Solution Brief

Intel Connectivity
Ethernet Products Group

Performance Testing Application Device Queues (ADQ) with NVMe/TCP Using SPDK

Testing confirms that Application Device Queues (ADQ) technology provides an additional performance boost to the Storage Performance Development Kit (SPDK) library for NVM Express (NVMe) over TCP storage applications.

The appeal of NVMe over Fabrics

The NVM Express (NVMe) standard was developed to accommodate the faster speeds of solid state drives (SSDs), replacing slower standards such as Serial Attached SCSI (SAS) and Serial ATA (SATA) that were designed for spinning disk drives. Initially, NVMe was used to attach SSDs directly to computers and servers using the PCIe bus. However, this kind of direct-attached storage (DAS) can lead to underutilization of siloed storage devices, inefficient data-duplication requirements, and difficulty scaling storage without scaling compute in modern data centers.

The NVMe over Fabrics (NVMe-oF) standard was developed to extend the performance of the NVMe standard over fabrics like Ethernet, InfiniBand, and Fibre Channel. Over Ethernet, the protocols supported are RDMA over Converged Ethernet (RoCE), iWARP, and Transmission Control Protocol (TCP). NVMe-oF enables a pool of NVMe SSDs to be shared by multiple hosts over the network. This kind of storage architecture can be appealing to data center operators who are eager to gain the efficiencies of fast, scalable, shared storage.

This paper focuses on test results showing the performance improvements achieved when using Application Device Queues (ADQ) with NVMe over TCP and the Storage Performance Development Kit (SPDK).

Decisions and tradeoffs in implementing NVMe-oF

NVMe-oF requires the underlying transport to provide reliable NVMe command and data delivery. Intel® Ethernet 800 Series Network Adapters support all three Ethernet-based transport protocol options for NVMe-oF implementation, as shown in Figure 1.

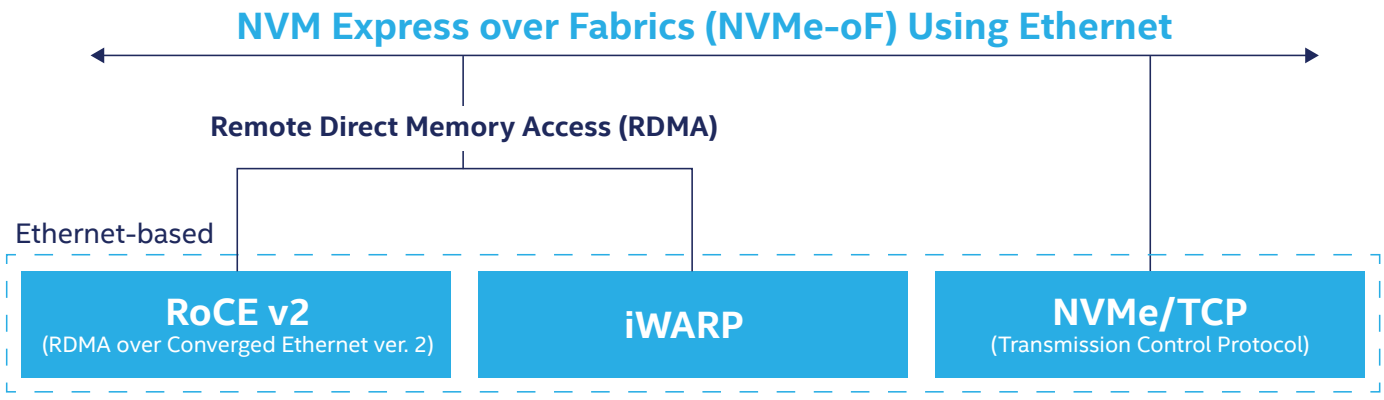

Figure 1. Three transport-protocol options for NVMe-oF using Ethernet

RoCE v2

Based on RDMA, RoCE v2 was the first transport protocol defined in the NVMe-oF specification, and it is therefore a widely adopted transport protocol. RoCE v2 is also the most challenging protocol to implement, as it requires lossless switches and additional network configuration that can come with calculable costs and additional operational complexity. It is also difficult to scale beyond the rack level.[1]

iWARP

Also based on RDMA, this protocol is newer and a bit easier to implement than RoCE v2 because it sits on top of TCP/IP and uses standard switches and network configurations. However, iWARP still requires RDMA-enabled network interface controllers (NICs).

