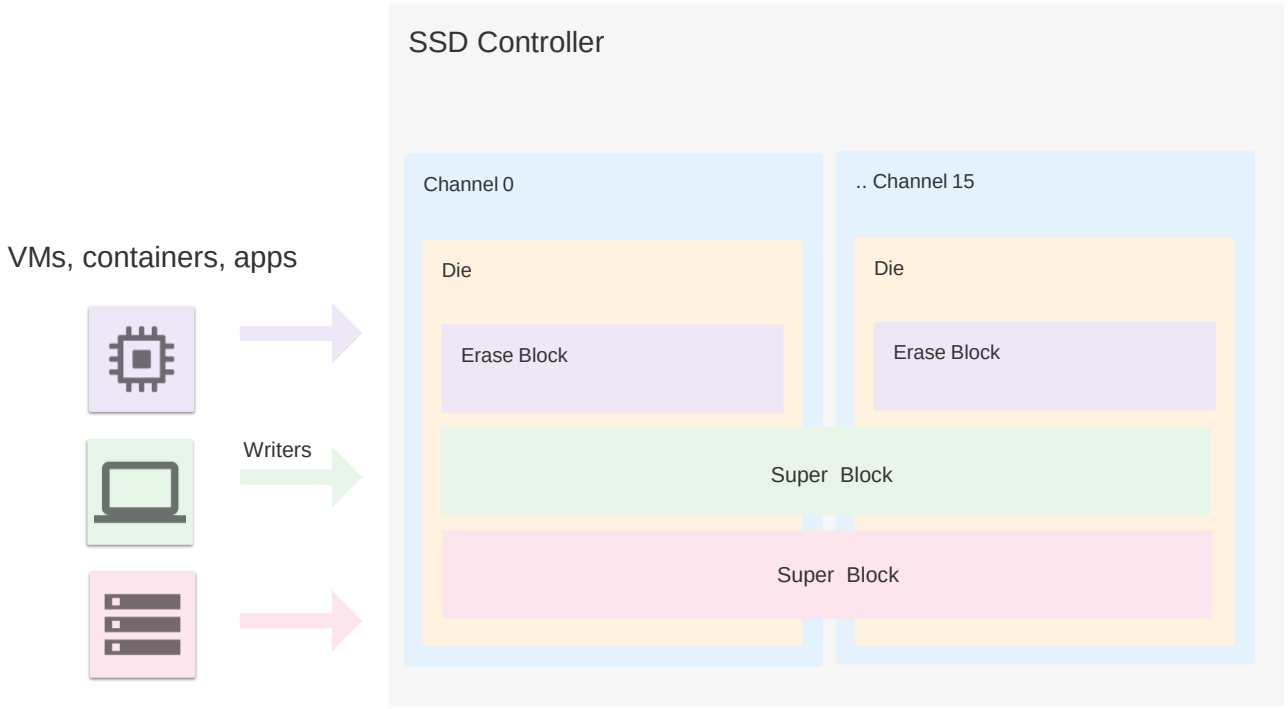
FARP-101-1 – FDP Multi-Namespace Experiment

Jonmichael Hands, Sr. Director Product Planning

07/22/24

What is Flexible Data Placement?

- Improved application performance and extended SSD endurance - reduce SSD write amplification (WAF)
- Software intelligently places data on SSD that is friendly with how flash works (blocks, channels, etc.)
- Improved isolation leads to better quality of service
- Fully backwards compatible, namespace support
- Scale SSD capacity with performance for compute workloads
- Lower overprovisioning → higher device utilization → lower TCO
- Lower WAF → higher endurance → lower TCO



Flexible Data Placement (FDP) Benefits

Enhanced performance: Lower WAF directly translates to faster write speeds.

Increased endurance: Less wear and tear on the NAND flash prolongs an SSD's lifespan.

Reduced overprovisioning: FDP allows for greater utilization of an SSD's raw capacity.

Improved Quality of Service (QoS): FDP's isolation capabilities ensure consistent performance even in multi-tenant environments

Key Advantages for Enterprises

Seamless Integration: FDP is fully backward compatible, allowing it to work with existing applications and infrastructure. Software can be enabled to support FDP, tagging directly or indirectly through multiple namespaces.

Scalability: FDP enables enterprises to scale SSD capacity in line with growing compute workloads.

Lower TCO: The combined benefits of improved performance, endurance, and capacity utilization significantly reduce the total cost of ownership for storage infrastructure.

