SPDK NVMe-oF RDMA (Target & Initiator) Performance Report
Release 20.07

Testing Date: August 2020
Performed by  Karol Latecki (karol.latecki@intel.com)
               Maciej Wawryk (maciejx.wawryk@intel.com)

Acknowledgments:
Audience and Purpose

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

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

### Target Configuration

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Platform</td>
<td>SuperMicro SYS-2029U-TN24R4T</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel® Xeon® Gold 6230 Processor (27.5MB L3, 2.10 GHz)</td>
</tr>
<tr>
<td></td>
<td>Number of cores 20 per socket, number of threads 40 per socket (Both sockets populated)</td>
</tr>
<tr>
<td></td>
<td>Microcode: 0x500002c</td>
</tr>
<tr>
<td>Memory</td>
<td>12 x 32GB Hynix HMA84GR7AFR4N-VK, DDR4, 2666MHz</td>
</tr>
<tr>
<td></td>
<td>Total of 384GB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Fedora 30</td>
</tr>
<tr>
<td>BIOS</td>
<td>3.1a</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>5.4.14-100.fc30</td>
</tr>
<tr>
<td>SPDK version</td>
<td>SPDK 20.07</td>
</tr>
<tr>
<td>Storage</td>
<td>OS: 1x 120GB Intel SSDSC2BB120G4</td>
</tr>
<tr>
<td></td>
<td>Storage Target: 16x Intel® SSD DC P4610™ 1.6TB (FW: QDV10190) (8 on each CPU socket)</td>
</tr>
<tr>
<td>NIC</td>
<td>2x 100GbE Mellanox ConnectX-5 NICs. Both ports connected.</td>
</tr>
<tr>
<td></td>
<td>1 NIC per CPU socket.</td>
</tr>
</tbody>
</table>
## Initiator 1 Configuration

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Platform</td>
<td>Intel® Server System R2208WFTZSR</td>
</tr>
<tr>
<td>CPU</td>
<td><strong>Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz</strong> <em>(35.75MB Cache)</em>&lt;br&gt;Number of cores 24 per socket, number of threads 48 per socket (Both sockets populated)&lt;br&gt;Microcode: 0x500002c</td>
</tr>
<tr>
<td>Memory</td>
<td>6 x 32GB Micron M393A1G40EB1-CRC, DDR4, 2933MHz&lt;br&gt;Total 192GBs</td>
</tr>
<tr>
<td>Operating System</td>
<td>Fedora 30</td>
</tr>
<tr>
<td>BIOS</td>
<td>02.01.0008 03/19/2019</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>5.4.14-100.fc30</td>
</tr>
<tr>
<td>SPDK version</td>
<td>SPDK 20.07</td>
</tr>
<tr>
<td>Storage</td>
<td><strong>OS</strong>: 1x 240GB INTEL SSDSC2BB240G6</td>
</tr>
<tr>
<td>NIC</td>
<td>1x 100GbE Mellanox ConnectX-5 Ex NIC. Both ports connected to Target server.&lt;br&gt;(connected to CPU socket 0)</td>
</tr>
</tbody>
</table>

## Initiator 2 Configuration

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Platform</td>
<td>Intel® Server System R2208WFTZSR</td>
</tr>
<tr>
<td>CPU</td>
<td><strong>Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz</strong> <em>(35.75MB Cache)</em>&lt;br&gt;Number of cores 24 per socket, number of threads 48 per socket (Both sockets populated)&lt;br&gt;Microcode: 0x500002c</td>
</tr>
<tr>
<td>Memory</td>
<td>6 x 32GB Micron M393A1G40EB1-CRC, DDR4, 2933MHz&lt;br&gt;Total 192GBs</td>
</tr>
<tr>
<td>Operating System</td>
<td>Fedora 3002.01.0008 03/19/2019</td>
</tr>
<tr>
<td>BIOS</td>
<td>02.01.0008 03/19/2019-1 06/08/2018</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>5.4.14-100.fc30</td>
</tr>
<tr>
<td>SPDK version</td>
<td>SPDK 20.07</td>
</tr>
<tr>
<td>Storage</td>
<td><strong>OS</strong>: 1x 240GB INTEL SSDSC2BB240G6</td>
</tr>
<tr>
<td>NIC</td>
<td>1x 100GbE Mellanox ConnectX-5 Ex NIC. Both ports connected to Target server.&lt;br&gt;(connected to CPU socket 0)</td>
</tr>
</tbody>
</table>
BIOS settings

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOS (Applied to all 3 systems)</td>
<td>Hyper threading Enabled</td>
</tr>
<tr>
<td></td>
<td>CPU Power and Performance Policy:</td>
</tr>
<tr>
<td></td>
<td>• “Extreme Performance” for Target</td>
</tr>
<tr>
<td></td>
<td>• “Performance” for Initiators</td>
</tr>
<tr>
<td></td>
<td>CPU C-state No Limit</td>
</tr>
<tr>
<td></td>
<td>CPU P-state Enabled</td>
</tr>
<tr>
<td></td>
<td>Enhanced Intel® SpeedStep® Tech Enabled</td>
</tr>
<tr>
<td></td>
<td>Turbo Boost Enabled</td>
</tr>
</tbody>
</table>

Kernel & BIOS spectre-meltdown information

All three server systems use Fedora 5.4.14-100.fc30 kernel version available from DNF repository with default patches for spectre-meltdown issue enabled.

BIOS on all systems was updated to post spectre-meltdown versions as well.


