Achieving Optimal Performance and Endurance on Coarse-grained Indirection Unit SSDs

Whitepaper

October 2021

Document Number: 338395-003US

Ordering Information

Contact your local Intel sales representative for ordering information.

Revision History

Revision Number	Description	Revision Date
001	• Initial release	October 2018
002	• Introduced OS level tools and processes for retrieving WAF information from Intel NVMe SSDs • Added next generation Intel SSD D5-P5316 performance and endurance data for synthetic and real-world workloads • Updated techniques and emerging industry trends to maximize performance and endurance of Intel SSD D5-P5316	September 2021
003	• Table 5, changed ***random*** to ***sequential*** in header cells	October 2021

Key Contacts

Andrzej Jakowski, Solution Architect	andrzej.jakowski@intel.com
Chunhong Mao, Solution Architect	chunhong.mao@intel.com
Esmond Chang, Solution Architect	esmond.chang@intel.com
Sarika Mehta, Platform Architect	sarika.k.mehta@intel.com
Yuyang Sun, NPSG Product Marketing Manager	yuyang.sun@intel.com
Sagar Kumathe, NPSG Product Marketing Engineer	sagar.kumathe@intel.com
Gert Pauwels, NPSG Technical Support Specialist	gert.pauwels@intel.com

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

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates.

Tested by Intel from April – July 2021. See Section 6.1 for configuration details.

No product or component can be absolutely secure.

Your costs and results may vary.

Contents

Tables

Figures

Glossary

Table 1: Glossary

Indirection Unit (IU)	An object of a particular size that can be accessed using some kind of reference, e.g., name, id, pointer, etc. For example, logical block (sector) represents IU for the storage devices. Actual user data on the storage device may be stored in different physical locations on the media, but users always reference that data by providing the logical block address (LBA). SSD IU refers to the internal construct that the SSD uses to manage data placement on the physical media.
Read-Modify-Write	A process of updating content in memory or storage. It consists of reading old content, merging it with new content, and subsequently writing the updated data back to the media
Write Amplification Factor (WAF)	An industry standard term quantifying the amount of data written to a storage target (media, disk, etc.) relative to amount of data written by an application. It is calculated by dividing the amount of data written to storage target by the amount of data written by application. For example, WAF=4 means that 4x more data has been written to the storage target than was written by application. On SSD level WAF is calculated as amount of data written to physical media relative to host writes. SSD WAF increases due to various factors for example wear leveling causes the same data to be rewritten to different physical locations to maintain SSD endurance.
Write Shaping	Name for techniques intended to improve overall performance and endurance of bulk capacity (e.g., QLC SSDs) by pairing it with different (e.g., faster) media. Write shaping leverages unique capabilities of those different media (e.g., small IO handling is optimal on faster media than on capacity media) to deliver optimized solution on system level.

1 Introduction

Recent increases in storage density allow larger capacity SSDs to be built, utilizing larger Indirection Unit (IU) sizes. In this document we present techniques on how to efficiently use coarse-grained IU SSDs.

1.1 Audience and Purpose

1.1.1 Target Audience

IT infrastructure architects, solution architects, and system administrators who plan and implement hardware and software storage infrastructure updates will benefit from the information presented in this document. This document assumes familiarity with basic storage and memory terminology, as well as working knowledge of performance and endurance management concepts.

1.1.2 Purpose

The whitepaper provides best known methods (BKMs) and practices to efficiently use high-capacity SSDs that utilize greater than 4KiB IU. This includes, but is not limited to, some of the latest Intel QLC 3D NAND SSDs. Industry standard terminology pertaining to endurance and performance management of modern SSDs is provided in the Glossary.

1.1.3 Scope

This whitepaper explains how endurance related telemetry can be retrieved from Intel SSDs and demonstrates how non-optimal vs. optimal writes can impact performance and endurance. Performance and endurance impacts are quantified on example synthetic benchmarks and a real-world storage stack (e.g., DAOS). Finally, we explore some of the techniques and emerging industry trends intended to maximize performance and endurance for coarse-grained IU SSDs.

2 Write Amplification Factor and Retrieving Endurance Telemetry from Intel SSDs

With the advancements of NAND technology and form factors innovations, larger and denser NAND SSDs are being built. Today's SSDs are as large as 30.72TiB and are expected to double soon.

Traditional NAND SSDs need to maintain an indirection system to translate logical block addresses (LBA) accessed by users into physical locations on the media. The indirection system, called Logical to Physical (L2P), is typically stored and managed inside the SSD. L2P hides the complexities related to media management – such as wear-leveling – from users but requires that resources like DRAM and compute are added into the SSD. L2P is often represented as a table stored inside an SSD's DRAM and its size is proportional to the SSD capacity. For example, a large SSD L2P table can reach tens of GiB. Because of the size requirement, the DRAM required to store the L2P table becomes a significant cost driver, therefore a need to reduce DRAM cost arises. One of the techniques that can be employed to reduce L2P size is associate a larger, user-accessible capacity unit (e.g., 64KiB vs 4KiB) – Indirection Unit (IU) – with the physical location on the media. This approach reduces the total number of IUs needed to cover the entire SSD capacity and thus reduces SSD DRAM requirements. For example, a 16x increase of IU size from 4KiB to 64KiB will roughly reduce L2P size by 16x, significantly helping reduce the SSD cost.

Smaller capacity SSDs, which utilize 4KiB IUs, work well with existing host software. This is because most writes submitted by host software to the SSD are in page (4KiB for x86 based systems) granularity and alignment. It is possible for the host system to submit sub-IU-sized or misaligned writes to the SSD (e.g., 512B) but typically these writes are not optimal from a performance and endurance perspective. To realize full performance and endurance potential of the SSD, adherence to IO size and alignment recommendations of the SSD is required.

One example of SSDs that utilizes larger IU is the Intel® SSD D5-P5316, which is available in 15.36TiB and 30.72TiB capacities. The D5-P5316 utilizes a larger indirection unit of 64KiB. Customers who want to efficiently use coarse-grained IU SSDs should modify their software so that writes issued by host systems are aligned to a multiple of the IU and are sized as a multiple of the IU as well. While it is still possible for the host system to issue writes that are smaller than the IU (e.g., 4KiB), SSD performance and endurance will be impacted.

In the following sections, we explain techniques that enable users to determine if a workload is optimal from the SSD perspective using tools widely available in the industry. We also provide a quantification of the endurance and performance penalties when a workload is not optimal. Finally, we will present some techniques to alleviate those challenges along with emerging industry trends coupling coarse-grained IU SSDs with fast media to leverage the unique characteristics of both, thereby delivering a TCO optimized solution.

2.1 SSD Endurance Management and Read-Modify-Write Cycle

To maximize endurance, modern SSDs apply wear leveling techniques to determine the best data placement on the media. One wear leveling technique is to map logical blocks (LB) accessible by the host system to physical locations on the media, so that a given LB may point to different physical locations inside the SSD over time. The SSD controller manages that map (L2P) and determines where to place user data on the media. This allows all media blocks to be written approximately the same number of times, thus maximizing SSD endurance.

