NVMe “TIPS AND TRICKS”

Jonmichael Hands, Strategic Planner / Product Line Manager for Intel Data Center SSDs

Intel Non-Volatile Memory Solutions Group

Dec 2018
JONMICHAEL HANDS
“Strategic Plancer NVMe Wizard”
Agenda

• Using NVMe-cli to manage SSDs
  • How to perform a format / secure erase, monitor temperature, updating device firmware, read logs, etc.
• Setting up NVMe RAID in Linux
• Tuning Linux for high performance NVMe SSDs with storage class memory
• Overprovision NVMe SSD to improve endurance and performance
• Using Intel Storage Analytics Tool to evaluate workloads
MANAGE NVME SSDs WITH NVME-CLI
NVMe CLI overview

• nVMe-cli is the standard open source tool to manage NVMe SSD in Linux
• It can perform standard management commands like reading SMART and error logs, updating firmware, secure erase, and monitoring temperature

✓ Open Source
✓ Vendor plugins
✓ Supports new NVMe specs, commands, features
✓ Powerful feature set

Common datacenter management needs
• Format
• Secure erase
• FW update
• Controller reset to load FW
• Health status
• Log page reads including vendor log pages
• SMART status
• List devices
• Get/set features
• Namespace management
• Identify controller, namespace, and NVM Set
• Endurance log page
## Some helpful commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ nvme --version</td>
<td>NVMe CLI version</td>
</tr>
<tr>
<td>$ nvme help</td>
<td>List of all commands</td>
</tr>
<tr>
<td>$ sudo nvme list</td>
<td>Lists out nvme devices, capacity, namespace, format and FW rev</td>
</tr>
<tr>
<td>$ sudo nvme format /dev/nvme0 -s 1</td>
<td>Format with secure erase option</td>
</tr>
<tr>
<td>$ sudo nvme id-ctrl /dev/nvme0</td>
<td>Identify controller details</td>
</tr>
<tr>
<td>$ sudo nvme intel id-ctrl /dev/nvme0n1</td>
<td>Identify controller details - few additional details added (e.g. drive</td>
</tr>
<tr>
<td></td>
<td>health, boot loader</td>
</tr>
<tr>
<td>$ sudo nvme intel market-name /dev/nvme0n1</td>
<td>Marketing series name</td>
</tr>
<tr>
<td>$ sudo nvme id-ns /dev/nvme0n1</td>
<td>Identify namespace details</td>
</tr>
<tr>
<td>$ sudo nvme get-log /dev/nvme0 --log-id=2 --log-len=512</td>
<td>Read log page</td>
</tr>
<tr>
<td>$ sudo nvme smart-log /dev/nvme0</td>
<td>Smart log</td>
</tr>
<tr>
<td>$ sudo nvme error-log /dev/nvme0</td>
<td>less</td>
</tr>
<tr>
<td>$ sudo nvme fw-download /dev/nvme0n1 -f &lt;fw binary&gt;</td>
<td>Expected result: Firmware download success</td>
</tr>
<tr>
<td>$ sudo nvme fw-commit /dev/nvme0n1 -s 1 -a 1</td>
<td>Expected result: Success activating firmware action:1 slot:1</td>
</tr>
<tr>
<td>$ sudo nvme reset /dev/nvme0</td>
<td>Resets the nvme controller</td>
</tr>
</tbody>
</table>
Format - Secure Erase

nvme format -s 1

SES = 0 results in drive being formatted and trimmed

SES = 1 for Block Erase – low level block erase on media (physically erase NAND blocks, or overwrite with 0 on SCM)

SES = 2 for Crypto Erase - change media encryption key

-l for LBA format, find capabilities in NVMe identify controller

-t timeout for devices that will take a long time

-p for protection information with variable sector size
Sanitize (not widely supported yet)

nvme sanitize

Alters user data so that it is unrecoverable by erasing media, metadata, and cache

Use when retiring SSD from use, reusing for new use case, or end of life

Modes in Sanitize

• Block Erase – low level block erase on media (physically erase NAND blocks)
• Crypto Erase - change media encryption key
• Overwrite – overwrite with data patterns (not good or recommended for NAND based SSDs due to endurance)

Sanitize vs Format Unit in NVMe – keeps going after reset, and erases all metadata, log pages and status during operation
NVMe-cli includes full support for NVMe commands

nvme-1.6
usage: nvme <command> [<device>] [<args>]

