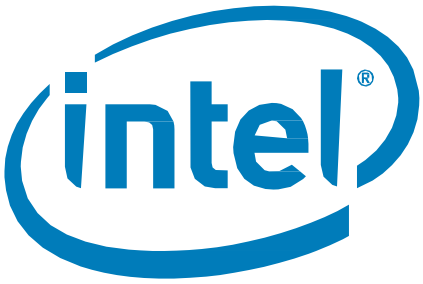


Intel® Optane™ Solid State Drive DC P4800X (Linux)

Performance Evaluation Guide

June 2020

Order Number: 335647-009US

Ordering Information

Contact your local Intel sales representative for ordering information.

Revision History

Revision Number	Description	Revision Date
001	• Initial release	March 2017
002	• Revised for re-testing of specifications and new kernels • Added new sections.	July 2017
003	• Added SPDK and Linux paging content	August 2017
004	• Updated Section 4. • Moved instructions for Creating Swap Area from original Appendix B, to main document. • Appendix C became new Appendix B. • Reorganized Exhibits section & System Tuning section	October 2017
005	• Update data section on comparing Intel® Memory Drive Technology to Linux Swap – graph only, not complete in description	April 2018
006	• Provides clearer positioning on the advantages of choosing Intel® Memory Drive Technology versus Linux Swap	June 2018
007	• Added Appendix C: Updated Configuration and Performance	February 2019
008	• Updated Best Practices for Linux Kernel 4.20 and beyond (5.x)	February 2020
009	• Changed document classification from Intel Confidential to Public	June 2020

All information provided here is subject to change without notice.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

For copies of this document, documents that are referenced within, or other Intel literature please contact you Intel representative.

Intel does not control or audit third-party data. You should consult other sources to verify accuracy.

Results have been estimated or simulated.

© 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.

Contents

Terms and Acronyms

Term	Definition
AHCI	Advanced Host Controller Interface
API	Application Programming Interface
ATA	Advanced Technology Attachment
DIPM	Device Initiated Power Management
GB	Gigabyte
HDD	Hard Disk Drive
KB	Kilobytes
I/O	Input/Output (the typical block used in specifications is 4 kB)
IOPS	Input/Output Operations Per Second
MB	Megabytes
NCQ	Native Command Queuing
PCH	Platform Controller Hub
RAID	Redundant Array of Independent Disks
SATA	Serial Advanced Technology Attachment
SSD	Solid State Drive

Sources

Additional information for topics discussed in this guide:

- https://en.wikipedia.org/wiki/Kernel_(operating_system)
- http://www.linfo.org/kernel.html
- http://www.ntop.org/pf_ring/irq-balancing/hybrid
- https://en.wikipedia.org/wiki/I/O_scheduling#/media/File:IO_stack_of_the_Linux_kernel.svg
- https://en.wikipedia.org/wiki/Scheduling_(computing)
- https://en.wikipedia.org/wiki/Udev
- https://github.com/axboe/fio
- http://vldb2010.org/proceedings/files/vldb_2010_workshop/ADMS_2010/adms10-foong.pdf
- http://ieeexplore.ieee.org/document/7495289/
- https://www.usenix.org/conference/fast12/when-poll-better-interrupt
- https://www.kernel.org/doc/Documentation/IRQ-affinity.txt

1 Introduction

This guide outlines the best know practices and configurations for evaluating Intel® Optane™ Solid State Drive Data Center P4800X Series performance in a Linux environment.

1.1 Overview

Intel® Optane™ SSD DC P4800X features Intel® Optane™ memory media. Intel® Optane™ memory media is a class of memory technology that does not store data by trapping electrons in the memory cell, as NAND does; instead it utilizes the property-change of the memory material itself, to store the data. Intel® Optane™ memory media, coupled with Intel-developed controller and firmware, takes SSD performance to the next level. For example, just two direct-attached Intel® Optane™ SSDs deliver over 1 million I/O operations per second (IOPS). In addition, this performance is achieved at very low latencies. For example, at queue depth of one, 99 out of 100 read operations of a 4KB-sized-aligned-workload, complete in single digit microseconds. With the reduction in latencies, Intel® Optane™ SSD performance becomes sensitive to some of the variables that NAND SSDs are indifferent to. These variables include CPU speed, number of sockets, and relevant hardware and OS settings. This guide explores those variables in detail, enabling users to evaluate optimum SSD performance.

1.2 Typical I/O Execution

Let us first take a high-level look at a typical I/O access cycle. When an application requests data for processing, the kernel -- the core of the operating system, which controls system hardware -- checks for that data in main memory. If the required data does not exist in main memory, physical I/O is initiated through the device driver to the given device. The device processes the command and sends an interrupt at the completion of the command. The device driver acknowledges this interrupt command and the cycle ends.

Figure 1 shows the ways that a PCIe SSD can be installed in a system. Essentially, it is key to understand that-end-to-end latency depends on many variables. For example, a faster CPU would allow the kernel to process and direct I/O requests faster. Similarly, less context-switching and better interrupt handling improves overall latency. Data transfer through various buses will contribute towards total latency. That is, if a PCIe SSD is attached through PCH, then the DMI bus would also incur latency. Finally, in multi-socket Intel server systems (Figure 2), if a process transitions from one socket to another, it uses QPI bus for data transfer which also adds to latency in the hundreds of nanoseconds. This evaluation guide will show you how to tune for latency improvement.

