Zoned namespace storage 분석

(ZNS SSD 중점)

Dankook Univ. Embedded Lab.

Yejin Han

목차

1. Zoned namespace interface

1.1 Introduction

■ Zoned storage는 기존의 저장 장치와는 다르게 쓰기 제약 조건이 있는 영역 (Zone)으로 주소 공간이 구분되는 저장 장치이다.

1.2 Principle of Operation

■ Zoned storage의 영역(Zone)은 순차적으로 기록되어야 한다.

■ 주소 공간의 각 영역(Zone)에는 다음 쓰기 위치를 가리키는 쓰기 포인터 (Write pointer)가 있다.

■ 영역(Zone)에는 덮어쓰기(Overwrite)가 허용되지 않는다.

■ 덮어쓰기(Overwrite)전에 zone reset 명령으로 영역(Zone)을 지워야 한다.

■ 기존의 일반 저장 장치와는 다르게 순차 쓰기 제약 조건이 있는 영역 (zone)으로 구분된다.

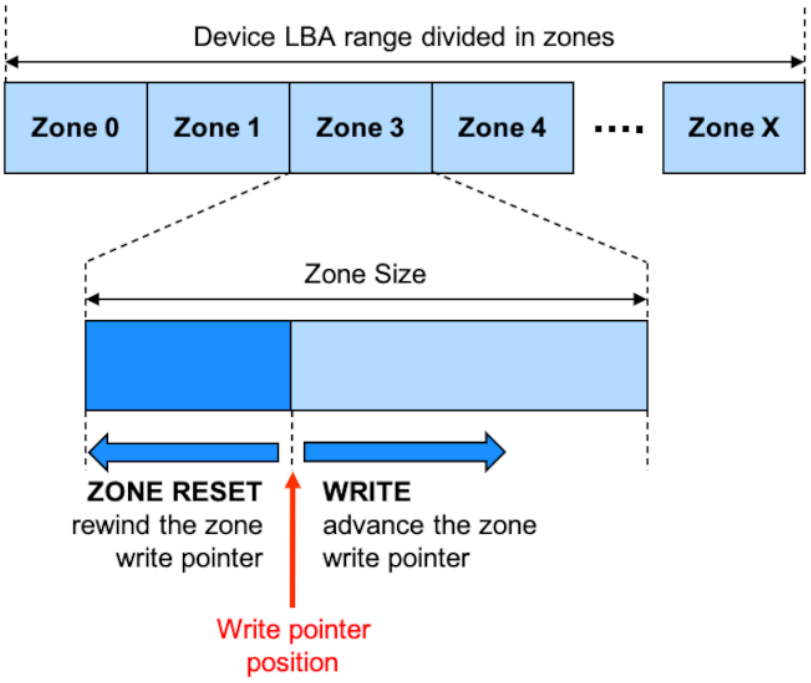

Figure 1 : Zoned Storage Devices Principle[1]

■ Zoned 스토리지 인터페이스를 구현하기 위해 NVMe 표준 위원회는 NVMe Zoned NameSpace(ZNS) 표준[2]을 발표하였다.

1.3 Zone size and Zone capacity

■ Zone size

✓ Zoned storage의 영역 크기(Zone size)는 영역 내의 총 논리 블록(Logical block) 수이다.

■ Zone capacity

✓ Zoned storage의 영역 용량(Zone capacity)은 각 저장장치에 따라 다르다.

✓ 영역(Zone)의 첫 번째 논리 블록(Logical block)부터 시작하여 영역 내에서 사용 가능한 논리 블록의 수를 나타낸다

✓ 영역 용량(Zone capacity)는 항상 영역 크기(Zone size)보다 작거나 같다.

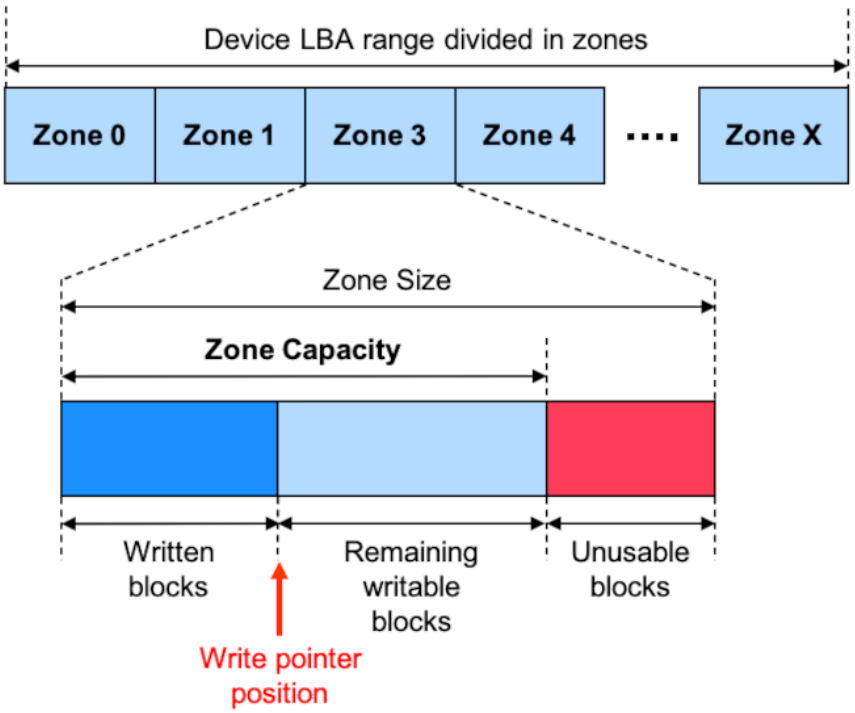

Figure 2 : Zone size and Zone capacity

✓ 영역 크기(Zone size)와 다른 영역 용량(Zone capacity)를 사용하면 기본 저장장치 특성에 대한 영역 저장 용량의 최적화된 매핑을 허용하면서 영역 크기(Zone size)가 모든 영역(Zone에 대해 일정하게 유지될 수 있다.

✓ 플래시 기반 장치의 경우 삭제 블록(Erase block)의 크기에 대해 호스트에 특정 요구사항을 부과하지 않고 영역 용량(Zone capacity)를 삭제 블록의 크기에 맞출 수 있다.

1.4 Zone models

Zoned storage의 영역(Zone) 인터페이스는 다양한 모델을 취할 수 있다. 모델의 선택은 호스트와 사용자에게 영향을 미친다. 특정 스토리지 애플리케이션에 모든 구현 옵션이 적합한 것은 아니기 때문에 모델의 차이를 이해하는 것이 중요하다.

■ Host-Managed Model

- ✓ 호스트 관리(Host-Managed) 모델은 기존 호스트 스토리지 스택과의 호환성을 제공하지 않는다.
- ✓ 순차 쓰기 워크로드만 허용하며 호스트 관리 장치를 사용하기 위해서는 호스트 소프트웨어를 수정해야 한다.
- ✓ 쓰기 포인터(Write pointer)를 사용해 영역(Zone) 내에서 순차 쓰기를 수행함으로써 순차 쓰기 제약을 지킨다.

■ Host-Aware Model

- ✓ 호스트 인식(Host-Aware) 모델은 일반 블록 장치와의 하위 호환성을 제공한다.
- ✓ 모든 섹터에 대해 임의 쓰기(Random write) 작업을 수행할 수 있다.
- ✓ 호스트 관리(Host-Managed) 모델과 동일한 제어 인터페이스를 같이 제공한다.

1.5 Zone Types

■ Conventional zones

- ✓ 전체 블록 장치 중에서 매우 작은 비율을 차지하고 있다.
- ✓ 기존 영역(Conventional zones)에 대한 접근은 일반 블록 장치의 접근과 유사하다.
- ✓ 일반적으로 메타데이터를 저장하는데 사용되며, 임의 쓰기 작업이 가능하다.
- ✓ 기존 영역(Conventional zones)에는 쓰기 포인터(Write pointer)가 존재하지 않는다.

■ Sequential-write-required zones

- ✓ 순차 쓰기 제약 영역(Sequential-write-required zones)은 임의 쓰기 작업이 불가하며 순차적으로만 쓸 수 있다.
- ✓ 기존 블록 장치와 같이 임의 읽기 명령이 가능하다.
- ✓ 모든 쓰기 명령은 영역(Zone)의 쓰기 포인터(Write pointer)와 정렬된 시작 주소를 가리켜야 한다.