The following are all implemented sub-commands:

list                  List all NVMe devices and namespaces on machine
list-subsys            List nvme subsystems
id-ctrl               Send NVMe Identify Controller
id-ns                 Send NVMe Identify Namespace, display structure
list-ns               Send NVMe Identify List, display structure
ns-descs              Send NVMe Namespace Descriptor List, display structure
id-nvmset             Send NVMe Identify NVM Set List, display structure
create-ns             Creates a namespace with the provided parameters
delete-ns             Deletes a namespace from the controller
attach-ns             Attaches a namespace to requested controller(s)
detach-ns             Detaches a namespace from requested controller(s)
list-ctrl             Send NVMe Identify Controller List, display structure
get-ns-id             Retrieve the namespace ID of opened block device
get-log               Generic NVMe get log, returns log in raw format
telemetry-log         Retrieve FW Telemetry log to file
fw-log                Retrieve FW Log, show it
changed-ns-list-log   Retrieve Changed Namespace List, show it
smart-log             Retrieve SMART Log, show it
error-log             Retrieve Error Log, show it
effects-log           Retrieve Command Effects Log, show it
endurance-log         Retrieve Endurance Group Log, show it
get-feature           Get feature and show the resulting value
device-self-test      Perform the necessary tests to observe the performance
self-test-log          Retrieve the SELF-TEST Log, show it
set-feature           Set a feature and show the resulting value
set-property           Set a property and show the resulting value
get-property           Get a property and show the resulting value
format                Format namespace with new block format
fw-commit             Verify and commit firmware to a specific slot (fw-activate in old version < 1.2)
fw-download           Download new firmware
admin-passthru        Submit an arbitrary admin command, return results
io-passthru           Submit an arbitrary IO command, return results
security-send         Submit a Security Send command, return results
security-recv         Submit a Security Receive command, return results
resv-acquire          Submit a Reservation Acquire, return results
resv-register         Submit a Reservation Register, return results
resv-release          Submit a Reservation Release, return results
resv-report           Submit a Reservation Report, return results
dsm                   Submit a Data Set Management command, return results
flush                 Submit a Flush command, return results
compare               Submit a Compare command, return results
read                  Submit a read command, return results
write                 Submit a write command, return results
write-zeroes          Submit a write zeroes command, return results
write-uncor           Submit a write uncorrectable command, return results
sanitize              Submit a sanitize command
sanitize-log          Retrieve sanitize log, show it
reset                 Resets the controller
subsystem-reset       Resets the controller
ns-rescan             Rescans the NVMe namespaces
show-regs             Shows the controller registers or properties. Requires character device
discover              Discover NVMeoF subsystems
connect-all           Discover and Connect to NVMeoF subsystems
connect               Connect to NVMeoF subsystem
disconnect            Disconnect from NVMeoF subsystem
gen-hostnqn           Generate NVMeoF host NQN
dir-receive           Submit a Directive Receive command, return results
dir-send              Submit a Directive Send command, return results
version               Shows the program version
help                  Display this help
Learn about SSD functionality

nvme id-ctrl

5.15.3 Identify Controller data structure (CNS 01h)
The Identify Controller data structure (refer to Figure 111) is returned to the host for this controller.