**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 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, and iWARP. 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 P4610 SSDs which were used as block devices for NVMe-oF subsystems and two 100GbE Mellanox ConnectX-5 NICs connected to provide up to 200GbE of network bandwidth. Each Initiator system has one Mellanox ConnectX-5 Ex 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 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 performance testing setup
Test Case 1: SPDK NVMe-oF RDMA Target I/O core scaling

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

The SPDK NVMe-oF RDMA target was configured to run with 16 NVMe-oF subsystems. Each NVMe-oF subsystem ran on top of an individual bdev backed by a single Intel® SSD DC P4610 device. Each of the 2 initiators were connected to 8 individual NVMe-oF subsystems which were exposed via SPDK NVMe-oF Target over 1x 100GbE NIC. SPDK bdev FIO plugin was used to target 8 individual NVMe-oF bdevs on each of the initiators. SPDK Target Reactor Mask was configured to use 1, 2, 3, 4, 5 and 6 cores tests while running following workloads on each initiator:

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

The table below contains more information the test configuration. The SPDK NVMe-oF Target was configured using JSON-RPC; the table contains a 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. We preconditioned the SSDs once before running the 4KB Rand Read and 4KB Rand 70/30 Read/Write workloads to ensure that the SSDs reached their steady state where we get repeatable results. However, for the 4KB Rand Write workload we didn’t precondition the NVMe devices to ensure workload saturated the network rather than being limited to the steady state performance of the SSDs which is much lower than the available network bandwidth.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case</td>
<td>SPDK NVMe-oF Target I/O core scaling</td>
</tr>
<tr>
<td>SPDK NVMe-oF Target configuration</td>
<td>The commands below were executed with <code>spdk/scripts/rpc.py</code> script.</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme0 -a 0000:60:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme1 -a 0000:61:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme2 -a 0000:62:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme3 -a 0000:63:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme4 -a 0000:64:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme5 -a 0000:65:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme6 -a 0000:66:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme7 -a 0000:67:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme8 -a 0000:b5:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme9 -a 0000:b6:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme10 -a 0000:b7:00.0</td>
</tr>
<tr>
<td></td>
<td>construct_nvme_bdev -t PCIe -b Nvme11 -a 0000:b8:00.0</td>
</tr>
</tbody>
</table>
construct_nvme_bdev -t PCIe -b Nvme12 -a 0000:b9:00.0
construct_nvme_bdev -t PCIe -b Nvme13 -a 0000:ba:00.0
construct_nvme_bdev -t PCIe -b Nvme14 -a 0000:bb:00.0
construct_nvme_bdev -t PCIe -b Nvme15 -a 0000:bc:00.0

nvme_create_transport -t RDMA
(creates RDMA transport layer with default values:
  trtype: "RDMA"
  max_queue_depth: 128
  max_qpairs_per_ctrlr: 64
  in_capsule_data_size: 4096
  max_io_size: 131072
  io_unit_size: 8192
  max_aq_depth: 128
  num_shared_buffers: 4096
  buf_cache_size: 32)

for i in $(seq 1 16); do
  nvme_subsystem_create nqn.2018-09.io.spdk:cnode${i} -s SPDK00${i} -m 8
done

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 $(seq 1 4); do
    nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode$i -t rdma -f ipv4 -s 4420 -a $ip
          
    ((i++))
  done
done

BDEV.conf
[Nvme]
TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode0" Nvme0
TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme1

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, 8, 16, 32}
time_based=1
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
[filename7]
filename=Nvme7n1
## 4KB Random Read Results

**Test Result: 4KB 100% Random Read IOPS QD=64**

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>3952.67</td>
<td>1011.9</td>
<td>1011.6</td>
</tr>
<tr>
<td>2 cores</td>
<td>9151.72</td>
<td>2342.8</td>
<td>436.8</td>
</tr>
<tr>
<td>3 cores</td>
<td>15523.36</td>
<td>3974.0</td>
<td>257.0</td>
</tr>
<tr>
<td>4 cores</td>
<td>21043.94</td>
<td>5387.2</td>
<td>189.9</td>
</tr>
<tr>
<td>5 cores</td>
<td>21609.59</td>
<td>5532.1</td>
<td>185.1</td>
</tr>
<tr>
<td>6 cores</td>
<td>22026.52</td>
<td>5638.8</td>
<td>182.1</td>
</tr>
</tbody>
</table>

**Figure 2:** SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Read workload at QD=64
4KB Random Write Results

Test Result: 4KB 100% Random Writes IOPS QD=32

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>5569.08</td>
<td>1425.7</td>
<td>357.0</td>
</tr>
<tr>
<td>2 cores</td>
<td>12383.21</td>
<td>3170.1</td>
<td>160.7</td>
</tr>
<tr>
<td>3 cores</td>
<td>18883.67</td>
<td>4834.2</td>
<td>103.9</td>
</tr>
<tr>
<td>4 cores</td>
<td>22976.34</td>
<td>5881.9</td>
<td>85.5</td>
</tr>
<tr>
<td>5 cores</td>
<td>22848.22</td>
<td>5849.1</td>
<td>86.6</td>
</tr>
<tr>
<td>6 cores</td>
<td>22952.29</td>
<td>5875.8</td>
<td>86.5</td>
</tr>
</tbody>
</table>

Figure 3: SPDK NVMe-oF RDMA Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Write workload at QD=32
4KB Random Read-Write Results

