SNIA®

Introduction to Flexible Data Placement:

A New Era of Optimized Data Management

Storage Data Placement Technical Work Group

February 2026

Introduction

NVM Express® (NVMe®) provides the capabilities to optimize data placement when writing to media using Storage Data Placement (SDP) capabilities, i.e., Flexible Data Placement (FDP), Zoned Namespaces (ZNS), or the Streams directive, which allow for reduction of the Write Amplification Factor (WAF), improvements in performance in NVMe SSDs, and segregation of data to isolate WAF impact between different applications.

This white paper will address how an FDP NVMe SSD should be configured and used for optimal performance and endurance. This white paper will also cover differences between FDP, ZNS, and the Streams directive.

This paper provides:

- An overview of the FDP, ZNS, and Streams capabilities and how they are different from a conventional (e.g., non-SDP capable) NVMe SSD.
- A definition for WAF and why it is important to minimize.
- A comparison of FDP to other SDP capabilities defined by NVM Express.
- Recommendations for how the host software should use FDP to minimize WAF.
- An overview of the Linux software stack support for FDP.

Write Amplification Factor

A host writes in logical blocks to an SSD and the SSD controller writes those logical blocks to physical blocks. The SSD maintains a mapping table of Logical Block Addresses (LBAs) to Physical Block Addresses (PBAs). NAND requires that after a physical block is written with logical block data, that physical block must be erased before new logical block data can be written to that physical block. Logical block data that replaces existing logical block data requires writing that new logical block data to a different location in either the same physical block or a different physical block. This process marks the physical blocks previously associated with that logical block data as invalid whether in the same physical block or in a different physical block. NAND is internally organized in pages which are generally larger than a logical block, and data can only be written to it on a page-by-page basis. Physical blocks are required to be erased together. These physical blocks are also called erase blocks. Typically, these erase blocks are further grouped into superblocks (see “How conventional NVMe SSDs Work” section).

In a conventional NVMe SSD, host software has no control of how logical blocks are assigned to superblocks and therefore cannot assist in managing the use of superblocks. A physical block is required to be erased before logical block data can be written to pages that have been written with other logical block data since the last erase was performed.

If a physical block has some blocks marked as invalid and some blocks that contain logical block data, in order to utilize the capacity that is marked as invalid, logical block data that is stored in blocks within that physical block has to be moved to a different physical block in order to erase that physical block and prepare it to store new logical block data. This process is called

garbage collection. In a conventional NVMe SSD, there are no methods providing host-software direct control over how logical blocks are placed in the physical NAND, which could allow the host to reduce garbage collection.

The write amplification factor (WAF) is a measure of the number of writes of logical block data to physical blocks compared to the number of host writes of logical block data. This accounts for the storing of logical block data to a new physical block in order to erase an erase block. The simple WAF equation when considering logical blocks written to NAND is:

WAF = (Number of logical blocks written to NAND by the SSD) / (Number of logical blocks submitted to be written by the host)

If the SSD only writes host logical block data to physical blocks once for a write by the host of that logical block data, then WAF is one. If an SSD has to move logical block data from one physical block to another physical block for the purpose of garbage collection, then WAF is greater than one.

Ideally, the WAF should be as close to one as possible. However, it depends on the nature of the workload and the amount of stored data on SSD. A large WAF negatively impacts SSD performance and its usable life. Therefore, the SSD controller needs to be designed with careful considerations to handle WAF.

This white paper describes how FDP can be used to reduce WAF.

How conventional NVMe SSDs function

Figure 1 illustrates a high-level view of a conventional NVMe SSD. Erase blocks from each NAND die are grouped into a superblock. For example, erase block 0 from each NAND die across all of the NAND channels can form a single superblock. Prior to selecting a particular superblock for writing by the host, all of the erase blocks within the superblock are erased. Then, as the host writes logical blocks to namespaces on the SSD, those logical blocks are buffered by the controller until the amount of logical block data that the SSD requires to be written as a complete unit is accumulated. At that point, the SSD writes that accumulated logical block data to a superblock. The process of buffering logical blocks and then writing those logical blocks to a superblock continues until that superblock is written to capacity. A superblock may contain logical block data associated with different namespaces.

An SSD may manage the writes to erase blocks within the superblock in a manner that maximizes writing to NAND dies in parallel (e.g., maximizing write performance) while maintaining the expected advertised power.

Figure 1. Conventional NVMe SSD

The writing of logical block data to a superblock can be represented as logically adding the logical block data to previously written logical block data as shown in Figure 2. The write commands specify the namespace and the logical block address within that namespace that are written. Therefore, a superblock may contain logical blocks from different namespaces and the logical block addresses may be non-sequential within the superblock.

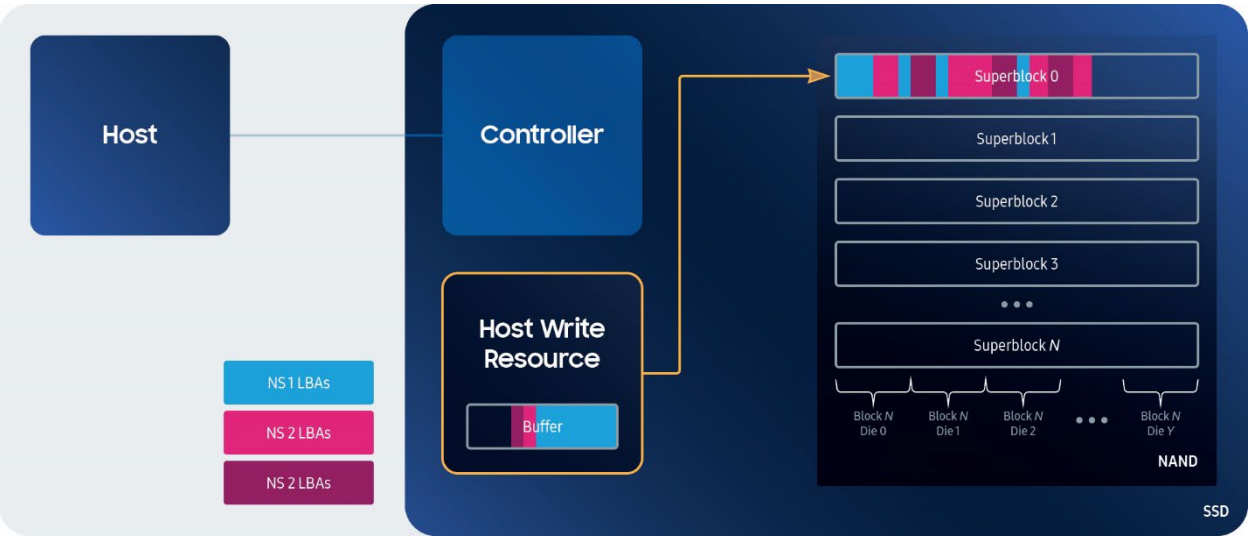

Figure 2. Logical view of a conventional NVMe SSD using superblocks

Once a superblock is written to capacity, the SSD selects a different superblock to continue writing host logical blocks. The SSD provides a pool of erased superblocks for use. Prior to erasing the NAND blocks in a superblock, if that superblock contains valid, previously written logical blocks, then those valid logical blocks are written to another superblock by the SSD (i.e., garbage collected) as illustrated in Figure 3.