Figure 1: Available slots for NVMe SSDs

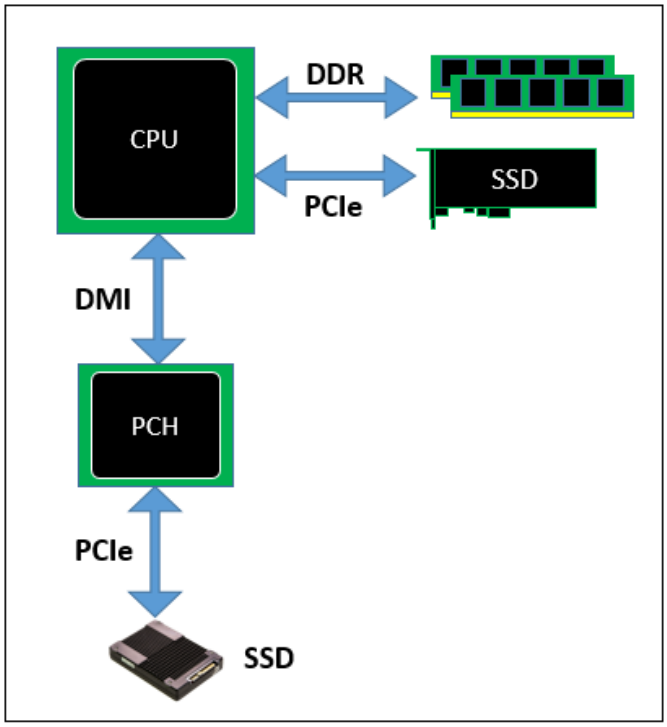


Figure 2: Quick Path Interconnect

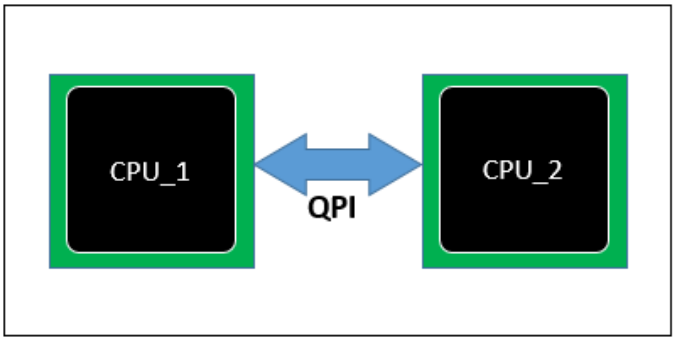
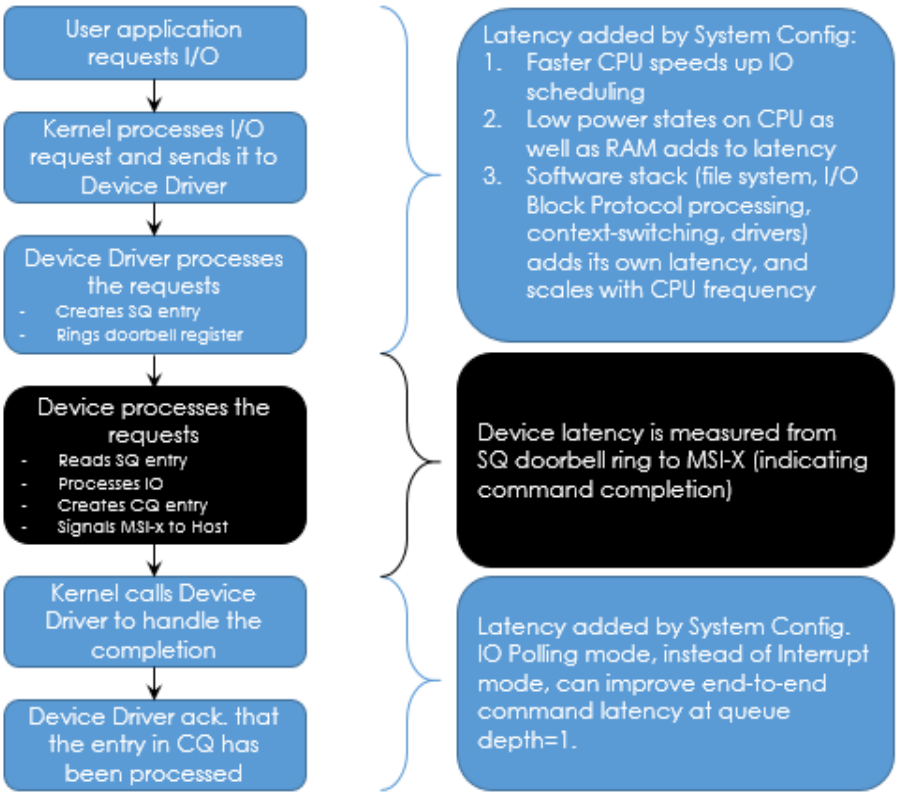


Figure 3 shows a typical command execution flow of a PCIe/NVMe SSD. Each stage incurs latency. End-to-end command latency indicates the total latency from I/O request to command completion acknowledgement. For this evaluation guide, we will divide latencies into two broad categories: SSD Latency and System Latency.

Figure 3: Typical Linux flow of command execution on NVMe SSDs



SSD Latency: SSD latency is measured from the Submission Queue Tail Doorbell pointer update (submission command) to the MSI-X completion interrupt generated by the SSD (update of the Completion Queue Tail Doorbell pointer, indicating command completion). The Intel SSD controller has optimized this latency for 4KB aligned workloads through hardware acceleration. Note that sub-4KB or unaligned workloads, as well as any type of error correction, or any demands on firmware interception will increase SSD latency, when the SSD ramps the IO from the PCIe bus and to and from the submission and completions queues.

System Latency: This broad category includes end-to-end latency (from the originating user space application) for command completion, less SSD device latency, as captured above. This document is intended to guide you in tuning your system configuration to achieve low latency and properly evaluate SSD performance.

§

2 System Tuning for Optimum Performance

The Intel® Optane™ SSD DC P4800X Series is designed for a PCIe Gen 3.0 x4 interface and NVMe 1.0c protocol. The tuning method shown in Table 1 focuses on Linux OS, but the BIOS or Intel components can be generalized to any general-purpose OS. That said, implementation of actual settings will most likely vary with the OS and Intel is working on producing more OS-specific guides.

To benchmark Intel® Optane™ SSD DC P4800X Series against its specification, use a system with a CPU frequency of 3.0GHz or higher. We actually lock the CPU frequency to maintain the best specification matching. This does not mean Turbo is not good for Intel® Optane™ SSDs, but to get the expected best quality of service, it is recommended to set up your configuration as follows:

Table 1: Best Known System Configuration Summary

Setting	Status
Hyper threading	Disabled
Enhanced Intel Speed Step Technology	Disabled
Intel Turbo Mode	Disabled
C-States	Disabled
P-States	Disabled
CPU Governor (through the OS)	Performance mode
IRQ Balancing Service (OS Service)	Disabled
SMP Affinity (via bash script)	Set interrupts to run in the proper CPU cores
I/O Polling or Hybrid Mode	Enabled (if supported by your kernel) **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.

Note: Configuration is based on Intel BIOS version SE5C610.86B.01.01.0019. C-States and P-States are disabled through the BIOS.

Best Known Configuration Details

Place the NVMe-based Intel® Optane™ SSD in a PCIe slot that is directly connected to the CPU, and not the platform controller hub (PCH), as PCH would incur additional latency.

1. Turn off CPU hyper-threading, disable C-states and any power saving modes in BIOS.
2. Turn off Turbo mode.

   Turbo mode improves average latency and IOPS, especially at a lower queue depth. However, data indicates negative impact of Turbo mode on quality of service. It is recommended to explore both options and choose the one that best fits your system/configuration.

3. Set CPU Governor to "Performance"

   This ensures that the CPU operates at the highest frequency, which is necessary to achieve low end-to-end latencies, there are multiple ways to accomplish this.

   Option 1: Execute prior to performance evaluation.

   Note: This setting is not retained on reboot.

   - Set governor to performance:

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

   - Ensure the drive is running in performance mode:

     ```
     # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
     ```

   - Change `cpu0` to "`cpuX`", where X = any CPU core number.

   Option 2: Update the `ondemand` file**.**

   Note: This setting is not retained on reboot.

   - In */etc/init.d/ondemand*, comment out all cases of: `GOVERNOR="xxxxx"` .

     See Exhibit-1: Change CPU Governor to Performance Mode for details.

   - Reboot the system.
   - Ensure the drive is running in performance mode:

     ```
     # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
     ```

   - Change `cpu0` to "`cpuX`", where X = any CPU core number.

