SAMSUNG

Getting Started with Flexible Data Placement (FDP)

Global Open-ecoSystem Team (GOST)

Authors:
Arun George, Vikash Kumar, Roshan Nair, Vivek Shah, Klaus Jensen,
Ankit Kumar, Kanchan Joshi, Kundan Kumar, Nitesh Shetty

Table of Contents

SAMSUNG

1. Introduction to FDP

NVM Express® (NVMe®) released the ratified technical proposal TP4146a Flexible Data Placement (FDP) that defines a new method for placing logical blocks into the non-volatile storage in an effort to reduce Write Amplification Factor (WAF) by the SSD. This provides a new data placement mechanism that allows the host to control which logical blocks are written into a set of NAND blocks managed by the SSD called a Reclaim Unit. Moreover, 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 (referred to as hot and cold data).

The concepts of FDP are explained in the published whitepaper, "Introduction to Flexible Data Placement: A New Era of Optimized Data Management". It explains the device internals of the FDP technology, and also compares FDP with previous data placement technologies such as Zoned Name Space (ZNS) and Streams.

The key ideas of FDP technology can be summarized as follows:

1. Data Segregation: The host system can separate user data of different longevity patterns into different NAND blocks of the SSD device. This feature is similar to NVMe-Streams in concept, but comes with fewer restrictions such as not being namespace-specific. FDP further extends the 'Data Segregation' feature in such a way that the host is able to separate the data into different NAND dies.

2. Data Alignment: The host is able to know the SSD superblock boundaries and align the user data accordingly. This feature is useful when the host wants to align different objects of similar longevity type into different NAND media for eventual efficient de-allocation.

3. Device Feedback: Log messages defined in the FDP spec allow the Host to know details about the data placement in the Device. For example, the Host will be able to determine the frequency of garbage collection (GC) occurrences, Super Block re-assignments, etc.

The following diagrams from the whitepaper summarize the FDP proposal and compare an FDP-enabled SSD architecture to a conventional SSD one, illustrating how FDP helps the host to achieve a reduction in device Write Amplification Factor.

