HITACHI
Inspire the Next

SPDK NVMe-oF Performance Report

By Taisuke Fukuyama, Shuhei Matsumoto, October 2018

Contents

Purpose

We evaluated SPDK performance for both local NVMe poll-mode driver and NVMe-oF Initiator and Target. We focused on IOPS and latency in this report. This work was also intended to follow Intel's published results [1] and give our team hand-on experience with the SPDK NVMe-oF Initiator and Target.

Test Configuration

System Configuration

Table 1: NVMe-oF Initiator Server Configuration

Item	Description
Server model	Hitachi HA8000 RS210BM (http://www.hitachi.co.jp/Prod/comp/OSD/pc/ha/index.html) (This model is for Japan only)
Processor	2x Intel® Xeon® Processor E5-2420 (Sandy Bridge) (6core / 12thread / 1.9GHz / 15MB cache)
Memory	64GB (DDR3-1600 16GB x 4DIMMs)

Item	Description
Network	1 Mellanox® ConnectX-4 100Gb/s MT27700 (dual port)
OS	Red Hat® Enterprise Linux® 7.3
Kernel	Linux Kernel 4.15.1[1]
SPDK	18.04
DPDK	17.11
FIO	3.3

Table 2: NVMe-oF Target Server Configuration

Item	Description
Server Model	Supermicro® 6028R-TDWNR (https://www.supermicro.com/products/system/2u/6028/sys-6028r-tdwnr.cfm)
Processor	2x Intel® Xeon® Processor E5-2620v4 (Broadwell) (8core / 16thread / 2.1GHz / 20MB cache)
Memory	64GB (DDR4-2400 8GB x 8DIMMs)
NVMe	4x Intel® SSD DC P3700 Series 400GB (2.5inch MLC PCIe NVMe 3.0)
Network	1 Mellanox® ConnectX-4 100Gb/s MT27700 (dual port)
OS	Red Hat® Enterprise Linux® 7.3
Kernel	Linux Kernel 4.15.11[1]
SPDK	18.04
DPDK	17.11
FIO	3.3

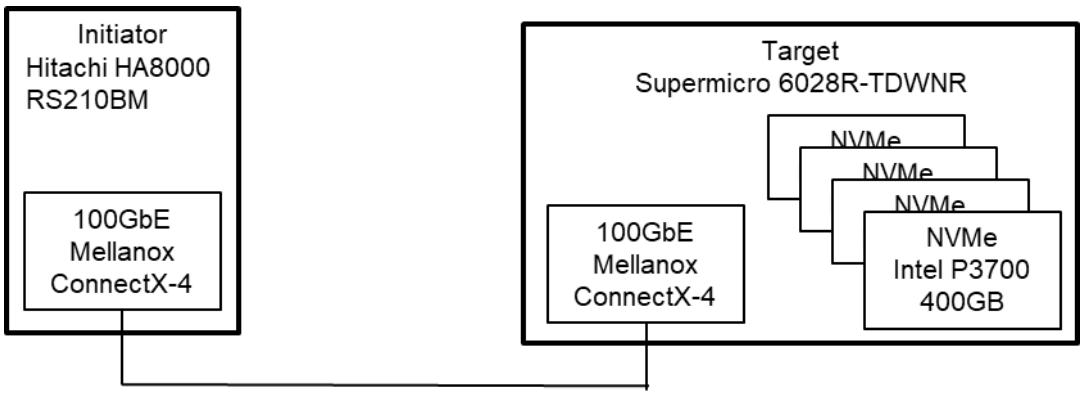

Figure 1: NVMe-oF Server Configuration

Table 3: BIOS and OS Settings related with performance

Item	Setting
Turbo mode	Enabled.
Hyper Threading	Disabled
C-States Control	Disabled
P-States Control	Disabled
Power Management	Max Performance
Linux IO Scheduler	noop, nomerge
CPU Scaling Governor	Performance

[1] Our performance data was collected without Spectre and Meltdown patches which would have further differentiated SPDK.

SPDK NVMe-oF Configuration Consideration

The SPDK NVMe-oF Target is an application running as a process. We evaluated the performance of three SPDK NVMe-oF Target configurations. We used SPDK NVMe-oF Initiator for the evaluation.

Configuration 1 - Single NVMe-oF Subsystem in the NVMe-oF Target Application

We started an NVMe-oF Target application, allocated all CPU cores to the NVMe-oF Target application, created an NVMe-oF subsystem in the NVMe-oF Target application, and added all NVMe namespaces to the NVMe-oF subsystem. We also allocated a queue pair to each NVMe namespace and pinned each queue pair to a CPU core.