TCP

Unlike the protocols based on RDMA, NVMe over TCP uses the standard Linux TCP stack that runs on the host CPU. This newer transport-protocol option enables efficient end-to-end NVMe operations between hosts and targets interconnected by any standard IP network. It allows large-scale data centers to utilize their existing ubiquitous Ethernet infrastructure with multi-layered switch topologies and traditional Ethernet network adapters.[2] However, it tends to deliver slower performance due to the software overhead.

Up until now, technical decision makers (TDMs) have been faced with a tradeoff between the ease of implementation and scalability of the TCP option versus the higher performance of the RDMA-based options, as summarized in Table 1.

Table 1. Tradeoffs in choosing an NVMe-oF transport protocol[3]

	NVMe/TCP	NVMe/RDMA iWARP	NVMe/RDMA RoCE v2
Network Infrastructure	Standard NICs	RDMA-enabled NICs	RDMA-enabled NICs
	Standard Ethernet switches	Standard Ethernet switches	Lossless Ethernet switches
Performance	Baseline input/output operations per second (IOPS) and CPU efficiency	High IOPS and CPU efficiency	High IOPS and CPU efficiency
	Highest tail latency	Lowest tail latency	Lowest tail latency
Operating System (OS) Network Software	Standard Linux TCP stack	RDMA-enabled stack	RDMA-enabled stack
Ease of Use	Standard network configuration	Standard network configuration	Requires additional network configuration
NVMe-oF Usage Model	Data-center wide	Rack-level, data-center wide	Rack-level, within lossless Ethernet domain

Clearly the TCP option would be most appealing if the performance gap could be closed, due to its standard network infrastructure and network configuration, in addition to its data-center-wide usage model. Two technologies show a lot of promise for closing that gap:

- SPDK provides open source components for implementing NVMe over TCP in userspace that use the Linux kernel's TCP stack. Intel is a key proponent of SPDK.
- ADQ is an open technology for system-level network input/output (I/O) performance that improves application response-time predictability. ADQ is now available in the Linux kernel (version 4.19 or later), and it is supported in Intel Ethernet 800 Series Network Adapters.

The testing presented in this paper measures the performance achieved by using these technologies together—which only became feasible when enhancements to support ADQ in SPDK were included in the July 2020 release (20.07) of SPDK.

Both ADQ and SPDK improve storage performance

ADQ is like Ethernet express lanes for your important application data. ADQ dedicates queues and shapes traffic for the transfer of data over Ethernet for critical applications. The goal of ADQ is to ensure that high-priority applications receive predictable high performance through dramatically reduced tail latency. Tail latency—the result of slow outliers in an otherwise fast system—becomes an increasingly significant factor as a data center scales with more servers and more storage.

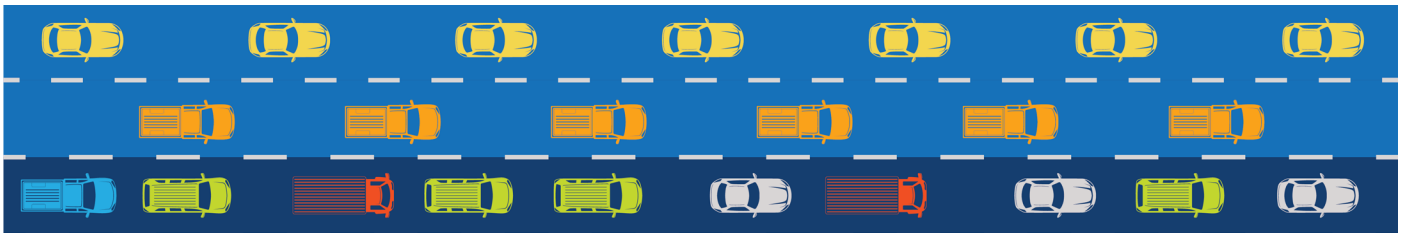

Figure 2. ADQ is like dedicated Ethernet express lanes for your application data storage

ADQ has been shown to significantly improve both tail latency and throughput when used with applications such as open source Redis,[4] Memcached,[5] and Aerospike.[6]

The SPDK components can reduce the software overhead incurred when accessing remote network-attached SSDs via NVMe over TCP.

Next, this paper takes a closer look at the question that has not been tested until now, "How much will ADQ improve the performance of an NVMe over TCP system that is using SPDK?"

ADQ and SPDK together: testing SPDK with and without ADQ enabled

Intel tested NVMe over TCP storage systems configured with SPDK for predictability, latency, and throughput for a baseline configuration without ADQ, then tested the same systems with ADQ enabled.