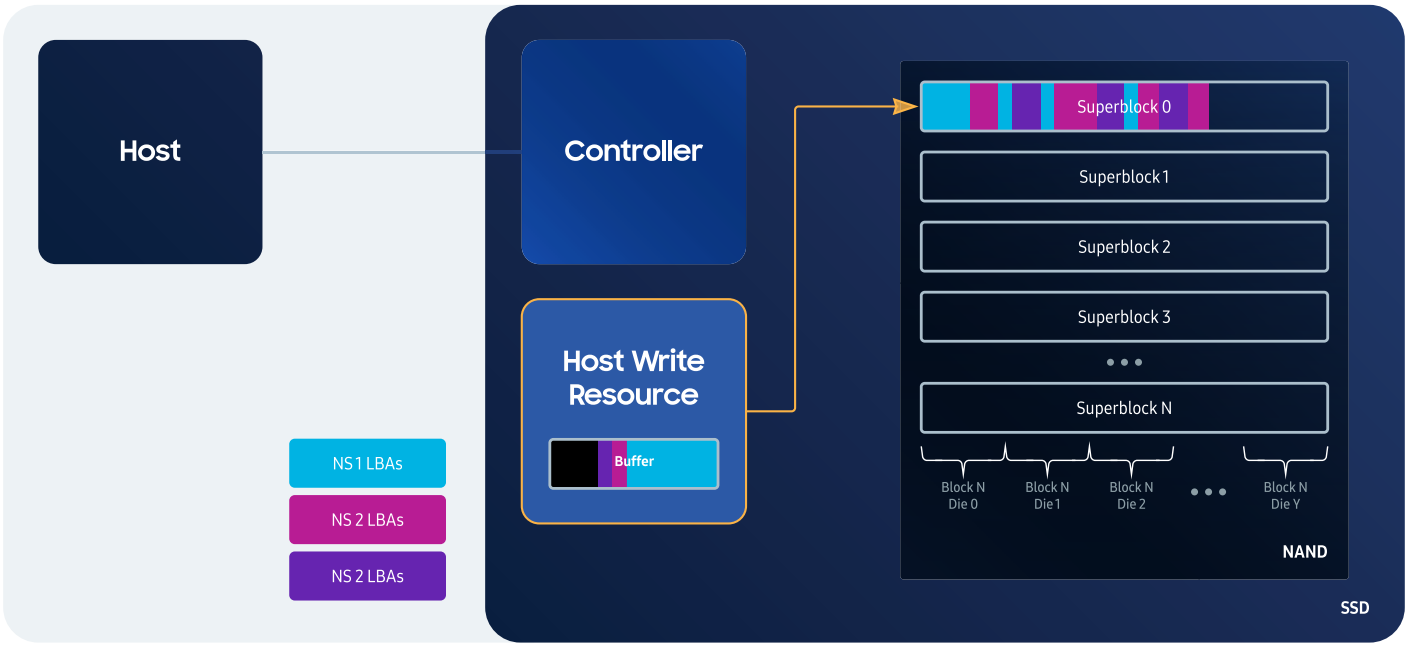

Figure 1: Logical view of a Typical Conventional NVMe SSD using Superblocks

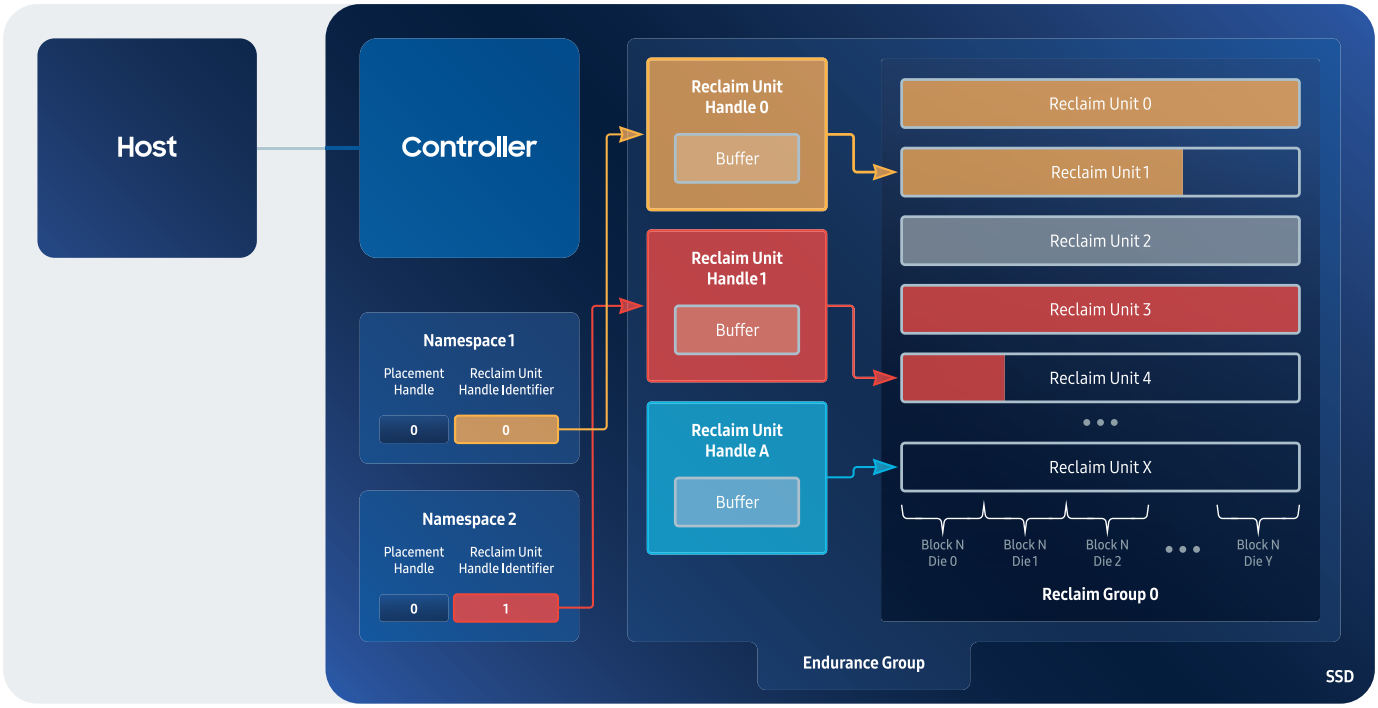

Figure 2: Logical view of an FDP NVMe SSD using Superblocks

SAMSUNG

The FDP architecture as shown in the above figure has the following components.

- **Reclaim Unit (RU):** A set of NAND blocks to which a host may write logical blocks. This is similar to the SuperBlock concept in the Flash Translation Layer (FTL) of SSD. The typical size is a few GBs (e.g. 6GB).

- **Reclaim Group (RG):** A collection of Reclaim Units. This concept is typically used when the device allows the host to segregate data in different NAND dies based on a host policy.

- **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 at the same time. Many FDP devices have the number of RUHs as 8, 16, 256 etc.

- **Placement Handle:** An index into the list of Reclaim Unit Handles, accessible by namespace, that is defined at namespace creation. Host writes to that namespace can only access the RUHs in that list.

- **Endurance Group:** A collection of NAND blocks for which endurance is managed as a single unit, with the intention that those NAND blocks wear out at the same time (i.e., SSD life). In a typical SSD, there exists a single Endurance Group.

The key takeaways are that Data Separation is achieved by means of Reclaim Unit Handle (RUH) and Reclaim Group (corresponding to die), and data alignment to superblocks is achieved by means of the Reclaim Unit concept.

A host may employ data separation using RUHs when the workloads have distinct IO patterns with different lifetimes. For example, when the host has hot/warm/cold data streams, RUHs may be used to separate them in the SSD. Data alignment on RU boundaries is typically useful when the host has a set of BLOBs or data objects that it wants to allocate in separate physical media because a particular object may have a different lifetime than the other ones.

2. Software Prerequisites for FDP

2.1 Install nvme-cli and libnvme

The NVMe-CLI tool is required to configure the FDP drive with the necessary settings. Install nvme-cli and the associated libnvme as a preliminary step.

```
#nvme-cli and libnvme
git clone https://github.com/linux-nvme/nvme-cli.git
cd nvme-cli
meson setup --force-fallback-for=libnvme .build
meson compile -C .build
meson install -C .build
#verify the installed version
nvme -version
```

If the meson version does not support "--force-fallback-for" argument, it will be necessary to install libnvme separately.

2.1 Install liburing

Many of the tools and ecosystems use the liburing library to send the FDP-enabled IOs to the Linux kernel. Install it from the repo https://github.com/axboe/liburing.git.

```
#liburing
git clone https://github.com/axboe/liburing.git
cd liburing
./configure --cc=gcc --cxx=g++
make -j$(nproc)
sudo make install
```

Note: There are reports of build issues due to different versions of liburing being present in the system. Removing the existing liburing versions

```
sudo apt remove liburing2
sudo apt remove liburing-dev
```

and installing from the source should fix this issue. One such issue is discussed at https://github.com/facebook/CacheLib/issues/302.

3. FDP Drive Initialization

FDP has to be explicitly enabled in the device and can only be done when no namespaces exist.

The NVMe-CLI tool ('nvme') is used for configuring NVMe FDP drive. The following example depicts the configuration of a 2TB FDP drive.

2TB FDP Device Setup

```
#Delete any pre-existing namespaces
nvme delete-ns /dev/nvme0 -n 1

#Disable FDP
nvme set-feature /dev/nvme0 -f 0x1D -c 0 -s
nvme get-feature /dev/nvme0 -f 0x1D -H

#enable FDP
nvme set-feature /dev/nvme0 -f 0x1D -c 1 -s
nvme get-feature /dev/nvme0 -f 0x1D -H

# Get capacity of drive and use it for further calculations
nvme id-ctrl /dev/nvme0 | grep nvmcap | sed “s/,//g” | awk ‘{print $3/4096}’

# Create namespace. use the capacity values from the above command in --nsze etc
nvme create-ns /dev/nvme0 -b 4096 --nsze=459076086 --ncap=459076086 -p 0,1,2,3 -n 4

#Attach namespace. Ex: NS id = 1, controller id = 0x7
nvme attach-ns /dev/nvme0 --namespace-id=1 --controllers=0x7

# Deallocate
nvme dsm /dev/nvme0n1 -n 1 -b 459076086
```

4. Getting to know your FDP device

4.1 FDP Device Information

4.1.1 FDP Device Configuration

Command	Output

`$ nvme fdp configs /dev/nvme0 -e 1`	`FDP Attributes: 0x80`
	`Vendor Specific Size: 144`
	`Number of Reclaim Groups: 1`
	`Number of Reclaim Unit Handles: 8`
	`Number of Namespaces Supported: 2`
	`Reclaim Unit Nominal Size: 6436159488`
	`Estimated Reclaim Unit Time Limit: 0`
	`Reclaim Unit Handle List:  (Initially Isolated/Persistently Isolated )`
	`[0]: Initially Isolated`
	`[1]: Initially Isolated`
	`[2]: Initially Isolated`
	`[3]: Initially Isolated`
	`[4]: Initially Isolated`
	`[5]: Initially Isolated`
	`[6]: Initially Isolated`
	`[7]: Initially Isolated`

Initially Isolated: The user data from a write command that utilizes this type of Reclaim Unit Handle is originally isolated in the referenced RU from other user data in write commands that utilize a different RU Handle in the same Reclaim Group. If the controller moves the user data due to vendor specific operations (i.e., garbage collection), then it may do so to an RU in the same Reclaim Group that contains other user data moved by the controller that was written by the host utilizing any RU Handle of the same type.

Persistently Isolated: The user data from a write command using this type of Reclaim Unit Handle is initially kept separate in its referenced RU from other data written using a different RU Handle. If the controller needs to move this data (like during garbage collection), it must relocate it to an RU within the same Reclaim Group, only containing data written by the host using the same RU Handle.

Reclaim Unit Nominal Size: This field indicates the nominal size, in bytes, of each Reclaim Unit of non-volatile storage within each Reclaim Group.

