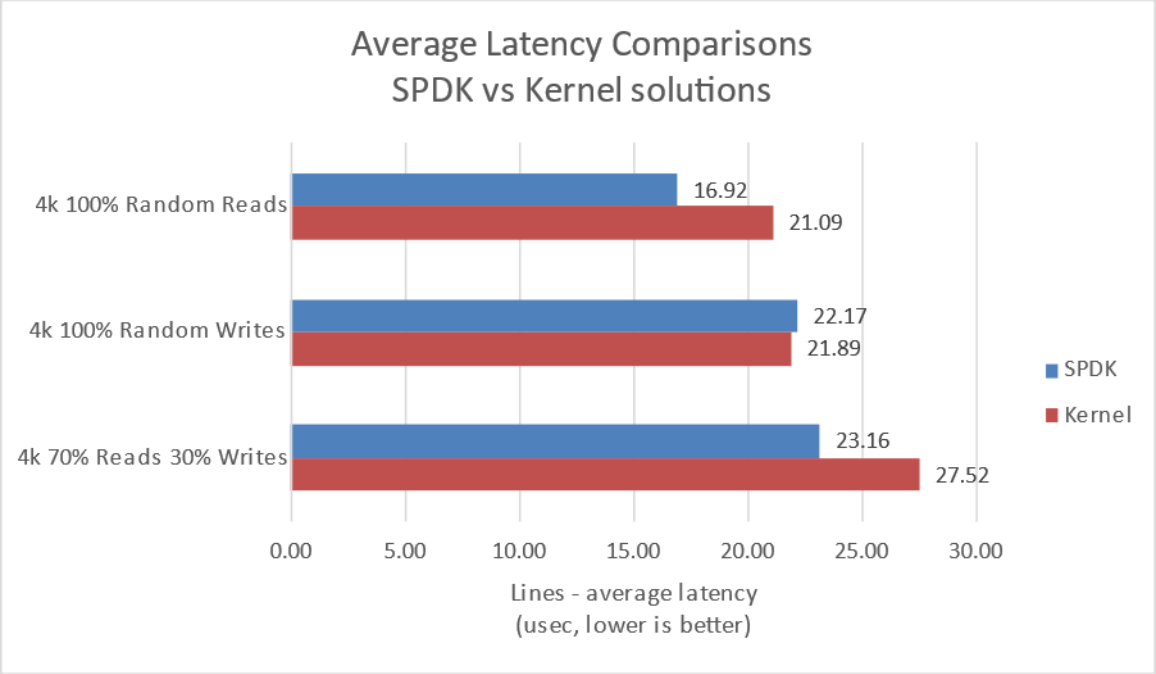

Figure 13: SPDK vs. Kernel NVMe-oF TCP solutions Average I/O Latency for various workloads

Conclusions

1. SPDK NVMe-oF TCP reduces latency by up to 1.45 usec. vs. Linux Kernel NVMe-oF TCP Initiator, which eliminates up to 5.27% of software overhead.

2. SPDK NVMe-oF Initiator reduces the average latency by up to 3.33 usec vs. the Linux Kernel NVMe-oF Initiator, which eliminates up to 16.45% NVMe-oF software overhead.

3. The SPDK NVMe-oF TCP target and initiator reduced the average latency by up to 4.17 usec vs. the Linux Kernel NVMe-oF target and initiator, which eliminates up to 19.77% NVMe-oF software overhead.

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

This test case was performed in order to understand throughput and latency capabilities of SPDK NVMe-oF Target vs. Linux Kernel NVMe-oF Target under increasing number of connections per subsystem. The number of connections (or I/O queue pairs) per NVMe-oF subsystem were varied and corresponding aggregated IOPS and number of CPU cores metrics were reported. The number of CPU cores metric was calculated from % CPU utilization measured using sar (systat package in Linux). The SPDK NVMe-oF Target was configured to run on 24 cores, 14 NVMe-oF subsystems (1 per Kioxia NVMe SSD) and 2 initiators were used both running I/Os to 7 separate subsystems using Kernel NVMe-oF initiator. We ran the following workloads on the host systems:

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

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

Item	Description
Test Case	NVMe-oF Target performance under varying # of connections
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 24 CPU cores.
Kernel NVMe-oF Target configuration	Target configuration file loaded using nvmet-cli tool. For detail configuration file contents please see Appendix E.
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 tcp –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode2 –t tcp –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode3 –t tcp –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode4 –t tcp –a 20.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode5 –t tcp –a 20.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode6 –t tcp –a 20.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode7 –t tcp –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 tcp –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode9 –t tcp –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode10 –t tcp –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode11 –t tcp –a 10.0.0.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode12 –t tcp –a 10.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode13 –t tcp –a 10.0.1.1 –s 4420 nvme connect –n nqn.2018-09.io.spdk:cnode14 –t tcp –a 10.0.1.1 –s 4420
FIO configuration (used on both initiators)	**FIO.conf** [global] ioengine=io_uring thread=1

	group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={128, 192, 384} 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 number of CPU cores used while running the SPDK NVMe-oF target was 24, whereas for the case of Linux Kernel NVMe-oF target there was no CPU core limitation applied.

The metrics in the graph represent relative efficiency in IOPS/core which was calculated based on total aggregate IOPS divided by total CPU cores used while running that specific workload. For the case of Kernel NVMe-oF target, total CPU cores was calculated from % CPU utilization which was measured using SAR utility in Linux.

