creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사학위논문

ZNS SSD를 활용한 메모리 스왑 서브시스템 성능 최적화

Optimizing Swap Subsystem Performance by Leveraging ZNS SSD

2023년 8월

서울대학교 대학원

컴퓨터 공학부

신 희 원

ZNS SSD를 활용한 메모리 스왑 서브시스템 성능 최적화

Optimizing Swap Subsystem Performance by Leveraging ZNS SSD

지도교수 염헌영

이 논문을 공학석사 학위논문으로 제출함

2023년 7월

서울대학교 대학원

컴퓨터 공학부

신희원

신희원의 공학석사 학위논문을 인준함

2023년 7월

위 원 장	김 진 수	(인)
부위원장	염 헌 영	(인)
위 원	전 병 곤	(인)

초록

스토리지 디바이스의 발달로 빠른 블록 SSD를 메모리 스왑 디바이스로 사용할 수 있지만 스왑 디바이스 활용도가 높아질수록 시스템 성능 저하가 야기된다는 심각한 문제가 있다. 이러한 문제는 운영체제가 무효화한 스왑 슬롯(swap slot)을 SSD는 유효하다고 인식하거나 스왑 아웃 패널티를 피하기 위해 사용되는 스왑 캐시(swap cache)가 블록 디바이스 내부 GC를 인지하지 않는 것과 같은 디바이스와 운영체제 간 의미 격차(semantic gap)으로 인한 것이다. 최근 제안된 Zoned Namespace(ZNS)는 논리 주소 공간을 고정된 크기의 존(Zone)으로 매핑하는 스토리지 인터페이스이다. ZNS SSD는 낸드 플래시가 덮어쓰기(overwrite)를 지원하지 않기 때문에 데이터를 존에 추가(append)하는 방식으로 쓰기를 지원한다는 특징이 있다. 기존 블록 SSD를 사용할 때 운영체제와 디바이스 간 의미 격차로 인해 성능이 저하되는 문제를 ZNS SSD의 호스트 측 관리를 사용하여 성능 저하를 완화할 수 있다. 메모리 스왑 서브시스템에 ZNS SSD를 적용한 최신 논문인 ZNSwap [1]에서는 운영체제에서 스왑을 수행할 때 ZNS SSD의 명시적인 가비지

컬렉션을 활용할 수 있도록 하였다. 이를 위해, 운영체제가 장치의 유효한 스왑 영역을 인지하는 가비지 컬렉터를 통해 공간 효율적인 호스트 가비지 컬렉션을 제안하였다. 기존 블록 SSD를 사용할 때 운영체제와 디바이스 내부 공간 관리 간 정보의 격차로 인해 성능이 저하되는 문제를 ZNS SSD의 호스트 측 관리를 사용하여 운영체제와 디바이스가 스왑 디바이스 내부 공간 관리에 스왑 페이지 정보를 공유함에 따라 성능 저하를 완화할 수 있다고 주장하였다. 그러나 ZNSwap은 특정 논리 블록 주소(Logical Block Address, LBA) 포맷에서만 동작하며, 존의 크기가 큰 ZNS SSD에서만 실험이 수행되었다는 한계가 있다. 본 연구에서는 ZNSwap을 개선하여 일반적인 기본 LBA 포맷에서 크고 작은 다양한 존 크기에 대해 동등한 스왑 아웃 성능을 보장할 수 있도록 한다. 실험 결과, 관련 연구인 ZNSwap [1]과 비교할 때 약 480MB/s으로 동등한 최대 스왑 아웃 대역폭을 보였다.

주요어: ZNS SSD, Operating System, Memory Swap Subsystem, Garbage Collection

학번: 2021-28075

목차

초록 **i**

목차 **iii**

그림 목차 **vi**

제 1 장 서 론 **1**

1.1 연구 배경 1

1.1.1 스왑 디바이스로써의 블록 SSD 1

1.1.2 Zoned Namespace(ZNS) 2

1.2 연구 목적 3

1.2.1 블록 SSD를 활용한 스왑의 성능 저하 4

1.2.2 스왑 디바이스로써의 ZNS SSD 5

1.3 구성 7

제 2 장 배 경 **9**

2.1 운영체제의 메모리 스왑 서브시스템과 페이지 회수 9

2.2 리눅스 운영체제의 메모리 역매핑 구조 10

2.3 블록 SSD의 내부 가비지 컬렉션 11

2.4 Zoned Namespace SSD(ZNS SSD) 12

제 3 장 본 론 **15**

3.1 연구 내용 15

3.1.1 다양한 특성의 ZNS SSD를 위한 커널과 패키지 코드 수정 . 15

3.1.2 역매핑 메타데이터 저장 17

3.1.3 스몰 존 ZNS SSD를 위한 가비지 컬렉션 알고리즘 23

제 4 장 실 험 **28**

4.1 실험 환경 28

4.2 스왑 서브시스템의 성능 측정 30

4.2.1 기존 ZNSwap와 제안된 수정 ZNSwap의 스왑 아웃 대역폭 비교 30

제 5 장 관 련 연 구 **33**

5.1 메모리 스왑 디바이스로 ZNS SSD 활용 33

5.2 호스트 측 SSD 관리 34

5.2.1 Open-Channel SSD(OCSSD) 34

5.2.2 MultiStreamed SSD 35

제 6 장 결 론 **36**

6.1 결과 분석 36

6.2 연구의 한계 36

6.3 향후 연구 37

Abstract **42**

그림 목차

그림 3.1 ZNSwap의 페이지 스왑 17

그림 3.2 다양한 ZNS SSD를 위해 수정한 ZNSwap의 페이지 스왑 . . . 18

그림 3.3 Chunk Aggregator 19

그림 3.4 역매핑을 위한 Chunk 데이터 구조 21

그림 3.5 수정된 ZnGC 25

그림 3.6 수정된 새 블록 IO 구성 26

그림 4.1 ZNSwap과 비교한 평균 스왑 아웃 대역폭 31

제 1 장 서 론

1.1 연구 배경

1.1.1 스왑 디바이스로써의 블록 SSD

기존에 메모리 오버플로우를 해결하기 위해 사용하던 스왑 서브시스템은 낸드 플래시와 PCIe 인터페이스로 더욱 성능이 좋아진 블록 SSD를 스왑에 사용함에 따라, 메모리 확장을 통해 시스템 고효율을 달성하는 중요한 요소로 인식되고 있다. [2, 3] 그러나, 스왑 디바이스 활용도가 높아질수록 시스템 스왑 성능 저하가 발생한다. [1]

이러한 성능 저하는 사용자가 사용하기 편리하도록 논리적으로 추상화 되어있는 블록 인터페이스로부터 기인한다. 블록 인터페이스는 플래시 미디어의 제자리 갱신(in-place update)을 하지 못하며 실제 플래시 디바이스 내부에서는 별도로 할당된 특정 플래시 블록에 데이터를 새로 작성한다(out-of-place update). 이러한 성질을 감추기 위해 SSD 컨트롤러는 플래시 변환 계층(Flash Translation Layer, FTL)을 두어 호스트의 임의 쓰기를 플래시 미디어에 요구되는 연속 쓰기로 변환

하고, 이때 발생하는 각 블록의 논리 주소와 물리 주소 간 매핑을 관리한다. FTL은 주기적인 가비지 컬렉션(Garbage Collection, GC) 프로세스를 통해 새 쓰기를 위한 공간을 확보한다(erase-before-write).

블록 인터페이스를 통한 추상화는 사용자로 하여금 플래시 미디어 관리에 신경쓰지 않고도 SSD를 활용할 수 있지만 스왑 측면에서는 운영체제가 디바이스에 데이터를 최적으로 배치할 수 없게 한다. 스왑 디바이스에 블록 SSD를 활용함에 따른 성능 저하는 일반적으로 GC 시 발생하는 쓰기 증폭(Write Amplification Factor, WAF) 때문이다. 만약 운영체제가 관리하는 디바이스 내 유효한 스왑 아웃 페이지 블록들을 GC 프로세스가 알 수 있다면 WAF를 크게 감소시킬 수 있다.

1.1.2 Zoned Namespace(ZNS)

최근 제안된 Zoned Namespace(ZNS)는 논리 주소 공간을 고정된 크기의 존(zone)으로 매핑하는 스토리지 인터페이스이다. [4] 특히, ZNS SSD는 낸드 플래시가 덮어쓰기를 지원하지 않기 때문에 존 안에서는 순차 쓰기(sequential write)가 보장되어야 한다는 특징이 있다. ZNS가 가지는 가장 큰 장점은 웨어 레벨링(wear-leveling)이나 오류 정정(error correction)과 같은 직접적인 저수준(low-level) 관리를 하지 않고도 호스트에서의 스토리지 관리가 가능하다는 것이다. [4,5] ZNS는

지속적인 순차 쓰기를 보장하기 위해 주기적으로 존 내부의 공간을 회수(reclaim) 해야 한다. FTL이 없기 때문에 제공되는 존-리셋(zone-reset) 명령어를 호스트가 활용하여 무효화된 존 내부 데이터를 삭제하며, 이를 통해 회수된 존에 대해 다시 쓰기가 가능하다.

