SPDK NVMe-oF (Target & Initiator) Performance Report Release 18.04

Testing Date: June-July 2018

Performed by:
Vishal Verma (vishal4.verma@intel.com)

Acknowledgments:
John Kariuki (john.k.kariuki@intel.com)
James R Harris (james.r.harris@intel.com)
Benjamin Walker (benjamin.walker@intel.com)

Revision History

Date	Revision	Comment
06/28/2018	V1.0	Complete performance runs
07/09/2018	V1.0	Review
07/16/2018	V2.0	Feedback
07/26/2018	V2.0	Review
08/10/2018	V3.0	Ready to publish

Contents

Audience and Purpose

This report is intended for people who are interested in SPDK NVMe-oF target and initiator performance and its comparison to the Linux kernel NVMe-oF target and initiator. The term *target* will be used to refer to the storage server, while the terms *initiator* and *host* will be used to refer to the client. This report provides performance and efficiency information on a set of underlying block devices in a set of common scenarios.

The purpose of reporting these tests is not to imply that there is a single superior solution, but rather to provide a baseline of well-tested configurations and procedures with repeatable and reproducible results. This report can also be viewed as documenting the best known methods for performance testing SPDK's NVMe-oF components.

Test setup

Target Configuration

Item	Description
Server Platform	SuperMicro SYS-2029U-TN24R4T  
CPU	Intel® Xeon® Platinum 8176 Processor (38.5MB L3, 2.10 GHz) https://ark.intel.com/products/120508/Intel-Xeon-Platinum-8176-Processor-38_5M-Cache-2_10-GHz Number of cores 28, number of threads 56
Memory	Total 192 GBs over 12 channels (16 GB DDR 2667 MHz 2R DIMMS each)
Operating System	Ubuntu 18.04 LTS

BIOS	2.0b
Linux kernel version	4.13.0-38-generic
SPDK version	SPDK 18.04
Storage	**OS:** 1x 200GB Intel SSD DC S3700 **Storage Target**: 16x Intel® P4600™ P4600x 1.6TB (FW: QDV10130) (8 on each CPU socket)
NIC	2x 100GbE Mellanox ConnectX-4 NICs. Both ports connected (1 on each CPU socket)

Initiator 1 Configuration

Item	Description
Server Platform	SuperMicro SYS-2028U TN24R4T+
CPU	Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (55MB Cache, 2.20 GHz) https://ark.intel.com/products/91317/Intel-Xeon-Processor-E5-2699-v4-55M-Cache-2_20-GHz Number of cores 22, number of threads 44 per socket (Both sockets populated)
Memory	Total 256 GBs (2 DIMMs/channel. DDR4 16GB DIMMs) @ 2400 MHz
Operating System	Ubuntu 18.04 LTS
BIOS	3.0a
Linux kernel version	4.13.0-38-generic
SPDK version	SPDK 18.04
Storage	**OS:** 1x 200GB Intel SSD DC S3700
NIC	1x 100GbE Mellanox ConnectX-4 NICs. Both ports connected (connected to CPU socket 0)

Initiator 2 Configuration

Item	Description
Server Platform	SuperMicro SYS-2028U TN24R4T+
CPU	Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (55MB Cache, 2.20 GHz) https://ark.intel.com/products/91317/Intel-Xeon-Processor-E5-2699-v4-55M-Cache-2_20-GHz Number of cores 22, number of threads 44 per socket (Only 1 CPU socket populated)
Memory	Total 128 GBs (2 DIMMs/channel. DDR4 16GB DIMMs) @ 2400 MHz
Operating System	Ubuntu 18.04 LTS
BIOS	3.0a

Linux kernel version	4.13.0-38-generic
SPDK version	SPDK 18.04
Storage	**OS:** 1x 200GB Intel SSD DC S3700
NIC	1x 100GbE Mellanox ConnectX-4 NICs. Both ports connected (connected to CPU socket 0)

BIOS settings

Item	Description
BIOS *(Applied to all 3 systems)*	Hyper threading Enabled CPU Power and Performance Policy <Performance> CPU C-state No Limit CPU P-state Enabled Enhanced Intel® Speedstep® Tech Enabled Turbo Boost Enabled

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 Intel® Omni-Path. 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 RDMA 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 uses the Infiniband/RDMA verbs API to access an RDMA-capable NIC. These should work on all flavors of RDMA transports, but are currently tested against RoCEv2, iWARP, and Omni-Path NICs. 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 sixteen Intel P4600 SSDs which were used as block devices for NVMe-oF subsystems and two 100GbE Mellanox ConnectX-4 NICs connected to provide up to 200GbE of network bandwidth. Each Initiator system has one Mellanox ConnectX-4 100GbE NIC 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 are reported for 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 using fio and CPU utilization was collected using sar (systat).

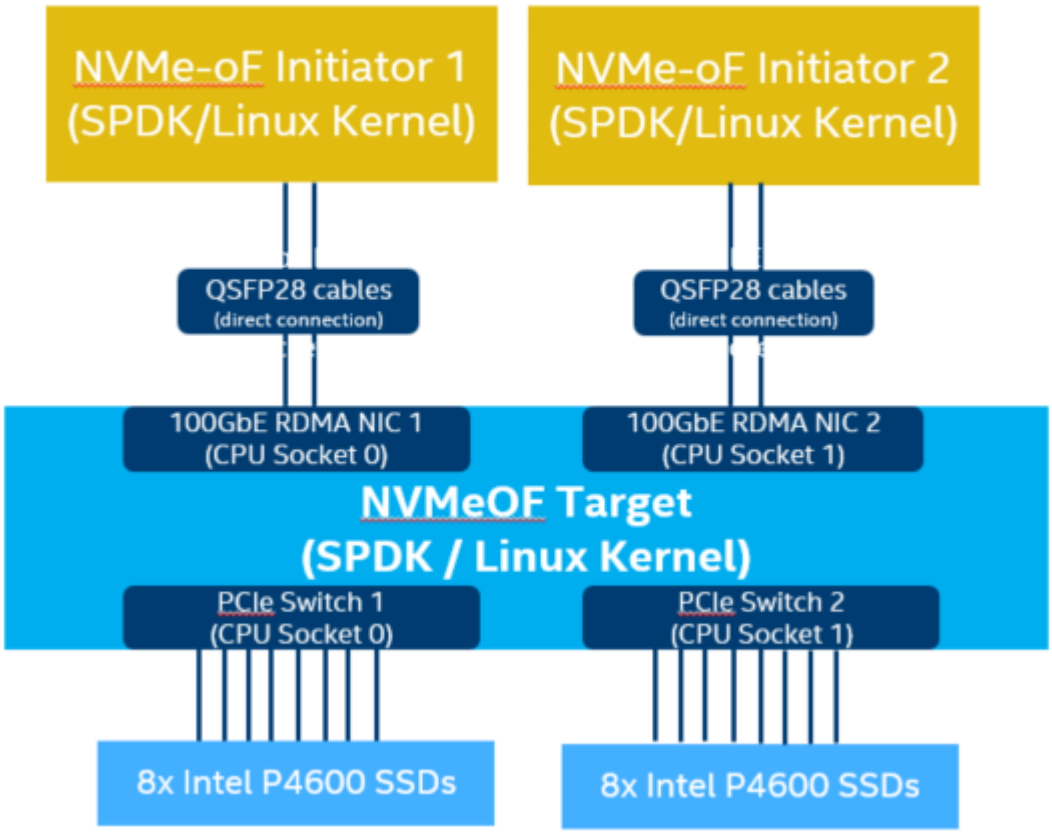

Figure 1: High level NVMe-oF performance testing setup

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

This test case is designed to illustrate how the SPDK NVMe-oF target scales the maximum number of I/O per second performed when additional CPU cores are added. The SPDK NVMe-oF target is configured to run with sixteen NVMe-oF subsystems. Each NVMe-oF subsystem contains a single namespace that corresponds to a single Intel P4600 device. Each of the initiators were connected to eight individual NVMe-oF subsystems, without overlap. The SPDK bdev fio plugin was run on the two initiators simultaneously. The SPDK target was configured to use 1, 2, 3 and 4 cores while running each of the following workloads on each initiator:

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

Item	Description
Test Case	Test SPDK NVMe-oF Target I/O core scaling
Test configuration	**Nvmf.conf:** # NVMf Target Configuration File [Global] ReactorMask 0x300000003 (This was modified depending on the number of cores tested) [Rpc] Enable Yes Listen 127.0.0.1 [Nvmf] # 89 choosen for 1 admin queue and 88 I/O queues. Initiator had 88 CPU cores. MaxQueuesPerSession 89 AcceptorPollRate 10000 [Nvme] TransportId "trtype:PCIe traddr:0000:60:00.0" Nvme0 TransportId "trtype:PCIe traddr:0000:61:00.0" Nvme1 TransportId "trtype:PCIe traddr:0000:62:00.0" Nvme2 TransportId "trtype:PCIe traddr:0000:63:00.0" Nvme3 TransportId "trtype:PCIe traddr:0000:64:00.0" Nvme4 TransportId "trtype:PCIe traddr:0000:65:00.0" Nvme5 TransportId "trtype:PCIe traddr:0000:66:00.0" Nvme6 TransportId "trtype:PCIe traddr:0000:67:00.0" Nvme7 TransportId "trtype:PCIe traddr:0000:b5:00.0" Nvme8 TransportId "trtype:PCIe traddr:0000:b6:00.0" Nvme9 TransportId "trtype:PCIe traddr:0000:b7:00.0" Nvme10 TransportId "trtype:PCIe traddr:0000:b8:00.0" Nvme11 TransportId "trtype:PCIe traddr:0000:b9:00.0" Nvme12 TransportId "trtype:PCIe traddr:0000:ba:00.0" Nvme13 TransportId "trtype:PCIe traddr:0000:bb:00.0" Nvme14 TransportId "trtype:PCIe traddr:0000:bc:00.0" Nvme15 RetryCount 4 Timeout 0 ActionOnTimeout None AdminPollRate 100000 HotplugEnable No [Subsystem1] NQN nqn.2016-06.io.spdk:cnode1 Listen RDMA 192.168.200.2:4420 AllowAnyHost Yes SN SPDK00000000000001 Namespace Nvme0n1 1 [Subsystem2] NQN nqn.2016-06.io.spdk:cnode2 Listen RDMA 192.168.100.2:4420 AllowAnyHost Yes SN SPDK00000000000002 Namespace Nvme1n1 1 [Subsystem3] NQN nqn.2016-06.io.spdk:cnode3

```
  Listen RDMA 192.168.100.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000003
  Namespace Nvme2n1 1

[Subsystem4]
  NQN nqn.2016-06.io.spdk:cnode4
Listen RDMA 192.168.100.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000004
  Namespace Nvme3n1 1

[Subsystem5]
  NQN nqn.2016-06.io.spdk:cnode5
  Listen RDMA 192.168.101.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000005
  Namespace Nvme4n1 1

[Subsystem6]
  NQN nqn.2016-06.io.spdk:cnode6
  Listen RDMA 192.168.101.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000006
  Namespace Nvme5n1 1

[Subsystem7]
  NQN nqn.2016-06.io.spdk:cnode7
  Listen RDMA 192.168.101.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000007
  Namespace Nvme6n1 1

[Subsystem8]
  NQN nqn.2016-06.io.spdk:cnode8
  Listen RDMA 192.168.101.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000008
  Namespace Nvme7n1 1

[Subsystem9]
  NQN nqn.2016-06.io.spdk:cnode9
  Listen RDMA 192.168.200.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000009
  Namespace Nvme8n1 1

[Subsystem10]
  NQN nqn.2016-06.io.spdk:cnode10
  Listen RDMA 192.168.200.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000010
  Namespace Nvme9n1 1

[Subsystem11]
  NQN nqn.2016-06.io.spdk:cnode11
  Listen RDMA 192.168.200.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000011
  Namespace Nvme10n1 1
[Subsystem12]
  NQN nqn.2016-06.io.spdk:cnode12
  Listen RDMA 192.168.200.2:4420
  AllowAnyHost Yes
  SN SPDK00000000000012
  Namespace Nvme11n1 1

[Subsystem13]
  NQN nqn.2016-06.io.spdk:cnode13
  Listen RDMA 192.168.201.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000013
  Namespace Nvme12n1 1

[Subsystem14]
  NQN nqn.2016-06.io.spdk:cnode14
  Listen RDMA 192.168.201.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000014
  Namespace Nvme13n1 1

[Subsystem15]
  NQN nqn.2016-06.io.spdk:cnode15
  Listen RDMA 192.168.201.2:4421
  AllowAnyHost Yes
  SN SPDK00000000000015
  Namespace Nvme14n1 1

[Subsystem16]
  NQN nqn.2016-06.io.spdk:cnode16
  Listen RDMA 192.168.201.2:4421
  AllowAnyHost Yes
```

	SN SPDK00000000000016 Namespace Nvme15n1 1 **BDEV.conf** [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode1" Nvme0 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode2" Nvme1 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode3" Nvme2 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode4" Nvme3 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode5" Nvme4 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode6" Nvme5 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode7" Nvme6 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode8" Nvme7
FIO config on initiator	[global] ioengine=examples/bdev/fio_plugin/fio_plugin spdk_conf=bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0} iodepth=32 time_based=1 ramp_time=60 runtime=300 [filename1] filename=Nvme0n1 [filename2] filename=Nvme1n1 [filename3] filename=Nvme2n1 [filename4] filename=Nvme3n1 [filename5] filename=Nvme4n1 [filename6] filename=Nvme5n1 [filename7] filename=Nvme6n1 [filename8] filename=Nvme7n1

Results in the table represent aggregate performance (IOPS & avg. latency) observed:

Test Result: 4K 100% Random Read IOPS

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1268.3	402.6
2 cores	2749.3	186.0
3 cores	3673.7	139.1
4 cores	4164.0	122.7

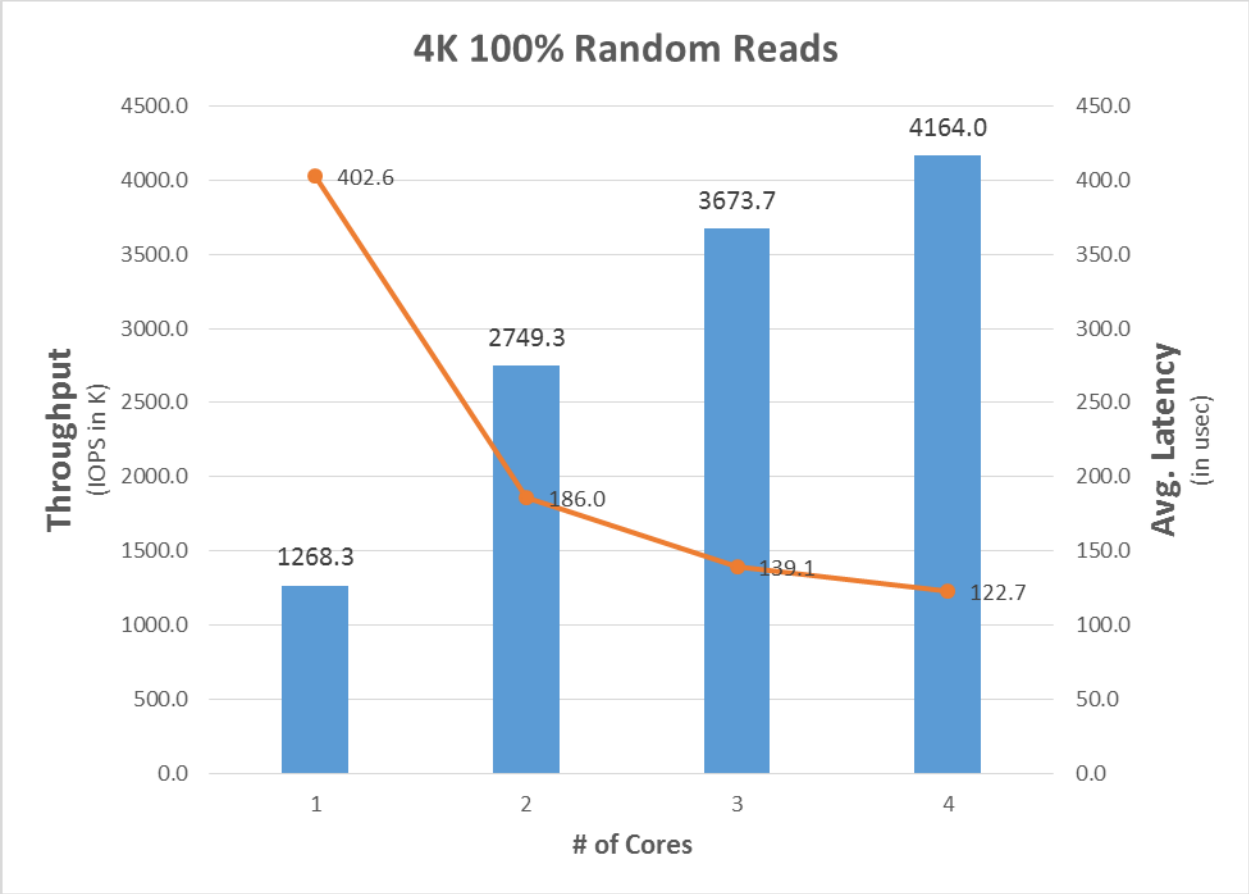

Figure 2: SPDK NVMe-oF Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random read workload

Drives were not pre-conditioned while running 100% Random write I/O Test. This artificially increases the number of IOPS that the storage devices are capable of, allowing the target to fully saturate the network.