Figure 111: Identify – Controller Data Structure

<table>
<thead>
<tr>
<th>Bytes</th>
<th>O/M</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:08</td>
<td>M</td>
<td>PCI Vendor ID (VID): Contains the company vendor identifier that is assigned by the PCI SIG. This is the same value as reported in the ID register in section 2.11.</td>
</tr>
<tr>
<td>03:02</td>
<td>M</td>
<td>PCI Subsystem Vendor ID (SSVID): Contains the company vendor identifier that is assigned by the PCI SIG for the subsystem. This is the same value as reported in the SS register in section 2.1.17</td>
</tr>
<tr>
<td>23:04</td>
<td>M</td>
<td>Serial Number (SN): Contains the serial number for the NVM subsystem that is assigned by the vendor as an ASCII string. Refer to section 7.10 for unique identifier requirements. Refer to section 1.5 for ASCII string requirements.</td>
</tr>
<tr>
<td>63:24</td>
<td>M</td>
<td>Model Number (MN): Contains the model number for the NVM subsystem that is assigned by the vendor as an ASCII string. Refer to section 7.10 for unique identifier requirements. Refer to section 1.5 for ASCII string requirements.</td>
</tr>
<tr>
<td>71:64</td>
<td>M</td>
<td>Firmware Revision (FR): Contains the currently active firmware revision for the NVM subsystem. This is the same revision information that may be retrieved with the Get Log Page command, refer to section 5.14.3. Refer to section 1.5 for ASCII string requirements.</td>
</tr>
<tr>
<td>72</td>
<td>M</td>
<td>Recommended Arbitration Burst (RAB): This is the recommended Arbitration Burst size. The value is in commands and is reported as a power of two (2^n). This is the same units as the Arbitration Burst limit.</td>
</tr>
<tr>
<td>75:73</td>
<td>M</td>
<td>IEEE OUI Identifier (IEEE): Contains the Organization Unique Identifier (OUI) for the controller vendor. The OUI shall be a valid IEEE/RAF assigned identifier that may be registered at <a href="http://standards.ieee.org/developers/rafi/public.html">http://standards.ieee.org/developers/rafi/public.html</a>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NVME Identify Controller:</th>
</tr>
</thead>
<tbody>
<tr>
<td>vid : 0x8086</td>
</tr>
<tr>
<td>ssvid : 0x8086</td>
</tr>
<tr>
<td>sn : PHLE7042009M1P6CGN</td>
</tr>
<tr>
<td>mn : INTEL SSDPE2KE016T7</td>
</tr>
<tr>
<td>fr : QDV10150</td>
</tr>
<tr>
<td>rab : 0</td>
</tr>
<tr>
<td>ieee : 5cd2e4</td>
</tr>
<tr>
<td>cmic : 0</td>
</tr>
<tr>
<td>mdts : 5</td>
</tr>
<tr>
<td>cntlid : 0</td>
</tr>
<tr>
<td>ver : 10200</td>
</tr>
<tr>
<td>rtd3r : 1e8480</td>
</tr>
<tr>
<td>rtd3e : 2dc6c0</td>
</tr>
<tr>
<td>oaes : 0</td>
</tr>
<tr>
<td>ctratt : 0</td>
</tr>
<tr>
<td>rrls : 0</td>
</tr>
<tr>
<td>oacs : 0x6</td>
</tr>
<tr>
<td>acl : 3</td>
</tr>
<tr>
<td>aerl : 3</td>
</tr>
<tr>
<td>frmw : 0x2</td>
</tr>
<tr>
<td>lpa : 0x2</td>
</tr>
<tr>
<td>elpe : 63</td>
</tr>
<tr>
<td>npss : 0</td>
</tr>
<tr>
<td>avsc : 0</td>
</tr>
<tr>
<td>apsta : 0</td>
</tr>
<tr>
<td>wctemp : 343</td>
</tr>
<tr>
<td>cctemp : 353</td>
</tr>
<tr>
<td>mtfa : 0</td>
</tr>
<tr>
<td>mtpre : 0</td>
</tr>
<tr>
<td>mhmmin : 0</td>
</tr>
<tr>
<td>tmvcap : 1600321314816</td>
</tr>
<tr>
<td>unvmcap : 0</td>
</tr>
<tr>
<td>rpmbs : 0</td>
</tr>
<tr>
<td>edstt : 0</td>
</tr>
<tr>
<td>dsto : 0</td>
</tr>
</tbody>
</table>

| fwug : 0            |
| kas : 0             |
| hctma : 0           |
| mntmt : 0           |
| mxmt : 0            |
| sanicap : 0         |
| hcmmd : 0           |
| nsetidmax : 0       |
| sqes : 0x66         |
| cqes : 0x44         |
| fnc : 0x4           |
| wvc : 0             |
| awun : 0            |
| awuwpt : 0          |
| nsvcc : 0           |
| acwu : 0            |
| sgls : 0            |
| subqon : 0          |
| ioccsz : 0          |
| iorcsz : 0          |
| icdoff : 0          |
| cstratr : 0         |
| msdbdd : 0          |
| ps : 0              |
| enlat0exlat0rrt0rrl0 : 0 |
| rwt0rwt0idle_power : 0 |
| active_power : -     |
Discover info about namespace

nvme id-ns

NVME Identify Namespace 1:
nsze : 0xba4d4ab0
ncap : 0xba4d4ab0
nuse : 0xba4d4ab0
nsfeat : 0
nlbaf : 1
fibas : 0
mc : 0
dpc : 0
dps : 0
nmic : 0
rescap : 0
fpi : 0
dlfeat : 0
nawun : 0
nawupf : 0
nacwu : 0
nabsn : 0
nabo : 0
nabspf : 0
noiob : 0
nvmcap : 1600321314816
nvmsetid: 0
endgid : 0
nguid : 01000000010000005cd2e482ec674d51
eui64 : 5cd2e482ec670000
lbaf 0 : ms:0 lbads:9 rp:0x2 (in use)
lbaf 1 : ms:0 lbads:12 rp:0
Standard SMART

nvme smart-log

Smart Log for NVME device: nvme0 namespace-id: ffffffff

critical_warning: 0
temperature: 43 C
available_spare: 100%
available_spare_threshold: 10%
percentage_used: 0%
data_units_read: 87
data_units_written: 0
host_read_commands: 2,882
host_write_commands: 0
controller_busy_time: 0
power_cycles: 16
power_on_hours: 2
unsafe_shutdowns: 11
media_errors: 0
num_err_log_entries: 0
Warning Temperature Time: 0
Critical Composite Temperature Time: 0
Thermal Management T1 Trans Count: 0
Thermal Management T2 Trans Count: 0
Thermal Management T1 Total Time: 0
Thermal Management T2 Total Time: 0
NVMe RAID WITH MDADM
Intel® VROCD provides compelling RAID solution for NVMe* SSDs

