공학석사학위논문

작은 존 ZNS SSD에 맞는 RocksDB와 ZenFS 분석 및 최적화

An Analysis and Optimization of Key-Value Store for small zone ZNS SSD

2022년 8월

서울대학교 대학원

컴퓨터 공학부

이 정 은

작은 존 ZNS SSD에 맞는 RocksDB와 ZenFS 분석 및 최적화

An Analysis and Optimization of Key-Value Store for small zone ZNS SSD

지도교수 염헌영

이 논문을 공학석사 학위논문으로 제출함

2022년 4월

서울대학교 대학원

컴퓨터 공학부

이정은

이정은의 공학석사 학위논문을 인준함

2022년 6월

위 원 장	장 병 탁	(인)
부위원장	염 헌 영	(인)
위 원	전 병 곤	(인)

초록

ZNS SSD는 전체 용량을 일정한 크기의 존(Zone)으로 나누고 사용 주기가 비슷한 데이터를 같은 존에 저장하는 차세대 SSD이다. 존의 크기에 따라 ZNS SSD의 이점이 달라진다. 여러 존을 동시에 읽고 쓰는 패러랠 I/O를 한다면 작은 존 ZNS SSD의 이점을 살릴 수 있다. 그러나 현재 RocksDB와 ZenFS는 패러랠 I/O를 지원하지 않는다. 본 논문에서는 작은 존 ZNS SSD를 활용하기 위해서 RocksDB와 ZenFS에 패러랠 I/O 적용하였고, 더블 버퍼링과 컴팩션 최적화를 통해 성능 향상을 이루었다. 이를 통해 총 소요 시간은 약 57% 줄었고, ZNS SSD 읽기/쓰기 처리량은 약 3배 증가하였다.

주요어: ZNS SSD, RocksDB, ZenFS, 컴팩션, 패러랠 I/O, 컴팩션 최적화

학번: 2020-26170

목차

그림 목차

제 1 장 서 론

1.1 연구의 배경

1.1.1 ZNS SSD

ZNS SSD란, SSD의 전체 용량을 일정한 크기의 존(Zone)으로 나누고 사용주기가 비슷한 데이터를 같은 존에 저장하여 일반적인 SSD보다 효율적으로 활용할 수 있는 SSD이다. 각 존에 사용주기가 비슷한 데이터를 저장함으로써 사용되지 않는 공간인 가비지(Garbage) 영역이 생기지 않아서 가비지 컬렉션(Garbage Collection)을 할 필요가 없다. 이 과정에서 추가적인 읽기/쓰기가 발생하지 않는다. 낸드 플래시는 쓰기에 대한 수명이 있기 때문에 ZNS SSD는 기존 SSD보다 수명이 길다. 또한, 일반적인 SSD는 낸드 칩의 성능 향상과 효율적인 관리를 위해 전체 용량의 일정 부분을 OP(Over-Provisioning) 영역으로 할당해 두는데, ZNS SSD에서는 별도의 OP 영역을 할당할 필요가 없어 SSD를 최대 용량으로 활용할 수 있다. [1]

그림 1.1은 일반 SSD와 ZNS SSD의 데이터 저장 방식을 비교한 그림이다.

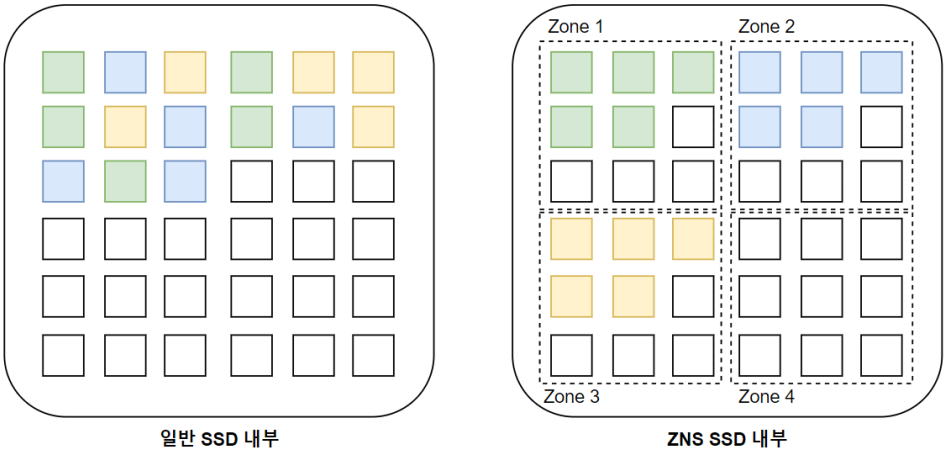


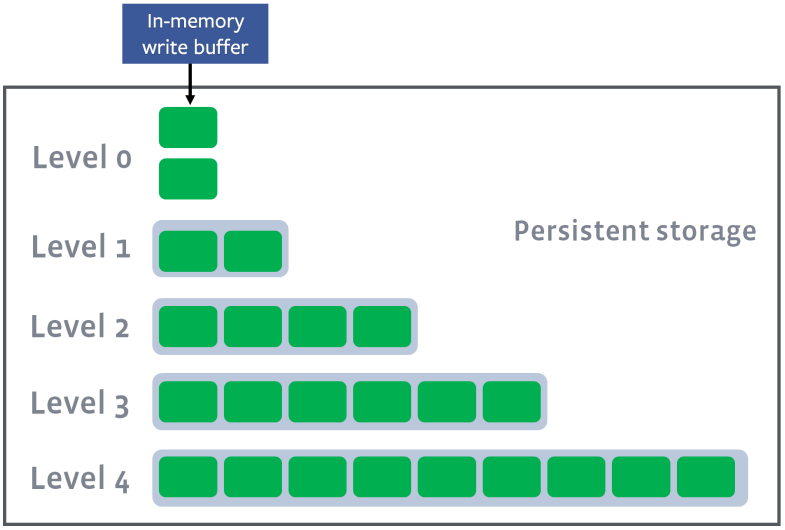
그림 1.1 일반 SSD와 ZNS SSD의 데이터 저장 방식 비교

ZNS SSD는 일반 SSD와 달리 각 존에 사용주기가 비슷한 데이터를 저장하는 것을 그림 1.1을 통해 확인할 수 있다.