Test Result: 4K 100% Random Writes IOPS

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1090	469.05
2 cores	2346	216.3
3 cores	3525.3	143.79
4 cores	4242	114.8

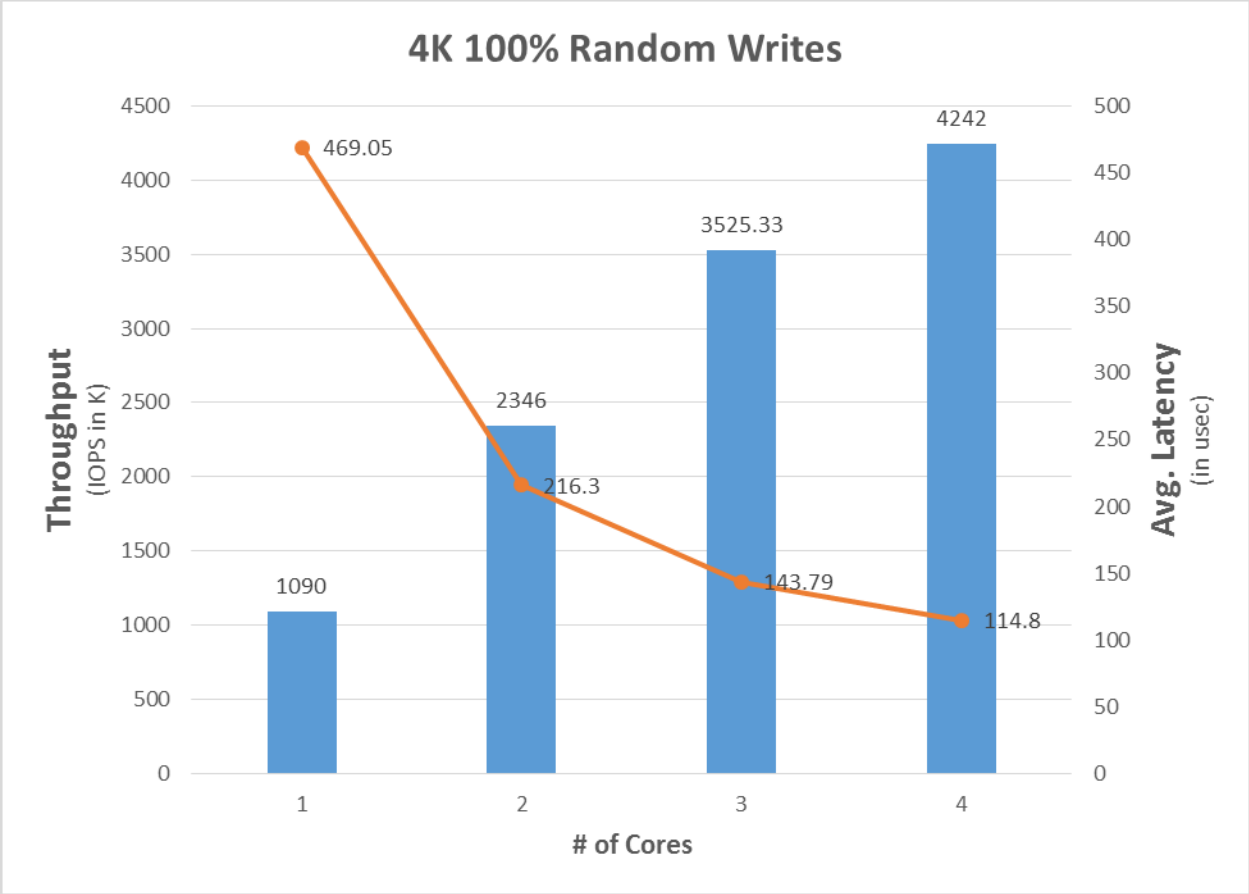

Figure 3: SPDK NVMe-oF Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random write workload

Test Result: 4K 70% Read 30% Write IOPS

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1195.0	427.8
2 cores	2203.7	231.7
3 cores	2369.0	215.9
4 cores	2390.3	213.9

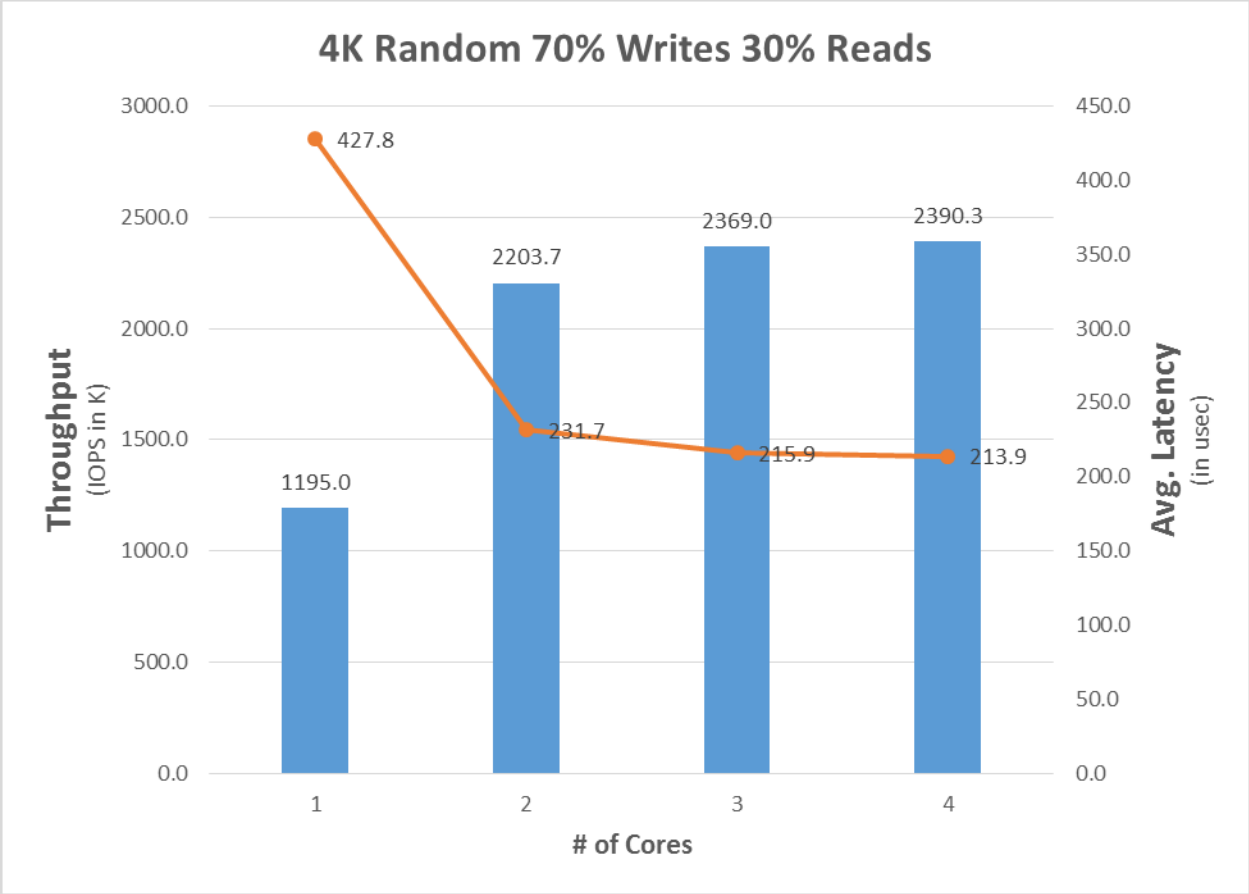

Figure 4: SPDK NVMe-oF Target I/O core scaling: IOPS vs. Latency while running 4KB Random 70% read 30% write workload

Conclusion:

1. For 100% Reads and 100% Writes, throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF Target I/O cores until hitting network bottleneck.