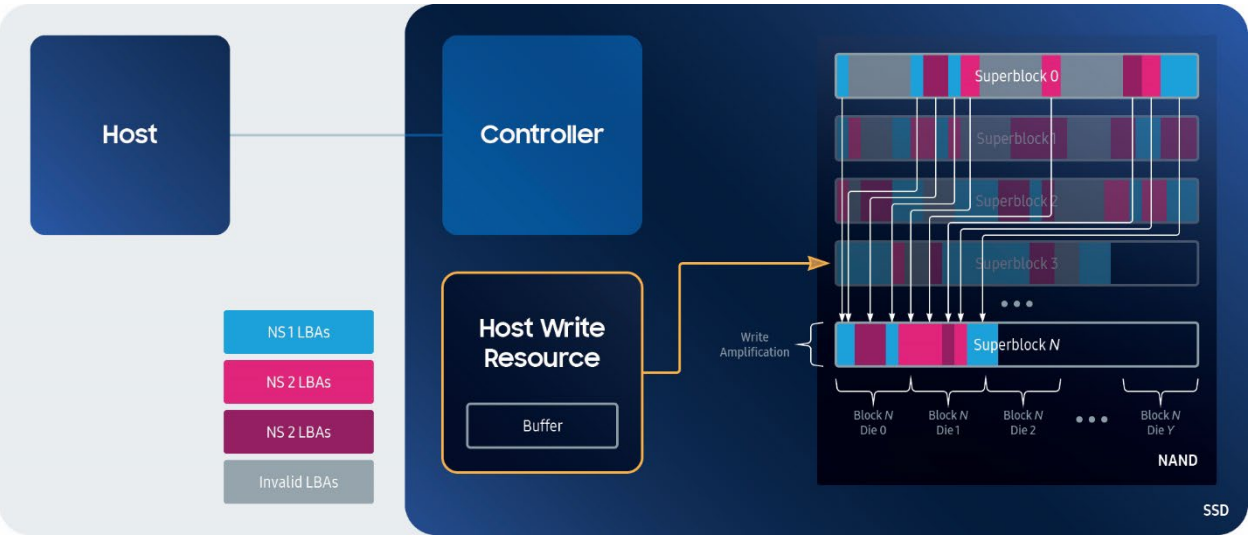

Figure 3. Host write resource switching to Logical view of a conventional NVMe SSD using superblocks

In order to manage superblock usage, an SSD keeps track of each logical block written to NAND. When the host writes a logical block to a namespace, that logical block data becomes the valid logical block data for that namespace, and any previously written instance of that logical block data in the NAND becomes invalid as that logical block data has been replaced with newer data in a new location. These invalid logical blocks may still exist in NAND blocks, but are not required to be moved as part of garbage collection.

There are mechanisms that mark logical block data as invalid. For logical blocks that are not being replaced with newer data but are no longer required by the host to be accessible (e.g., a file is deleted), the host may deallocate logical blocks (e.g., using the NVMe Dataset Management command) causing the SSD to mark the data associated with the deallocated logical blocks as invalid. The deletion of a namespace causes all logical block data associated with that namespace to become invalid. When data is marked as invalid, that data is not moved as part of garbage collection.

Since the host has no knowledge of the boundaries of blocks that are required to be erased as a unit, it also has no knowledge of what LBAs exist in those specific blocks. This lack of knowledge prevents the host from determining which blocks should be rewritten or deallocated to avoid garbage collection.

The theory of data placement is that the host can manage the logical blocks written to specific superblocks. This allows the host to attempt to prevent valid logical block data being in a superblock before the SSD erases the NAND blocks in that superblock for future writes, thereby minimizing garbage collection.

Flexible Data Placement (FDP)

The NVM Express Flexible Data Placement capability provides a mechanism that allows the host to control which logical blocks are written into a set of NAND blocks managed by the SSD, allowing potential reduction in WAF and potentially improving performance. This group of NAND blocks is called a Reclaim Unit. The host is able to write to more than one Reclaim Unit at a time allowing the host to isolate the data written to a Reclaim Unit per application or even within an application to separate data written that has different life cycles (e.g., hot data and cold data).

Figure 4 shows an overview of the FDP architecture that has the components shown in Table 1.

Table 1 FDP Architecture Components

• **Reclaim Unit (RU):**	A set of NAND blocks on an SSD where a host may write logical blocks.
• **Reclaim Group (RG):**	A collection of Reclaim Units.
• **Reclaim Unit Handle (RUH):**	A resource within the SSD to manage and buffer the logical blocks to write to a Reclaim Unit (i.e., a host write resource). A namespace is allowed access to one or more RUHs. If a namespace has access to more than one RUH, the host is allowed to write to multiple RUs concurrently. Multiple namespaces may use the same RUH.
• **Placement Handle:**	An index into the list of RUHs accessible by namespace that is defined at namespace creation. Host writes to that namespace can only access the RUHs in that list.
• **Placement Identifier**	The combination of the RG and the Placement Handle.
• **Endurance Group:**	A collection of NAND blocks that endurance is managed as a single unit with the intention that the NAND blocks wear out at the same time (e.g.., SSD life). In a typical SSD, there exists a single Endurance Group.
• **FDP Configuration**	A host selectable configuration that may be enabled that defines the Reclaim Unit size, number of RGs, the number of RUHs, and other information not addressed by this paper.
• **Data Placement Directive**	A directive in a write command that uses the Directive Specific (DSPEC) field in that command to identify the RUH and RG. The host is requesting that logical blocks for a write command be place in the Reclaim Unit referenced by the tuple of namespace, RG, and Placement Handle. If a write command does not specify this directive, then the SSD uses the RUH associated with Placement Handle 0h and selects the RG to place the logical blocks into.