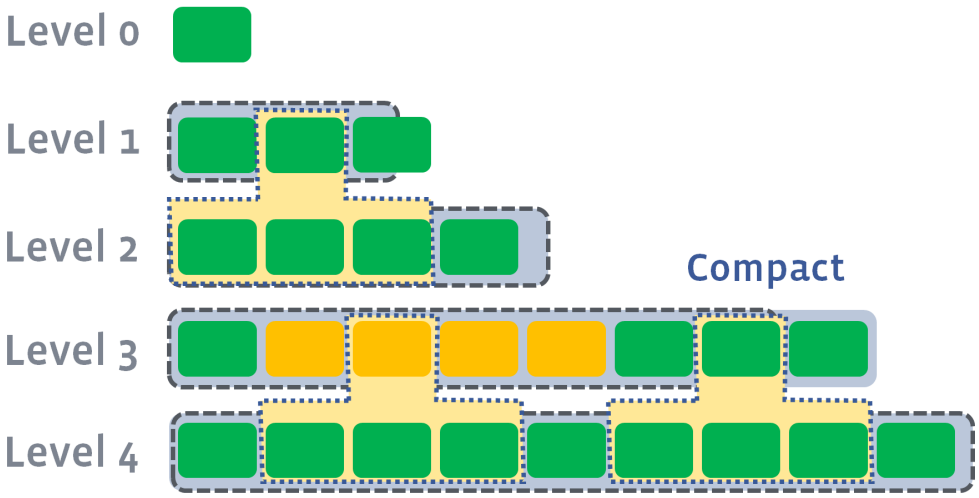
1.1.2 RocksDB

RocksDB란, LSM-tree(Log-Structured Merge tree) 기반 키-밸류 스토어이다. [2] LSM-tree는 키-밸류를 계층적 관계로 여러 레벨로 분류하여 데이터를 할당하는 자료구조이다. RocksDB는 플러시(Flush)와 컴팩션(Compaction)이라는 작업을 통해 LSM-tree를 유지한다. RocksDB는 각 레벨에 특정 크기(기본: 64MB)로 SST 파일을 생성하여 키-밸류를 저장한다. 먼저, RocksDB는 클라이언트로부터 쓰기 요청받으면 멤테이블(Memtable)에 저장한다. 멤테이블이란, SST 파일로

플러시를 하기 전에 저장하는 인-메모리 쓰기 버퍼이다.



(a) 플러시



(b) 멀티 쓰레드 컴팩션

그림 1.2 RocksDB의 플러시/컴팩션 과정

그림 1.2(a)과 그림 1.2(b) RocksDB의 플러시와 컴팩션을 나타낸 그림이다.

멤테이블이 가득 차면 SST 파일로 변환하여 LSM-tree의 레벨0으로 플러시가 발생하고, 각 레벨에 존재하는 SST 파일들의 크기가 일정한 값을 넘어가면 해당 레벨에서 컴팩션이 발생한다. 컴팩션은 서로 다른 SST 파일들을 합치고 겹치는 키-밸류는 삭제하여 그다음 레벨에 새로운 SST 파일을 생성하는 작업이다.

1.1.3 ZenFS

ZenFS는 RocksDB의 파일시스템 인터페이스를 활용하여 ZNS SSD의 존에 파일 읽기/쓰기를 하는 파일시스템 플러그인이다.

그림 1.3은 ZenFS의 구조를 나타낸 그림이다. ZenFS를 통해 파일을 서로 다른 존으로 나누고 사용주기 힌트를 활용하여 유사한 사용주기의 데이터를 같은 존에 배치함으로써 WAF(Write Amplification)를 일반적인 SSD에 비해 크게 줄였다. ZenFS는 파일 시스템이나 디스크에 백그라운드 가비지 컬렉션이 없도록 하여 처리량(Throughput), 꼬리 응답시간(Tail Latency) 및 디스크 내구성 측면에서 성능을 향상시킨다. [3, 4]