The LB-to-physical location map typically associates 512B LB to larger units such as 4KiB IUs. This technique achieves an SSD cost reduction associated with storing the indirection map but may negatively impact SSD write performance and endurance. Figure 1 shows an example storage device implementing this concept. The example storage device is divided into smaller units called sectors. A sector is the minimal unit that can be read from, or written to, and is typically 512B in size. Several sectors (8 in this example) are tracked together internally by the storage device controller which can only access the underlying media in the larger units (4KiB in size in this example). The image illustrates optimal and non-optimal writes to that storage device. An optimal write has the following characteristics:

1. **Correctly sized** – size is equal to the multiple of the "Internal storage device unit," e.g., 4KiB.
2. **Correctly aligned** – starting address is aligned to a multiple of the "Internal storage device unit," e.g., write starts at the beginning of the eighth sector.

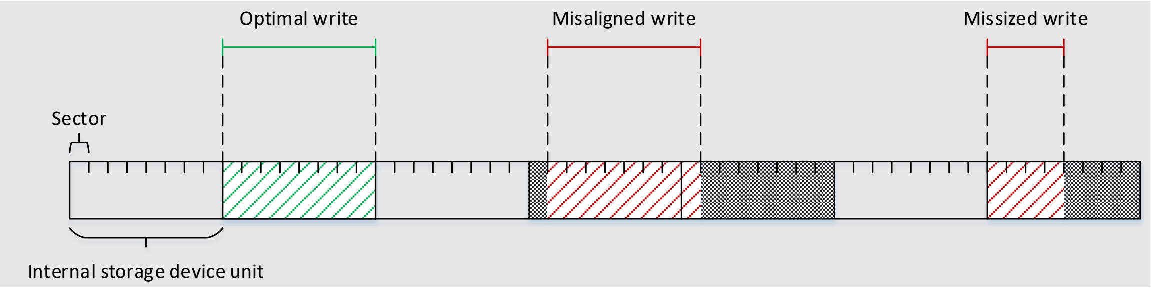
Figure 1 also illustrates writes that are not optimal from the storage device perspective. The first red-colored write is misaligned as it starts at the 25th sector. This write requires the storage device to read old data from two units (shown in grey), merge it with new data, and write two units back to the media. A similar situation happens for the second red-colored write operation where the I/O is properly aligned but its size is smaller than the unit size. As a result, the storage device controller needs to read the whole unit, merge it with new data, and write the single unit to media causing the amount of data written to media to be larger than the user-initiated write.

Finally, Figure 1 also illustrates the concept of the Read/Modify/Write (RMW) cycle and its negative impact on:

1. **Performance:** A single write operation requires the storage device to perform two I/Os: read the old content from the media, merge it with new content, and write all the data pieces back to the underlying media
2. **SSD endurance:** In the RMW cycle, the storage device must write more data than was originally requested. As shown by the first red colored write in the Figure 1, the storage device was requested to store eight sectors. This write triggered the RMW cycle, which ultimately caused sixteen sectors to be written back to the underlying media.

As indicated earlier, for certain high-capacity Intel SSDs SKUs the IU size is increased to enable SSD cost reductions. While different IU sizes may be offered (e.g., 16KiB, 64KiB) depending on the SSD SKU, this whitepaper provides BKMs and practices that are applicable to any IU size.

Figure 1: Example Storage Device Layout



2.2 Retrieving Endurance Information from Intel SSDs

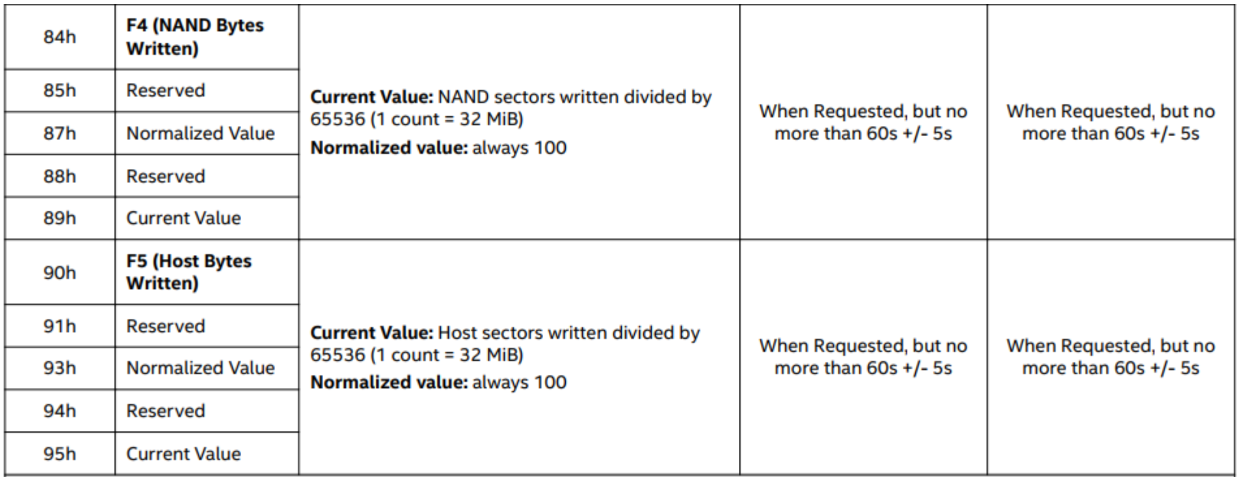
Write Amplification Factor (WAF) is an industry-wide term that quantifies how much actual data is written to storage media relative to host writes. The WAF for SSDs is a function of multiple factors. These factors include internal mechanisms such as media wear leveling algorithms and external factors such as the size and alignment of host-initiated writes. Regardless of source, application developers should focus on minimizing WAF to maximize SSD lifetime and performance. The following formula can be used to calculate the average WAF for a given workload:

$$\mathrm{WAF} = \frac{\Delta media\_writes}{\Delta host\_writes} = \frac{media\_writes_{after} - media\_writes_{before}}{host\_writes_{after} - host\_writes_{before}}$$

Calculating WAF requires retrieving telemetry from the SSD before and after running a workload. The above formula can be easily adapted to calculate the average lifetime WAF for fresh out-of-the-box SSDs ($media\_writes_{before} = host\_writes_{before} = 0$).

There are several ways to retrieve WAF telemetry information from an Intel NVMe SSDs. The Intel MAS tool (Intel® Memory and Storage Tool) is the easy way to access the SMART (Self-Monitoring, Analysis, and Reporting Technology) log page. Similarly, `endurance_profiler` tool provides user-friendly way to retrieve WAF telemetry. It is also possible to retrieve WAF telemetry using other tools (e.g., `nvme-cli`) that rely on the WAF information stored in SSD SMART log pages. Figure 2 shows how to retrieve WAF telemetry for Intel® D5-P5316 using the SMART log page.