Test Result: 4KB 70% Read 30% Write IOPS, QD=128

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>3964.99</td>
<td>1015.0</td>
<td>2017.1</td>
</tr>
<tr>
<td>2 cores</td>
<td>9059.51</td>
<td>2319.2</td>
<td>881.9</td>
</tr>
<tr>
<td>3 cores</td>
<td>14731.65</td>
<td>3771.3</td>
<td>557.1</td>
</tr>
<tr>
<td>4 cores</td>
<td>20609.63</td>
<td>5276.1</td>
<td>390.5</td>
</tr>
<tr>
<td>5 cores</td>
<td>22404.87</td>
<td>5735.6</td>
<td>356.6</td>
</tr>
<tr>
<td>6 cores</td>
<td>24785.04</td>
<td>6345.0</td>
<td>323.9</td>
</tr>
</tbody>
</table>

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

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

1. For all tested workloads throughput scales up and latency decreases almost linearly with the addition of I/O cores up to 4 cores. Adding more CPU cores results in small performance gain as network is almost saturated.
Large Sequential I/O Performance

128KB block size I/O tests were performed with sequential I/O workloads at queue depth 8. The rest of the FIO configuration is similar to the 4KB test case in the previous part of this document. We used iodepth=8 (iodepth=1 for write workload) because higher queue depth resulted in negligible bandwidth gain and a significant increase in the latency.

Test Result: 128KB 100% Sequential Reads QD=8

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>24739.41</td>
<td>197.9</td>
<td>646.7</td>
</tr>
<tr>
<td>2 cores</td>
<td>24687.37</td>
<td>197.5</td>
<td>648.1</td>
</tr>
<tr>
<td>3 cores</td>
<td>24796.48</td>
<td>198.4</td>
<td>645.1</td>
</tr>
<tr>
<td>4 cores</td>
<td>24863.67</td>
<td>198.9</td>
<td>643.3</td>
</tr>
</tbody>
</table>

Test Result: 128KB 100% Sequential Writes QD=1

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>22967.66</td>
<td>183.7</td>
<td>86.9</td>
</tr>
<tr>
<td>2 cores</td>
<td>23184.38</td>
<td>185.5</td>
<td>86.1</td>
</tr>
<tr>
<td>3 cores</td>
<td>23187.66</td>
<td>185.5</td>
<td>86.1</td>
</tr>
<tr>
<td>4 cores</td>
<td>23200.40</td>
<td>185.6</td>
<td>86.0</td>
</tr>
</tbody>
</table>

Test Result: 128KB 70% Reads 30% Writes QD=8

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>25614.07</td>
<td>204.9</td>
<td>627.9</td>
</tr>
<tr>
<td>2 cores</td>
<td>26796.34</td>
<td>214.4</td>
<td>601.0</td>
</tr>
<tr>
<td>3 cores</td>
<td>25763.89</td>
<td>206.1</td>
<td>626.4</td>
</tr>
<tr>
<td>4 cores</td>
<td>29917.93</td>
<td>239.3</td>
<td>537.8</td>
</tr>
</tbody>
</table>
Conclusions

1. For large sequential I/Os, a single CPU core saturated the network bandwidth. The SPDK NVMe-oF target running on 1 core does close to 24-25 GBps of bandwidth in all tested workloads. Therefore, adding more CPU cores did not result in increased performance for these workloads because the network was the bottleneck.
**Test Case 2: SPDK NVMe-oF RDMA Initiator I/O core scaling**

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

The SPDK NVMe-oF RDMA Target was configured using 6 cores; all the other configurations are similar to test case 1. The SPDK bdev FIO plugin was used to target 8 NVMe-oF bdevs on each of the 2 initiators. The following workloads were executed on both of the initiators using fio in client-server mode.

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

Depending on the number of initiators and initiator cores, we varied the number of NVMe-oF subsystems exported by the target as shown below, instead of exposing all 16 NVMe-oF subsystems.

1 core: 1 initiator running on a single core connecting to 4 subsystems.

2 cores: 2 initiators, each running on a single core. Each initiator connected to 4 subsystems.

3 cores: 2 initiators, the first one running on 1 core and other one running on 2 cores. Initiator 1 connected to 4 subsystems and initiator 2 connected to 8 subsystems.

4 cores: 2 initiators, both running on 2 cores each. Both initiators connected to 8 subsystems.

This was done to avoid a situation where single initiator would connect to all 16 target subsystems, which resulted in unnatural latency spike in the 1 CPU core test case.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case</strong></td>
<td>SPDK NVMe-oF RDMA Initiator I/O core scaling</td>
</tr>
<tr>
<td><strong>SPDK NVMe-oF Target configuration</strong></td>
<td>Same as in Test Case #1, using 6 CPU cores.</td>
</tr>
<tr>
<td><strong>SPDK NVMe-oF Initiator 1 - FIO plugin configuration</strong></td>
<td>BDEV.conf</td>
</tr>
<tr>
<td></td>
<td>[Nvme]</td>
</tr>
<tr>
<td></td>
<td>TransportId &quot;trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1&quot; Nvme0</td>
</tr>
<tr>
<td></td>
<td>TransportId &quot;trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2&quot; Nvme1</td>
</tr>
<tr>
<td></td>
<td>TransportId &quot;trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5&quot; Nvme4</td>
</tr>
<tr>
<td><strong>FIO.conf</strong></td>
<td>[global]</td>
</tr>
<tr>
<td></td>
<td>ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin</td>
</tr>
<tr>
<td></td>
<td>spdk_conf=/tmp/spdk/bdev.conf</td>
</tr>
</tbody>
</table>
thread=1
group_reporting=1
direct=1

norandommap=1
rw=randrw
rwmixread={100, 70, 0}
bs=4k
iodepth={32, 64, 128, 256}
time_based=1
ramp_time=60
runtime=300

{filename_section}

**FIO.conf filename section for 1 & 2 CPU test run**

[filename0]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1

**FIO.conf filename section for 3 & 4 CPU test run**

[filename0]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1

Similar as Initiator 1. The only differences are were in the filename section which are shown below.