1.2 연구 목적

리눅스 스왑 서브시스템에서 블록 SSD를 활용할 때 디바이스 내부 GC로 인한 스왑 성능 저하를 개선한 최신 연구인 ZNSwap은 운영체제에서 ZNS SSD의 명시적 GC를 활용하여 디바이스의 유효한 스왑 영역을 인지하는 공간 효율적인 호스트 GC를 제공한다. [1] 그러나 ZNSwap은 특정 논리 블록 주소(Logical Block Address, LBA) 포맷에서만 동작하며, 존의 크기가 큰 ZNS SSD에서만 실험이 수행되었다는 한계가 있다. 본 연구에서는 ZNSwap을 개선하여 일반적인 기본 LBA 포맷에서 크고 작은 다양한 존 크기에 대해 동등한 스왑 아웃 성능을 보장할 수 있도록 한다.

1.2.1 블록 SSD를 활용한 스왑의 성능 저하

블록 SSD를 스왑 디바이스로 활용하면, 스왑 아웃 페이지가 많아질 수록 디바이스 활용률은 높아지지만 스왑 아웃 대역폭이 감소한다. 디바이스 내부 GC의 데이터 복사 오버헤드로 인해 스왑 아웃 대역폭과 경쟁하기 때문이다. 특히, 스왑 디바이스 측면의 디바이스 활용률은 운영체제가 인식하고 있는 것보다 더 높게 인지한다. 이는 운영체제가 무효화한 스왑 슬롯을 SSD는 알 수 없기 때문에 디바이스 내부 GC에서 스왑 아웃 되었으나 운영체제에 의해 이미 무효화된 페이지까지 유효하다고 인식하기 때문이다. 이러한 문제를 해결하기 위해 리눅스 운영체제에서는 SSD에 힌트를 주는 TRIM 기능을 구현하였다. [6] 그러나 TRIM은 무효화된 스왑 슬롯이 논리 주소 공간에서 연속적으로 배치되어 있어야 한다는 제약이 있다. 이때 스왑에서는 멀티 스레드가 쓰기를 요청함에 따라 디바이스에서 쓰기 요청이 섞이기 때문에 무효화된 스왑 슬롯의 연속 LBA 보장이 어렵다. 위와 같은 이유로 스왑에서의 실질적인 GC 오버헤드를 줄이는 영향이 미미함에 따라 배포된 운영체제에서는 스왑에서 TRIM 사용을 하지 않고 있다. [7]

또, 리눅스 운영체제에서 스왑 아웃 패널티를 피하기 위해 사용되는 스왑 캐시(swap cache)가 블록 디바이스 내부 GC를 인지하지 않기 때문에 성능 저하가

발생한다. 스왑 캐시는 메모리로 스왑 인(swap-in)되는 페이지들로 구성되며, 운영체제는 스왑 캐시에 대해서도 스왑 디바이스에 복사본을 유지한다. 스왑 디바이스의 활용률이 미리 정해진 값인 50%를 초과하면 새로 스왑 인 된 페이지를 스왑 캐시에 추가하는 것을 멈추고 해당 스왑 슬롯을 바로 무효화한다. 즉, 이러한 페이지들에 대한 스왑 아웃(swap-out) 패널티는 페이지들이 스왑 캐시에 있더라도 메모리에서 페이지를 버리지 않고 스왑 디바이스에 페이지를 쓰는 것을 초래한다. 스왑 디바이스가 가득 차게 되면 스왑 슬롯 배열을 선형 탐색하는 스왑 슬롯 할당 알고리즘으로 인해 속도가 느려진다. 메모리의 스왑 캐시에 있는 페이지의 스왑 슬롯 무효화를 미루고 스왑 디바이스에 페이지를 계속 스왑 아웃 하게 되면 디바이스 활용률이 증가하게 되고 불가피한 블록 디바이스 내부 GC로 인해 성능이 저하된다.

1.2.2 스왑 디바이스로써의 ZNS SSD

이미 키-밸류 스토리지와 파일 시스템과 같은 다양한 연구 분야에서 애플리케이션과 디바이스 통합에 ZNS SSD의 호스트 측 물리적 데이터 배치 및 관리의 용이함을 이용한 최적화 시도가 존재한다. [8–11] 기존 블록 SSD를 사용할 때 운영체제와 디바이스 내부 공간 관리 간 의미 격차(semantic gap)로 인해 성능이

저하되는 문제를 ZNS SSD의 호스트 측 관리를 사용하여 운영체제와 디바이스가 스왑 디바이스 내부 공간 관리에 스왑 페이지 정보를 공유함에 따라 성능 저하를 완화할 수 있다.

블록 SSD에서의 GC 프로세스는 FTL을 통해 유저에게 보이는 LBA가 공간 회수 이후에도 유지되기 때문에 호스트에서 회수된 블록의 위치 등 메타데이터 관리를 별도로 하지 않고도 디바이스 내부 논리 블록의 이동이 자유롭다. 반면 FTL이 없는 ZNS SSD를 스왑 디바이스로 사용하게 될 때에는 각 4KB 블록에 대한 역매핑(reverse-mapping) 메타데이터를 관리하여 GC 이후에 변경된 스왑 아웃 페이지가 저장된 블록의 LBA를 페이지 테이블에 갱신해야 한다.

기존 ZNSwap에서는 역매핑 메타데이터를 기업형 SSD 컨트롤러에서 제공하는 특정 LBA 포맷을 활용하여 저장한다. 이때 사용되는 LBA 포맷은 일반적인 것과 다르게 LBA 마다 일정 영역을 제공하여 해당 LBA에 대한 부가적인 메타데이터를 유저가 저장할 수 있다. [1] ZNSwap은 이 영역을 이용하여 호스트가 ZNS SSD의 GC를 수행한 뒤 옮겨진 스왑 아웃 페이지들의 새 LBA를 각 페이지 테이블에 갱신한다. 그러나 이러한 추가 영역은 일반적인 SSD에서는 제공하지 않는 포맷이라는 단점이 존재한다. 본 논문에서는 일반 LBA 포맷만을 가지는 ZNS SSD

에서도 역매핑 메타데이터를 활용할 수 있도록 한다.

ZNS SSD는 존의 크기에 따라 나눌 수 있다. 약 100MB 이하 크기의 존을 가지는 ZNS SSD를 스몰 존 ZNS SSD, 그 이상 크기의 존을 가지는 ZNS SSD를 라지 존 ZNS SSD라고 나누고 있으며, ZNSwap의 경우에는 라지존 ZNS SSD에 초점을 두었다. ZNS SSD는 존 크기에 따라 내부 병렬 단위 구성이 달라지기 때문에 성능과 그에 따른 특성이 다르다. 본 연구에서는 라지 존 ZNS SSD 뿐만 아니라 스몰 존 ZNS SSD에서도 스왑 성능이 동등하게 보장되도록 ZNSwap을 최적화한다.

1.3 구성

본 논문은 다음과 같은 구성을 갖는다.

- **제 2 장**은 배경 지식으로 기존 Linux 운영체제 스왑 과정과 역매핑 방식, 블록 SSD의 내부 공간 회수 방식, ZNS SSD의 특성에 대해 설명한다.
- **제 3 장**은 스왑 디바이스로 ZNS SSD를 사용할 때의 이점과 기본 LBA 포맷을 활용하는 자세한 방법론에 대해 설명한다.
- **제 4 장**은 실험 방법 및 결과에 대해서 설명한다.

- **제 5 장**은 관련 연구를 통해 다양한 SSD를 스왑 디바이스에 활용한 예시를 설명한다.

- **제 6 장**은 연구의 요약 및 결론을 짓는다.