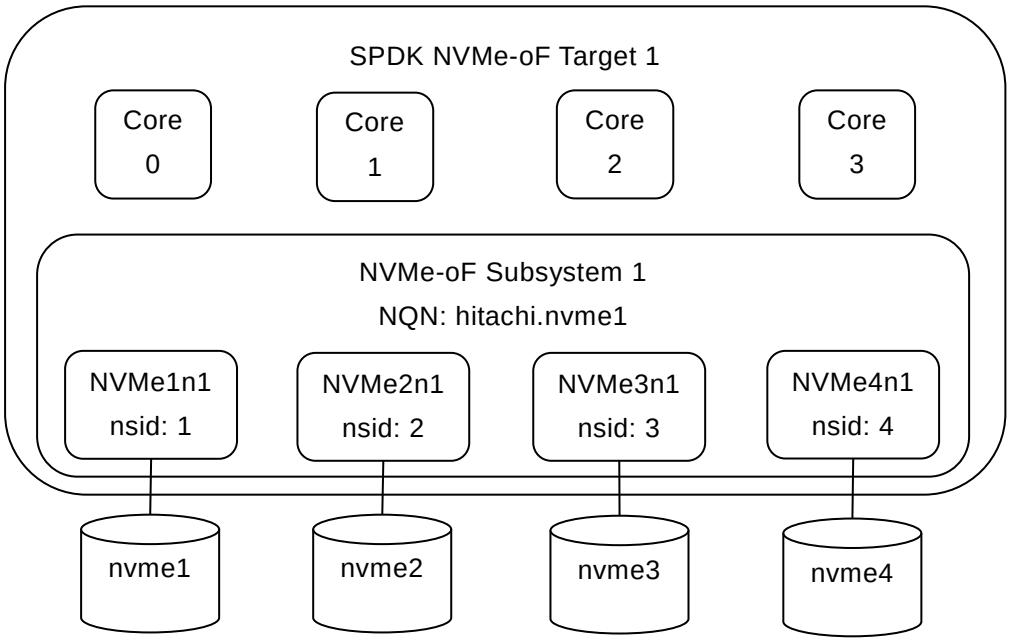

Figure 2: Single NVMe-oF Subsystem in the NVMe-oF Target Application

Configuration 2 - Four NVMe-oF Subsystems in the NVMe-oF Target Application

We started an NVMe-oF Target application, allocated all CPU cores to the NVMe-oF Target application, created four NVMe-oF subsystems in the NVMe-oF Target application, and added an NVMe namespace to each NVMe-oF subsystem. We also allocated a queue pair to each NVMe namespace and pinned each queue pair to a CPU core.

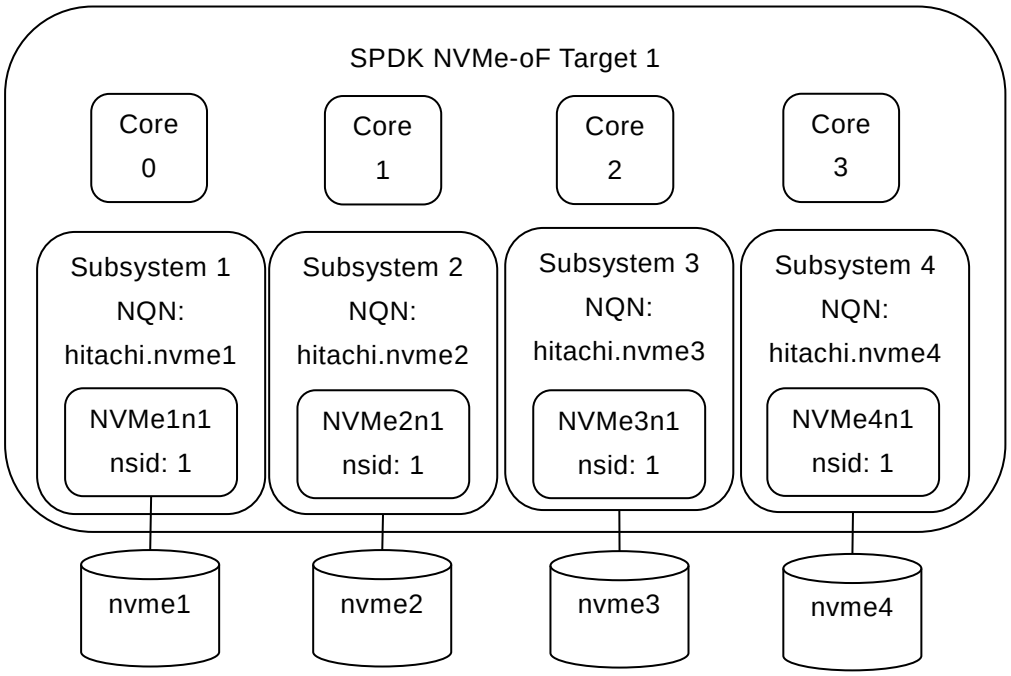

Figure 3: Four NVMe-oF Subsystems in the NVMe-oF Target Application

Configuration 3 - Four NVMe-oF Subsystems in four NVMe-oF Target Applications

We started four NVMe-oF Target applications, allocated a CPU core to each NVMe-oF Target application, created an NVMe-oF subsystem in each NVMe-oF Target application, and added a namespace to each NVMe-oF subsystem. We also allocated a queue pair to each namespace and pinned each queue pair to a CPU core.