**FIO.conf filename section for 1 CPU test run**
N/A (only Initiator 1 is used)

**FIO.conf filename section for 2 CPU test run**

[filename0]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1
FIO.conf filename section for 3 CPU test run
[filename0]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1

FIO.conf filename section for 4 CPU test run
[filename0]
filename=Nvme0n1
filename=Nvme1n1
filename=Nvme2n1
filename=Nvme3n1
[filename1]
filename=Nvme4n1
filename=Nvme5n1
filename=Nvme6n1
filename=Nvme7n1
4KB Random Read Results

Test Result: 4KB 100% Random Read, QD=64, SPDK Target 6 CPU cores

<table>
<thead>
<tr>
<th># of Initiator CPU Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>5793.48</td>
<td>1483.13</td>
<td>558.01</td>
</tr>
<tr>
<td>2 cores</td>
<td>15514.13</td>
<td>3971.61</td>
<td>225.96</td>
</tr>
<tr>
<td>3 cores</td>
<td>18572.74</td>
<td>4754.62</td>
<td>206.21</td>
</tr>
<tr>
<td>4 cores</td>
<td>22211.27</td>
<td>5686.08</td>
<td>179.67</td>
</tr>
<tr>
<td>5 cores</td>
<td>22358.36</td>
<td>5723.74</td>
<td>178.79</td>
</tr>
<tr>
<td>6 cores</td>
<td>22544.87</td>
<td>5771.48</td>
<td>177.78</td>
</tr>
<tr>
<td>7 cores</td>
<td>21443.51</td>
<td>5489.54</td>
<td>186.76</td>
</tr>
<tr>
<td>8 cores</td>
<td>21433.95</td>
<td>5487.09</td>
<td>186.83</td>
</tr>
</tbody>
</table>

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

*Note:* The SSDs were pre-conditioned just once with 2x Sequential Write workload before running all the 100% Random Write test cases. This allowed the throughput to scale to the 2x 100GbE network bandwidth when testing with 3 and 4 CPU cores rather than limiting the workload performance to the storage bottleneck (which is approx. 3.2M IOPS).

**Test Result: 4KB 100% Random Write, QD=32, SPDK Target 6 CPU Cores**

<table>
<thead>
<tr>
<th># of Initiator CPU Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>5709.81</td>
<td>1461.71</td>
<td>283.33</td>
</tr>
<tr>
<td>2 cores</td>
<td>13410.36</td>
<td>3433.05</td>
<td>119.96</td>
</tr>
<tr>
<td>3 cores</td>
<td>18071.06</td>
<td>4626.19</td>
<td>102.85</td>
</tr>
<tr>
<td>4 cores</td>
<td>22718.49</td>
<td>5815.93</td>
<td>85.53</td>
</tr>
<tr>
<td>5 cores</td>
<td>22638.64</td>
<td>5795.49</td>
<td>86.44</td>
</tr>
<tr>
<td>6 cores</td>
<td>22958.68</td>
<td>5877.42</td>
<td>85.99</td>
</tr>
<tr>
<td>7 cores</td>
<td>22960.78</td>
<td>5877.96</td>
<td>86.12</td>
</tr>
<tr>
<td>8 cores</td>
<td>22866.44</td>
<td>5853.81</td>
<td>86.64</td>
</tr>
</tbody>
</table>

![Figure 6: SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Write workload at QD=32](image-url)
## 4KB Random 70/30 Read/Write Results

Test Result: 4KB 70% Random Read 30% Random Write QD=128, SPDK Target 6 CPU Cores

<table>
<thead>
<tr>
<th># of Initiator CPU Cores</th>
<th>Bandwidth (MBps)</th>
<th>Throughput (IOPS k)</th>
<th>Avg. Latency (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>4849.21</td>
<td>1241.39</td>
<td>1331.20</td>
</tr>
<tr>
<td>2 cores</td>
<td>11241.22</td>
<td>2877.75</td>
<td>590.86</td>
</tr>
<tr>
<td>3 cores</td>
<td>17508.32</td>
<td>4482.12</td>
<td>464.71</td>
</tr>
<tr>
<td>4 cores</td>
<td>21419.95</td>
<td>5483.50</td>
<td>372.42</td>
</tr>
<tr>
<td>5 cores</td>
<td>21889.43</td>
<td>5603.69</td>
<td>361.96</td>
</tr>
<tr>
<td>6 cores</td>
<td>21405.97</td>
<td>5479.92</td>
<td>373.87</td>
</tr>
<tr>
<td>7 cores</td>
<td>20961.55</td>
<td>5366.15</td>
<td>383.55</td>
</tr>
<tr>
<td>8 cores</td>
<td>21355.74</td>
<td>5467.06</td>
<td>379.39</td>
</tr>
</tbody>
</table>

![4KB 70/30% Random Read/Write](image)

**Figure 7:** SPDK NVMe-oF RDMA Initiator I/O core scaling: IOPS vs. Latency while running 4KB Random 70% Read 30% Write workload at QD=128
Conclusions

1. IOPS scaling was linear or close to linear for all tested workloads as we increased the number of initiator cores up to 4 CPU cores. Beyond 4 CPU cores there was no increase in IOPS because the 200 Gbps network was saturated.