   Option 3: Utilize `cpufrequtils`.

   See Exhibit-1: Change CPU Governor to Performance Mode for details.

4. Disable the IRQ Balancing Service.

   IRQ balancer is responsible for balancing interrupts across cores. Disabling this service gives more control over managing interrupts across cores.

   Note: This setting is not retained on reboot.

   - Change `'Enabled'` to `"0"` in `/etc/default/irqbalance` to turn off the irq balancer.
   - Stop IRQ service:

     ```
     # systemctl stop irqbalance
     ```

     Or disable the service with this command:

     ```
     # systemctl disable --now irqbalance
     ```

   See Exhibit-2: Disable IRQ Balance on older than 4.8 kernels for details.

   - Ensure the irqbalance status is enabled:

     ```
     # systemctl status irqbalance
     ```

5. Set the SMP Affinity Values

   Note: On kernels 4.8 and newer this is automatic, and unnecessary)

   SMP Affinity associates certain tasks/interrupts to certain cores.

   Execute the following shell script:

   Note: This setting is not retained on reboot.

```
#!/bin/bash
folders=/proc/irq/*;

for folder in $folders; do
   files="$folder/*";
   for file in $files; do
      if [[ $file == *"nvme" ]]; then
         echo $file;
         contents=`cat $folder/affinity_hint`;
         echo $contents > $folder/smp_affinity;
         cat $folder/smp_affinity;
      fi
   done
done
```

6. 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 Intel® Optane™ technology.

     Enable polling mode [Linux Kernel 4.4 and above]:

     ```
     # 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]:

     ```
     # echo 0 > /sys/block/<dev>/queue/io_poll_delay
     ```

The following table shows the performance comparison of three modes.

Table 2: Performance Comparison for Polling, Hybrid and Interrupt Modes (4.10.8 kernel)

Mode	4KB Random Read, Queue Depth = 1				
	Latency (µs)				IOPS
	Avg	99%	99.999%	CPU util	
I/O Polling	7.34	16	40	99.9%	132,000
Hybrid	7.54	15	41	58.4%	127,000
Interrupt	11.02	11	40	38.5%	86,000

7. Run fio (Open Source) tool.

   For this evaluation guide, we used FIO (github: axboe/fio) to generate synthetic I/O traffic. It is recommended to use FIO version 2.18 or higher as it supports pvsync2 ioengine, which improves I/O performance by up to 20% at queue depth 1.

The following table shows the performance comparison between libaio and pvsync2.

Table 3: QD1 Performance Comparison for ioengine and pvsync2

Ioengine Option (with "Turbo off")	4KB Random Read, Queue Depth =1				
	Latency (µs)				IOPS
	Avg	99%	99.999%	99.99999%	
libaio	8.97	11	45	161	94.3k
pvsync2 w/ hipri	7.57	9	53	135	126k

It is important to balance the number of threads versus the number of outstanding I/Os, as well as the pinning of threads to cores, to take best advantage of system resources and avoid accidentally limiting performance.

This can be done by increasing by adding threads at I/O depth 1. Once the number of threads reaches 8 (depending on the number of CPUs in the test system), double the I/O depth to increase the load until the desired queue depth is achieved. The following table shows how to increase the queue depth.

Table 4: Queue Depths

# of Threads (numjobs)	1	2	4	8	8	8	8	8
Depth/thread (iodepth)	1	1	1	1	2	4	8	16
Total I/O Depth	1	2	4	8	16	32	64	128

§

3 Results

Table 5: System Configuration

Type	Configuration
CPU	Intel® Xeon® E5-2687W v4 @ 3.0 GHz (12 cores x 2)
Motherboard	Intel® S2600WT
Memory	32GB DDR4 @ 2133Mhz (32G X 1 DIMM)
BIOS version	SE5C610.86B.01.01.0019
BIOS configuration	Hyper-threading disabled, CPU C-state disabled, P-states disabled
Linux version	Centos- 7.3
Kernel version	4.10.8
FIO rev	2.18

Table 6: Performance Table

Settings	4KB Random Read, Queue Depth =1[1]					4KB Random Write, Queue Depth =16				
	Latency (µs)				IOPS	Latency (µs)				IOPS
	Avg	99%	99.999%	Max		Avg	99%	99.999%	99.99999%	
Stock OS Results	8.43	17	45	139	115k	25.37	65	334	4,576	550k
Governor set to Performance	8.16	16	45	77	119k	25.61	64	119	1,192	555k
IRQ Balancing Service Turned off	8.01	16	39	69	120k	25.69	65	237	2,416	554k
SMP Affinity Set	7.96	15	44	141	122k	25.81	64	123	221	553k
Turbo off 2.3 GHz	9.2	11	36	87	107K	24.44	64	255	2,576	556K
Turbo off 3 GHz	7.57	9	53	135	126k	24.82	63	129	1,416	556k

Notes:

Max values can be affected by system services (example: udevd system service) and are not deterministic or statistically significant based on configuration variances. Therefore 99,99999% data is shown.

1. QD1 performance uses ioengine=pvsync2 w/ hipri, kernel 4.8.6

Performance measured by Intel using FIO rev 2.18, with 4 workers and total Queue Depth of 16 in most case, except where specified. System configuration: Intel® Xeon® CPU E5-2687W v4 @ 3.00GHz, Intel® R2208WT2YS-IDD Server Board, 32GB DRAM, Hyper-threading and CPU C-states disabled, P states disabled, CentOS 64bit Server, Linux Kernel 4.8 with polling enabled.

§

4 Best Practices for Linux Kernels 4.20 and 5.x

In the case of your test or production environment moving to most recent kernel, you may see performance differences compare with the product specification. Here we provide best practices for your reference.

1. Put your CPU's in performance mode

   ```
   # echo "performance" | tee
   /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
   ```

   Ensure the CPU scaling governor is in performance mode by checking the following; here you will see the setting from each processor (vcpu).

   ```
   # cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
   ```

   You should see performance as the return of this command.
   We recommend you make this setting persistent between reboots by changing your Linux restart configuration (i.e. rc).

2. Disable IRQ balance (only in kernel versions prior to Linux 4.8)

   In kernels before version 4.8 the irq balancing was not managed as efficiently as in the current in-box Linux nvme driver. So, if you are on a kernel version older than 4.8, please turn off the irqbalance service and run a short script (below) to balance your irq's, this will allow for the best IO processing possible.

   You can stop and disable the service with the following command on CentOS and Ubuntu:

   ```
   #! /bin/bash
   folders=/proc/irq/*;
   for folder in $folders; do
   files="$folder/*";
   for file in $files; do
   if [[ $file == *"nvme"* ]]; then
   echo $file;
   contents=`cat $folder/affinity_hint`;
   echo $contents > $folder/smp_affinity;
   cat $folder/smp_affinity;
   fi
   done
   done
   ```

3. Enable polling or poll queues in your Linux in-box NVMe driver

   Since Linux 4.20 there have been optimizations to the NVMe driver to allow for a new parameter that governs polling. Polling should not involve interrupts of any kind, and NVMe driver developers needed to make changes to allow for this improvement. This brought the advent of poll queues which are now available in 4.20 and later.