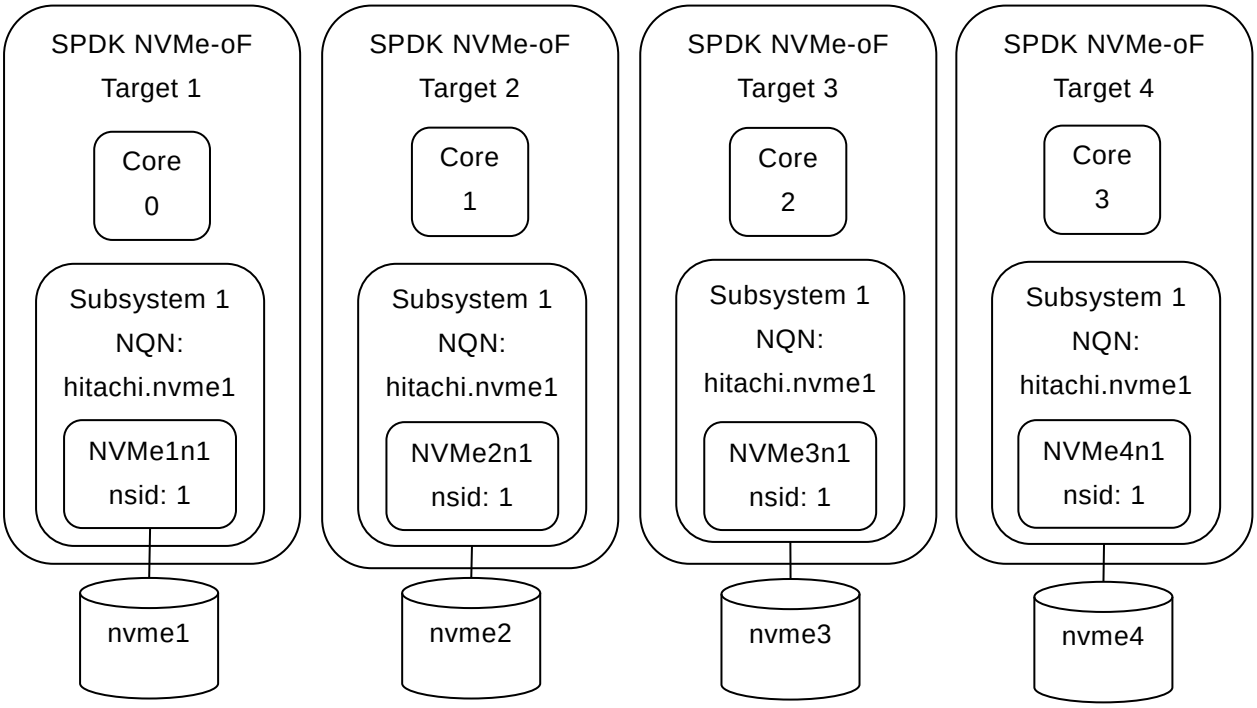

Figure 4: Four NVMe-oF Subsystems in four NVMe-oF Target Applications

Table 4: 4KB Random Read IOPS for the Three NVMe-oF Target Configurations (FIO + SPDK NVMe-oF Initiator)

Configuration	Number of Cores	FIO Cores	IOPS (K)
1	4	1	737
		2	1455
		4	1422
		8	1412
2		1	737
		2	1454
		4	1436
		8	1422
3		1	733
		2	976
		4	1012
		8	1410

Table 5: 4KB Random Write IOPS for the Three NVMe-oF Target Configurations (FIO + SPDK NVMe-oF Initiator)

Configuration	Number of Cores	FIO Cores	IOPS (K)
1	1	1	356
		2	357
		4	364

Configuration	Number of Cores	FIO Cores	IOPS (K)
	4	1	367
		2	345
		4	345
2		1	336
		2	332
		4	331
3		1	350
		2	357
		4	352

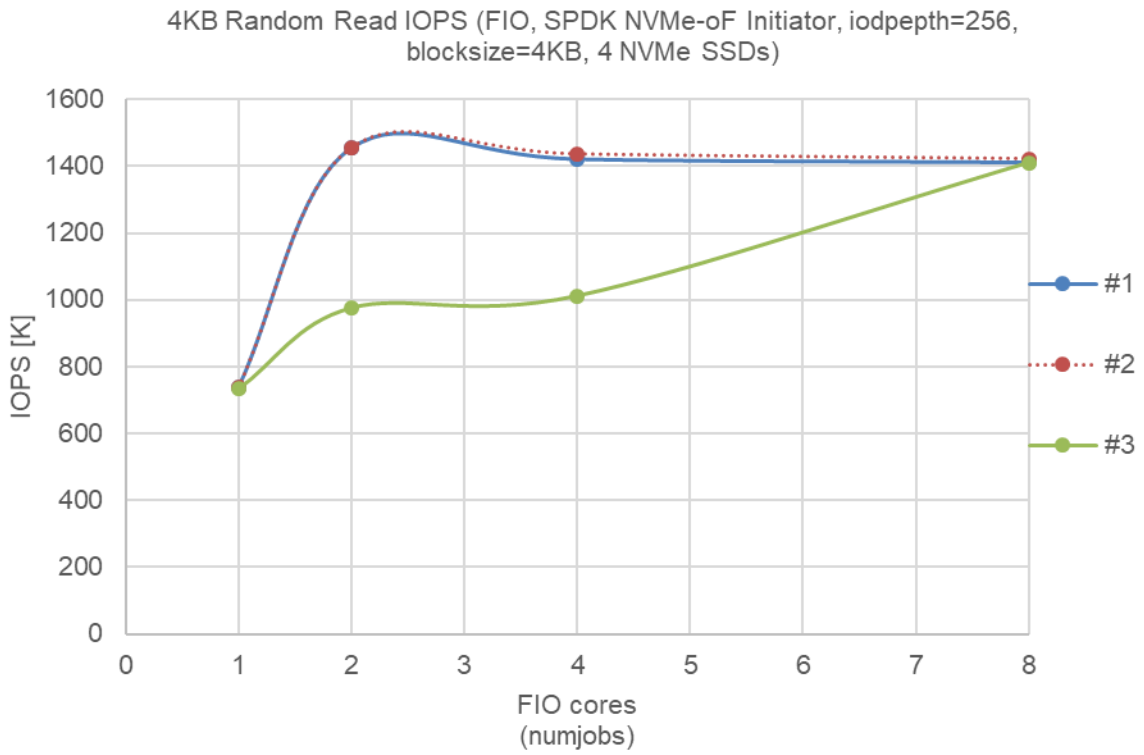

Figure 5: 4KB Random Read IOPS for the Three NVMe-oF Target Configurations (FIO, SPDK NVMe-oF Initiator, iodepth=256, blocksize=4KB, 4 NVMe SSDs)