Figure 2: Intel® SSD D5-P5316 SMART log page (log identifier CAh) specification. *†*



84h	**F4 (NAND Bytes Written)**	**Current Value:** NAND sectors written divided by 65536 (1 count = 32 MiB) **Normalized value:** always 100	When Requested, but no more than 60s +/- 5s	When Requested, but no more than 60s +/- 5s
85h	Reserved			
87h	Normalized Value			
88h	Reserved			
89h	Current Value			
90h	**F5 (Host Bytes Written)**	**Current Value:** Host sectors written divided by 65536 (1 count = 32 MiB) **Normalized value:** always 100	When Requested, but no more than 60s +/- 5s	When Requested, but no more than 60s +/- 5s
91h	Reserved			
93h	Normalized Value			
94h	Reserved			
95h	Current Value			

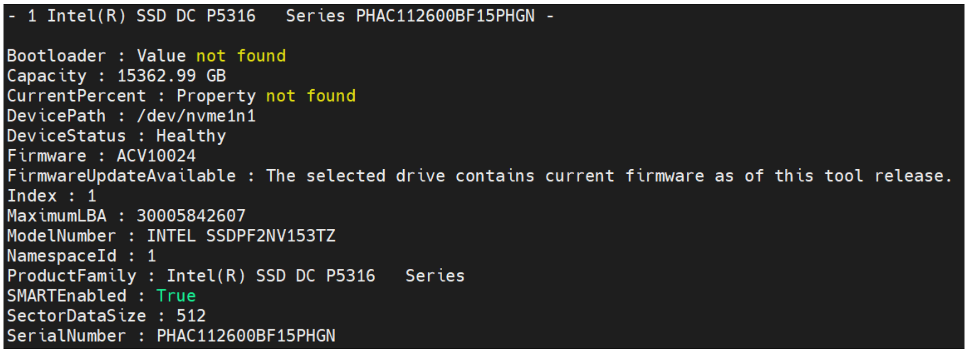
2.2.1 Retrieving WAF information using Intel MAS tool

Intel® MAS tool retrieves F4 and F5 attributes in a user-friendly manner. The following commands can be used to retrieve WAF telemetry:

```
#intelmas show -intelssd   (discovers Intel SSDs index number)
```

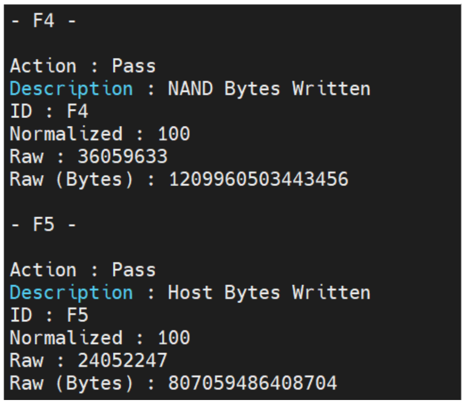
See example output for an Intel® SSD D5-P5316 where the Index number can be found in the "Index" row representing the device path to /dev/nvme1n1.

Figure 3: Example output from Intel MAS tool showing discovered Intel SSDs



```
- 1 Intel(R) SSD DC P5316   Series PHAC112600BF15PHGN -

Bootloader : Value not found
Capacity : 15362.99 GB
CurrentPercent : Property not found
DevicePath : /dev/nvme1n1
DeviceStatus : Healthy
Firmware : ACV10024
FirmwareUpdateAvailable : The selected drive contains current firmware as of this tool release.
Index : 1
MaximumLBA : 30005842607
ModelNumber : INTEL SSDPF2NV153TZ
NamespaceId : 1
ProductFamily : Intel(R) SSD DC P5316   Series
SMARTEnabled : True
SectorDataSize : 512
SerialNumber : PHAC112600BF15PHGN
```

Figure 4: Intel MAS tool output highlighting F4 and F5 attributes.



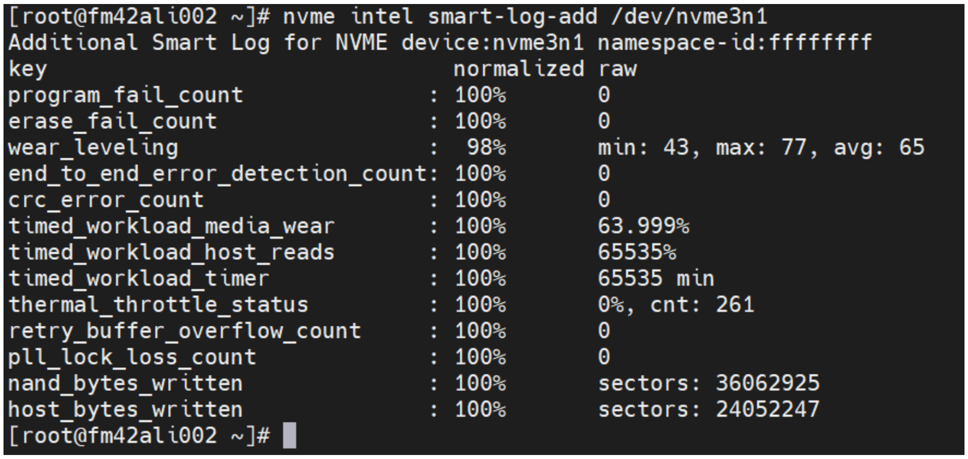
```
- F4 -

Action : Pass
Description : NAND Bytes Written
ID : F4
Normalized : 100
Raw : 36059633
Raw (Bytes) : 1209960503443456

- F5 -

Action : Pass
Description : Host Bytes Written
ID : F5
Normalized : 100
Raw : 24052247
Raw (Bytes) : 807059486408704
```

Alternatively, it is possible to retrieve WAF information using the `nvme-cli` utility. This can be achieved using the "intel" plug-in which parses and prints out WAF information in a user-friendly way or by manually retrieving and parsing the 202 log page.

The following command shows how to retrieve WAF information using NVMe CLI with the "intel" plugin and Figure 5 shows an example output.

```
#nvme intel smart-log-add /dev/nvmeXn1
```

Figure 5: **Example output from NVMe CLI utility providing parsed WAF telemetry in nand_bytes_written and host_bytes_written parameters.**



```
[root@fm42ali002 ~]# nvme intel smart-log-add /dev/nvme3n1
Additional Smart Log for NVME device:nvme3n1 namespace-id:ffffffff
key                               normalized raw
program_fail_count              : 100%       0
erase_fail_count                : 100%       0
wear_leveling                   :  98%       min: 43, max: 77, avg: 65
end_to_end_error_detection_count: 100%       0
crc_error_count                 : 100%       0
timed_workload_media_wear       : 100%       63.999%
timed_workload_host_reads       : 100%       65535%
timed_workload_timer            : 100%       65535 min
thermal_throttle_status         : 100%       0%, cnt: 261
retry_buffer_overflow_count     : 100%       0
pll_lock_loss_count             : 100%       0
nand_bytes_written              : 100%       sectors: 36062925
host_bytes_written              : 100%       sectors: 24052247
[root@fm42ali002 ~]# 
```

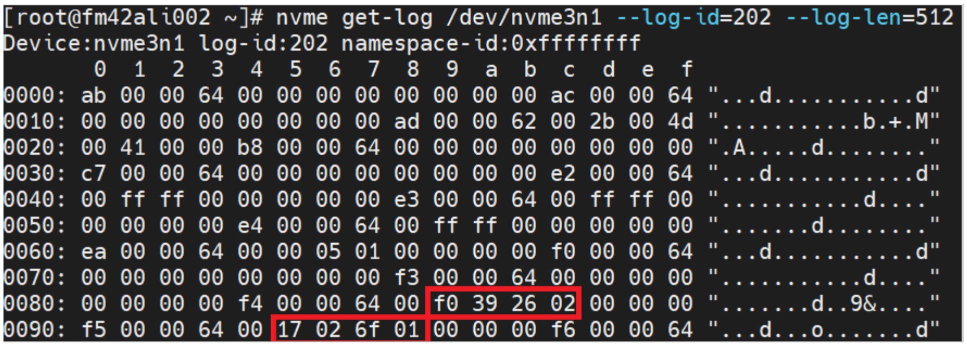
The command below shows how to retrieve 202 log page, then Figure 6 shows highlighted F4 and F5 attributes on NVMe CLI output.

```
#nvme get-log /dev/nvmeXn1 --log-id=202 –log-len=512
```