4KiB Random Read Results

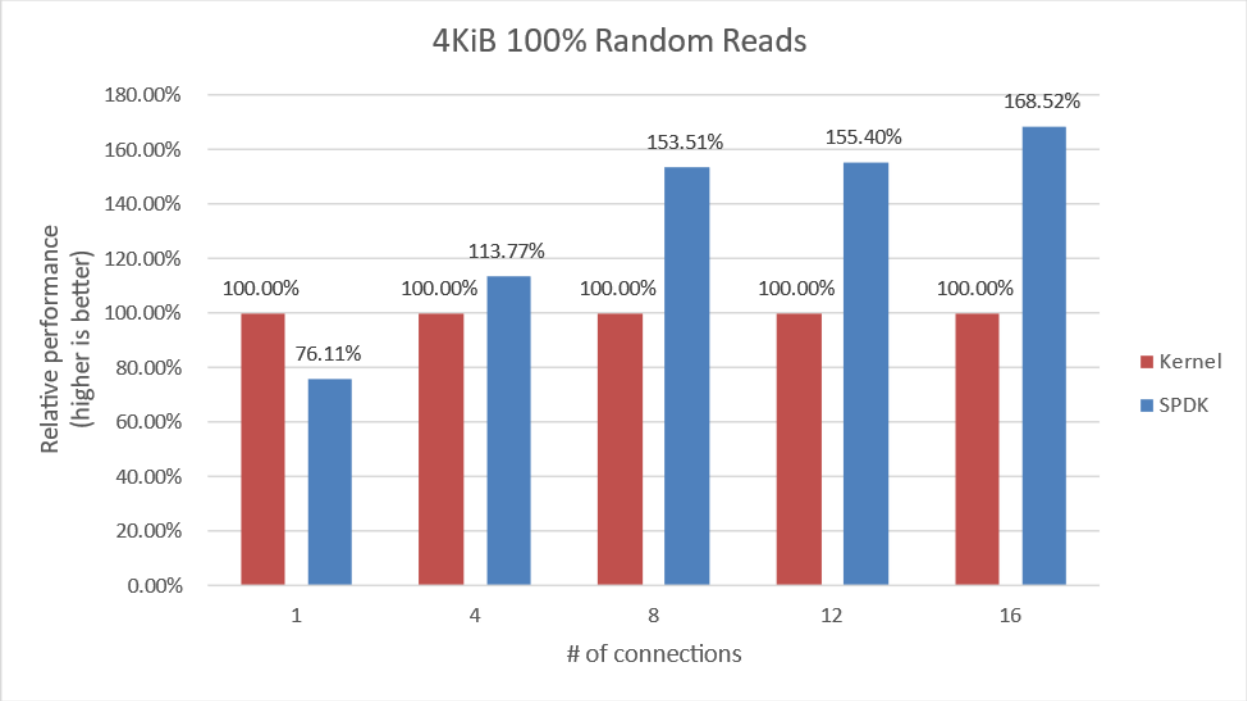

Figure 14: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF Target IOPS/Core for 4KiB 100% Random Reads QD=384 using the Kernel Initiator

Table 24: Linux Kernel NVMe-oF TCP Target: 4KiB 100% Random Reads, QD=384

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	132.33	4038.3	1554.7	20.8
4	274.13	8365.8	751.5	36.6
8	323.93	9885.6	643.1	58.1
12	292.81	8935.7	601.0	61.1
16	303.45	9260.7	704.1	66.6

Table 25: SPDK NVMe-oF TCP Target: 4KiB 100% Random Reads, QD=384

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	139.32	4251.8	1476.7	28.7
4	245.42	7489.6	862.4	28.8
8	259.64	7923.7	845.3	30.3
12	237.27	7240.8	742.1	31.9
16	242.29	7394.2	927.8	31.6