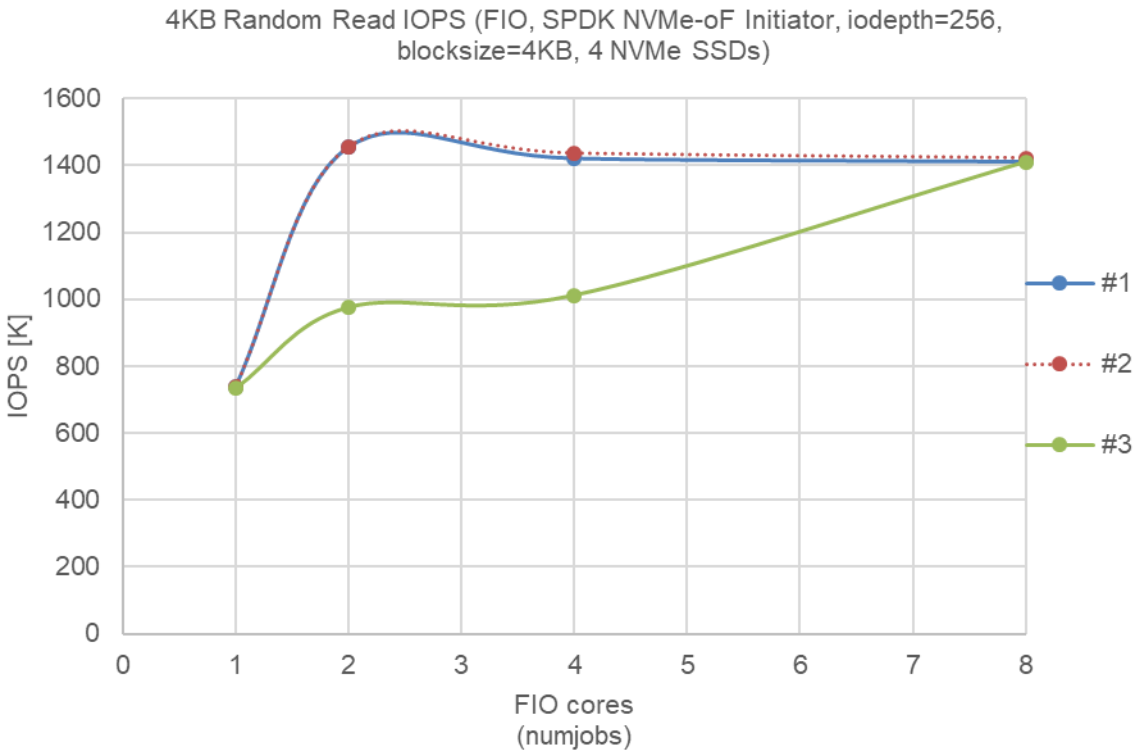

Figure 6: 4KB Random Write IOPS for the Three NVMe-oF Target Configurations (FIO, SPDK NVMe-oF Initiator, iodepth=256, blocksize=4KB, 4 NVMe SSDs)

As a summary of the results, for random reads, performance of configuration 1 and 2 were better than that of configuration 3, and for random writes, performance of configuration 1 was slightly better than that of configuration 2 and 3. Hence, we used configuration 1 in our report.

Kernel NVMe-oF Settings

Table 6: Kernel NVMe-oF Settings

Item	Description
Kernel NVMe-oF Target	modprobe nvmet modprobe nvmet-rdma modprobe configfs mkdir /sys/kernel/config/nvmet/subsystems/nqn.2016-06.io.kernel:dev16 cd /sys/kernel/config/nvmet/subsystems/nqn.2016-06.io.kernel:dev16 echo 1 > attr_allow_any_host mkdir namespaces/10 cd namespaces/10 echo -n /dev/nvme0n1 > device_path echo 1 > enable cd ../.. mkdir namespaces/20 cd namespaces/20 echo -n /dev/nvme1n1 > device_path echo 1 > enable cd ../.. mkdir namespaces/30

Item	Description
	cd namespaces/30 echo -n /dev/nvme2n1 > device_path echo 1 > enable cd ../.. mkdir namespaces/40 cd namespaces/40 echo -n /dev/nvme3n1 > device_path echo 1 > enable mkdir /sys/kernel/config/nvmet/ports/1 cd /sys/kernel/config/nvmet/ports/1 echo 192.168.10.32 > addr_traddr echo rdma > addr_trtype echo 4421 > addr_trsvcid echo ipv4 > addr_adrfam ln -s /sys/kernel/config/nvmet/subsystems/nqn.2016-06.io.kernel:dev16 subsystems/nqn.2016-06.io.kernel:dev16
Kernel NVMe-oF Initiator	modprobe -r mlx5_ib modprobe mlx5_ib dyndbg=+pmf modprobe nvme-rdma dyndbg=+pmf modprobe nvme dyndbg=+pmf modprobe nvme-fabrics dyndbg=+pmf nvme discover -t rdma -a 192.168.10.32 -s 4421 nvme connect -t rdma -n nqn.2016-06.io.kernel:dev16 -a 192.168.10.32 -s 4421