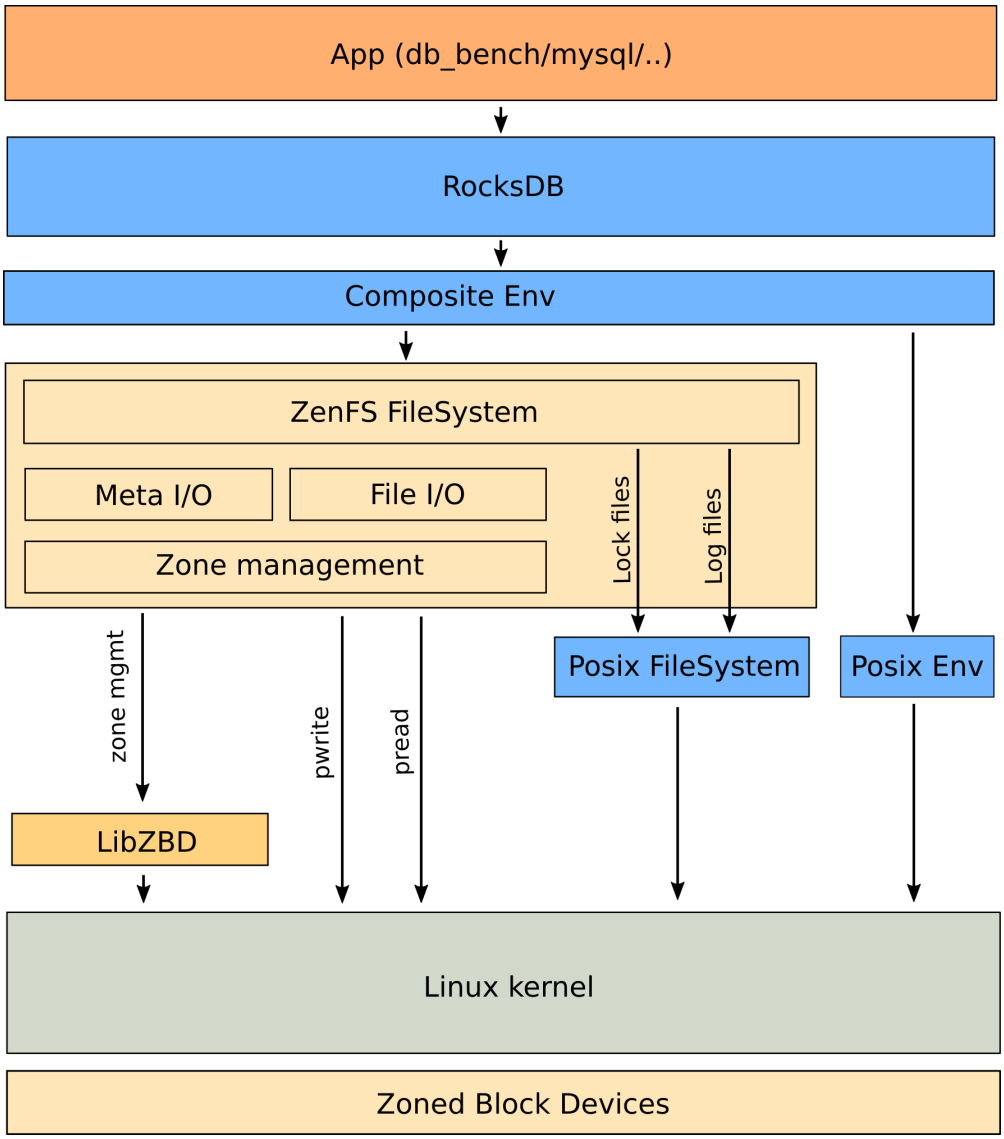


그림 1.3 ZenFS 구조

1.2 연구의 목적

ZNS SSD는 존의 크기에 따라서 이점이 달라진다. 존의 크기가 커지면 전체 존의 개수가 작아지고, 반대로 존의 크기가 작아지면 존이 많아진다. 존의 개수

가 많이 없다면 하나의 존에 상대적으로 많은 채널이 할당되어서 각 존의 성능이 좋을 것이다. 반대로, 존의 개수가 많아지면 하나의 존 성능이 상대적으로 떨어질 것이다. 그러나 ZNS SSD의 존은 독립적인 I/O를 할 수 있기 때문에 존의 개수가 많아지면 패러랠(Parallel) I/O를 통해 여러 존을 동시에 활용한다면 존의 크기가 클 때보다 더 좋은 I/O성능을 낼 수 있을 것이다. [5] 또한, ZNS SSD는 존 안의 모든 데이터가 무효화(Invalid)되면 존-리셋(zone-reset) 명령어를 통해 존 안의 데이터를 삭제한다. 존의 크기가 크면 클수록 존 안의 모든 데이터가 무효화될 확률이 적어지고 존-리셋을 할 일이 많이 없어져 단편화(Fragmentation)가 심해진다. 현재 RocksDB와 ZenFS에서는 패러랠 I/O를 하고 있지 않기 때문에, 큰 존을 활용했을 때 보다 성능이 많이 떨어지는 것을 확인하였다. 작은 존 ZNS SSD를 잘 활용하기 위해 RocksDB와 ZenFS에 패러랠 I/O를 구현과 새로운 플러시 / 컴팩션 정책이 필요하다.

1.3 구성

본 논문은 다음과 같은 구성을 갖는다.

- **제 2 장**은 RocksDB와 ZenFS의 기존 형태와 패러랠 I/O를 적용한 형태를 비교하여 설명하고, 컴팩션 최적화 기법에 대해 설명한다. 또한, 실험을 통해 적용한 기법의 성능향상 정도를 보여준다.

- **제 3 장**은 연구의 요약 및 결론을 짓는다.

제 2 장 본 론

2.1 연구의 내용

현재 RocksDB와 ZenFS에서는 패러랠 I/O를 지원하지 않는다. 그러나 작은 존 ZNS SSD에 패러랠 I/O를 적용하여 여러 존을 활용하면 성능을 높일 수 있다.