4KiB Random Write Results

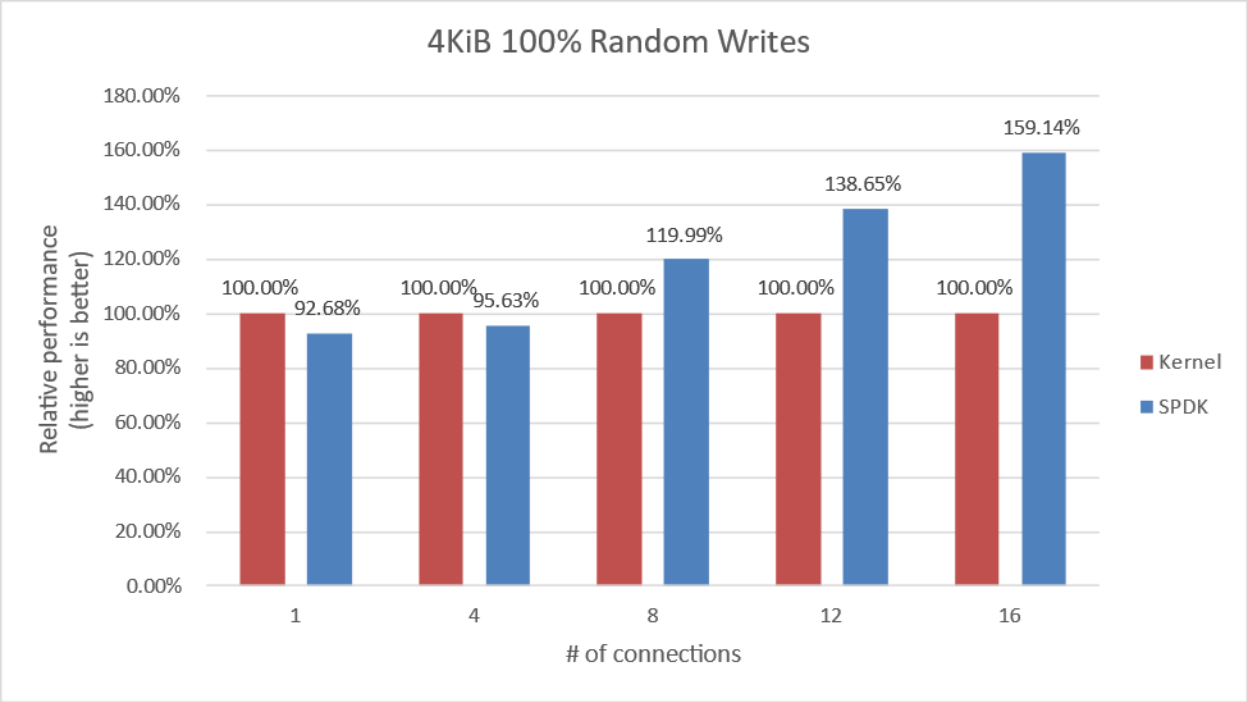

Figure 15: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF Target IOPS/Core for 4KiB 100% Random Writes QD=192 using the Kernel Initiator

Note: Drives were not pre-conditioned while running 100% Random write I/O Test

Table 26: Linux Kernel NVMe-oF TCP Target: 4KiB 100% Random Writes, QD=192

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	136.18	4155.8	750.9	136.18
4	186.25	5683.8	570.2	186.25
8	233.71	7132.3	462.2	233.71
12	232.47	7094.5	382.6	232.47
16	237.22	7239.3	477.3	237.22

Table 27: SPDK NVMe-oF TCP Target: 4KiB 100% Random Writes, QD=192

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	122.15	3727.7	831.8	30.5
4	156.46	4774.8	714.3	30.8
8	164.31	5014.3	696.3	31.8
12	160.95	4911.7	547.1	33.4
16	162.27	4952.1	715.0	32.2

4KiB Random Read-Write Results

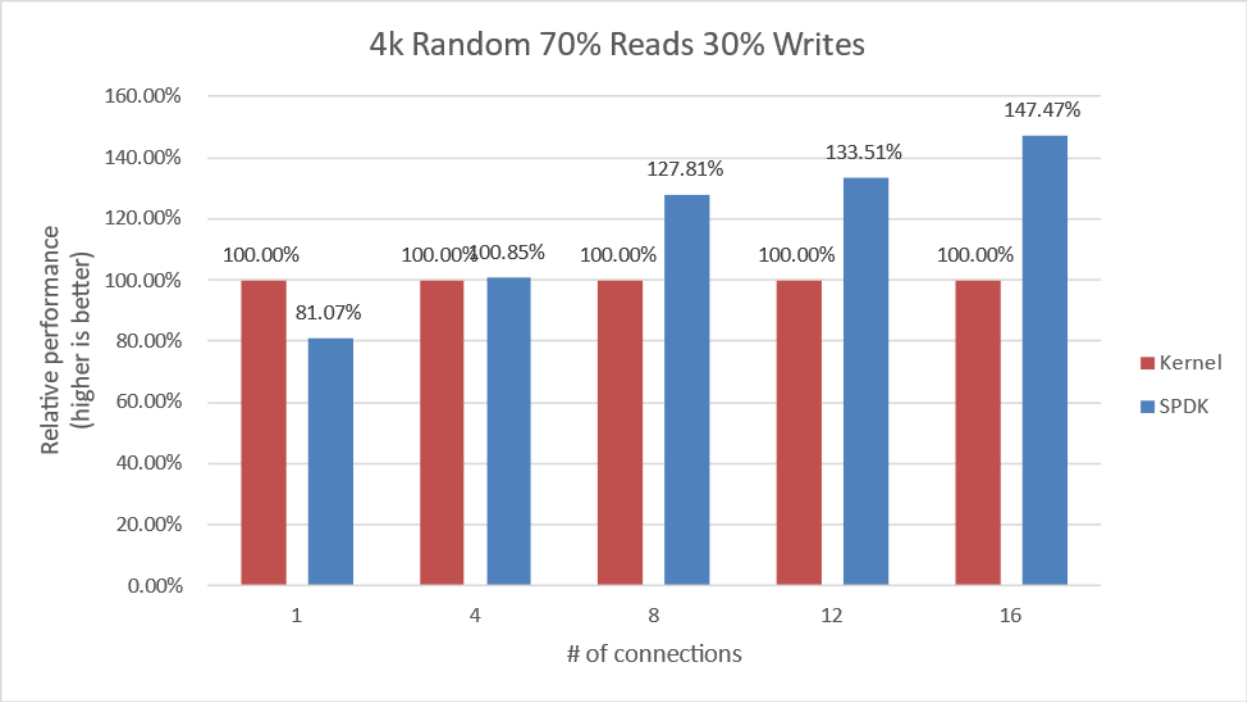

Figure 16: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF Target IOPS/Core for 4KiB Random 70% Reads 30% Writes QD=192 using Kernel Initiator