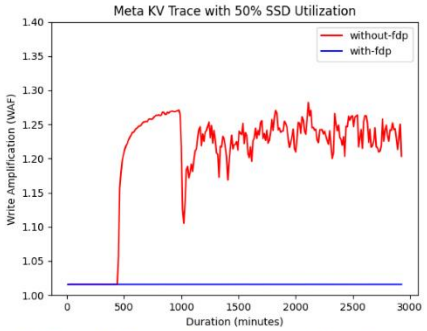
FDP Customer Use case

- Developed by Google, Meta, SSD vendors
- Multi-tenant, multiple namespaces, multiple applications, VMs, containers, microservices, log structured DB, caching
- Presentation at OCP detailing customer use cases

Cachelib: Case Study

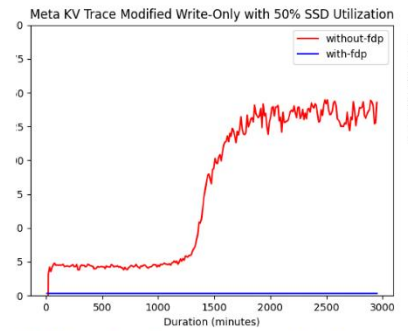
❖ **Significant improvement to Cachelib WAF without major changes**

- No impact on other metrics: Throughput, latency, cache hit-rate
- Cachelib code in process to merge in *under 5 months*

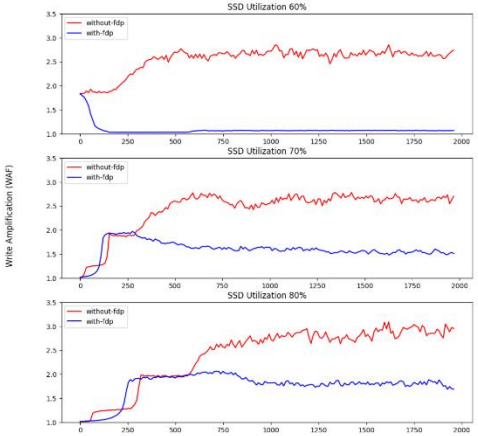


1. Raw Meta read-heavy workload

https://cachelib.org/docs/Cache_Library_User_Guides/Cachebench_FB_HW_eval/#running-cachebench-with-the-trace-workload