제 2 장 배 경

2.1 운영체제의 메모리 스왑 서브시스템과 페이지 회수

메모리 사용량이 많아지면 운영체제는 스왑 디바이스로 특정 페이지들의 스왑 아웃(swap-out)을 수행한다. 스왑 아웃은 스왑 디바이스로 보내 메모리에서 제거할(evict) 메모리 페이지를 선택하고 이 페이지가 포함된 페이지 테이블에서 매핑 해제 후 스왑 디바이스에 쓰는 일련의 동작이다. 리눅스 운영체제는 스왑 디바이스의 공간을 메모리 페이지 크기인 4KB 블록의 스왑 슬롯(swap slot)으로 나눈다. 운영체제는 스왑 아웃되는 각각의 페이지에 대해 새 스왑 슬롯을 할당한다. 페이지가 스왑 인(swap-in) 되고 스왑 디바이스의 활용량이 특정 비율 미만일 때, 운영체제는 스왑 인 페이지에 대한 스왑 아웃 패널티를 피하기 위해 스왑 디바이스에 수정되지 않은 스왑 인 메모리 페이지의 복사본을 유지한다. 즉, 페이지의 복사본이 스왑 디바이스와 메모리에 모두 보관이 되며 이를 스왑 캐시(swap cache)라고 한다. 리눅스 운영체제의 경우, 이러한 스왑 캐시 페이지들이 디스크의 공간에서 50%로 점유할 수 있도록 정적으로 고정되어 있다. 운영체제는 스왑 디바이스에

다시 쓰지 않고 스왑 캐시 페이지들을 제거한다. 페이지 크기의 스왑 슬롯은 스왑 캐시 페이지에 처음 쓸 때 해제되고, 페이지는 스왑 캐시에서 제거된다.

2.2 리눅스 운영체제의 메모리 역매핑 구조

프로세스 주소 공간의 가상 메모리 페이지는 가상 메모리 할당을 나타내는 가상 메모리 영역(vma)에 속하며, vma는 페이지 테이블 디렉토리(pgd)를 보유하는 프로세스의 가상 메모리 주소 공간(mm_structs)에 속한다. 물리 페이지 디스크립터(struct page)는 역매핑을 가능하게 하는 메타데이터를 가지고 있다. 페이지가 회수되어야 할 때 역매핑 과정이 수행되는데 anon_vma를 통해 vma들의 연결리스트를 순회하며 페이지들을 매핑 해제한다. 각 페이지 구조체(struct page)의 mapping 필드는 anon_vma를 가리키며, anon_vma는 페이지 프레임이 매핑된 가상 주소를 index 필드에 씀에 따라 vma들을 연결하는 이중 연결리스트를 관리한다. 이때, vma는 가상 메모리 시작 주소가 변경되더라도 연결 리스트로 이루어져 있기 때문에 알맞게 다시 찾아갈 수 있다. 페이지를 매핑할 수 있는 각 vma의 mm_struct에 접근하고 페이지 테이블 엔트리를 찾기 위해 페이지 테이블을 탐색한다. 페이지 테이블을 탐색하는 데 사용되는 가상 주소를 계산하기 위해 vma의 시작 가상

메모리 주소와 함께 인덱스가 사용된다.

2.3 블록 SSD의 내부 가비지 컬렉션

블록 SSD의 FTL은 LBA(Logical Block Address)를 디바이스의 이레이즈-블록(erase-block) 내 데이터의 PBA(Physical Block Address)에 매핑한다. 논리 블록에 대한 업데이트는 별도의 이레이즈-블록에 새 데이터를 쓰고, 호스트 LBA를 새 블록에 다시 매핑한 다음 이전 블록에 대해 무효화하는 방식으로 수행한다. 새로운 쓰기를 위한 공간을 확보하기 위해 SSD 컨트롤러에 의해 디바이스 내부 GC가 수행된다. GC는 무효화된 블록을 회수하고 여러 개의 이레이즈-블록들의 유효 블록들을 하나의 새 이레이즈 블록으로 통합한 다음 해제된 이레이즈 블록들을 삭제함에 따라 공간을 회수한다. GC 중에 발생하는 데이터 복사의 수를 줄이기 위해 플래시 미디어의 오버 프로비저닝(Over Provisioning, OP) 영역이 필요하다. OP 영역이 작을 수록 GC에서의 데이터 복사량이 큼에 따라 WAF가 높아지며 유저가 체감할 수 있는 디바이스의 성능이 낮아진다. 운영체제는 이러한 SSD 내부 GC의 부하를 줄이기 위해 특정 LBA들의 할당 해제에 대해 힌트를 주는 TRIM 기능을 사용한다. [6] TRIM 기능은 디스패칭(dispatching) 오버헤드와 긴 대기 시

간(latency)등의 불필요한 추가 오버헤드로 인해 스왑 시스템에서 거의 사용하지 않는 추세이다. ZNSwap에서는 운영체제의 페이지 관리 구조를 직접 활용할 수 있음에 따라 TRIM 기능 없이 GC 오버헤드를 줄일 수 있다. [1]

2.4 Zoned Namespace SSD(ZNS SSD)

Zoned Namespace(ZNS)는 새롭게 추가된 NVMe 스펙 중 하나로, 논리 주소 공간을 고정된 크기의 존(zone)으로 나누어 구성한다. 각각의 존은 물리적으로 SSD의 이레이즈-블록(erase-block) 크기에 맞게 구성된다. 존 내부에서 임의 읽기가 가능하지만 쓰기는 순차 쓰기가 보장되어야 한다. 쓰기는 일반적인 쓰기(write) 명령어나 존-추가(zone-append) 명령어를 통해 수행할 수 있다. 존-추가 명령어는 호스트가 지정한 존에 쓰기를 수행하고 SSD가 내부적으로 관리하는 쓰기 포인터를 통해 쓰여진 블록 위치를 I/O 수행 완료(completion)시 반환하기 때문에 일반 쓰기 명령과 달리 동일 존에 대해 여러 개의 존-추가 요청을 허용한다.

SSD의 내부는 여러 개의 채널(channel)로 구성이 되며, 채널 하나에 여러 개의 웨이(way)가 있고 하나의 웨이에 플래시가 붙게 된다. 웨이는 내부 병렬 단위(internal parallel unit)로 서로 독립적으로 실행이 가능하다. 존을 구성하기 위해

서는 플래시 여러 개를 조합해야 하는데 이때의 디바이스 내부 병렬 단위의 구성에 따라 ZNS SSD는 다양한 특성을 가진다. 예를 들어, 서로 다른 채널의 플래시를 한 개의 존으로 사용하는 경우에는 존 내부 병렬성(intra-zone parallelism)이 보장된다. 반면 한 채널의 플래시를 한 개의 존으로 사용하는 경우에는 존 내부 병렬성은 없지만 존 간의 병렬성(inter-zone parallelism)은 상대적으로 높아진다. 예를 들어, 존의 크기가 크다면 상대적으로 존 하나가 많은 채널들로 구성되어 존 내부 병렬성을 활용할 수 있기 때문에 개별 존의 성능이 우수할 가능성이 높다. 이때, 존의 크기가 클 수록 개별 존 리셋에 걸리는 시간이 길 가능성이 높으며, 존 안의 모든 데이터가 무효화될 가능성이 적으므로 상대적으로 무효 존 회수가 더디어 단편화(fragmentation)가 심해진다. 반면, 존의 크기가 작으면 개별 존의 성능이 상대적으로 낮을 수 있으므로 작은 크기의 존 에서는 존 간의 병렬성을 활용하여 여러 개의 존을 입출력에 병렬화하는 것이 중요하다. 존의 크기가 작다면 존 내부 데이터가 모두 무효화 될 확률이 큼에 따라 상대적으로 존 회수 오버헤드가 적고 단편화 가능성이 적다.

각 존의 상태는 크게 3가지로, 비어있는 초기 상태(Empty), 첫 번째 쓰기 후 열린 상태(Open), 존 사용량을 다 채워 더이상 쓸 수 없는 상태(Full)으로 구분된

다. SSD는 각각의 열린 존의 다음 쓰기 포인터를 유지한다. 존에 대해 다시 쓰기(rewrite)를 수행하려면 존을 리셋(reset)하여 비어있는 상태로 전환해야 한다. 내부 하드웨어 구성에 따라 동시에 열 수 있는 존의 최대 수는 제한이 되어 있다.

제 3 장 본 론

3.1 연구 내용

본 연구에서는 ZNSwap를 수정하여 메타 영역을 지원하는 특정 SSD 컨트롤러의 LBA 포맷이 아닌 일반 LBA 포맷에 대해 역매핑(reverse-mapping) 메타데이터를 관리하도록 한다. 또, 존의 크기가 큰 ZNS SSD 뿐만 아닌 작은 크기의 존을 가지는 ZNS SSD에 대해서도 동일한 성능을 보장할 수 있도록 가비지 컬렉션 알고리즘을 수정한다. 이를 통해, 다양한 특성을 가지는 ZNS SSD에 대해 동등한 스왑 성능을 보장한다.