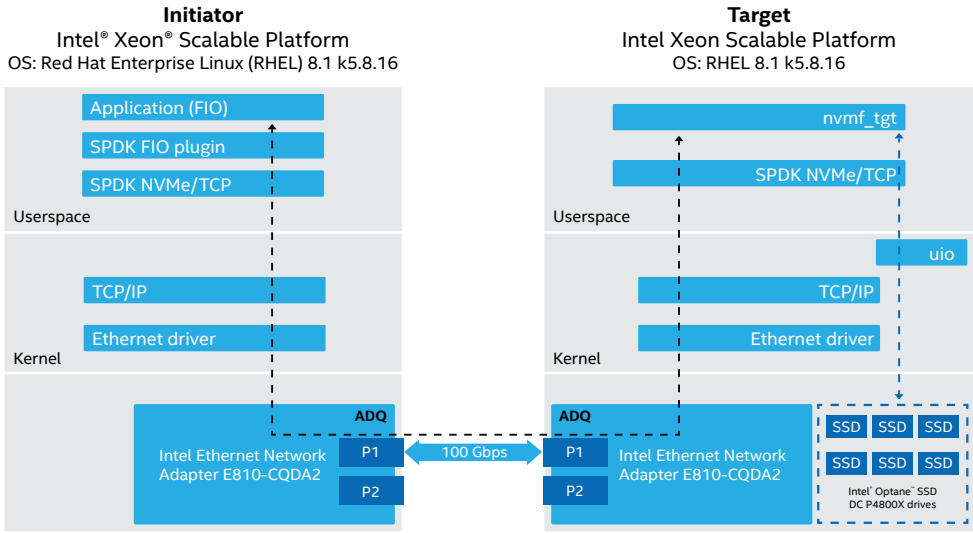

Figure 3. SPDK NVMe over TCP with ADQ test topology

The test setup consisted of two 2nd Generation Intel® Xeon® Scalable processor–based platforms configured as initiator and target. The two platforms were connected back-to-back with a 100 gigabit per second (Gbps) Ethernet link using an Intel Ethernet 800 Series Network Adapter in each server. ADQ is enabled or disabled in each adapter. I/O read and write commands are generated by the fio in the initiator. Then they are sent to the target to access a set of six Intel® Optane™ SSD DC P4800X drives in the target. Test configuration details are shown in the Appendix.

Predictability

Testing measured the predictability of the storage system, as defined by the lowest latency that could be achieved 99.99 percent of the time, a standard known as P99.99. The improvement in predictability with ADQ enabled ranged up to 33 percent with different numbers of cores in use, as shown in Figure 4.