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

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	130.31	3976.7	787.8	23.2
4	191.83	5854.3	544.6	32.6
8	238.50	7278.4	438.6	54.2
12	220.34	6724.2	399.1	60.5
16	222.44	6788.2	480.4	63.3

Table 29: SPDK NVMe-oF TCP Target: 4KiB 70% Random Read 30% Random Write, QD=128

Connections per subsystem	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	134.42	4102.1	763.2	29.5
4	174.68	5330.9	607.4	29.5
8	180.32	5502.9	604.3	32.1
12	159.70	4873.7	551.2	32.9
16	172.13	5252.9	654.5	33.2

Low Connections Results

During testing, it was initially observed that the relative efficiency of SPDK Target was about 50-60% of Kernel Target. This was primarily because SPDK traditionally used a fixed number of CPU cores configured at startup, without an intrinsic mechanism to decrease the number of I/O cores on-the-fly if the SPDK target does not need all the CPU resources.
However, with the implementation of the dynamic scheduler, SPDK now can adjust CPU resource allocation based on demand. The dynamic scheduler allows SPDK to dynamically reduce or increase the number of I/O cores in response to changing workloads, thereby optimizing CPU resource utilization. This significantly enhances SPDK's efficiency and performance, ensuring it does not underutilize or overcommit CPU resources. For detailed information, please refer to the section on the dynamic scheduler in the documentation (appendix A).

The test cases with 1 connection per subsystems were re-run with SPDK using only 4 CPU cores.

Table 30: SPDK & Kernel NVMe-oF TCP Target relative efficiency comparison for various workloads, QD=128, 1 connection per subsystem

Workload	Target	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
Random Read	Linux	93.51	2853.7	627.7	10.0
	SPDK	72.36	2208.4	811.3	5.5
Random Write	Linux	90.60	2765.0	647.9	13.1
	SPDK	47.36	1445.4	1239.5	6.1
Random Read/Write	Linux	98.85	3016.8	593.8	11.8
	SPDK	61.74	1884.1	950.9	5.8

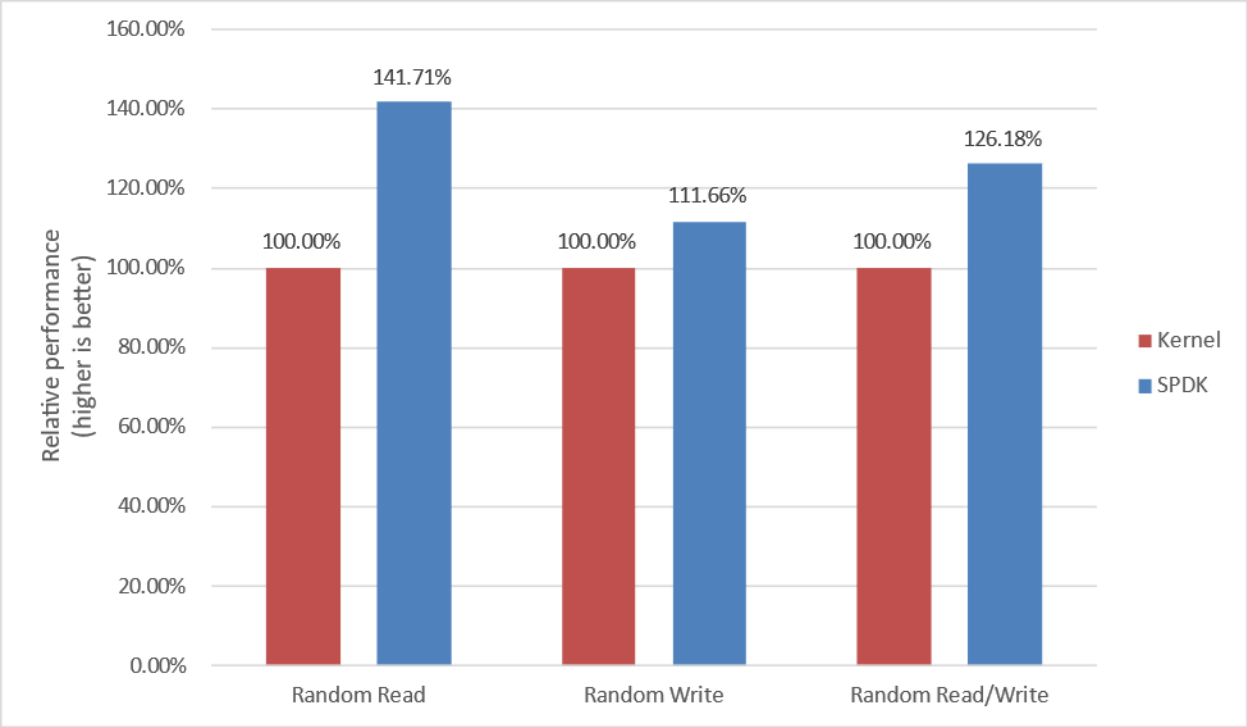

Figure 17: Relative Efficiency Comparison of Linux Kernel vs. SPDK NVMe-oF Target IOPS/Core for various workloads, 1 connection per subsystem and reduced number of SPDK Target CPU Cores (4)

Conclusions

1. The Linux Kernel NVMe-oF TCP target relative efficiency in IOPS/Core was better than SPDK when there was low number of connections per subsystem because the SPDK NVMe-oF target uses a fixed number of CPU cores when configured with the static scheduler. Therefore, we re-run the test cases with 1 connection per subsystem but lowered the number of I/O cores used by the SPDK Target to 4 and added the results to the tables which show a relative performance better than Linux Kernel NVMe-oF TCP target up to 1.42x for Random Read, 1.11x times for Random Writes and 1.26x times for Random Read/Write workloads.