*Other names and brands may be claimed as the property of others.
## Intel® VROC Major Features

<table>
<thead>
<tr>
<th>Major Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bootable RAID</strong></td>
<td>Build redundancy to protect your system volume</td>
</tr>
<tr>
<td><strong>Hot Insert and Surprise Removal</strong></td>
<td>Expand volume, replace bad drive without system reboot</td>
</tr>
<tr>
<td><strong>LED Management</strong></td>
<td>Visually display RAID status via LEDs</td>
</tr>
<tr>
<td><strong>Close RAIDX Write Hole</strong></td>
<td>Maintain data integrity with no battery backup unit</td>
</tr>
<tr>
<td><strong>RAID management</strong></td>
<td>Remote management via webpage or RESTful APIs, pre-OS management via UEFI HII and CLI, OS management via GUI and CLI</td>
</tr>
<tr>
<td><strong>3rd Party vendor SSD support</strong></td>
<td>Validation, WHQL and support for selected 3&lt;sup&gt;rd&lt;/sup&gt; party SSDs</td>
</tr>
<tr>
<td><strong>OS support</strong></td>
<td>Windows* and Linux*. Intel® VROC for Linux is open-source, builds on MDRAID</td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.*
Performance – RAID vs Pass-thru
RHEL7.4 with Intel® Optane™ SSD DC P4800X³
(4k Random)

- Pass-thru raw data:
  - 4k Rand Write: 558k IOPS
  - 4k Rand Mixed: 506k IOPS
  - 4k Rand Read: 586k IOPS

- 4-Disk RAID0 Read: 2.3M IOPS

- Physical CPU Cores Used:
  - 4-Disk RAID0 Read: 3.4 Cores
  - 4-Disk RAID5 Write: 3.3 Cores

See appendix for footnotes.

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 September 12, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.
Performance – RAID vs Pass-thru
RHEL7.4 with Intel® Optane™ SSD DC P4800X4
(128 Seq., 1 Worker)

- Pass-thru raw data:
  - 128k Seq. Write: 1.7GB/s
  - 128k Seq. Read: 2.7 GB/s
- 4-Disk RAID 0 Read: 8.1 GB/s
- Physical CPU Cores Used:
  - 4-Disk RAID0 Read: 0.5 Cores
  - 4-Disk RAID5 Write: 0.9 Cores

52 total physical cores on this 2 socket, Intel® Xeon®
8170 based system

See appendix for footnotes.
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](http://www.intel.com/benchmarks).
Performance results are based on testing as of September 12, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.
Setting up basic RAID volume in Linux

- Enable Intel VMD in BIOS (varies by vendor)
- Remove the metadata:
  
  ```
  # mdadm --zero-superblock /dev/nvme0n1
  ```
- Format SSD:
  
  ```
  # sudo nvme format /dev/nvme0
  ```
- Create RAID Array: Example of RAID 5 of 4 drives
  
  ```
  # mdadm -C /dev/md0 /dev/nvme[0-3]n1 -n 4 -e imsm
  # mdadm -C /dev/md/Volume0 /dev/md0 -n 4 -l 5
  ```
RAID Volume Creation

WARNING: Creating a RAID array will permanently delete any existing data on the selected drives. Back up all important data before beginning these steps.

The following shows an example of how to create a RAID5 array with 4 Intel NVMe SSDs:

1. First, a container that establishes Intel IMSM metadata must be created.

   ```bash
   # mdadm -C /dev/md0 /dev/nvme[0-3]n1 -n 4 -e imsm Continue creating array? Y
   mdadm: container /dev/md0 prepared.
   ```

   The command creates a RAID container with Intel® Matrix Storage Manager metadata format. The device node for the container will be /dev/md0. In this example drives nvme0n1, nvme1n1 nvme2n1 and nvme3n1 are used for this RAID container and the total number of drives is 4. The wildcard expression /dev/nvme[0-3]n1 can be used to specify the range of drives. Individual drives can be also used.

2. Next, a RAID 5 volume is created.

   ```bash
   # mdadm -C /dev/md/Volume0 /dev/md0 -n 4 -l 5
   ```

   The command creates a RAID 5 volume /dev/md/Volume0 within the /dev/md0 container.