Final version of converted F4 and F5 attributes:

- Attribute F4: 0x022639f0=36059632 (32MiB)
- Attribute F5: 0x016f0217=24052247 (32MiB)

Figure 6: **Example output from NVMe CLI for 202 log page. NAND writes and host writes highlighted in red rectangle.**



```
[root@fm42ali002 ~]# nvme get-log /dev/nvme3n1 --log-id=202 --log-len=512
Device:nvme3n1 log-id:202 namespace-id:0xffffffff
       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
0000: ab 00 00 64 00 00 00 00 00 00 00 00 ac 00 00 64 "...d...........d"
0010: 00 00 00 00 00 00 00 00 ad 00 00 62 00 2b 00 4d "...........b.+.M"
0020: 00 41 00 00 b8 00 00 64 00 00 00 00 00 00 00 00 ".A.....d........"
0030: c7 00 00 64 00 00 00 00 00 00 00 00 e2 00 00 64 "...d...........d"
0040: 00 ff ff 00 00 00 00 00 e3 00 00 64 00 ff ff 00 "...........d...."
0050: 00 00 00 00 e4 00 00 64 00 ff ff 00 00 00 00 00 ".......d........"
0060: ea 00 00 64 00 00 05 01 00 00 00 00 f0 00 00 64 "...d...........d"
0070: 00 00 00 00 00 00 00 00 f3 00 00 64 00 00 00 00 "...........d...."
0080: 00 00 00 00 f4 00 00 64 00 f0 39 26 02 00 00 00 ".......d..9&...."
0090: f5 00 00 64 00 17 02 6f 01 00 00 00 f6 00 00 64 "...d...o.......d"
```

2.2.3 Retrieving WAF Information Using endurance_profiler Tool

Manual retrieval and conversion of F4 and F5 attributes from SSD to calculate WAF may be prone to errors. To improve that intel offers `nvme-cli` wrapper script called `endurance_profiler`. This utility is a Linux Bash script that uses `nvme-cli` to extract F4 and F5 attributes before and after a workload run.

The `endurance_profiler` utility is an open-source utility that is available on https://github.com/intel/endurance_profiler.

The comments below show how the endurance profiler is configured and started.

```
# endurance_profiler.sh setDevice nvme1n1
[CHECKNVMENAMESPACE] nvme device nvme1n1 exists
[SETDEVICE] Device set to nvme1n1
# endurance_profiler.sh start
[START] Starting endurance_profiler.sh
[START] Logging namespace nvme1n1. Log filename
/var/log/endurance_profiler/endurance_profiler.log
[START] endurance_profiler.sh has pid=7936
[STATUS] Service endurance_profiler.sh with pid=7936 running
```

The sub command “start” will retrieve the F4 and F5 attributes for the configured NVMe device. It is recommended to run a workload for more than an hour with `endurance_profiler` running in background. When calling the sub command “WAFinfo” the F4 and F5 attributes are retrieved again, and the Write Application Factor is calculated using the formula described above. The calculated WAF is printed out on the console.

```
# endurance_profiler.sh WAFinfo
Drive                            : Intel(R) SSD DC P5510 Series 3840GB
Serial number                    : BTAC050304NY3P8AGN
Device                           : /dev/nvme1n1
smart.write_amplification_factor : 3.14
smart.media_wear_percentage      : 0.030%
smart.host_reads                 : 19%
smart.timed_work_load            : 102 minutes
```

The sub command “start” is started as a background process. To stop it simply invoke “stop” sub command.

```
# endurance_profiler.sh stop
[STOP] Stopping endurance_profiler.sh with pid=7936
[STOP] kill 7936
```

2.3 Performance and Endurance Impact of Non-optimal Workload on Coarse-grained IU SSD

In this section we compare the performance and endurance impact of non-optimal IO and optimal IO for coarse-grained IU SSDs. In the experiments shown in this section we used an Intel® SSD D5-P5316 64KiB IU drive. We used synthetic benchmarks to mimic the scenario where optimal and non-optimal IO operations are issued by the host.

Before each experiment we performed SSD preconditioning steps to ensure stable and repeatable measurements. Refer to 6.2 for details on SSD preconditioning.

2.3.1 Performance and Endurance Comparison Between Small Writes and IU Sized Writes

In this experiment we focus on demonstrating the WAF impact by contrasting 2 workloads. Both workloads write the same amount of data to the SSD. The first workload used 4KiB block size writes that are smaller than the IU size of the SSD while the second workload used a block size equal to the IU (64KiB). In both workloads, writes are aligned to the block size; for the first workload writes are aligned to 4KiB, for the second workload writes are aligned to 64KiB. Refer to 6.3 for details of benchmark configuration used in these examples.

Tables 2 and 3 show the performance and WAF numbers observed after random and sequential workload tests. Random 4KiB write workloads resulted in a WAF around 16 times higher than the 64KiB write workload. Average completion latency was around 2ms better for 64KiB than for 4KiB. This endurance and performance impact is a result of the RMW cycle incurred for small (sub-IU) writes to SSD.

Table 2: Performance and WAF numbers for 4KiB and 64KiB random write workload. 300GiB of data was written into SSD in both scenarios.

Metric\workload	4KiB random write	64KiB random write
WAF	74.77	4.53
Bandwidth	23.9 MiB/s	424 MiB/s
Average completion latency	21.9 ms	19.7 ms

Interestingly, we did not observe a significant WAF and performance impact when comparing 4KiB and 64KiB sequential write workloads. In both cases WAF was close to the ideal value of 1. We attribute this behavior to the SSD's firmware coalescing capability.

Table 3: Performance and WAF numbers for 4KiB and 64KiB sequential write workload. 5TiB of data was written to the SSD in each scenario.

Metric\workload	4KiB sequential write	64KiB sequential write
WAF	1.02	1.02
Bandwidth	1,360 MiB/s	2,985 MiB/s
Average completion latency	0.093 ms	0.688 ms

2.3.2 Performance and Endurance Comparison Between Misaligned and IU Aligned Writes

In the second experiment we focused on a scenario comparing two workloads issuing writes that are identically sized but differently aligned. The first workload (non-optimal) issues writes that are 4KiB aligned while in the second workload (optimal) all writes are 64KiB aligned (IU aligned). Detailed benchmark configuration is provided in 6.3.

Tables 4 and 5 provide the performance and WAF numbers we observed. Table 4 presents data for the random write workload. The observed WAF is around 2x higher for the 4KiB aligned workload than the 64KiB aligned workload. A similar trend can be observed for performance numbers where bandwidth and latency for the optimal workload (64KiB aligned) is more than two times better than the misaligned workload (4KiB).

Table 4: Performance and WAF numbers for 64KiB 4KiB aligned and 64KiB aligned random write workload. 300GiB of data was written into SSD in both scenarios.

Metric\workload	64KiB random write 4KiB alignment	64KiB random write 64KiB alignment
WAF	9.41	4.53
Bandwidth	208 MiB/s	424 MiB/s
Average completion latency	40.2 ms	19.7 ms

Similar observations can be made for sequential write workloads with differing alignment. Table 5 shows significant improvements for WAF and performance (latency and bandwidth) for workloads using optimal alignment.