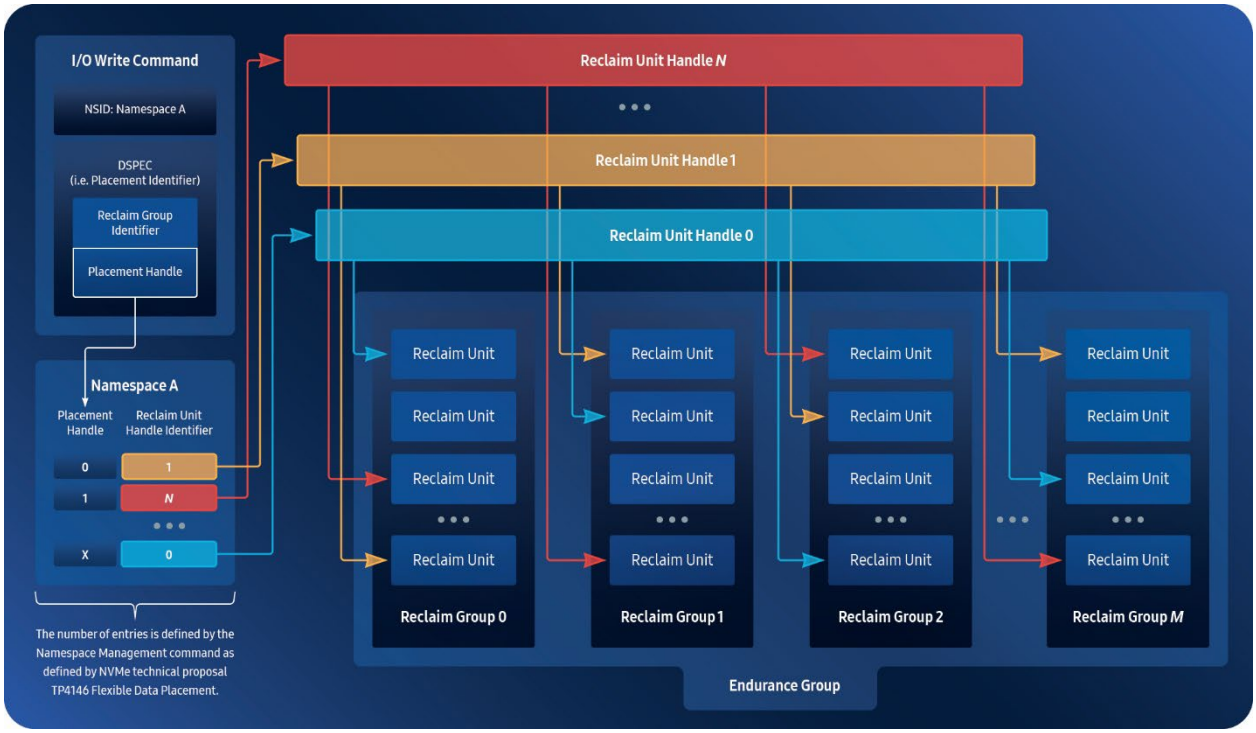

Figure 4. Multiple RG Flexible Data Placement (FDP) architecture

At namespace creation, the host specifies a list of RUHs accessible by that namespace. Each write command to that namespace that specifies the Data Placement Directive uses the Directive Specific (DSPEC) field to specify the Reclaim Unit where the logical blocks requested to be written. The Placement Identifier specifies an RG and a Placement Handle. The Reclaim Unit that is being referenced by the RUH in the RG is where the host is requesting to write the logical blocks.

A Reclaim Unit is one or more superblocks. In a FDP NVMe SSD implementation, a Reclaim Unit is one superblock. The host is provided the size of the Reclaim Unit and can manage which logical blocks are written to a Reclaim Unit.

Figure 5 shows an example where there is a single RG in which the SSD is managing the writing of logical blocks across NAND dies within each RU.

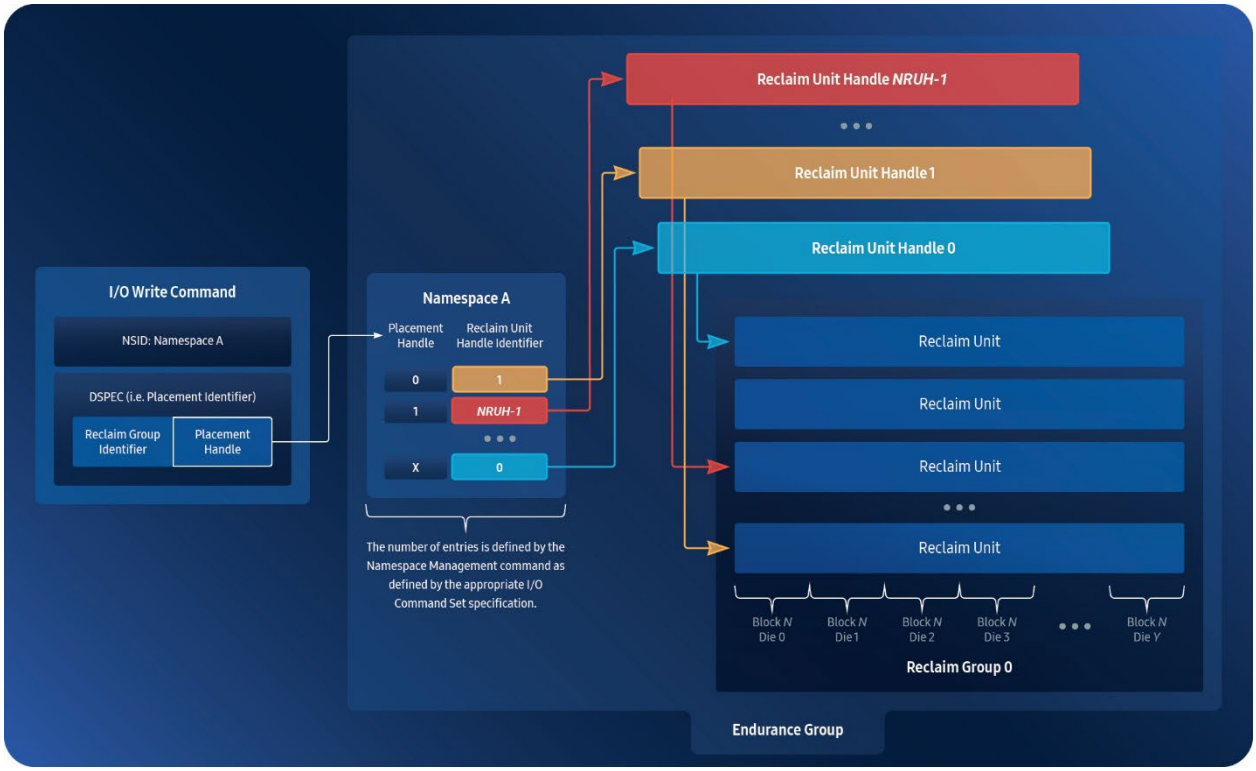

Figure 5. Single RG Flexible Data Placement architecture

Designing backward compatibility into the FDP architecture gives a host the option of planning how to modify the host software to take full advantage of FDP and fully manage the garbage collection of the SSD. For example, FDP can be enabled on an SSD and inserted into a server that is not FDP aware. In this environment, host writes to namespaces do not contain FDP placement information and the SSD selects the default RUH and RG for placement of the logical blocks. When FDP is enabled, the host software can be updated to allow software to utilize FDP.

WAF is reported for an entire SSD. To reduce WAF, a host may either deallocate the logical blocks written to a RU or re-write those logical blocks to a different RU such that the SSD does not have to move those logical blocks prior to erasing that RU. In either of these cases, all the previously written data is invalid, and the SSD would not require garbage collection on that Reclaim Unit. If the host has not written or deallocated all logical blocks that had been previously written to a Reclaim Unit, then the logical blocks that were not re-written or deallocated may be copied to another Reclaim Unit by the host to reduce garbage collection. In this case, there is no write amplification associated with the erase of that Reclaim Unit; but since the host rewrote the logical blocks there is an increase in the number of writes to the SSD for those logical blocks that were copied, thereby consuming more of the endurance of the SSD.