   To enable NVMe to run with poll queues, load the driver with the IO polling enabled. You might want to setup poll queues equal to the number of virtual cores in your system, or decide based on your implementation goals, there is no simple answer here.

   Example here:
   To enable polling mode by device (before kernel 4.20) :

   ```
   # echo 1 > /sys/block/nvme0n1/queue/io_poll
   ```

   To enable poll_queues in the NVMe driver system wide (kernel 4.20) and later:

   ```
   # modprobe -r nvme && modprobe nvme poll_queues=4
   ```

The above is for a quick system test. If you want to enable it on boot, or if you are booting from an NVMe drive do not use the method above; instead use the following method.

```
# more /etc/modprobe.d/nvme.conf
options nvme poll_queues=4
```

Next, rebuild initramfs so the parameter in the module parameter is picked up. Using these commands, first backup the image, and then rebuild it.

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).img.$(date +%m-
%d-%H%M%S).bak
# dracut -force
```

Now reboot your system and check your work.

```
# systool -vm nvme
You should set that poll_queues is set to your desired setting, and no longer 0.
poll_queues         = "4"
```

You can also setup Grub2 to add this kernel option by following instruction in the CentOS wiki for CentOS 7. https://wiki.centos.org/HowTos/Grub2

4. Choose appropriate fio ioengine, and I/O polling mode

Next, check the configuration of the system you built. The most critical performance will show itself at QD1 (queue depth 1) with just 1 worker thread. You can run this with any number of ioengines, but we recommend pvsync2 or io_uring in hipri mode. Here are the requirements:

- Polling mode per device requires Linux kernel 4.8 or newer.
- Poll queues require Linux kernel 4.20 or newer.
- io_uring requires Linux kernel 5.1 or newer.

If you are new to developing with io_uring, you should move to the most stable Linux 5.x kernel available when you do this.
Below is a recommended fio script:

```
[global]
name= OptaneFirstTest
ioengine=pvsync2
hipri
direct=1
size=100%
randrepeat=0
time_based
ramp_time=0
norandommap
refill_buffers
log_avg_msec=1000
log_max_value=1
group_reporting
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:99
.999999:100.0
filename=/dev/nvme0n1
[rd_rnd_qd_1_4k_1w]
bs=4k
iodepth=1
numjobs=1
rw=randread
cpus_allowed=0-17 # dependent on your NUMA goals
runtime=300
write_bw_log=bw_rd_rnd_qd_1_4k_1w
write_iops_log=iops_rd_rnd_qd_1_4k_1w
write_lat_log=lat_rd_rnd_qd_1_4k_1w
```

We use cpus_allowed for numa locality here and for no other reason. Based on the server I ran this on, this job will burn just one core. You may also need to compile fio a specific way to get the particular ioengines that you desire, in fact around here at Intel Storage land, there are custom fio's all over the place.

Results from a build with Intel Xeon Gold 6254 CPUs and Linux 5.4.1-1 kernel

Summary output from fio on test system:

```
rd_rnd_qd_1_4k_1w: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T)
4096B-4096B, ioengine=pvsync2, iodepth=1
fio-3.16-64-gfd988
Starting 1 process
Jobs: 1 (f=1): [r(1)][100.0%][r=487MiB/s][r=125k IOPS][eta 00m:00s]
rd_rnd_qd_1_4k_1w: (groupid=0, jobs=1): err= 0: pid=3036: Wed Jan 15 14:00:45 2020
  read: IOPS=125k, BW=487MiB/s (511MB/s)(143GiB/300001msec)
    clat (usec): min=7, max=202, avg= 7.76, stdev= 1.29
     lat (usec): min=7, max=202, avg= 7.78, stdev= 1.29
    clat percentiles (usec):
     |  1.000000th=[    8], 25.000000th=[    8], 50.000000th=[    8],
     | 75.000000th=[    8], 90.000000th=[    8], 99.000000th=[   10],
     | 99.900000th=[   33], 99.990000th=[   38], 99.999000th=[  106],
     | 99.999900th=[  159], 99.999990th=[  167], 99.999999th=[  204],
     | 100.000000th=[  204]
   bw (  KiB/s): min=498144, max=500708, per=100.00%, avg=499144.54, stdev=489.10,
samples=299
   iops        : min=124536, max=125177, avg=124786.12, stdev=122.32, samples=299
  lat (usec)   : 10=99.27%, 20=0.55%, 50=0.17%, 100=0.01%, 250=0.01%
  cpu          : usr=5.80%, sys=94.07%, ctx=1018, majf=0, minf=23
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=37437069,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=487MiB/s (511MB/s), 487MiB/s-487MiB/s (511MB/s-511MB/s), io=143GiB
(153GB), run=300001-300001msec
Disk stats (read/write):
  nvme0n1: ios=37424564/0, merge=0/0, ticks=261212/0, in_queue=0, util=99.99%
```

Note: All tests in this section were done on a CentOS 8.0 distribution using kernel 5.4.1-1.el8.elrepo.x86_64 and fio-3.16-64-gfd988. Your hardware and software configuration may not be compatible with everything you read in this section.

§

5 Exhibits

5.1 Exhibit-1: Change CPU Governor to Performance Mode

The bash script example below can be placed in */etc/rc.local*, and anything in that folder will be executed at Linux startup. In Ubuntu, you can change the variable for GOVERNOR in */etc/default/cpufrequtils*, assuming you have installed the cpufrequtils package.

Install `cpufrequtils` and follow the steps.

1. `# cpupower frequency-set -g performance`
2. Reboot the system.
3. Ensure the drive is performance mode by checking the following:

   `# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor`

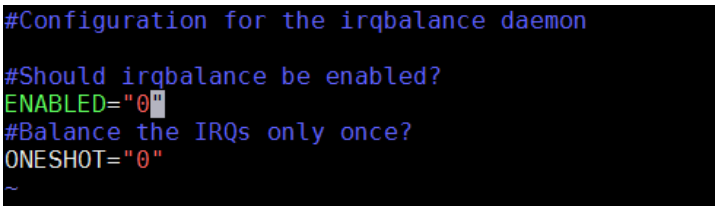
Verify if this setting is retained after reboots (depending on the OS version). The above "cat" command shown above will show you what governor your CPUs are running on.

5.2 Exhibit-2: Disable IRQ Balance on older than 4.8 kernels

Set '`Enabled`' to "0" in */etc/default/irqbalance* on Ubuntu. As shown here, you can disable the service with the following command on CentOS:

```
# sudo systemctl disable –now irqbalance
```

Figure 4: Disable irqbalance Confirmation



In CentOS, you can edit */etc/sysconfig/irqbalance* file to disable IRQ balance. Make sure you have irqbalance-1.0.7-5.el7.x86_64 rpm package installed.

5.3 Exhibit-3: FIO Script Examples

Intel uses the following fio script to test Random IO. The most critical performance will show itself at QD1 with 1 worker thread as shown here (see Table 2 for performance target):