Table 5: Performance and WAF numbers for 64KiB 4KiB aligned and 64KiB aligned sequential writes workload. 5TiB of data was written into SSD in both scenarios.

Metric\workload	64KiB sequential write 4KiB alignment	64KiB sequential write 64KiB alignment
WAF	1.4	1.02
Bandwidth	1,874 MiB/s	2,985 MiB/s
Average completion latency	4.4 ms	0.688 ms

2.3.3 Performance and Endurance Comparison Across Mixed Block Size Writes

The third experiment focused on workloads of 4KiB and 64KiB block sizes where both are performed in sequence. In our simulation we matched a majority (95%) of writes to SSD IU size (64KiB) and defined the remaining at 4KiB size. A primary focus was placed on the WAF impact. We observed a WAF of 8.2, around 1.8x higher, compared to the scenario of a 4.53 WAF from experiment 2.3.1. Detailed benchmark configuration is provided in 6.3.

Table 6: Performance and WAF numbers for 4KiB and 64KiB mixed random write workload. 300GiB of data was written into SSD in both scenarios.

Metric\workload	4KiB random write 5%	64KiB random write 95%
WAF	8.20	8.20
Bandwidth	14.1MiB/s	242.3MiB/s
Average completion latency	33.2ms	30.7ms

Table 6 provides the observed performance and WAF metrics from the tests. Performance measurements show that WAF performance in a mixed write workload falls between WAF measurements for complete 4KiB & 64KiB random writes as seen in Table 2. Bandwidth and latency show performance drops in this workload because of mixing write sizes and alignments.

3 Techniques to Optimize IO for Course-grained IU SSDs

This section focuses on demonstrating techniques that can be employed to optimize performance and endurance of coarse-grained IU SSDs. It provides not only simple techniques that can be leveraged today in existing operating environments like Windows and Linux, but also provides technical insights and data on emerging industry trends intended to optimize performance and endurance of coarse-grained IU SSD.

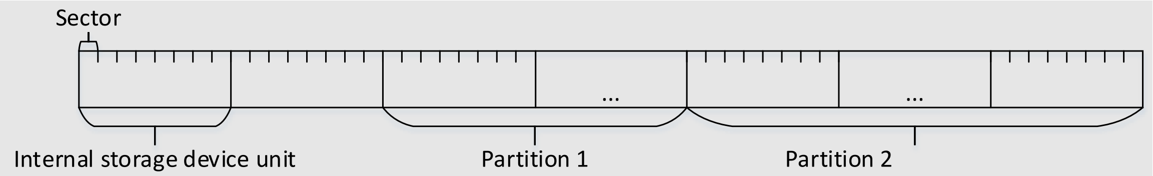
3.1 Recommended Techniques and Considerations for Optimizing IO

To achieve optimal performance and endurance on coarse-grained IU SSDs, consider the following techniques:

1. Properly size and align partitions to a multiple of the IU, as illustrated in Figure 7 below. The same technique may apply to any software or hardware solution implementing logical volume capability on coarse-grained IU SSDs. For example, it may include selecting proper stripe size / width / stride for RAID. Please refer to the documentation for specific products and operating systems.

```
# Example command line that will use “optimal” alignment for
# partitioning
# NOTE: when a non-optimal start address is specified
# the command will succeed with a warning. Use 1M alignment as it
# fits most storage topologies

parted /dev/nvmeXnY -a optimal -s mkpart primary 1M 100G

# or

parted /dev/nvmeXnY --align optimal --script mkpart primary 1M 100G

# To determine if a partition is optimally aligned use the below
# command.
parted /dev/nvmeXnY --script align-check optimal <part_id>
# Retrieve status of above command zero indicates optimal
# alignment
echo $?
```

Figure 7: Optimal SSD Partition Alignment



2. To achieve optimal I/O performance and endurance on coarse-grained IU SSDs, perform I/O directly on the raw block device file (e.g., `/dev/nvme0n1`) bypassing the logical file system and page cache (open file with `O_DIRECT`). For optimal performance, I/O should be sized and aligned to IU or its multiple.

3.2 Leveraging Fast Media to Deliver Cost and Density Optimized Solution with QLC

Recent innovations in QLC SSD technology achieve higher storage densities at lower cost than TLC SSDs. This is very attractive especially in big datacenter deployments where cost savings are realized at a server, rack, and data center levels. Pairing Intel® Optane™ persistent memory (PMem) with QLC SSDs, provides the solution with higher endurance stemming from higher Optane endurance and from staging non-optimal IO for coarse-grained IU SSDs until it can be converted to optimal IO. It also offers consistent low latency write performance for small writes serviced by Intel® Optane™ persistent memory. With such a combination, a solution can continue to experience higher write performance with increased storage density resulting in cost saving from higher density per storage box and cost effective QLC SSDs.

3.2.1 Introduction to Write Shaping

This section introduces write shaping concept and provides references to example implementations.

In order to realize full performance potential of coarse-grained IU SSDs broad ecosystem changes (e.g., filesystems using coarse-grained IU SSD attributes) are required. We have observed emerging industry trends coupling of coarse-grained IU SSDs (QLC SSDs) with faster media (e.g. Intel Optane persistent memory or Intel Optane SSD) to achieve optimal performance and endurance of QLC SSDs. Small host software building block implementing data placement algorithms, staging, aggregation and sequentialization of small IO into larger IO can be used to realize full potential of QLC SSD. Techniques leveraging caching, especially write caching (writeback) or write buffering of user data onto faster media, can help efficiently manage the performance and endurance of QLC SSDs. These and similar concepts fall into the category of "write shaping."