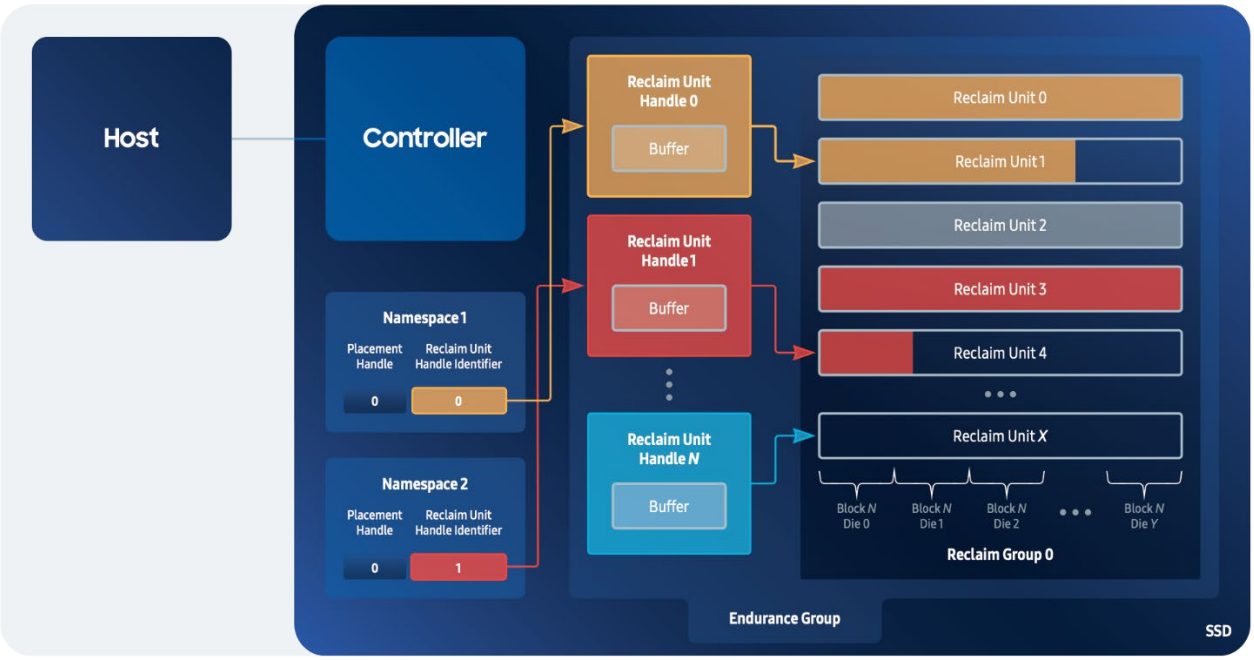

Figure 6. FDP Reducing WAF

Figure 6 is an illustration of two namespaces having access to two different RUHs which means that when the host initially writes logical blocks to those namespaces, the logical blocks for the different namespaces are isolated in different Reclaim Units. The separation of namespace logical blocks makes it easier for the host to track the LBAs to Reclaim Units.

In this example:

- Reclaim Unit 0 was previously written to capacity with logical blocks for Namespace 1.
- Reclaim Unit 1 is the current Reclaim Unit being written by RUH 0 and future writes to Namespace 1 are written to this Reclaim Unit.
- Reclaim Unit 2 was previously written to capacity with logical blocks for Namespace 2 but the host has either rewritten the logical blocks in another Reclaim Unit or has deallocated the logical blocks.
- Reclaim Unit 3 was previously written to capacity with logical blocks for Namespace 2.
- Reclaim Unit 4 is the current Reclaim Unit being written by RUH 1 and future writes to Namespace 2 are written to this Reclaim Unit.

Because the host causes all logical blocks in Reclaim Unit 2 to be invalid, when the SSD uses the NAND associated with Reclaim Unit 2 for future writes, the SSD has no valid logical blocks to copy to another Reclaim Unit and there is no WAF increase.

Logical blocks with similar data life should be written to the same Reclaim Unit so that all of the logical blocks become obsolete (i.e., invalid) at around the same time due to being re-written or deallocated.

Investing in obtaining the best SSD WAF

Benefits of FDP

FDP enables the host software stack to introduce WAF and performance benefits incrementally. This gives host designers an explicit trade-off between the desired WAF and the investment in the engineering effort.

FDP use cases

Refer to Figure 7[1] and Figure 8[1] for measured WAF with CacheLib at varying SSD utilizations. The SSDs were configured based on the recommendations in this white paper. Unlike conventional NVMe SSD benchmarking, there is no pre-conditioning using sequential and random writes. Such pre-conditioning would alter the results as the writes change the GC status and the availability of free blocks, causing the results to be different from what would be expected from an FDP based segregation. The only preconditioning that was performed before each test was trimming the SSD to bring the FTL tables to their original state, prior to the measurements. The preconditioning does not include any writes to the SSD.[1]

[1] *Allison, M., et al., "Towards Efficient Flash Caches with Emerging NVMe Flexible Data Placement SSDs", EuroSys '25, Twentieth European Conference on Computer Systems, 30 March 2025, pp. 1142 – 1160, https://dl.acm.org/doi/10.1145/3689031.3696091*