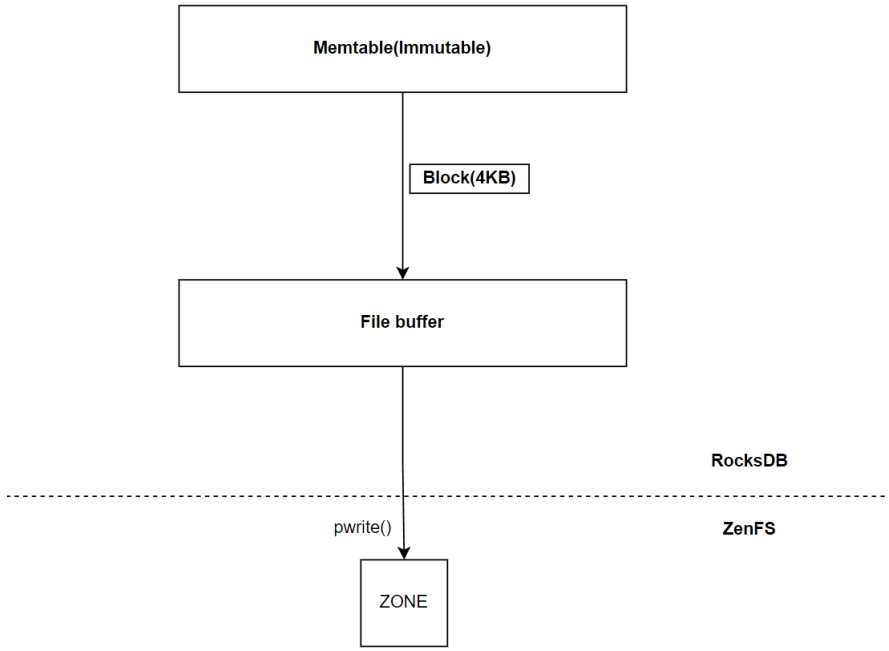


그림 2.1 RocksDB 기존 플러시 쓰기 경로

그림 2.1는 기존 RocksDB의 플러시 쓰기 경로를 나타낸 그림이다. 멤테이블

이 가득 차면 멤테이블을 SST파일 형태로 변형하여 데이터를 파일 버퍼에 쓴다. 파일 버퍼가 가득 차면 ZNS SSD의 존에 데이터를 쓴다. 기존에는 파일 버퍼(기본: 1MB) 1개 존재하고, 순차적으로 데이터 쓰기를 했다. 그렇게 되면 존을 동시에 2개 이상 사용하기 어렵기 때문에 ZNS SSD의 대역폭을 1개 존에 대해서만 활용할 수 있었다. 작은 존 ZNS SSD의 대역폭을 여러존에 대하여 사용하기 위해서는 패러랠 I/O가 필요하다.

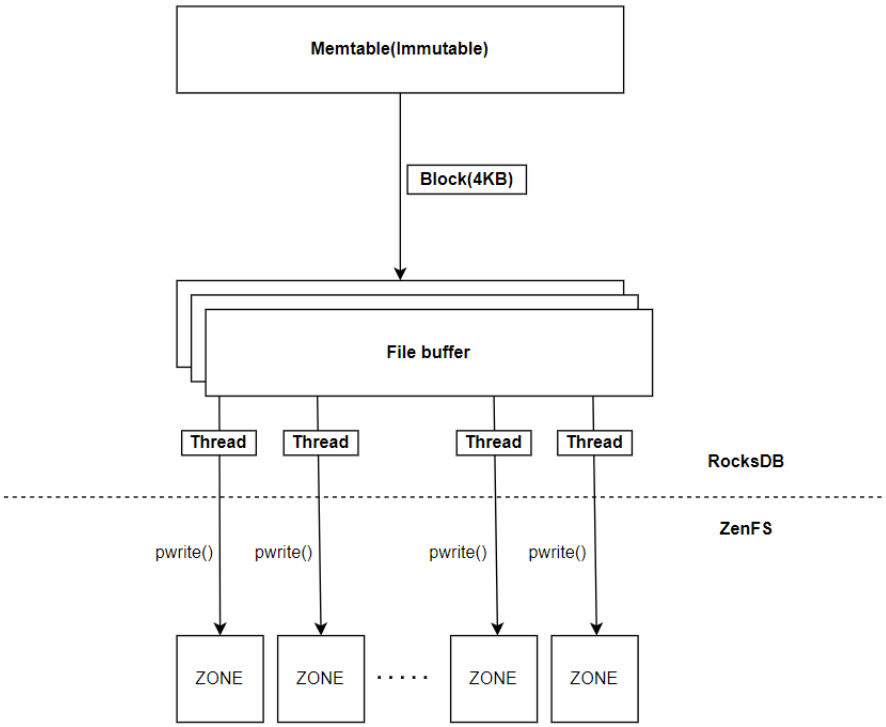


그림 2.2 RocksDB 패러랠 I/O 플러시 쓰기 경로

그림 2.2는 패러랠 I/O를 적용한 플러시의 쓰기 경로를 나타낸 그림이다. SST 파일 마다 특정 개수의 존을 할당하고 파일 버퍼가 가득 차면 쓰레드를 생성하여 라운드 로빈(Round-robin)형태로 각 존에 쓰기를 진행한다. 그렇게 되면 동시에 사용하고 있는 존의 개수 만큼 대역폭을 사용할 수 있다.