FDP Attributes: Refer to the NVMe FDP spec (TP4146) for details of this field. The MSB (Bit 7) Indicates whether FDP configuration is valid or not. FDP is enabled in the given example output.

4.1.2 FDP Device Stats

Command	Output

`$ nvme fdp stats /dev/nvme0 -e 1`	`Host Bytes with Metadata Written (HBMW): 277,943,428,800,512`
	
	`Media Bytes with Metadata Written (MBMW): 610,715,721,687,040`
	
	`Media Bytes Erased (MBE): 610,752,503,218,176`

Host Bytes with Metadata Written (HBMW): Contains the total number of bytes of user data the host has written to the specified Endurance Group as part of processing I/O commands as defined in the appropriate I/O Command Set specification.

Media Bytes with Metadata Written (MBMW): Contains the total number of bytes of user data that have been written to the specified Endurance Group including both host and controller writes (e.g., garbage collection and background media scan operations) as part of processing as defined in the appropriate I/O Command Set specification.

Media Bytes Erased (MBE): Contains the total number of bytes of data that have been erased in the specified Endurance Group.

FDP Attributes: Refer to the NVMe FDP spec (TP4146) for details of this field. The MSB (Bit 7) Indicates whether FDP configuration is valid or not. FDP is enabled in the given example output.

4.1.3 FDP Device Status

Command	Output

`$ nvme fdp status /dev/nvme0 --namespace-id=1`	`Placement Identifier 0; Reclaim Unit Handle Identifier 0`
	`  Estimated Active Reclaim Unit Time Remaining (EARUTR): 0`
	`  Reclaim Unit Available Media Writes (RUAMW): 452484`
	
	`Placement Identifier 1; Reclaim Unit Handle Identifier 1`
	`  Estimated Active Reclaim Unit Time Remaining (EARUTR): 0`
	`  Reclaim Unit Available Media Writes (RUAMW): 1571328`
	
	`Placement Identifier 2; Reclaim Unit Handle Identifier 2`
	`  Estimated Active Reclaim Unit Time Remaining (EARUTR): 0`
	`  Reclaim Unit Available Media Writes (RUAMW): 1571328`
	
	`Placement Identifier 3; Reclaim Unit Handle Identifier 3`
	`  Estimated Active Reclaim Unit Time Remaining (EARUTR): 0`
	`  Reclaim Unit Available Media Writes (RUAMW): 1571328`

Placement Identifier (PID): This field indicates the Placement Identifier containing the Placement Handle and Reclaim Group Identifier for this Reclaim Unit Handle Status Descriptor.

Reclaim Unit Handle Identifier: This field indicates the Reclaim Unit Handle for the Placement Identifier field.

Reclaim Unit Available Media Writes (RUAMW): This field indicates the number of logical blocks currently able to be written to the media associated with the Reclaim Unit referenced by the Placement Identifier field.

4.2 FDP Events

FDP Events inform the host about the activities happening on the device. These events are a means to help the host know when its data placement scheme is not functioning as expected.

4.2.1 FDP Event Types

FDP Event Type	Event Description
RU Not Written to Capacity (0h)	The Reclaim Unit Handle reported that it was forced to reference a different RU due to an I/O Management Send command issued by the host prior to the previously referenced RU being written to capacity.
RU Time Limit Exceeded (1h)	The capacity of a Reclaim Unit was not completely written within the time defined in the "Estimated Reclaim Unit Time Limit" field due to the controller modifying the Reclaim Unit Handle to reference a different Reclaim Unit.
Controller Reset Modified RUH's (2h)	A write command specified an invalid Placement Identifier due to an invalid Placement Handle or an invalid Reclaim Group Identifier.
Media Reallocated (80h)	User data contained in a Reclaim Unit was written by the host specifying an RU Handle with a type of Initially Isolated, and was moved by the controller to a different Reclaim Unit (e.g. during garbage collection).
Implicitly modified RUH (81h)	The controller modified a Reclaim Unit Handle to reference a different RU where that modification was not due to any host action (e.g. a write command that has a size greater than the remaining capacity of the RU being written to).

4.2.2 Querying FDP Events Generated by NVMe device

FDP Event Logs are generated by the NVMe device and they need to be polled by the host system.

```
$ nvme fdp events /dev/nvme0 --endgrp-id=1
Event[0]
  Event Type: 0x80 (Media Reallocated)
  Event Timestamp: 1707300380230 (Wed 07 Feb 2024 15:36:20 IST IST)
  Number of LBAs Moved (NLBAM): 65535
Event[1]
  Event Type: 0x80 (Media Reallocated)
  Event Timestamp: 1707300381105 (Wed 07 Feb 2024 15:36:21 IST IST)
  Number of LBAs Moved (NLBAM): 65535
```

5. Linux SW stack support overview

The following diagram gives an overview of the Linux software stack support for FDP. Each of the modules will be explained in detail in the sections that follow.