2. The performance peaked for all workloads at 8 connections per subsystem for both SPDK and Kernel NVMe-oF TCP Target.

3. The SPDK NVMe-oF TCP target relative efficiency in IOPS/Core was up to 1.68x, 1.59x and 1.47x times better than the Linux Kernel NVMe-oF target for Random Read, Random Writes and Random Read/Write workloads respectively.

Summary

This report showcased performance results with SPDK NVMe-oF TCP 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 Linux Kernel NVMe-oF (Target/Initiator) against the accelerated polled-mode driven SPDK NVMe-oF (Target/Initiator) implementation.

Throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF target I/O cores when serving 4KiB random workloads. The SPDK NVMe-oF target saturates a 400 Gbps network link using 48 CPU cores for the 4KiB Random Read and 200 Gbps for Random Write workload at 32 CPU cores. The IOPS scalability remains close to linear for all workloads until the results are close to saturating network link (or NVMe drives throughput in case of Random Write workload).

For the SPDK NVMe-oF TCP Initiator running Random Read and Random Read/Write workloads the IOPS throughput scales almost linearly with addition of CPU cores until the network was almost saturated. Further increasing the number of CPU cores results in performance degradation. A single initiator was able to saturate a 200Gb link for these workloads.

For the NVMe-oF TCP latency comparison, the SPDK NVMe-oF Target and Initiator average latency is up to about 16.45% and 19.77% lower than their Linux Kernel counterparts respectively when testing against null block device-based backend.

The SPDK NVMe-oF TCP Target performed up to 1.68 times better w.r.t IOPS/core than Linux Kernel NVMe-oF target while running 4KiB 100% Random Read workload with increasing number of connections per NVMe-oF subsystem. For 4KiB 100% Random Write workloads, the SPDK Target performed 1.59 times better than the Kernel Targets. Additionally, in scenarios involving mixed 4KiB 100% Random Read/Write workloads, the SPDK Target maintained better performance than the Kernel Targets by 1.47 times.

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 the hardware and software configurations.

List of Figures

List of Tables

Appendix A – SPDK NVMe-oF Target Dynamic Scheduler

SPDK supports scheduling of lightweight threads by use of schedulers which are provided as plugins. By default, SPDK uses "static" scheduler where threads are distributed round-robin among application reactors and no thread is ever moved. The available "dynamic" scheduler allows moving threads between application reactors – active threads are distributed equally between reactors taking desired cpu_mask into consideration, while idle threads are moved to the main core. In case a thread becomes active again it is once again moved to appropriate reactor. Thank to this approach "dynamic" scheduler can be used for power saving and reduction of CPU utilization.

The number of CPU cores metric was calculated from % CPU utilization measured using sar utility (systat package in Linux). The "average power consumption" metric refers to the overall, system-wide power draw of the Target system during the test. It was calculated by using ipmitool to extract power statistics from sensors exposed by the system's BMC.

The test cases from Test Case 4 ("NVMe-oF Performance with increasing number of connections") with 1 connection per subsystem were re-run with SPDK with 24 CPU cores and scheduler set to both "static" and "dynamic" for comparison. We used a target configuration that scales to saturate 100 Gbps and observed that dynamic scheduling reduced the number of CPU cores used by the target at a slight cost of IOPS performance. For high queue workloads CPU utilization and IOPS are similar for both schedulers, except for Random Read workload where Dynamic Scheduler still shows lower CPU consumption.

Table 31: SPDK NVMe-oF TCP Target static and dynamic scheduler performance comparison for various workloads, QD=1-128, 1 connection per subsystem

Workload	Scheduler	Queue Depth	Bandwidth (Gbps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores	Average Power Consumption [W]
Random Read	Static	1	1.60	49.0	284.7	24.4	605.3
		8	12.54	382.6	292.4	24.9	619.3
		32	40.10	1223.7	365.9	25.4	635.6
		64	67.66	2064.8	433.7	25.6	649.5
		128	96.47	2944.0	608.5	25.9	673.3
	Dynamic	1	0.72	22.1	631.2	1.2	512.1
		8	7.13	217.7	514.2	1.5	529.6
		32	28.66	874.7	512.0	4.0	553.9
		64	49.09	1498.1	597.9	5.3	571.0
		128	74.10	2261.2	792.3	7.6	601.2
Random Write	Static	1	7.59	231.5	60.1	25.3	624.9
		8	14.44	440.6	253.9	25.1	627.0
		32	46.80	1428.3	313.4	26.1	656.5
		64	71.73	2188.9	409.1	26.9	681.0
		128	90.99	2776.8	645.1	27.4	708.4