SPDK NVMe-oF Settings

Table 7: SPDK NVMe-oF Settings

Item	Description
SPDK NVMe-oF Target	modprobe ib_cm modprobe ib_core modprobe ib_ucm modprobe ib_umad modprobe ib_uverbs modprobe iw_cm modprobe rdma_cm modprobe rdma_ucm HUGEMEM=20480 /root/spdk-master/scripts/setup.sh /root/spdk-master/app/nvmf_tgt/nvmf_tgt \ -c /root/spdk-master/etc/spdk/nvmf.conf -s 10240 # cat /root/spdk-master/etc/spdk/nvmf.conf [Global] ReactorMask 0xFFFF [Nvmf] MaxQueuesPerSession 65 MaxQueueDepth 512 AcceptorPollRate 10000 [Nvme] TransportId "trtype:PCIe traddr:0000:01:00.0" Nvme1 TransportId "trtype:PCIe traddr:0000:03:00.0" Nvme2 TransportId "trtype:PCIe traddr:0000:82:00.0" Nvme3 TransportId "trtype:PCIe traddr:0000:84:00.0" Nvme4

Item	Description
	RetryCount 4 Timeout 0 ActionOnTimeout None AdminPollRate 100000 HotplugEnable No [Subsystem1] NQN nqn.2016-06.io.spdk:dev16.nvme1 Listen RDMA 192.168.10.32:4421 AllowAnyHost Yes SN SPDK00000000000001 Namespace Nvme1n1 1 Namespace Nvme2n1 2 Namespace Nvme3n1 3 Namespace Nvme4n1 4
SPDK NVMe-oF Initiator	modprobe ib_cm modprobe ib_core modprobe ib_ucm modprobe ib_umad modprobe ib_uverbs modprobe iw_cm modprobe rdma_cm modprobe rdma_ucm HUGEMEM=20480 /root/spdk-master/scripts/setup.sh

FIO Configuration

To use the SPDK FIO plugin with FIO, we specified the plugin binary using LD_PRELOAD when running FIO and set ioengine=spdk in the FIO configuration file.

Table 8: FIO Configuration

Item	Description

FIO configuration file	# cat config.fio
	[global]
	rw=${iotype} # randread/randwrite
	bs=${bs_size} # 4KB
	numjobs=${numjobs_set} # local: 1 to16, nvmf: 1 to 12 (1 is used in latency test)
	iodepth=${iodepth_set} # For IOPS tests, this is adjusted so that QD per device is 128.
	runtime=${Max_runtime} # IOPS test: 600, Latency test: 1800
	group_reporting
	name=test
	direct=1
	ioengine=${ioengine} #kernel: libaio, SPDK: spdk
	time_based
	loops=1
	invalidate=1
	ramp_time=${ramp_time} #IOPS test: 180, Latency test: read:300, write:1800
	randrepeat=0
	norandommap
	exitall
	cpus_allowed_policy=split
	thread=1
	lockfile=none

Item	Description
	[job1] cpus_allowed=${CPUCORE_NVME0} filename=trtype=RDMA adrfam=IPv4 traddr=192.168.10.32 trsvcid=4421 subnqn=nqn.2016-06.io.spdk:dev16.nvme1 ns=1 #nvmf #filename=trtype=PCIe traddr=0000.01.00.0 ns=1 #local [job2] cpus_allowed=${CPUCORE_NVME1} filename=trtype=RDMA adrfam=IPv4 traddr=192.168.10.32 trsvcid=4421 subnqn=nqn.2016-06.io.spdk:dev16.nvme1 ns=2 #filename=trtype=PCIe traddr=0000.03.00.0 ns=1 [job3] cpus_allowed=${CPUCORE_NVME2} filename=trtype=RDMA adrfam=IPv4 traddr=192.168.10.32 trsvcid=4421 subnqn=nqn.2016-06.io.spdk:dev16.nvme1 ns=3 #filename=trtype=PCIe traddr=0000.82.00.0 ns=1 [job4] cpus_allowed=${CPUCORE_NVME3} filename=trtype=RDMA adrfam=IPv4 traddr=192.168.10.32 trsvcid=4421 subnqn=nqn.2016-06.io.spdk:dev16.nvme1 ns=4 #filename=trtype=PCIe traddr=0000.84.00.0 ns=1 #Set the values of cpus_allowed to CPU core IDs that are mutually exclusive among jobs.

SPDK Perf Configuration

We also collected performance data using the SPDK Perf performance test tool. To test the Linux Kernel components with Perf we configured Perf to consume Linux AIO BDEVs.

Table 9: SPDK Perf Configuration

Item	Description
SPDK Perf	/root/spdk-master/examples/nvme/perf/perf -q ${iodepth_set} -s ${bs_size} -w ${iotype} -t ${Max_runtime} -c 0x${CPUCORE} -D -r 'trtype:RDMA adrfam:IPv4 traddr:192.168.10.32 trsvcid:4421' -r 'trtype:RDMA adrfam:IPv4 traddr:192.168.10.32 trsvcid:4422' -r 'trtype:RDMA adrfam:IPv4 traddr:192.168.10.32 trsvcid:4423' -r 'trtype:RDMA adrfam:IPv4 traddr:192.168.10.32 trsvcid:4424’ ${iodepth_set}: # For IOPS tests, it is adjusted so that QD per device is 128. ${bs_size}: 4KB ${iotype}: randread or randwrite ${Max_runtime}: #IOPS: 100, Latency: read:60, write:60 ${CPUCORE}: specify the CPU cores used by perf via the core mask

Performance Results

The following was some changes we made to get the best performance:

- ✓ About the queue depth, P3700 NVMe SSD saturate at a queue depth of 128. Hence, we adjusted the queue depth per SSD to 128 for all IOPS test cases.
- ✓ About pre-conditioning,
  - ➢ We used the following command to pre-condition SSDs for 20 minutes for random read tests, ./perf -q 32 -s 131072 -w write -t 1200.
  - ➢ We used the following command to pre-condition SSDs for 90 minutes for random write tests, ./perf -q 32 -s 4096 -w randwrite -t 5400.
- ✓ About the ordering of test cases, to keep SSDs used in subsequent test case active we ran test cases such that the ordering of running is the descending order to number of SSDs.