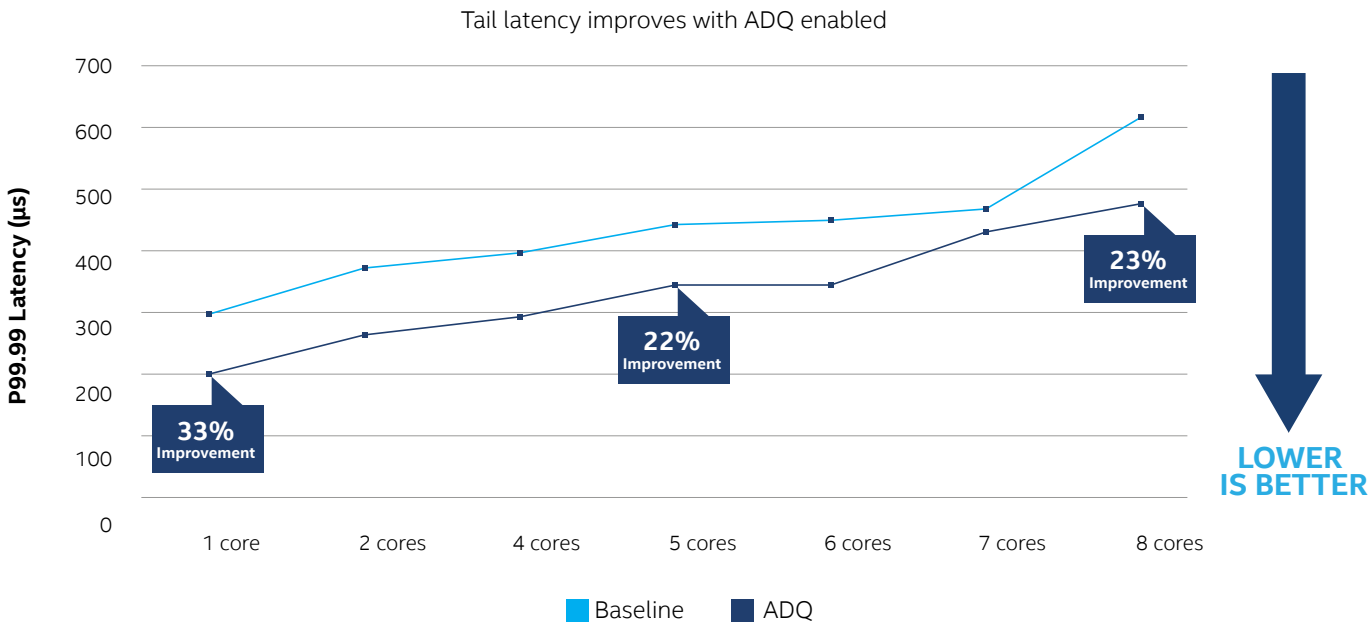

Figure 4. Tail latency (P99.99) improves with ADQ enabled (4 KB random read, 1,500 maximum transmission units [MTU], queue depth [QD]=32, three connections per target core)

Note that these test results include measurements on systems using up to eight target cores. However, as will be discussed in the Throughput section later, the 100 Gbps link was saturated with just five cores with ADQ enabled. Therefore, the ADQ results with more than five cores are indicative of the effect of an I/O workload exceeding the available network bandwidth. Typically, one would not push the system beyond the link saturation, as that would negatively impact the latency.

Latency

Average latency in the testing was reduced when ADQ was enabled. Latency was reduced by 30 percent at five cores, which is where throughput with ADQ enabled saturates a 100 Gb link

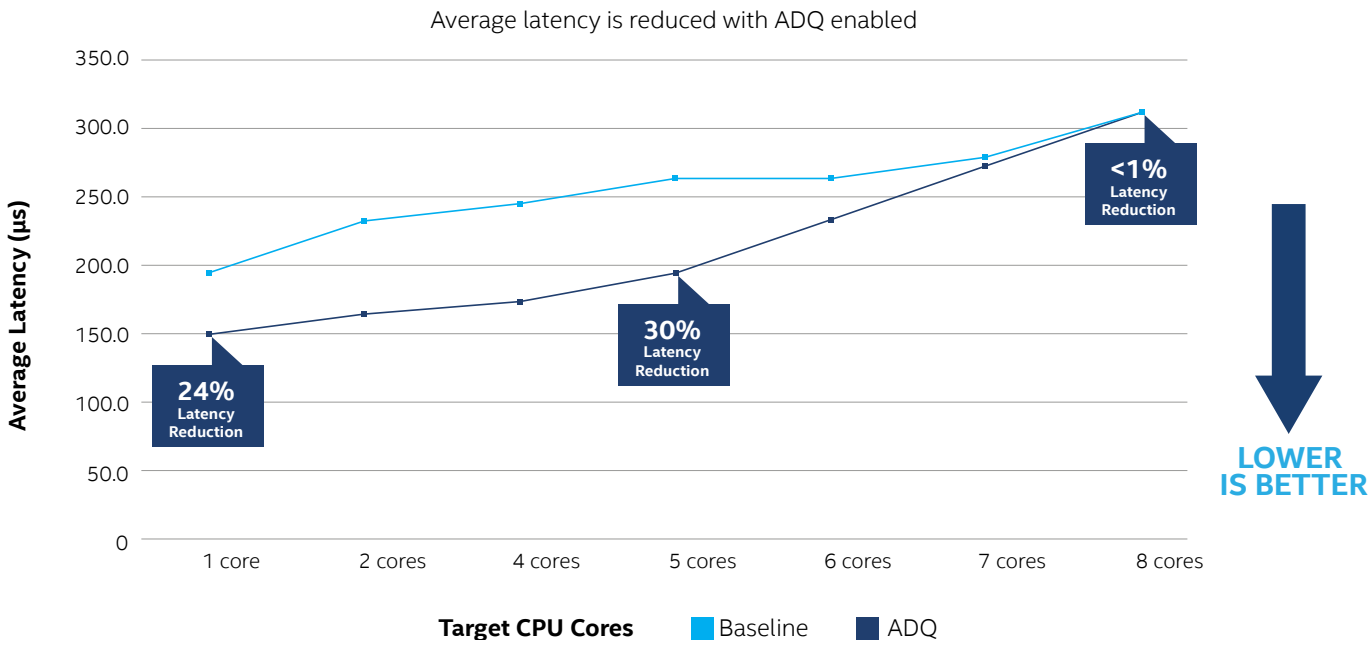

Figure 5. Average latency is reduced with ADQ enabled (4 KB random read, 1,500 MTU, QD=32, three connections per target core)