	Dynamic	1	0.89	27.1	514.2	1.3	514.8
		8	6.31	192.7	605.0	2.1	533.4
		32	30.94	944.2	474.6	5.7	571.9
		64	49.79	1519.5	590.2	8.6	600.8
		128	64.06	1955.0	918.3	11.2	634.7
Random Read/Write	Static	1	1.71	52.3	266.2	24.4	608.0
		8	12.85	392.2	285.2	25.0	621.9
		32	41.92	1279.2	350.0	25.6	643.2
		64	69.92	2133.7	419.7	26.1	664.1
		128	100.86	3077.9	581.9	26.7	694.4
	Dynamic	1	0.82	25.0	557.0	1.3	514.0
		8	7.20	219.7	517.0	2.1	533.4
		32	32.87	1003.1	446.4	5.4	565.1
		64	53.48	1632.1	548.8	7.6	586.7
		128	79.27	2419.2	740.7	10.9	627.6

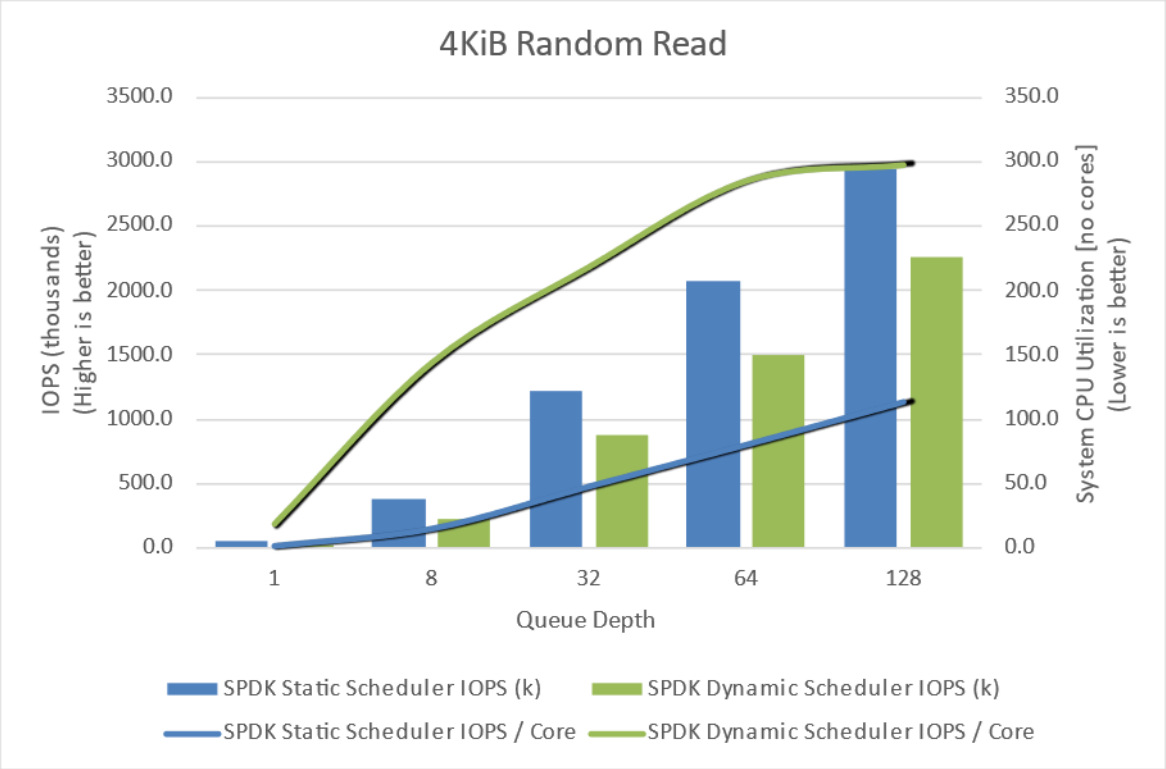

Figure 18: IOPS/Core Performance comparison of SPDK NVMe-oF Target static and dynamic scheduler for 4 KiB Random Read workload, 1 connection per subsystem, 24 CPU cores SPDK Target