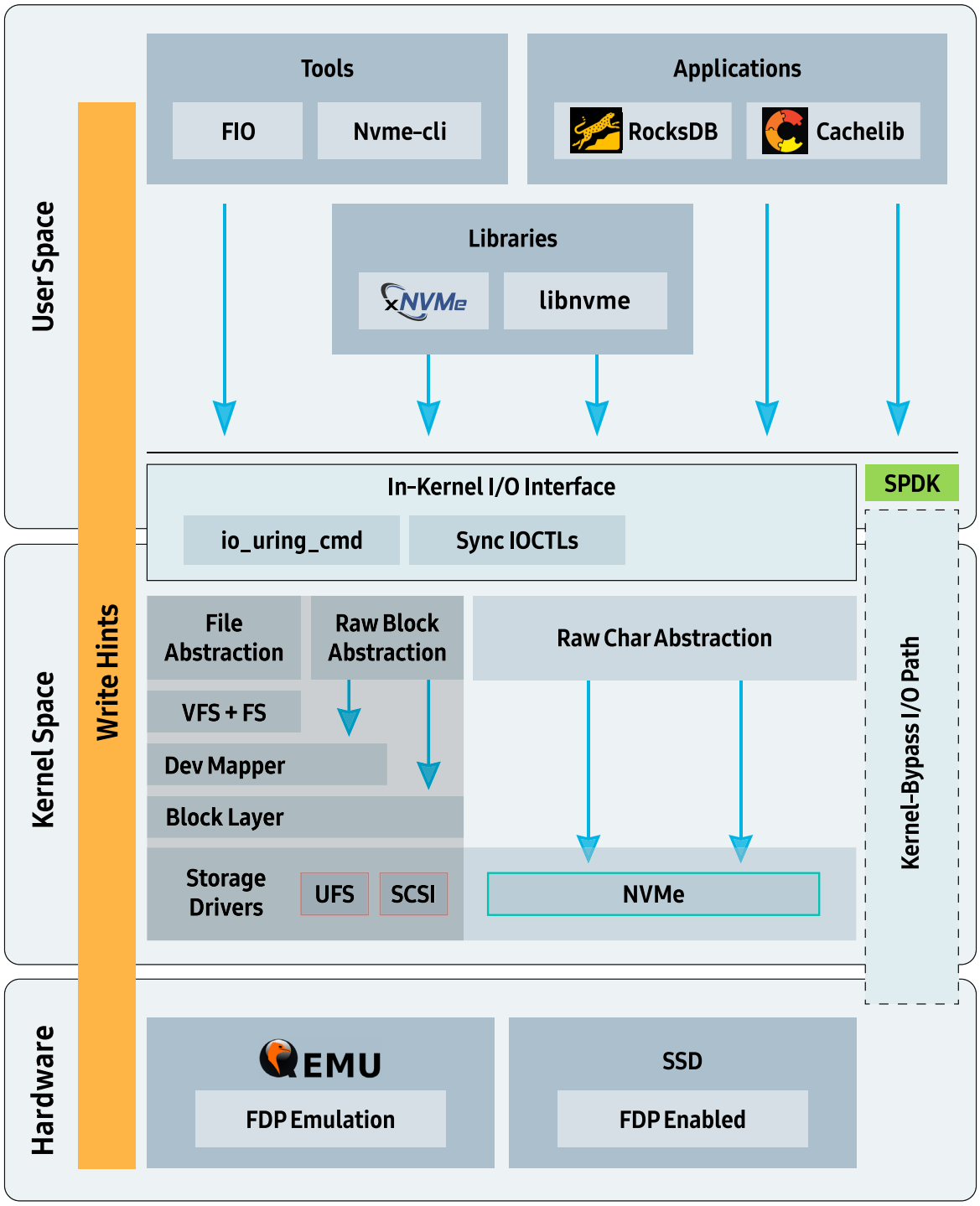

Figure 3: FDP support in Linux Software Stack

SAMSUNG

6. NVMe-CLI

NVMe-CLI is a command-line tool used for managing NVMe drives on Linux platform. It allows users to interact with and configure NVMe storage devices through a terminal interface. With NVMe-CLI, users can query device information, manage namespaces, update firmware, and more. It supports FDP feature specification from version 2.3.

These are the supported FDP commands in NVMe-CLI.

FDP Command	FDP Event Type	Event Description
configs	#nvme fdp configs /dev/nvme0 --end-grp-id=1	The FDP Configurations log page identifies a list of static FDP configurations that are allowed to be applied to the specified Endurance Group.
usage	#nvme fdp usage /dev/nvme0 endgrp-id=1	The Reclaim Unit Handle Usage log page provides information about the Reclaim Unit Handles associated with the Placement Handles of the namespaces in the specified Endurance Group.
stats	#nvme fdp stats /dev/nvme0 endgrp-id=1	The FDP Statistics log page provides information about the FDP configuration over the life of the FDP configuration in an Endurance Group.
events	#nvme fdp events /dev/nvme0 endgrp-id=1	The FDP Events log page provides information about events affecting Reclaim Units and media usage in an Endurance Group that has FDP enabled.
status	#nvme fdp status /dev/nvme0 --name-space-id=1	The Reclaim Unit Handle Status management operation provides information about Reclaim Unit Handles that are accessible by the specified namespace.
update	#nvme fdp update /dev/nvme0 --name-space-id=1 --place-ment-handle=0,1,2,3	The Reclaim Unit Handle Update management operation generates the I/O Management Send command to update the reclaim unit handles.
set-events	#nvme fdp set-event /dev/nvme0 -n 1 -p <placement handle> -e <enable/disable> -t <event type>	The Set Events command is used to enable or disable FDP events.
version	#nvme fdp version	The Version command shows FDP and libnvme versions.

6.1 Obtaining Device WAF using NVMe-CLI

NVMe-CLI can be used to obtain the device Write Amplification Factor (WAF) when FDP is enabled. WAF can be obtained by dividing the amount of data written to flash media (MBMW) to the amount of data written by the host (HBMW).

Command	Output

`$ nvme fdp stats /dev/nvme0 -e 1`	`Host Bytes with Metadata Written (HBMW): 277,943,428,800,512`
	
	`Media Bytes with Metadata Written (MBMW): 610,715,721,687,040`
	
	`Media Bytes Erased (MBE): 610,752,503,218,176`

`$ nvme fdp stats /dev/nvme0 -e 1 | awk '/(HBMW)/ {hbmw = $7} /(MBMW)/ {mbmw = $7} END {print "Device WAF = " mbmw/hbmw}'`	`Device WAF = 2.19`

Note: This will only work with FDP enabled mode in the devices. And the above command gives the amount of writes accumulated during life time of the device. The user need to baseline the numbers by collecting it before and after the experimentation.

7. FIO – Flexible I/O Tool

FIO is a flexible I/O tool essentially used for performance measurement. It can spawn a number of threads or processes, each doing a particular type of I/O action as specified by the user.

FDP support has been available in fio from 3.34 (Mar 2023) from this commit. It is recommended to use the latest master branch to take advantage of all available features.

The user can enable the FDP feature in fio with the new "fdp=1" parameter for fio's io_uring_cmd ioengine. By default, the fio jobs will cycle through all the available placement identifiers for write commands to that namespace. The user can limit the placement identifiers that can be used with an additional parameter, "fdp_pli=<list,>", which can be useful for separating write-intensive jobs from less write-intensive ones. Setting up the namespace for FDP is outside the scope of 'fio', so the description below assumes the namespace is already properly configured for the mode.

The current FDP support in FIO provides a sample fdp job that is more like a 'hello world' app for developers and users. It allows the FDP device users to clarify the WAF hypothesis and test performances in FDP and non-FDP cases.

fio/examples/uring-cmd-fdp.fio

```
# io_uring_cmd I/O engine for nvme-ns generic character device with FDP enabled
# This assumes the namespace is already configured with FDP support and has at
# least 8 available reclaim units.
#
# Each job targets different ranges of LBAs with different placement
# identifiers, and has different write intensity.

[global]
filename=/dev/ng0n1
ioengine=io_uring_cmd
cmd_type=nvme
iodepth=32
bs=4K
fdp=1
time_based=1
runtime=1000

[write-heavy]
rw=randrw
rwmixwrite=90
fdp_pli=0,1,2,3
offset=0%
size=30%

[write-mid]
rw=randrw
rwmixwrite=30
fdp_pli=4,5
offset=30%
size=30%

[write-light]
rw=randrw
rwmixwrite=10
fdp_pli=6
offset=60%
size=30%
```

FDP specific options

1. Activate: --fdp=1
    - Read available Placement Identifiers (PID) from device.
2. Select PIDs: --fdp_pli=[OFFSET_LIST]
    - Specify the offset of placement identifiers to be used from the available placement identifier.
    - If not specified, fio will use all available placement identifiers.
3. Selection method: --fdp_pli_select=[TYPE] -random -roundrobin (default)
    - random: select random placement identifiers for each write operation from the available ones (passed via option --fdp_pli)
    - roundrobin: roundrobin across all available placement identifiers for each write operation (passed via option --fdp_pli)

Examples

To test with io_uring_cmd i.e. bypassing block layer, use this example configuration

- https://github.com/axboe/fio/blob/master/examples/uring-cmd-fdp.fio

To test with xnvme ioengine, use this example configuration

- https://github.com/axboe/fio/blob/master/examples/xnvme-fdp.fio

8. Linux Kernel support - I/O Passthru

Kernel support for FDP is provided through the I/O passthru mechanism.
(Note: This may change, as FDP patches for Linux block layer are being discussed in the Linux kernel community.)

