Pusan National University
Computer Science and Engineering
Technical Report 2023-10

ZNS 를 이용한 키-밸류 스토어 성능 개선 연구

저자 1 (201724485 배재홍)
저자 2 (201224507 이재석)
저자 3 (201724588 조준호)

지도교수 안 성 용

목 차

1. 서론

최근 키-밸류 스토어(Key-Value Stores)는 고성능, 유연성 등과 같은 여러 이점이 있어 새로운 기술들에 많이 사용되고 있다. 특히 NoSQL(Not Only SQL) 데이터베이스(Database) 기술로 비정형(Unstructured)의 빅 데이터(Big Data)를 저장하는데 적합하다. 대표적인 키-밸류 스토어인 RocksDB나 LevelDB는 LSM-Tree(Log Structured Merge-Tree) 구조로 플래시(Flash) 기반 저장 장치인 SSD(Solid State Drive)에서 높은 입/출력 성능을 발휘한다. LSM-Tree는 순차 쓰기(Sequential Write)를 수행하여 데이터를 write할 때 높은 성능을 보인다. 그리고 데이터를 SSTable(Sorted String Table) 단위로 저장한다. 이는 키를 기준으로 정렬된 구조로 탐색(Search)이 빨라 read 성능을 향상시켜준다.

RocksDB는 내장 파일시스템(FileSystem)인 ZenFS를 통해 ZNS SSD(Zoned Namespace SSD)를 지원하여 성능을 향상시키고자 한다. ZenFS는 RocksDB내 파일(File)의 hot/cold 정도를 알 수 있는 Lifetime과 ZNS SSD의 Zone이라는 논리적(Logical) 단위를 이용한다. Lifetime이 비슷한 파일을 같은 Zone에 모아 Garbage Collection(이하 GC)을 줄이고 GC발생 시 쓰기 증폭(Write Amplification)을 최소화한다. 이를 통해 입/출력에 직접적인 영향을 미치는 GC를 줄임으로써 성능을 향상시킨다. 더 나아가 GC시 쓰기 증폭을 줄여 SSD 수명을 개선한다.

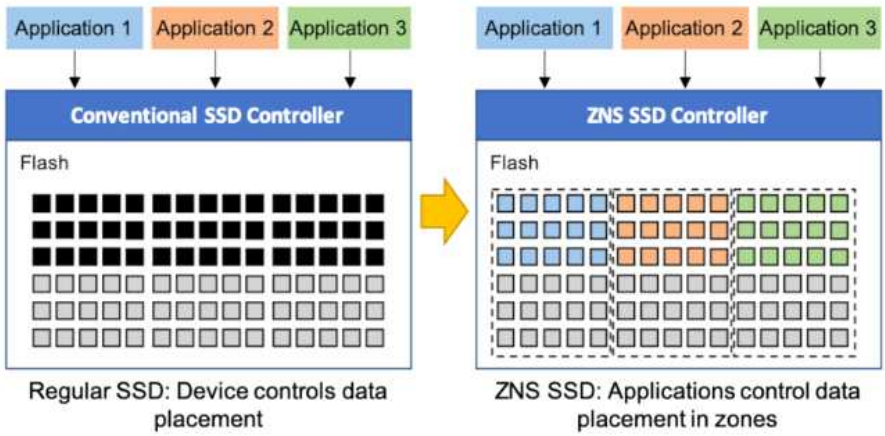
하지만 현 RocksDB는 ZNS SSD를 사용했을 때 전체 용량의 약 75.8%만 사용해도 새로운 Zone을 할당할 수 없다는 오류가 발생하는 것을 실험을 통해 발견했다. 추가적으로 로그(Log) 및 코드(Code) 분석을 통해 현재 ZenFS가 데이터를 write할 때 어떠한 기준으로 Zone을 할당하는지 확인했고 ZNS SSD의 여유 공간을 확보하기 위한 Zone Erase 정책도 존재함을 확인했다. 그리고 해당 Zone 할당 방식은 ZNS SSD에 많은 무효(Invalid) 데이터를 남겨 GC를 많이 발생시키고 Zone Erase 정책을 의도대로 사용하지 못한다고 판단했다.

위와 같은 문제점을 바탕으로 본 연구에서는 ZenFS의 Zone Erase 정책을 잘 활용할 수 있는 개선된 Zone 할당 정책 SLA(Strong Lifetime based Allocation)를 구현한다. 그리고 개선된 Zone 할당 정책에 따라 발생할 수 있는 문제를 해결한다. 해당 연구 결과로 같은 크기의 write를 발생시켰을 때 기존보다 약 11.7%만큼 Zone을 덜 사용했고 GC시 쓰기 증폭의 원인인 복사(Migration)의 크기도 약 98.3% 줄였다.

2. 배경 지식

2.1 ZNS SSD

SSD는 HDD(Hard Disk Drive)와 달리 덮어쓰기(Overwrite)가 불가능하여 데이터 수정 시 해당 데이터를 무효화시키고 새로운 곳에 write해야 하는 특성이 있다. 그리고 무효화된 영역을 다시 사용하기 위해서 erase가 필요하다. 이때 erase의 단위가 write 단위보다 더 크다. 이로 인해 erase 시 단위 내 유효(Valid) 한 데이터가 존재한다면 해당 데이터를 다른 위치에 복사한 후 erase를 수행한다. 하지만 이 복사와 erase가 프로그램-삭제 횟수(Program-Erase Cycle)가 정해져있는 SSD의 수명에 큰 영향을 미친다. 따라서 SSD는 데이터 수정 시 기존 데이터를 다른 곳으로 옮겨 새로 저장하고 기존 데이터는 무효화 시킨 후 erase 단위 내 무효 데이터가 많아질 때까지 erase를 미룬다. 그리고 무효 데이터가 많아져 SSD의 용량이 부족할 때 SSD는 자체적으로 무효화 영역을 지워 SSD의 공간을 확보한다. 이 과정을 Garbage Collection(GC)이라 하고 GC 발생 시 유효 데이터를 복사해야 하는 경우가 생긴다. 이러한 의도하지 않은 write가 발생하는 현상을 쓰기 증폭이라 한다. GC 및 쓰기 증폭은 SSD의 수명을 단축시키고 장치 입/출력에 직접적인 영향을 미쳐 성능 오버헤드(overhead)가 크다.