Figure 7. CacheLib results with 50% SSD utilizations

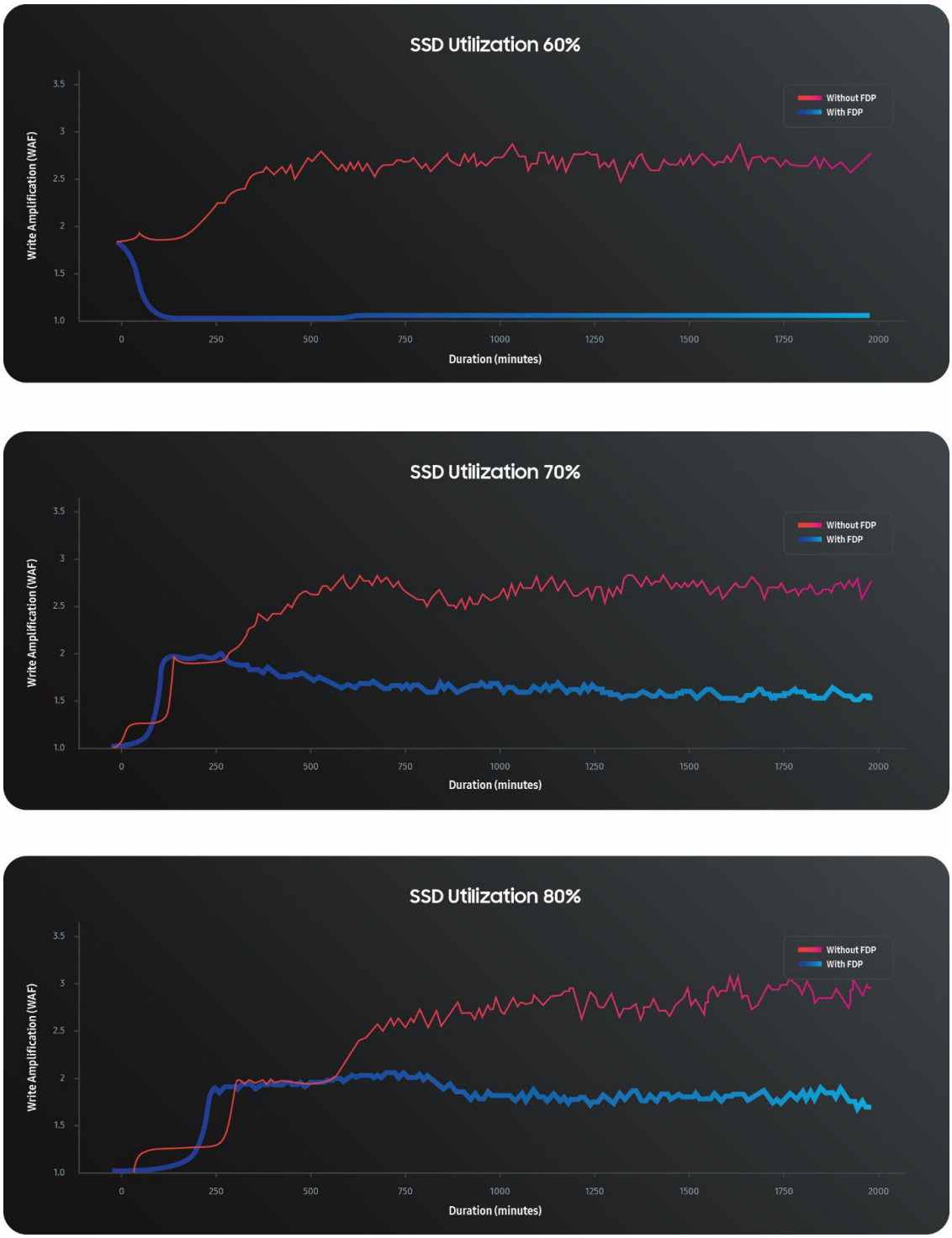

Figure 8. CacheLib results with various SSD utilizations

Computing Return on Investment (ROI) with FDP

Reducing the WAF of the SSD improves the endurance and performance of the SSD. To achieve acceptable levels of endurance or performance, storage systems commonly do not fully utilize the SSD Capacity, increasing the over-provisioning (OP) to lower WAF. The relationship between OP and WAF depends on the workload; however, the WAF of random write workloads is well understood and can be approximated using well-known mathematical methods as illustrated in Figure 9. The technique used to derive the graph is a Lambert equation method for a random write workload.

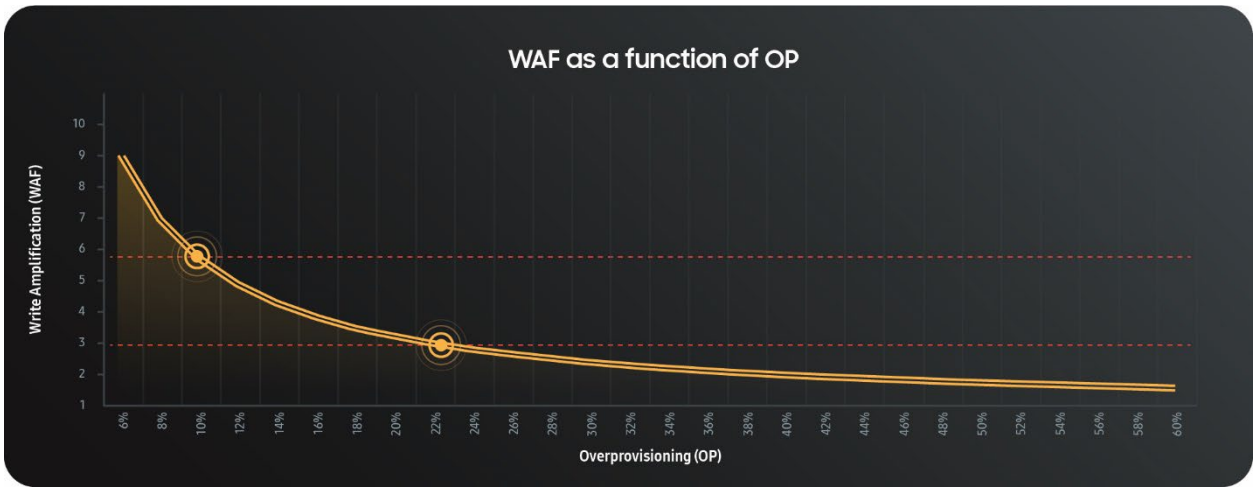

Figure 9. WAF as a function of OP

As an example, if a random workload is experiencing a WAF of 5.7 at an OP of 10% but endurance is required to be increased by 2X, WAF must be reduced by half. OP of approximately 23% is required to achieve WAF of 2.85. The additional 13% OP is lost SSD logical capacity.

OP = SSD physical capacity/logical capacity – 1

Another reason a host may increase OP is to improve the performance of random writes. A WAF > 1 requires SSD garbage collection. Garbage collection involves reading and writing valid logical blocks from a NAND block prior to erasing that NAND block for future writes. This internal work reduces the performance of the SSD. The reduction in random write performance can be approximated with a simple equation that takes into account the extra internal garbage collection's read and write traffic.

Random write performance at WAF of N = RWP_N

$RWP_N = RWP_1/(1 + (WAF-1)*1.1)$

Furthermore, random mixed read/write workloads can be modeled under the assumption that available SSD resources, such as NAND die, are equally available to be utilized by both reads and writes. Reads do not require write resources and therefore do not impact WAF. Therefore, the performance impact of WAF increases as the percent of writes increases.

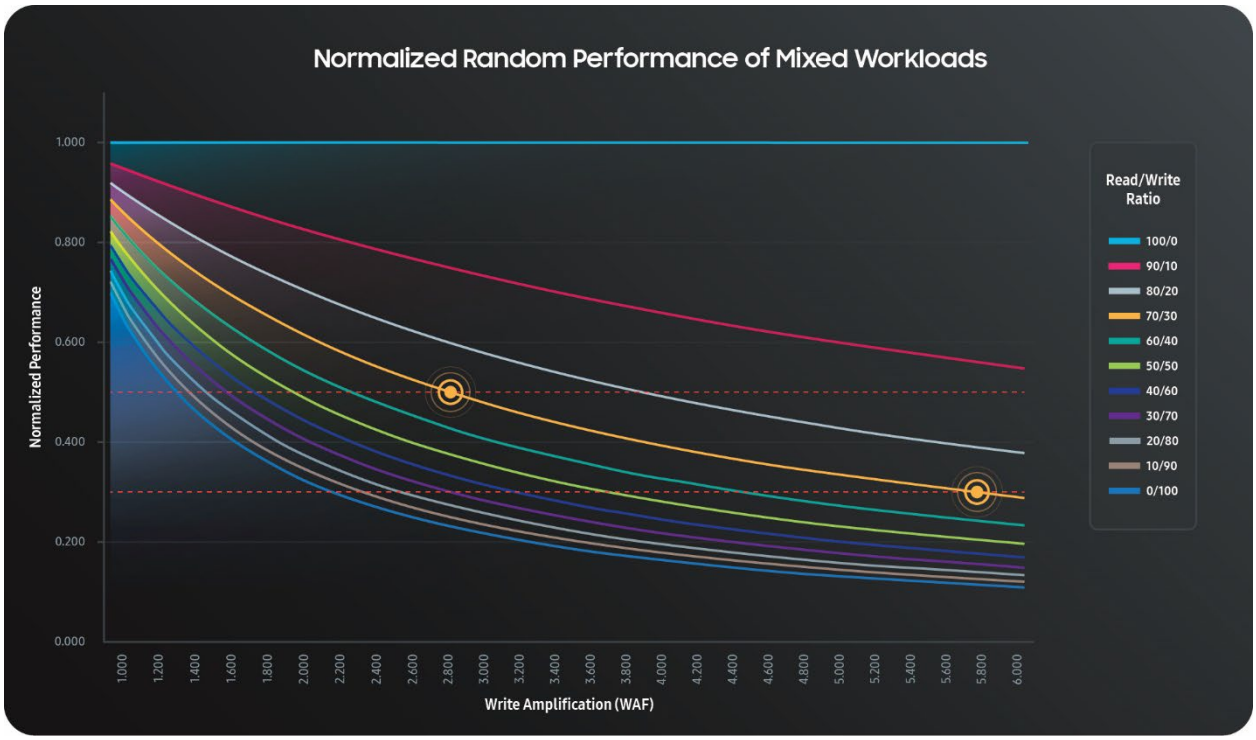

Figure 10. Normalized Random Performance

Repeating the example above and using Figure 10, if a workload is encountering a WAF of 5.7 and assuming a read/write ratio of 70/30, its normalized performance is 0.3. Reducing WAF to 2.85 increases normalized performance to 0.5. An approximate 67% increase in performance but at the same expense as above; OP increased from 10% to 23%.