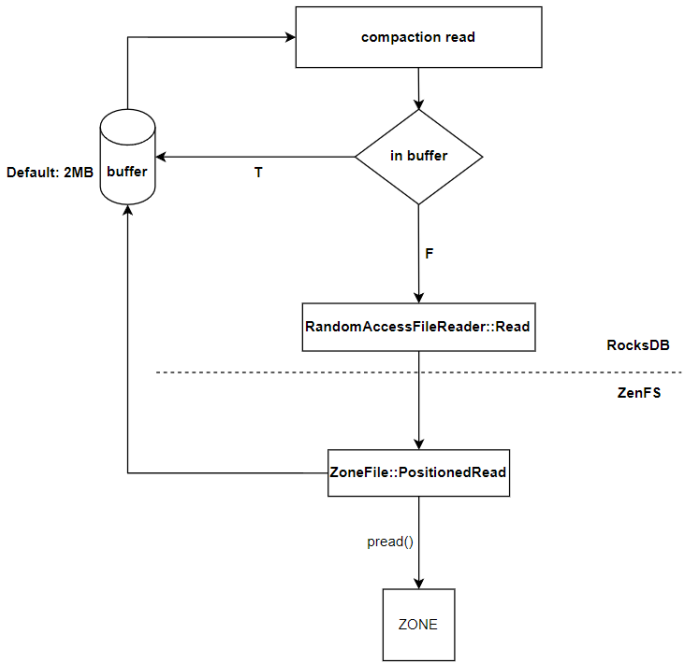


그림 2.3 RocksDB 기존 컴팩션 읽기 경로

그림 2.3는 기존 컴팩션 과정에서 발생하는 읽기 경로를 나타낸 그림이다. 컴팩션 과정에서 컴팩션 인풋으로 지정된 파일의 데이터를 읽어와 보관하는 버퍼(기본: 2MB)가 존재한다. 버퍼에 원하는 데이터가 없는 경우 버퍼의 크기만큼 ZNS SSD

에서 읽어온다.

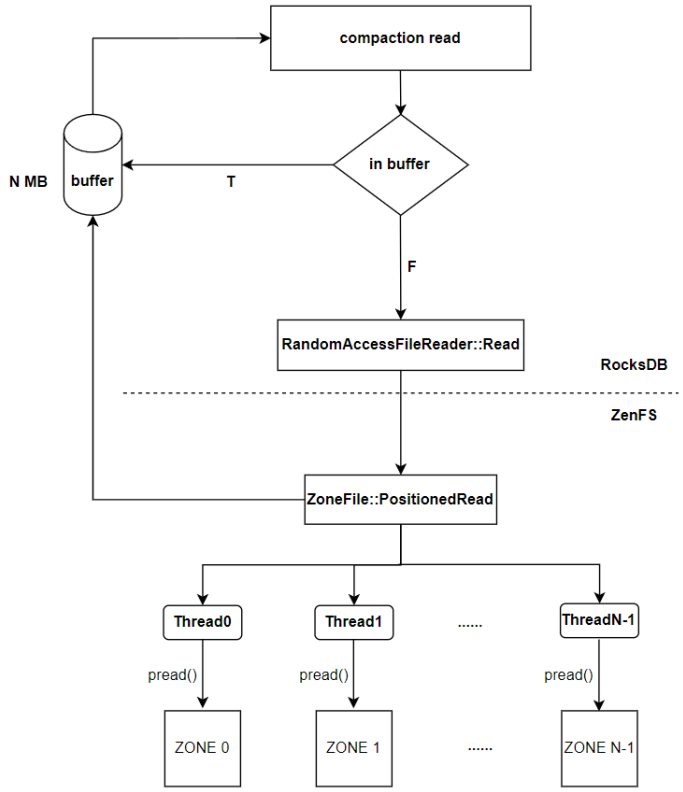


그림 2.4 RocksDB 패러랠 I/O 컴팩션 읽기 경로

그림 2.4는 패러랠 I/O를 적용한 컴팩션 읽기 경로를 나타낸 그림이다. SST 파일 쓰기를 라운드 로빈 형태로 진행하였기 때문에 여러 존에서 동시에 데이터를 읽어올 수 있다. 이때, 버퍼 사이즈를 N MB로 증가시키고 N개의 존에서 동시에 읽기를 수행한다. 그럼 N개 존 만큼 ZNS SSD의 대역폭을 사용할 수 있다.

2.1.1 더블 버퍼링

컴팩션 읽기 과정에서 버퍼에 원하는 데이터가 없을 경우 ZNS SSD에서 버퍼 크기만큼 데이터를 읽어온다. 컴팩션은 SST 파일을 순차적으로 읽기 때문에 버퍼를 다 읽을 때까지 기다리지 않고, 더블 버퍼링을 적용하여 백그라운드로 미리 그다음 필요한 데이터까지 읽어올 수 있다. 그림2.5는 더블 버퍼링을 적용한 컴팩션 읽기 경로를 나타낸 그림이다. 버퍼를 2개 할당하여 첫 번째 버퍼를 읽어오고 컴팩션이 진행되는 동안 두번째 버퍼를 백그라운드에서 읽어온다. 첫 번째 버퍼에 원하는 데이터가 없을때 두 번째 버퍼를 사용하고, 다시 첫 번째 버퍼는 백그라운드에서 읽어오는 방식의 더블 버퍼링을 사용하여 ZNS SSD에서 읽어오는 시간을 줄일 수 있다.

2.1.2 컴팩션 최적화

기존 RocksDB에서 컴팩션 대상 SST 파일을 선정할 때, 베이스(Base) 레벨에서 가장 큰 파일을 선정하고 아웃풋(Output) 레벨에서 겹치는 파일을 선정하여 컴팩션을 진행하였다.