Figure 8: Architecture diagram for write shaping concept

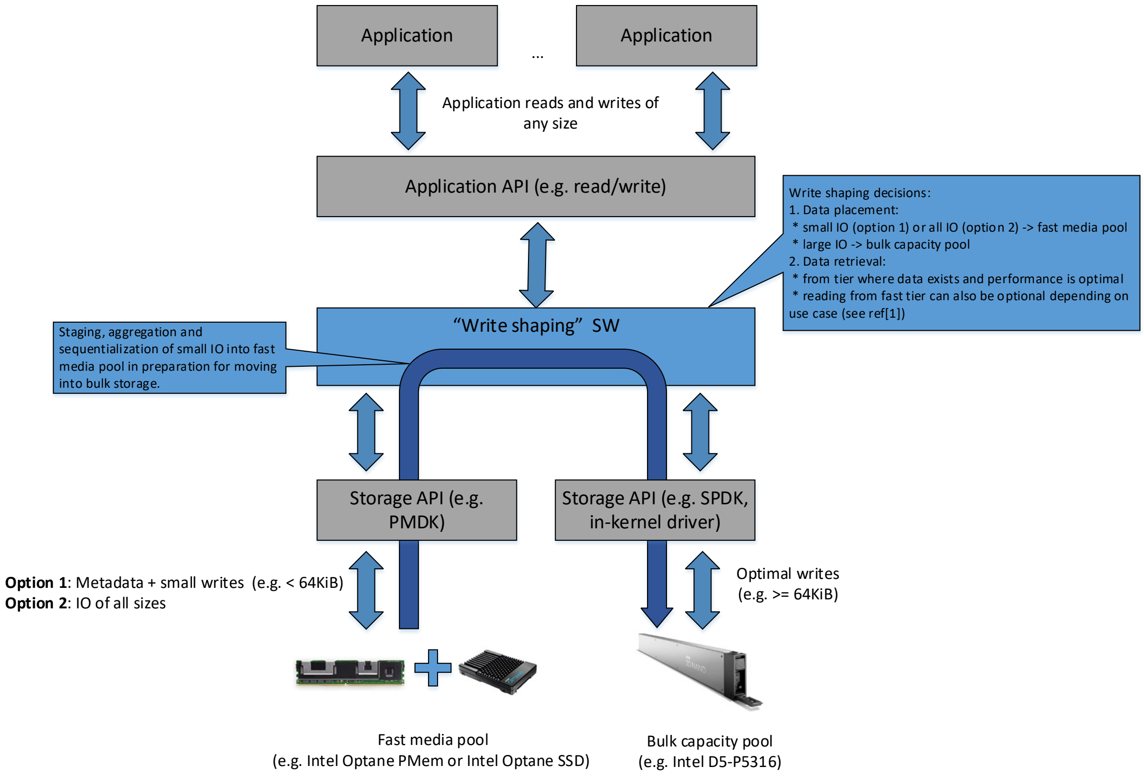

Application
...
Application
Application reads and writes of any size
Application API (e.g. read/write)
Write shaping decisions:
1. Data placement:
* small IO (option 1) or all IO (option 2) -> fast media pool
* large IO -> bulk capacity pool
2. Data retrieval:
* from tier where data exists and performance is optimal
* reading from fast tier can also be optional depending on use case (see ref[1])
"Write shaping" SW
Staging, aggregation and sequentialization of small IO into fast media pool in preparation for moving into bulk storage.
Storage API (e.g. PMDK)
Storage API (e.g. SPDK, in-kernel driver)
Option 1: Metadata + small writes (e.g. < 64KiB)
Option 2: IO of all sizes
Optimal writes (e.g. >= 64KiB)
Fast media pool (e.g. Intel Optane PMem or Intel Optane SSD)
Bulk capacity pool (e.g. Intel D5-P5316)

Figure 8 highlights an example of a write shaping architecture. Write shaping software integrates with the platform's storage stack and exposes media capacity to upper software layers. This is transparent to the application layer, which issues IO in the same manner as to physical storage devices. The write shaping layer, based on the incoming IO, makes data placement decisions where small IOs are staged in the fast media pool (e.g., Intel Optane) and large IOs are written directly into the bulk capacity pool (e.g., QLC SSDs – Intel SSD D5-P5316). Write shaping software asynchronously de-stages small writes from fast media to the bulk capacity pool. These algorithms aggregate small IO blocks into large, sequential IO blocks so the full bandwidth of bulk capacity can be used when data is de-staged.

There are multiple examples of storage stacks adopting this and similar concepts. In the next section we will focus on demonstrating the effectiveness of write shaping with an example implementation provided by the DAOS (Distributed Asynchronous Object Storage) stack.

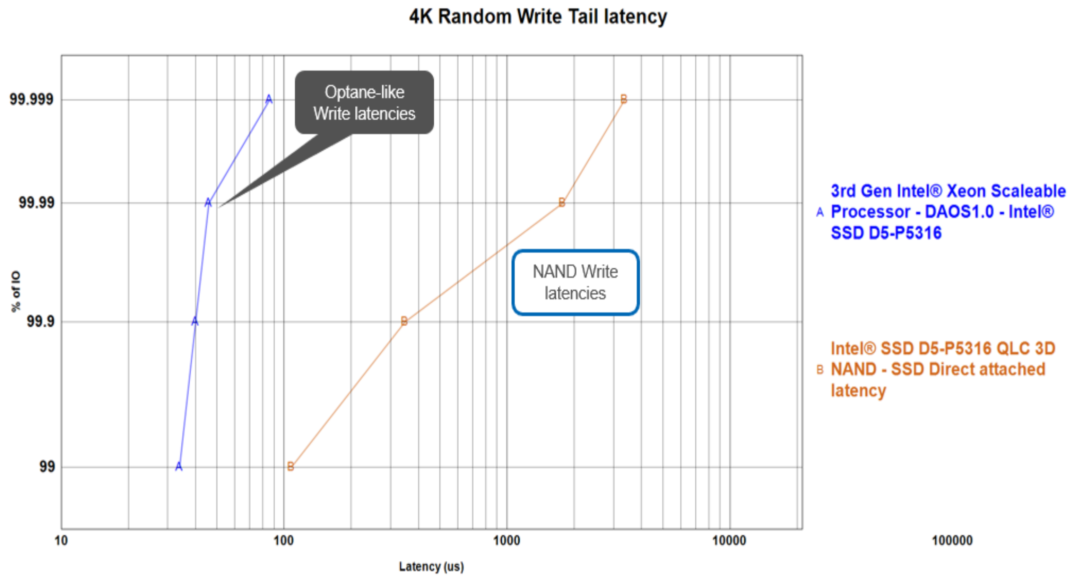
3.2.2 Reference Storage Platform – Example Implementation of Write Shaping with DAOS

This section presents performance measured with a reference storage platform which includes the DAOS software stack, Intel® Xeon® Platinum processor, a 100 gigabit Ethernet (GbE) NIC, Intel Optane PMem, and Intel QLC 3D NAND SSDs.

The reference storage platform uses 64KiB write IO aggregation and de-staging to align to the 64KiB indirection unit of Intel's second-generation QLC 3D NAND SSDs (D5-P5316). The write shaping mechanism routes IO with sizes less than the indirection unit to Intel Optane PMem and IO with sizes equal to or higher than the IU threshold to bulk QLC 3D NAND SSD storage. IO routed to Intel Optane persistent memory is aggregated until it reaches the 64KiB threshold. Upon reaching the threshold or if the persistent memory is running out of free space, the data is de-staged to bulk QLC 3D NAND SSD storage.

By configuring Optane persistent memory to service small IO, we observed ultra-low latencies. See Figure 9 for write latency comparison between Intel Optane PMem and NAND.

Figure 9: Reference Storage Platform write latency



Write shaping also helps improve bandwidth and IOPS for small random writes by leveraging the higher performance of faster media. Up to 2x improvement can be seen for small transfer sizes and lower queue depths as seen in Figures 10 & 11 below.