I/O passthru is a mechanism through which user space applications can access the NVMe device directly without going through file system or block layer APIs. This mechanism is often used by experimental features in NVMe until they are proven and merged into the kernel.

I/O passthru is upstream since kernel v5.13. More information is available in this paper:
https://www.usenix.org/conference/fast24/presentation/joshi

Sample application code that uses I/O passthru can be found here:
https://github.com/axboe/liburing/blob/master/test/io_uring_passthrough.c

Also, a real deployment example of I/O passthru with FDP directives can be found at CacheLib code here:
https://github.com/facebook/CacheLib/blob/main/cachelib/navy/common/FdpNvme.cpp#L132.

9. Linux Kernel File System/Block IO support

This section outlines using Linux FS/Block IO over the NVMe-FDP device that is separate from using the I/O passthru mechanism – that is, how Linux application(s) sitting atop the filesystem can use the data-placement capabilities offered by FDP.

Within this approach, there are two ways for a filesystem to leverage the FDP hints:

- Application-driven placement. The application can assign different hints to file data based on an understanding of the expected lifetime. This approach does not require the filesystem to make any decision on the file data.

- Filesystem-driven placement. The filesystem can implement a fully automated policy of segregating data and metadata. F2FS presents one such example as it provides a ‘fs-based’ mount option.

9.1 Application-driven placement

Linux 6.9 kernel re-enables the write-hint infrastructure that allows application to set/query data lifetime hints on the file.

Reference: https://elixir.bootlin.com/linux/v6.9-rc1/source/include/linux/rw_hint.h#L9

The Linux SCSI driver can pass lifetime information to an underlying device. However, the upstream NVMe driver does not have similar functionality. Samsung has prepared a custom kernel in which the NVMe driver maps write-hint values to the FDP placement identifiers.

Source: https://github.com/SamsungDS/linux/tree/feat/fdp_fs

This enables an application to leverage FDP using the write-hint interface.

Guiding the data-placement using the write-hint interface

Users can set/get write-hint to a file using the fcntl() interface.

- F_GET_RW_HINT: allows getting the hint from the file's inode
- F_SET_RW_HINT: allows setting the hint to the file's inode

List of available write hints:

FDP Event Type	Event Description
RWH_WRITE_LIFE_NOT_SET	Not to set the write life
RWH_WRITE_LIFE_NONE	No hints about write lifetime
RWH_WRITE_LIFE_SHORT	Data written has a short lifetime
RWH_WRITE_LIFE_MEDIUM	Data written has a medium lifetime
RWH_WRITE_LIFE_LONG	Data written has a long lifetime
RWH_WRITE_LIFE_EXTREME	Data written has an extremely long lifetime

The write-hints are passed down for both buffered and unbuffered (direct) I/O.

The NVMe driver converts the write-hint to the device-specific placement-identifier, and sends that to the device in the write command.

Examples

Fio on XFS volume

fio only supports 5 values:

none
No particular life time associated with this file.

short
Data written to this file has a short life time.

medium
Data written to this file has a medium life time.

long
Data written to this file has a long life time.

extreme
Data written to this file has a very long life time.

These are sent using the "--write_hint" option.

```
mkfs.xfs /dev/nvme0n1
mount /dev/nvme0n1 /mnt
fio --filename=/mnt/f1 --bs=4k --ioengine=io_uring --group_reporting=1 --unlink=0 --size=1M
--direct=1 --name=f1 --rw=randwrite --write_hint=extreme
fio --filename=/mnt/f2 --bs=4k --ioengine=io_uring --group_reporting=1 --unlink=0 --size=1M
--direct=1 --name=f1 --rw=randwrite --write_hint=short
```

Custom C program (showing the fcntl interface)

```
/*
 * writehint.c: check or set a file/inode write hint
 */
#define _GNU_SOURCE
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>

#define BUF_SIZE 4096
```

```
static char *str[] = { “WRITE_LIFE_NOT_SET”, “WRITE_LIFE_NONE”,
                       “WRITE_LIFE_SHORT”, “WRITE_LIFE_MEDIUM”,
                       “WRITE_LIFE_LONG”, “WRITE_LIFE_EXTREME” };

int main(int argc, char *argv[])
{
        uint64_t hint;
        int fd, ret;
        int i = 0;
        char * buf;
        struct stat fstat;
        int blksize;
        int opt_direct;

        if (argc < 3) {
                fprintf(stderr, “Usage: %s file <b/d(buffered/direct)> <hint>\n”, argv[0]);
                return 1;
        }

        if (strncmp(argv[2], “b”, strlen(“b”)) == 0) {
                opt_direct = 0;
        } else if (strncmp(argv[2], “d”, strlen(“d”)) == 0) {
                opt_direct = 1;
        } else {
                printf(“provide arg 2 as \”b\” or \”d\”\n”);
                exit(0);
        }

        if (opt_direct)
                fd = open(argv[1], O_CREAT|O_RDWR|O_DIRECT, 0644);
        else
                fd = open(argv[1], O_CREAT|O_RDWR|O_APPEND, 0644);

        if (fd < 0) {
                perror(“open”);
                return 1;
        }

        /* fetch the block size*/
        if (stat(argv[1], &fstat) == 0) {
                blksize = (int)fstat.st_blksize;
        } else {
                perror(“stat”);
                return 1;
        }

        /* Reminder: int posix_memalign(void **memptr, size_t alignment, size_t size); */
        if (posix_memalign((void *)&buf, blksize, BUF_SIZE)) {
                perror(“posix_memalign failed”);
                return 1;
        }
```

```
        memset(buf, 'A', BUF_SIZE);
        hint = atoi(argv[3]);
        ret = fcntl(fd, F_SET_RW_HINT, &hint);
        if (ret < 0) {
                perror("fcntl: F_SET_RW_HINT");
                return 1;
        }

        ret = fcntl(fd, F_GET_RW_HINT, &hint);
        if (ret < 0) {
                perror("fcntl: F_GET_RW_HINT");
                return 1;
        }
        printf("Hint fetched from file : %ld\n", hint);

        ret = write(fd, buf,  BUF_SIZE);
        if (ret < 0) {
                printf("Couldn't write to the file\n");
        } else {
                printf("Written %d bytes to the file\n", ret);
        }

        printf("%s: hint %s\n", argv[1], str[hint]);
        close(fd);
        return 0;
}

#cc -o whint <above_program.c>

/* Direct IO */
# ./whint /mnt/a.txt d 3

/* Buffered IO */
# ./whint /mnt/b.txt b 4
```

9.2 Filesystem-driven placement

Shown below is the "bio" data structure that file-systems use to form the I/O request.
The file system can pass lifetime information by setting the "bi_write_hint" value.

```
struct bio {
      struct bio              *bi_next;       /* request queue link */
      struct block_device     *bi_bdev;
      blk_opf_tbi_opf;        /* bottom bits REQ_OP, top bits req_flags. */
      unsigned short          bi_flags;       /* BIO_* below */
      unsigned short          bi_ioprio;
      enum rw_hint            bi_write_hint;
      ...
}
```

10. SPDK - Storage Performance Development Kit

The Storage Performance Development Kit (SPDK) provides a set of tools and libraries for writing high performance, scalable, user-mode storage applications (source - https://github.com/spdk/spdk). SPDK achieves high performance by moving all of the necessary drivers into user space and operating in a polled mode instead of relying on interrupts, which avoids kernel context switches and eliminates interrupt handling overhead. FDP support was added to SPDK with v23.05 release. This list shows the currently supported FDP features.