```
[global]
name= Schedule_CS_prd2.csv
ioengine=pvsync2
hipri
direct=1
buffered=0
size=100%
randrepeat=0
time_based
ramp_time=0
norandommap
refill_buffers
log_avg_msec=1000
log_max_value=1
group_reporting
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:99.999
999:100.0
filename=/dev/nvme0n1

[rd_rnd_qd_1_4k_1w]
stonewall
bs=4k
iodepth=1
numjobs=1
rw=randread
runtime=300
write_bw_log=bw_rd_rnd_qd_1_4k_1w
write_iops_log=iops_rd_rnd_qd_1_4k_1w
write_lat_log=lat_rd_rnd_qd_1_4k_1w
Here is an example of 16 QD job:
[global]
name= Schedule_CS_prd2.csv
ioengine=libaio
direct=1
buffered=0
size=100%
randrepeat=0
time_based
ramp_time=0
norandommap
refill_buffers
log_avg_msec=1000
log_max_value=1
group_reporting
#pwravg=300
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999
:99.999999:100.0
filename=/dev/nvme0n1

[wr_qd_16_4k_4w]
stonewall
bs=4k
iodepth=4
numjobs=4
rw=write
runtime=300
write_bw_log=bw_wr_qd_16_4k_4w
write_iops_log=iops_wr_qd_16_4k_4w
write_lat_log=lat_wr_qd_16_4k_4w
```

5.4 Exhibit-4: Pre-conditioning

NAND devices are required to be written fully before getting stable performance. Background Data Refresh is a "periodic refresh" of the data stored on the drive to prevent the cell voltage levels from shifting outside an acceptable range. When the drive detects that the data stored on it has reached a certain age, it will rewrite that data in the background.

Intel® Optane™ memory media is a completely new technology. Intel® Optane™ memory media also need periodic refreshing. Intel® Optane™ SSDs have a background refresh policy enabling the data in the media to remain refreshed even when it is not accessed.

If an Intel® Optane™ SSD DC P4800X remains de-energized for an extended period of time, faster background data refresh will be invoked by firmware design, impacting performance.

The length and temperature of the de-energized state will impact the likelihood of triggering faster background data refresh (currently set at ~3 hours).

With this, Intel recommends minimum power-on time of 3 hours before running performance evaluations.

§

6 Storage Performance Development Kit (SPDK)

The Storage Performance Development Kit (SPDK) is a set of libraries and polled-mode drivers (PMDs) for building high performance storage applications. These PMDs operate in userspace, bypassing the kernel to avoid software overhead associated with interrupts, system calls and kernel locking.

Utilizing the SPDK NVMe driver with Intel® Optane™ SSD DC P4800X enables a number of performance advantages compared to the Linux kernel NVMe driver. For queue depth 1 workloads, SPDK enables up to 15% better performance and latency. SPDK can also maintain this latency while scaling performance with higher queue depths on a single worker thread – up to full SSD bandwidth as low as queue depth 5. These advantages enable storage applications to keep a smaller queue of outstanding I/O at the SSD, allowing for better prioritization to meet QoS requirements.

6.1 SPDK Configuration

6.1.1 Downloading SPDK

SPDK has implemented a number of driver optimizations and benchmarking since the SPDK v17.03 release. Until SPDK v17.07 is released, git commit ID e6d05312 should be used for benchmarking:

```
git clone https://github.com/spdk/spdk.git
cd spdk
git checkout e6d05312
```

Please refer to the README in the root SPDK directory for package dependencies that must be installed prior to building SPDK.

6.1.2 Downloading fio

SPDK provides a plugin for fio, the popular benchmarking utility. Building the SPDK fio plugin requires fio source code.

```
git clone https://github.com/axboe/fio
cd fio
git checkout fio-2.21
make
```

6.1.3 Building SPDK

SPDK can be built after installing the SPDK package prerequisites, and downloading and building the fio source code.

```
cd <path/to/spdk>
./configure --with-fio=<path/to/fio>
make
```

SPDK also requires binding NVMe SSDs to UIO or VFIO and allocating huge page memory. SPDK provides a script to automate this process. This script needs to be re-run once after a reboot, but does not need to be re-run between tests.

```
cd <path/to/spdk>
scripts/setup.sh
```

To switch back to testing with the kernel NVMe driver, use the same script with the “reset” option.

```
cd <path/to/spdk>
scripts/setup.sh reset
```

6.2 System Configuration

For benchmarking SPDK with Intel® Optane™ SSD DC P4800X, please start with the system tuning guidelines described in this guide under *System Tuning for Optimum Performance*, with the following adjustments:

1) Ignore the section related to SMP IRQ affinity. SPDK does not use NVMe interrupts, so these settings are not required.
2) Ignore the section related to NVMe kernel driver I/O polling mode. The SPDK NVMe PMD takes direct control of the NVMe device, completely bypassing the NVMe kernel driver, so these settings are not required.
3) Run the prep_benchmarks.sh script found in the SPDK scripts subdirectory. This script will place CPUs in performance mode and disable the irqbalance service, as recommended in Section 2 *System Tuning for Optimum Performance*, steps 3 and 4. It will also ensure no interrupts are bound to CPU core 0, so that the SPDK benchmarking running on core 0 is not affected by unrelated device interrupts.

6.3 Collecting Results with SPDK

SPDK provides two methods for collecting benchmark results: an FIO plugin and a custom SPDK benchmarking app called “perf”.

6.3.1 FIO

SPDK can be used with fio by preloading the fio plugin using LD_PRELOAD and specifying the PCI address (bus, device, function) and NVMe namespace via the filename parameter. Additional details can be found in the SPDK fio documentation at examples/nvme/fio_plugin/README.md.

6.3.1.1 Create SPDK FIO Configuration File

This section contains a sample fio configuration file for benchmarking an NVMe SSD with SPDK NVMe polled mode driver. This example is for 4KB random reads at queue depth 1 for 5 minutes. Note the traddr field in the filename attribute. This example describes an NVMe SSD at PCI bus 4, device 0, function 0. This field must be updated to match the PCI address of the Intel® Optane™ SSD DC P4800X in the system under test.