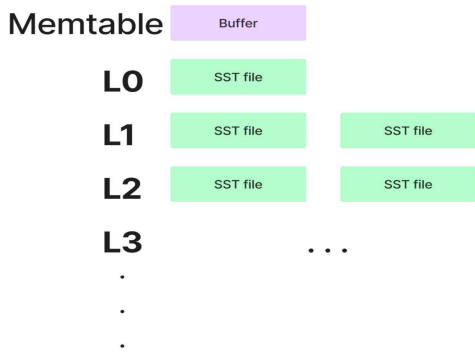


[그림-1] (좌)기존 SSD와 (우)ZNS SSD 저장방식 비교

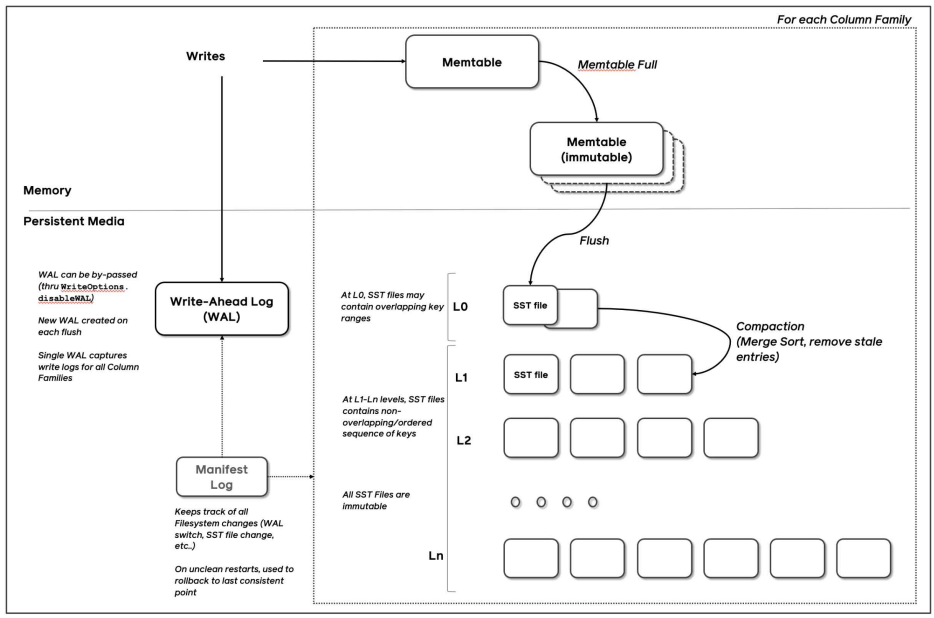
최근 SSD의 수명과 성능을 개선하기 위해 SSD를 Zone이라는 논리적 영역으로 나눈 ZNS SSD가 제안됐다. **[그림-1]**은 기존 SSD와 ZNS SSD의 저장 방식 차이를 나타낸 그림이다. 여러 애플리케이션(Application)이 데이터를 write 할 때 기존 SSD는 순서에 상관없이 플래시 메모리(Flash Memory)에 저장된다. 반면 ZNS SSD는 각 애플리케이션의 데이터가 서로 다른 Zone에 순차 쓰기 방식으로 저장된다. 이러한 방식은 각 Zone에 저장된 데이터가 비슷한 시

기에 무효화되는 것을 기대한다. 또 ZNS SSD의 erase 단위는 Zone 단위이므로 GC 시 유효 데이터 복사를 최소화해 쓰기 증폭을 최소화할 수 있다. [1]

2.2 LSM-Tree 및 RocksDB



[그림-2] LSM-Tree 구조



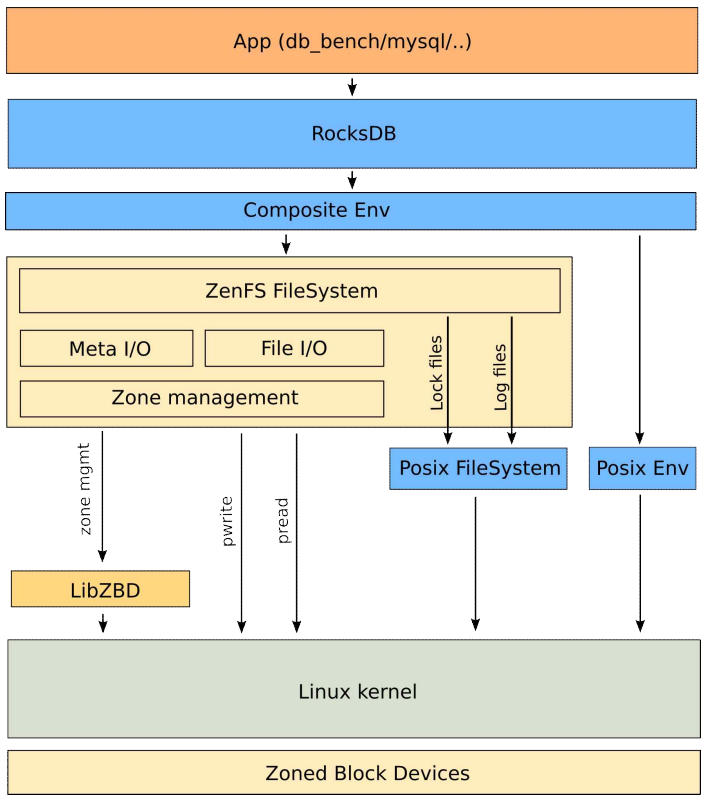
[그림-3] RocksDB 구조

[그림-3]은 RocksDB의 구조를 나타낸 그림이다. RocksDB는 Memtable, SSTfile 및 Logfile 세 가지로 구성된다. 먼저 Memtable은 in-memory data structure로 write 발생 시 데이터가 Memtable에 저장된다. 그리고 전원이 차단되는 등 메모리의 데이터가 소실될 것을 방지하여 WAL(Write-Ahead Log)

라는 Logfile을 ZNS SSD와 같은 영구 저장 장치에 플러시(Flush) 한다. 그리고 Memtable이 가득 차게 되면 영구 저장 장치에 플러시 되고 Logfile은 안전하게 삭제된다. Memtable이 플러시 되면 키-밸류(Key-Value)들로 구성된 SSTfile이 저장된다. [2, 3, 4]

그리고 RocksDB는 LSM-Tree(Log Structured Merge-Tree) 구조를 사용한다. LSM-Tree는 순차 쓰기(Sequential Write)를 수행하여 데이터를 write할 때 높은 성능을 보인다. 그리고 **[그림-2]**에서 보이는 L0, L1 등과 같은 레벨로 이루어져 있다. 각 레벨은 저장할 수 있는 데이터양이 다르고 레벨이 커질수록 저장할 수 있는 데이터양도 많아진다. 기본적으로 이전 레벨의 10배 크기만큼 데이터를 더 많이 저장할 수 있다. 어느 한 레벨이 가득 차게 되면 다음 레벨과 컴팩션(Compaction)이 수행된다. 컴팩션은 Ln과 Ln+1의 SSTfile들 중 키값이 겹치는 SSTfiile을 새로운 SSTfile을 생성해 Ln+1에 저장하는 과정이다. [4, 5]

2.3 ZenFS



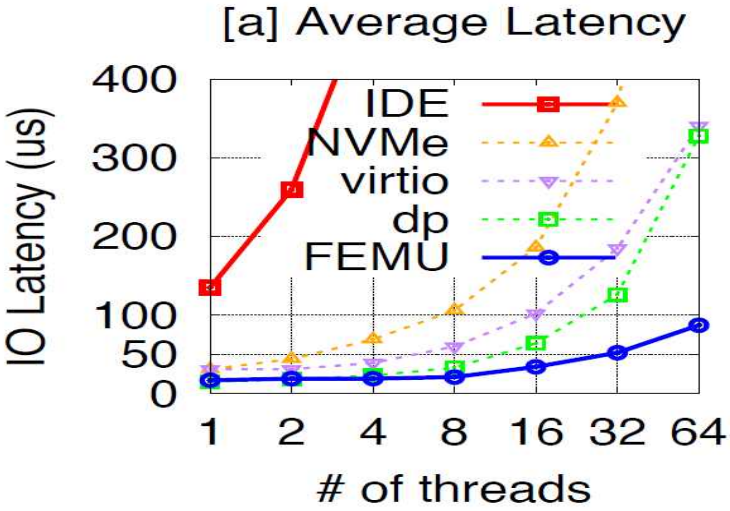
[그림-4] 전체 시스템 구성

ZenFS는 ZNS SSD에 파일을 write할 수 있게 하는 RocksDB의 내장 파일시스템이다. RocksDB의 Lifetime 정보를 바탕으로 유사한 Lifetime의 데이터를

같은 Zone에 배치하고 GC를 줄여 쓰기 증폭을 줄인다. 그리고 GC를 줄여 입/출력 성능을 높이고 ZNS SSD의 수명을 향상시킨다.