Throughput

Throughput in the testing was increased across all core counts when ADQ was enabled. Throughput increased by 36 percent at five SPDK target cores, which is where ADQ enabled it to reach the saturation point for a 100 Gb link at around 2.8 million IOPS. While the benefits of ADQ above five SPDK target cores are diminished by the saturation point of the link, the real advantage in practice is that ADQ reduces the number of SPDK target cores required to saturate the link. In the Baseline case without ADQ, it takes eight SPDK target cores to saturate the link. The three cores that would otherwise be required to saturate the link when ADQ is not being used become available for other compute purposes.

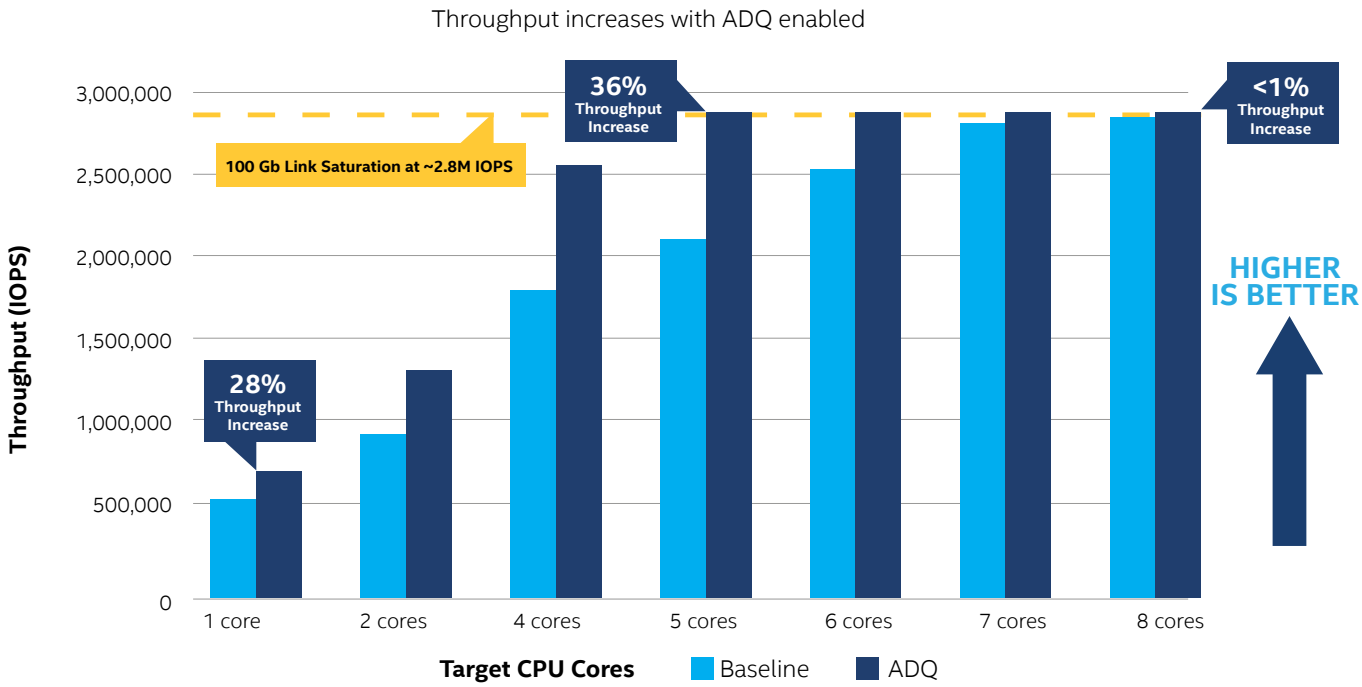

Figure 6. Throughput increases with ADQ enabled (4 KB random read, 1,500 MTU, QD=32, three connections per target core)

Analysis

Using SPDK with ADQ enabled, the predictability, latency, and throughput performance of NVMe over TCP improves significantly, providing an additional boost above the already improved NVMe over TCP with SPDK performance. This changes the game favorably for those who must decide which route to take to implement NVMe-oF. The ease of implementation and the scalability of the TCP route can be weighed against a less significant performance gap.