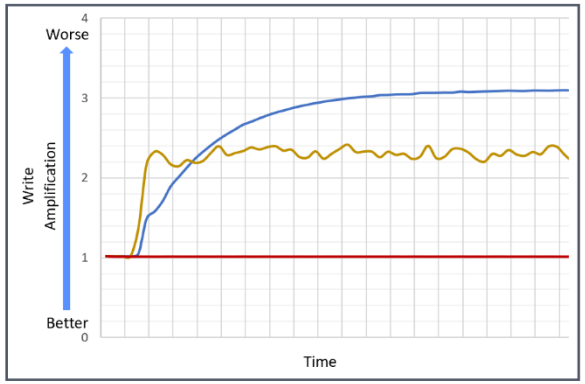


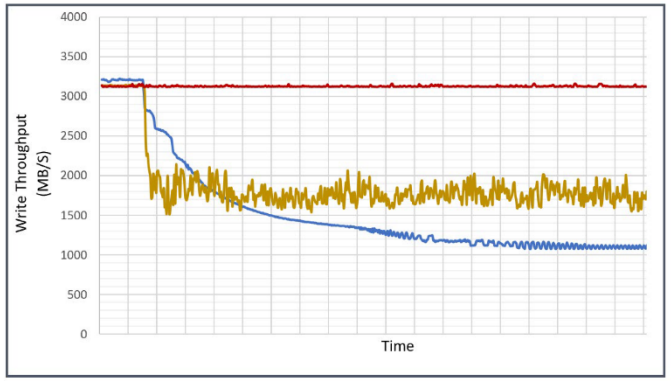
2. Altered write-heavy workload



3. Increased utilization on Meta write-heavy workload

Log Structured Workload Pattern

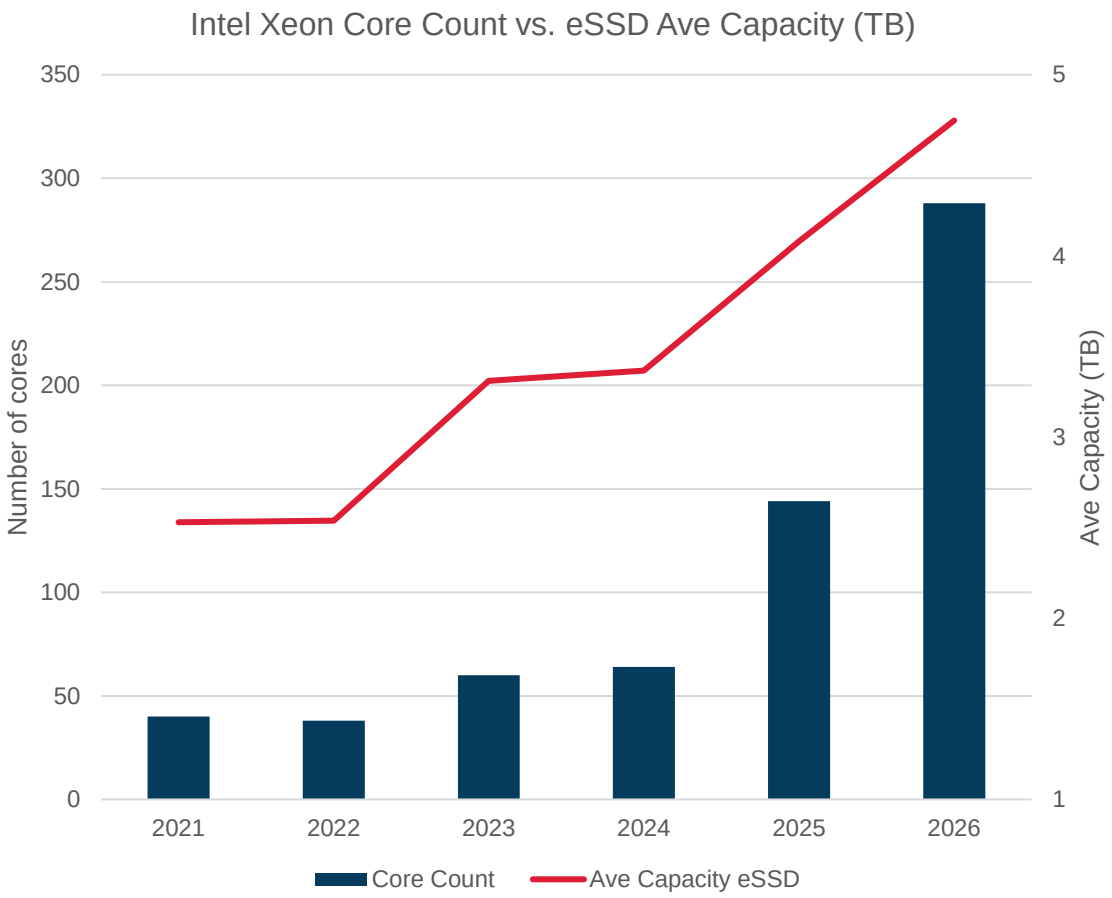




Standard SSD Workload Summary:
- Blue: 64KB Random Write
- Yellow: Log Structured 8 Writers 64KB
- Red: Log Structured 8 Writers 64KB with FDP

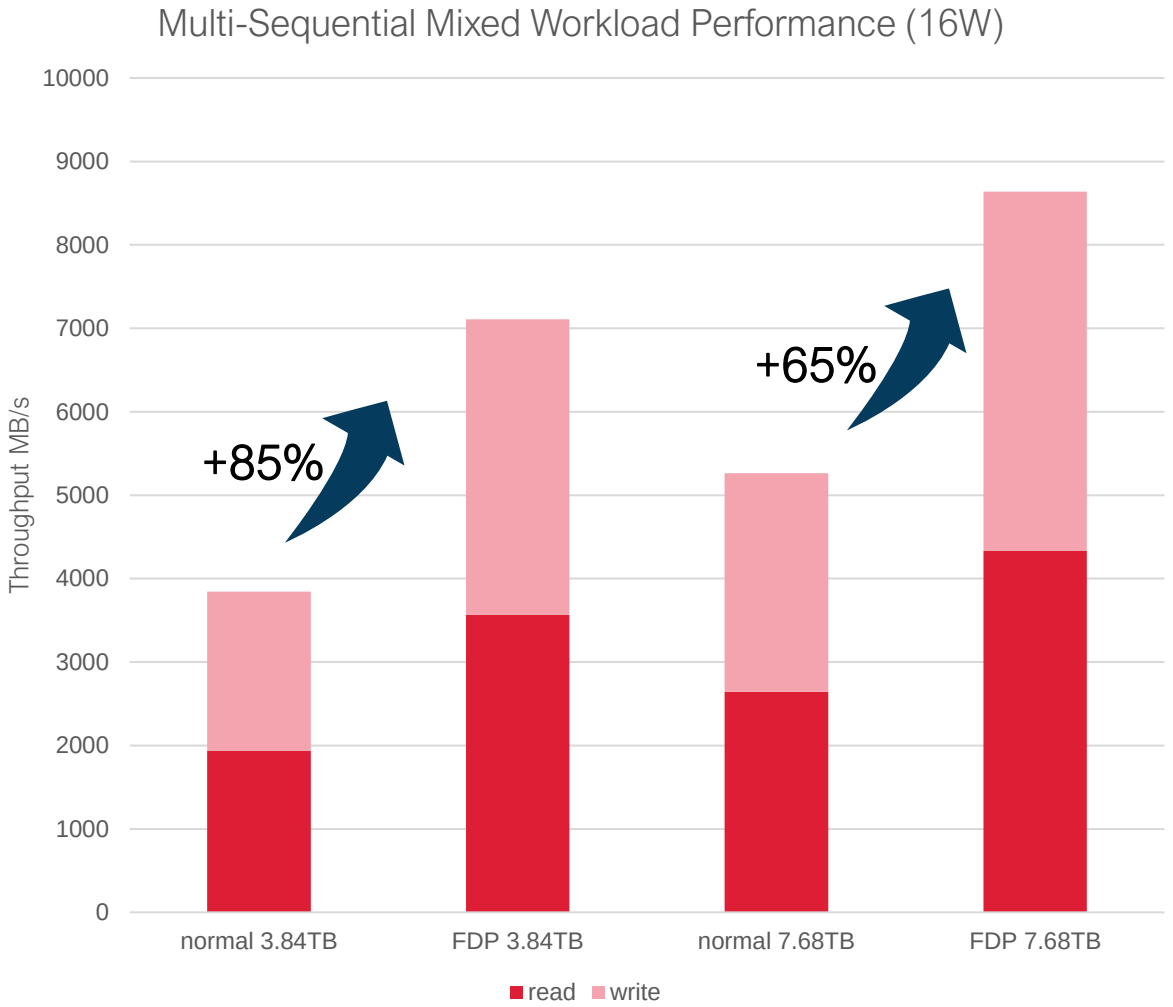
Example Workload Summary:
- FDP Reduces Wear: ~**2-3X**
  - SSD lasts 2-3X longer
- FDP Improved Write Throughput: ~**2-3X**
  - Improved Performance and Quality Of Service