2. For 4K Random 70% Reads, 30% Writes performance doesn't scale from 2 to 3 and 4 cores due to some other bottleneck. It was observed that while running this test case locally without involving any network it can hit > 4M aggregate IOPS. But, while running this over the network, it could only achieve 2.3-2.4M IOPS max. This points to some other platform or network bottleneck while running this workload.

Large I/O Block Size sequential performance

128K block size I/O tests were performed with 100% sequential IO at queue depth 8 to each NVMe-oF namespace. The remainder of the configuration is identical to the tests above.

Aggregate IOPS, bandwidth, and average latency is as follows:

Test Result: 128K 100% Sequential Reads

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	23936	187	682
2 cores	23936	187	683.3
3 cores	23936	187	683
4 cores	23936	187	683

Test Result: 128K 100% Sequential Writes

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	23040	180	707.9
2 cores	23040	180	708
3 cores	23040	180	707
4 cores	23040	180	707.4

Test Result: 128K 70% Reads 30% Writes

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	29440	230	555.21
2 cores	29440	230	554
3 cores	29440	230	553
4 cores	29440	231	553

Conclusion:

1. A single CPU core saturated the network bandwidth. The SPDK NVMe-oF target running on 1 core does close to 23-24 GBps 100% Reads/Writes and ~29GBps 70-30 reads/writes, which is close to 2x 100GbE NICs network bandwidth. Therefore, adding more CPU cores did not result in increased performance because the network was the bottleneck.

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

This test case was performed in order to understand the performance of the SPDK NVMe-oF initiator as the number of available CPU cores is increased. The SPDK NVMe-oF target was configured similarly to the test cases above using four cores. The SPDK bdev fio plugin ran workloads targeting eight individual NVMe-oF namespaces on each of the two initiators, without overlap. The fio cpumask was varied in order to run 1, 2, 3 and 4 core tests while running following workloads from both the initiators:

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

1 core: 1 initiator was used.

2 cores: 2 separate initiators, each running a single core.

3 cores: 2 separate initiators, one with a single core and other with two cores.

4 cores: 2 separate initiators, each running two cores.

Item	Description
Test Case	Test SPDK NVMe-oF Target I/O core scaling
Test configuration	**Nvmf.conf:** Same as used in Test Case 1 **BDEV.conf** [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode1" Nvme0 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode2" Nvme1 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode3" Nvme2 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode4" Nvme3 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode5" Nvme4 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode6" Nvme5 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode7" Nvme6 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode8" Nvme7
FIO config on initiator	**1 core:** iodepth = 512 is a global parameter meaning it will be spread over 16 drives which are being handled by single FIO thread. So each nvme drive will get 512/16 = 32 iodepth [global] ioengine=examples/bdev/fio_plugin/fio_plugin spdk_conf=bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0}iodepth=512 time_based=1 ramp_time=60 runtime=300 [filename1] 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
filename=Nvme14n1
filename=Nvme15n1
```

2core:

```
[global]
ioengine=examples/bdev/fio_plugin/fio_plugin
spdk_conf=bdev.conf
thread=1
group_reporting=1
direct=1
norandommap=1
bs=4k
rw=randrw
rwmixread={0,100,70}
iodepth=256
time_based=1
ramp_time=60
runtime=300

[filename1]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1
filename=Nvme4n1
filename=Nvme5n1
filename=Nvme6n1
filename=Nvme7n1
```

3core:

1st initiator:

```
[global]
ioengine=examples/bdev/fio_plugin/fio_plugin
spdk_conf=bdev.conf
thread=1
group_reporting=1
direct=1
norandommap=1
bs=4k
rw=randrw
rwmixread={100,70,0}
iodepth=256
time_based=1
ramp_time=60
runtime=300

[filename1]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1
filename=Nvme4n1
filename=Nvme5n1
filename=Nvme6n1
filename=Nvme7n1
```

2nd initiator:

```
[global]
ioengine=examples/bdev/fio_plugin/fio_plugin
spdk_conf=bdev.conf
thread=1
group_reporting=1
direct=1
norandommap=1
bs=4k
rw=randrw
rwmixread={0,100,70}
iodepth=128
time_based=1
ramp_time=60
runtime=300