그림 2.6(a)는 기존 RocksDB에서 컴팩션 대상 파일을 선정하는 방식을 나타낸 그림이다. L1이 베이스 레벨이고 L2가 아웃풋 레벨일때, L1에서 가장 큰 파일인 a

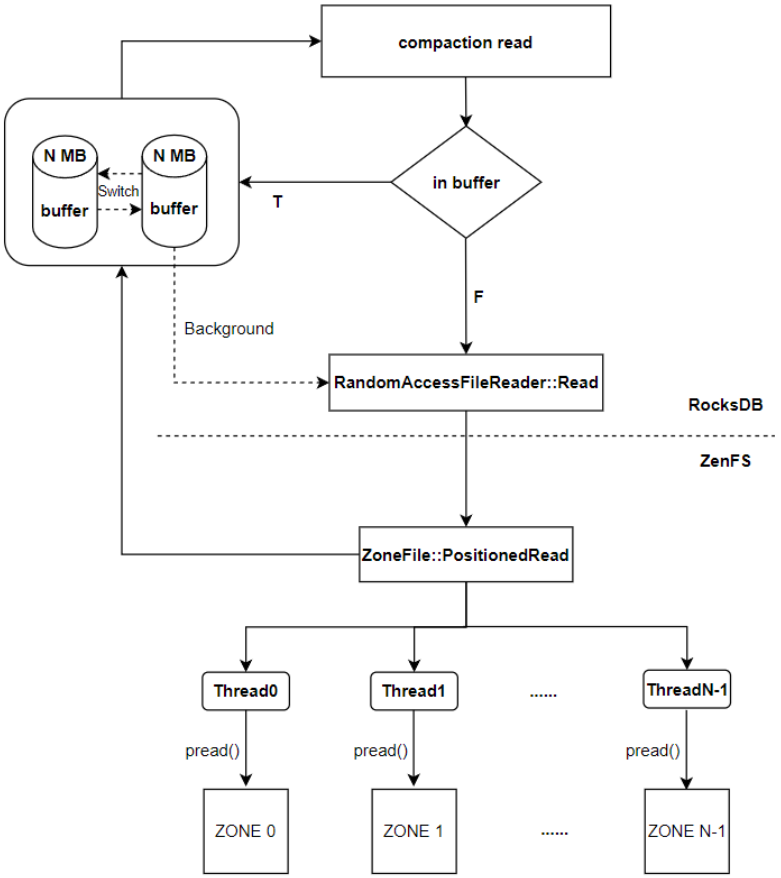
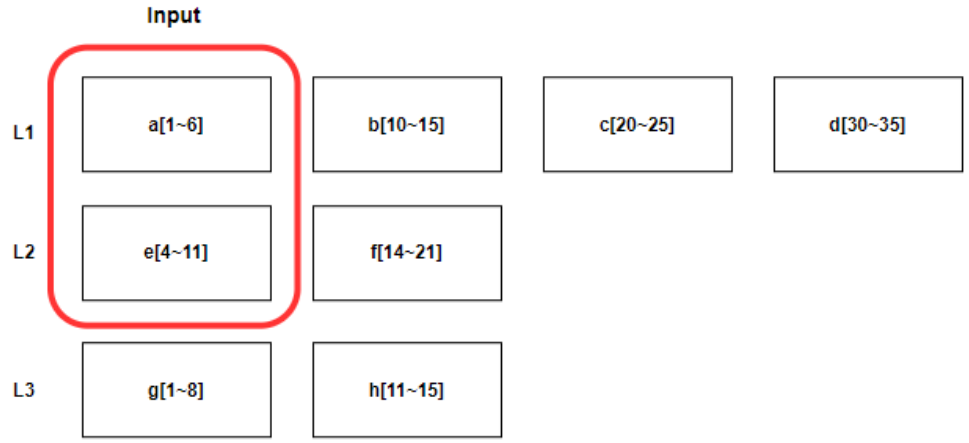
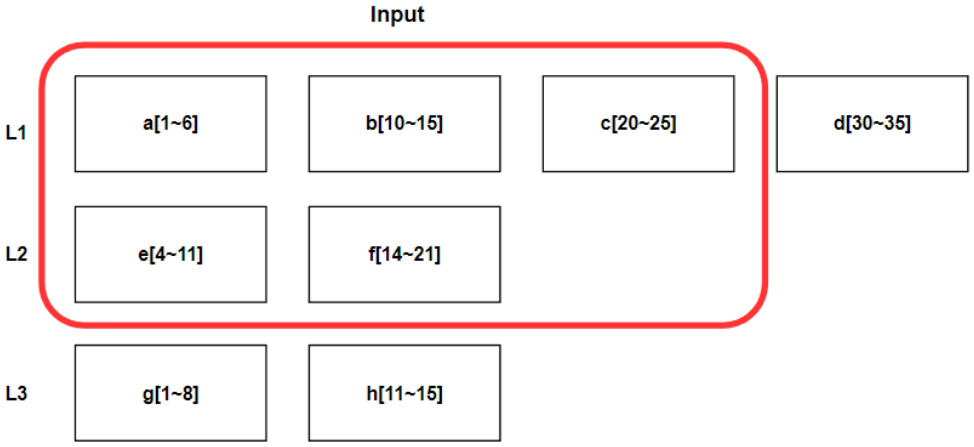


그림 2.5 더블 버퍼링 구조

를 선정하고 L2에서 a와 겹치는 키를 갖고있는 e를 선정하여 컴팩션을 진행한다. 그 결과로 L2에 i[1 11] 파일이 생성되고 b와 컴팩션이 진행된다면, 겹치지 않는 부분인 [1 9]을 한번 더 읽고 쓰게 된다. 위와 같은 불필요한 중복 읽기 과정을 최소화하기 위해 그림 2.6(b)와 같이 겹치는 모든 파일을 컴팩션 대상으로 선정하는



(a) 기존 컴팩션 대상 파일 선정



(b) 그리디 컴팩션 대상 파일 선정

그림 2.6 RocksDB의 컴팩션 대상 파일 선정

그리디(Greedy)한 방식을 적용하였다.