Figure 10: Reference Storage Platform write bandwidth

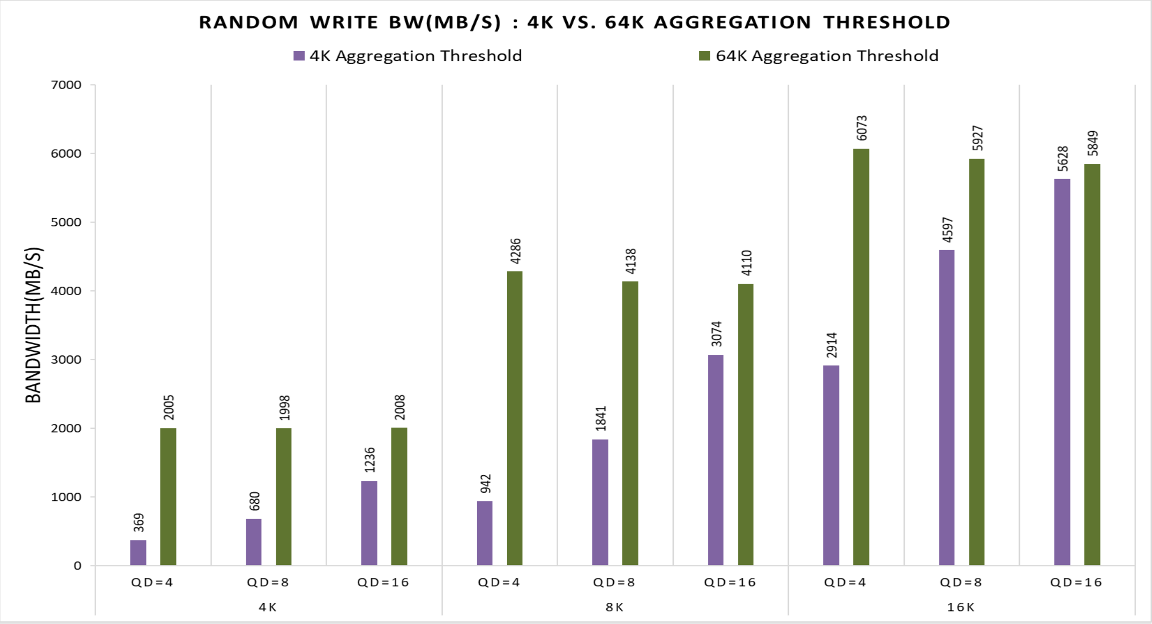
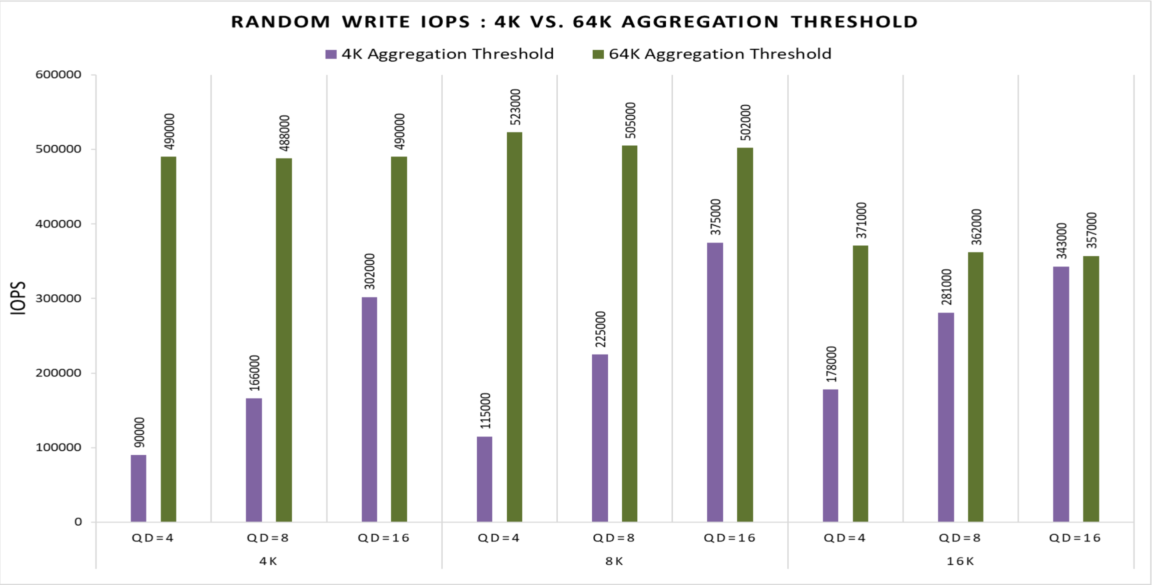


Figure 11: Reference Storage Platform write IOPS



Routing IO through Intel Optane PMem requires additional faster media capacity to be provisioned. The additional capacity required is also a factor of the distribution of IO sizes presented by the workload and of the data de-staging mechanism. This can be effectively managed through an efficient write shaping policy. There are multiple ways to implement such a policy that include platform considerations, SLA requirements, optimizations to take advantage of idle time during workload, and optimization of the capacity requirements on the faster media employed to house small random writes.

For reads, impact is minimal as QLC drives deliver a read performance on-par with TLC drives. For latency sensitive workloads reads may benefit when they are serviced from Optane persistent memory.

4 Coarse-grained IU QLC SSD Usages, Solutions and Eco-systems

Coarse-grained IU QLC SSDs fit well in many applications such as data analytics, HPC, object storage, cDVR, and CDN where workloads are read intensive and data blocks are larger than IU size.

In addition, there are a great variety of solutions developed and supported by Intel to enable coarse-grained IU SSD adoption. These solutions can be integrated into or referenced by proprietary storage system designs. In addition to DAOS introduced in this paper, MASSe (Media Aware Smart Storage Engine) provides an effective storage solution that improves bandwidth and latency by combining fast and slow media. Write Shaping RAID (WSR) is an intelligent storage software tiering architecture, optimized for the next generation of storage media and developed to take full advantage of the latest I/O improvements (e.g., PCIe gen4 and gen5). WSR shapes writes for sequentialization and reduces writes to NAND as much as possible by intelligently placing parts of the workload on media with better performance and endurance characteristics such as Intel Optane media. These techniques increase overall storage performance and minimize endurance impact from misaligned writes. WSR enables coarse-grained IU SSDs to be easily integrated into environments without updating applications.

The eco-system is evolving to embrace coarse-grained IU SSDs. For example, Ceph BlueStore allows users to configure the min_alloc_size to 16K or 64K so that the write IOs to the drives are aligned to the SSD IU size. In addition, the Intel SSD D5-P5316 is certified by Microsoft Azure stack HCI.

There are many innovators such as Lightbits, PLIOPS, and VAST that integrated coarse-grained IU QLC SSD for fast and cost-effective storage solutions. Lightbits offers a disaggregated software defined storage solution that is fully optimized for coarse-grained IU QLC SSDs to improve storage performance. PLIOPS leveraged their software to extend QLC SSD endurance and performance. VAST leveraged Optane media and QLC to offer all-flash scale-out file and object data storage.

5 Future Work and Final Notes

Potential future Intel work includes:

- **Ecosystem enabling** that includes work with standardization bodies, open source communities, and OSVs to enhance infrastructure and SW stacks so they can leverage QLC SSDs in optimal way.
- **Benchmarking**, performance and endurance modeling of existing storage solutions using QLC SSDs and Optane technology, and documenting value through solutions brief and customer-ready documentation.
- **Documenting** the techniques for achieving optimal performance and endurance through whitepapers/how-to document, etc.

6 Appendix A: Preconditioning Steps and Benchmark Configuration Details

6.1 System Configuration Details

The tables below contain the configuration details of the system used for tests in section 2.3, tested April to July 2021.