[filename1]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
```

	filename=Nvme3n1 [filename2] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 **4core:** [global] ioengine=examples/bdev/fio_plugin/fio_plugin spdk_conf=bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0} iodepth=128 time_based=1 ramp_time=60 runtime=300 [filename1] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 [filename2] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1

Results in the table represent aggregate performance (IOPS & Avg. latency) observed:

Test Result: 4K 100% Random Read

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1263	306.4
2 cores	2555	189.83
3 cores	3529	131.37
4 cores	4091	124.36

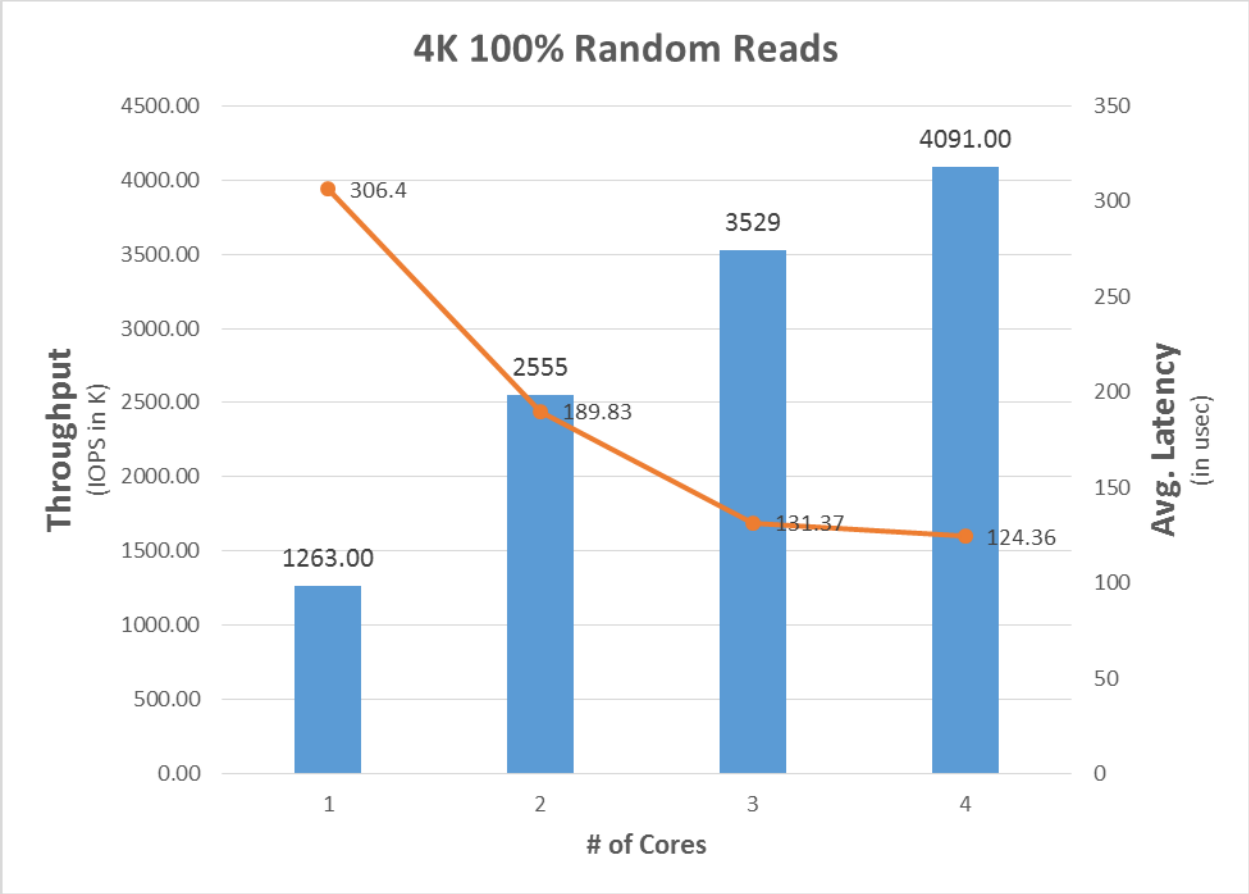

Figure 5: SPDK NVMe-oF Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random read workload

Note: Drives were not pre-conditioned while running the 100% random write I/O test. This helps scale throughput and thereby avoids storage bottlenecks especially when testing the 3 and 4 cores case until hitting 2x 100GbE bottleneck.

Test Result: 4K 100% Random Write

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1257.0	120.7
2 cores	2716.6	112.6
3 cores	3820.3	100.18
4 cores	4242.0	114.8

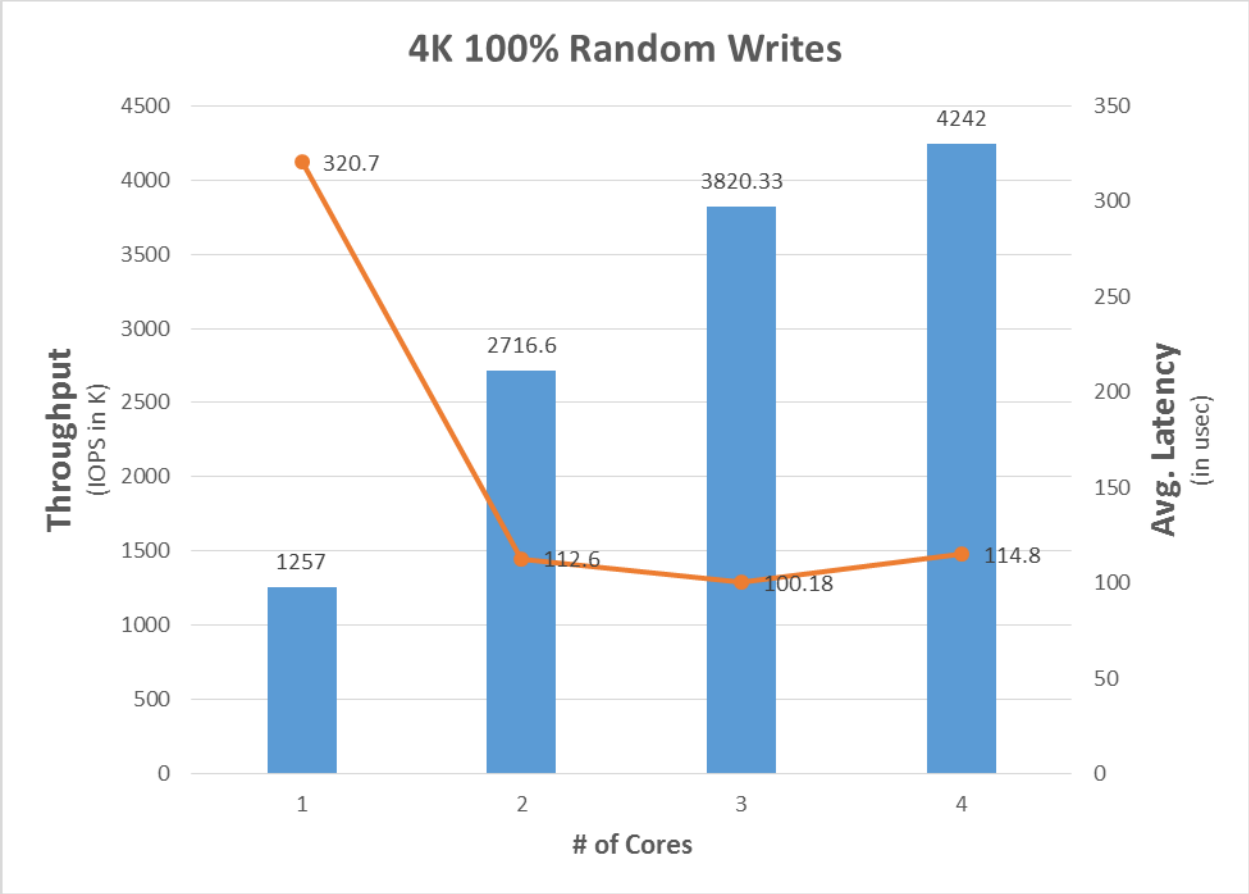

Figure 6: SPDK NVMe-oF Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random write workload

Test Result: 4K 70% Random Read 30% Random Write

# of Cores	Throughput (IOPS)	Avg. Latency (usec)
1 core	1363.00	308.56
2 cores	2018.00	252.19
3 cores	2381.67	211.91
4 cores	2124.33	240.71

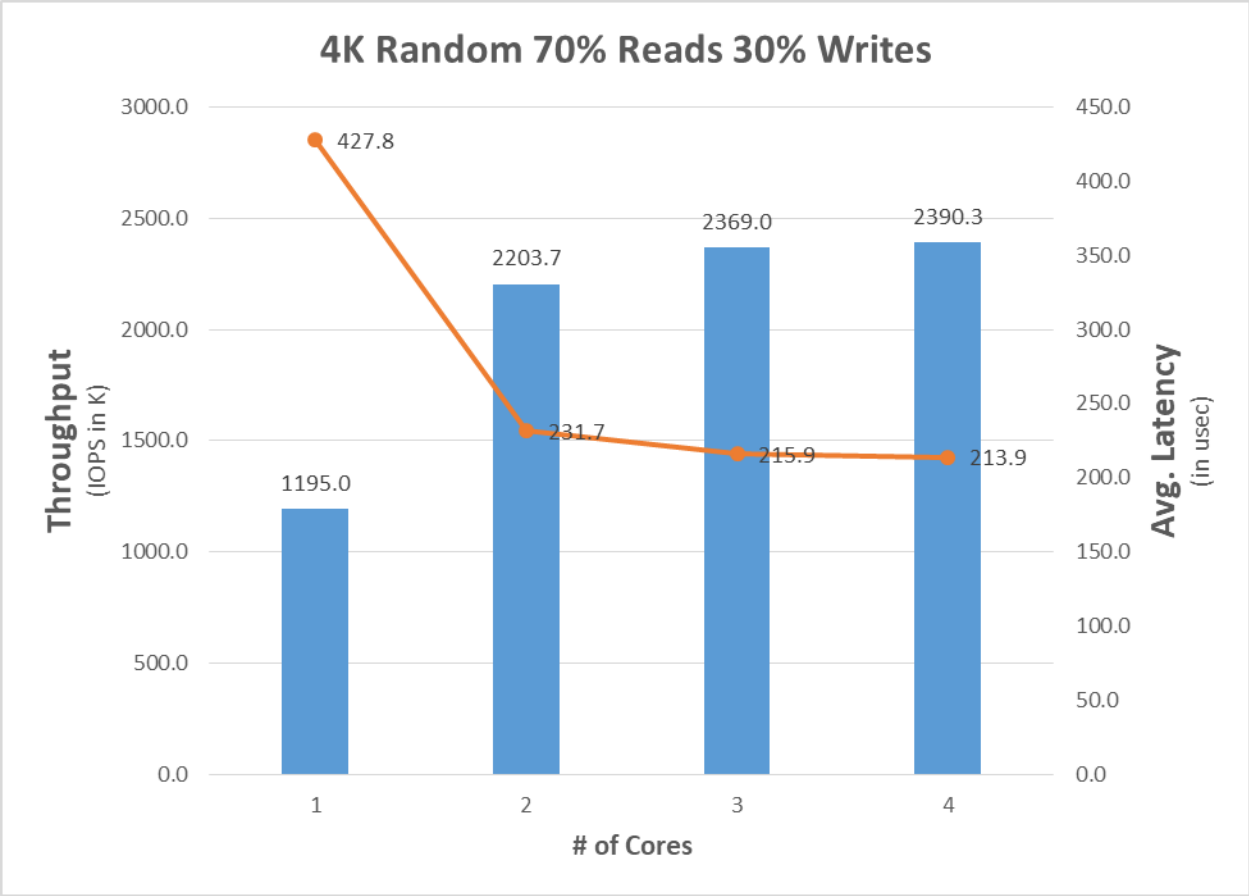

Figure 7: SPDK NVMe-oF Initiator I/O core scaling: IOPS vs. Latency while running 4KB Random 70% read 30% write workload

Conclusion:

1. For 100% Reads and 100% Writes, throughput scales up and latency decreases almost linearly with the addition of CPU cores until hitting network limit.
2. For 4K Random 70% Reads, 30% Writes performance doesn't scale linearly due to some other bottleneck. See test case #1 for additional discussion.

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

This test case was designed to compare the latency of the SPDK NVMe-oF target and initiator vs. the Linux Kernel NVMe-oF target and initiator using a single NVMe-oF subsystem. Average I/O latency and 99th percentile latency is reported. The SPDK NVMe-oF target was configured to run on a single NVMe-oF subsystem with a single namespace backed by a ***null block device***, running on a single core. A null block device (bdev) was chosen as the backend block device to avoid media latency during these tests.

Linux Kernel NVMe-oF Target vs. SPDK Kernel NVMe-oF Target

Item	Description
Test Case	NVMe-oF Target Latency