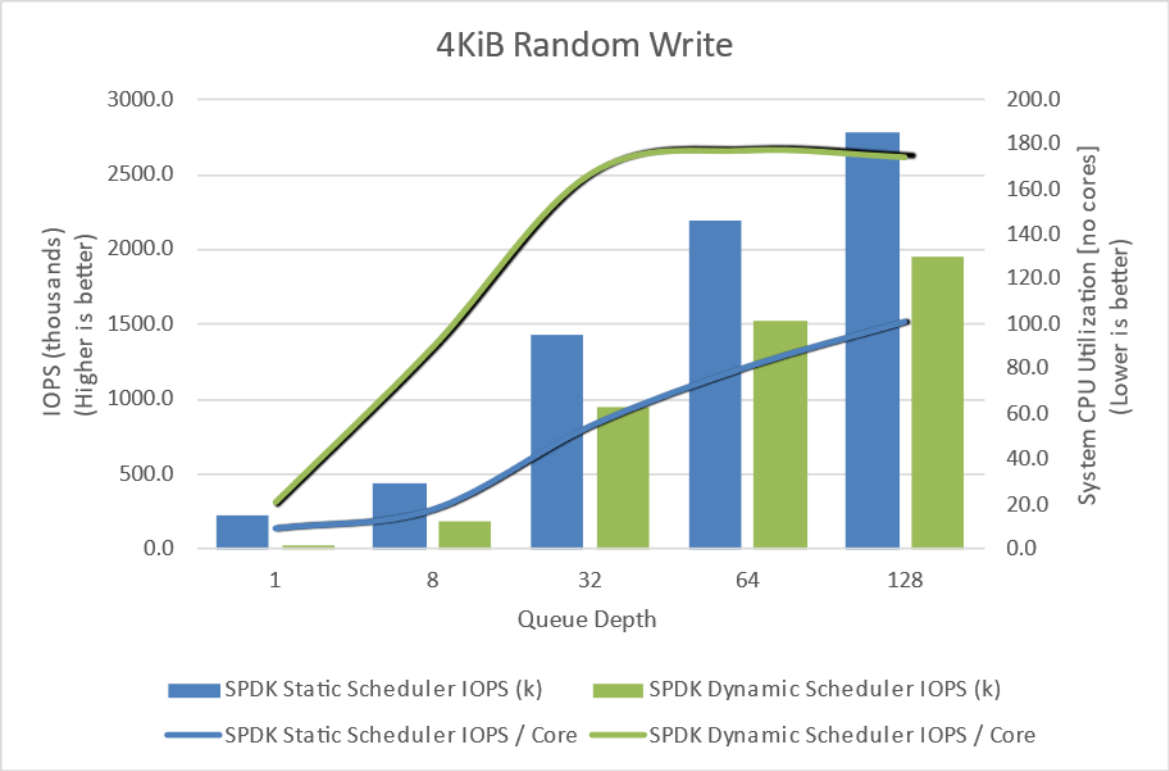

Figure 19: IOPS/Core Performance comparison of SPDK NVMe-oF Target static and dynamic scheduler for 4 KiB Random Write workload, 1 connection per subsystem, 24 CPU cores SPDK Target

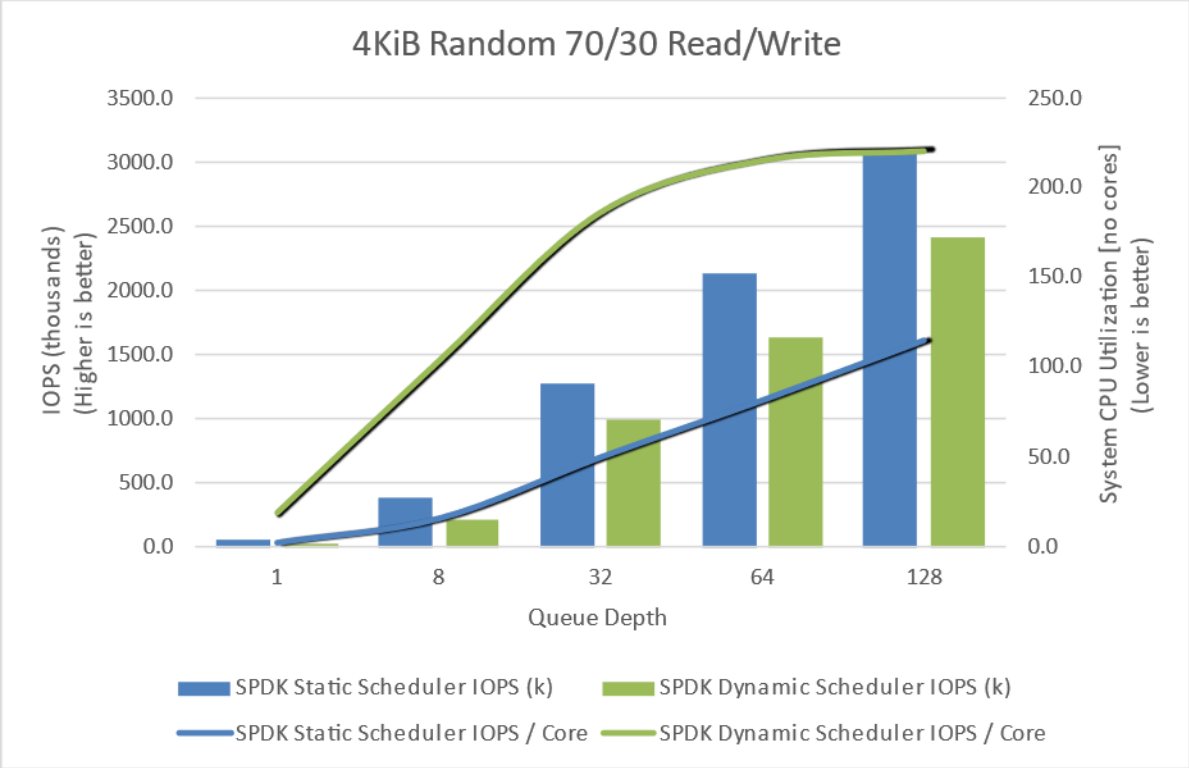

Figure 20: IOPS/Core Performance comparison of SPDK NVMe-oF Target static and dynamic scheduler for 4 KiB Random 70/30 Read/Write workload, 1 connection per subsystem, 24 CPU cores SPDK Target

Appendix B – Test Case 1 SPDK NVMe-oF Initiator bdev configuration

Initiator system 1

```
{
  "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"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme1",
            "trtype": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode7",
            "adrfam": "IPv4"
          }
        }
      ]
    },
    {
      "subsystem": "iobuf",
      "config": [
        {
          "method": "iobuf_set_options",
          "params": {
            "small_pool_count": 32768,
            "large_pool_count": 16384
          }
        }
```

```
      ]
    }
  ]
}
```