1.6 Zone Management Commands

Zoned storage는 블록 장치의 기본 명령 집합과 더불어 영역(Zone) 검색 및 관리 명령을 제공한다.

■ RESET ZONE WRITE POINTER

- ✓ 호스트 소프트웨어가 영역(Zone) 쓰기 포인터(Write pointer)으 위치를 영역의 시

작 부분으로 재설정하는데 사용된다.

- ✓ 이 명령을 실행하면 영역 내의 모든 데이터가 삭제되기 때문에 이전 데이터에 대해 접근이 불가하다.

■ OPEN ZONE

- ✓ 애플리케이션이 영역(Zone)을 명시적으로 열 때 사용된다.
- ✓ 영역(Zone)이 완전히 작성되거나 CLOSE Zone 명령으로 영역을 닫기 전까지 쓰기에 필요한 자원이 사용 가능한 상태로 유지되어야 함을 장치에 알린다.

■ CLOSE ZONE

- ✓ OPEN ZONE 명령을 사용한 영역(Zone)을 명시적으로 닫을 수 있다.
- ✓ 영역(Zone)을 쓰는데 사용되는 자원이 더 이상 필요하지 않으며 해제할 수 있음을 장치에 알린다.

■ FINISH ZONE

- ✓ 애플리케이션이 쓰기 포인터(Write pointer)를 영역(Zone)의 끝으로 이동하여 재설정될 때까지 영역에 대한 추가 쓰기 작업을 방지할 수 있다.

1.7 Zone States and State Transitions

Zoned storage의 영역(Zone)에는 블록 사용량과 사용 중인 장치 자원을 나타내는 상태 속성이 존재한다.

■ Zone States

- ✓ Empty – 영역(Zone)내의 어떤 블록도 유효한 데이터를 포함하지 않는다.
- ✓ Full – 영역(Zone)내의 모든 블록이 작성되었거나 호스트가 FINISH ZONE 명령을 사용하여 작업을 완료했다.
- ✓ Implicit Open – 영역(Zone)의 일부 블록이 호스트에 의해서 최근 작성이 되었고, 장치 내부 자원을 사용하고 있다.
- ✓ Explicit Open – OPEN ZONE 명령을 통해 호스트 소프트웨어가 영역(Zone)의 장치 내부 자원을 할당받았다.
- ✓ Closed – CLOSE ZONE 명령을 통해 호스트가 명시적으로 영역(Zone)의 장치 내부 자원을 해제하였다. 또는 다른 영역(Zone)을 대상으로 하는 쓰기 작업을 위해 영역(Zone)에 할당된 내부 자원을 암시적으로 해제하였음을 의미한다.
- ✓ Read Only – 해당 영역(Zone)은 읽기만 가능하다. 일반적으로 장치의 결함 상태를 나타낸다.

✓ Offline – 영역(Zone)을 읽거나 쓸 수 없다. Read Only와 마찬가지로 장치의 결함 상태에 해당한다.

Table 1 : Zone Characteristics

State	Zone Characteristics		
	Valid Write Pointer	Active Resources	Open Resources
Empty	Yes	No	No
Implicitly Open	Yes	Yes	Yes
Explicitly Open	Yes	Yes	Yes
Close	Yes	Yes	No
Full	No	No	No
Read Only	No	No	No
Offline	No	No	No

■ State Transitions

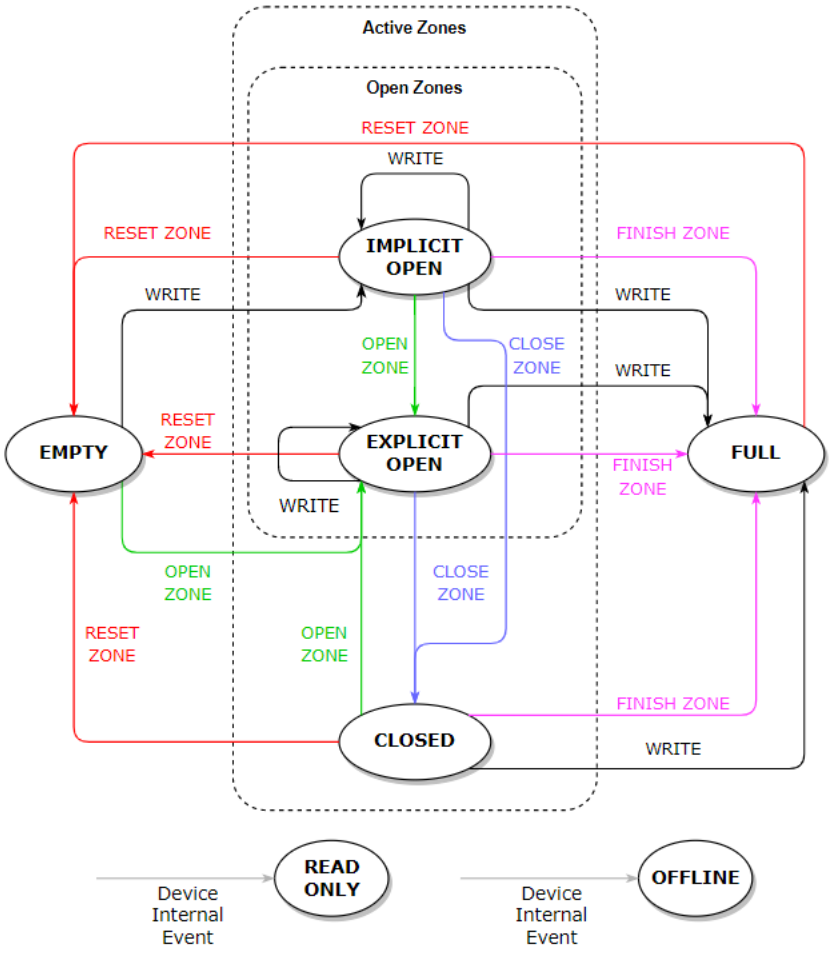

Figure 3 : Zone State Transitions Overview

✓ 읽기 전용(Read Only) 또는 오프라인(Offline) 상태로의 전환은 장치 내부 이벤트로 인해 결함이 나타난 후에 발생한다. 이 상태는 원래의 완전히 작동되는 상태로 돌아갈 수 없다.

✓ 위 두가지 경우 외 다른 모든 상태의 경우 RESET ZONE 명령을 실행하면 항상 영

역(Zone) 상태가 비어있음(Empty)으로 변경된다. 해당 영역(Zone)의 블록에 유효한 데이터가 포함되어 있지 않음을 나타낸다.

- ✓ 비어 있는(Empty) 상태의 영역(Zone)에 대한 쓰기 작업은 영역 상태를 암시적 열기(Implicit Open)으로 변경한다. 이후 암시적으로 열린 영역의 모든 블록을 쓰면 Full로 변경된다. 또한 FINISH ZONE 명령도 영역 상태를 Full로 변경한다.
- ✓ 비어 있거나(Empty) 암시적으로 열린(Implicit Open) 영역은 Explicit open명령을 이용해 명시적으로 열린 상태로 전환할 수 있다. 반대로 암시적 혹은 명시적으로 열린 명령은 CLOSE ZONE 명령을 사용해 닫힌 상태로 전환할 수 있다.

1.8 Zone Resources Limits

Zoneed storage의 영역(zone)에 쓰기 연산을 수행하기 위해서는 쓰기 버퍼(write buffer)와 같이 내부 자원의 할당을 요구한다. 또한 블록 장치는 부분적으로 쓰기가 가능한 상태의 영역(Zone)의 양을 제한할 수도 있다.

■ Open Zones Limit

- ✓ 쓰기 작업을 수행하기 위해 Zoned storage에서 사용할 수 있는 내부 자원의 총량은 제한되어 있다. 따라서 암시적으로 열린(Implicit open) 혹은 명시적으로 열린(Explicit open) 상태에 동시에 놓일 수 있는 총 영역(Zone)의 수 역시 제한이 있다.
- ✓ 이러한 제한이 있는 경우 Zoned storage는 허용된 최대 개방 영역의 수를 초과하지 않도록 쓰기 및 OPEN ZONE 명령이 실패할 수 있다.
- ✓ Open Zones Limit 제한은 읽기 작업에는 영향을 미치지 않는다.