FDP Improves Mixed Workload Performance



CPU core count driving need for higher capacity SSD

Source: ark.intel.com, Forward Insights Q'24



Drives designed for multi-tenant and cloud-native applications

Source: Measured by Fadu. 13th Gen Intel(R) Core(TM) i5-13600K, ASRock Z790, Ubuntu 22.04.2 LTS, kernel 6.2.10,fio-3.35

FDP Use Cases

Cloud-Native Environments: Multi-tenant, containerized workloads benefit greatly from FDP's ability to optimize garbage collection and enhance QoS.

Caching Applications: FDP can dramatically improve the efficiency and performance of caching solutions like CacheLib, one of the most popular hyperscale applications

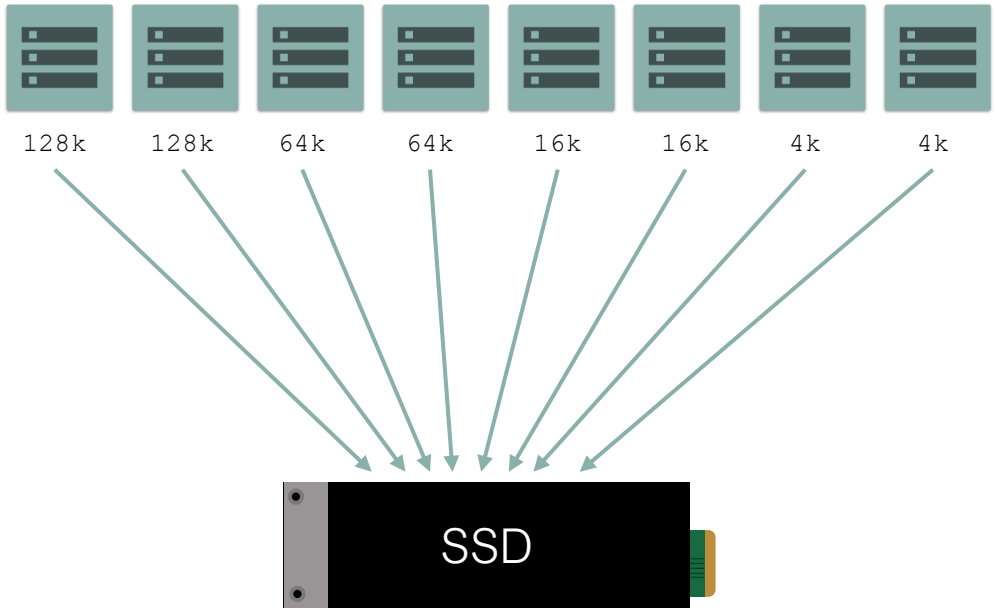
Mixed Workloads: FDP can improve QoS and performance in mixed workloads by separating them.