Initiator system 2


```
{
  "subsystems": [
    {
      "subsystem": "bdev",
      "config": [
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme0",
            "trtype": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
            "traddr": "10.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode7",
            "adrfam": "IPv4"
          }
        }
      ]
    },
    {
      "subsystem": "iobuf",
      "config": [
        {
          "method": "iobuf_set_options",
          "params": {
            "small_pool_count": 32768,
            "large_pool_count": 16384
          }
        }
      ]
    }
```

```
}
```

Appendix C – Test Case 2 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"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme1",
            "trtype": "tcp",
            "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": "tcp",
            "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": "tcp",
            "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": "tcp",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "subnqn": "nqn.2018-09.io.spdk:cnode4",
        "adrfam": "IPv4"
      }
    },
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
        "name": "Nvme5",
        "trtype": "tcp",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "subnqn": "nqn.2018-09.io.spdk:cnode5",
        "adrfam": "IPv4"
      }
    },
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
        "name": "Nvme6",
        "trtype": "tcp",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "subnqn": "nqn.2018-09.io.spdk:cnode6",
        "adrfam": "IPv4"
      }
    },
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
        "name": "Nvme7",
        "trtype": "tcp",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "subnqn": "nqn.2018-09.io.spdk:cnode7",
        "adrfam": "IPv4"
      }
    },
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
        "name": "Nvme8",
        "trtype": "tcp",
        "traddr": "20.0.1.1",
        "trsvcid": "4420",
        "subnqn": "nqn.2018-09.io.spdk:cnode8",
        "adrfam": "IPv4"
      }
    },
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
```

```
      "name": "Nvme9",
      "trtype": "tcp",
      "traddr": "20.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode9",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme10",
      "trtype": "tcp",
      "traddr": "20.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode10",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme11",
      "trtype": "tcp",
      "traddr": "20.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode11",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme12",
      "trtype": "tcp",
      "traddr": "20.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode12",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme13",
      "trtype": "tcp",
      "traddr": "20.0.1.1",
      "trsvcid": "4420",
      "subnqn": "nqn.2018-09.io.spdk:cnode13",
      "adrfam": "IPv4"
    }
  },
  {
    "method": "bdev_nvme_attach_controller",
    "params": {
      "name": "Nvme14",
      "trtype": "tcp",
```

```
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode14",
            "adrfam": "IPv4"
          }
        },
        {
          "method": "bdev_nvme_attach_controller",
          "params": {
            "name": "Nvme15",
            "trtype": "tcp",
            "traddr": "20.0.1.1",
            "trsvcid": "4420",
            "subnqn": "nqn.2018-09.io.spdk:cnode15",
            "adrfam": "IPv4"
          }
        }
      ]
    },
    {
      "subsystem": "iobuf",
      "config": [
        {
          "method": "iobuf_set_options",
          "params": {
            "small_pool_count": 32768,
            "large_pool_count": 16384
          }
        }
      ]
    }
  ]
}
```

Appendix D – 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"
          }
        }
      ]
    },
```

```
    {
      "subsystem": "iobuf",
      "config": [
        {
          "method": "iobuf_set_options",
          "params": {
            "small_pool_count": 32768,
            "large_pool_count": 16384
          }
        }
      ]
    }
  ]
}
```

Appendix E – Kernel NVMe-oF TCP Target configuration

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

```
{
  "ports": [
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "trtype": "tcp"
      },
      "portid": 1,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode1"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4421",
        "trtype": "tcp"
      },
      "portid": 2,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode2"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4422",
        "trtype": "tcp"
      },
      "portid": 3,
      "referrals": [],
      "subsystems": [
```

```
        "nqn.2018-09.io.spdk:cnode3"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4423",
        "trtype": "tcp"
      },
      "portid": 4,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode4"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4424",
        "trtype": "tcp"
      },
      "portid": 5,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode5"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4425",
        "trtype": "tcp"
      },
      "portid": 6,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode6"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "20.0.1.1",
        "trsvcid": "4426",
        "trtype": "tcp"
      },
      "portid": 7,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode7"
      ]
    },
    {
      "addr": {
```

```
      "adrfam": "ipv4",
      "traddr": "20.0.1.1",
      "trsvcid": "4427",
      "trtype": "tcp"
    },
    "portid": 8,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode8"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.0.1",
      "trsvcid": "4428",
      "trtype": "tcp"
    },
    "portid": 9,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode9"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.0.1",
      "trsvcid": "4429",
      "trtype": "tcp"
    },
    "portid": 10,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode10"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.0.1",
      "trsvcid": "4430",
      "trtype": "tcp"
    },
    "portid": 11,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode11"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.0.1",
      "trsvcid": "4431",
      "trtype": "tcp"
    },
```

```
      "portid": 12,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode12"
      ]
    },
{
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4432",
        "trtype": "tcp"
      },
      "portid": 13,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode13"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4433",
        "trtype": "tcp"
      },
      "portid": 14,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode14"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4434",
        "trtype": "tcp"
      },
      "portid": 15,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode15"
      ]
    },
    {
      "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4435",
        "trtype": "tcp"
      },
      "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.

§