FDP Commands	Specifics	Supported
Log pages	FDP configuration	Yes
	Reclaim unit handle usage	Yes
	FDP statistics	Yes
	FDP events	Yes
Set / Get Features	Flexible data placement	• Get feature - Yes • Set feature - No (i.e. can't enable disable FDP)
	FDP Events	Yes: Both Set and Get features
I/O commands	Write with hints	• SGL - Yes • PRP - No
	I/O management send	Yes: Reclaim unit handle update
	I/O management receive	Yes: Reclaim unit handle status
Namespace management	Namespace create / delete	No

Example

SPDK has a dedicated sample test application that can be used to test all the supported FDP features:

- https://github.com/spdk/spdk/blob/master/test/nvme/fdp/fdp.c

The following output from an emulated FDP device under qemu demonstrates the sample test application in use.

```
./scripts/setup.sh
0000:00:04.0 (1af4 1001): Active devices: mount@vda:vda1, so not binding PCI dev
0000:00:03.0 (1b36 0010): nvme -> uio_pci_generic

./test/nvme/fdp/fdp
Initializing NVMe Controllers
Attaching to 0000:00:03.0
Controller supports FDP Attached to 0000:00:03.0
Namespace ID: 1 Endurance Group ID: 1
Initialization complete.

==================================
== FDP tests for Namespace: #01 ==
==================================

Get Feature: FDP:
=================
  Enabled:                 Yes
  FDP configuration Index: 0

FDP configurations log page
===========================
Number of FDP configurations:         1
Version:                               0
Size:                                  96
FDP Configuration Descriptor:         0
  Descriptor Size:                     80
  Reclaim Group Identifier format:    2
  FDP Volatile Write Cache:            Not Present
  FDP Configuration:                  Valid
  Vendor Specific Size:               0
  Number of Reclaim Groups:            2
  Number of Reclaim Unit Handles:      4
  Max Placement Identifiers:          128
  Number of Namespaces Supported:      256
  Reclaim unit Nominal Size:           6000000 bytes
  Estimated Reclaim Unit Time Limit:  Not Reported
    RUH Desc #000:          RUH Type: Initially Isolated
    RUH Desc #001:          RUH Type: Initially Isolated
    RUH Desc #002:          RUH Type: Initially Isolated
    RUH Desc #003:          RUH Type: Initially Isolated

FDP reclaim unit handle usage log page
======================================
Number of Reclaim Unit Handles:       4
  RUH Usage Desc #000:   RUH Attributes: Unused
  RUH Usage Desc #001:   RUH Attributes: Host Specified
  RUH Usage Desc #002:   RUH Attributes: Host Specified
  RUH Usage Desc #003:   RUH Attributes: Host Specified
```

```
FDP statistics log page
=======================
Host bytes with metadata written:  0
Media bytes with metadata written: 0
Media bytes erased:                0

FDP Reclaim unit handle status
==============================
Number of RUHS descriptors:   6
RUHS Desc: #0000  PID: 0x0000  RUHID: 0x0001  ERUT: 0x00000000  RUAMW: 0x0000000000006000
RUHS Desc: #0001  PID: 0x4000  RUHID: 0x0001  ERUT: 0x00000000  RUAMW: 0x0000000000006000
RUHS Desc: #0002  PID: 0x0001  RUHID: 0x0002  ERUT: 0x00000000  RUAMW: 0x0000000000006000
RUHS Desc: #0003  PID: 0x4001  RUHID: 0x0002  ERUT: 0x00000000  RUAMW: 0x0000000000006000
RUHS Desc: #0004  PID: 0x0002  RUHID: 0x0003  ERUT: 0x00000000  RUAMW: 0x0000000000006000
RUHS Desc: #0005  PID: 0x4002  RUHID: 0x0003  ERUT: 0x00000000  RUAMW: 0x0000000000006000

FDP write on placement id: 0 success

Set Feature: Enabling FDP events on Placement handle: #0 Success

IO mgmt send: RUH update for Placement ID: #0 Success

Get Feature: FDP Events for Placement handle: #0
========================
Number of FDP Events: 6
FDP Event: #0  Type: RU Not Written to Capacity     Enabled: Yes
FDP Event: #1  Type: RU Time Limit Exceeded         Enabled: Yes
FDP Event: #2  Type: Ctrlr Reset Modified RUH's     Enabled: Yes
FDP Event: #3  Type: Invalid Placement Identifier   Enabled: Yes
FDP Event: #4  Type: Media Reallocated              Enabled: No
FDP Event: #5  Type: Implicitly modified RUH        Enabled: No

FDP events log page
===================
Number of FDP events: 1
FDP Event #0:
  Event Type:                       RU Not Written to Capacity
  Placement Identifier:             Valid
  NSID:                             Valid
  Location:                         Valid
  Placement Identifier:             0
  Event Timestamp:                  2
  Namespace Identifier:             1
  Reclaim Group Identifier:         0
  Reclaim Unit Handle Identifier:   1

FDP test passed
```

11. xNVMe - Cross-platform NVMe Library

xNVMe provides a user space library libxnvme and a suite of tools and libraries that work with NVMe devices. The NVMe driver being used by libxnvme is re-targetable, and can be any one of the GNU/Linux Kernel NVMe driver via libaio, IOCTLs and io_uring, the SPDK NVMe driver, or your own custom NVMe driver.

- **Source —** https://github.com/OpenMPDK/xNVMe
- **Getting started —** https://xnvme.io/docs/latest/

FDP support was added to xNVMe with the v0.7.0 release. These included changes to the APIs for new log pages, new feature identifiers, write with new directive, and the new I/O management commands. The xnvme and lblk tools have been modified to allow the end user to test these functionalities.

- To list out the supported commands by these tools

```
#xnvme --help
```

- For a specific command

```
#xnvme log-fdp-config --help
```

  This shows a list of arguments that can be provided with the tool xnvme. Arguments in [ ] are optional whereas all other arguments are mandatory.