RocksDB에서 컴팩션을 수행할 때, 컴팩션 결과 파일의 크기를 제한할 수 있다. 결과 파일에 키-밸류를 더하였을 때 그 크기가 최대 결과 파일 크기보다 큰

Algorithm 2.1: Compaction

```
Data: Compaction Input Files
Result: Compaction Output Files
for key : input_keys do
    write key to current SST file;
    if current file size is bigger than max_compaction_bytes then
        create new SST file;
    else
        continue;
```

경우 새로운 결과 파일을 생성한다. 겹치는 파일이 아무것도 없어 컴팩션 대상 SST 파일이 1개일 경우, 파일을 모두 읽고 쓰는 과정을 생략하고 아웃풋 레벨로 이동한다. 그러나 컴팩션 대상 파일이 최대 결과 파일 크기보다 큰 경우 파일을 모두 읽고 쓰는 컴팩션 과정을 수행한다. 또한, 컴팩션 결과 파일 크기를 제한할 때 알고리즘 2.1과 같이 마지막 키-밸류를 더하고 최대 결과 파일 크기와 비교하기 때문에 최대 결과 파일보다 큰 파일이 존재하게 된다. 결국 SST파일의 크기는 최대 결과 파일 크기보다 키-밸류 한개만큼 커지게 된다. 이러한 경우 레벨만 이동하게 하여 불필요한 컴팩션을 없앴다.

2.2 실험의 내용

2.2.1 실험 환경

하드웨어 : Intel(R) Core(TM) i7-12700K 20개, 48GB 메모리, SK Hynix ZNS SSD를 사용하였고, ZNS SSD의 존 크기는 96MB, 존 실제 용량은 96MB, 총 용량 34TB이다.

시스템 구성 : 우분투 21.04, 커널 5.16.0 환경에서 RocksDB v6.25.3, ZenFS v1.0.0를 기반으로 구현하였다. 멤테이블 크기 2GB, 최대 컴팩션 결과 파일의 크기 2GB, 읽기 버퍼 16MB, 쓰기 파일 버퍼 1MB로 설정하였다.

워크로드 : db_bench의 fillrandom 워크로드를 사용하여 실험하였다. db_bench는 RocksDB 성능을 벤치마킹하는 주요 도구이다. db_bench에 waitforcompaction 옵션을 설정하여 모든 컴팩션이 완료되게 하였다.

2.2.2 실험 결과

패러랠 I/O와 더블 버퍼링

총 소요 시간 : 그림 2.7는 기존 RocksDB와 ZenFS와 패러랠 I/O와 더블 버퍼링을 적용한 버전을 db_bench를 통해 실험하여 총 소요 시간을 비교한 그래

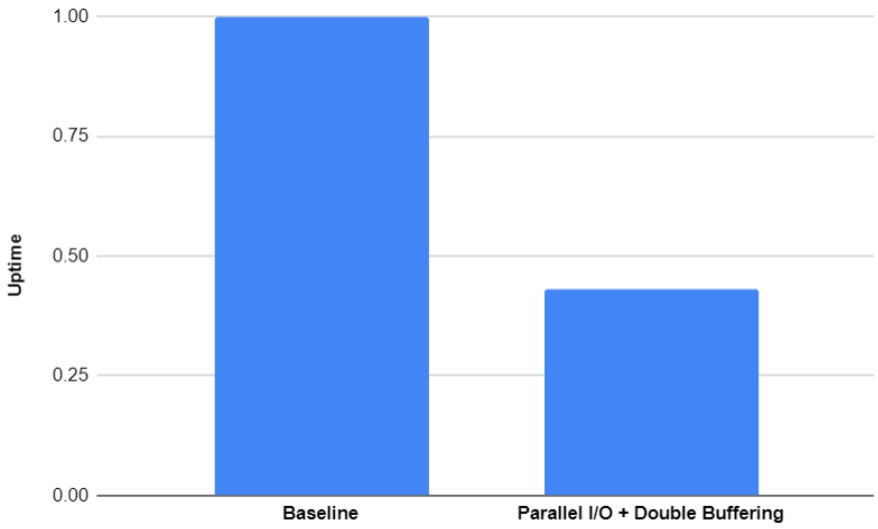


그림 2.7 버전별 실험 총 소요 시간 비율

프이다. 기존 버전의 시간을 1로 설정할 때 패러랠 I/O와 더블 버퍼링을 적용한 버전의 총 소요 시간은 0.43이다. 패러랠 I/O와 더블 버퍼링을 통해 약 57%의 총 소요 시간을 단축할 수 있다.

처리량(Throughput) : db_bench를 수행하는 동안 ZNS SSD의 처리량을 iostat 명령어를 통해 측정하였다. 그림 2.8는 버전별 ZNS SSD의 읽기/쓰기 처리량 최댓값 비율을 나타낸 그림이다. 기존 버전에서 패러랠 I/O와 더블 버퍼링을 적용하여 약 3배 정도의 처리량 최댓값이 증가한 것을 확인할 수 있다.