2. Peak performance of 5.7 million IOPS was reached at 4 Initiator CPU cores for random read workload, 5.8 million at 4 CPU cores for random write workload and 5.6 million at 5 CPU cores for mixed random read/write workload.
### Test Case 3: Linux Kernel vs. SPDK NVMe-of RDMA Latency

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case</td>
<td>Linux Kernel vs. SPDK NVMe-of RDMA Latency</td>
</tr>
<tr>
<td>Test configuration</td>
<td></td>
</tr>
<tr>
<td>SPDK NVMe-of Target configuration</td>
<td>The following commands are executed with spdk/scripts.rpc.py script to configure the SPDK NVMe-of target.</td>
</tr>
<tr>
<td></td>
<td><code>nvmmf_create_transport -t RDMA</code> (creates RDMA transport layer with default values:</td>
</tr>
<tr>
<td></td>
<td><code>trtype&quot;: &quot;RDMA&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>max_queue_depth: 128</code></td>
</tr>
<tr>
<td></td>
<td><code>max_qpairs_per_ctrlr: 64</code></td>
</tr>
<tr>
<td></td>
<td><code>in_capsule_data_size: 4096</code></td>
</tr>
<tr>
<td></td>
<td><code>max_io_size: 131072</code></td>
</tr>
<tr>
<td></td>
<td><code>io_unit_size: 8192</code></td>
</tr>
<tr>
<td></td>
<td><code>max_aq_depth: 128</code></td>
</tr>
<tr>
<td></td>
<td><code>num_shared_buffers: 4096</code></td>
</tr>
<tr>
<td></td>
<td><code>buf_cache_size: 32</code>)</td>
</tr>
<tr>
<td></td>
<td><code>construct_null_bdev NvmeOn1 10240 4096</code></td>
</tr>
<tr>
<td></td>
<td><code>nvmmf_subsystem_create nqn.2018-09.io.spdk:cnode1 -s SPDK001 -a -m 8</code></td>
</tr>
<tr>
<td></td>
<td><code>nvmmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode1 Nvme0n1</code></td>
</tr>
<tr>
<td></td>
<td><code>nvmmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t rdma -f ipv4 -s 4420 -a 20.0.0.1</code></td>
</tr>
<tr>
<td>Kernel NVMe-of Target configuration</td>
<td>The following target configuration file loaded using nvmet-cli tool.</td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td>&quot;ports&quot;: [</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;addr&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;adrfam&quot;: &quot;ipv4&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;traddr&quot;: &quot;20.0.0.1&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;trsvcid&quot;: &quot;4420&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;trtype&quot;: &quot;rdma&quot;</td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>&quot;portid&quot;: 1,</td>
</tr>
<tr>
<td></td>
<td>&quot;referrals&quot;: []},</td>
</tr>
</tbody>
</table>
"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"
]

<table>
<thead>
<tr>
<th><strong>FIO configuration</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPDK NVMe-oF Initiator FIO plugin configuration</strong></td>
</tr>
<tr>
<td><strong>BDEV.conf</strong></td>
</tr>
<tr>
<td><strong>FIO.conf</strong></td>
</tr>
<tr>
<td>[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</td>
</tr>
<tr>
<td>[filename0] filename=Nvme0n1</td>
</tr>
<tr>
<td><strong>Kernel initiator configuration</strong></td>
</tr>
<tr>
<td><strong>Device config</strong></td>
</tr>
<tr>
<td>The following configuration was performed using nvme-cli tool. modprobe nvme-fabrics nvme connect –n nqn.2018-09.io.spdk:cnode1 –t rdma –a 20.0.0.1 –s 4420</td>
</tr>
<tr>
<td>FIO.conf</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>[global] ioengine=libaio</td>
</tr>
<tr>
<td>thread=1</td>
</tr>
<tr>
<td>group_reporting=1</td>
</tr>
<tr>
<td>direct=1</td>
</tr>
<tr>
<td>norandommap=1</td>
</tr>
<tr>
<td>rw=randrw</td>
</tr>
<tr>
<td>rwmixread={100, 70, 0}</td>
</tr>
<tr>
<td>bs=4k</td>
</tr>
<tr>
<td>iodepth=1</td>
</tr>
<tr>
<td>time_based=1</td>
</tr>
<tr>
<td>ramp_time=60</td>
</tr>
<tr>
<td>runtime=300</td>
</tr>
<tr>
<td>[filename0]</td>
</tr>
<tr>
<td>filename=/dev/nvme0n1</td>
</tr>
</tbody>
</table>
SPDK vs Kernel NVMe-oF RDMA Target Results

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

![Average Latency Comparisons SPDK Target vs Kernel Target](image)

**Figure 8**: Average I/O Latency comparisons at QD=1 between SPDK and Linux Kernel NVMe-oF RDMA Target for various workloads using the Linux Kernel Initiator