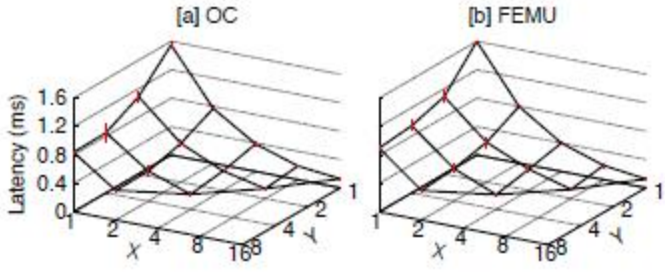
2.4 FEMU

본 과제에서는 ZNS SSD를 이용하기 위해 FEMU라는 가상 머신(Virtual Machine)을 사용했다. FEMU는 QEMU 기반 플래시(Flash) 에뮬레이터(Emulator)이다. [6, 7]



[그림-5] 에뮬레이터간 확장성 비교

[그림-5]는 여러 에뮬레이터의 확장성을 비교한 그래프이다. IO 스레드(Thread)가 늘어남에 따른 IO 대기시간을 보여주고 있다. 최신 SSD의 평균 IO 대기시간은 100μs 미만이다. FEMU는 64개의 IO 스레드가 있더라도 100μs 미만의 대기시간을 보여준다. 하지만 대부분의 에뮬레이터가 16개의 IO 스레드만 존재하더라도 대기시간이 100μs가 넘어가고 32개의 IO 스레드가 있으면 모두 대기시간이 100μs가 넘어간다. FEMU는 IO 스레드가 늘어나더라도 안정적인 IO 지연시간을 유지해 정확하게 SSD를 에뮬레이트할 수 있다. [6, 7]



[그림-6] 실제 SSD 및 FEMU IO 대기시간

[그림-6]은 실제 SSD와 FEMU의 IO 대기시간을 보여주는 그래프이다. X와 Y는 SSD 내부 설계인 plane과 channel을 나타낸 것이다. FEMU는 SSD의 내부 설계까지 에뮬레이트 가능한 것을 알 수 있고 IO 대기시간이 실제 SSD와 거의 흡사하다. 이를 통해 FEMU가 SSD를 정확히 에뮬레이트할 수 있다는 것을 알 수 있다. [6, 7]

3. 연구 동기

이번 3장에서는 현재 RocksDB를 벤치마크를 통해 테스트하고 로그 및 코드 분석을 통해 문제점을 파악한다.

3.1 현 RocksDB 벤치마크

현재 RocksDB의 성능을 알아보기 위해 벤치마크를 수행했다.

3.1.1 실험 환경

실험 환경으로 가상 머신인 FEMU를 사용했다.

CPU 코어 수	32
Memory 크기	32GB
저장소(Storage)	에뮬레이트 된 ZNS SSD
커널(Kernel) 버전(Version)	Linux Kernel 5.10.179

[표-1] 가상머신 사양

용량(Capacity)	32GB
섹터(Sector) 크기	512Bytes
Zone 크기	128MB
Zone 개수	256

[표-2] 에뮬레이트 한 ZNS SSD 사양

[표-1]은 가상머신의 사양을 작성한 표이고 **[표-2]**는 FEMU를 통해 에뮬레이트 한 ZNS SSD 사양을 작성한 표이다. RocksDB의 성능 평가 도구는 RocksDB에 내장된 db_bench 프로그램을 사용했다.

3.1.2 벤치마크(BenchMark)

키(Key) 크기	16Bytes
밸류(Value) 크기	100Bytes
SSTfile 크기	64MB
총 write 크기	약 27GB

[표-3] db_bench 옵션(Option)

[표-3]은 db_bench를 사용해 성능을 측정할 때 사용한 옵션이다. 실제 사용하는 서버 환경보다 작은 ZNS SSD의 용량을 고려해 키, 밸류, SSTfile 크기를 기본값(default)으로 설정했다. 이외 키-밸류 저장 방식은 비동기(Asynchronous) 랜덤 오더(Random Order)로 키값을 무작위 순서로 write한다. [8, 9]

3.1.3 결과

해당 벤치마크 프로그램으로 성능을 측정한 결과 약 26GB를 write한 후 더 이상 새로운 Zone을 할당할 수 없다는 에러(Error)와 함께 종료됐다. 이 결과는 현 RocksDB가 ZNS SSD를 사용할 때 전체 용량의 약 75.8%만 사용해도 모든 Zone을 사용해 더 이상 데이터를 write할 수 없음을 나타낸다.

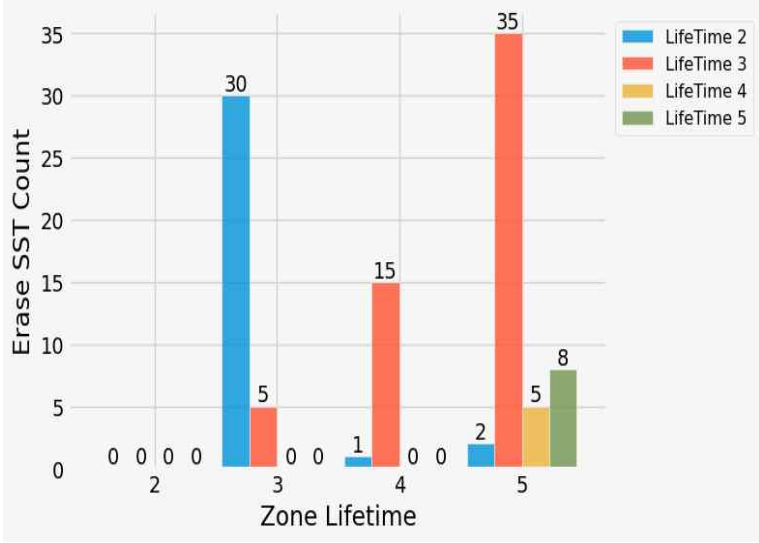
3.2 로그 분석

위의 벤치마크 수행 중 발생한 GC에 대한 정보를 로그로 출력해 분석했다. 다음 **[표-4]**는 로그를 출력 및 분석하기 위한 변수이다.

SSTfile Lifetime	RocksDB의 LSM-Tree 레벨 정보를 바탕으로 결정되는 변수 (2~5)
Zone Lifetime	현재 Zone에 저장된 SSTfile의 Lifetime 정보를 바탕으로 결정되는 변수 (2~5)

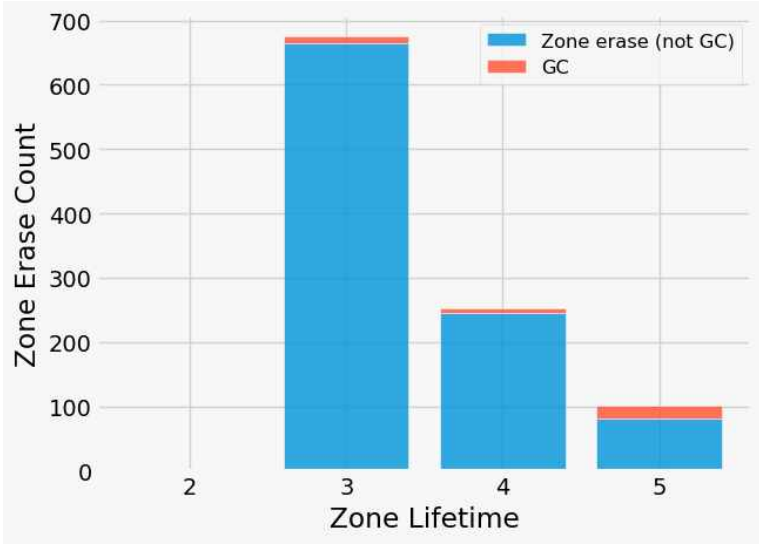
[표-4] ZenFS의 변수

위 두 변수 모두 Lifetime이 유사한 SSTfile을 같은 Zone에 모으기 위해 유지하는 변수이다. 두 변수를 이용해 GC가 발생할 때 erase 대상이 되는 Zone의 Lifetime과 해당 Zone의 무효 SSTfile의 Lifetime을 출력했다.