   The following command parameters may also be used to give finer control for the creation of the RAID volume.
   - `-n` Number of active RAID devices to be used in the volume.
   - `-x` Specifies the number of spare devices in the initial array.
   - `-c` Specifies the chunk (strip) size in Kilobytes. The default is 128KiB. This value must be a multiple of 4KiB. Optimal chunk size should be considered depending on expected workload profiles.
   - `-l` specifies the RAID level. The options are 0, 1, 5, 10.
   - `-z` Specifies the size (in Megabytes) of space dedicated on each disk to the RAID volume. This must be a multiple of the chunk size. For example:

   ```bash
   # mdadm -C /dev/md/Volume0 /dev/md0 -n 4 -l 5 -z $((100*1024*1024))
   ```

   The command above creates a RAID volume inside the /dev/md0 container with 100GiB of disk space used on each drive member.

   A suffix of ‘M’ or ‘G’ can be given to indicate Megabytes or Gigabytes respectively. This applies also to the `-c` parameter. So the above command is equivalent to this:

   ```bash
   # mdadm -C /dev/md/Volume0 /dev/md0 -n 4 -l 5 -z 100G
   ```
TUNING LINUX FOR TESTING HIGH-PERFORMANCE AND LOW LATENCY NVMe
## Improved Efficiency for PCIe and NVMe SSDs

### 256B Max Payload Size

<table>
<thead>
<tr>
<th>PCIe Link</th>
<th>I/O size (B)</th>
<th>Read BW (GB/s) @ kIOPS</th>
<th>Write BW (GB/s) @ kIOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen 3 x 4</td>
<td>4096</td>
<td>3.5 GB/s @ 850 kIOPS</td>
<td>3.5 GB/s @ 850 kIOPS</td>
</tr>
<tr>
<td>Gen 3 x 8</td>
<td>4096</td>
<td>7.0 GB/s @ 1700 kIOPS</td>
<td>7.0 GB/s @ 1700 kIOPS</td>
</tr>
<tr>
<td>Gen 3 x 4</td>
<td>131072</td>
<td>3.55</td>
<td>3.5</td>
</tr>
<tr>
<td>Gen 3 x 8</td>
<td>131072</td>
<td>7.1</td>
<td>7.0</td>
</tr>
</tbody>
</table>

### 512B Max Payload Size

<table>
<thead>
<tr>
<th>PCIe Link</th>
<th>I/O size (B)</th>
<th>Read BW (GB/s) @ kIOPS</th>
<th>Write BW (GB/s) @ kIOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen 3 x 4</td>
<td>4096</td>
<td>3.6 GB/s @ 880 kIOPS</td>
<td>3.6 GB/s @ 880 kIOPS</td>
</tr>
<tr>
<td>Gen 3 x 8</td>
<td>4096</td>
<td>7.3 GB/s @ 1780 kIOPS</td>
<td>7.3 GB/s @ 1780 kIOPS</td>
</tr>
<tr>
<td>Gen 3 x 4</td>
<td>131072</td>
<td>3.7</td>
<td>3.65</td>
</tr>
<tr>
<td>Gen 3 x 8</td>
<td>131072</td>
<td>7.45</td>
<td>7.3</td>
</tr>
</tbody>
</table>

### Gen 3x4 128K and 4K Reads

- **128/130 Encoding**
- **SKP ordered Sets**
- **NVMe TLPs**
- **NVMe Data Headers**
- **NVMe Data Payload**

---

Bank 1/2: 0

- No 128K Reads
- No 4K Reads
NVMe lowers latency

NVMe improved software latency overhead of NAND SSDs by removing controller latency

Software overhead of HBA/RAID cards and legacy protocols significantly higher than media latency of next generation storage class memory
Typical I/O Access Cycle

- End-to-end command latency is the total latency from I/O request to command completion acknowledgement.

- Figure shows a typical command execution flow of a PCIe/NVMe SSD. Each stage incurs latency.

- End-to-end latency depends on many variables. These variables include CPU speed, number of sockets, and relevant hardware and OS settings.

Typical Linux* flow of command execution on NVMe SSDs

- User application requests I/O
- Kernel processes I/O request and sends it to Device Driver
- Device Driver processes the requests
  - Creates SQ entry
  - Rings doorbell register
- Device processes the requests
  - Reads SQ entry
  - Processes IO
  - Creates CQ entry
  - Signals MIF to Host
- Kernel calls Device Driver to handle the completion
- Device Driver ack. that the entry in CQ has been processed

Latency added by System Config:
1. Faster CPU speeds up IO scheduling
2. Low power states on CPU as well as RAM adds to latency
3. Software stack (file system, I/O Block Protocol processing, context-switching, drivers) adds its own latency, and scales with CPU frequency

Device latency is measured from SQ doorbell ring to MIF-X (indicating command completion)

Latency added by System Config. IO Polling mode, instead of Interrupt mode, can improve end-to-end command latency at queue depth=1.