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>Average Latency (usec)</th>
<th>IOPS</th>
<th>p99 (usec)</th>
<th>p99.9 (usec)</th>
<th>p99.99 (usec)</th>
<th>p99.999 (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB 100% Random Read</td>
<td>12.61</td>
<td>77744</td>
<td>17.4</td>
<td>26.0</td>
<td>38.4</td>
<td>143.5</td>
</tr>
<tr>
<td>4KB 100% Random Write</td>
<td>11.31</td>
<td>86388</td>
<td>17.6</td>
<td>26.0</td>
<td>35.9</td>
<td>143.7</td>
</tr>
<tr>
<td>4KB 70/30% Random Read/Write</td>
<td>13.26</td>
<td>74042</td>
<td>20.4</td>
<td>27.2</td>
<td>38.3</td>
<td>131.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>Average Latency (usec)</th>
<th>IOPS</th>
<th>p99 (usec)</th>
<th>p99.9 (usec)</th>
<th>p99.99 (usec)</th>
<th>p99.999 (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB 100% Random Read</td>
<td>17.51</td>
<td>56220</td>
<td>21.5</td>
<td>30.9</td>
<td>35.3</td>
<td>73.3</td>
</tr>
<tr>
<td>4KB 100% Random Write</td>
<td>16.23</td>
<td>60649</td>
<td>20.3</td>
<td>29.7</td>
<td>35.8</td>
<td>67.8</td>
</tr>
<tr>
<td>4KB 70/30% Random Read/Write</td>
<td>17.57</td>
<td>56099</td>
<td>19.2</td>
<td>29.3</td>
<td>33.0</td>
<td>68.1</td>
</tr>
</tbody>
</table>
Conclusions

1. For the RDMA transport, the SPDK NVMe-oF Target reduces the NVMe-oF average round trip I/O latency (reads/writes) by up to 5 usec vs. the Linux Kernel NVMe-oF target used in Fedora 30 5.4.14 setup. This is entirely software overhead, therefore, using the SPDK NVMe-oF target reduces the NVMe-oF software overhead by approximately 30% vs. the Linux Kernel NVMe-oF target.
SPDK vs Kernel NVMe-oF RDMA Initiator Results

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

**Figure 9:** Average I/O Latency Comparisons at QD=1 between SPDK and Linux Kernel NVMe-oF RDMA Initiator for various workloads against SPDK NVMe-OF Target

### SPDK NVMe-oF RDMA Initiator QD=1 Latency for a Null block device

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>Average Latency (usec)</th>
<th>IOPS</th>
<th>p99 (usec)</th>
<th>p99.9 (usec)</th>
<th>p99.99 (usec)</th>
<th>p99.999 (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB 100% Random Read</td>
<td>4.47</td>
<td>215081</td>
<td>4.6</td>
<td>18.2</td>
<td>20.5</td>
<td>46.5</td>
</tr>
<tr>
<td>4KB 100% Random Write</td>
<td>4.08</td>
<td>234765</td>
<td>4.2</td>
<td>18.2</td>
<td>20.0</td>
<td>46.2</td>
</tr>
<tr>
<td>4KB 70/30% Random Read/Write</td>
<td>4.62</td>
<td>209226</td>
<td>4.7</td>
<td>18.0</td>
<td>20.6</td>
<td>46.7</td>
</tr>
</tbody>
</table>

### Linux Kernel NVMe-oF RDMA Initiator QD=1 Latency for a Null block device

<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>Average Latency (usec)</th>
<th>IOPS</th>
<th>p99 (usec)</th>
<th>p99.9 (usec)</th>
<th>p99.99 (usec)</th>
<th>p99.999 (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB 100% Random Read</td>
<td>12.61</td>
<td>77744</td>
<td>17.4</td>
<td>26.0</td>
<td>38.4</td>
<td>143.5</td>
</tr>
<tr>
<td>4KB 100% Random Write</td>
<td>11.31</td>
<td>86388</td>
<td>17.6</td>
<td>26.0</td>
<td>35.9</td>
<td>143.7</td>
</tr>
<tr>
<td>4KB 70/30% Random Read/Write</td>
<td>13.26</td>
<td>74042</td>
<td>20.4</td>
<td>27.2</td>
<td>38.3</td>
<td>131.1</td>
</tr>
</tbody>
</table>
Conclusions

1. The SPDK NVMe-oF initiator reduces the NVMe-oF software overhead by up to 2.9x times vs.
   the Linux Kernel NVMe-oF Initiator for the RDMA transport.
Test Case 4: NVMe-oF RDMA Performance with increasing # of connections

This test case was designed to demonstrate the throughput and latency of the SPDK NVMe-oF RDMA Target vs. Linux Kernel NVMe-oF RDMA Target under increasing number of connections per subsystem. The number of connections (or I/O queue pairs) per NVMe-oF subsystem were varied, we measured the aggregated IOPS and number of CPU cores used by each target. The number of CPU cores metric was calculated from %CPU utilization measured using sar (systat package in Linux). The SPDK NVMe-oF RDMA Target was configured to run on 4 CPU cores, export 16 NVMe-oF subsystems (1 per Intel P4610) and 2 initiators were used both running the I/O workloads below to 8 separate subsystems using Kernel NVMe-oF RDMA initiator.