Write-Intensive Applications: FDP's ability to reduce the WAF is especially beneficial for write-heavy workloads with multiple write streams, extending an SSD's lifespan and ensuring more consistent performance.

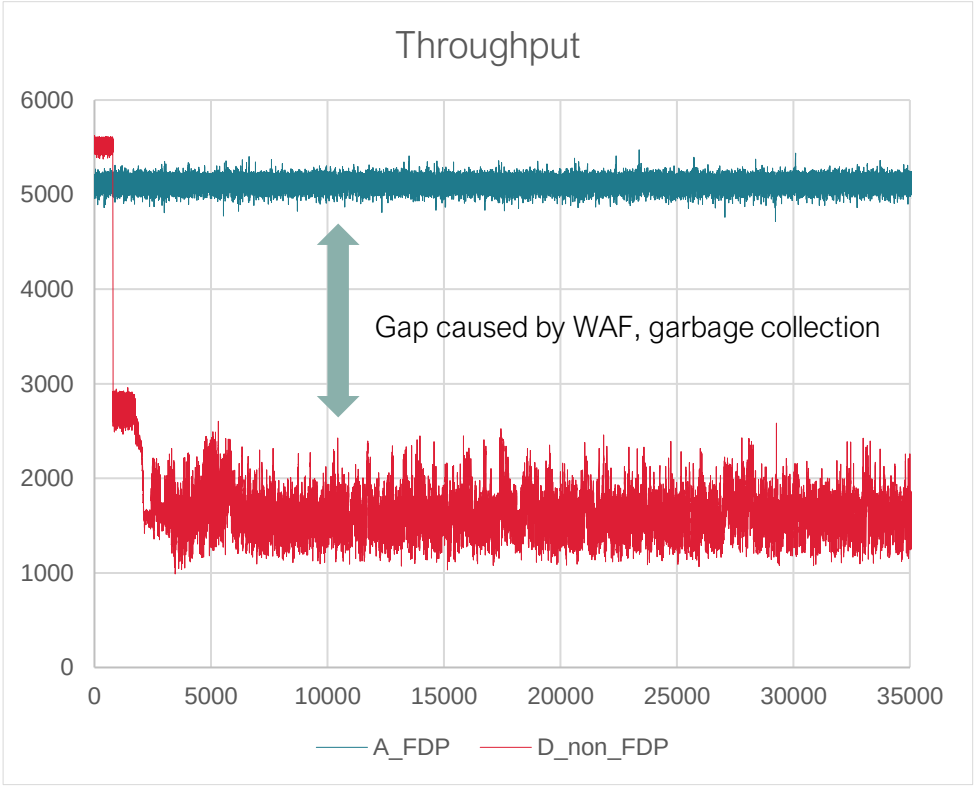
Multiple Namespaces in FDP

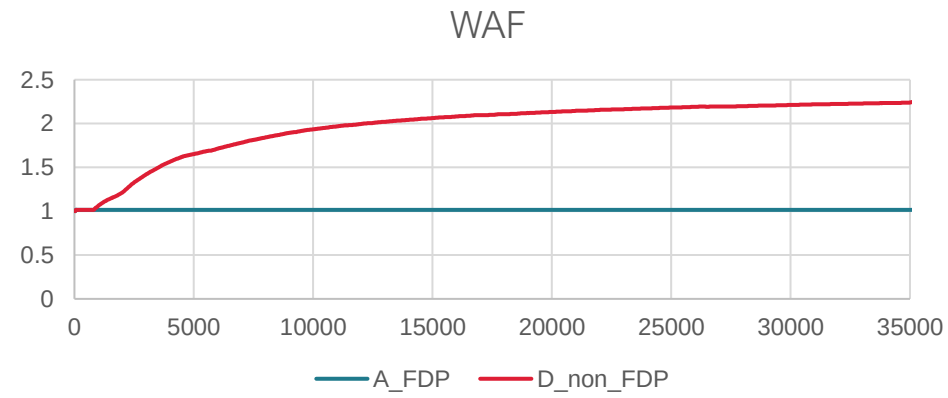
- **Reduced "Noisy Neighbor" Effect**: In multi-tenant environments, a heavy workload in one application can impact the performance of others workloads sharing the same SSD. Namespace isolation with FDP minimizes this interference, ensuring consistent performance for each application.
- **Optimized Garbage Collection**: FDP optimizes garbage collection within each RUH independently. This means that the GC process for one namespace won't disrupt the performance of another, resulting in smoother, more predictable performance overall.
- **Simplified Management**: Namespaces provide a logical way to group and manage data for different applications or tenants. This granular control simplifies administration and troubleshooting.
- **Performance sharing**: While some VMs are idle, others can still get the maximum SSD performance