3.1.1 다양한 특성의 ZNS SSD를 위한 커널과 패키지 코드 수정

ZNS SSD는 물리적인 채널과 웨이 등의 디바이스 내부 병렬 단위를 어떻게 한 개의 존으로 구성하는지에 따라 갖는 특성이 다양하다. 본 논문에서는 연구에 앞서, 여러 특성을 가지는 ZNS SSD를 위해 기존 ZNSwap 커널 코드와 리눅스 표준 패키지인 util-linux의 코드를 수정하였다.

non-power-of-2 문제 해결을 위한 커널 드라이버 코드 수정

리눅스 커널 블록 계층은 ZNS 블록 디바이스를 인식할 때 다음과 같은 과정을 거친다. 먼저 NVMe 디바이스 드라이버 계층에서 PCI 디바이스를 인식한다. 이때 Zoned Namespace를 지원하는 디바이스라고 인식을 하여 해당 정보를 블록 계층에 등록한다. 등록된 정보로는 존 크기(zone size)와 존 용량(zone capacity) 등이 있으며 블록 계층에서 존 크기가 2의 승수인지 검사하게 된다. 일반적으로 블록 계층에서 입출력 수행 단위가 페이지 단위인 4KB와 2MB로 최적화가 되어있다. 그렇기 때문에 존 크기가 2의 승수가 아닌 경우에는 커널에서 해당 ZNS SSD를 인식하지 못하는 문제가 발생한다. 본 논문에서는 리눅스 커널에서 블록 계층으로 ZNS SSD를 등록하는 과정에서 존 크기를 2의 승수로 변경하고 실제 사용가능한 크기로 존 용량은 그대로 둠에 따라 커널이 디바이스를 인식하고, 디바이스 사용 크기에 문제가 없도록 코드를 수정하였다.

Util-linux 리눅스 패키지 수정

스왑 디바이스를 설정함에 있어서 파일시스템의 mkfs와 같이 mkswap을 수행하여 사용할 스왑 디바이스에 스왑 헤더를 작성한다. mkswap 후 swapon을 통해

스왑 디바이스에 적힌 스왑 헤더를 읽어 확인 후 메모리에 올리고 스왑에 해당 디바이스를 사용하게 된다. mkswap과 swapon은 리눅스 표준 패키지인 util-linux에서 제공된다. 디바이스 내부 병렬 단위의 구성에 의해 스몰 존 ZNS SSD 중 네임스페이스의 논리 블록에 최소 쓰기 단위(write granularity)를 가지는 경우가 있다. 이때 페이지 단위인 4KB가 아닌 다른 크기의 쓰기 단위를 가진다면, util-linux의 mkswap과 swapon을 수정하여 입출력 오류가 나지 않도록 해야 한다.

3.1.2 역매핑 메타데이터 저장

기존 ZNSwap에서의 역매핑 메타데이터 저장 방식

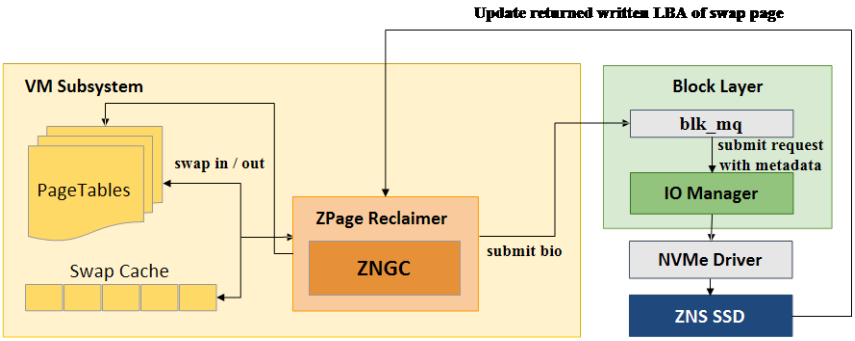

그림 3.1 ZNSwap의 페이지 스왑

논문 [1]에서 제안하는 페이지 스왑 아웃과 함께 역매핑 메타데이터를 저장하는 순서는 그림 3.1과 같다. 운영체제에 의해 스왑 아웃될 페이지가 선택된 후 페이지

테이블 및 스왑 캐시를 처리하는 ZPage Reclaimer로 보내진다. 이때 ZNSwap에서는 ZNS에서 제공하는 존-추가(zone-append) 명령어를 사용하여 페이지를 쓴다. 페이지를 쓰기 이전에 Zpage Reclaimer는 ZnGC 여부를 판단하여 새로운 존을 할당하거나 기존에 사용하던 존을 활용하여 존-추가를 수행할 존을 결정한다. 페이지는 블록 계층에 제출되며, 페이지를 ZNSwap의 IO Manager로 전달한다. IO Manager는 가능할 때마다 존-추가 작업에 대해 블록 플러그(block plugging)을 통해 통합하고 페이지 데이터와 함께 ZNGC에서 사용하는 역매핑 메타데이터를 포함하여 입출력 요청을 생성한다. 최종적으로 IO Manager는 입출력 요청을 ZNS SSD에 쓰는 NVMe 드라이버로 전달하고 ZPage Reclaimer가 SSD에 저장된 페이지 LBA를 업데이트하게 한다.

제안된 역매핑 메타데이터 저장 방식

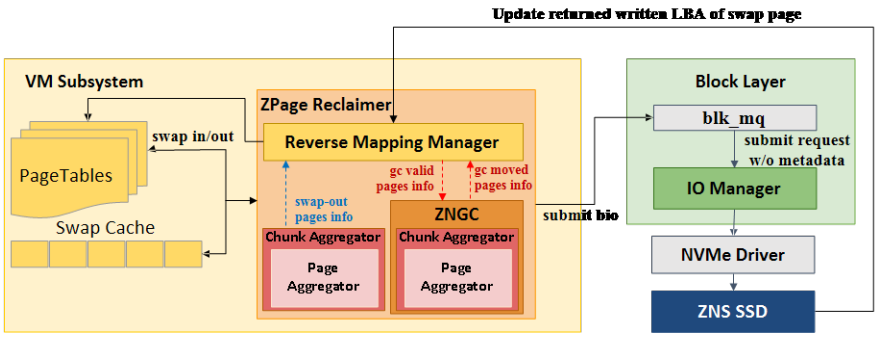

그림 3.2 다양한 ZNS SSD를 위해 수정한 ZNSwap의 페이지 스왑

그림 3.2은 본 논문에서 제안하는 다양한 쓰기 단위를 갖는 ZNS SSD를 위해 Chunk Aggregator를 적용한 스왑 방식이다. 디바이스 내부 병렬 단위를 어떻게 한 존에 구성하는지에 따라 최소 쓰기 단위가 블록 단위인 4KB와 다를 수 있으므로 Chunk Aggregator는 이러한 최소 쓰기 단위의 크기인 Chunk로 여러 개의 페이지를 모아서 블록 IO로 내리게 된다. 각각의 페이지에 대해 역매핑 메타데이터를 관리하는 Reverse Mapping Manager를 메모리에 두었다. 역매핑 메타데이터는 기존 리눅스 기반 역매핑 구조에서 사용하는 데이터와 동일하다. Reverse Mapping Manager를 통해 GC 수행 시 새로운 위치에 쓰여지는 페이지의 페이지 테이블 엔트리를 찾아갈 수 있도록 역매핑 메타데이터를 활용하게 된다.