[그림-7] GC시 Erase 대상 Zone의 Lifetime과 저장된 무효 데이터의 Lifetime

[그림-7]을 바탕으로 Lifetime이 2인 Zone은 GC가 발생하지 않음을 알 수 있었다. 그리고 Erase 대상인 Zone의 Lifetime보다 큰 Lifetime의 SSTfile은 존재하지 않았다.



[그림-8] Zone Erase 수

로그 분석 중 Zone erase가 GC 시에만 발생하는 것이 아닌 것을 알게됐다. **[그림-8]**을 통해 GC를 통해 erase되는 Zone의 수보다 더 많은 양의 Zone이 erase 되는 것을 확인할 수 있다. 이를 통해 ZenFS에는 ZNS SSD의 여유 공간을 위해 주기적으로 Zone을 erase하는 정책이 있음을 알게됐다.

3.3 코드 분석

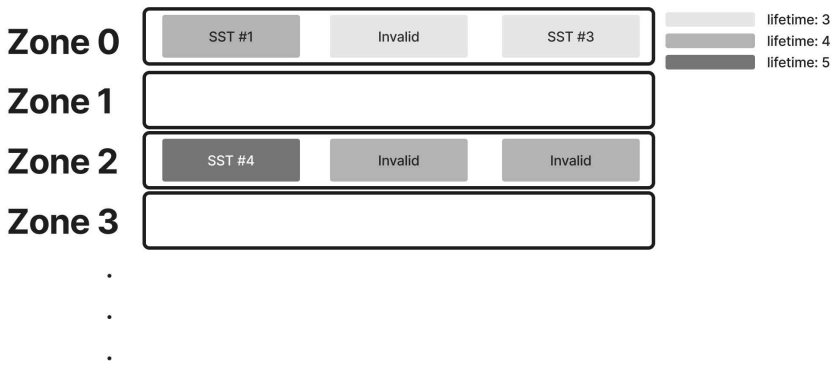
위 로그 분석 내용을 자세히 알아보기 위해 코드 레벨로 ZenFS의 동작을 분석하고 Zone 할당 정책과 Zone Erase 정책에 대해 정리했다.

3.3.1 레벨별 SSTfile의 LIfetime 할당 방식 분석

ZenFS는 데이터의 hot/cold 정도를 분석하고 Zone별로 나누어 저장하기 위해 SSTfile마다 Lifetime을 지정한다. Lifetime의 지표는 LSM-Tree의 레벨이다. 레벨이 낮을수록 새로운 데이터가 많이 write되고 컴팩션이 자주 발생해 SSTfile이 자주 삭제된다. 따라서 레벨이 높을수록 높은 Lifetime을 가진다. ZenFS에서 Lifetime은 2에서 5사이 값을 갖는다. Memtable이 플러시 되면 삭제되는 Logfile은 Lifetime이 2이고 SSTfile은 레벨에 따라 3에서 5의 값을 갖게 된다.

3.3.2 ZenFS Zone 할당 방식 분석

ZenFS는 새로운 SSTfile을 Lifetime에 따라 각 Zone에 나누어 write한다. 그리고 비슷한 Lifetime을 가진 SSTfile을 같은 Zone에 최대한 배치한다. 이때 ZenFS는 Zone 할당을 위해 Zone의 Lifetime도 함께 유지한다. Zone의 Lifetime은 Zone에 처음 write된 SSTfile의 Lifetime으로 설정되고 Zone이 erase 될 때까지 변하지 않는다. Zone Lifetime과 write될 SSTfile의 Lifetime을 바탕으로 모든 Zone을 스캔(Scan) 하여 SSTfile의 Lifetime보다 Lifetime이 크고 차이가 가장 적은 Zone이 선택된다. 즉, 한 Zone에는 처음 write된 SSTfile의 Lifetime보다 작은 Lifetime의 SSTfile이 write된다. 예외적으로 GC가 발생했을 때 유효 SSTfile을 옮길 Zone을 선택할 경우에는 유효 SSTfile보다 Lifetime이 큰 Zone이 없을 시 Lifetime이 같은 Zone에 write 되기도 한다. [3, 10]



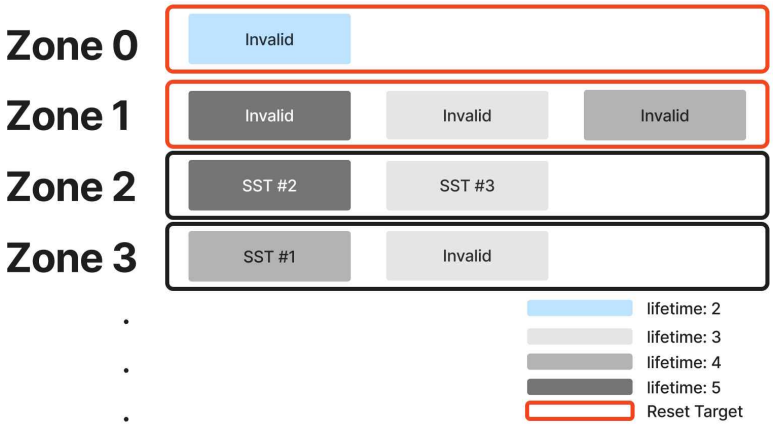
[그림-9] 현 ZenFS의 Zone 할당 방식

현 ZenFS의 Zone 할당 방식을 통해 GC 발생 시 각 Zone에 복사할 데이터를 최소화한다. 예를 들어 **[그림-9]**의 Zone 2와 같이 가장 Lifetime이 높은

SSTfile만 유효하고 Lifetime이 낮은 SSTfile은 모두 무효하다면 GC 시 한 개의 SSTfile에 대해서만 복사가 발생한다.

3.3.3 ZenFS의 Zone erase 정책 분석

SSD는 write 전 erase를 해야 하는 특성 때문에 데이터를 삭제해도 해당 데이터가 존재했던 위치에 바로 write할 수 없다. 데이터가 삭제되면 무효 상태로 남게 되는데 이 무효화된 데이터를 erase해서 공간을 확보해야 한다.



[그림-10] 현 ZenFS의 Zone erase 정책

ZenFS에는 주기적으로 ZNS SSD의 공간을 확보할 수 있는 Zone erase 정책이 있다. 해당 정책은 SSTfile이 삭제될 때마다 수행된다. SSTfile이 삭제되면 모든 Zone을 스캔해 무효한 데이터만 가진 Zone을 erase한다. 이는 Zone이 가득 차있지 않더라도 erase한다. **[그림-10]**은 ZenFS의 Zone erase 정책을 그림으로 나타낸 것이다. 무효 데이터만 존재하는 Zone 0과 1에 대해서 erase가 발생한다. 해당 Zone erase 정책을 통해 무효 데이터를 제거해 공간을 확보하고 GC를 늦출 수 있다.