FDP Improves Write Bandwidth



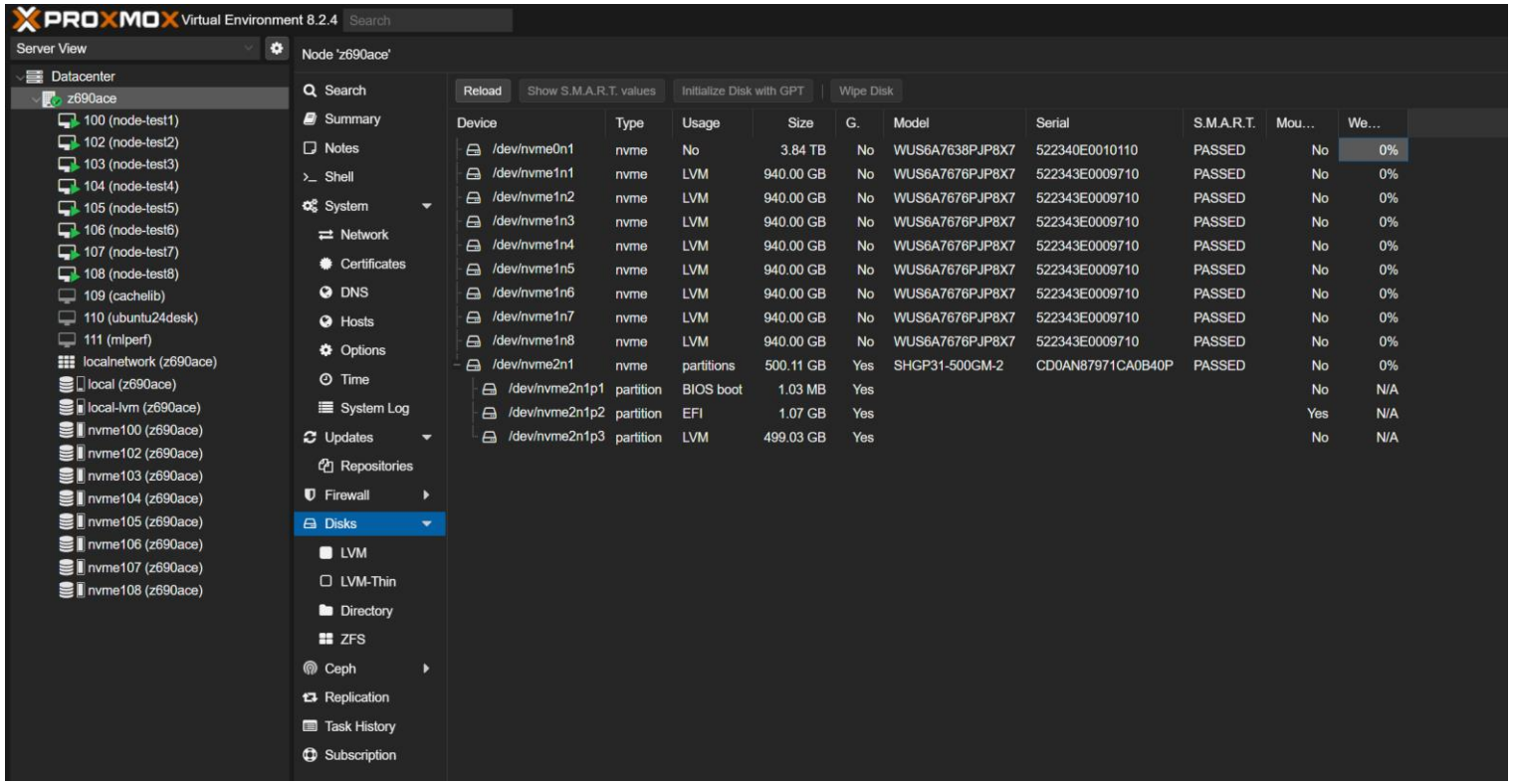
8 different write threads of different block sizes
Achieve WAF=1, full NAND bandwidth with FDP





Experiment - Multi-Namespace FDP (no SRIOV)

- With FDP enabled, each namespace gets its own RUH (up to 8 on this drive)
- No software development changes required
- Use Proxmox virtual environment to create 8 Ubuntu 22.04 VMs
- Create an LVM per namespace and attach to VM
- Ansible playbooks and scripts here
- https://github.com/jmhands/scripts/tree/main/fdp