Table 2. With ADQ, NVMe over TCP becomes a more attractive option[3]

	NVMe/TCP	NVMe/RDMA iWARP	NVMe/RDMA RoCE v2	NVMe/TCP with ADQ
Network Infrastructure	Standard NICs	RDMA-enabled NICs	RDMA-enabled NICs	Standard NICs
	Standard Ethernet switches	Standard Ethernet switches	Lossless Ethernet switches	Standard Ethernet switches
Performance	Baseline (IOPS) and CPU efficiency	High IOPS and CPU efficiency	High IOPS and CPU efficiency	High IOPS and CPU efficiency
	Highest tail latency	Lowest tail latency	Lowest tail latency	Low tail latency

OS Network Software	Standard Linux TCP stack	RDMA-enabled stack	RDMA-enabled stack	Standard Linux TCP ADQ-enabled stack
Ease of Use	Standard network configuration	Standard network configuration	Requires additional network configuration	Standard network configuration
NVMe-oF Usage Model	Data-center wide	Rack-level, data-center wide	Rack-level, within lossless Ethernet domain	Data-center wide

ADQ also provides a benefit to the approach of using NVMe/TCP in the area of CPU utilization. Unlike RDMA, the TCP transport stack is part of the Linux kernel consuming CPU core cycles. However, ADQ lessens the impact of CPU core cycles consumed by decreasing the number of cores needed. The TCP storage transport option with ADQ can saturate a 100 Gb link using fewer cores (five with ADQ, versus eight without ADQ, in this testing), freeing the additional cores that would be necessary to saturate the link without ADQ for other compute purposes.

Conclusion

Every organization must choose the NVMe-oF transport method that best suits their requirements. For situations where high performance is critical and expert resources are available to achieve that goal, one of the RDMA options might be the best choice. But for companies who just want to move to NVMe-oF as painlessly and cost effectively as possible without taking a major performance hit, NVMe/TCP and SPDK enhanced by ADQ represents an attractive route to easy deployment with high performance and future scalability.[7]

Similar to the performance improvements seen by enabling ADQ for Memcached,[5] open source Redis,[4] and Aerospike applications,[6] NVMe/TCP using SPDK receives a significant performance boost when the Intel Ethernet 800 Series with ADQ is added.

Increases application **predictability**

Up to 22%

At link saturation using ADQ with 5 target cores[7]

Reduces application **latency**

Up to 30%

At link saturation using ADQ with 5 target cores[7]

Improves application **throughput**

Up to 36%

At link saturation using ADQ with 5 target cores[7]

Learn more

Contact your Intel sales representative or distributor for more details about Intel Ethernet 800 Series with ADQ, and visit the ADQ Resource Center at **intel.com/adq**.

Appendix: Test configuration

Table 3. Test system configuration

	Target	Initiator
Test by	Intel	Intel
Test date	11/24/2020	11/24/2020
Platform	Dell EMC PowerEdge R740xd	Dell EMC PowerEdge R740xd
Number of nodes	1	1
Number of sockets	2	2
CPU	Intel Xeon Platinum 8280 processor at 2.70 GHz	Intel Xeon Platinum 8280 processor at 2.70 GHz
Cores/socket, threads/socket	28 cores per socket, 56 threads per socket	28 cores per socket, 56 threads per socket
Microcode	0x500001c	0x500001c
Intel Hyper-Threading Technology (Intel HT Technology)	Disabled*	Disabled
Intel Turbo Boost Technology	Enabled	Enabled
BIOS version	2.1.8	2.1.8
System DDR memory configuration: slots/cap/run-speed	12 slots, 16 GB, 2,933 megatransfers per second (MT/s)	16 slots, 16 GB, 2,933 MT/s
Total memory/node (DDR+PMem)	192 GB DDR4-2,933 DIMM	192 GB DDR4-2,933 DIMM
Storage (boot)	100 GB SATA SSD	100 GB SATA SSD
Storage (application drives)	6 x Intel Optane SSD DC P4800X, PCIe 3.0, x4	Not applicable (N/A)
Network interface card (NIC)	Intel Ethernet Network Adapter E810-CQDA2	Intel Ethernet Network Adapter E810-CQDA2
PCH	Intel C620 series chipset	Intel C620 series chipset
OS	Red Hat Enterprise Linux (RHEL) 8.1 (Ootpa)	RHEL 8.1 (Ootpa)
Kernel	5.8.16	5.8.16
Workload	Fio 3.15	Fio 3.15
Compiler	GCC 8.3.1 20190507 (Red Hat 8.3.1-4)	GCC 8.3.1 20190507 (Red Hat 8.3.1-4)
NIC driver	ice-1.2.1; firmware: 0x80004fb3	ice-1.2.1; firmware: 0x80004fb3