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case</strong></td>
<td>NVMe-oF RDMA Target performance with increasing # of connections</td>
</tr>
<tr>
<td><strong>SPDK NVMe-oF Target</strong></td>
<td>Same as in Test Case #1, using 4 CPU cores.</td>
</tr>
<tr>
<td><strong>configuration</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Kernel NVMe-oF Target</strong></td>
<td>Target configuration file loaded using nvmet-cli tool.</td>
</tr>
<tr>
<td><strong>configuration</strong></td>
<td>For detailed configuration file contents please see Appendix A.</td>
</tr>
<tr>
<td><strong>Kernel NVMe-oF</strong></td>
<td>Device config</td>
</tr>
<tr>
<td><strong>Initiator #1</strong></td>
<td>Performed using nvme-cli tool.</td>
</tr>
<tr>
<td></td>
<td>modprobe nvme-fabrics</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode1 –t rdma –a 20.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode2 –t rdma –a 20.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode3 –t rdma –a 20.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode4 –t rdma –a 20.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode5 –t rdma –a 20.0.1.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode6 –t rdma –a 20.0.1.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode7 –t rdma –a 20.0.1.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode8 –t rdma –a 20.0.1.1 –s 4420</td>
</tr>
<tr>
<td><strong>Kernel NVMe-oF</strong></td>
<td>Device config</td>
</tr>
<tr>
<td><strong>Initiator #2</strong></td>
<td>Performed using nvme-cli tool.</td>
</tr>
<tr>
<td></td>
<td>modprobe nvme-fabrics</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode9 –t rdma –a 10.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode10 –t rdma –a 10.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode11 –t rdma –a 10.0.0.1 –s 4420</td>
</tr>
<tr>
<td></td>
<td>nvme connect –n nqn.2018-09.io.spdk:cnode12 –t rdma –a 10.0.0.1 –s 4420</td>
</tr>
</tbody>
</table>
The SPDK NVMe-oF Target was configured to use 4 CPU cores, whereas we did not limit the number of CPU cores for the Linux Kernel NVMe-oF target. The graph below shows the relative performance in terms of IOPS/core which was calculated by dividing the total aggregate IOPS by the total CPU cores used while running that specific workload. For the case of Kernel NVMe-oF target, total CPU cores was calculated from % CPU utilization which was measured using sar utility in Linux.
4KB Random Read Results

![Graph showing relative performance comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target for 4KB 100% Random Read QD=64, using Kernel Initiators.](image)

**Figure 10:** Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target for 4KB 100% Random Read QD=64, using Kernel Initiators.

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16862.99</td>
<td>4316.9</td>
<td>237.1</td>
<td>10.6</td>
</tr>
<tr>
<td>4</td>
<td>13146.69</td>
<td>3365.5</td>
<td>303.8</td>
<td>20.4</td>
</tr>
<tr>
<td>16</td>
<td>11845.44</td>
<td>3032.4</td>
<td>336.5</td>
<td>27.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14654.43</td>
<td>3751.5</td>
<td>272.8</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>13155.36</td>
<td>3367.8</td>
<td>303.6</td>
<td>4.0</td>
</tr>
<tr>
<td>16</td>
<td>11562.23</td>
<td>2959.9</td>
<td>344.7</td>
<td>4.0</td>
</tr>
</tbody>
</table>
4KB Random Write results

![Graph showing relative performance comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target for 4KB 100% Random Write QD=64, using Kernel Initiators]

**Figure 11:** Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target for 4KB 100% Random Write QD=64, using Kernel Initiators

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

**Linux Kernel NVMe-oF RDMA Target: 4KB 100% Random Writes, QD=64**

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22342.71</td>
<td>5719.7</td>
<td>178.9</td>
<td>11.4</td>
</tr>
<tr>
<td>4</td>
<td>15766.84</td>
<td>4036.3</td>
<td>253.9</td>
<td>19.9</td>
</tr>
<tr>
<td>16</td>
<td>13428.55</td>
<td>3437.7</td>
<td>296.7</td>
<td>26.7</td>
</tr>
</tbody>
</table>

**SPDK NVMe-oF RDMA Target: 4KB 100% Random Writes, QD=64**

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18097.41</td>
<td>4632.9</td>
<td>221.1</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>16164.64</td>
<td>4138.1</td>
<td>247.1</td>
<td>4.0</td>
</tr>
<tr>
<td>16</td>
<td>13737.85</td>
<td>3516.9</td>
<td>290.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>
4KB Random Read-Write Results

Figure 12: Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF RDMA Target for 4KB Random 70% Read 30% Write QD=64, using Kernel Initiators

Linux Kernel NVMe-oF RDMA Target: 4KB 70% Random Read 30% Random Write, QD=64

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17672.51</td>
<td>4524.2</td>
<td>226.2</td>
<td>10.6</td>
</tr>
<tr>
<td>4</td>
<td>17182.05</td>
<td>4398.6</td>
<td>232.3</td>
<td>21.3</td>
</tr>
<tr>
<td>16</td>
<td>14330.68</td>
<td>3668.7</td>
<td>277.9</td>
<td>30.9</td>
</tr>
</tbody>
</table>

SPDK NVMe-oF RDMA Target: 4KB 70% Random Read 30% Random Write, QD=64

<table>
<thead>
<tr>
<th>Connections per subsystem</th>
<th>Bandwidth (MBps)</th>
<th>IOPS (k)</th>
<th>Avg. Latency (usec)</th>
<th># CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13549.62</td>
<td>3468.7</td>
<td>295.1</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>13631.62</td>
<td>3489.7</td>
<td>292.9</td>
<td>4.0</td>
</tr>
<tr>
<td>16</td>
<td>11942.88</td>
<td>3057.4</td>
<td>333.6</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Conclusions

1. When the SPDK NVMe-oF Target was configured with 4 CPU cores the performance peaked at 1 connection per subsystem for all workloads. Increasing the number of connections per subsystem beyond these values resulted in higher average latency and lower IOPS.

2. The performance for the Linux Kernel NVMe-oF Target peaked at 1 connection per subsystem for all workloads. Increasing the number of connections only increases the latency and CPU utilization.