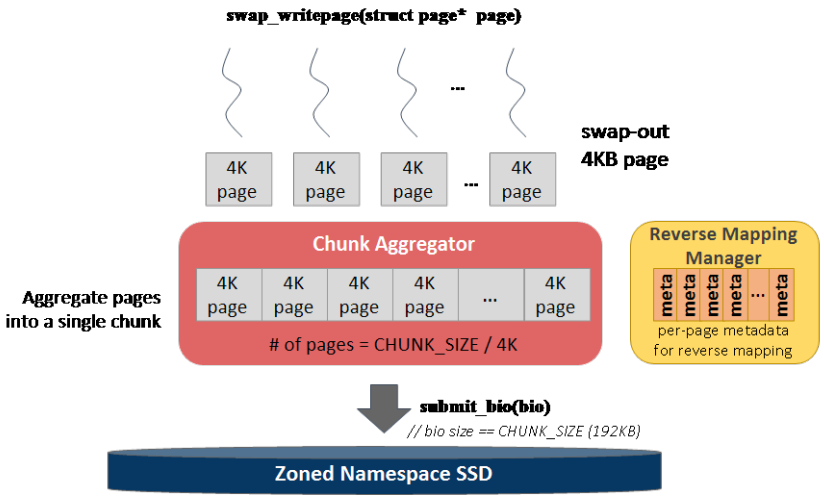

그림 3.3 Chunk Aggregator

그림 3.3은 제안된 Chunk Aggregator의 동작 방식을 나타낸다. swap_writepage 함수를 통해 여러 개의 스레드가 스왑 아웃된 페이지들을 블록 IO(struct bio)로 제출하고자 한다. Chunk Aggregator는 블록 IO를 구성하고자 하는 페이지들을 최소 쓰기 단위에 맞게 Chunk 내 포함되어야 할 페이지 수를 계산하여 그 수만큼 모아서 블록 IO를 수행한다. 이때 블록 IO 구조체는 각각의 활성화된 존에 대해 하나씩 가지고 있도록 하여 같은 존에 스왑 아웃되고자 하는 페이지들을 모을 수 있도록 함에 따라 여러 존을 스왑 아웃에 활용할 수 있도록 하였다. 그림 3.3에서는 192KB의 Chunk 크기를 갖는 쓰기 단위의 ZNS SSD에 대한 예시이며, 이 경우 한 개의 Chunk에 48개의 페이지가 포함될 수 있다. 이때, 마지막 한 개의 페이지에는 스왑 아웃된 47개의 페이지에 대한 역매핑 메타데이터를 배열로 저장하고 있다. 역매핑 메타데이터에 사용되는 메타데이터는 기존 리눅스 역매핑과 ZNSWap 역매핑에 사용되는 anon_vma 포인터인 매핑과 페이지 인덱스, 접근된 비트맵 등이 있다.

그림 3.4는 역매핑을 위한 Chunk의 데이터 구조를 세부적으로 나타낸 것이다. 하나의 Chunk는 ZNS SSD가 가지는 최소 쓰기 단위에 따라 크기가 달라진다. 그림 3.4에서 사용된 예시는 192KB 최소 쓰기 단위를 가지는 ZNS SSD에서의

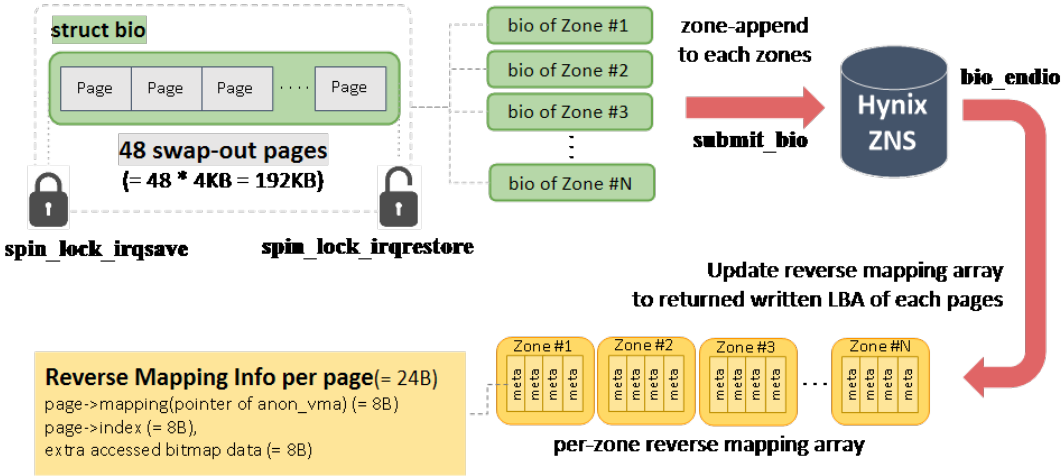

그림 3.4 역매핑을 위한 Chunk 데이터 구조

Chunk 데이터 구조이다.

일반적으로 리눅스에서 블록 IO는 페이지 단위인 4KB로 입출력을 수행한다. 본 연구에서는 최소 쓰기 단위를 위해 여러 개의 페이지로 구성된 하나의 Chunk를 블록 IO의 단위로써 사용한다. 이때 최소 쓰기 단위 크기를 페이지 크기인 4KB로 나누어 Chunk 내에 포함되어야 할 페이지의 수를 정한다. GC에서 사용되는 페이지 역매핑 메타데이터는 8바이트의 anon_vma 포인터, 8바이트의 페이지 인덱스 오프셋, 8바이트의 접근 비트맵 데이터로 구성되며 한 개의 4KB 페이지 당 24바이트가 필요하다. 페이지를 Chunk에 추가할 때마다 이러한 역매핑 메타데이터를 구성하여 Reverse Mapping Manager에 관리한다.

스왑 동작이 멀티 스레드로 동작하고, 디바이스 내부 병렬성을 효과적으로 활

용하기 위해서는 여러 존에 동시에 스왑 아웃되도록 하는 것이 중요하다. 이를 위해 본 연구에서는 존 별로 블록 IO 구조체(struct bio)를 두고 블록 IO는 Chunk 크기에 맞을 때까지 bio_add_page를 통해 페이지를 bio에 추가한다. 각각의 블록 IO는 스왑 존과 1:1로 할당되어 여러 개의 존에 스왑 아웃이 내려갈 수 있도록 하여 존 간 병렬성을 활용할 수 있다. 스왑 아웃될 때 Chunk 단위로 내린 페이지 역매핑 메타데이터를 이용하여 GC를 수행한 유효 스왑 페이지의 페이지 테이블 엔트리에 LBA 업데이트를 수행한다.

구현 과정에서 각 Chunk를 페이지들로 구성하는 도중 멀티 스레드 간 컨텍스트 스위칭으로 인해 ZNS SSD를 인식하지 못하는 문제가 있었다. 이는 컨텍스트 스위칭이 되어서 스왑 이외의 스레드가 다른 락을 잡을때 swap on을 한번 더 호출하게 되는 경우가 있는데 이때 ZNS 디바이스의 정보를 저장하는 구조체가 초기화 후 재설정되기 전에 다시 컨텍스트 스위칭이 발생하였기 때문이다. 따라서 페이지를 Chunk 크기의 bio로 모으는 동안에 스핀 락를 사용하여 해당 구간을 보호하여 해결하였다.

3.1.3 스몰 존 ZNS SSD를 위한 가비지 컬렉션 알고리즘

기존 ZNSwap의 가비지 컬렉션: ZnGC

ZnGC는 비어있는 존의 수가 미리 설정한 값보다 적을 때 트리거된다. 블록 SSD와 달리 ZnGC에 의해 이동된 페이지에는 호스트에서 보일 새로운 LBA가 할당된다. ZNS SSD의 경우 FTL이 없음에 따라 원래의 페이지 스왑 슬롯을 가지고 있는 페이지 테이블을 새로 쓰여진 블록 위치를 반영하도록 업데이트해야 한다. ZnGC는 이러한 페이지 테이블 LBA 업데이트를 지원하기 위해 스왑 아웃 페이지를 쓰면서 ZNS SSD 특정 LBA 포맷의 LBA별 메타데이터 영역에 역매핑 메타데이터를 저장한다. ZnGC는 역매핑 메타데이터를 스캔하여 주어진 페이지의 역방향 조회(reverse lookup)를 수행한 다음 해당 페이지 테이블을 업데이트한다.

먼저 ZnGC는 GC를 수행할 존의 스왑 슬롯을 확인한다. 현재 스왑 캐시에 캐시된 페이지의 스왑 슬롯은 스왑 캐시에서 제거되고 해당 스왑 슬롯은 무효화된다. 유효한 스왑 슬롯은 디바이스 GC 읽기를 수행하기 위해 미리 할당된 블록 IO(struct bio) 배열이 가득 차거나 GC를 수행하고자 하는 존 끝에 도달할 때까지 배열에 모은다. 모아진 블록 IO는 디바이스에 대한 읽기 요청으로써 배치 방식으로 전달된다. 각 읽기 요청은 미리 할당된 페이지 풀(page pool)에 읽은 페이지 데이터

를 저장한다. 각 스왑 슬롯의 역매핑 메타데이터는 디바이스에서 버퍼로 읽혀진다. 모든 읽기 작업이 완료되면 페이지 풀의 각 페이지가 유효한지 검사하고 ZnGC에 의해 새로 쓸 존을 할당한다. 블록 IO 배열은 GC 중에 계속 재사용되어 배치로 전달되는 보류 중인 모든 쓰기 요청을 보관한다. 쓰기 작업이 완료된 후 원래 존의 해당 스왑 슬롯의 유효성이 다시 검사된다. 스왑 슬롯이 여전히 유효하고 사용 중인 경우, 이러한 스왑 슬롯이 현재 사용 중임을 다른 커널 프로세스에 나타내기 위해 원래 존과 새로 쓸 존 모두에서 스왑 캐시에 상주하는 것처럼 표시된다. 원래 스왑 슬롯에 해당하는 페이지 테이블 엔트리는 역매핑 메타데이터를 통해 새로 쓸 스왑 슬롯을 유지하도록 다시 매핑된다. 마지막으로 원래 스왑 슬롯을 지운다. 한 개의 존에 대한 ZnGC 순회가 끝나면 해당 존을 존-리셋 명령어를 통해 비어있는 존으로 만든다.