***Intel HT Technology was turned off for benchmark purposes to not schedule fio on threads of the same physical core.**

Table 4. System under test (SUT) network-adapter configuration settings

	ADQ Off Baseline	ADQ On
System Settings		
Interrupt Moderation	Fixed, rx/tx=50us	Fixed, rx=0,tx=500us
IRQ Balance	No	No
Interrupt Affinitization	Yes	Yes
ADQ Settings		
Epoll Busy Poll	No	Yes
Socket Option for NAPI ID	No	Yes
TC-Mqprio Hardware Offload and Shaper	No	Yes
TC- Cloud Filter Enabling with TC-flower	No	Yes

Table 5. SUT and client OS and adapter configuration

	Red Hat Enterprise Linux (RHEL) 8.1 Settings Target and Client	**Adapter Settings** Linux Driver: ice-1.2.1 NVM: 0x80004fb3
ADQ off (baseline)	Stopped and disabled: firewalld, NetworkManager SELINUX disabled x86_energy_perf_policy performance tuned-adm profile latency-performance systemctl stop irqbalance sysctl -w net.core.somaxconn=4096 sysctl -w net.core.netdev_max_backlog=8192 sysctl -w net.ipv4.tcp_max_syn_backlog=16384 sysctl -w net.ipv4.route.flush=1 sysctl -w net.core.busy_poll=0 **Target only:** sysctl -w net.ipv4.tcp_mem="268435456 268435456 268435456" sysctl -w net.core.rmem_max=268435456 sysctl -w net.core.wmem_max=268435456 sysctl -w net.ipv4.tcp_rmem="8192 1048576 33554432" sysctl -w net.ipv4.tcp_wmem="8192 1048576 33554432" **Host only:** sysctl -w net.ipv4.tcp_mem="764688 1019584 16777216" sysctl -w net.core.rmem_max=16777216 sysctl -w net.core.wmem_max=16777216 sysctl -w net.ipv4.tcp_rmem="8192 87380 16777216" sysctl -w net.ipv4.tcp_wmem="8192 65536 16777216“	Adapter MTU:1500 <path-to-ice>/set_irq_affinity -X local $iface ethtool --coalesce $iface adaptive-rx off rx-usecs 50 ethtool --coalesce $iface adaptive-tx off tx-usecs 50

	Red Hat Enterprise Linux (RHEL) 8.1 Settings Target and Client	**Adapter Settings** Linux Driver: ice-1.2.1 NVM: 0x80004fb3
ADQ on	**Same as ADQ off baseline, plus:** sysctl -w net.core.busy_read=1	TC0=2 cores TC1=<# ADQ queues> # TC1 = # SPDK cores tc qdisc add dev $iface root mqprio num_tc 2 map 0 1 queues 2@0\ <# ADQ cores>@2 hw 1 mode channel tc qdisc add dev $iface ingress tc filter add dev $iface protocol ip parent ffff: prio 1 flower dst_ip $addr/32 \ ip_proto tcp dst_port $((SPDK app port)) skip_sw hw_tc 1 ethtool --set-priv-flags $iface channel-pkt-inspect-optimize off ethtool --set-priv-flags $iface channel-inline-flow-director on ethtool --offload $iface hw-tc-offload on ethtool --coalesce $iface adaptive-rx off rx-usecs 0 ethtool --coalesce $iface adaptive-tx off tx-usecs 500 <path-to-ice>/set_xps_rxqs $iface