Color Legend:
- SSD Optimized
- Host Optimized
Best Known System Configuration for NVMe SSD Evaluation

<table>
<thead>
<tr>
<th>Setting</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Frequency</td>
<td>&gt;= 3.0 GHz</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux performs better than Windows at lower QDs</td>
</tr>
<tr>
<td></td>
<td>SPDK performs better than native Linux NVMe driver</td>
</tr>
<tr>
<td>Hyper threading</td>
<td>Disabled</td>
</tr>
<tr>
<td>EIST (Enhanced Intel Speed Step Technology)</td>
<td>Disabled</td>
</tr>
<tr>
<td>Intel Turbo Mode</td>
<td>Disabled</td>
</tr>
<tr>
<td>C-States</td>
<td>Disabled</td>
</tr>
<tr>
<td>P-States</td>
<td>Disabled</td>
</tr>
<tr>
<td>CPU Governor (through the OS)</td>
<td>Performance mode</td>
</tr>
<tr>
<td>IRQ Balancing Service (OS Service)</td>
<td>Disabled</td>
</tr>
<tr>
<td>SMP Affinity (via bash script)</td>
<td>Set interrupts to run in the proper CPU cores</td>
</tr>
<tr>
<td>I/O Polling or Hybrid Mode</td>
<td>Enabled (if supported by your kernel)</td>
</tr>
<tr>
<td></td>
<td>Note: Polling is a new feature in Linux and fio since kernel 4.8, and hybrid mode was made available in version 4.10 kernel.</td>
</tr>
<tr>
<td>Workload</td>
<td>4k aligned</td>
</tr>
</tbody>
</table>
Result shows tuning works!

Table 5: System Configuration

<table>
<thead>
<tr>
<th>Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel® Xeon® E5-2687W v4 @ 3.0 GHz (12 cores x 2)</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Intel® S2600WT</td>
</tr>
<tr>
<td>Memory</td>
<td>32GB DDR4 @ 2133Mhz (32G X 1 DIMM)</td>
</tr>
<tr>
<td>BIOS version</td>
<td>5ES6910.86B.01.01.0019</td>
</tr>
<tr>
<td>BIOS configuration</td>
<td>Hyper-threading disabled, CPU C-state disabled, P states disabled</td>
</tr>
<tr>
<td>Linux version</td>
<td>centos-7.3</td>
</tr>
<tr>
<td>Kernel version</td>
<td>4.10.8</td>
</tr>
<tr>
<td>FIO rev</td>
<td>2.18</td>
</tr>
</tbody>
</table>

Table 6: Performance Table

<table>
<thead>
<tr>
<th>Settings</th>
<th>4KB Random Read, Queue Depth =1</th>
<th>4KB Random Write, Queue Depth =16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latency (μs) IOPS</td>
<td>Latency (μs) IOPS</td>
</tr>
<tr>
<td></td>
<td>Avg 99% 99.9999% Max</td>
<td>Avg 99% 99.9999% 99.999999%</td>
</tr>
<tr>
<td>Stock OS Results</td>
<td>8.43 17 45 139 115k</td>
<td>25.37 65 334 4576 550k</td>
</tr>
<tr>
<td>Governor set to Performance</td>
<td>8.16 16 45 77 119k</td>
<td>25.61 84 119 1192 555k</td>
</tr>
<tr>
<td>IRQ Balancing Service Turned off</td>
<td>8.01 16 39 69 120k</td>
<td>25.69 65 237 2416 554k</td>
</tr>
<tr>
<td>SMP Affinity Set</td>
<td>7.96 15 44 141 122k</td>
<td>25.81 64 123 221 553k</td>
</tr>
<tr>
<td>Turbo off 2.3 GHz</td>
<td>9.2 11 36 87 107k</td>
<td>24.44 64 255 2576 556K</td>
</tr>
<tr>
<td>Turbo off 3 GHz</td>
<td>7.57 9 53 135 126k</td>
<td>24.82 63 129 1416 556k</td>
</tr>
</tbody>
</table>
Polling mode or Hybrid mode improves low queue-depth performance

Utilize I/O Polling Mode or Hybrid Polling Mode to improve performance at queue depth of 1
Queue Depth 1 (QD1) I/O performance can be enhanced by utilizing I/O Polling Mode or Hybrid Polling Mode.

- **I/O Polling Mode** was introduced with Linux Kernel 4.4. In polling mode, the CPU continuously checks if the I/O command completed, resulting in improved I/O performance for devices based on 3D XPoint technology.
  
  Enable polling mode [Linux Kernel 4.4 and above]:
  
  ```bash
  # echo 1 > /sys/block/nvme0n1/queue/io_poll
  ```

- **Hybrid Polling Mode** was introduced in Linux Kernel 4.10. Hybrid polling mode improved QD1 performance closer to that of polling mode performance, but with lower CPU utilization.
  