4KB Random Read IOPS on Local NVMe

Observation:

- ✓ The SPDK polled-mode driver saturated all 4 P3700 NVMe SSDs using a single CPU core with the SPDK Perf tool.
- ✓ The CPU utilization of the SPDK polled-mode driver was 8 times more efficient than the Linux Kernel NVMe driver with the SPDK Perf tool vs. 4 times more efficient with FIO.
- ✓ The implementation efficiency of the application contributed performance improvement for SPDK (no effect for Kernel).
- ✓ The 2.1M IOPS with the SPDK polled-mode driver and the SPDK Perf tool exceeded the P3700 NVMe SSD specification. We haven't found the cause yet.

Table 10: 4KB Random Read IOPS on Local NVMe

IO tool	IO engine	IO tool cores	IOPS (K)
Perf	SPDK	1	2147
		2	2148
		4	2130
		8	2138
		16	2155
FIO Plugin		1	1438
		2	1814
		4	1813
		8	1813
		16	1856
Perf	Kernel	1	427
		2	798
		4	1471
		8	2142
		16	2143
FIO		1	316
		2	602
		4	1141
		8	1841
		16	1848

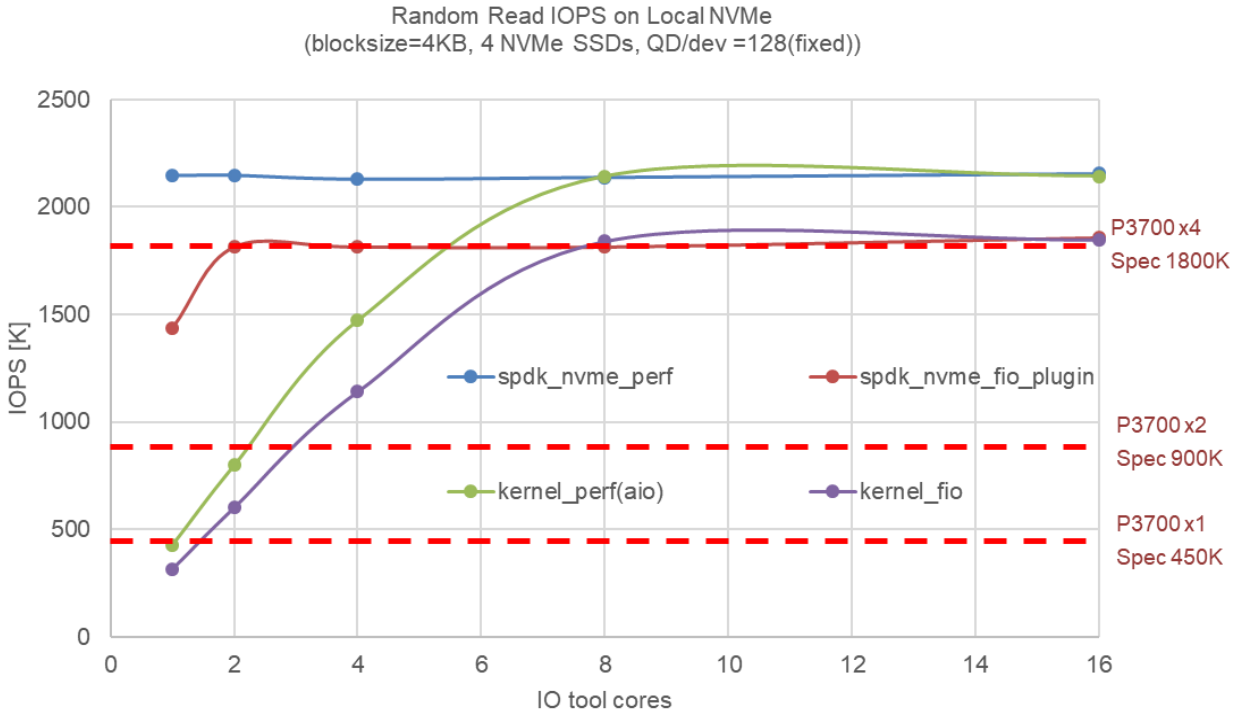

Figure 7: 4KB Random Read IOPS on Local NVMe

4KB Random Write IOPS on Local NVMe

Observation:

- ✓ The SPDK polled-mode driver used less CPU cores than the Linux kernel to saturate all 4 P3700 NVMe SSDs.
- ✓ With the SPDK Perf tool, the maximum IOPS exceeded the P3700 NVMe SSD specification due to the issue with the random number generator in the SPDK Perf tool. We have added a request to implement a perfectly uniform random number generator to the SPDK community backlog.
- ✓ The Intel® Turbo Boost Technology improved the performance of the Kernel by 22%.
- ✓ Additional pre-conditioning was essential to obtain accurate and repeatable results for the 4K 100% random write workload. We ran the workload for 90 minutes before starting the benchmark test and collecting performance data to precondition the NVMe SSDs.