Table 6. SPDK configuration

SPDK Configuration Notes	SPDK Target Configuration
SPDK Version: 20.10 (pre-release) git sha1 99d3695c2 with zerocopy enabled **SPDK Configuration commands:** HUGEMEM=32768 <path-to-spdk>/scripts/setup.sh	Start SPDK Target Where m[1,3,...n*2-1] is the number of SPDK target CPU cores specified, located on the local NUMA node to the NVMe drives: nohup /opt/spdk/build/bin/nvmf_tgt --wait-for-rpc -m[1,3,5,7,9,11,13,15,17,19] 2>&1 > nvmf_tgt.log & All the commands below were executed with the spdk/scripts/rpc.py script: nvmf_set_config -r 10000 -s transport bdev_nvme_set_options -n 4 -t 0 -a none -p 100000 sock_impl_set_options --enable-placement_id -i posix --enable-zerocopy-send /opt/spdk/scripts/rpc.py framework_start_init nvmf_create_transport -t tcp -q 128 -m 64 -c 4096 -i 131072 -u 131072 -a 128 -n 4096 -b 32 -y 1 bdev_nvme_attach_controller -b Nvme0 -t pcie -a 0000:b3:00.0 bdev_nvme_attach_controller -b Nvme1 -t pcie -a 0000:90:00.0 bdev_nvme_attach_controller -b Nvme2 -t pcie -a 0000:8a:00.0 bdev_nvme_attach_controller -b Nvme3 -t pcie -a 0000:b4:00.0 bdev_nvme_attach_controller -b Nvme4 -t pcie -a 0000:8b:00.0 bdev_nvme_attach_controller -b Nvme5 -t pcie -a 0000:b1:00.0 nvmf_create_subsystem nqn.2021-01.io.spdk:NVMe0 -s SPDK00000000000000 -a nvmf_subsystem_allow_any_host -e nqn.2021-01.io.spdk:NVMe0 nvmf_subsystem_add_listener -t tcp -a 13.100.1.11 -s 7777 nqn.2021-01.io.spdk:NVMe0 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme0n1 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme1n1 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme2n1 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme3n1 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme4n1 nvmf_subsystem_add_ns nqn.2021-01.io.spdk:NVMe0 Nvme5n1

Table 7. Fio configuration

Fio Configuration Notes	Host Benchmark Read Configuration
rw 4K message size random read iodepth 4K message size reads/writes queue depth 32 cpus_allowed CPU cores used for read = (#Target SPDK cores * 3) *CPU cores are on local NUMA node, on same CPU socket as Intel® 800 Series Ethernet Adapter	Fio Version 3.15 - Message sizes: 4K Duration: 90 secs – Queue Depth: 32 Average of 3 iterations Example fio configuration file: [global] ioengine=/spdk/build/fio/spdk_bdev spdk_conf=/hst/spdk_bdev.conf rw=<randread,read> rwmixread=100 numjobs=<#Target SPDK cores * 3> ramp_time=10s runtime=90s bs=4K iodepth=32 iodepth_batch=8 iodepth_batch_complete_max=32 iodepth_batch_complete_min=1 cpus_allowed_policy=split direct=1 time_based=1 thread=1 norandommap=1 percentile_list=99:99.9:99.99 group_reporting=1 cpus_allowed=1,3,5,7,...,29,31,33,35 [job1] stonewall filename=Nvme0n1 filename=Nvme0n2 filename=Nvme0n3 filename=Nvme0n4 filename=Nvme0n5 filename=Nvme0n6

[1] Don Stanwyck. "The Evolution of RDMA in Windows: now extended to Hyper-V Guests." TechNet. November 2017. https://techcommunity.microsoft.com/t5/networking-blog/the-evolution-of-rdma-in-windows-now-extended-to-hyper-v-guests/ba-p/339699.

[2] Sagi Grimberg and Dave Minturn. "Welcome NVMe/TCP to the NVMe-oF Family of Transports." NVM Express, Inc. November 2018. https://nvmexpress.org/welcome-nvme-tcp-to-the-nvme-of-family-of-transports/.

[3] Intel. "SDC2020: Tuning and Optimizing Ethernet-based NVMe over Fabric transport Protocols." October 2020. youtube.com/watch?v=4lD15s9gPtU&list=PLH_ag5Km-YUYEHj-8YEhmlA6z7Bml_3Kq&index=41.

[4] Based on Intel testing as of February 2019. For configuration disclosure details, see: Intel. "Performance Testing Applications Device Queues (ADQ) with Redis." March 2019. intel.com/content/www/us/en/architecture-and-technology/ethernet/application-device-queues-with-redis-brief.html.

[5] Based on Intel testing as of February 2020. For configuration disclosure details, see: Intel. "Performance Testing Application Device Queues (ADQ) with Memcached." April 2020. intel.com/content/www/us/en/architecture-and-technology/ethernet/performance-testing-application-device-queues-with-memcached.html.

[6] Based on Aerospike testing as of September 2019. For configuration disclosure details, see: Intel. "Performance Testing Application Device Queues (ADQ) with Aerospike." April 2020. intel.com/content/www/us/en/architecture-and-technology/ethernet/performance-testing-application-device-queues-with-aerospike.html.

[7] At link saturation with ADQ "on" using five target cores vs. the ADQ "off" Baseline. See Figures 4, 5, and 6, in addition to the appendix, for configuration info.

Printed in USA 0321/TK/PRW/PDF Please Recycle 344871-001US