■ Active Zones Limit

- ✓ 암시적 열림(Implicit Open), 명시적 열림(Explicit Open), 닫힘(Closed) 상태의 모든 영역은 활성 영역(Active Zones)으로 정의된다. 활성 영역은 기록 중이거나 부분적으로만 기록된 모든 영역에 해당한다.
- ✓ Zoned storage는 활성화될 수 있는 영역(Zone)의 최대 수에 제한을 둘 수 있으며, 항상 열려 있는 영역(Open Zones)의 최대 수의 제한보다 크거나 같다.
- ✓ 최대 활성 영역(Active Zones) 수는 데이터 저장을 위해 선택할 수 있는 영역 수에 제한을 둔다. 만약 최대 활성 영역 수에 도달하면 호스트 소프트웨어가 다른 영역(Zone)을 선택하여 쓰기 전 일부 활성 영역(Active Zones)을 재설정하거나 완료해야 한다.

1.9 Zone Append

■ 순차 쓰기 제약 조건은 호스트 IO 스택에 영향을 미친다. 순차 쓰기 제약이 있는 영역(Zone)으로의 쓰기 명령은 장치에서 수신하여 실행하기 전에 재정렬되면 안 된다. 그렇지 않다면,

순차 쓰기 요구 사항이 충족되지 않아 쓰기 오류가 발생한다.

■ 호스트 소프트웨어는 영역(Zone)당 쓰기 명령 수를 하나로 제한하여 쓰기 오류를 방지할 수 있다. 그러나 이로 인해 성능이 저하될 수 있으며, 특히 주로 작은 쓰기 작업(small write operation)을 실행하는 워크로드의 경우 더욱 그러하다.

■ 이를 방지하기 위해 일부 Zoned storage는 Zone Append 명령을 정의하여 영역(Zone)의 첫 번째 논리 블록을 쓰기 위치로 지정하는 쓰기 작업을 허용한다. 이 명령은 가리키는 영역(Zone) 내에서 데이터를 쓰지만 현재 영역(Zone)의 쓰기 포인터(Write pointer) 위치에서 수행한다.

■ 쓰기 위치의 변경은 자동으로 이루어지며 데이터의 유효한 쓰기 위치는 해당 명령이 완료되었다는 정보를 호스트에 보낸다. 이를 통해 호스트는 여러 Zone Append 작업을 동시에 수행하고 블록 장치에서는 임의의 순서로 이를 처리할 수 있다.

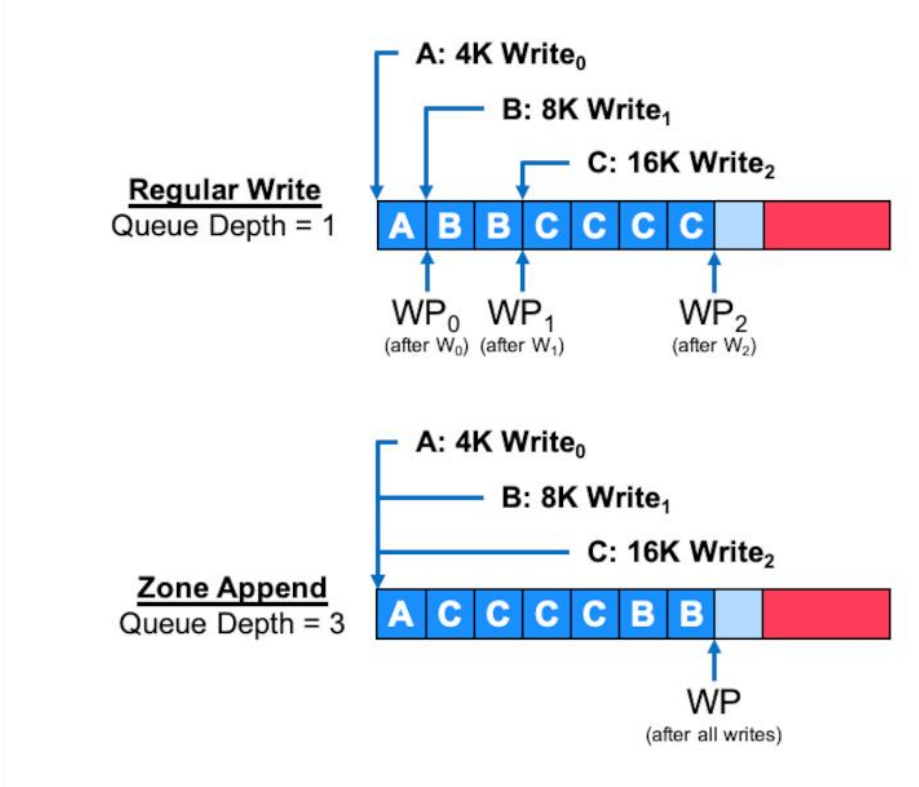

Figure 4 : Regular Writes and Zone Append Writes

- ✓ Figure 4에서 호스트는 데이터 A(4KB), B(8KB) 및 C(16KB)에 대해 동일한 영역(Zone)에 세 가지 다른 쓰기 작업을 실행해야 한다.
- ✓ 일반 쓰기(Regular Write) 명령을 사용하면 영역(Zone)당 깊이가 1인 write queue에서만 수행할 수 있다. 호스트는 다음 쓰기 요청을 발행하기 전에 끝나지 않은 쓰기 작업이 완료될 때까지 기다려야 한다.
- ✓ Zone Append 명령을 사용하면 write queue의 깊이 제약이 사라지고 호스트는 세 가지 쓰기 요청을 모두 동시에 실행할 수 있다. 그런데 명령이 장치로 가는 도중

재정렬되었을 가능성이 있으므로 영역(Zone)내에서 기록된 데이터의 위치는 호스트 명령 발행 순서와 일치하지 않을 수 있다.

2. NVMe Zoned Namespaces (ZNS) Devices

NVMe Express 기관은 NVMe 2.0 specification의 일부로 NVMe ZNS Command Set specification을 발표하였다. NVMe ZNS specification은 모든 NVMe 명령에 적용되는 인터페이스를 정의한다.

2.1 Overview

ZNS specification은 2.2의 Zoned Storage Model을 따른다. 이 표준 기반 아키텍쳐는 HDD의 SMR(Shingled Magnetic Recording)과 ZNS SSD가 통합 소프트웨어 스택을 공유할 수 있도록 스토리지에 대한 통합 접근 방식을 취한다.

특히 ZNS SSD의 경우 영역(Zone) 추상화를 통해 호스트가 쓰기를 플래시 기반 SSD의 순차 쓰기 속성에 맞출 수 있으므로 SSD에 데이터 배치를 최적화할 수 있다. 한편 스토리지의 신뢰성(reliability)는 ZNS SSD 하드웨어의 책임이며, 기존 SSD와 동일한 방식으로 관리해야 한다.

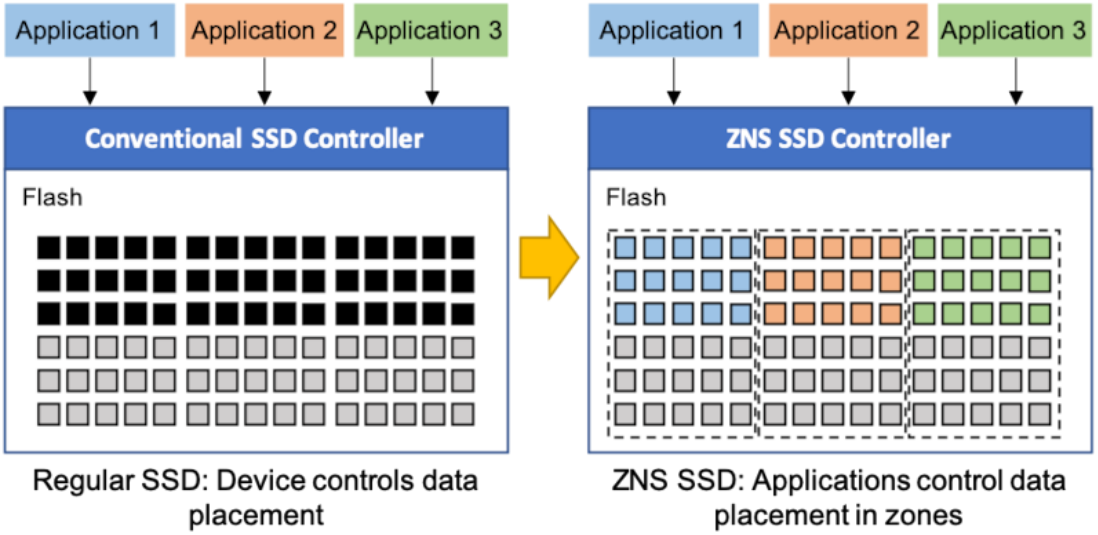

그림 5 : Conventional SSDs and ZNS SSDs internal data placement

2.2 The ZNS Zoned Storage Model

■ Overview

- ✓ ZNS command specification은 SCSI ZBC(Zoned Block Command) 표준 및 ATA ZAC(Zoned ATA Commands) 표준을 사용하여 SMR HDD용으로 처음 도입된 1.4절의 호스트 관리형 영역 지정 저장소 모델을 기반으로 한다.
- ✓ 이러한 유사성은 호스트 관리 SMR 하드 디스크와 ZNS SSD를 동시에 지원하기 위한 호스트 스토리지 스택 및 애플리케이션의 구현을 단순화한다.