3.4 분석 내용 종합

ZenFS는 RocksDB의 LSM-Tree가 낮은 레벨에서 데이터 변화가 많은 것을 바탕으로 데이터의 Lifetime을 정한다. 그리고 데이터의 Lifetime을 바탕으로 비슷한 Lifetime을 가진 데이터를 같은 Zone에 모은다. ZenFS는 해당 Zone 할당 방식을 통해 각 Zone의 비슷한 Lifetime을 가진 데이터가 모두 무효화되어 Zone erase 정책에 따라 공간을 확보하고 GC 발생을 늦추는 것을 기대한다. 또, Zone의 모든 데이터가 무효화되지 않더라도 가장 높은 Lifetime을 가진 SSTfile 한 개만 남아 GC 시 쓰기 증폭을 최소화하는 것을 기대한다.

하지만 기대와는 달리 현 ZenFS는 Zone에 많은 무효 데이터를 남겨 GC가 많이 발생한다. 그리고 GC가 발생되지 않는 이상 처음 Zone에 배치된 SSTfile

의 Lifetime과 같은 Lifetime의 SSTfile을 함께 배치시키지 않아 사용되는 Zone이 많아진다. write되는 데이터가 많을수록 높은 Lifetime 5를 가진 SSTfile이 많아지고 해당 SSTfile은 잘 삭제되지 않는다. 이로 인해 Lifetime이 5인 Zone은 한 개의 SSTfile로 인해 다른 데이터가 모두 무효화되더라도 erase되지 않게 된다. 그리고 Lifetime이 5인 SSTfile은 GC가 발생하지 않으면 항상 새로운 Zone을 할당해야한다. 이러한 이유로 ZNS SSD의 여유 공간 확보가 잘되지 않아 GC가 더 빨리 발생하고 입/출력 성능이 하락한다. 그리고 **3.1절**의 실험 결과와 같이 할당할 Zone이 부족해진다.

4. 연구 내용

4.1 개선된 Zone 할당 정책

4.1.1 Weakly/Strong Lifetime based Allocation: WLA/SLA

본 과제에서는 두 가지 Zone 할당 방식을 구현 및 평가했다.

첫 번째 방식은 Lifetime이 같은 SSTfile을 우선적으로 같은 Zone에 모으고 해당 Lifetime과 같은 Zone이 없을 시 현 ZenFS의 정책을 따른다. 즉, SSTfile이 write되면 모든 Zone을 스캔해 SSTfile의 Lifetime과 Lifetime이 같은 Zone을 할당하여 저장하고 그러한 Zone이 없을 시 SSTfile의 Lifetime보다 Lifetime이 크고 가까운 Zone에 저장한다. 해당 방식을 WLA(Weakly Lifetime based Allocation)라 부르겠다.

두 번째 방식은 Lifetime이 같은 SSTfile만 같은 Zone에 모으는 방식이다. 마찬가지로 SSTfile이 write되면 모든 Zone을 스캔해 Lifetime이 같은 Zone에 저장하고 그러한 Zone이 없을시 새로운 빈 Zone을 할당한다. 해당 방식을 SLA(Strong Lifetime based Allocation)라 부르겠다.

[그림-11] (좌) 기존 Zone 할당 정책 (우) WLA/SLA

	우선	차선
ZenFS	Zone lifetime > SST lifetime	New Zone Allocate
WLA	Zone lifetime = SST lifetime	Zone lifetime > SST lifetime
SLA	Zone lifetime = SST lifetime	New Zone Allocate

[표-5] Zone 할당 방식 비교

[그림-11]은 기존의 Zone 할당 정책과 WLA/SLA를 비교한 그림이고 **[표-5]**는 Zone 할당 방식을 비교한 표이다. WLA/SLA는 같은 Lifetime의 SSTfile을 같은 Zone에 우선적으로 모은다. WLA는 차선인 Zone마저 없을 시 새로운 Zone을 할당한다.

4.1.2 벤치마크

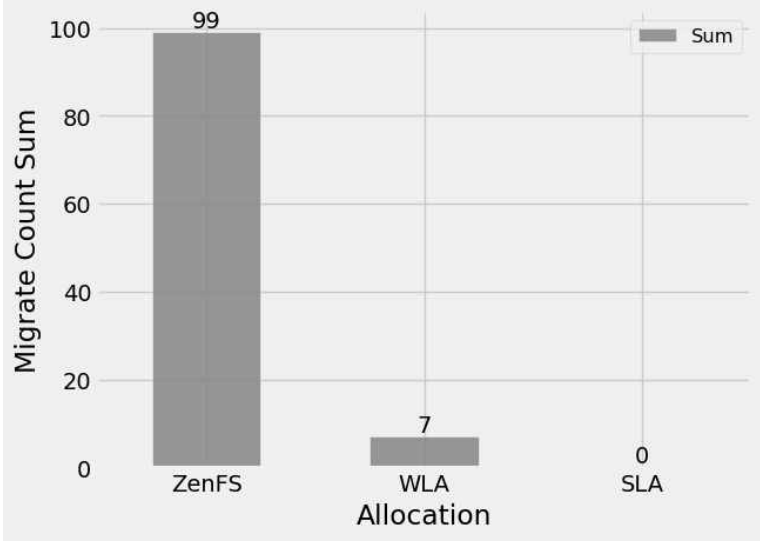
세 Zone 할당 방식의 성능을 측정하기 위해 벤치마크를 수행했다. **3.1절**의 벤치마크와 마찬가지로 가상환경에서 실험을 진행하였고 db_bench 옵션은 아래 **[표-6]**과 같다.

키(Key) 크기	16Bytes
밸류(Value) 크기	100Bytes
SSTfile 크기	64MB
총 write 크기	약 21GB

[표-6] db_bench 옵션(Option)

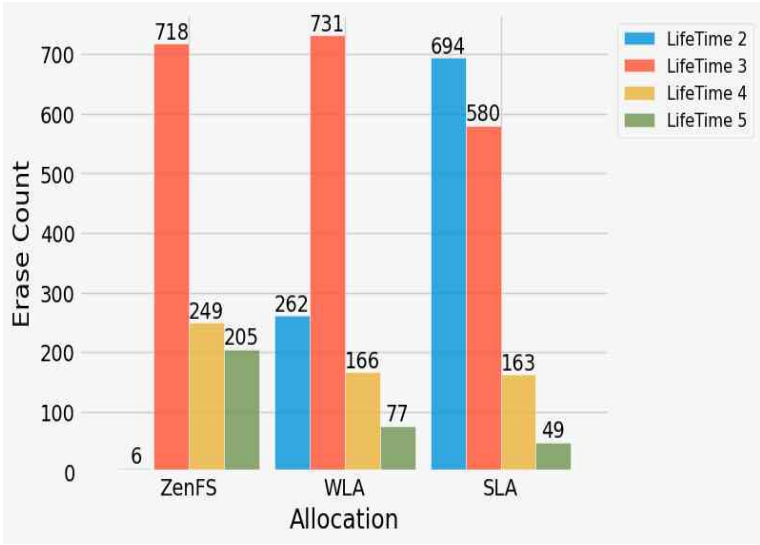
다른 기본적 옵션은 **3.1절**과 같고 기존 ZenFS에서 저장 공간이 부족하지 않도록 총 write 크기를 약 21GB로 지정했다. 그리고 실제 사용하는 서버 환경보다 작은 ZNS SSD의 용량을 고려해 LSM-Tree의 각 레벨이 증가할 때마다 용량이 2배씩 증가하도록 해 다양한 SSTfile의 Lifetime이 저장되도록 했다.