```
[global]
ioengine=spdk
thread=1
group_reporting=1
direct=1
verify=0
norandommap=1
percentile_list=50:90:99:99.9:99.99:99.999:99.99999
bs=4096
rw=randread
iodepth=1
time_based=1
runtime=300s
cpus_allowed=0

[perf_test]
filename=trtype=PCIe traddr=0000.04.00.0 ns=1
```

6.3.1.2 Run fio with SPDK

FIO can collect results using the SPDK NVMe PMD with the following example commands. These commands assume that spdk has been cloned into the /home/user/spdk directory, FIO has been cloned into the /home/user/fio directory, and the SPDK fio configuration file has been written to /home/user/spdk/fio.conf

```
cd /home/user/spdk
LD_PRELOAD=examples/nvme/fio_plugin/fio_plugin /home/user/fio/fio fio.conf
```

6.3.2 SPDK perf

SPDK also provides a custom performance benchmarking tool called “perf”. perf can provide very detailed sub-microsecond latency histograms and is a bit simpler to get running. perf also adds less CPU overhead per I/O but this is typically only noticeable for high queue depth workloads with multiple NVMe SSDs.

6.3.2.1 Run perf with SPDK

Perf can collect results using the SPDK NVMe PMD with the following examples commands. These commands assume that SPDK has been cloned into the /home/user/spdk directory. This example is for a 4KB random read workload at queue depth 1 for 5 minutes. It will generate IOPs data, a latency summary and a detailed latency histogram. By default, perf will collect performance data for any NVMe SSD bound to the uio or vfio drivers, so a PCIe address is not required.