- ✓ ZNS specification은 비휘발성 메모리를 사용하여 구현되는 ZNS SSD를 효율적으로 활성화하는 추가 기능을 도입한다.

■ Zone

- ✓ NVMe는 기존(Conventional) I/O 접근을 허용하는 별도의 여러 namespace를 지원한다.
- ✓ 따라서 ZNS specification은 기존 영역(Conventional zones)인 선택적 임의 쓰기 영역(Zone)을 정의하지 않는다.
- ✓ ZNS specification은 zoned namespace의 모든 영역(Zone)들이 순차 쓰기 제약을 가져야 한다고 규정하고 있다.

■ Zone Capacity and Zone Size

- ✓ ZNS specification은 ZBC 및 ZAC 표준에 정의되어 있지 않았던 영역 용량(Zone Capacity)의 개념을 도입하였다.
- ✓ 영역 크기(Zone Size)를 영역(Zone) 내의 총 논리 블록 수이며, 영역 용량(Zone Capacity)은 각 영역(Zone) 내에서 사용 가능한 논리 블록의 수를 나타내는 추가적인 특성이다.
- ✓ 이 새로운 속성은 논리 블록에서 영역(Zone) 번호로의 변환을 용이하게 하도록 영역 크기(Zone Size)가 논리 블록의 2의 거듭제곱의 수만큼 유지되도록 허용하기 위해 도입되었다.
- ✓ 영역 용량과 영역 끝 사이의 논리적 블록 주소는 물리적 스토리지 블록에 매핑되지 않으므로 이러한 블록에 대한 쓰기 액세스는 오류를 발생시킨다.
- ✓ 영역 크기(Zone size)보다 작은 영역 용량(Zone Capacity)을 가진 영역(Zone)은 쓰여진 블록의 수가 영역 용량과 같을 때 Full 상태로 전환된다.

■ Zone resources Limits

- ✓ ZNS Specification을 통해 ZNS 컨트롤러는 암시적 열기(Implicit open) 또는 명시적 열기(Explicit open) 상태에서 동시에 있을 수 있는 총 영역 수에 대한 제한을 보고할 수 있다.
- ✓ 이는 ZBC 및 ZAC 표준과 유사하게 정의되나 ZNS specification은 특히 활성 영역(Active zones)에 대한 추가 제한을 정의한다.
- ✓ 활성 영역(Active zones)은 항상 열려 있는 영역(open zone)의 최대 수의 제한보다 크거나 같다.

✓ **Table 2 : Zone Resources**

Resource	States	Comment
Active	Implicit Open Explicit Open Closed	Maximum Active Resources field에 의해 제한됨
Open	Implicit Open Explicit Open	Maximum Open Resources field에 의해 제한됨

■ Zone Append

- ✓ NVMe specification을 통해 장치 컨트롤러는 순서에 상관없이 여러 제출 큐(submission queue)에 있는 명령을 실행할 수 있다.
- ✓ 이는 호스트 IO 스택에 영향을 미치게 되는데, 호스트가 영역(Zone)에 대한 쓰기 명령을 순차적으로 제출하더라도 명령이 처리되기 전에 재정렬되어 오류가 발생할 수 있다.
- ✓ 호스트 소프트웨어는 영역(Zone)당 미해결 쓰기 명령 수를 하나로 제한하여 이러한 오류를 방지할 수 있다. 그러나 이로 인해 작은 쓰기 작업을 실행하는 워크로드의 성능이 저하될 수 있습니다.
- ✓ 이 문제를 방지하기 위해 ZNS specification은 새로운 Zone Append 명령을 선택적으로 도입하도록 하였다.

3. Linux Zoned Storage Ecosystem

3.1 Linux Zoned Storage Support Overview

■ Zoned storage 디바이스에 대한 지원은 Linux 버전 4.10.0에 추가되었다. 이로 인해 디스크 드라이버, 파일 시스템, 장치 매퍼 드라이버 등에서 zoned storage를 지원하게 되었고 애플리케이션 지원을 위한 다양한 옵션 또한 제공된다.

■ 영역 블록 장치(Zoned block device, ZBD)는 영역 저장 장치의 일반적인 표현이다. ZBD 장치 추상화와 관련된 인터페이스는 기존 Linux 블록 장치 인터페이스를 확장한 것이다.

■ ZBD 인터페이스는 파일 시스템과 같은 커널 하위 시스템과 모든 영역 장치(Zoned device) 유형 및 모든 액세스 프로토콜에서 호환되는 사용자 응용 프로그램에 대한 영역 관리 인터페이스(Zone-management interface)를 제공한다.

■ ZBD 인터페이스를 포함한 커널 구조는 아래와 같다.

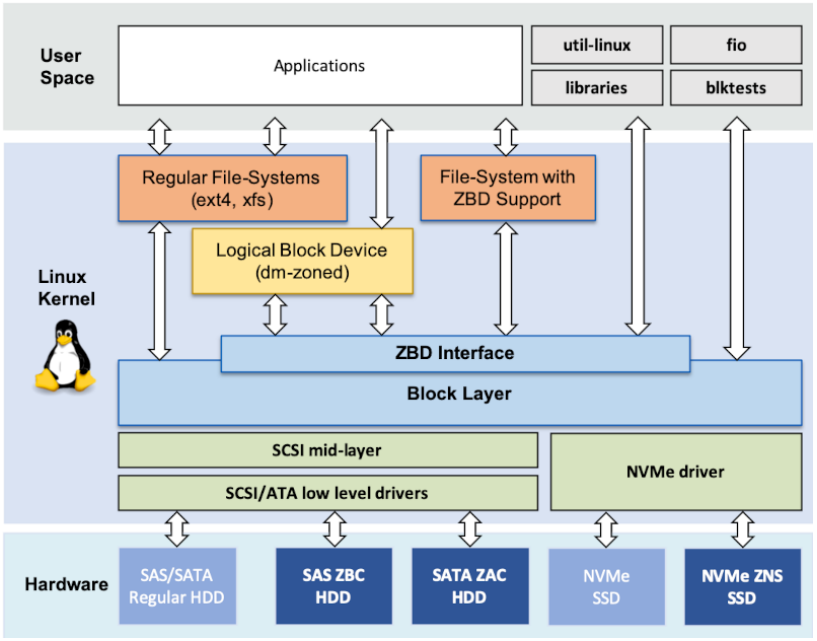

Figure 6 : Linux kernel Zoned Storage Device Support Overview

3.2 Developing for Zoned Storage

Zoned storage에 대한 시스템 응용 프로그램을 설계하는 방법에는 여러 가지가 있다. 해당 시스템 구조와 응용 프로그램의 계층의 수정 정도 및 특정 Linux 커널 버전에 따라 선택한다.

■ Direct device management with passthrough commands

- ✓ 운영 체제 및 파일 시스템에 의존하지 않고 zoned storage 디바이스를 제어하기 원하는 사용자는 passthrough 인터페이스를 통해 직접 장치 인터페이스 프로토콜을 사용할 수 있다.
- ✓ 애플리케이션은 새로운 명령 세트(command set)로 다시 작성되어야 하며, 모든 데이터 스트림이 순차적인지 확인해야 한다.
- ✓ libzbc 라이브러리는 이 접근 방식을 지원하는 기능을 제공한다.

■ Direct device management with block device file access

- ✓ 응용 프로그램 계층에서 직접 zoned storage를 관리하는 것은 구현하기 어려울 수 있다.
- ✓ 일반 POSIX 시스템 호출을 사용하여 zoned storage에 접근하고 관리할 수 있는 커널 zoned block device지원에 따르면 애플리케이션 지원을 단순화한다는 장점이 있다. 즉 순차 쓰기 스트림 및 영역 관리(zoend management)의 구현이 간소화된다.
- ✓ libzbd 라이브러리는 이 접근 방식을 지원하는 기능을 제공합니다. 또한 zonefs 파일

시스템을 사용하면 응용 프로그램의 구현도 단순화할 수 있다.