4.1.3 결과 및 WLA/SLA의 문제점

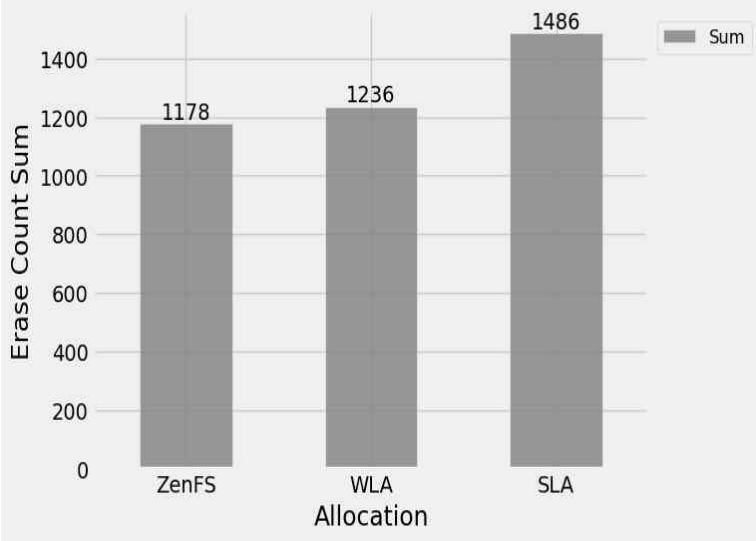


[그림-12] Zone 할당 정책에 따른 복사된 SSTfile 수

[그림-12]는 Zone 할당 정책에 따른 복사된 SSTfile의 개수를 측정한 그래프이다. 기존 ZenFS는 ZNS SSD에 무효 데이터가 많이 유지돼 GC가 많이 발생한다. 따라서 복사된 SSTfile의 수가 많고 이는 쓰기 증폭과 직결된다. 반면 WLA와 SLA는 같은 양의 데이터를 write했을 때 복사가 거의 일어나지 않음을 보여준다. 이는 Lifetime이 동일한 SSTfile을 같은 Zone에 모으는 방식이 현 ZenFS의 Zone erase 정책과 더 좋은 효율을 내는 것을 보여준다.



[그림-13] Zone 할당 정책에 따른 Lifetime별 Zone erase 수



[그림-14] Zone 할당 정책에 따른 Erase 된 Zone 수

[그림-13]은 Zone 할당 정책에 따라 erase가 발생한 Zone을 Lifetime별로 나눠 개수를 측정한 그래프이고 **[그림-14]**는 Zone 할당 정책별 erase가 발생한 Zone의 개수를 측정한 그래프이다. 기존 ZenFS에 비해 WLA는 약 5% 증가했고 SLA는 약 26% 증가했다. 동일한 Lifetime만 같은 Zone에 write할 수 있는 SLA에서 erase 수가 많이 증가한 것을 볼 수 있다.

[그림-13]의 결과를 보면 WLA와 SLA는 기존 ZenFS에 비해 Lifetime이 2인 Zone의 erase 수가 많이 증가했다. Lifetime이 2인 Zone의 erase 수를 전체 erase 수와 비교해 보면 기존 ZenFS는 약 0.5%를 차지하는 반면 WLA는 약 21%, SLA는 약 47%를 차지한다.

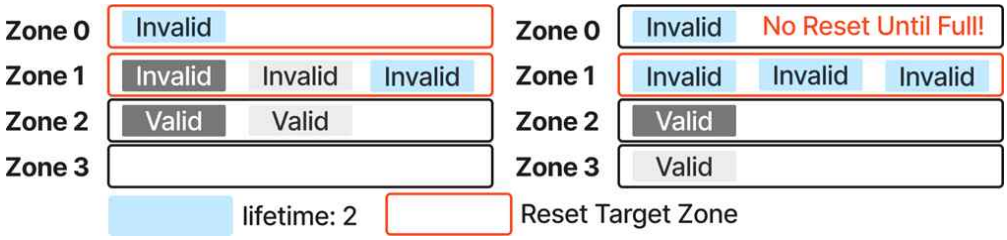
이러한 erase 수의 증가는 Zone erase 정책과 연관이 있다. **3.3절**의 Zone erase 정책 분석 내용에 따르면 ZenFS는 ZNS SSD의 공간을 주기적으로 확보하기 위해 SSTfile이 삭제될 때마다 무효 데이터만 존재하는 Zone을 erase하는 정책을 사용한다. 기존 ZenFS는 Lifetime이 2인 Logfile을 Lifetime이 3, 4, 5인 Zone에도 write할 수 있다. 하지만 SLA는 Lifetime이 2인 Logfile은 Lifetime이 2인 Zone에만 write 가능하다. 그리고 Logfile은 Memtable이 플러시 되면 바로 삭제되는 특성이 있다. 이러한 특성 때문에 SLA에서는 Logfile이 자주 write되고 삭제되면서 무효 데이터만 있는 Zone이 자주 생성되고 이는 erase 수 증가로 연결된다.

Zone 할당 정책 개선을 통해 Zone erase 정책을 더 잘 활용하고 여유 공간을 확보했다. 하지만 Zone의 erase가 증가하면 ZNS SSD의 수명이 감소한다. 따라서 erase 수를 줄일 수 있는 새로운 Zone erase 정책이 필요하다.

4.2 개선된 Zone erase 정책

4.2.1 Lazy Zone Reset: LZR

4.1절의 결과와 같이 현 ZenFS의 Zone erase 정책은 Lifetime이 2인 Zone의 erase가 빈번히 발생한다. 따라서 본 과제에서는 Lifetime이 2인 Zone의 erase를 줄이는 방식을 구현 및 평가했다. 해당 개선된 Zone erase 정책을 LZR(Lazy Zone Reset)이라 부르겠다.



[그림-15] (좌) 기존 Zone erase 정책 (우) LZR

Algorithm 1: LZR(Lazy Zone Reset)

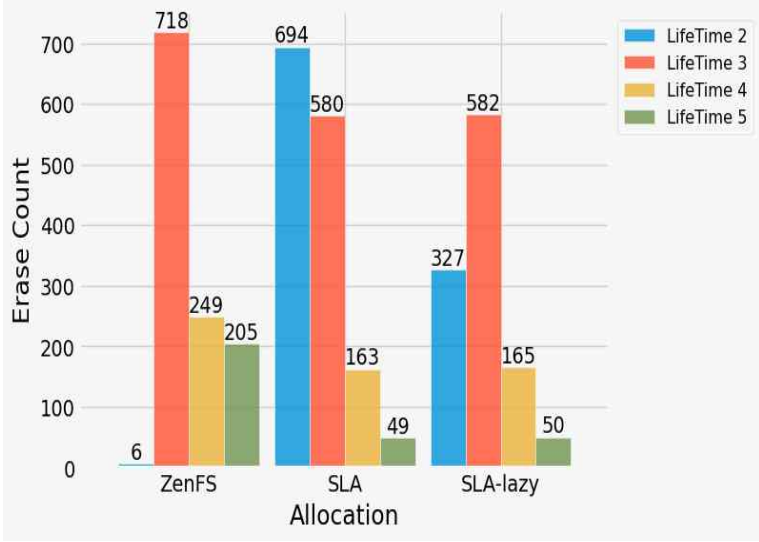
```
Data: Zones: array of pointers to io_zones
for i ← 1 to length of Zones do
    z ← Zones[i];
    if z → lifetime_ = 2 and z is not Full() then
        continue;
    if z is not Empty() and z is not Used() then
        Reset z
```