  Enable hybrid polling mode [Linux Kernel 4.10 and above, 1 = always poll (default) 0= hybrid]:
  
  ```bash
  # echo 0 > /sys/block/<dev>/queue/io_poll_delay
  ```

The following table shows the performance comparison of three modes.

<table>
<thead>
<tr>
<th>Mode</th>
<th>4KB Random Read, Queue Depth =1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latency (μs)</td>
<td>Avg</td>
<td>99%</td>
<td>99.999%</td>
<td>CPU util</td>
</tr>
<tr>
<td>I/O Polling</td>
<td></td>
<td>7.34</td>
<td>16</td>
<td>40</td>
<td>99.9%</td>
</tr>
<tr>
<td>Hybrid</td>
<td></td>
<td>7.54</td>
<td>15</td>
<td>41</td>
<td>58.4%</td>
</tr>
<tr>
<td>Interrupt</td>
<td></td>
<td>11.02</td>
<td>11</td>
<td>40</td>
<td>38.5%</td>
</tr>
</tbody>
</table>

Table 2: Performance Comparison for Polling, Hybrid and Interrupt Modes (4.10.8 kernel)
Pvsync2 engine improves low queue-depth performance

Table 3: QD1 Performance Comparison for ioengine and pvsync2

<table>
<thead>
<tr>
<th>Ioengine Option (with “Turbo off”)</th>
<th>Latency (µs)</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4KB Random Read, Queue Depth =1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>99%</td>
</tr>
<tr>
<td>libaio</td>
<td>8.97</td>
<td>11</td>
</tr>
<tr>
<td>pvsync2 w/ hipri</td>
<td>7.57</td>
<td>9</td>
</tr>
</tbody>
</table>
Typical I/O Execution in SPDK

User application requests I/O with completion callback and context

SPDK userspace device
Driver processes the requests
- Creates SQ entry
- Rings doorbell register

Device processes the requests
- Reads SQ entry
- Processes I/O
- Creates CQ entry

User application polls SPDK for I/O completion

SPDK calls user application completion callback when I/O is complete

Latency added by System Config:
1. Faster CPU speeds up IO scheduling
2. Low power states on CPU as well as RAM adds to latency
3. Very little software stack overhead (measured around 500ns for SPDK vs. 3000-5000ns for kernel)

Device latency is measured from SQ doorbell ring to CQ entry phase bit change detected (indicating command completion)

Latency added by System Config. Polling w/ SPDK eliminates interrupt overhead.

Color Legend:
- SSD Latency
- System Latency less SSD Latency
SPDK architecture improves Optane performance over Linux kernel

SPDK achieves device saturation at QD5 with 573k IOPS @ 8uS ave latency
Script to pull system info:

```bash
#!/bin/bash

# Author: Satvik Vyas
# Objective: gather system info
# 1. OS Kernel info
# 2. CPU info
# 3. Memory info
# 4. PCIe devices
# 5. Storage devices

# 1. OS Kernel info
echo "Kernel version________________________" > ./sysInfo_out.txt
cat /proc/version >> ./sysInfo_out.txt

# 2. CPU info
echo "________________________" >> ./sysInfo_out.txt
cat /proc/cpuinfo >> ./sysInfo_out.txt
cat /sys/devices/system/cpu/cpu/cpuinfo/scaling_governor >> ./sysInfo_out.txt
echo "________________________" >> ./sysInfo_out.txt

# 3. Memory info
echo "Memory info________________________" >> ./sysInfo_out.txt
cat /proc/meminfo >> ./sysInfo_out.txt
df -h >> ./sysInfo_out.txt
echo "________________________" >> ./sysInfo_out.txt

# 4. PCIe devices
echo "PCle devices info________________________" >> ./sysInfo_out.txt
sudo lspci -v >> ./sysInfo_out.txt
echo "________________________" >> ./sysInfo_out.txt

# 5. Storage devices
echo "Storage devices info________________________" >> ./sysInfo_out.txt
sudo nvme list >> ./sysInfo_out.txt
lsblk >> ./sysInfo_out.txt
df -h >> ./sysInfo_out.txt
sudo fdisk -l >> ./sysInfo_out.txt
echo "________________________" >> ./sysInfo_out.txt

# end
```
Script to tune system for improved I/O performance:

1. Disable IRQ Balance
2. Set SMP Affinity
3. Set CPU to Performance Mode

```bash
#!/bin/bash

systemctl stop irqbalance
folders=/proc/irq/*/;
for folder in $folders; do
    files="$folder/*";
    for file in $files; do
        if [[ $file =~ *"name"* ]]; then
            echo $file;
            contents='cat $folder/affinity_hint';
            echo $contents > $folder/smp_affinity;
            cat $folder/smp_affinity;
        fi
    done
done

for CPUPRIME in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do
    [ -f $CPUPRIME ] || continue
    echo -n performance > $CPUPRIME
    done

cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor >> sysUpdateOut.txt
```
OVERPROVISION NVME SSD FOR BETTER ENDURANCE, PERFORMANCE, AND QoS
Basics of NAND Endurance

- NAND P/E Cycles = amount of program / erase cycles NAND can do before wearing out
- WAF = NAND writes / host writes
- TBW or PBW – amount of host writes to SSD before wearing out
  - TBW = drive capacity * cycles / WAF
- DWPD (drive writes per day): amount of data you can write to device each day of the warranty (typically 5 years) without wearing out
  - DWPD = TBW/365/5/drive capacity
Methods of overprovisioning NVMe SSD

First format or sanitize SSD to TRIM all LBAs

1. Create partition smaller than total user capacity of SSD
2. Limit max LBA of drive through vendor specific tools
3. Create a namespace smaller than total user capacity
4. Limit application use to smaller LBA range
Example with Intel® SSD DC P4510 Series 3.2TB
STORAGE ANALYTICS
Optimize platform by identifying storage-bound workloads

Where average data center workloads spends their time:

<table>
<thead>
<tr>
<th>CPU IS BUSY</th>
<th>CPU IS NOT BUSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Running the application</td>
<td>Idle Limited application parallelism</td>
</tr>
<tr>
<td>Sys Running the OS/VMM</td>
<td>IOwait Waiting for Disk</td>
</tr>
<tr>
<td>IOwait Waiting for Disk</td>
<td>Idle Imbalances</td>
</tr>
<tr>
<td>Idle Indirectly waiting for network</td>
<td>Idle Waiting for network</td>
</tr>
<tr>
<td>Idle (hidden IOwaits)</td>
<td>Idle Indirectly waiting for network</td>
</tr>
</tbody>
</table>

- Running the application
- Running the OS/VMM
- Waiting for Disk
- Limited application parallelism
- Imbalances
- Waiting for network
- Indirectly waiting for network (hidden IOwaits)
Storage Performance Snapshot

https://software.intel.com/sites/products/snapshots/storage-snapshot/
Dig Deeper – Where am I Bottlenecked?

Which disks contribute to bottlenecks and when.
JM doing Power Point and Email...not IO Bound
Appendix

3. System configuration: Intel® Server Board S2600WFT family, Intel® Xeon® 8170 Series Processors, 26cores@ 2.1GHz, RAM 192GB , BIOS Release 06/26/2018, BIOS Version: SE5C620.86B.00.01.0014.070920180847
OS: RedHat* Linux 7.4, kernel- 3.10.0-693.33.1.el7.x86_64, mdadm - v4.0 - 2018-01-26 Intel build: RSTe_5.4_WW4.5, Intel ® VROC Pre-OS version 5.4.0.1039, 4x Intel® SSD DC P4800X Series 375GB drive firmware: E2010423, Retimer
BIOS setting: Hyper-threading enabled, Package C-state set to C6(non retention state) and Processor C6 set to enabled, P-states set to default and SpeedStep and Turbo are enabled
Workload Generator: FIO 3.6, RANDOM: Workers- 8, Iodepth- 256, No Filesystem, CPU Affinitized

Pass-Thru Baseline: 1x Intel® SSD DC P4800X Series, 375 GB, Firmware: E2010423, SSDPE21K375GA)
Performance results are based on testing as of September 12, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

4. System configuration: Intel® Server Board S2600WFT family, Intel® Xeon® 8170 Series Processors, 26cores@ 2.1GHz, RAM 192GB , BIOS Release 06/26/2018, BIOS Version: SE5C620.86B.00.01.0014.070920180847
OS: RedHat* Linux 7.4, kernel- 3.10.0-693.33.1.el7.x86_64, mdadm - v4.0 - 2018-01-26 Intel build: RSTe_5.4_WW4.5, Intel ® VROC Pre-OS version 5.4.0.1039, 4x Intel® SSD DC P4800X Series 375GB drive firmware: E2010423, Retimer
BIOS setting: Hyper-threading enabled, Package C-state set to C6(non retention state) and Processor C6 set to enabled, P-states set to default and SpeedStep and Turbo are enabled
Workload Generator: FIO 3.6, SEQUENTIAL: Workers- 1, Iodepth- 128, No Filesystem, CPU Affinitized
Pass-Thru Baseline: 1x Intel® SSD DC P4800X Series, 375 GB, Firmware: E2010423, SSDPE21K375GA)
Performance results are based on testing as of September 12, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.