스몰 존 ZNS SSD를 위한 가비지 컬렉션

그림 3.5는 본 논문에서 제안하는 스몰 존 ZNS SSD를 위한 수정된 ZnGC이다. 기존 ZnGC와 마찬가지로 GC를 수행할 존의 스왑 슬롯의 유효성을 검사하며 읽을 페이지를 블록 IO 배열에 채운다. 이때 블록의 요소인 블록 IO는 Chunk 크기가 될 때까지 계속해서 페이지를 추가하며 유지되도록 변경하였다. GC 수행할 존의

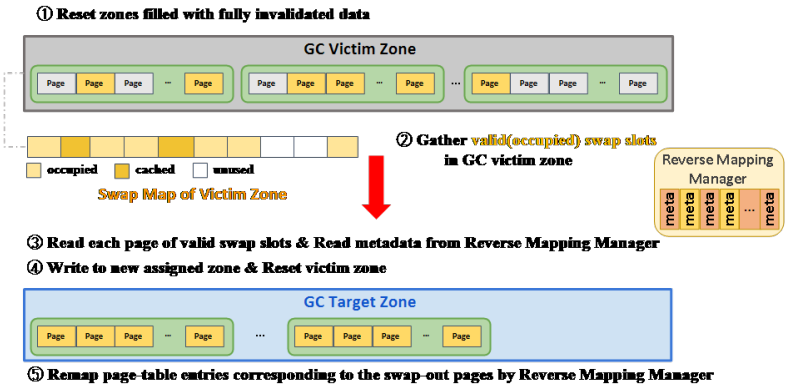

그림 3.5 수정된 ZnGC

유효한 스왑 슬롯들의 페이지를 읽고, 각 페이지에 해당하는 역매핑 메타데이터를 Reverse Mapping Manager로부터 읽어온다. 이후 기존에 GC를 수행하고 빈 공간이 남은 존을 재활용하거나, 직전에 GC 이후 회수된 존에 유효한 페이지들을 Chunk 단위로 모아서 쓰기를 수행한다. 회수하고자 하였던 존의 유효한 모든 스왑 슬롯을 쓰기 완료하였다면 존-추가 명령어를 통해 반환된 쓰여진 LBA를 각각 해당하는 페이지의 역매핑 메타데이터를 활용하여 페이지 테이블을 찾아 업데이트한다. 이후, GC를 수행한 존을 존-리셋 명령어로 재설정하여 최종적으로 공간을 회수한다.

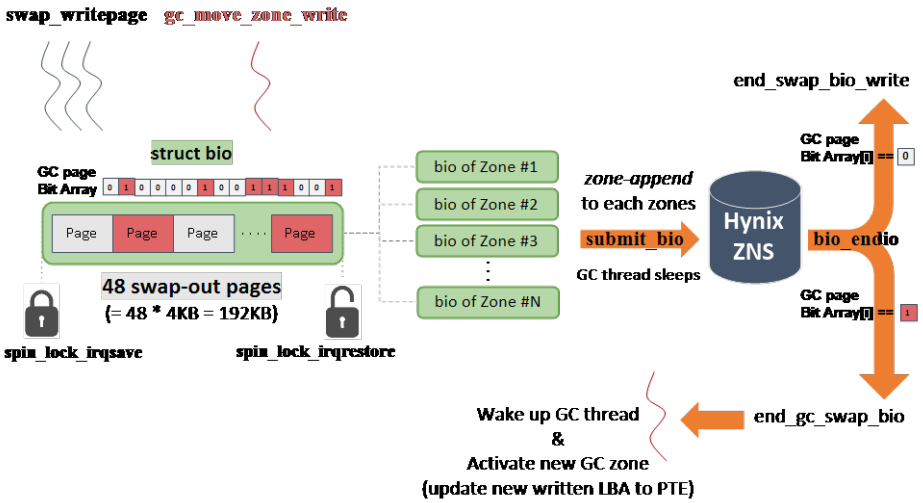

그림 3.6 수정된 새 블록 IO 구성

수정된 블록 IO 구성 방식

그림 3.6은 수정된 ZnGC에서 Chunk Aggregator를 위해 수정된 블록 IO를 설명한다. 기존 ZNSwap은 GC 스레드와 스왑아웃 스레드들이 서로 다른 bio를 사용하고 있다. 그러나 본 연구에서 Chunk Aggregator를 적용함에 따라 기존 방식을 그대로 사용하기에는 bio에 일정량의 페이지를 모으는 동안에 서로 다른 여러 스레드에서 사용한 존 크기 관리 문제가 있다.

이를 해결하기 위해, GC 스레드와 스왑 아웃 스레드에 대해 bio를 분리하지 않고, 쓰기 IO에 대한 bio는 함께 사용하도록 하여 항상 특정 Chunk 크기가 되거나 일정 시간이 되면 flush될 수 있도록 수정하였다. 이때 GC 스레드의 유효한

페이지와 스왑 아웃 스레드의 스왑 아웃 페이지를 구분하는 비트 어레이를 두어 블록 IO 제출 이후 endio 콜백 함수를 분리하여 GC 스레드와 스왑 아웃 스레드의 페이지 처리를 분리하였다.

현재 GC 스레드는 단일 스레드로 동작하며 초기 상태는 IDLE이다. 먼저 오픈 존들 중 최대 무효 카운터(max_invalid_counter)가 가장 큰 존을 Victim Zone으로 선정하고, 해당 존의 유효한 스왑 페이지를 읽는다. 이후 GC_READ로 상태를 변경하고 읽혀진 페이지들에 대해 다시 페이지 유효 검사를 수행하고, v유효한 페이지들만을 쓰기 IO를 위한 공유 bio에 추가한다. Chunk 크기만큼 bio가 채워지면 쓰기 bio를 제출하고 상태를 GC_WRITE로 변경한다. 이후, bio 내 각 페이지에 대해 비트 어레이를 검사하여 GC 스레드에 의한 페이지일때 해당되는 페이지 프레임에 대해 새로운 LBA를 쓴다. 모든 과정이 끝나면 GC_ACTIVATE로 상태가 바뀌며, Victim Zone의 모든 유효한 페이지에 대해 GC가 끝날때까지 과정을 반복한다.

제 4 장 실 험

4.1 실험 환경

본 연구는 Intel i7-12700k 3.6Ghz 8 코어 머신에서 48GB의 RAM, SK Hynix ZNS SSD 32TB를 사용하였다. 운영체제 버전은 ubuntu 22.04이며, ZNSwap과 본 연구에서 개발한 커널을 사용하였다. 스왑 디바이스로 사용한 ZNS SSD는 존 크기가 100MB 미만의 스몰 존 ZNS SSD이며, 스왑에 사용하는 존의 개수는 5480개로 기존 ZNSwap의 실험 결과와 비교하기 위해 같은 크기를 갖도록 개수를 조정하였다.

평가를 위한 벤치마크로는 vm-scalability [12]를 사용하였다. vm-scalability는 표준 스왑 성능 측정 벤치마크로, 다양한 메모리 접근 패턴에 따른 스왑 서브시스템의 성능을 평가할 수 있다. cgroup을 이용하여 메모리의 양을 2GB로 제한하여 실험을 수행하였다. 스왑 디바이스의 활용도에 따른 평가를 하기 위해 각 활용도에 맞게 메모리에 일정량을 미리 할당하고 실험을 수행하도록 하였다. vm-scalability의 case-anon-w-rand-mt 워크로드를 사용하여 메모리에 임의 쓰기

를 수행하도록 하였고, 이를 통해 스왑 디바이스에 반복적인 임의 읽기와 쓰기가 일어나도록 하였다. 최대한의 성능을 도출하기 위해 보유한 머신 코어의 2배에 해당하는 16 스레드로 실험을 수행하였다. 매 실험을 수행할 때마다 ZNS SSD는 존-리셋 명령어로 모든 존이 비워진 상태이다.