[코드-1] LZR 의사 코드

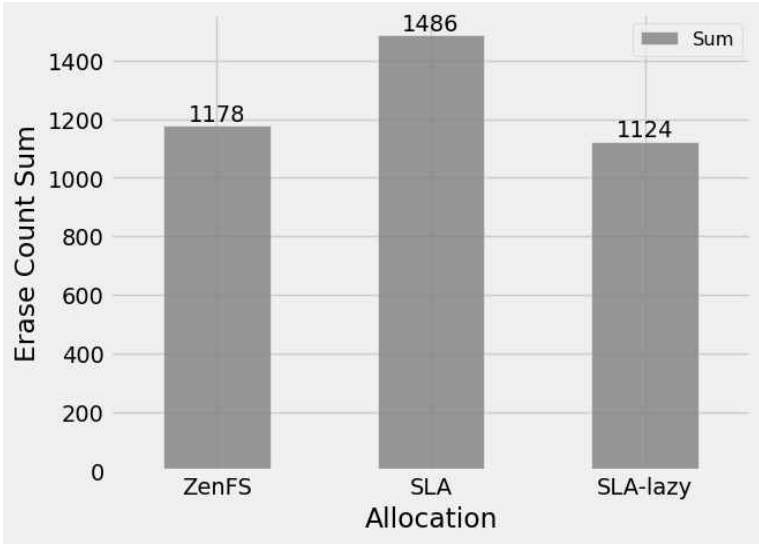
[그림-15]는 현 ZenFS의 Zone erase 정책과 LZR을 비교한 그림이고 **[코드-1]**은 LZR을 의사 코드(Pseudo-Code)로 나타낸 것이다. 데이터가 write되는 모든 Zone이 저장된 배열을 io_zones라 하고 해당 배열을 전부 스캔한다. 그리고 각 Zone을 검사해 lifetime이 2인 Zone은 가득 차 있지 않다면 더 이상 수행하지 않고 다음 Zone을 검사한다. 그리고 Lifetime이 3 이상이거나 2이고 가득 차있다면 무효 데이터만 존재하는지 검사하고 erase한다. 비어있지 않지만 사용 중인 데이터가 없는 Zone은 무효 데이터만 존재하는 Zone이다.

4.2.2 벤치마크 및 결과

LZR이 Lifetime이 2인 Zone의 erase 횟수를 얼마나 줄여주는지 알아보기 위해 실험을 진행했다. 실험 환경과 db_bench 옵션은 **4.1절**의 벤치마크 옵션과 동일하다.



[그림-16] Zone 할당 정책 및 Zone erase 정책에 따른 Lifetime별 Zone erase 수



[그림-17] Zone 할당 정책 및 Zone erase 정책에 따른 Zone erase 수

[그림-16]과 **[그림-17]**은 Zone 할당 방식 및 Zone erase 정책에 따른 Zone erase 횟수를 보여준다. 비교를 위해 기존의 Zone erase 정책을 사용하는 ZenFS, SLA와 개선된 Zone Erase 정책을 사용하는 SLA-lazy를 대상으로 진행한다. **[그림-16]**에서 SLA에 비해 SLA-lazy는 Lifetime이 2인 Zone의 erase 수가 약 52% 감소한 것을 확인할 수 있다. 기존 ZenFS에 비해 Lifetime이 2인 Zone의 erase 수는 대폭 증가하였지만 다른 Lifetime을 가진 Zone의 erase 수는 감소했다. 이는 **[그림-17]**의 결과와 같이 전체 erase 수가 소폭 감소한 것을 알 수 있다.

5. 연구 결과 분석 및 평가

5.1 실험 환경 및 평가

Used Zone	Zone 할당 정책의 효율성
Migration	GC의 영향
Erase Count	Zone erase 정책의 효율성

[표-7] 평가지표

우선 실험 환경은 위 **3.1절** 및 **4장**에서 수행한 환경과 동일한 가상 머신 FEMU를 사용했다. 평가 지표는 **[표-7]**과 같다.

키(Key) 크기	16Bytes
밸류(Value) 크기	100Bytes
SSTfile 크기	64MB
총 write 크기	약 22.4GB

[표-8] db_bench 옵션

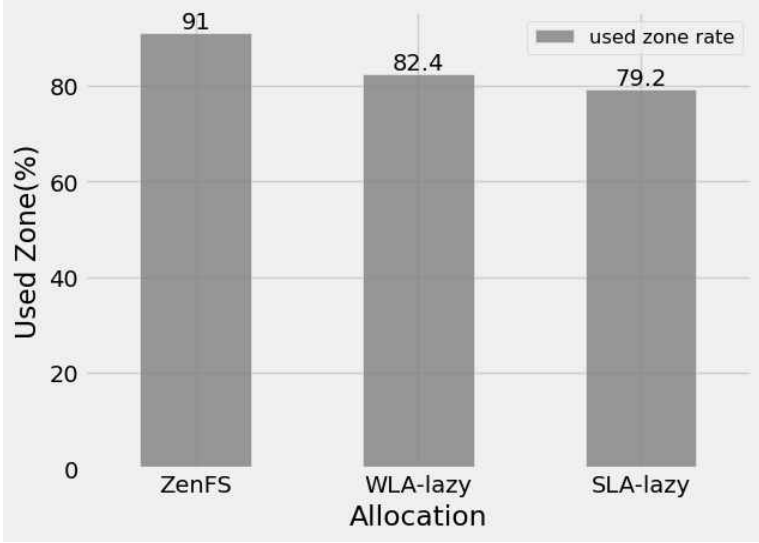
벤치마크 옵션은 **[표-8]**과 같다. LSM-Tree의 레벨당 저장 가능한 용량은 3장에서와 같이 2배씩 증가하게 해 실제 서버 환경에서와 같이 다양한 Lifetime의 SSTfile이 write되도록 했다.

	Zone 할당 정책	Zone erase 정책
ZenFS	Zone lifetime > SST lifetime	default
WLA-lazy	Zone lifetime >= SST lifetime	if z->lifetime == 2 && z->full()
SLA-lazy	Zone lifetime = SST lifetime	if z->lifetime == 2 && z->full()

[표-9] 평가 대상

[표-9]는 평가 대상이다. ZenFS는 기존 ZenFS이고 WLA-lazy와 SLA-lazy는 각각 **4.1절**의 WLA와 SLA의 Zone 할당 정책과 본 과제에서 제안한 LZR을 사용했다.

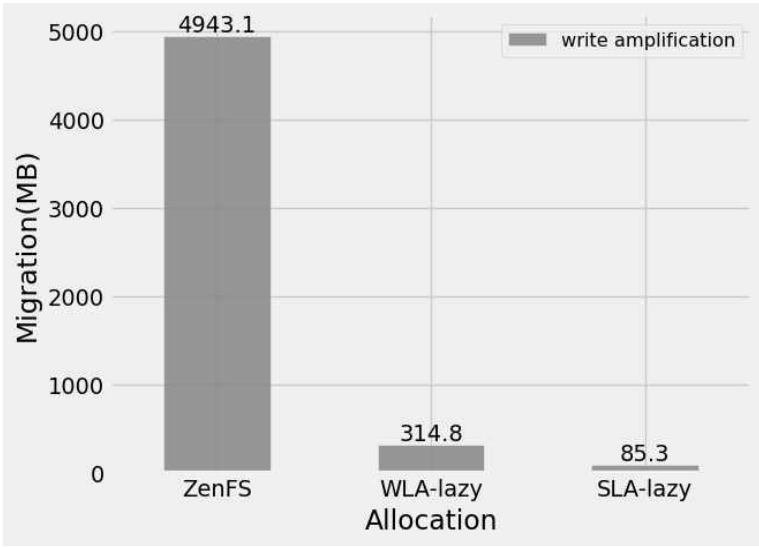
5.2 Used Zone



[그림-18] Zone 할당 방식 및 Zone erase 정책에 따른 사용 Zone 수

[그림-18]은 모든 write가 종료된 후 사용 중인 Zone의 비율을 측정한 그래프이다. 전체 Zone 수는 256개이고 ZenFS, WLA-lazy, SLA-lazy는 각각 233, 211, 203개의 Zone을 사용했다. 새로운 두 Zone 할당 정책이 더 적은 Zone을 사용했고 WSL-lazy가 가장 적은 Zone을 사용했음을 보여주는 결과이다.