```
cd /home/user/spdk
examples/nvme/perf/perf –s 4096 –w randread –q 1 –t 300 –LL
```

6.4 Performance Measurements

SPDK’s architecture enables performance and efficiency improvements with the Intel® Optane™ SSD DC P4800X compared to the Linux kernel benchmarks described earlier in this document. For 4KB random read workloads, the peak latency and IOPs data points are limited by the queue depth 1 limitation for high priority I/O with the pvsync2 ioengine. SPDK results in lower average and 99% tail latency at queue depth 1, but is also able to sustain this average latency while scaling IOPs at higher queue depth with a single worker thread.

Table 7: SPDK 4K Random Read Performance with "ioengine" Option

System optimized with ioengine option	4KB Random Read					
	Latency (µs)					IOPS
	Queue Depth	Avg	99%	99.999%	99.99999%	
pvsync2 w/ hipri	1	7.57	9	53	135	126k
spdk	1	6.41	8	40	57	149k
spdk	2	6.55	14	43	59	292k
spdk	3	6.68	16	45	61	430k
spdk	4	7.16	17	47	61	536k
spdk	5	8.42	19	48	62	573k

SPDK realizes similar benefits for 4KB random write workloads at queue depth 16. SPDK maximizes the performance of the Intel® Optane™ SSD DC P4800X with a single worker thread and can maintain this performance at a much lower queue depth with a significant improvement in average latency.

Table 8: SPDK 4K Random Write Performance with "ioengine" Option

System Optimized with ioengine Option	4KB Random Write					
	Latency (µs)					IOPS
	Queue Depth	Avg	99%	99.999%	Max	
libaio	16	24.78	62	135	2,192	556k
spdk	16	28.39	66	125	163	558k
spdk	12	21.21	58	111	138	558k
spdk	8	14.16	52	95	119	553k
spdk	6	11.21	50	91	117	520k
spdk	4	9.78	47	88	112	396k

Table 9: System Configuration

System Configuration	
CPU	Intel® Xeon® E5-2695 v4 @ 2.1 GHz (18 cores x 2)
Motherboard	Supermicro X10DRi
Memory	32GB DDR4 @ 2133Mhz (4G X 8 DIMM)
BIOS configuration	Hyper-threading disabled, CPU C-state disabled, P states disabled
Linux version	Ubuntu 17.04
Kernel version	4.10.0-19-generic
SPDK commit ID	e6d053125b6b
FIO rev	2.21
Data Collection Date	March 2017

§

7 Intel® Memory Drive Technology and Using Memory Extension on the Linux Operating System

Using Intel® Memory Drive Technology, you can make Intel® Optane™ SSD DC P4800X an extension of main memory. With Intel® Memory Drive Technology, which is an advanced software-defined memory product, the Intel® Optane™ SSDs are booted into memory mode and transparently become a part of the operating system's virtual memory system, and this solution is fully supported by Intel. A lower-performance alternative can be implemented using Linux swap, and the necessary steps to properly configure swap on Intel® Optane™ SSDs are also included in this guide. The full documentation for Intel® Memory Drive Technology is available on the Intel® Memory Drive Technology landing page, including many solutions for common workloads, that have been tested for Linux kernels 2.6.32 (x64) and higher on both Red Hat/CentOS and SUSE Linux Enterprise Server.

Intel® Memory Drive Technology landing page:

https://www.intel.com/content/www/us/en/software/intel-memory-drive-technology.html

Please contact your Intel representative to gain more information about this product and to schedule an evaluation. Intel® Memory Drive Technology is the only supported technology that allows for OS memory virtualization using Intel® Optane™ SSDs.

If using the Linux swap

To improve swap performance, Intel recommends using **Kernel 4.11 or later**. Linux Kernel 4.11 introduced patches to the paging subsystem which enable paging to scale across multiple CPUs and swap devices, increasing the maximum virtual memory page access throughput from 250K pages/s to at least 1M pages/s. Note that with Linux swap, the variance in memory access time (latency) per page is significantly increased.

Information and instructions for installing a newer kernel on Ubuntu or RHEL are included below. For other Linux distributions, consult the distribution's documentation.

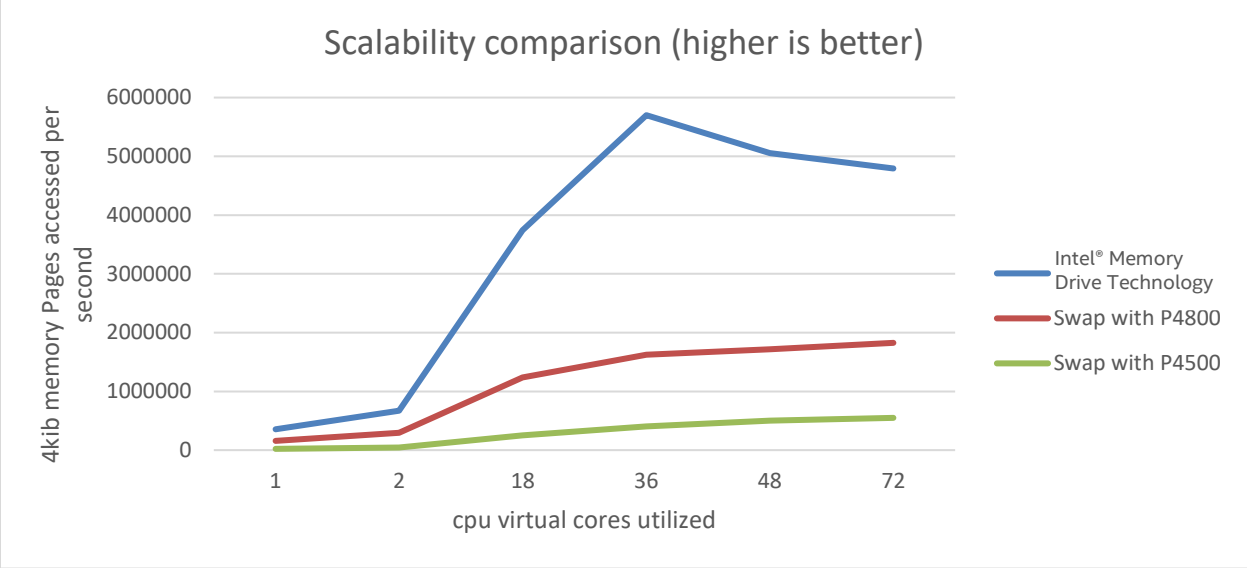
Ubuntu: https://wiki.ubuntu.com/Kernel/MainlineBuilds

RHEL: https://fedoraproject.org/wiki/Kernel_Vanilla_Repositories

7.1 Performance Comparison: Intel® Memory Drive Technology vs. Linux Swap

The following chart illustrates the improvement in virtual memory page access when using Intel® Memory Drive Technology versus Linux swap, using the PMbench benchmark. It is evident that Intel® Memory Drive Technology is approximately 3-4x faster for throughput and latency compared to Linux swap. PMbench is a micro-benchmark which stimulates virtual memory page access throughput by mapping and accessing a large virtual memory space. In this case, we used 4 Intel® Optane™ SSD DC P4800X for memory expansion, by: (1) using Intel® Memory Drive Technology (doubling the size of system memory compared to the installed DRAM); and (2) by mapping virtual memory space twice as large as the physical DRAM size as equal-priority swap devices. We mapped a total of 64GB of memory so that the benchmark can use double the available DRAM in the system. It is important to note that we used the same exact physical system for comparing Intel® Memory Drive Technology against Linux swap. Swap patches are not back-ported to main Linux distributions and hence lack vendor support, but we used them to show swap at its best. Note that while this swap configuration is not vendor supported, Intel® Memory Drive Technology is fully supported by Intel across a variety of distributions. The following graph compares the scalability between Intel® Memory Drive Technology and Linux swap.

Figure 5: Scalability of Page Access between Intel® Memory Drive Technology and Linux Swap, on Intel® Optane™ Memory, and NVMe NAND



Test and System Configuration: 2x Intel® Xeon® Gold 6154 CPU @ 3.00GHz (72 HT cores), 32GB DDR4 2666MHz DRAM, 4 Intel® Optane™ SSD DC P4800X as equal-priority swap devices, PMbench v0.7 with a 64GB mmap, 1 PMbench job per CPU core, running Kernel 4.14.18 with Intel® Memory Drive Technology, and Kernel 4.14.18

Implementation details: System BIOS: 00.01.0012 (https://downloadcenter.intel.com/download/27523/Intel-Server-Board-S2600WF-Family-BIOS-and-Firmware-Update-Package-for-UEFI-?product=89014); Kernel 4.14.18; Mitigation was validated for variants 1 through 3 for Spectre and Meltdown using a checker script (accessed and tested Apr'18).

7.2 Configuring Intel® Memory Drive Technology for Performance

With Intel® Memory Drive Technology the only OS configuration recommendation is that you ensure that Linux Transparent Huge Pages (THP) is enabled. THP is turned on by default in recent Linux distributions such as Redhat Enterprise Linux (RHEL) 7.x, and SUSE Linux Enterprise Server, but not in RHEL 6.x.

7.3 Configuring Linux Swap for Performance

The following instructions describe the best known method for configuring system HW and SW to enable scaling of Linux swap throughput across CPU cores and sockets:

1. It is highly recommended to have at least one Intel® Optane™ SSD DC P4800X for each CPU socket in the system, with all SSDs divided equally across all sockets. Consult your system manual for PCIe to socket mapping.

   For example, in a dual socket system, two 375GB DC P4800X drives attached to each CPU socket would provide better performance than a single 750GB DC P4800X drive attached to one of the sockets.

   Avoid connecting an Intel® Optane™ SSD DC P4800X to a PCIe slot associated with a PCH.

2. Determine the device name(s) of your SSD(s).

   ```
   lsblk --output NAME,MODEL,SIZE,REV,VENDOR
   ```

   The device name of an NVMe SSDs like the Intel® Optane™ SSD DC P4800X should be similar to `nvme0n1`. Use this name wherever the text `[devicename]` appears.

   Ensure that the block device(s) represents your Intel® Optane™ SSD DC P4800X(s) and not, for example, a data drive or your system boot drive.

3. For each Intel® Optane™ SSD DC P4800X:

   Note: These settings are not retained on reboot.

   Turn off all existing swap partitions.

   - `swapoff -a`

   Wipe all filesystems on the SSD.

   WARNING: This will delete all data on the SSD.

   - `wipefs -a /dev/[devicename]`

   Make a swap area on the SSD.

   - `mkswap /dev/[devicename]`

   Enable the swap area with priority 10.

   - `swapon -p 10 /dev/[devicename]`
   - It is highly recommended that all DC P4800X swap devices use the same swap priority level, so that swap accesses are equally distributed across both devices.

4. Follow the **Best Known Configuration Details** in Section 1.2: Typical I/O Execution, steps 1-6.

Additional advanced optimizations can be found in the appendix.

- Appendix A: Set all CPUs in a system to No-CBs CPUs to reduce paging p99.999 outliers.
- Appendix B: Advanced workload/system-specific tuning for Optimum Paging Performance. This section contains configuration changes that may not apply to all systems or workloads.

7.4 Running the PMbench Benchmark

1. Run Pmbench v0.8 (https://bitbucket.org/jisooy/pmbench).

   Pmbench is a microbenchmark which generates synthetic virtual memory page access traffic. It returns a histogram of page access times and the total number of page accesses during a specified test duration.

   Usage: `./pmbench -z -r 0 -a histo -j [num_jobs] -s [size_MiB] -m [size_MiB] [duration_secs]`

   You will likely need to run multiple Pmbench jobs in order to saturate paging. For best paging throughput, the Pmbenches should be evenly distributed across NUMA nodes using `numactl`.

   The `[size_MiB]` argument for the `-s` and `-m` parameters should be the same value, and it represents the total memory used by that pmbench process's jobs. We suggest having the total memory use of all Pmbench processes be 2x the total physical DRAM in the system. Total DRAM can be found using the `free` command.

   The following sample commands are for a system with 32GB of DRAM, two NUMA nodes, and at least 8 CPU cores per node. The commands run 16 Pmbench jobs, binding each Pmbench process to two different NUMA nodes (8 jobs per node) with 64GB total memory use (32GB per process), recording over 5 minutes.

   ```
   numactl -m 0 -N 0 -- ./pmbench -z -r 0 -a histo -j 8 -s 32768 -m 32768 300 &
   numactl -m 1 -N 1 -- ./pmbench -z -r 0 -a histo -j 8 -s 32768 -m 32768 300
   ```

   Note: The Pmbench warmup needed to populate the memory before the test may take a very long time, longer than the specified test duration.

§

Appendix A: Set All CPUs in a System to No-CBs CPUs to Reduce Paging p99.999 Outliers

When extending main memory using the Intel® Optane™ SSD DC P4800X and Linux paging, it is highly recommended to set all CPUs in the system as No-Callbacks (No-CBs) CPUs. A No-CBs CPU is a CPU whose RCU callbacks are allowed to run on CPUs other than the CPU making the callback, which improves the page access quality of service (QoS) of the workload CPU but may lower average page access throughput.

Read-Copy-Update (RCU) is a Linux Kernel locking mechanism for shared memory. When a shared object in memory is updated, a new object is created in memory and all pointers to the old object are set to point to the new object. Old objects then need to be periodically cleaned out (garbage collected).

The RCU callback is an implementation of garbage collection. On a default Linux kernel, RCU callbacks are run in softirq context on the same CPU that performed the callback. This can cause jitter and poor page access QoS when RCU callbacks are run on workload CPUs, especially when the workload makes heavy use of swap space. CPUs which are set to be "No Callbacks CPUs," or "No-CBs CPUs," are allowed to offload their RCU callbacks to threads which can then run on other (non-workload) CPUs. This allows for better page access QoS on the workload CPU.

1. Confirm that your existing Linux kernel configuration supports No-CBs CPUs.

   No-CBs CPU support was introduced in Linux Kernel 3.10, but not all Linux distribution kernels enable No-CBs CPU support.

   CentOS 7.3 supports No-CBs CPUs – skip to Step 4.

   Ubuntu 16.04.1 LTS does not support No-CBs CPUs – see Step 3.

   For all other distributions, check for the `CONFIG_RCU_NOCB_CPU` config option in your kernel's config file.

   - `grep "CONFIG_RCU_NOCB_CPU" /boot/config-`uname -r``

   If the command returns `CONFIG_RCU_NOCB_CPU=y`, your kernel supports No-CBs CPUs. Skip to Step 4.

   - If `CONFIG_RCU_NOCB_CPU_ALL=y`, your kernel is already set to have all CPUs be No-CBs CPUs, and you do not need to follow this guide.

   Otherwise, if the command returns `# CONFIG_RCU_NOCB_CPU is not set` or does not return any text, your kernel does not support No-CBs CPUs. See Step 3.

2. If your kernel does not support No-CBs CPUs, you will need to rebuild your kernel with enabled No-CBs support.

   Obtain the kernel source of your distribution, run `make oldconfig`, then edit the `.config` file and add the following lines:

   - `CONFIG_RCU_NOCB_CPU=y`
   - `CONFIG_RCU_NOCB_CPU_NONE=y`

   Build and install the kernel.

3. Set the `rcu_nocbs` kernel boot parameter in the GRUB config.

   `rcu_nocbs` specifies which CPUs are to be No-CBs CPUs at boot time.

   On a Red Hat/CentOS machine:

   - Use grubby to add `rcu_nocbs=0-X` (where `X` is the highest CPU core number in the system) to the kernel boot arguments list.
   - For example, on a 32-core machine:

     ```
     grubby --update-kernel=ALL --args=rcu_nocbs=0-31
     ```

   On an Ubuntu machine:

   - Edit `/etc/default/grub` and append `rcu_nocbs=0-X` (where `X` is the highest CPU core number in the system) to `GRUB_CMDLINE_LINUX_DEFAULT`.
   - For example, on a 32-core machine:

     ```
     GRUB_CMDLINE_LINUX_DEFAULT="rcu_nocbs=0-31"
     ```

   - Run `update-grub`, then reboot your machine.

`rcu_nocbs` is a tunable boot parameter which can also specify only specific CPUs to be No-CBs CPUs, instead of all CPUs. Additionally, the RCU callback threads may be affinitized to specific CPUs. Such tweaks are beyond the scope of this guide.

§

Appendix B: System-Specific Tuning for Optimum Linux Swap Performance

The following configuration changes may further improve paging performance, but are not appropriate for all systems or workloads.

- If benchmarking, follow the steps in "System Tuning for Optimum Performance."
- Disable page clustering

  By default, when reading a page from swap, multiple consecutive pages are read into DRAM at once. On Optane, page clustering may incur a latency penalty overhead when reading multiple random individual pages.