■ Device mapper and POSIX file systems

✓ ZBD compliant File System – 수정되지 않은 파일 시스템이 사용되며, 순차 쓰기 제약 조건은 zoned block device를 일반 블록 장치로 보여주는 장치 매퍼 드라이버에 의해 처리된다. 이 지원을 통해 zoned block device 위에 dm-zoned, dm-linear 및 dm-flakey를 사용할 수 있다.

✓ Legacy file systems (POSIX) - 파일 시스템은 Zoned block device의 순차 쓰기 제약 조건을 직접 처리하도록 수정된다. 애플리케이션이 수행하는 임의 쓰기는 파일 시스템에 의해 순차 쓰기로 변환된다. 응용프로그램에서 순차 쓰기 제약을 숨길 수 있으며, 이에 대한 예로 F2FS 파일 시스템이 있다.

3.3 Kernel version

커널 4.10의 zoned block device의 지원을 시작으로 ZBD 사용자 인터페이스, SCSI 계층 순차 쓰기 제어 및 F2FS 파일 시스템, 장치 매퍼 드라이버 및 블록 다중 큐 인프라 지원등이 추가되었다. 아래는 커널 버전과 함께 zoned block device 지원의 발전을 보여준다.

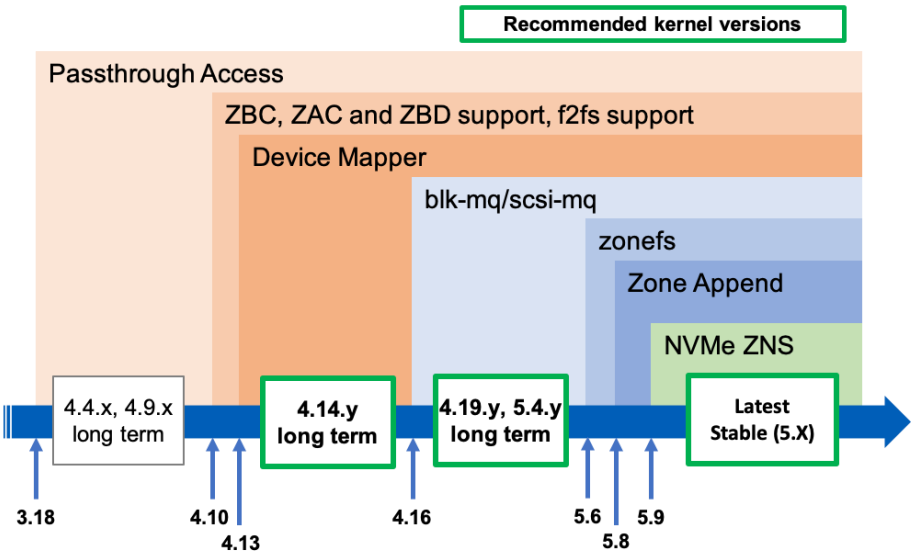

Figure 7 : Kernel versions and ZBD features

■ 권장 커널 버전은 4.14, 4.19, 5.4, 5.10, 5.15로 장기적으로 안정적인 커널 버전(Long Term Stable, LTS)이다. 이 버전은 상위 버전용으로 개발된 안정성 개선의 이점을 제공하며, zoned-block-device 지원에 대한 수정 사항도 포함한다.

3.4 Device mapper

Zoned block device 지원이 커널 버전 4.13의 장치 매퍼 하위 시스템에 추가되었다. Dm-linear, dm-flakey에 이어 ZBD 지원에 새로운 드라이버인 dm-zoned도 추가적으로 지원하도록 하였다.

■ dm-linear[3]

- ✓ dm-linear는 장치 매퍼 디바이스의 선형 범위 블록을 백엔드 디바이스의 선형 범위에 매핑한다.
- ✓ 백엔드 디바이스가 Zoned block device인 경우 모든 디바이스의 영역(Zone) 크기는 동일해야 한다.
- ✓ 대상 장치의 서로 다른 범위를 매핑하는 데 사용되는 모든 백엔드 디바이스에는 동일한 영역(Zone) 모델이 있어야 한다.
- ✓ 매핑된 범위는 정렬된 영역(zone)이어야 하며 부분 영역 매핑이 불가능하다.

■ dm-flakey[4]

- ✓ dm-flakey는 주기적으로 신뢰할 수 없는 행동을 보인다는 점을 제외하고 dm-linear와 유사하다.
- ✓ dm-flakey의 대상은 테스트 중 고장난 장치를 시뮬레이션 하는데 유용하다.
- ✓ Zoned block device의 경우 순차 쓰기 제약이 있는 영역(Zone)에 대한 쓰기 오류를 시뮬레이션하여 응용 프로그램이 쓰기 포인터(Write pointer)를 디버깅할 수 있다.
- ✓ dm-linear와 같은 제한이 dm-flakey에도 적용된다.

■ dm-zoned[5]

- ✓ dm-zoned 장치 매퍼 대상은 zoned block device(ZBC 및 ZAC 호환 장치)에 대한 임의 쓰기 접근을 제공한다.
- ✓ 기존 블록 장치에 접근하는 파일 시스템 또는 응용 프로그램으로부터 순차 쓰기 제약 조건을 숨긴다.
- ✓ 순차 쓰기 제약이 있는 영역(Zone)에 대한 임의 쓰기 접근을 처리하기 위해 온디스크 쓰기 버퍼링을 구현한다.
- ✓ 백엔드 장치의 기존 영역(zone)은 임의 접근을 버퍼링하고 내부 메타 데이터를 저장하는데 사용된다.

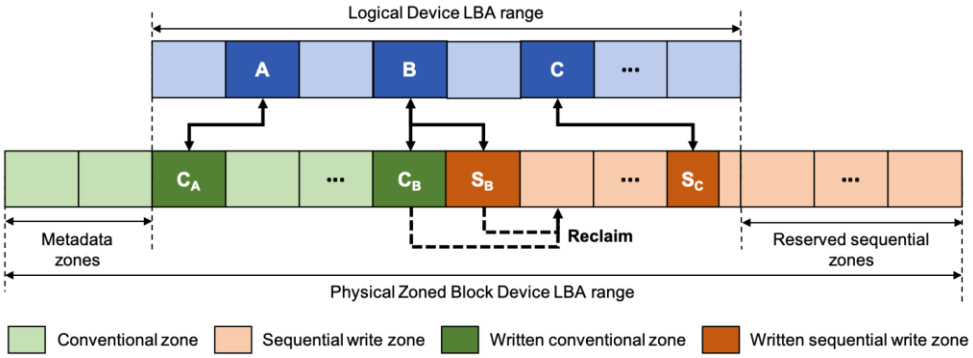

Figure 8 : Zone mapping overview of the dm-zoned device mapper target

✓ dm-zoned 대상에 사용되는 장치의 모든 영역은 두 가지 유형으로 구분된다.

Table 3 : dm-zoned target types

Type	Comment
Metadata zones	임의 쓰기가 가능한 영역(Zone)으로 메타 데이터를 저장하는데 사용된다. 사용자에게 사용 가능한 용량으로 보고되지 않는다.
Data zones	메타데이터 영역을 제외한 장치의 나머지 모든 영역(Zone)이다. 순차 쓰기 제약이 있으며, 사용자의 데이터 저장에만 사용된다.

Table 4 : dm-zoned mapping types

Type	Comment
Conventional or cache zone mapping	Chunk A의 경우로 영역(Zone) C_A에 매핑된다. 빈 청크에 첫 번째 쓰기 명령이 실행될 때 초기화되는 기본 매핑이다. 청크가 기존 영역(Conventional zone)에 매핑되는 한, 모든 쓰기 요청은 매핑된 기존 영역을 사용하여 직접 실행될 수 있다.
Sequential zone mapping	Chunk C의 경우로 영역(Zone) S_C에 매핑된다. 이 경우 이미 작성된 청크 블록을 직접 수정할 수 없다. 이를 처리하기 위해서는 다음 매핑 유형이 사용된다.
Dual conventional-Sequential zone mapping	순차 쓰기 제약이 있는 영역(Zone)에 매핑된 청크의 블록 데이터를 업데이트할 경우 청크 매핑에 기존 영(Zone)을 일시적으로 추가한다.