5.3 Migration



[그림-19] Zone 할당 방식 및 Zone erase 정책에 따른 쓰기 증폭

[그림-19]는 GC로 인한 쓰기 증폭 양을 MB로 측정한 그래프이다. **5.2절**의 결과와 함께 보면 기존 ZenFS는 거의 대부분의 Zone을 사용하면서 저장 공간이 부족해지고 GC를 많이 수행하게 된다. 하지만 WLA-lazy와 SLA-lazy는

Zone을 효율적으로 사용하여 여유 공간이 충분해 GC가 적게 발생한다. 기존 ZenFS는 약 5GB의 쓰기 증폭을 발생시켰고 WLA-lazy와 SLA-lazy는 각각 약 314.8MB, 약 85.3MB로 큰 차이를 보였다.

5.4 Zone Erase Count

[그림-20] Zone 할당 방식 및 Zone erase 정책에 따른 Zone erase 수

위 **5.2절**, **5.3절**의 실험을 통해 Lifetime별로 SSTfile을 완전히 분리하는 SLA-lazy방식이 더 좋은 성능을 내는 것을 알 수 있었다. 이번 실험에서는 LZR의 성능을 알아보기 위해 SLA와 SLA-lazy를 비교한다. **[그림-20]**은 Zone erase 정책과 GC로 인해 erase 된 Zone의 수를 나타낸 그래프이다. SLA는 SLA-lazy와 동일한 Zone 할당 정책을 따르고 Zone erase 정책은 기본 ZenFS의 정책을 따른다. LZR을 구현한 SLA가 기존의 Zone erase 정책인 SLA에 비해 Zone erase 수가 약 24.7% 감소했다. 그리고 SLA-lazy는 현 ZenFS와 거의 같은 Zone erase 수를 보였고 GC로 인한 Zone erase 수까지 포함된다면 더 나은 모습을 보여준다.

6. 결론 및 향후 연구 방향

본 과제를 통해 ZNS SSD를 이용해 키-밸류 스토어 성능을 향상시켜줄 수 있는 파일시스템인 ZenFS를 분석하고 평가하였다. 현 ZenFS는 비슷한 Lifetime을 가진 데이터를 같은 Zone에 모아 GC를 늦추고 성능을 향상시키는 효과를 기대하고 있다. 하지만 실제 실험으로 확인해 본 결과 많은 무효 데이터들을 유지함으로 인해 ZNS SSD 전체 용량의 약 75.8%만 사용해도 저장 공간이 부족한 현상이 발생했다.

이에 본 과제는 현 ZenFS의 Zone 할당 정책을 개선해 같은 Lifetime의 데이터를 같은 Zone에 모으는 방식을 제안했다. 해당 Zone 할당 방식이 ZenFS의 Zone erase 정책과 더 좋은 효율을 내는 것을 확인했다. 이를 통해 무효 데이터를 덜 유지하여 여유 공간을 확보하고 GC를 덜 발생시켜 쓰기 증폭을 약 98.3% 감소시켰다. 그리고 GC로 인한 입/출력 성능 오버헤드를 감소시켰다. 추가적으로 과제 수행 중 해당 Zone 할당 방식이 Zone erase 수를 증가시켜 ZNS SSD의 수명에 영향을 미치는 것을 확인했고 새로운 Zone erase 정책인 LZR을 제안했다. 약 24.7% 증가했던 Zone erase 수도 LZR을 통해 기존의 ZenFS와 거의 같아졌다.

최근 ZenFS의 Zone erase 정책에 관한 논문을 접했다. 해당 논문에서는 ZenFS의 무분별한 Zone erase가 ZNS SSD의 수명에 악영향을 미친다고 말하고 새로운 Zone erase 정책을 제안했다. 현재 본 과제는 Lifetime이 2인 Zone에 관해서만 erase 수를 개선했지만 해당 논문과 같이 다양한 Lifetime의 Zone의 erase 수도 개선할 수 있는 방안에 연구할 계획이다.

7. 구성원별 역할 및 개발 일정

배재홍	- RocksDB 코드 분석 및 동작분석 - Zone 할당 정책 테스트 - Zone erase 정책 고안 및 구현
이재석	- RocksDB 코드 분석 및 동작분석 - 성능 개선점 확인 - 테스트 및 디버깅
조준호	- ZenFS 코드 분석 및 동작분석 - Zone 할당 정책 구현 - 성능 평가 및 결과 분석

공통	- 보고서 작성 - 발표 준비

8. 멘토링 결과 반영 내용

연구 목적 중 하나인 GC를 늦춰 쓰기 증폭을 줄이고 입/출력 오버헤드를 줄이는 것임에 따라 GC에 대한 더 자세한 설명이 필요하다고 피드백 받았다. 해당 피드백을 바탕으로 **2.1절** ZNS SSD에 관한 설명에 무효 데이터가 생기는 이유와 GC가 필요한 이유, 쓰기 증폭이 발생하는 이유 등 상세한 설명을 추가했다.

본 연구가 ZNS SSD를 에뮬레이트 한 가상 환경에서 진행하는 만큼 해당 가상 머신인 FEMU의 정확성에 대한 설명이 필요하다 피드백 받았다. 이에 FEMU 제작자가 쓴 논문을 바탕으로 **2.4절** FEMU의 정확성을 중심으로 한 배경지식을 추가했다.

중간 결과에서의 RocksDB 및 ZenFS의 코드 레벨 분석은 좋았지만 해당 내용이 본 연구와 어떠한 연관이 있는지에 대한 설명이 부족하다 피드백 받았다. 따라서 동작을 이해하기 위한 분석은 제외하고 본 연구와 관련된 분석 위주의 내용과 연구 목적을 연관 지어 **3장** 연구 동기에 작성했다.

9. 참고 문헌

[1] Zone Storage IO [Online]. Available: https://zonedstorage.io/docs/introduction/zns

[2] Facebook RocksDB github wiki [Online]. Available: https://github.com/facebook/rocksdb/wiki

[3] G. Oh, J. Yang, S. Ahn, “Efficient Key-Value Data Placement for ZNS SSD” Applied Sciences, Nov, 2021

[4] H. Lee, C. Lee, S. Lee, Y. Kim, “Compaction-Aware Zone Allocation for LSM based Key-Value Store on ZNS SSDs” In proceedings of 14th ACM Workshop on Hot Topics in Storage and File System(HotStorage’ 22), 2022

[5] github io getchan LSM-Tree [Online]. Available: https://getchan.github.io/data/LSM_tree/

[6] FEMU github [Online]. Available: https://github.com/vtess/FEMU

[7] Huaicheng Li, Mingzhe Hao and Michael Hao Tong. “The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator” In proceedings of the 16th USENIX Symposium on File and Storage Technologies(FAST), pp. 83-90, Feb, 2018

[8] ZenFS github [Online]. Available:
https://github.com/westerndigitalcorporation/zenfs/blob/master/README.md

[9] RocksDB db_bench github wiki [Online]. Available:
https://github.com/EighteenZi/rocksdb_wiki/blob/master/Benchmarking-tools.md

[10] 이영재, 정지윤, 신동군, “Zoned Namespace SSD 를 위한 LSM-tree 최적화 기법”, 한국소프트웨어종합학술대회, pp. 1107-1109, 2021