FDP data placement is a way to improve the endurance and performance of a SSD. How much WAF can be lowered by adding support in a host is a function of the host workload; therefore each workload may have a different solution when modifying the software to support FDP. As one example use case, CacheLib was utilized to assess the improvements to WAF. CacheLib is a C++ library for accessing and managing cache data. It is a thread-safe API that enables

developers to build and customize scalable, concurrent caches. It is targeted at applications that dedicate gigabytes of memory to cache information[2].

CacheLib issues both metadata writes and user data writes. Example CacheLib workloads were converted to support FDP by using a dedicated RUH for metadata and a different dedicated RUH for user data. This conversion required minimal effort for FDP since CacheLib inherently sequentially writes a large portion of its data. Other hosts may need to alter the nature of how data is written and trimmed/unmapped to maximize benefits of FDP. For this particular example, no other host software changes were required. Metadata is written randomly while user data is written sequentially. The sequentially written data is overwritten thus encounters a WAF of 1. The SSD OP is applied to the random data by the SSD since it is not being used for the sequential data. Since the capacity of the random data written is relatively small, a few percent of SSD capacity, the SSD OP is able to achieve a WAF of close to 1 even for the random data. Converting to FDP dramatically improved the observed SSD WAF. Note that the host still encounters non-SSD based WAF for its internal log-structured write methods. FDP did not impact the non-SSD based WAF.

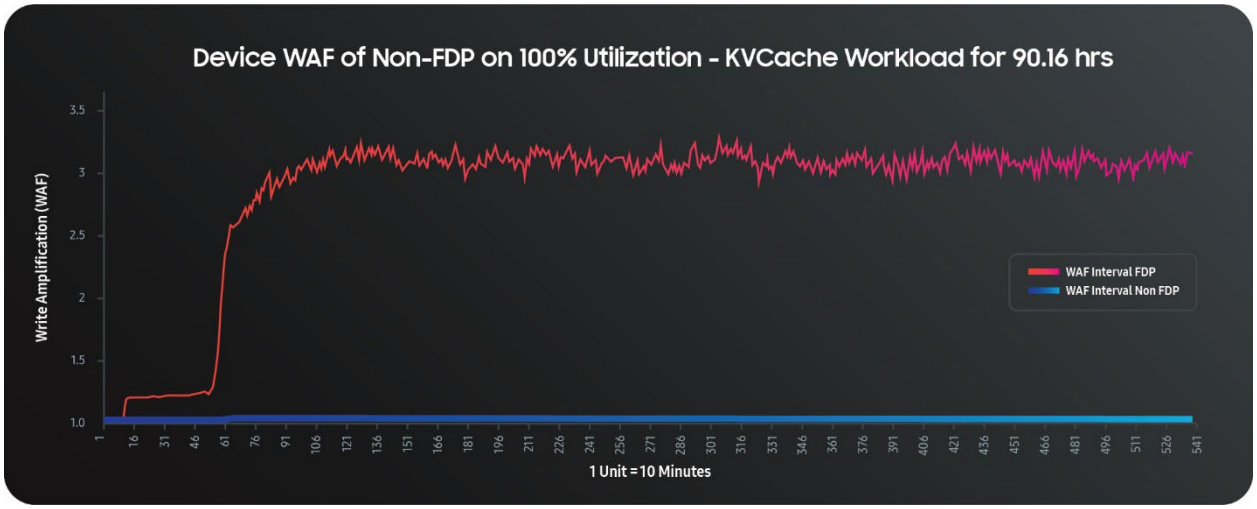

Figure 11. CacheLib WAF

Figure 11 illustrates the observed WAF improvement with FDP on CacheLib. SSD WAF dramatically improved from approximately 3 to 1. The endurance improvement directly scales with the improvement in WAF. Performance improvements are a function of the read/write ratio. This workload is dominated by approximately 90% reads. Using the translation provided by Figure 10 normalized performance increased from 0.74 to 0.96. If this increased performance is

[2] CacheLib. *Cachelib.org*, cachelib.org/. Accessed 18 Oct. 2023.

SNIA®

consumed then the lifetime of the SSD is impacted by the increased write activity. FDP conversion enabled increased performance of (0.96/0.74 = 1.3) and increased life of 3/1.3 = 2.3. Note: increased life is based on DWPD rather than warranty period. Note that the performance improvement estimates only include the impact of WAF. Vendor specific FDP implementation design decisions may impact performance.

Other workloads and hosts are bound to exhibit different gains and value propositions. CacheLib analysis of FDP indicated very significant value.

FDP and the Linux Software Stack

FDP is supported in several parts of the Linux ecosystem as shown in Figure 12.