실험 결과에서 사용되는 디바이스 활용도(Device Utilization)의 정의는 다음과 같다. 디바이스 활용도란 스왑 존 크기에 대해 워크로드가 활용하는 스왑 영역의 크기 비율을 의미한다. 즉, 워크로드의 크기와 의미 상 동일하다. 예를 들어, cgroup으로 메모리를 2GB로 한정하였고, 스왑 존 크기가 576GB 라고 하면 워크로드가 60GB를 쓴다면 메모리 2기가를 제외한 58GB에 대해 디바이스 활용도는 10%이다. 워크로드가 118GB를 쓴다면 메모리 2GB를 제외한 116GB에 대해 디바이스 활용도는 20%이다.

이때 실험에 사용된 워크로드는 디바이스 활용도가 N%라면 해당 스왑 디바이스 크기의 N%만큼을 먼저 스왑 아웃만을 진행한다. 즉, 디바이스 활용도만큼 스왑 아웃으로 채워둔 뒤 스왑 인과 아웃을 반복하는 워크로드이다. 스왑 인을 하였을 때 스왑 캐시에 저장되어 무효화되므로, 디바이스 활용도가 커지면 실험 수행 초반에 각 존의 무효화된 페이지 비율이 극히 적어질 가능성이 있다.

4.2 스왑 서브시스템의 성능 측정

4.2.1 기존 ZNSwap와 제안된 수정 ZNSwap의 스왑 아웃 대역폭 비교

ZNSwap에서 제시된 실험 결과와 본 논문에서 제안한 수정 ZNSwap(ZNSwap-modified)의 스왑 아웃 대역폭을 비교한다. 이때, ZNSwap은 약 1GB의 존 크기를 가지는 라지 존 ZNS SSD에서 특정 LBA 포맷에서 제공되는 LBA 당 64바이트씩의 메타데이터 영역을 활용하여 역매핑 메타데이터를 저장한다. 본 논문에서는 스몰 존 ZNS SSD을 사용하였으며, ZNSwap에 필요한 특정 LBA 포맷을 지원하지 않음에 따라 ZNS SSD의 최소 쓰기 단위에 맞게 Chunk로 모아서 입출력을 할 때 역매핑 메타데이터를 각 Chunk의 가장 마지막에 추가하여 GC에 사용할 수 있도록 하였다.

인 메모리 역매핑 메타데이터에서의 스왑 아웃 대역폭 비교

본 논문에서 제안한 수정 ZNSwap은 인 메모리에 역매핑 메타데이터를 관리하는 최소한의 배열을 두었을 때의 성능을 먼저 비교한다. 이때, 약 530GB의 스왑 디바이스 크기에 대해 인 메모리에 관리되는 역매핑 메타데이터 배열은 (96MB / 4KB * 24B) * 5480개 존으로, 약 3GB이다. 기존 ZNSwap에서 최대 수 MB만을

메모리에 추가로 사용한다고 주장한 것과 달리 실제 ZNSwap을 스왑 온 하였을 때 약 1.1GB이상의 메모리를 소모함을 확인하였다.

스몰 존 ZNS SSD는 약 100MB의 작은 존 크기를 갖기 때문에 라지 존 ZNS SSD에 비해 상대적으로 존-리셋에 걸리는 시간 오버헤드가 적으며, 존 하나가 모두 무효한 데이터로 채워질 가능성이 높다. 현재 ZnGC와 수정 ZnGC 모두 GC가 트리거가 되더라도 해당 존이 이미 모두 무효화된 데이터로 채워진 상태라면 존-리셋 명령어를 통해 존을 회수하여 공간을 확보한다. 따라서, 반복적인 임의 쓰기와 읽기로 인해 스왑 인과 스왑 아웃이 빈번한 워크로드에서 수정된 ZNSwap은 스몰 존 ZNS SSD에서 평균적으로 안정된 스왑 아웃 성능을 보인다.

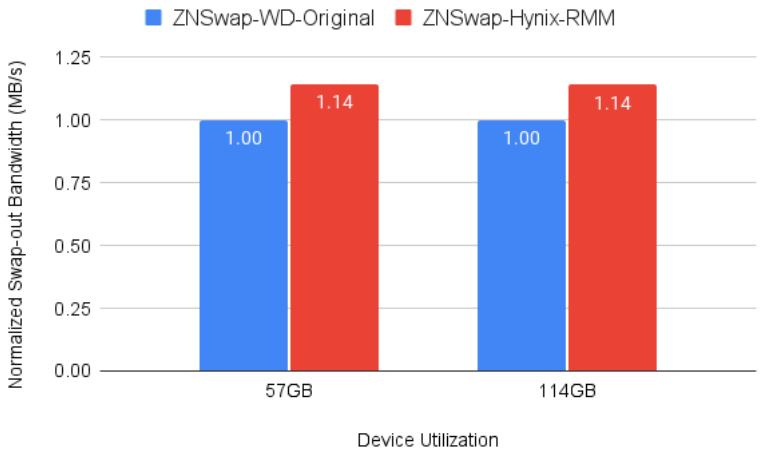

그림 4.1 ZNSwap과 비교한 평균 스왑 아웃 대역폭

그림 4.1은 기존 ZNSwap과 인 메모리에 역매핑 메타데이터를 관리하고 스몰 존 ZNS SSD를 활용하도록 수정한 ZNSwap-modified의 평균 스왑 아웃 대역폭을 기존 ZNSwap의 성능에 정규화하여 비교한 그래프이다. 스몰 존 ZNS SSD의 특성상 존 하나의 성능이 라지 존 ZNS SSD보다 떨어지지만, 멀티 스레딩을 활용한 스왑 아웃으로 GC가 동작하는 동안에도 이미 다른 오픈 존들이 많아서 스왑 아웃 스레드들이 여러 존에 동작할 수 있다. 상대적으로 오픈 존의 개수가 극히 적어서 과도하게 잔여 존이 없는 상황에서 GC와 스왑 아웃이 같은 존을 공유할 때, 스왑 아웃 스레드에 존을 할당하지 못해 잦은 지연(suspend)이 일어나는 라지 존 ZNS SSD에 비해 GC 동작 중 스왑 아웃 성능에서 이점이 있다. 따라서 논문 [1]의 ZNSwap 대비 수정된 ZNSwap이 비슷한 스왑 아웃 성능을 보임을 확인할 수 있다.

제 5 장 관 련 연 구

5.1 메모리 스왑 디바이스로 ZNS SSD 활용

ZNS SSD를 메모리 스왑 디바이스로 활용하도록 제안한 최신 논문인 ZNSwap은 [1] 스왑 영역의 효과적인 호스트 가비지 컬렉션인 ZnGC를 제안한다. 디바이스 내부 GC와 비교할 때 ZNGC은 운영체제가 가지는 정보에 접근하여 스왑 입출력 성능을 최적화할 수 있다. 예를 들어 ZNGC는 운영체제에서 유지하는 스왑 슬롯 배열을 활용하여 운영체제에서 무효화된 스왑 슬롯을 식별하여 중복되는 페이지 복사를 피할 수 있다. znGC는 스왑 아웃된 페이지가 여전히 해당 페이지 테이블에 유지된다는 것을 이용한다. GC 진행에 있어 필요한 논리-물리 페이지 블록 주소에 대한 추가적인 계층을 두지 않고 각 페이지를 스왑 아웃하면서 역방향 매핑 메타데이터를 함께 저장한다. 이때 특정 NVMe SSD 컨트롤러의 LBA 포맷에서 제공하는 LBA당 메타데이터 영역을 사용하여 역방향 매핑을 관리하기 위한 I/O 오버헤드를 줄이고자 하였다. 그러나 이러한 메타데이터 영역은 일부 기업형 SSD에서만 제공되는 LBA 포맷으로 일반 SSD에서는 제공되지 않는다는 특징이 있다.

5.2 호스트 측 SSD 관리

같은 블록 위치에 덮어쓰기가 불가능하고, 공간 회수 시 쓰기와 지우기 단위가 각각 페이지와 블록으로 다른 플래시 특성 때문에 기존 블록 SSD에서는 제자리 덮어쓰기(in-place update)가 불가능하고 다른 위치에 써야 한다(out-of-place update). 이러한 이유로 SSD 내부 DRAM에서 블록 매핑 관리가 필요하고 더이상 사용하지 않는 블록에 대한 GC를 위해 오버 프로비저닝 비용(Over Provisioning(OP) Cost)가 발생한다. 이러한 블록 SSD 사용의 비용 오버헤드를 줄이기 위해 여러 새로운 스펙의 SSD가 제안되었다.