Example

```
// Bit 19 is set to 1, for FDP to be supported
$ xnvme idfy-ctrlr /dev/ng0n1 | grep ctratt
  ctratt: 0x88010

// 512 bytes data buffer, endurance group id as log specific identifier field
$ xnvme log-fdp-config /dev/ng0n1 --lsi 0x1 --data-nbytes 512
# Allocating and clearing buffer...
# Retrieving FDP configurations log page ...
xnvme_spec_log_fdp_conf:
  ncfg: 0
  version: 0
  size: 96
  config_desc: 0
  ds: 80
  fdp attributes: {    rgif: 2    fdpvwc: 0    fdpcv: 1    val: 0x82  }
  vss: 0
  nrg: 2
  nruh: 4
  maxpids: 127
  nns: 256
```

```
  runs: 100663296
  erutl: 0
   - ruht[0]: 1
   - ruht[1]: 1
   - ruht[2]: 1
   - ruht[3]: 1

// Probe if fdp is enabled and what fdp configuration index is present from fdp config log
page
$ xnvme feature-get /dev/ng0n1 --fid 0x1d --cdw11 0x1
# cmd_gfeat: {nsid: 0x1, fid: 0x1d, sel: 0x0}
feat: { fdpe: 1, fdpci: 0 }

// Write 4 lba at start lba 0, to placement ID 1
$ lblk write-dir /dev/ng0n1 --slba 0x0 --nlb 3 --dtype 2 --dspec 0x1
# Writing nsid: 0x1, slba: 0x0000000000000000, nlb: 3
# Alloc/fill dbuf, dbuf_nbytes: 16384
# Preparing and sending the command...

// 4 LBAs or 16K written
$ xnvme log-fdp-stats /dev/ng0n1 --lsi 0x1
# Allocating and clearing buffer...
# Retrieving FDP statistics log page ...
xnvme_spec_log_fdp_stats:
  hbmw: [16384, 0]
  mbmw: [16384, 0]
  mbe: [0, 0]

// Fetch the reclaim unit handle usage log page for 4 RUH
$ xnvme log-ruhu /dev/ng0n1 --lsi 0x1 --limit 4
# Allocating and clearing buffer...
# Retrieving ruhu-log ...
# 4 reclaim unit handle usage:
xnvme_spec_log_ruhu:
  nruh: 4
  - ruhu_desc[0]:  0
  - ruhu_desc[1]:  0x1
  - ruhu_desc[2]:  0x1
  - ruhu_desc[3]:  0x1

// Enable all supported fdp events placement handle 0x0
$  xnvme set-fdp-events /dev/ng0n1 --fid 0x1e --feat 0xFF0000 --cdw12 0x1
# cmd_sfeat: {nsid: 01, fid: 0x1e, save: 0x0, feat: 0xff0000, cdw12: 0x1}

// Create an event, by updating reclaim unit handle, for placement ID 0x0
$ xnvme fdp-ruhu /dev/ng0n1 --pid 0x0
# Updating ruh ...

// Fetch the log page for FDP event.
$ xnvme log-fdp-events /dev/ng0n1 --nsid 0x1 --limit 1 --lsi 0x1 --lsp 0x1
# Allocating and clearing buffer...
# Retrieving fdp-events-log ...
```

```
# 1 fdp events log page entries:
xnvme_spec_log_fdp_events:
  nevents: 1
  - {type: 0, fdpef: 0x7, pid: 0, timestamp: 564658057972825, nsid: 1, rgid: 0, ruhid: 1, }

// Fetch the enabled FDP events
$ xnvme feature-get /dev/ng0n1 --fid 0x1e --cdw11 0xFF0000 --data-nbytes 510
# cmd_gfeat: {nsid: 0x1, fid: 0x1e, sel: 0x0}
nevents: 6 }
{ type: 0, event enabled: 1 }
{ type: 0x1, event enabled: 1 }
{ type: 0x2, event enabled: 1 }
{ type: 0x3, event enabled: 1 }
{ type: 0x80, event enabled: 0 }
{ type: 0x81, event enabled: 0 }

// Reclaim unit handle status
$ xnvme fdp-ruhs /dev/ng0n1 --limit 6
# Allocating and clearing buffer...
# Retrieving ruhs ...
# 6 reclaim unit handle status:
xnvme_spec_ruhs:
  nruhsd: 6
  - ruhs_desc[0] : { pi: 0 ruhi: 1 earutr: 0 ruamw: 24576}
  - ruhs_desc[1] : { pi: 16384 ruhi: 1 earutr: 0 ruamw: 24576}
  - ruhs_desc[2] : { pi: 1 ruhi: 2 earutr: 0 ruamw: 24572}
  - ruhs_desc[3] : { pi: 16385 ruhi: 2 earutr: 0 ruamw: 24576}
  - ruhs_desc[4] : { pi: 2 ruhi: 3 earutr: 0 ruamw: 24576}
  - ruhs_desc[5] : { pi: 16386 ruhi: 3 earutr: 0 ruamw: 24576}
```

12. Application Enablement - CacheLib

A detailed documentation on the FDP setup in CacheLib is being integrated to the CacheLib website. The PR for the same and details can be found at https://github.com/facebook/CacheLib/pull/308.

12.1 Purpose of FDP support in CacheLib

CacheLib's BigHash and BlockCache engines of the SSD Cache layer (Navy) produce distinct IO patterns on the SSD. BigHash generates a random write pattern while BlockCache generates a sequential write pattern.

In a conventional SSD, these writes get mixed up in the physical NAND media. This intermixing can lead to a higher SSD Write Amplification Factor (WAF). To combat this in production environments CacheLib uses up to 50% of the SSD as host over-provisioning (https://www.usenix.org/system/files/osdi20-berg.pdf).

The Flexible Data Placement (FDP) support within CacheLib aims to segregate the BigHash and BlockCache data of CacheLib within the SSDs. This reduces the device WAF even when configured with 0% host over-provisioning and improves device endurance. FDP support within Navy is optional.

The FDP support for CacheLib has been upstreamed and can be found at: https://github.com/facebook/CacheLib (the closed PR: https://github.com/facebook/CacheLib/pull/277).

For more details on BigHash and BlockCache and other CacheLib specific information, refer to https://cachelib.org/.

12.2 Setting up CacheLib code and FDP device for testing

12.2.1 Prerequisites

Install nvme-cli, libnvme, and liburing following the steps mentioned in the "Software Prerequisites for FDP" section.

12.2.2 Build

```
$ git clone https://github.com/facebook/CacheLib
$ cd CacheLib
$ ./contrib/build.sh -j -T

# The resulting library and executables:
$ ./opt/cachelib/bin/cachebench –help
```

Note: For more information on build options, refer to https://cachelib.org/docs/installation

12.2.3 Set up a device with FDP enabled

Refer to the "FDP Drive Initialization" and "Configuring FDP on the NVMe device" sections and set up the SSD with FDP enabled.

12.2.4 Run CacheBench to test FDP-enabled CacheLib:

The CacheLib production workload traces of KV Cache can be obtained and run using the following steps.

1. Install and set up the AWS CLI.

2. Download the tracedata.

```
$ aws s3 cp --no-sign-request --recursive s3://cachelib-workload-sharing/pub/
kvcache/202206/ ./
```

3. Use an editor to add flags into the config_kvcache.json file to run CacheLib with FDP enabled.

   ```
   $ vi ./config_kvcache.json
   ```

   Add the following flags in the “cache_config” section.