3.5 File Systems

■ zonefs

zonefs[6]는 zoned block device의 영역(Zone)을 파일로 보여주는 매우 간단한 파일시스템이다. Linux kernel 버전 5.6.0부터 포함되어 있다.

✓ Zoned block device의 순차 쓰기 제약 조건을 사용자에게 숨기지 않는다는 점에서 일반 POSIX 파일 시스템과 다르다.

✓ zonefs의 목표는 응용 프로그램 계층에서 zoned block device 지원의 구현을 단순화하는 것이며 원시 block device 파일 접근을 정규 파일(regular file) API로 대체함으로써 수행한다.

- ✓ 영역(Zone)을 나타내는 파일은 영역 유형(Zone type)별로 그룹화되며 영역 유형 자체는 하위 디렉터리로 표시된다.
- ✓ zonefs의 파일 구조는 디바이스에서 제공하는 영역(Zone) 정보를 사용하여 작성되므로 복잡한 온디스크 메타 데이터 구조가 필요하지 않는다.
- ✓ Zonefs의 user space tool로 mkzonefs[+]가 사용된다.

■ f2fs

- ✓ f2fs(Flash-Friendly File System)[7]에 zoned block device의 지원은 커널 버전 4.10과 함께 추가되었다.
- ✓ f2fs는 고정 블록 위치를 가진 메타데이터 블록 형식을 사용하기 때문에 기존 영역(Conventional zone)을 포함하는 zoned block device만 지원된다. 순차 쓰기 제약이 있는 영역(Zone)으로만 구성된 zoned device는 f2fs와 함께 사용할 수 없다.
- ✓ 임의 쓰기가 가능한 스토리지에 메타데이터 블록을 배치하기 위해 다중 디바이스 설정이 필요하다.
- ✓ Section Alignment – f2fs에서 section은 고정 크기 segment (2MB)의 그룹이다. Section의 segment 수는 zoned device의 영역(Zone) 크기와 일치하도록 결정된다.
- ✓ Foced LFS mode – f2fs는 segment 내에서 일부 임의 쓰기를 허용하여 블록 할당을 최적화한다. LFS 모드는 segment에 순차 쓰기 제약을 걸고 section내의 segment의 순차 사용을 강제한다.
- ✓ Zone reset as discard operation – 이전에는 block discard 또는 trim을 통해 블록이 더 이상 사용되지 않음을 디바이스에 표시하였다. 이는 zone write pointer reset 명령으로 대체되었으며, 이로 인해 section을 재사용할 수 있게 된다.

4. Tools and Libraries

4.1 Linux System Utilities

util-linux는 linux utilities 모음으로, zoned block device를 나열하고 zone configuration을 보여주기 위해 lsblk와 blkzone CLI tool을 제공한다. 이 utilities는 쉘 스크립트 및 사용자 응용 프로그램의 영역(Zone) 관리 문제를 해결하는데 유용하다.

■ lsblk

- ✓ lsblk는 block device의 유형에 관계없이 zoned block device를 포함하여 시스템의 모든 블록 장치를 나열한다.

```
root@yejin:/home/yejin# lsblk
NAME     MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda        8:0    0   477G  0 disk
└─sda1     8:1    0   477G  0 part /home/yejin/DNN
sdb        8:16   0   1.8T  0 disk
sdc        8:32   0 238.5G  0 disk
├─sdc1     8:33   0   512M  0 part
└─sdc2     8:34   0   238G  0 part /
nvme0n1 259:1     0     2T  0 disk
```

Figure 9 : lsblk output

✓ 나열된 블록 장치의 영역(Zone) 모델에 대한 표시를 보기 위해서 **-z** 옵션을 사용한다.

```
root@yejin:/home/yejin# lsblk -z
NAME    ZONED
sda     none
└─sda1  none
sdb     none
sdc     none
├─sdc1  none
└─sdc2  none
nvme0n1 none
```

Figure 10 : lsblk -z output

■ blkzone

✓ blkzone은 커널 제공 ZBD 인터페이스의 **ioctl()**을 사용하여 zoned block device의 영역(Zone)을 나열하고 순차 쓰기 영역의 쓰기 포인터(Write pointer)를 재설정 한다.

✓ 디바이스에 직접 SCSI, ATA 또는 NVMe 명령을 실행하지 않는다.

```
root@yejin:/home/yejin# blkzone --help

Usage:
 blkzone <command> [options] <device>

Run zone command on the given block device.

Commands:
 report       Report zone information about the given device
 reset        Reset a range of zones.

Options:
 -o, --offset <sector>  start sector of zone to act (in 512-byte sectors)
 -l, --length <sectors> maximum sectors to act (in 512-byte sectors)
 -c, --count <number>   maximum number of zones
 -v, --verbose          display more details

 -h, --help             display this help
 -V, --version          display version
```

Figure 11 : blkzone --help output

4.2 nvme-cli

■ nvme-cli는 오픈소스 툴로서 버전 1.12이후부터 zns 지원이 제공된다. ZNS namespace를 식별하거나 현재 영역(Zone)상태와 쓰기 포인터(Write pointer)에 대한 정보를 제공한다.

■ nvme 명령으로 영역(Zone)에 대해 open, close, finish, offline, append를 수행할 수 있다.

4.3 libzbc user library

libzbc[8]는 ZBC 및 ZAC 디스크를 조작하기 위한 기능을 제공하는 사용자 라이브러리이다. 영역(Zone) 모델과 사용 중인 디스크의 인터페이스와 독립적인 통합 API를 제공한다. 내부적으로 디바이스 인터페이스에 속한 명령을 처리하기 위해 네 가지 유형의 장치 드라이버가 사용된다.

■ Different types of device drivers

- ✓ ZAC ATA Driver – SCSI 일반 드라이버를 통해 ZAC 디스크에 ATA 명령을 직접 전달하는데 사용된다.
- ✓ ZBC SCSI Driver – 주로 ZBC SCSI 디스크를 대상으로 하는 SCSI 명령을 처리하지만 AZC ATA 디스크를 제어하는 데도 사용 가능하다.
- ✓ Zoned Block Device Driver – 커널 ZBD 인터페이스를 사용하여 ZBC 및 ZAC 디스크를 모두 제어한다.
- ✓ File Emulation Driver – 일반 파일 또는 일반 블록 장치를 백엔드 저장소로 사용하여 호스트 관리 ZB 디스크를 에뮬레이션 한다.

Table 5 : libzbc library functions

Function	Description
zbc_open()	Open a zoned device
zbc_close()	Close a zoned device
zbc_get_device_info()	Get device information
zbc_report_nr_zones()	Get the number of zones
zbc_report_zones()	Get zone information
zbc_list_zones()	Get zone information
zbc_zone_operation()	Execute a zone operation
zbc_open_zone()	Explicitly open a zone
zbc_close_zone()	Close an open zone
zbc_finish_zone()	Finish a zone
zbc_reset_zone()	Reset a zone write pointer
zbc_pread()	Read data from a zone
zbc_pwrite()	Write data to a zone
zbc_flush()	Flush data to disk

4.4 libzbd user library

libzbd[9]는 zoned block device를 조작하기 위해 함수를 제공하는 사용자 라이브러리이다. libzbc 라이브러리와 달리 zoned block device에 대한 직접 명령 접근을 구현하지 않는다. 대신 libzbd는 ioctl() 시스템 호출을 기반으로 하는 커널 제공 zoned block device interface를 사용한다. 즉 실행 중인 커널에서 지원하는 zoned block device에만 접근을 허용한다.

■ libzbd 함수는 바이트 단위를 사용하여 영역(Zone) 관련 정보를 측정하며, 영역의 시작 위

치, 크기, 쓰기 포인터(Write pointer) 위치가 포함된다.

Table 6 : libzbd library functions

Function	Description
zbd_open()	Open a zoned blockdevice
zbd_close()	Close an open zoned block device
zbd_get_info()	Get a device information
zbd_report_nr_zones()	Get the number of zones of a device
zbd_report_zones()	Get a device zone information
zbd_list_zones()	Get a device zone information
zbd_zone_operation()	Execute an operation on a range of zones
zbd_open_zones()	Explicitly open a range of zone
zbd_close_zones()	Close a range of zones
zbd_reset_zones()	Reset the write pointer of a range of zones
zbd_finish_zones()	Finish a range of zone

5. Performance Benchmarking

5.1 fio

fio(Flexible I/O tester)는 커널 block I/O stack을 테스트하기 위한 도구이다. Zoned block device에 대한 지원은 fio 버전 3.9와 함께 추가되었다.