Table 11: 4KB Random Write IOPS on Local NVMe

IO tool	IO engine	IO tool cores	IOPS (K)
Perf	SPDK	1	410
		2	384
		4	739
		8	481
		16	466
FIO Plugin		1	336
		2	336
		4	336
		8	335
		16	336
FIO	Kernel	1	313
		2	336
		4	336
		8	337
		16	337

CPU Efficiency of 4KB Random Read/Write on Local NVMe

Observation:

- ✓ The SPDK polled-mode driver used less CPU cores than the Linux Kernel to saturate all 4 P3700 NVMe SSDs.
- ✓ For the 4KB random read workload, the Linux Kernel NVMe driver used up to 8 times more CPU cores obtained than SPDK with the SPDK Perf tool.
- ✓ For the 4KB random read workload, the SPDK polled-mode driver used 2 times more CPU cores with FIO vs. the SPDK Perf tool.
- ✓ For the 4KB random write workload, we didn't compare the IOPS obtained SPDK + Perf and SPDK + FIO due to the issue with the random number generator in the SPDK + Perf.
- ✓ For the 4KB random write workload, 4 P3700 NVMe SSDs were not enough to saturate a single CPU core with the SPDK polled-mode driver.

Table 12: CPU Efficiency of 4KB Random Read/Write on Local NVMe

IOPS on Local NVMe (K)		Required IO tool cores (Less is better)		
		Kernel FIO	Kernel Perf	SPDK FIO Plugin
4KB Random Read	450	2	1	1
	900	4	4	1
	1800	8	8	2
4KB Random Write	75	1	No data	1
	150	1	No data	1
	300	1	No data	1

4KB Random Read IOPS on NVMe-oF

Observation:

- ✓ The SPDK NVMe-oF Initiator and Target attained up to 8 times more IOPS/CPU core vs. the Linux Kernel NVMe-oF Initiator and Target with the SPDK Perf tool.
- ✓ Using the SPDK NVMe-oF Initiator with the SPDK Perf tool, we achieved 2.1M IOPS on a single CPU core, which saturated the 4 P3700 NVMe SSDs on the SPDK NVMe-oF Target.
- ✓ The maximum IOPS on the system was limited by the number of SSDs.
- ✓ The 2.1M IOPS that we attained with the SPDK Perf tool exceeded the P3700 NVMe SSD specification. We haven't found the cause yet.
- ✓ The SPDK NVMe-oF Initiator used up to 2 times the number of CPU cores with FIO vs. the SPDK Perf tool to saturate all 4 P3700 NVMe SSDs.
- ✓ We had to use both the SPDK NVMe-oF Initiator and Target to get the maximum IOPS of 4 P3700 NVMe SSDs.

Table 13: 4KB Random Read IOPS on NVMe-oF

IO tool	Target	Initiator	Target cores	IO tool cores	IOPS (K)
Perf	SPDK	SPDK	1	1	1163
			1	2	1164
			1	4	1160
			2	1	2150
			2	2	2106
			2	4	2136
FIO Plugin			1	1	1048
			1	2	1130
			1	4	1110
			2	1	1046
			2	2	1848
			2	4	1845
Perf	Kernel	Kernel	1	1	178
			2	2	443
			4	4	618
			8	8	1039
FIO			1	1	126
			2	2	297

IO tool	Target	Initiator	Target cores	IO tool cores	IOPS (K)
			4	4	448
			8	8	713
Perf	SPDK		1	1	170
			1	2	439
			1	4	542
			1	8	1004
			2	1	166
			2	2	435
			2	4	554
			2	8	1013
	Kernel	SPDK	1	1	186
			2	2	457
			4	4	637
			8	8	1089

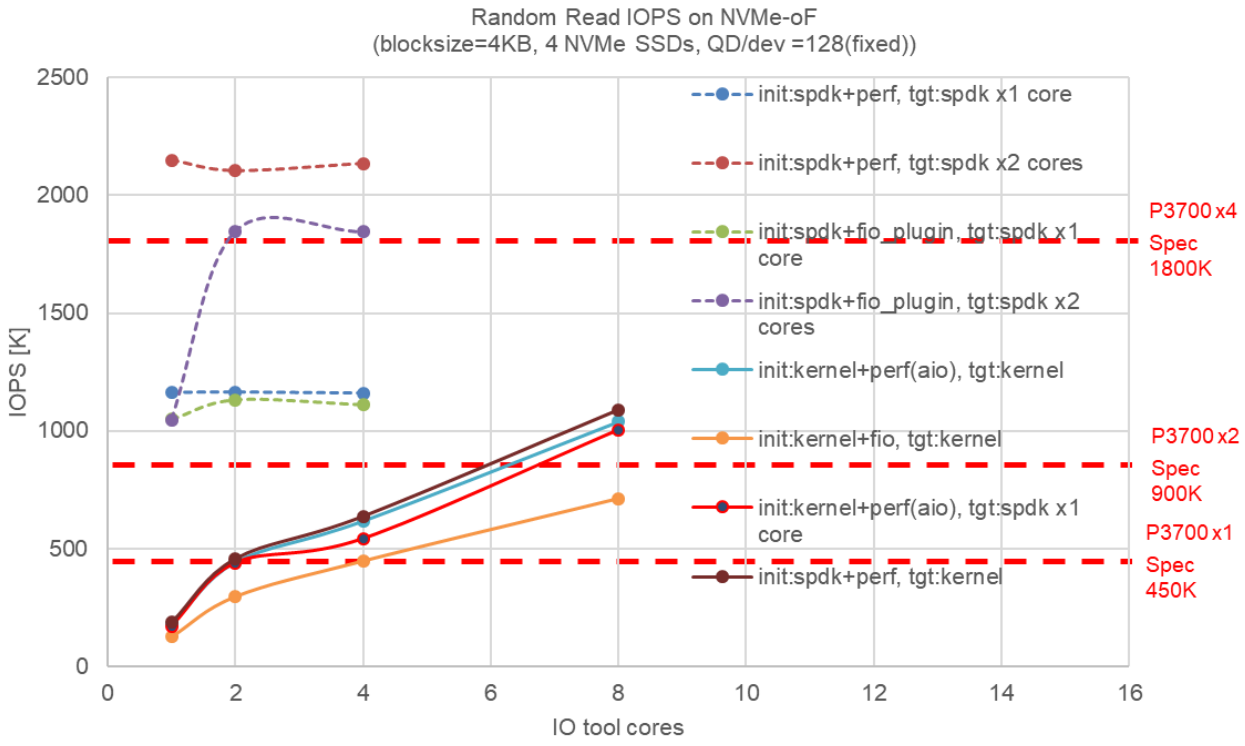

Figure 8: 4KB Random Read IOPS on NVMe-oF

4KB Random Write IOPS on NVMe-oF

Observation:

- ✓ For the 4KB random write workload, the SPDK NVMe-oF Initiator and Target can saturate more than 4 P3700 NVMe SSDs on a single CPU core.