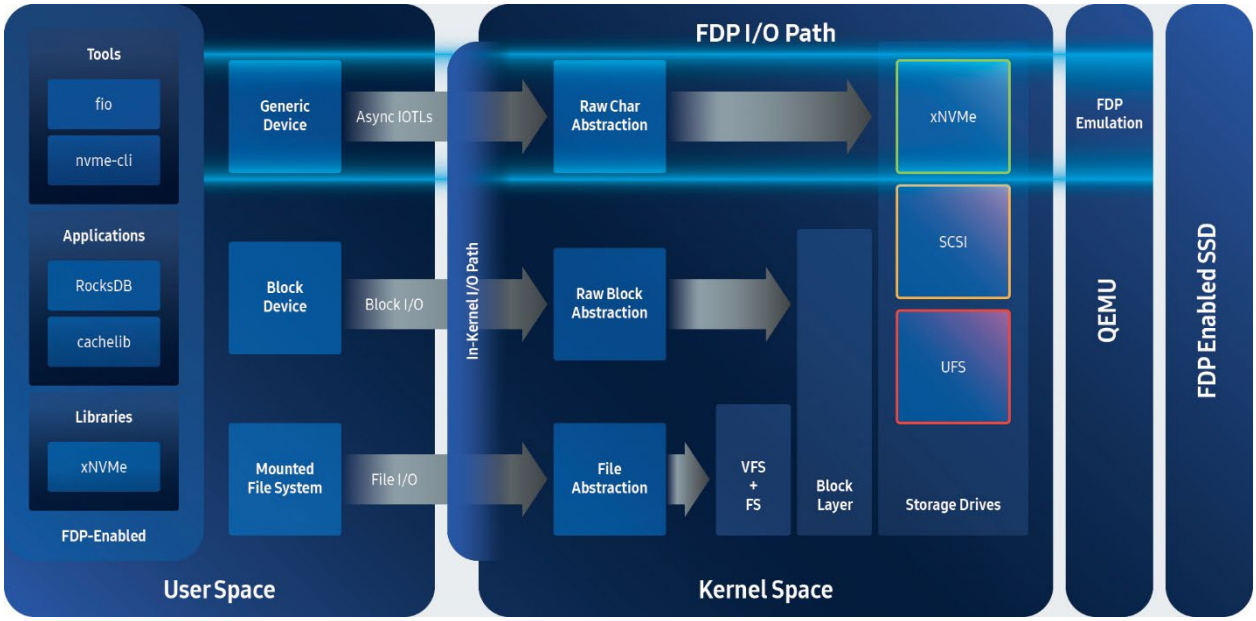

Figure 12. FDP Support in the Linux Ecosystem

In the Linux kernel, FDP leverages the I/O Passthru interface [3], an asynchronous path based on io_uring to the NVMe generic character device that allows user-space applications to directly access the Kernel NVMe driver, bypassing the block layer. This means that applications can leverage an end-to-end architecture similar to SPDK [8], but using the in-kernel NVMe driver instead. This simplifies the host stack as it removes the need to deploy and manage user-space

[3] "SDC2021: Enabling Asynchronous I/O Passthru in NVMe-Native Applications." *Www.youtube.com*, www.youtube.com/watch?v=mtiQlPZsw4w. Accessed 18 Oct. 2023.

drivers, delegating this responsibility to the operating system. I/O Passthru is fully supported in the Linux kernel since version 6.2.

In user-space, both xNVMe[4] and SPDK[5] provide full support for FDP. xNVMe enables FDP in different storage paths by leveraging the SPDK NVMe driver and the I/O Passthru. The ability to connect FDP-enabled SSDs with standard storage interfaces makes adoption easy for application developers and storage infrastructure architects. FDP is also present in widely used tools such as fio and nvme-cli, which enable respectively testing and management.

From an application point of view, FDP is supported in CacheLib. Here, FDP allows to separate CacheLib's two caches: BigHash – the cache for small items, and BlockCache – the cache for big items. By separating these two caches, CacheLib can achieve better WAF[6]. The changes to CacheLib are minimal and are contained within CacheLib itself – no changes are needed to the applications on top of CacheLib.

From an emulation perspective, FDP is fully supported as of QEMU[7] version 8.0. Using QEMU, operating system and application developers are free to develop in advance of FDP-enabled SSDs becoming widely available.

Configuring FDP

Overview

FDP offers a high degree of flexibility. There is no host control of Physical Placement of Reclaim Units. The following guidelines for using FDP help ensure consistency and compatibility among hosts and SSDs. By constraining FDP SSD configurations, interoperability and consistent performance may be achieved between host systems and SSD vendors. These configuration recommendations maintain non-FDP performance for non-FDP-aware hosts, and for FDP-aware hosts that have mixed workloads which may not be fully optimized for FDP.

[4] "XNVMe: Cross-Platform Libraries and Tools for NVMe Devices." *GitHub*, 29 Aug. 2023, github.com/OpenMPDK/xNVMe.
[5] "Storage Performance Development Kit." *GitHub*, 29 Aug. 2023, github.com/spdk/spdk.
[6] "Overview: A Random Walk down the Cache Library | CacheLib." *Cachelib.org*, cachelib.org/docs/Cache_Library_Architecture_Guide/overview_a_random_walk/. Accessed 18 Oct. 2023.
[7] https://gitlab.com/qemu-project/qemu

Recommendations

The following are a list of recommendations for FDP configuration:

- Single RG for entire SSD storage capacity (host configurable may be SSD limited)
- RU size equal to superblock size (SSD characteristic)
- Limit the number of RUHs to a maximum of 8 to 32 (host configurable but may be SSD limited)
- Do not provide for physical placement constraints (SSD characteristic)

Single RG

We recommend using a single RG that includes the entire SSD storage capacity.

- **Background**
  - Each RUH becomes unique within an RG resulting in a multiplication of RUH resources by the number of RGs.
  - The controller must isolate garbage collection (GC) within an RG.
- **Reasoning**
  - Using a single RG leverages existing NAND management algorithms that address performance and endurance.
  - Multiple RGs require hosts to manage parallelism across the multiple RG to match the write performance of a non-FDP enabled SSD.
  - Persistently Isolated RUH may provide benefits similar to those of additional RG in allowing isolation in both placement and for garbage collection.
  - Supporting multiple RGs reduces the number of RUHs that can be supported.

Superblock Sized RU

We recommend setting the Reclaim Unit (RU) Size to match the superblock size of the SSD. Vendors should determine the superblock size based on their SSD specifications and desired WAF and write performance.

- **Background**
  - An RU in NVMe FDP is a set of data that will be reclaimed together as a unit.
  - A superblock in an SSD is a set of blocks that are garbage collected together before being erased and reused. Often a superblock may be composed of one erase block from every plane of every NAND die. This definition of a superblock is common because it enables maximum parallelism for maximum write performance from an SSD.
  - Selection of an appropriate superblock size can help with WAF due to the host extent not being aligned to RU size. With large host extents RU size can be optimized to tradeoff between WAF and write performance.

  - An RU can consist of multiple superblocks, but a superblock cannot contain multiple RUs.

- **Reasoning**
  - For RU Sizes less than or equal to a superblock, RU Size is directly proportional to performance for any given RUH, since RU size creates a limit on the number of active die writes per RUH.
  - Small RU require hosts to manage parallelism across multiple RUH to match the write performance of a non-FDP enabled SSD.

Limit Number of RUH

Typical SSDs provide between 8 and 32 RUHs. A host should be developed to operate within this limit based on the maximum RUH count indicated by the SSD. The creation of a namespace associates up to 128 PHs with RUHs and this association is limited by the maximum number of RUHs that the SSD supports.

- **Background**
  - The number of RUH is limited by controller resources such as isolated stream support, open superblocks, open superblock time, and write buffering capacity.
  - Different NAND technologies will require different amounts of buffering. For example, QLC NAND requires more buffer than TLC NAND, and QLC NAND has a lower write speed while simultaneously reducing the amount of time until the superblock must be filled to ensure data integrity. The lower write speed paired with the smaller open superblock times mean that some SSD configurations must have a lower RUH count in order to have sufficient write bandwidth to fill each RUH prior to the required closure time of the superblock to protect data integrity.
- **Reasoning**
  - Current products in the industry align around supporting 16-32 RUHs. There are no significant technology trends indicating higher numbers would become standard practice.
  - Smaller capacity, lower performance, and QLC SSDs may support only 8 RUH.
  - The use cases requiring very large RUH counts targets specialized deployments outside of the guidance of our recommended configurations.

Physical Placement Constraints

FDP does not provide any ability to define physical placement characteristics. Our recommendation is that there not be any attempt to enhance SSD requirements to provide this capability.

Attempting to constrain physical placement per RG causes the detrimental characteristics listed here.