Test configuration	**SPDK** **Nvmf.conf:** # NVMf Target Configuration File [Global] ReactorMask 0x1 [Rpc] Enable Yes Listen 127.0.0.1 [Nvmf] MaxQueuesPerSession 89 AcceptorPollRate 10000 [Null] Dev Nvme0n1 102400 4096 [Subsystem1] NQN nqn.2016-06.io.spdk:cnode1 Listen RDMA 192.168.200.2:4420 AllowAnyHost Yes SN SPDK00000000000001 Namespace Nvme0n1 1 **BDEV.conf** [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode1" Nvme0 **Linux Kernel** Nvmetcli tool was used to configure Kernel NVMe-oF target. Backend used was /dev/nullb0 **NVMe-oF Initiator** Nvme-cli tool. Default # of I/O queues per subsystem
FIO configuration	**Linux Kernel Initiator** [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0} iodepth=1 time_based=1 ramp_time=30 runtime=300 numjobs=1 [filename1] filename=/dev/nvme0n1

This following data was collected using Kernel initiator against both SPDK & Kernel Nvme-oF target.

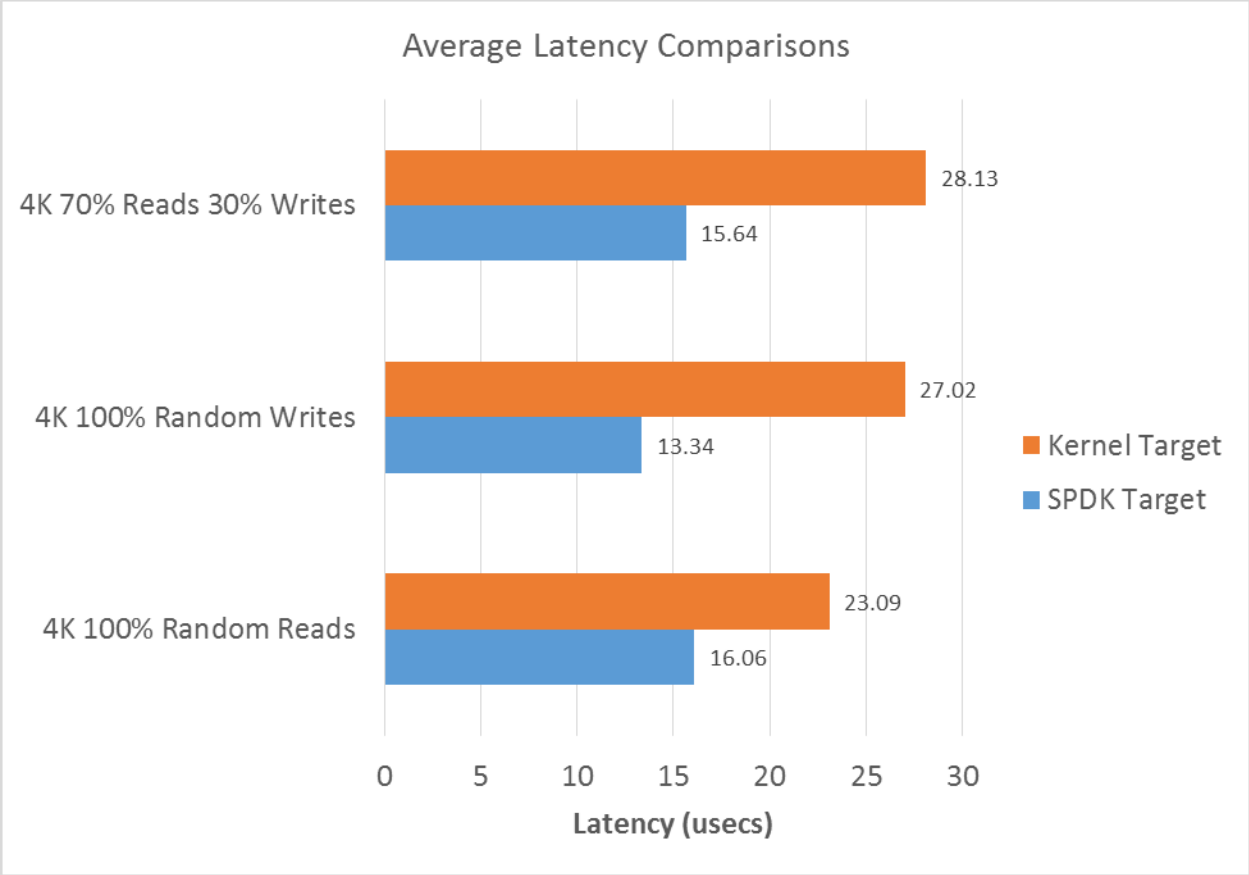

Figure 8: Average I/O Latency Comparisons b/w SPDK vs. Kernel NVMe-oF Target for various workloads (Kernel Initiator)

SPDK NVMe-oF Target

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	16.06	60000	16.1
4K 100% Random Writes IOPS	13.34	71000	13.5
4K 100% Random 70% Reads 30% Writes IOPS	15.64	62000	15.6

Linux Kernel NVMe-oF Target

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	23.09	42000	23.6
4K 100% Random Writes IOPS	27.02	35000	64
4K 100% Random 70% Reads 30% Writes IOPS	28.13	32800	95

Conclusion:

1. SPDK NVMe-oF Target average round trip I/O latency (reads/writes) is up to 13usec faster than the Linux kernel NVMe-oF target. This is entirely software overhead.

Linux Kernel NVMe-oF Initiator vs. SPDK NVMe-oF Initiator

Item	Description
Test Case	NVMe-oF Initiator Latency
Test configuration	**SPDK** **Nvmf.conf:** # NVMf Target Configuration File [Global] ReactorMask 0x1 [Rpc] Enable Yes Listen 127.0.0.1 [Nvmf] MaxQueuesPerSession 89 AcceptorPollRate 10000 [Null] Dev Nvme0n1 102400 4096 [Subsystem1] NQN nqn.2016-06.io.spdk:cnode1 Listen RDMA 192.168.200.2:4420 AllowAnyHost Yes SN SPDK00000000000001 Namespace Nvme0n1 1 **BDEV.conf** [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode1" Nvme0 **Kernel NVMe-oF Initiator** Nvme-cli tool. Default # of I/O queues per subsystem
FIO configuration	**Linux Kernel** [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0} iodepth=1 time_based=1 ramp_time=30 runtime=300 numjobs=1 [filename1] filename=/dev/nvme0n1 **SPDK** [global] ioengine=examples/bdev/fio_plugin/fio_plugin spdk_conf=bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={0,100,70} iodepth=1 time_based=1 ramp_time=60 runtime=300 [filename1] filename=Nvme0n1

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

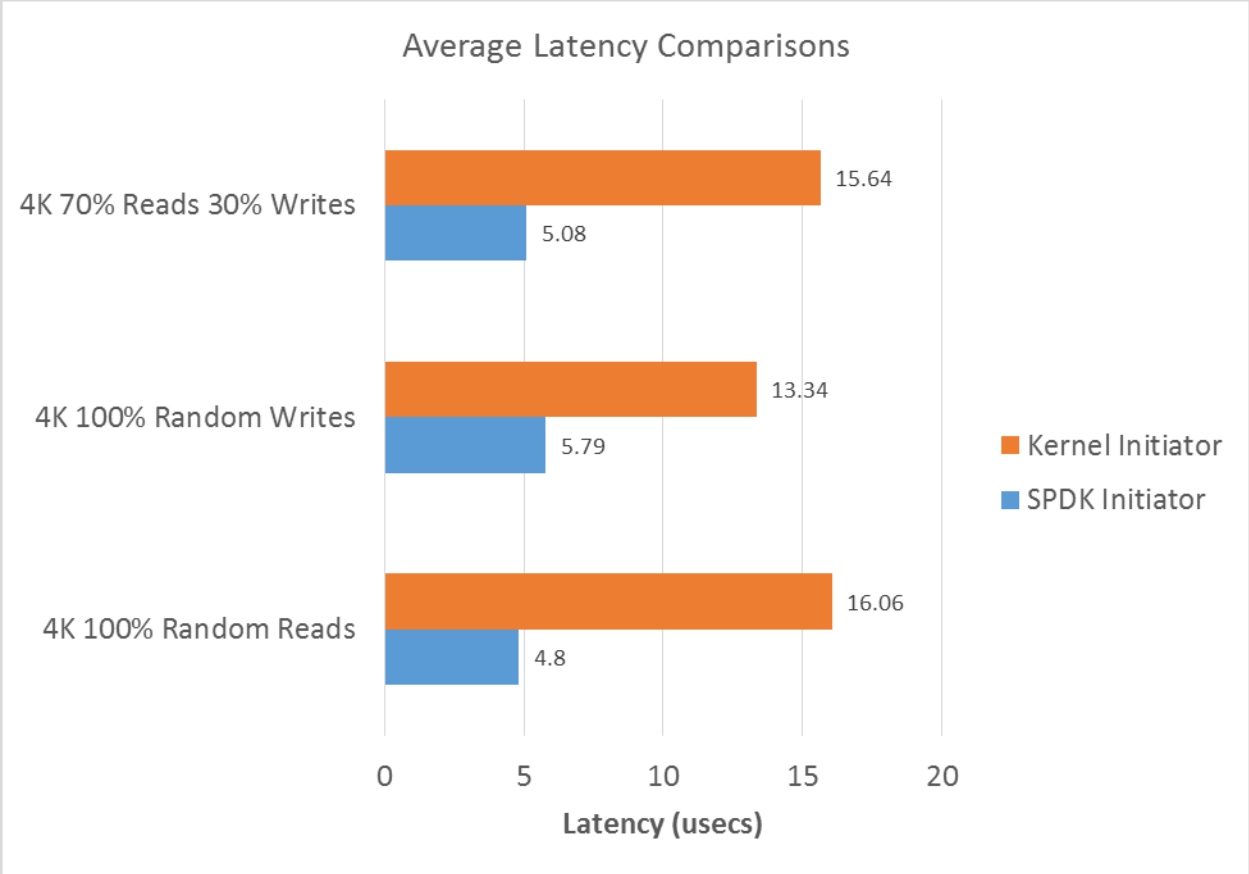

Figure 9: Average I/O Latency Comparisons b/w SPDK vs. Kernel NVMe-oF Initiator for various workloads (SPDK Target)

Linux Kernel NVMe-oF Initiator

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	16.06	60000	16.1
4K 100% Random Writes IOPS	13.34	71000	13.5
4K 100% Random 70% Reads 30% Writes IOPS	15.64	62000	15.6

SPDK NVMe-oF Initiator

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	4.8	200000	4.83
4K 100% Random Writes IOPS	5.79	167000	5.9