- ✓ The Kernel NVMe-oF Initiator and Target used 2 CPU cores each to attain the maximum IOPS of the 4 P3700 NVMe SSDs. Therefore, the initiator overhead consumed an entire CPU core because locally the Kernel NVMe driver used a single CPU core to attain the maximum IOPS of the 4 P4700 NVMe SSDs with FIO.
- ✓ When using the Kernel NVMe-oF Initiator and Target, the Kernel NVMe-oF Target automatically used the same number of CPU cores specified in the Kernel NVMe-oF Initiator.
- ✓ The Intel® Turbo Boost Technology improved the performance of the Kernel by 26%.
- ✓ We did not use the SPDK Perf tool for the 4KB random write workload over NVMe-oF due to an issue with the quality of the random number generator that we found in the local NVMe test.

Table 14: 4KB Random Write IOPS on NVMe-oF

IO tool	Target	Initiator	Target cores	IO tool cores	IOPS (K)
FIO Plugin	SPDK	SPDK	1	1	336
			1	2	336
			1	4	334
			1	8	336
FIO	Kernel	Kernel	1	1	137
			2	2	298
			4	4	333
			8	8	336

CPU Efficiency of 4KB Random Read/Write on NVMe-oF

Observation:

For the 4KB random read workload,

- ✓ SPDK used fewer CPU cores than Kernel. The CPU efficiency increased with IOPS. We had to use both the SPDK Initiator and Target to saturate the 4 P3700 NVMe SSDs.
- ✓ on Initiator, the Kernel Target used up to 8 times more CPU cores than the SPDK Target when testing with the SPDK Perf tool.
- ✓ on Initiator, the SPDK Initiator with the SPDK Perf tool used half of CPU cores with FIO.
- ✓ on Target, the Kernel Target used up to 8 times more CPU cores than the SPDK Target when testing with the SPDK Perf tool.

For the 4KB random write workload,

- ✓ All four P3700 NVMe SSDs were saturated using a single CPU core with SPDK. We need to increase the number of NVMe SSDs to evaluate the CPU efficiency for this workload.

Table 15: CPU Efficiency of 4KB Random Read Workload on NVMe-oF

IOPS on NVMe-oF (K)		(Kernel Initiator, Kernel Target) + FIO	(Kernel Initiator, Kernel Target) + Perf	(SPDK Initiator, SPDK Target) + FIO	(SPDK Initiator, SPDK Target) + Perf	(Kernel Initiator, SPDK Target) + Perf	(SPDK Initiator, Kernel Target) + Perf
Initiator	450	4	2	1	1	2	2
	900	No data	8	1	1	8	8
	1800	No data	No data	2	1	No data	No data
Target	450	4	2	1	1	1	2
	900	No data	8	1	1	1	8
	1800	No data	No data	2	2	No data	No data

4KB Random Read/Write Latency

We collected the latencies using FIO and Perf IO tools and compared them.

Observation:

For FIO results,

- ✓ The total overhead of the Kernel NVMe-oF Initiator and Target was 23.27 usec, which was obtained by reducing the latency of the SPDK Target and SPDK Initiator with FIO Plugin (84.69 usec) from the latency of the Kernel Target and Kernel Initiator with FIO (107.96 usec).
- ✓ Our methodology was blackbox testing. We will perform an internal breakdown of the latency for SPDK and Kernel.

Table 16: 4KB Random Read/Write Latency

Access	IO tool	Target	Initiator	Latency (usec)		
				Local	NVMe-oF	Difference of Local and NVMe-oF
Ramdom Read	Perf	SPDK	SPDK	82.29	82.98	+0.69
	FIO Plugin	SPDK	SPDK	82.43	84.69	+2.26
	Perf	Kernel	Kernel	82.03	98.24	+16.21
	FIO	Kernel	Kernel	89.98	107.96	+17.98
Random Write	FIO Plugin	SPDK	SPDK	14.71	14.71	+2.74
	FIO	Kernel	Kernel	17.16	26.79	+9.63

Conclusion

Through this work we verified the superior efficiency of SPDK as reported in [1] and acquired deeper understanding of SPDK, NVMe, and NVMe-oF. For the 4KB random read workload, the SPDK NVMe-oF Initiator and Target used CPU cores 8 times more efficiently than the Kernel with Perf. For latency, SPDK was better than Kernel, however, our current methodology is not enough, and further investigation will be necessary.

Acknowledgements

John Kariuki, James R. Harris, Vishal Verma, and Benjamin Walker at Intel, and Kazuhiro Akimoto at Hitachi provided excellent feedback and guidance on this work. Yuma Higuchi at Intel provided excellent coordination.

Reference

[1] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, Luse E Paul, SPDK: A development kit to build performance storage applications, In IEEE 9th International Coference on Cloud Computing Technology and Science, 2017, http://ieeexplore.ieee.org/document/8241103/