   ```
   “navyQDepth”: 1,
   “navyEnableIoUring”: true,
   “deviceEnableFDP”: true,
   ```

   A sample config file looks like:

```
“cache_config”:
  {
    “cacheSizeMB”: 20000,
    “cacheDir”: “/root/cachelib_metadata-1”,
    “allocFactor”: 1.08,
    “maxAllocSize”: 524288,
    “minAllocSize”: 64,
    “navyReaderThreads”: 72,
    “navyWriterThreads”: 36,
    “nvmCachePaths”: [“/dev/nvme0n1”],
    “nvmCacheSizeMB” : 878700,
    “writeAmpDeviceList”: [“nvme0n1”],
    “navyBigHashBucketSize”: 4096,
    “navyBigHashSizePct”: 4,
    “navySmallItemMaxSize”: 640,
    “navySegmentedFifoSegmentRatio”: [1.0],
    “navyHitsReinsertionThreshold”: 1,
    “navyBlockSize”: 4096,
    “deviceMaxWriteSize”: 262144,
    “nvmAdmissionRetentionTimeThreshold”: 7200,
    “navyParcelMemoryMB”: 6048,
    “enableChainedItem”: true,
    “htBucketPower”: 29,
    “navyQDepth”: 1,
    “navyEnableIoUring”: true,
    “deviceEnableFDP”: true,
    “moveOnSlabRelease”: false,
    “poolRebalanceIntervalSec”: 2,
    “poolResizeIntervalSec”: 2,
    “rebalanceStrategy”: “hits”
  },
```

```
  "test_config":
  {
    "opRatePerSec": 1000000,
    "opRateBurstSize": 200,
    "enableLookaside": false,
    "generator": "replay",
    "replayGeneratorConfig":
    {
        "ampFactor": 200
    },
    "repeatTraceReplay": true,
    "repeatOpCount" : true,
    "onlySetIfMiss" : false,
    "numOps": 100000000000,
    "numThreads": 10,
    "prepopulateCache": true,
    "traceFileNames": [
        "kvcache_traces_1.csv",
        "kvcache_traces_2.csv",
        "kvcache_traces_3.csv",
        "kvcache_traces_4.csv",
        "kvcache_traces_5.csv"
    ]
  }
}
```

4. Execute the KV Cache trace workload cachebench.

```
$ ./opt/cachelib/bin/cachebench --json_test_config=./config_kvcache.json --progress=600
--progress_stats_file=cachebench-progress.log > cachebench-run.log 2>&1
```

Note: For more details on workload traces, refer to
https://cachelib.org/docs/Cache_Library_User_Guides/Cachebench_FB_HW_eval

12.3 Results

Using FDP along with CacheLib resulted in greatly improved device WAF when tested with the KV Cache production workload.

- The WAF at 100% utilization of the SSD for the NVM Cache without FDP was ~3.2 and this stayed at ~ 1 with FDP.
- No degradation was observed, with or without FDP, for metrics like latency, throughput, etc.
- FDP enablement introduced no additional application-level WAF.

13. QEMU Emulation Support

QEMU is an open source emulator and virtualizer, supporting emulation of NVMe FDP devices since v8.0. This platform can be used to experiment and familiarize yourself with FDP features. It can also be used for functional verification of newly developed FDP features and applications.

To configure the QEMU emulated NVMe device for FDP support, first, create a file to back the namespace. This example creates a 1 GB file.

```
$ qemu-img create -f raw data.img 1G
```

Because FDP is a feature that is enabled at the Endurance Group level, it is necessary to configure an “NVMe Subsystem” device that will serve as the Endurance Group “container” and configure some FDP values on it. Adding such a device requires the following command line parameter for QEMU.

```
-device “nvme-subsys,id=nvme-
subsys0,fdp=on,fdp.runs=96M,fdp.nrg=1,fdp.nruh=16”
```

This configures the endurance group with a reclaim unit nominal size of 96M (fdp.runs), 1 reclaim group (fdp.nrg), and 16 reclaim unit handles (fdp.nruh).
The next command configures the controller and sets it up to be linked to the subsystem.

```
-device “nvme,id=nvme0,serial=deadbeef,subsys=nvme-subsys0”
```

It is necessary to create a namespace. The following command configures both the “drive” (file) that holds the data and the emulated NVMe namespace.

```
-drive “id=fdp-1,file=data.img,format=raw,if=none -device nvme-ns,id=fdp-1,drive=fdp-
1,nsid=1,fdp.ruhs=1-15”
```

At this point, reclaim unit handles 1 through 15 (both inclusive) will be assigned to the namespace. Reclaim unit handle 0 will remain as a controller-specified reclaim unit handle – if other namespaces are added to the subsystem without fdp.ruhs being specified, they will automatically use reclaim unit handle 0.

`fdp.ruhs` accepts a semi-colon (“;”) separated list of identifiers and may include ranges. For example, to assign reclaim unit handle 1, 2, 3 and 8, set `fdp.ruhs` to “1-3;8” which will yield the same result as “1,2,3,8”.

Putting it all together, an example full QEMU command line could be:

```
qemu-system-x86_64 \
  -nodefaults \
  -display “none” \
  -machine “q35,accel=kvm” \
  -cpu “host” \
  -smp “4” \
  -m “8G” \
  -netdev “user,id=net0,hostfwd=tcp::2222-:22” \
  -device “virtio-net-pci,netdev=net0” \
  -device “virtio-rng-pci” \
  -drive “id=boot,file=boot.qcow2,format=qcow2,if=virtio” \
  -device “nvme-subsys,id=nvme-subsys0,fdp=on,fdp.runs=96M,fdp.nrg=1,fdp.nruh=16” \
  -device “nvme,id=nvme0,serial=deadbeef,bus=pcie_root_port1,subsys=nvme-subsys0” \
  -drive “id=fdp-1,file=fdp-1.img,format=raw,if=none” \
  -device “nvme-ns,id=fdp-1,drive=fdp-1,nsid=1” \
  -drive “id=fdp-2,file=fdp-2.img,format=raw,if=none” \
  -device “nvme-ns,id=fdp-2,drive=fdp-2,nsid=2,fdp.ruhs=1-15” \
  -serial “mon:stdio”
```

The example shown creates two namespaces within the same endurance group. Since namespace 1 does not specify `fdp.ruhs`, the controller will assign reclaim unit handle 0 and make it the controller-specified reclaim unit handle.

Boot it up and use the tools specified above to interact with it!

Things to note:

- The FDP stats and accounting are not persisted across invocations.
- QEMU NVMe emulation does not support Namespace Management, so FDP configuration must be set statically.
- The emulation differs slightly from the spec by always enabling FDP on the Endurance Group if fdp=on is set.

For more information

For more information about the Samsung Semiconductor products, visit semiconductor.samsung.com.

About Samsung Electronics Co., Ltd.

Samsung Electronics Co. Ltd inspires the world and shapes the future with transformative ideas and technologies. The company is redefining the worlds of TVs, smartphones, wearable devices, tablets, digital appliances, network systems, memory, system LSI and LED solution. For the latest news, please visit the Samsung Newsroom at news.samsung.com.

Samsung Electronics Co., Ltd.
129 Samsung-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do 16677, Korea **www.samsung.com** 1995-2021

SAMSUNG