- **Background**
  - The NVMe specification does not define mechanisms associating an RG or an RUH with specific die on an SSD.
  - The NVMe specification only defines Namespaces as having a capacity.
- **Reasoning**
  - Allowing the SSD to select physical placement maximizes conventional host SW infrastructure, and permits the SSD to optimize performance and endurance decisions.
  - The lack of standardization for a host writing more to an RG than the capacity of a set of die will result in undefined and inconsistent behavior for this specification compliant case.
  - Performance is dependent on host workloads since SSDs cannot utilize parallelism between dies to address write throughput, sequential read performance as well as channel and die conflicts
  - FDP restricts the SSD from managing endurance across the entire SSD capacity, and the endurance impact of writes is limited to specific dies. The host would be responsible for variance of endurance between dies.

Considerations for Implementations in Heterogeneous SSD Environments

In addition to the recommendations listed above for mixed implementations (non-FDP- and FDP-aware hosts), you may have a host/system environment that is fully FDP-aware but may include multiple SSD device configurations (e.g., capacities, superblock sizes, different suppliers). In this scenario, the above recommendations can certainly be successfully applied, but it is also possible to make finer customizations that reflect the varying characteristics of the installed SSDs and not have to consider non-FDP-aware application impacts.

The recommendations above could be modified to optimize to the characteristics of the fleet, or to where low WAF is most important for application performance or high-use SSDs.

This improved flexibility with FDP settings creates opportunities to better optimize for WAF.

Other SDP Capabilities

ZNS

In ZNS, a namespace that is associated with the Zoned Namespace Command Set is divided into zones. Zones in a zoned namespace are fixed size entities where logical blocks are written sequentially and that are managed as a unit. A zoned namespace allows collaboration between

a host and an SSD for data placement. This collaboration may include isolating data to specific zones or utilizing specific zones for data with different life cycles (e.g., hot and cold data). The Zoned Namespace Command Set Specification shares commonality with other zoned device models (i.e., SCSI ZBC, ATA ZAC, host-managed SMR HDDs, and Zoned UFS). This commonality enables a singular software ecosystem for devices that share this functionality. See the SNIA Zoned Storage Models for detailed recommendations on configuring ZNS devices.

To use zoned namespaces for data placement:

- an SSD must support the Zoned Namespace Command Set; and
- host software must support the zoned namespace functionality through support in the host file system and applications.

Streams

In the Streams directive, the host can indicate to the controller that the specified user data (e.g., logical blocks in a write command) are part of one group of associated data. This information may be used by the controller to store related data in associated locations or for other performance enhancements. This information is a hint to the controller that may or may not be honored.

Comparing FDP to Streams and ZNS

Table 1 provides a side-to-side comparison that shows that FDP is the most flexible for allowing host software to achieve the best WAF, based on investment in adapting host software. The following paragraphs provide details of the rows highlighted in orange.

FDP was architected to complete a host-issued write command with a Data Placement Directive that does not conform to the defined protocol and to provide feedback to the host that such a write did not conform to the protocol. For example, if a host issues a write command that the identified RUH and RG in the Data Placement Directive is invalid, then the write is completed by the SSD and is placed in a Reclaim Unit accessible to the namespace selected by the controller and the event is logged in an FDP event, if enabled by the host.

If the host issues a write to an invalid Stream the command is aborted.

If a host issues a write command to a ZNS namespace that crosses a zone boundary, then the write command is aborted.

The ability for namespaces to share access to the same RUH is important for FDP for a number of reasons. An example of namespaces sharing access to the same RUH include ensuring backwards compatibility with host software that is not FDP aware. This allows an SSD that has

FDP enabled to be plugged into a server that is not aware of FDP and that server is able to write logical blocks without getting an error.

A Reclaim Unit is similar to a zone in an NVMe Zoned Namespaces (ZNS) with the exception that:

- ZNS requires a strict association of LBAs to zones, while there is no association of an LBA to a specific Reclaim Unit in FDP; and
- In FDP, a host may:
  - Write random LBAs into a Reclaim Unit;
  - Write the same LBA more than once into the same Reclaim Unit;
  - Write the same LBA into a different Reclaim Unit; and
  - If different namespaces share access to the same RUH, then logical blocks from those namespaces may be written to the same Reclaim Unit.

Both Streams and FDP enabled SSDs can be removed from a server and plugged into an older server and would just work. The older server is able to create namespaces and write to namespaces without any errors. That is not true with ZNS. A ZNS SSD can only operate in a server that understands ZNS. FDP is well-suited for retrofitting existing applications that follow software data placement policies which can then be communicated to the SSD in exchange for a WAF benefit. FDP makes it possible to leverage host data-separation to improve end-to-end WAF incrementally.

For Streams, write commands may utilize the Stream Directive to indicate the write resources to use for writing logical blocks. In this case, the set of streams is analogous to the set of RUHs. The NVMe Streams capability has no mechanism for the host to determine the current alignment. For ZNS, the number of active zones is analogous to the set of RUHs.

From a hosts perspective, designing a NVMe-based data placement solution to reduce WAF presents trade-offs, as there are several alternatives in the standard. Specifically, the comparison between Streams, FDP, and ZNS becomes particularly interesting.

Table 2 Comparison between Streams, FDP, and ZNS

Streams	Flexible Data Placement	Zoned Namespaces
Error on non-conforming writes	**Non-conforming writes are logged**	**Error on non-conforming writes**
Known alignment only after format	Commands available to host to stay aligned	Always aligned by interface rules
WAF = 1 achievable without feedback	WAF = 1 achievable without feedback	WAF = 1 guaranteed
Backwards compatible	**Backwards compatible**	**Not backwards compatible**
No information that the controller moved logical blocks	Post log event that the controller moved logical blocks	Notification to host to move logical blocks
Placement identifier not tied to LBA	Placement identifier not tied to LBA	Placement identifier is the LBA
Stream Granularity Size (SGS)	Reclaim Units	Zones
SGS capacity = SGS size	Reclaim Unit capacity = Reclaim Unit size	Zone capacity <= zone size
No host metadata per SGS	No host metadata per Reclaim Unit	Host metadata per zone
Namespace capacity defines # SGS	Endurance Group capacity defines # Reclaim Units	Namespace capacity defines # zones
Sequential, random, and over write	Sequential, random, and over write	Sequential write
Writes allowed to cross boundaries	Writes allowed to cross boundaries	Writes not allowed across boundaries
QD > 1: LBA known at write submission	QD > 1: LBA known at write submission	QD > 1: LBA known at write completion (zone append cmd)
Stream written by a single namespace	Reclaim Unit written by one or more namespaces	Zone written by a single namespace
API is stateless	API is stateless	API is stateful
Requires full FTL table	Requires full FTL table	Full FTL table not required
Dynamic write resource allocation	Static write resource allocation	Dynamic write resource allocation

In Conclusion

This paper discusses FDP architecture and presents a compelling case on how FDP can help to improve the performance and endurance of an SSD. An example of CacheLib is demonstrated to show how FDP can help to achieve WAF of a SSD approaching 1.

Disclaimer: The performance analysis shown in this paper only considers the impact of WAF and does not include any specific FDP implementation choices.

About SNIA

SNIA is a not-for-profit global organization made up of corporations, universities, startups, and individuals. The members collaborate to develop and promote vendor-neutral architectures, standards, and education for management, movement, and security for technologies related to handling and optimizing data. SNIA focuses on the transport, storage, acceleration, format, protection, and optimization of infrastructure for data.

SNIA
5201 Great America Parkway, Suite 320, Santa Clara, CA, 95054
Phone:719-694-1380 • Fax: 719-694-1385 • www.snia.org