First Experiment – 128k seq write to each VM

High velocity writes resulted in WAF = 1, even with 8 write streams

Calculate WAF

- Use OCP C0 Log Page for NAND writes
  - ✓Physical media units written
- Use SMART for host writes
  - ✓Data Units Written
- Record before and after test, calculate the delta
- Much easier when FDP is enabled! Just use nvme fdp stats

```
~/scripts/fdp# nvme fdp stats /dev/nvme1 -e 1
Host Bytes with Metadata Written (HBMW): 1195187723509760
Media Bytes with Metadata Written (MBMW): 1195187723509760
Media Bytes Erased (MBE): 1107474392678400
```

```bash
#!/bin/bash

# Function to extract data units written from smart log
extract_data_units_written() {
    # $1 is the log file path
    grep "Data Units Written" $1 | awk '{print $5}'  # This extracts the raw value of data units written
}

# Function to extract physical media units written from ocp log
extract_physical_media_written() {
    # $1 is the log file path
    grep "Physical media units written" $1 | awk '{print $NF}'  # This extracts the last number in the line, which is the bytes count
}

# Calculate Delta WAF
calculate_delta_waf() {
    # $1 is initial data units written, $2 is final data units written
    # $3 is initial physical media units written, $4 is final physical media units written
    delta_host_writes=$(echo "($2 - $1) * 512 * 1000" | bc)
    if [ "$delta_host_writes" -eq 0 ]; then
        echo "Error: Delta host writes is zero, cannot divide by zero."
        exit 1
    fi
    delta_nand_writes=$(echo "$4 - $3" | bc)
    echo "scale=4; $delta_nand_writes / $delta_host_writes" | bc  # Using bc to perform floating point division
}

# Paths to the log files
SMART_BEFORE_LOG="/tmp/smart_before.log"
SMART_AFTER_LOG="/tmp/smart_after.log"
OCP_BEFORE_LOG="/tmp/ocp_before.log"
OCP_AFTER_LOG="/tmp/ocp_after.log"

# Extracting data units written and physical media units written
data_units_written_before=$(extract_data_units_written $SMART_BEFORE_LOG)
data_units_written_after=$(extract_data_units_written $SMART_AFTER_LOG)
physical_media_written_before=$(extract_physical_media_written $OCP_BEFORE_LOG)
physical_media_written_after=$(extract_physical_media_written $OCP_AFTER_LOG)

# Check if extraction is successful
if [ -z "$data_units_written_before" ] || [ -z "$data_units_written_after" ]; then
    echo "Error: Failed to extract data units written from SMART logs."
    exit 1
fi

if [ -z "$physical_media_written_before" ] || [ -z "$physical_media_written_after" ]; then
    echo "Error: Failed to extract physical media units written from OCP logs."
    exit 1
fi

# Calculate Delta WAF
delta_waf=$(calculate_delta_waf $data_units_written_before $data_units_written_after $physical_media_written_before $physical_media_written_after)

# Output results
echo "Write Amplification Factor (WAF):"
```

Control – no FDP

- Create a single namespace of 7.68TB
- Format with LVM
- Give 940GB virtio drive (/dev/vdb) to each host
- Run playbook
  - ✓ Collect SMART
  - ✓ Kick off fio on remove VMs
  - ✓ Ensure it started correctly
  - ✓ Ensure it completed
  - ✓ Collect SMART
  - ✓ Parse output

Test	FDP	Write Block Size	Read BW (MB/s)	Read IOPS	Write BW (MB/s)	Write IOPS
vm1	no	4k	24.13	5892.39	22.67	5533.57
vm2	no	4k	24.12	5887.55	22.65	5530.06
vm3	no	16k	23.52	5741.43	88.28	5388.26
vm4	no	16k	23.45	5724.29	88.19	5382.99
vm5	no	64k	22.14	5405.87	332.67	5076.19
vm6	no	64k	22.07	5388.41	332.24	5069.65
vm7	no	128k	22.32	5448.61	669.88	5110.78
vm8	no	128k	21.79	5319.87	654.84	4996.08
Total			**183.53**	**44808.42**	**2211.43**	**42087.57**

```
:~/scripts/fdp# ./waf1.sh
Write Amplification Factor (WAF):
WAF = 1.7417
```

Setup for FDP enabled

- https://blogs.fadu.io/flexible-data-placement/
- Delete namespaces, enable FDP, check config, create 8 namespaces, attach to controller
- Create LVM on each namespace, assign one to each

```
for i in {1..8}; do nvme delete-ns -n $i /dev/nvme0; done
nvme set-feature /dev/nvme0 -f 0x1d --cdw12=1 -s
sudo nvme fdp config /dev/nvme0 -e 1
for i in {0..7}; do nvme create-ns -S 940G -C 940G -f 0 -n 1 -p $i /dev/nvme0; done
for i in {1..8}; do nvme attach-ns /dev/nvme0 -n $i -c 1; done
```