4K 100% Random 70% Reads 30% Writes IOPS	5.08	187100	5.1

Conclusion:

1. SPDK NVMe-oF Initiator average round trip I/O latency (reads/writes) is up to 3x as fast as Kernel NVMe-oF Initiator using null bdev backend.

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. 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. Number of CPU cores metric was calculated from %CPU utilization measured using sar (systat package in linux). SPDK NVMe-oF Target was configured to run on 4 cores, 16 NVMe-oF subsystems (1 per Intel P4600) and 2 initiators were used both running I/Os to 8 separate subsystems using Kernel NVMe-oF initiator.

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

Item	Description
Test Case	NVMe-oF Target performance under varying # of connections
Test configuration	**SPDK** **Nvmf.conf:** Same as used in Test Case 1 **BDEV.conf** [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode1" Nvme0 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode2" Nvme1 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode3" Nvme2 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.100.2 trsvcid:4420 subnqn:nqn.2016-06.io.spdk:cnode4" Nvme3 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode5" Nvme4 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode6" Nvme5 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode7" Nvme6 TransportId "trtype:RDMA adrfam:IPv4 traddr:192.168.101.2 trsvcid:4421 subnqn:nqn.2016-06.io.spdk:cnode8" Nvme7 **Linux Kernel** Nvmetcli tool was used to configure Kernel NVMe-oF target **NVMe-oF Initiator** Nvme-cli tool. Default # of I/O queues per subsystem
FIO configuration	**Kernel** [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 bs=4k rw=randrw rwmixread={100,70,0} iodepth=32 time_based=1 ramp_time=30 runtime=300 numjobs={1,4,16} [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 [filename8] filename=/dev/nvme7n1

Number of CPU cores used while running SPDK Nvme-oF target were 4, whereas for the case of linux Kernel Nvme-oF target there was no cpu core limitation applied.

Numbers in the graph represent relative performance which are in terms of 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.

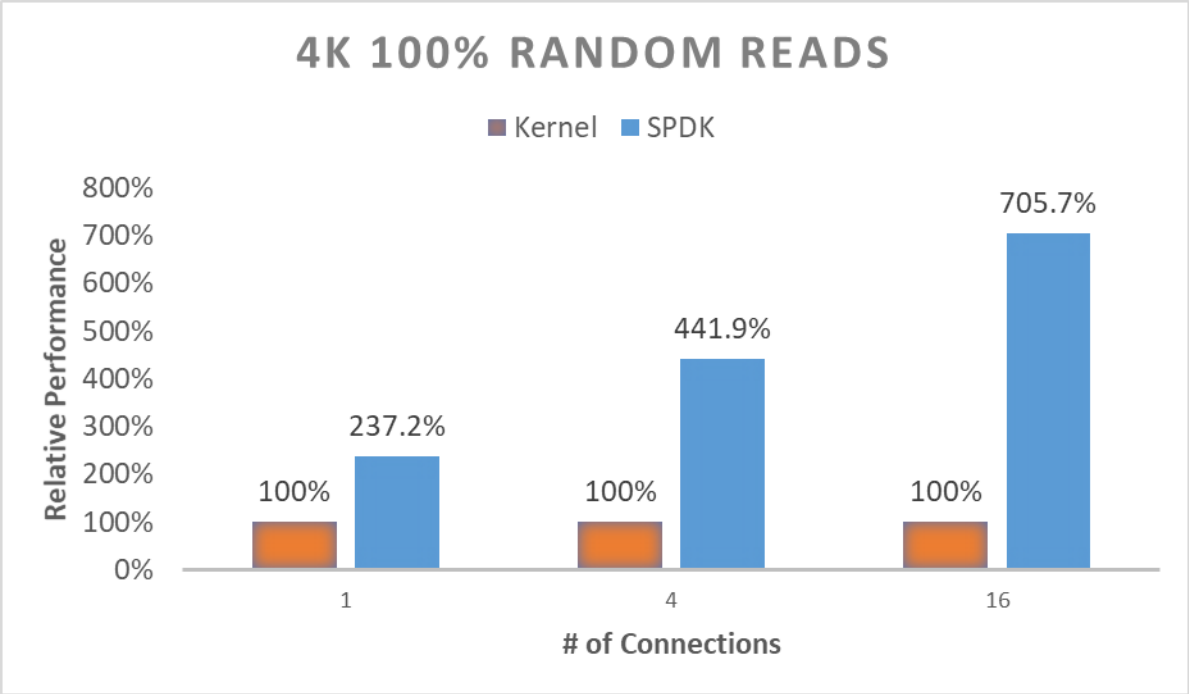

Figure 10: Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K 100% Random Reads (Kernel Initiator)

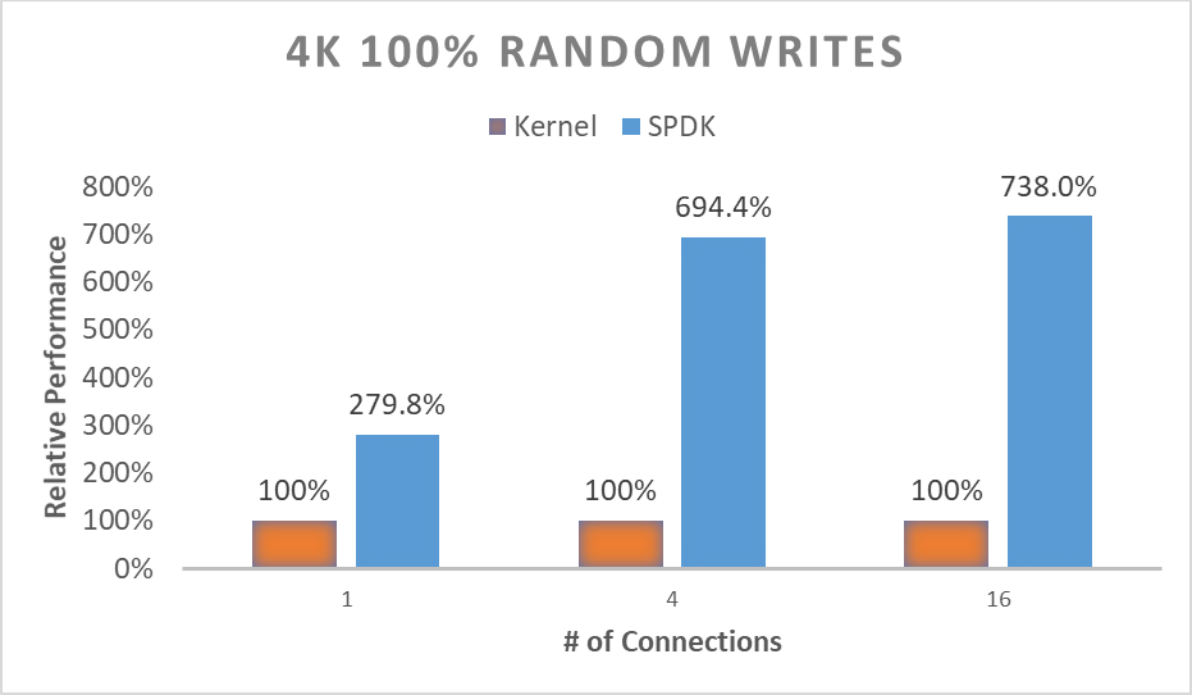

Figure 11: Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K 100% Random Writes

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

For 4K Random 70% reads, 30% writes workload it was noticed that performance didn't increase when going from 4 to 16 connections per subsystem. This was due to storage/platfrom bottleneck noticed as described in test case 1 and 2.

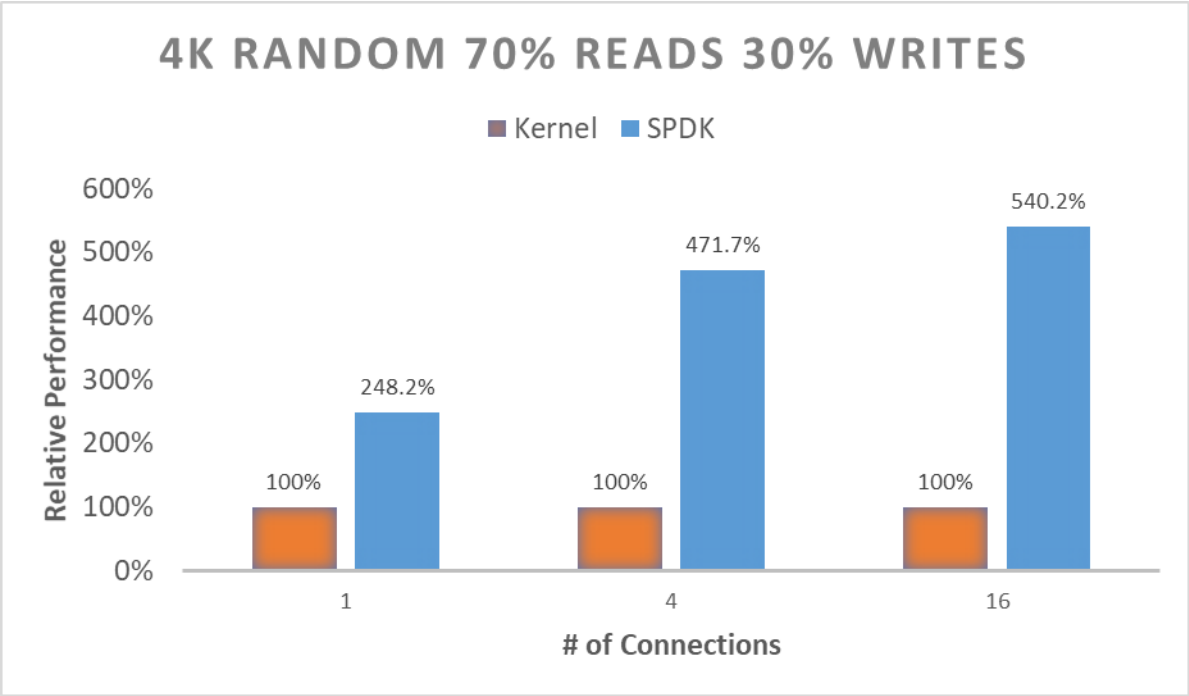

Figure 12: Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K Random 70% Reads 30% Writes

Linux Kernel NVMe-oF Target: 4K 100% Random Reads