■ Command line options

Table 7 : fio command line options – Often used

Command line option list	
filename	Test device or file path
rw	I/O types
bs	Block size
direct	True: direct I/O / False: buffered I/O
numjobs	Number of job
iodepth	Queue depth
ioengine	How the job issues I/O to the file
size	File size per job
zonemode	None, strided, zbd
zonesize	Size of a single zone when zonemode=zbd
log_avg_msec	Write log entry over the specified period of time

■ Benchmarks list

Table 8 : Benchmark list

Benchmark list	
write	Sequential writes
read	Sequential reads
randread	Random reads
Read random while writing sequentially	Mixed workload with write/reads

■ Experiment environment

Table 9 : Computer Environment

CPU	Intel® Core™ i5-3440 CPU @ 3.10GHz
RAM	32G
SSD	Skhynix PE8010 NVMe 2TB ZNS SSD

■ Performance test (write 이전 nvme format으로 초기화 작업 필수)

✓ Sequential write 성능

✓ rw=write, bs=192k, iodepth=1, direct=1, numjobs=1, ioengine=sync

```
root@yejin:/home/yejin/fiotest# fio znsfio0.fio
file1: (g=0): rw=write, bs=(R) 192KiB-192KiB, (W) 192KiB-192KiB, (T) 192KiB-192KiB, ioengine=sync, iodepth=1
fio-3.30-27-ga2840
Starting 1 process
Jobs: 1 (f=1): [W(1)][100.0%][w=98.3MiB/s][w=524 IOPS][eta 00m:00s]
file1: (groupid=0, jobs=1): err= 0: pid=427989: Wed Dec 28 01:06:48 2022
  write: IOPS=520, BW=97.6MiB/s (102MB/s)(2160MiB/22128msec); 0 zone resets
    clat (usec): min=148, max=8909, avg=1914.10, stdev=369.63
     lat (usec): min=153, max=8914, avg=1919.03, stdev=369.63
    clat percentiles (usec):
     |  1.00th=[ 1778],  5.00th=[ 1795], 10.00th=[ 1795], 20.00th=[ 1795],
     | 30.00th=[ 1811], 40.00th=[ 1860], 50.00th=[ 1893], 60.00th=[ 1909],
     | 70.00th=[ 1926], 80.00th=[ 1942], 90.00th=[ 2089], 95.00th=[ 2114],
     | 99.00th=[ 2212], 99.50th=[ 2245], 99.90th=[ 8586], 99.95th=[ 8717],
     | 99.99th=[ 8848]
   bw (  KiB/s): min=96768, max=103296, per=100.00%, avg=99962.18, stdev=1539.95, samples=44
   iops        : min=  504, max=  538, avg=520.64, stdev= 8.02, samples=44
  lat (usec)   : 250=0.26%
  lat (msec)   : 2=84.58%, 4=14.90%, 10=0.26%
  cpu          : usr=0.43%, sys=0.79%, ctx=11520, majf=0, minf=13
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,11520,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: bw=97.6MiB/s (102MB/s), 97.6MiB/s-97.6MiB/s (102MB/s-102MB/s), io=2160MiB (2265MB), run=22128-22128msec

Disk stats (read/write):
  nvme0n1: ios=0/0, merge=0/0, ticks=0/0, in queue=0, util=0.00%
```

Figure 12 : fio sequential write on ZNS SSD

✓ Random read

✓ rw=randread, bs=192k, iodepth=1, direct=1, numjobs=1, ioengine=sync

```
root@yejin:/home/yejin/fiotest# fio znsread0.fio
file1: (g=0): rw=randread, bs=(R) 192KiB-192KiB, (W) 192KiB-192KiB, (T) 192KiB-192KiB, ioengine=sync, iodepth=1
fio-3.30-27-ga2840
Starting 1 process
Jobs: 1 (f=1): [r(1)][100.0%][r=381MiB/s][r=2030 IOPS][eta 00m:00s]
file1: (groupid=0, jobs=1): err= 0: pid=429527: Wed Dec 28 02:00:59 2022
  read: IOPS=2019, BW=379MiB/s (397MB/s)(2160MiB/5703msec)
    clat (usec): min=460, max=657, avg=492.52, stdev= 9.22
     lat (usec): min=460, max=659, avg=492.67, stdev= 9.26
    clat percentiles (usec):
     |  1.00th=[  465],  5.00th=[  474], 10.00th=[  478], 20.00th=[  490],
     | 30.00th=[  490], 40.00th=[  490], 50.00th=[  498], 60.00th=[  498],
     | 70.00th=[  498], 80.00th=[  498], 90.00th=[  502], 95.00th=[  502],
     | 99.00th=[  506], 99.50th=[  506], 99.90th=[  515], 99.95th=[  523],
     | 99.99th=[  537]
   bw (  KiB/s): min=385536, max=391680, per=100.00%, avg=387979.64, stdev=2126.08, samples=11
   iops        : min= 2008, max= 2040, avg=2020.73, stdev=11.07, samples=11
  lat (usec)   : 500=86.63%, 750=13.37%
  cpu          : usr=0.89%, sys=3.23%, ctx=11520, majf=0, minf=58
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=11520,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=379MiB/s (397MB/s), 379MiB/s-379MiB/s (397MB/s-397MB/s), io=2160MiB (2265MB), run=5703-5703msec

Disk stats (read/write):
```

Figure 13 : fio random read on ZNS SSD

✓ Sequential read

✓ rw=read, bs=192k, iodepth=1, direct=1, numjobs=1, ioengine=sync

```
root@yejin:/home/yejin/fiotest# fio znsread0.fio
file1: (g=0): rw=read, bs=(R) 192KiB-192KiB, (W) 192KiB-192KiB, (T) 192KiB-192KiB, ioengine=sync, iodepth=1
fio-3.30-27-ga2840
Starting 1 process
Jobs: 1 (f=1): [R(1)][100.0%][r=382MiB/s][r=2037 IOPS][eta 00m:00s]
file1: (groupid=0, jobs=1): err= 0: pid=429594: Wed Dec 28 02:06:50 2022
  read: IOPS=2028, BW=380MiB/s (399MB/s)(2160MiB/5678msec)
    clat (usec): min=460, max=727, avg=491.54, stdev= 9.48
     lat (usec): min=460, max=729, avg=491.68, stdev= 9.52
    clat percentiles (usec):
     |  1.00th=[  465],  5.00th=[  474], 10.00th=[  478], 20.00th=[  486],
     | 30.00th=[  490], 40.00th=[  490], 50.00th=[  494], 60.00th=[  498],
     | 70.00th=[  498], 80.00th=[  498], 90.00th=[  502], 95.00th=[  502],
     | 99.00th=[  502], 99.50th=[  506], 99.90th=[  519], 99.95th=[  529],
     | 99.99th=[  553]
   bw (  KiB/s): min=387072, max=392064, per=100.00%, avg=389725.09, stdev=1746.71, samples=11
   iops        : min= 2016, max= 2042, avg=2029.82, stdev= 9.10, samples=11
  lat (usec)   : 500=92.40%, 750=7.60%
  cpu          : usr=0.39%, sys=3.47%, ctx=11521, majf=0, minf=60
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=11520,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=380MiB/s (399MB/s), 380MiB/s-380MiB/s (399MB/s-399MB/s), io=2160MiB (2265MB), run=5678-5678msec

Disk stats (read/write):
  nvme0n1: ios=0/0, merge=0/0, ticks=0/0, in_queue=0, util=0.00%
```