5.2.1 Open-Channel SSD(OCSSD)

오픈 채널 SSD는 채널의 정보를 호스트에게 제공하여 SSD 내부에서의 FTL 역할을 축소한다. 이를 통해 DRAM 오버헤드와 OP 비용은 줄일 수 있지만 호스트 소프트웨어가 디바이스의 안정성(reliability)를 책임져야 한다는 단점이 존재한다. 논문 [5]에서는 오픈 채널 SSD를 위해 필요한 호스트 레벨의 소프트웨어를 제공하는 리눅스 서브 시스템을 제안한다. 호스트를 위해 페이지 블록의 매핑과 GC와 같은 FTL 기능을 제공하고, 주소 공간 인코딩을 지원한다. 이를 통해 예측 가능한

IO 성능을 보장하고자 하였지만 오류 조절 매커니즘(error handling mechanism)과 같은 문제가 존재한다.

5.2.2 MultiStreamed SSD

멀티스트림 SSD [13]는 호스트와 디바이스 간 의미 격차(semantic gap)을 줄이기 위한 인터페이스 매커니즘이다. 호스트는 SSD의 스트림을 열어서 디바이스에 쓰여질 데이터의 예상되는 수명(lifetime)에 따라 서로 다른 스트림에 쓰기 요청을 보낼 수 있다. 스트림의 데이터가 연관된 낸드 플래시 블록에 함께 쓰이지 않는 것을 보장할 뿐만 아니라 다른 스트림 데이터와도 분리됨을 보장한다. 이를 통해 GC에서의 낸드 용량 단편화(NAND capacity fragmentation)을 줄일 수 있고 데이터 이동 오버헤드가 감소하긴 하지만 DRAM과 OP 비용이 여전히 발생한다.

제 6 장 결 론

6.1 결과 분석

본 연구에서는 기존 ZNSwap을 확장하여 특수 LBA 포맷이 아닌 일반 LBA 포맷에서 다양한 최소 쓰기 단위를 갖는 스몰 존 ZNS SSD에서 동등한 스왑 성능을 가질 수 있도록 수정하였다. 실험 결과, 기존 ZNSwap 논문에서 제시한 실험 결과와 동등한 페이지 스왑 아웃 대역폭을 확인하였다. 본 연구에서 제안한 수정 ZNSwap으로 보다 일반적인 ZNS SSD의 고유한 특성을 활용하여 동등한 스왑 성능을 보장할 수 있다.

6.2 연구의 한계

본 연구에서의 한계점은 다음과 같다. 현재 ZnGC와 수정된 ZnGC 모두 스왑 아웃이 멀티 스레드로 동작하는 동안 GC가 싱글 스레드로 한 번에 한 개의 존에 대해서만 반복문을 통해 GC 상태를 바꿔가며 GC를 수행하고 있다. 이런 상태에서는 본 연구에 사용된 디바이스의 존 크기가 작고 스왑 존 개수는 많기 때문에

상대적으로 존을 한 번에 여러 개 회수하기 어렵다. 즉, 보다 빠른 공간 회수 정책을 가지는 GC가 필요하다.

특히 본 연구에서 수행한 실험 워크로드의 경우, 디바이스 활용도만큼 먼저 스왑 아웃하는 과정이 있기 때문에 실험 수행 초반에는 디바이스 활용도가 커질수록 각 존의 최대 무효 카운터 수가 극도로 적다. 즉 GC에 있어 WAF가 초반에 급격히 높음에 따라 현재의 단일 스레드이자 한 번에 한 개 Victim Zone에 대해서만 수행하는 GC 속도는 오랜 시간 실험을 수행하기 어렵다.

6.3 향후 연구

기존 ZnGC가 한 번에 하나의 Victim Zone을 valid 페이지들을 모아 적는 일종의 컴팩션과 같은 양상으로 이뤄지기 때문에 빠른 공간 회수가 어렵다는 한계를 Victim Zone을 여러 개로 두고 여러 개의 존에 걸쳐 유효 페이지를 쓰도록 변경하여 최대한 빈 존을 빠르게 확보하는 방향으로 GC 방식을 변경할 것이다. 이 때 GC를 위한 오버 프로비저닝(Over Provisioning, OP) 존을 두어 스왑 아웃을 위한 빈 존이 없더라도 이러한 OP 존을 활용할 수 있도록 변경할 수 있다.

또, 스몰 존 ZNS SSD에서는 존 간 병렬성을 활용하는 것이 중요하다고 알려져

있다. 현재 ZNSwap의 기본 존 할당 정책에서는 단순히 해당 스레드의 프로세스 번호를 기준으로 존을 할당하기 때문에 코어 수보다 더 많은 수의 최대 오픈 존 개수를 가지는 스몰 존 ZNS SSD의 경우 존 할당에 있어서 존 간 병렬성을 극대화 할 수 있다면 스왑 아웃 대역폭이 더욱 향상할 것으로 기대한다. 향후 연구에서는 이를 토대로 스왑 성능의 최적화를 진행할 계획이다.

참고문헌

[1] S. Bergman, N. Cassel, M. Bjørling, and M. Silberstein, "Znswap: Un-block your swap," *ACM Transactions on Storage*, vol. 19, no. 2, pp. 1–25, 2023.

[2] T. Song, G. Lee, and Y. Kim, "Enhanced flash swap: Using nand flash as a swap device with lifetime control," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–5, 2019.

[3] J. Lee, S. Park, M. Ryu, and S. Kang, "Performance evaluation of the ssd-based swap system for big data processing," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 673–680, IEEE, 2014.

[4] H. H. A. R. D. L. M. G. R. G. M. Bjørling, A. Aghayev and G. Amvrosiadis, "Zns: Avoiding the block interface tax for flash-based ssds," in *In Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC'21).*, 2021.

[5] M. Bjørling, J. Gonzalez, and P. Bonnet, "Lightnvm: The linux open-channel ssd subsystem," in *USENIX Conference on File and Storage Technologies*, pp. 359–374, USENIX-The Advanced Computing Systems Association, 2017.

[6] https://wiki.ubuntuusers.de/SSD/TRIM/TRIM-der Swap-Partition, *Ubuntu: TRIM the swap partition*, 2021.

[7] https://man7.org/linux/man pages/man8/swapon.8.html, *swapon(8) Linux man pages*, 2021.

[8] K. Han, H. Gwak, D. Shin, and J. Hwang, "Zns+: Advanced zoned namespace interface for supporting in-storage zone compaction.," in *OSDI*, pp. 147–162, 2021.

[9] J. Zhang, Y. Lu, J. Shu, and X. Qin, "Flashkv: Accelerating kv performance with open-channel ssds," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–19, 2017.

[10] M. Im, K. Kang, and H. Yeom, "Accelerating rocksdb for small-zone zns

ssds by parallel i/o mechanism," in *Proceedings of the 23rd International Middleware Conference Industrial Track*, pp. 15–21, 2022.

[11] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2fs: A new file system for flash storage.," in *FAST*, vol. 15, pp. 273–286, 2015.

[12] https://git.kernel.org/pub/scm/linux/kernel/git/wfg/vm scalability.git, *vm-scalability*, 2021.

[13] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive," in *6th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

Abstract

With the advancement of storage devices, it is now possible to use fast block-based solid-state drives (SSDs) as memory swap devices. However, as the utilization of block SSDs increases, a significant problem arises, which is the degradation of system performance. This issue stems from a semantic gap between the device and the operating system, where the operating system may consider swap slots invalidated while the SSD recognizes them as valid, or the swap cache used to avoid swap-out penalties does not take into account the internal garbage collection (GC) of the block device. Recently proposed, the Zoned Namespace (ZNS) is a storage interface that maps the logical address space into fixed-sized zones and presents it to the operating system. In particular, ZNS SSDs support writes by adding data to zones since NAND flash does not support overwrites. By utilizing host-managed ZNS SSDs, it is possible to alleviate the performance degradation caused by the information gap between

the operating system and the device when using traditional block-based SSDs. In the state-of-the-art research, ZNSwap [1], which applies ZNS SSDs to the memory swap subsystem, explicit garbage collection of ZNS SSDs is leveraged when performing swap operations in the operating system. To achieve this, a garbage collector in the operating system is employed to recognize the valid swap areas of the device, providing space-efficient host garbage collection. By utilizing host-managed ZNS SSDs, the performance degradation caused by the information gap between the operating system and the internal space management of the device when using traditional block-based SSDs can be mitigated through the shared swap page information between the operating system and the swap device's internal space management. However, ZNSwap has limitations, as it operates only with specific Logical Block Address (LBA) formats and experiments were conducted only on ZNS SSDs with large zone sizes. In this study, we enhance ZNSwap to ensure equivalent swap-out performance for various zone sizes in a general LBA format. Experimental results show the same maximum swap-out bandwidth compared to the ZNSwap.

Keywords: ZNS SSD, Operating System, Memory Swap Subsystem, Garbage Collection

Student Number: 2021-28075