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	216	2362	10
4	820.5	2494	18
16	3684.1	2223	22.4

Linux Kernel NVMe-oF Target: 4K 100% Random Writes

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	152	3352	11.2
4	498	4106	25.2
16	1909	4289	28

Linux Kernel NVMe-oF Target: 4K 70% Random Read 30% Random Write

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	209	2439	10
4	647.3	3160	21
16	2877.5	2846	23.5

4K 100% Random read performance was best when # of connections was 4 per subsystem with iodepth=32 per subsystem.

SPDK NVMe-oF Target: 4K 100% Random Reads

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	227	2241	4
4	835	2449	4
16	3524	2324	4

SPDK NVMe-oF Target: 4K 100% Random Writes

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	157	3350	4
4	470	4526	4
16	1811	4522	4

SPDK NVMe-oF Target: 4K 70% Random Read 30% Random Write

# of Connections	Average Latency (usec)	IOPS	# of CPU cores utilized
1	210.8	2421	4

4	720.7	2839	4
16	3130	2617	4

4K 70% Random Reads 30% Random Write performance was best when # of connections was 4 and iodepth = 4 per subsystem. This seem to be the optimum configuration while achieving max performance as was seen for 4K 100% Random Reads case as well.

Conclusion:

1. SPDK NVMe-oF target performs up to 7.3x better w.r.t IOPS/core than linux kernel NVMe-oF target while running 4K 100% random write workload with increasing number of connections (16) per NVMe-oF subsystem.
2. SPDK NVMe-oF target performs up to 5.8x and 5.4x better than linux kernel NVMe-oF target while running 4K 100% random reads and 4K random 70% reads 30% writes respectively.

Summary

This report showcased performance results with SPDK NVMe-oF target and initiator under various test cases, including I/O core scaling, average I/O latency, and 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. It showcased that throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF target and initiator I/O cores until hitting network bottleneck for 4KB random 100% read and 100% write I/O workloads. It was also observed that SPDK NVMe-oF initiator is 3x faster than Kernel NVMe-oF initiator with null bdev based backend. Also, SPDK NVMe-oF target performed up to 7.3x better w.r.t IOPS/core than linux Kernel NVMe-oF target while running 4K 100% random write workload with increasing number of connections (16) 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.

DISCLAIMERS

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more information go to http://www.intel.com/performance

Intel® AES-NI requires a computer system with an AES-NI enabled processor, as well as non-Intel software to execute the instructions in the correct sequence. AES-NI is available on select Intel® processors. For availability, consult your reseller or system manufacturer. **For more information, see http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/**

The benchmark results may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

§