Node	Generic	SN	Model	Namespace	Usage	Format	FW Rev
/dev/nvme1n1	/dev/ng1n1	522343E0009710	WUS6A7676PJP8X7	0x1	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n2	/dev/ng1n2	522343E0009710	WUS6A7676PJP8X7	0x2	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n3	/dev/ng1n3	522343E0009710	WUS6A7676PJP8X7	0x3	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n4	/dev/ng1n4	522343E0009710	WUS6A7676PJP8X7	0x4	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n5	/dev/ng1n5	522343E0009710	WUS6A7676PJP8X7	0x5	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n6	/dev/ng1n6	522343E0009710	WUS6A7676PJP8X7	0x6	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n7	/dev/ng1n7	522343E0009710	WUS6A7676PJP8X7	0x7	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06
/dev/nvme1n8	/dev/ng1n8	522343E0009710	WUS6A7676PJP8X7	0x8	938.45 GB / 940.00 GB	4 KiB + 0 B	RG109W06

Enable FDP!


CPU Basic
100%
80%
60%
40%
20%
0%
12:45 13:00 13:15 13:30 13:45 14:00 14:15 14:30 14:45 15:00 15:15 15:30 15:45 16:00 16:15 16:30 16:45
Busy System
Busy User
Busy Iowait
Busy IRQs
Busy Other
Idle
Disk R/W Data
bytes read (-) / write (+)
1.50 GB/s
1 GB/s
500 MB/s
0 B/s
nvme1n3 - Written bytes 223 MB/s 221 MB/s 240 MB/s 213 MB/s
nvme1n4 - Written bytes 223 MB/s 220 MB/s 238 MB/s 210 MB/s
nvme1n5 - Written bytes 844 MB/s 832 MB/s 898 MB/s 796 MB/s
nvme1n6 - Written bytes 847 MB/s 842 MB/s 886 MB/s 798 MB/s
nvme1n7 - Written bytes 1.63 GB/s 1.62 GB/s 1.70 GB/s 1.53 GB/s
nvme1n8 - Written bytes 1.63 GB/s 1.64 GB/s 1.70 GB/s 1.57 GB/s
Disk IOps Completed
IO read (-) / write (+)
5K io/s
0 io/s
-5K io/s
-10K io/s
-15K io/s
-20K io/s
Name Mean Last * Max Min
dm-0 - Reads completed 0 io/s 0 io/s 0 io/s 0 io/s
dm-1 - Reads completed 0.000269 io/s 0 io/s 0.0667 io/s 0 io/s
dm-10 - Reads completed 0 io/s 0 io/s 0 io/s 0 io/s
dm-11 - Reads completed 0 io/s 0 io/s 0 io/s 0 io/s
Average Queue Size
aqu-sz
300
200
100
0
Name Mean Last * Max Min
dm-0 0 0 0 0
dm-1 0.000152 0.00100 0.00120 0
dm-10 0.0000349 0 0.00107 0
dm-11 0.0000304 0 0.000867 0
Disk Average Wait Time
time. read (-) / write (+)
20 ms
10 ms
0 s
-10 ms

FDP Enabled - Results

- Run same playbook, this time each VM has its own namespace, each with different RUH
- It works! Reduced WAF to 1
- Background 4k random read increased by **182%**
- Write bandwidth increased by **150%**
- Endurance increased by **170%**
- Applications don’t need to FDP aware to take advantage

Test	FDP	Write Block Size	Read BW (MB/s)	Read IOPS	Write BW (MB/s)	Write IOPS
vm1	yes	4k	67.18	16400.81	57.09	13938.97
vm2	yes	4k	67.10	16382.35	56.99	13914.59
vm3	yes	16k	65.64	16026.46	223.83	13661.51
vm4	yes	16k	65.75	16051.52	223.92	13666.82
vm5	yes	64k	63.46	15492.03	848.76	12951.09
vm6	yes	64k	63.50	15502.61	850.73	12981.18
vm7	yes	128k	61.97	15130.10	1634.38	12469.35
vm8	yes	128k	62.05	15149.12	1637.56	12493.56
Total			**516.65**	**126135.00**	**5533.27**	**106077.09**

```
~/scripts/fdp# nvme fdp stats /dev/nvme1 -e 1
Host Bytes with Metadata Written (HBMW): 119518772350976
Media Bytes with Metadata Written (MBMW): 119518772350976
Media Bytes Erased (MBE): 110747439267840

WAF=1!!
```

FADU

Creating Future