3. The SPDK NVMe-oF target shows up to 6.8x more IOPS/Core relative to the Linux Kernel NVMe-oF target as the number of connections per subsystem increased.
Summary

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

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

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

It was also observed that the SPDK NVMe-oF Target average latency is up to 4.9 usec lower than Kernel when testing against null bdev based backend. The advantage of SPDK is even greater when comparing NVMe-oF Initiators: the SPDK NVMe-oF RDMA average latency is up to 2.9 times lower than Kernel initiator.

The SPDK NVMe-oF Target performed up to 6.7 times better w.r.t IOPS/core than Linux Kernel NVMe-oF target while running 4KB 100% random read workload with increasing number of connections per NVMe-oF subsystem.

This report provides information regarding methodologies and practices while benchmarking NVMe-oF using SPDK, as well as the Linux Kernel. It should be noted that the performance data showcased in this report is based on specific hardware and software configurations and that performance results may vary depending on different hardware and software configurations.
Example Linux Kernel NVMe-of RDMA Target configuration for Test Case 4.

```json
{
  "ports": [
    {
      "addr": {
        "adrffam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4420",
        "trtype": "rdma"
      },
      "portid": 1,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode1"
      ]
    },
    {
      "addr": {
        "adrffam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4421",
        "trtype": "rdma"
      },
      "portid": 2,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode2"
      ]
    },
    {
      "addr": {
        "adrffam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4422",
        "trtype": "rdma"
      },
      "portid": 3,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode3"
      ]
    },
    {
      "addr": {
        "adrffam": "ipv4",
        "traddr": "20.0.0.1",
        "trsvcid": "4423",
        "trtype": "rdma"
      },
      "portid": 4,
      "referrals": [],
      "subsystems": [
        "nqn.2018-09.io.spdk:cnode4"
      ]
    }
  ]
}
```
"addr": {
    "adr
    "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.1.1",
    "trsvc
    "addr": {
    "adr
    "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.1.1",
    "trsvcid": "4426",
    "trtype": "rdma"
    },
    "portid": 7,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode7"
    ]
    },
    "addr": {
    "adr
    "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.1.1",
    "trsvcid": "4427",
    "trtype": "rdma"
    },
    "portid": 8,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode8"
    ]
    },
    "addr": {
    "adr
    "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.1.1",
    "trsvcid": "4424",
    "trtype": "rdma"
    },
    "portid": 5,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode5"
    ]
    },
    "addr": {
    "adrfam": "ipv4",
    "traddr": "20.0.1.1",
    "trsvcid": "4425",
    "trtype": "rdma"
    },
    "portid": 6,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode6"
    ]
    }
}
"traddr": "10.0.0.1",
"trsvcid": "4428",
"trtype": "rdma"
},
"portid": 9,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode9"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.0.1",
  "trsvcid": "4429",
  "trtype": "rdma"
},
"portid": 10,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode10"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.0.1",
  "trsvcid": "4430",
  "trtype": "rdma"
},
"portid": 11,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode11"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.0.1",
  "trsvcid": "4431",
  "trtype": "rdma"
},
"portid": 12,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode12"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.1.1",
  "trsvcid": "4432",
  "trtype": "rdma"
},
"portid": 13,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode13"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.1.1",
  "trsvcid": "4433",
  "trtype": "rdma"
},
"portid": 14,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode14"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.1.1",
  "trsvcid": "4434",
  "trtype": "rdma"
},
"portid": 15,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode15"
]
},
{
"addr": {
  "adrfam": "ipv4",
  "traddr": "10.0.1.1",
  "trsvcid": "4435",
  "trtype": "rdma"
},
"portid": 16,
"referrals": [],
"subsystems": [
  "nqn.2018-09.io.spdk:cnode16"
]
],
"hosts": [],
"subsystems": [
{
  "allowed_hosts": [],
  "attr": {
    "allow_any_host": "1",
    "version": "1.3"
  },
  "namespaces": [
  {  
    "device": {
      "path": "/dev/nvme0n1",  
    
  
```
"uuid": "b53be81d-6f5c-4768-b3bd-203614d8cf20",
  "enable": 1,
  "nsid": 1
},
"nqn": "nqn.2018-09.io.spdk:cnode1"
},
{
  "allowed_hosts": [],
  "attr": {
    "allow_any_host": "1",
    "version": "1.3"
  },
  "namespaces": [
    {
      "device": {
        "path": "/dev/nvme1n1",
        "uuid": "12f6c584-9c45-4b6a-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": "cea8569-69e9-4831-8e61-90725bdf768d",
        "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,
"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-5cfe23c7a6eb"
        },
        "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-f2d0b5ffe4c"
            },
            "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-94a9-8e75ff8436d"
            },
            "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": "f3d4017b-4f7d-454d-94a9-8e75ff8436d"
            },
            "enable": 1,
            "nsid": 14
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode14"
}
"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"
],
"namespaces": [
{
  "device": {
    "path": "/dev/nvme14n1",
    "uuid": "ed69ba4d-8727-43bd-894a-7b08ade4f1b1"
  },
  "enable": 1,
  "nsid": 15
},
"nqn": "nqn.2018-09.io.spdk:cnode15"
],
"namespaces": [
{
  "device": {
    "path": "/dev/nvme15n1",
    "uuid": "5b8e9af4-0ab4-47fb-968f-b13e4b607f4e"
  },
  "enable": 1,
  "nsid": 16
},
"nqn": "nqn.2018-09.io.spdk:cnode16"
]
Notices & Disclaimers

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 complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

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

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.