  ```
  echo '0' > /proc/sys/vm/page-cluster
  ```

- Disable transparent hugepages

  This will remove the overhead of coalescing pages into hugepages and then breaking them up in swap.

  ```
  echo 'never' > /sys/kernel/mm/transparent_hugepage/enabled
  echo 'never' > /sys/kernel/mm/transparent_hugepage/defrag
  ```

- Disable NUMA balancing.

  Both swap and NUMA balancing frequently access LRU lists, which maintain the hot/cold pages in the system. Disabling NUMA balancing prevents it from contending with swap over the LRU lists, improving performance.

  ```
  echo '0' > /proc/sys/kernel/numa_balancing
  ```

- (Kernel 4.6 and later) Increase the watermark_scale_factor

  Watermark_scale_factor is a tunable variable which affects the memory thresholds at which the swap daemon wakes up or sleeps. We want kswapd to wake up earlier, so we make it wake up when there is 4% of available memory left, instead of the default 0.1% of available memory.

  ```
  echo '400' > /proc/sys/vm/watermark_scale_factor
  ```

- (Kernel 4.11 or later) Recompile the kernel with multi-queue deadline I/O scheduler support.

  This scheduler prevents write I/O request merges from blocking read requests for a long time.

  In the kernel .config, set `CONFIG_MQ_IOSCHED_DEADLINE`, then recompile the kernel. Use a short queue depth of 64 and small max_sectors_kb to 32 prevent write from blocking read in I/O for swap for better swap in latency.

§

Appendix C: Updated Configuration & Performance

Table 10: System Configuration

Type	Configuration
CPU	Intel® Xeon® E5-2687W v4 3.0GHz 12 cores
Motherboard	Intel® S2600WT
Memory	32GB DDR4 (8GB x 4 DIMM)
BIOS version	SE5C610.86B.01.01.0024.021320181901
BIOS configuration	Hyper-threading disabled, CPU C-states disabled, P-states disabled
Other key settings	CPU Governor set to Performance Mode; IRQ Balancing Services (OS) = Off; SMP Affinity set in the OS; QD1 utilizes I/O Polling Mode
Linux version	CentOS 7.3.1611
Kernel version	4.14.50
FIO revision	3.5
Intel® Optane™ SSD SKU	INTEL SSDPED1K375GA, Intel® Optane™ SSD DC P4800X, 375GB, Add-in-Card form factor
Firmware	E2010435
Data collection date	August 13, 2018

Table 11: 4.x Kernel Performance Data with Configuration from Table 10

Settings	4KB Random Read, Queue Depth 1[1]					4KB Random Write, Queue Depth 16				
	Latency (µs)				IOPS	Latency (µs)				IOPS
	Avg	99%	99.999%	99.99999%		Avg	99%	99.999%	99.99999%	
Turbo off 3 GHz	8.82	10.5	44	136	109K	20.64	29.5	63	311	635K

Notes:

Max values can be affected by system services (example: udevd system service) and are not deterministic or statistically significant based on configuration variances. Therefore 99,99999% data is shown.

[1]QD1 performance uses ioengine=pvsync2 w/ hipri

Queue Depth of 16 assumed 4 workers