Figure 14 : fio sequential read on ZNS SSD

✓ Read random while writing sequentially

✓ Rw=write,randread, iodepth=1, direct=1, numjobs=1, ioengine=sync, io_uring

```
fio-3.30-27-ga2840
Starting 2 processes
Jobs: 1 (f=1): [W(1),_(1)][100.0%][r=14.6MiB/s,w=96.4MiB/s][r=3733,w=514 IOPS][eta 00m:00s]
write: (groupid=0, jobs=1): err= 0: pid=429626: Wed Dec 28 02:14:56 2022
  write: IOPS=477, BW=89.5MiB/s (93.8MB/s)(1024MiB/11445msec); 0 zone resets
    clat (usec): min=142, max=8909, avg=2088.56, stdev=400.72
     lat (usec): min=148, max=8914, avg=2093.70, stdev=400.74
    clat percentiles (usec):
     |  1.00th=[ 1778],  5.00th=[ 1811], 10.00th=[ 1876], 20.00th=[ 1909],
     | 30.00th=[ 1942], 40.00th=[ 2008], 50.00th=[ 2057], 60.00th=[ 2114],
     | 70.00th=[ 2147], 80.00th=[ 2212], 90.00th=[ 2311], 95.00th=[ 2409],
     | 99.00th=[ 2573], 99.50th=[ 2638], 99.90th=[ 8717], 99.95th=[ 8717],
     | 99.99th=[ 8848]
   bw (  KiB/s): min=82944, max=100992, per=99.80%, avg=91444.50, stdev=4031.68, samples=22
   iops        : min=  432, max=  526, avg=476.27, stdev=21.00, samples=22
  lat (usec)   : 250=0.27%
  lat (msec)   : 2=37.64%, 4=61.81%, 10=0.27%
  cpu          : usr=0.55%, sys=1.27%, ctx=5506, majf=0, minf=11
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,5462,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1
read: (groupid=0, jobs=1): err= 0: pid=429627: Wed Dec 28 02:14:56 2022
  read: IOPS=25.9k, BW=101MiB/s (106MB/s)(1024MiB/10135msec)
    slat (usec): min=2, max=188, avg= 3.84, stdev= 1.51
    clat (nsec): min=104, max=1854.8k, avg=33933.45, stdev=43632.70
     lat (usec): min=10, max=1858, avg=37.86, stdev=43.83
    clat percentiles (usec):
     |  1.00th=[    8],  5.00th=[    9], 10.00th=[    9], 20.00th=[    9],
     | 30.00th=[    9], 40.00th=[    9], 50.00th=[   10], 60.00th=[   12],
     | 70.00th=[   58], 80.00th=[   61], 90.00th=[   74], 95.00th=[   98],
     | 99.00th=[  198], 99.50th=[  302], 99.90th=[  375], 99.95th=[  388],
     | 99.99th=[  717]
   bw (  KiB/s): min=60528, max=240896, per=100.00%, avg=104051.20, stdev=47592.37, samples=20
   iops        : min=15132, max=60224, avg=26012.80, stdev=11898.09, samples=20
  lat (nsec)   : 250=0.02%, 500=0.01%
  lat (usec)   : 4=0.01%, 10=56.98%, 20=4.62%, 50=5.29%, 100=28.71%
  lat (usec)   : 250=3.65%, 500=0.72%, 750=0.01%, 1000=0.01%
  lat (msec)   : 2=0.01%
  cpu          : usr=6.34%, sys=17.71%, ctx=262116, majf=0, minf=10
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=262144,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: bw=101MiB/s (106MB/s), 101MiB/s-101MiB/s (106MB/s-106MB/s), io=1024MiB (1074MB), run=10135-10135msec
  WRITE: bw=89.5MiB/s (93.8MB/s), 89.5MiB/s-89.5MiB/s (93.8MB/s-93.8MB/s), io=1024MiB (1074MB), run=11445-11445msec

Disk stats (read/write):
  nvme0n1: ios=0/0, merge=0/0, ticks=0/0, in_queue=0, util=0.00%
```

Figure 15 : fio read random while sequential write on ZNS SSD

✓ IOPS test 결과 기존 SSD(Conventional SSD, Traditional SSD)는 garbage collection 작업으로 인해 성능이 하락하는 반면 garbage collection 작업을 수행하지 않는 ZNS SSD는 일정한 IOPS 성능을 보여준다.

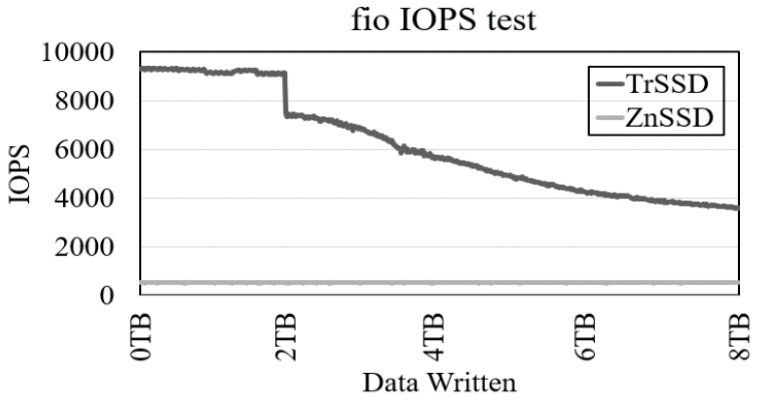

Figure 16 : Performance predictability on ZNS SSD

✓ fio write 작업이 수행되는 도중 checkpointing과 같은 다른 I/O 작업이 발생하였을 시 Performance interference에 따른 성능 결과를 보여준다[10]. 기존 SSD (Conventional SSD, Traditional SSD)는 동시에 수행되는 워크로드에 의한 간섭을 크게 받아 성능이 일정하게 유지되지 못한다. 그러나 ZNS SSD는 zone간의 독립성으로 인해 간섭에 따른 영향을 거의 받지 않아 일정하게 성능을 유지한다.

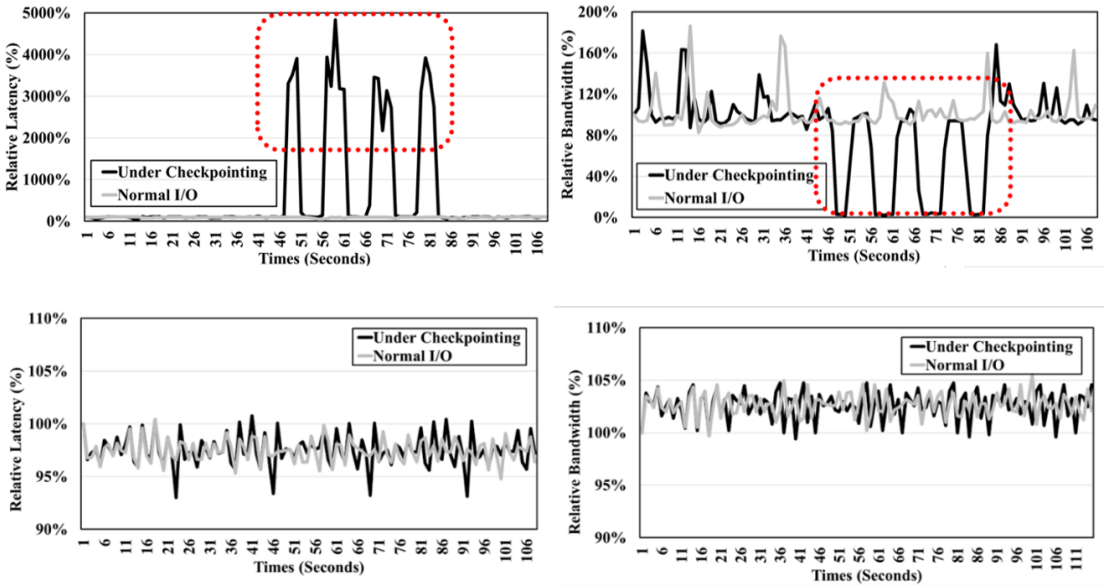

Figure 17 : Comparison of performance interference on conventional SSD and ZNS SSD

그림 목차

표 목차

Reference

[1] Zoned storage. https://zonedstorage.io/docs/introduction/zoned-storage

[2] NVM Express Zoned Namespace Command Set Specification. https://nvmexpress.org/developers/nvme-command-set-specifications/

[3] dm_linear. https://github.com/torvalds/linux/blob/81b0b29bf70bb8b459cf1f0b4a6a4898be457850/Documentation/admin-guide/device-mapper/linear.rst

[4] dm_flakey. https://github.com/torvalds/linux/blob/81b0b29bf70bb8b459cf1f0b4a6a4898be457850/Documentation/admin-guide/device-mapper/dm-flakey.rst

[5] dm_zoned. https://github.com/torvalds/linux/blob/81b0b29bf70bb8b459cf1f0b4a6a4898be457850/Documentation/admin-guide/device-mapper/dm-zoned.rst

[6] D. Moal et al., zonefs: Mapping POSIX File System Interface to Raw Zoned Block Device Accesses, USENIX VAULT, 2020

[7] C. Lee, et al., F2FS: A New File System for Flash Storage, USENIX FAST, 2015

[8] libzbc. https://github.com/westerndigitalcorporation/libzbc

[9] libzbd. https://github.com/westerndigitalcorporation/libzbd

[10] Yejin Han et al., ZNS SSD를 활용한 컨테이너 체크포인트 성능 간섭 최소화, 한국컴퓨터종합학술대회, 2022