Table 7: System Configuration for Experiments from Sections 2.3.1 & 2.3.2

CoyotePass System Configuration	
BIOS	SE5C6200.86B.0022.D08.2103221623
CPU	Intel® Xeon® Platinum 8360Y 2 x sockets @2.4GHz, 36 cores/per socket
NUMA Nodes	2
DRAM	Total 256G DDR4@2933MT/s
OS	CentOS 8.3.2011
Kernel	5.11.13-1.el8.elrepo.x86_64
NAND SSD	Intel® SSD D5-P5316 15.36TB, FW Rev: ACV10026
fio Version	3.25

Table 8: System Configuration for Experiments from Section 2.3.3

CoyotePass System Configuration	
BIOS	SE5C6200.86B.0019.D29.2011110211
CPU	Intel® Xeon® Gold 5318Y 2 x sockets @2.1GHz, 24 cores/per socket
NUMA Nodes	2
DRAM	Total 792G DDR4@2933MT/s
OS	CentOS 7.9.2009
Kernel	3.10.0-1160.31.1.el7.x86_64
NAND SSD	Intel® SSD D5-P5316 15.36TB, FW Rev: ACV10026
fio Version	3.7

Table 9: Storage Server Configurations for Experiments from Section 3.2.2

Storage Server – SuperMicro SYS-220U-TNR System Configuration	
BIOS	1.0b
CPU	Intel® Xeon® Platinum 8368Q 2 x sockets @ 2.6GHz, 38 cores per socket
NUMA Nodes	2
DRAM	Total 128G DDR4 @ 3200MT/s per socket
Optane™ Persistent Memory	Total 1TB DDR-T per socket
Network Interface Card	100GbE RoCE v2 Mellanox Technologies MT27800 Family [ConnectX-5] per socket
OS – Storage Server	openSUSE Leap 15.2
Kernel – Storage Server	5.3.18-lp152.66-default
NAND SSD	6 x Intel® SSD D5-P5316 15.36TB, FW Rev: ACV10101 per socket
DAOS version	DAOS 1.0
fio Version	3.26

Table 10: Client Server Configurations for Experiments from Section 3.2.2

Storage Server – SuperMicro SYS-6019P-WTR System Configuration	
BIOS	3.3
CPU	Intel® Xeon® Gold 6139 2 x sockets @2.3GHz, 18 cores per socket
NUMA Nodes	2
DRAM	Total 96G DDR4 @ 2666MT/s per socket
Optane™ Persistent Memory	Total 1TB DDR-T per socket
Network Interface Card	100GbE RoCE v2 Mellanox Technologies MT27800 Family [ConnectX-5]
OS – Client Server	CentOS 7.7
Kernel – Client Server	3.10.0-1062.el7.x86_64
DAOS version	DAOS 1.0
fio Version	3.26

6.2 SSD Preconditioning

SSD precondition is required step to ensure stable and repeatable tests results. We used following procedure for SSD preconditioning:

1. Secure erasure of the SSD to wipe all previous user data. To perform cryptographic erase use following command:

   ```
   nvme format /dev/<nvme> -n 1 -ses 2 # replace <nvme> with device node
   representing your SSD
   ```

2. Sequential write over whole capacity of the SSD. The following FIO configuration can be used:

   ```
   [global]
   name=driveprep
   ioengine=sync
   direct=1
   thread=1
   buffered=0
   size=100%
   randrepeat=0
   fill_device=1
   norandommap
   log_avg_msec=1000
   group_reporting
   filename=/dev/nvme2n1

   [job1]
   stonewall
   bs=128k
   iodepth=256
   numjobs=1
   rw=write
   ```

Note: Full capacity overwrite can be achieved by setting the `fill_device` parameter set to 1. To achieve a fully sequential workload, ensure that the `numjobs` parameter is set to 1.

3. For random workload testing also perform the random write preconditioning step after sequential overwrite of the whole SSD. This step ensures randomization of the data inside SSD and prevents observing elevated performance resulting from sequential overwrite in step #2. This step is not needed for testing sequential workload.

   Following fio config can be used to perform random write preconditioning step:

   ```
   [global]
   name=waf_test
   ioengine=libaio
   log_hist_msec=1000
   log_hist_coarseness=0
   direct=1
   thread=1
   buffered=0
   randrepeat=0
   time_based
   ```

```
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
random_distribution=random
refill_buffers=1
size=100%
bs=1024k
iodepth=32
numjobs=4
rw=randwrite
runtime=14400

[wr_rnd_qd_128_1024k_4w]
filename=/dev/nvme2n1
write_hist_log=wr_rnd_qd_128_1024k_4w
```

6.3 Benchmark Configuration

Fio job configuration file for 64KiB random write:

```
[global]
name=waf_test
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
random_distribution=random
refill_buffers=1
size=100%
bs=64k
iodepth=32
numjobs=4
rw=randwrite
io_size=300G

[wr_rnd_qd_128_64k_4w]
filename=/dev/nvme2n1
write_hist_log=wr_rnd_qd_128_64k_4w
```

Fio job configuration file for 4KiB random write:

```
[global]
name=waf_test
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
random_distribution=random
refill_buffers=1
size=100%
bs=4k
iodepth=32
numjobs=4
rw=randwrite
io_size=300G

[wr_rnd_qd_128_4k_4w]
filename=/dev/nvme2n1
write_hist_log=wr_rnd_qd_128_4k_4w
```

Fio job configuration file for 64KiB sequential write:

```
[global]
name=mergedfio
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
refill_buffers=1
size=100%
bs=64k
iodepth=32
numjobs=1
rw=write
io_size=5T

[wr_qd_32_64k_1w]
filename=/dev/nvme2n1
write_hist_log=wr_qd_32_64k_1w
```

Fio job configuration file for 4KiB sequential write:

```
[global]
name=mergedfio
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
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
refill_buffers=1
size=100%
bs=4k
iodepth=32
numjobs=1
rw=write
io_size=5T

[wr_qd_32_4k_1w]
filename=/dev/nvme2n1
write_hist_log=wr_qd_32_4k_1w
```

Fio job configuration file for 64KB sizes, 4KB aligned random writes:

```
[global]
name=waf_test
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
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
random_distribution=random
refill_buffers=1
blockalign=4K
size=100%
bs=64k
iodepth=32
numjobs=4
rw=randwrite
io_size=300G

[wr_rnd_qd_128_64k_4w]
filename=/dev/nvme2n1
write_hist_log=wr_rnd_qd_128_64k_4w
```

Fio job configuration file for 64KB sized, 4KB aligned sequential writes:

```
[global]
name=waf_test
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
refill_buffers=1
blockalign=4K
size=100%
bs=64k
iodepth=32
numjobs=4
rw=write
io_size=5T

[wr_qd_128_64k_4w]
filename=/dev/nvme2n1
write_hist_log=wr_qd_128_64k_4w
```

Fio job configuration file for 95% 64KiB – 5% 4KiB random write:

```
[global]
name=waf_test
ioengine=libaio
log_hist_msec=1000
log_hist_coarseness=0
direct=1
thread=1
buffered=0
randrepeat=0
norandommap
group_reporting=1
percentile_list=1.0:25.0:50.0:75.0:90.0:99.0:99.9:99.99:99.999:99.9999:99.99999:9
9.999999:100.0
random_distribution=random
refill_buffers=1
size=100%
rw=randwrite

[wr_rnd_qd_114_64k_4w]
new_group
numjobs=19
bs=64k
iodepth=6
filename=/dev/nvme2n1

[wr_rnd_qd_114_4k_4w]
new_group
numjobs=1
bs=4k
iodepth=114
filename=/dev/nvme2n1
```

7 Appendix B: References

1. Distributed Storage Trends and Implications for Cloud Storage Planners
2. MASSé: A High-Performance Storage Solution for Intel® 3D XPoint™ and Flash SSDs: https://software.intel.com/content/www/us/en/develop/download/masse-a-high-performance-storage-solution.html
3. What's New in LightOS 2.0: https://www.lightbitslabs.com/blog/whats-new-in-lightos-2-0/
4. Flash density meets performance: https://vastdata.com/architecture/#qlc