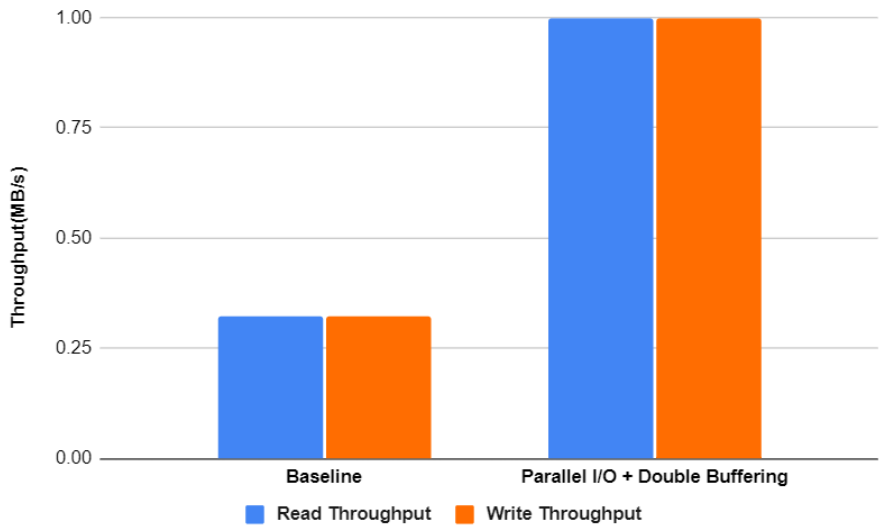


그림 2.8 버전별 ZNS SSD의 읽기/쓰기 처리량 최댓값 비율

	Uptime	Amount of Compaction Write	Amount of Compaction Read	Number of Compaction	WAF
Parallel I/O + Double Buffering	1.0	1.0	1.0	1.0	1.0
Compaction Optimization	0.91	0.87	0.85	0.61	0.88

그림 2.9 컴팩션 최적화에 따른 결과

컴팩션 최적화

그림 2.9은 패러랠 I/O와 더블 버퍼링을 적용한 버전과 거기에 컴팩션 최적화까지 적용한 버전을 비교한 것이다. 컴팩션 최적화를 통해 약 10%의 총 소요시간이 줄어들었고, 쓰기/읽기하는 양이 약 13%, 15%씩 각각 줄어든것을 확인할 수 있다. 쓰기 양이 줄어들어 WAF도 약 12% 감소하였고, 컴팩션 횟수도 약 39% 감소한

것을 확인할 수 있다. 컴팩션 최적화를 통해 컴팩션 횟수를 줄이고, 성능을 향상 시킬 수 있다.

제 3 장 결 론

3.1 결과 분석

ZNS SSD는 전체 용량을 일정한 크기의 존(Zone)으로 나누고 사용주기가 비슷한 데이터를 같은 존에 저장하는 차세대 SSD이다. ZNS SSD는 일반적인 SSD에 대비하여 가비지 컬렉션을 하지 않고 OP 영역을 할당할 필요가 없는 장점이 존재한다. ZNS SSD는 존의 크기에 따라 이점이 달라지는데, 본 논문에서는 작은 존의 이점을 살리기 위해 RocksDB와 ZenFS를 분석하였다. 크기가 작은 존을 사용하면 동시에 여러 존을 사용하는 패러랠 I/O를 사용하면 이점을 살릴 수 있다. 그러나 기존 RocksDB와 ZenFS에서는 지원하지 않았다. 본 논문에서는 패러랠 I/O, 더블 버퍼링을 구현하여 적용하였고, 컴팩션 최적화를 수행하였다. 패러랠 I/O와 더블 버퍼링을 통해 기존에 대비하여 총 소요 시간은 약 57% 줄일 수 있었고, ZNS SSD의 처리량 최댓값은 약 3배 정도 증가시킬 수 있었다. 또한, 컴팩션 최적화를 통해 컴팩션 횟수와 WAF가 줄어들어 성능향상을 얻을 수 있었다.

참고문헌

[1] "삼성전자, 차세대 기업 서버용 'ZNS SSD' 출시." https://news.samsung.com/kr/%EC%82%BC%EC%84%B1%EC%A0%84%EC%9E%90-%EC%B0%A8%EC%84%B8%EB%8C%80-%EA%B8%B0%EC%97%85-%EC%84%9C%EB%B2%84%EC%9A%A9-zns-ssd-%EC%B6%9C%EC%8B%9C.

[2] "A Persistent Key-Value Store for Fast Storage Environments." https://rocksdb.org/.

[3] "ZenFS: RocksDB Storage Backend for ZNS SSDs and SMR HDDs." https://github.com/westerndigitalcorporation/zenfs.

[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] K. L. J. C. J. J. G. Choi, M. Oh and Y. Oh, "A new lsm-style garbage collection scheme for zns ssds," in *USENIX Hotstorage.*, 2020.

Abstract

ZNS SSD is a next-generation SSD that divides the entire capacity into zones of a certain size and stores data with similar usage cycles in the same zone. The benefits of ZNS SSDs depend on the size of a zone. Parallel I/O that reads and writes multiple zones at the same time can take advantage of the small zone ZNS SSD. However, currently RocksDB and ZenFS do not support Parallel I/O. In this paper, parallel I/O was applied to RocksDB and ZenFS to utilize the small ZNS SSD, and performance was improved through double buffering and compaction optimization. Through this, the required time was reduced by about 60%, and the ZNS SSD read/write throughput increased by about 3 times.

Keywords: ZNS SSD, RocksDB, ZenFS, Compaction, Parallel I/O, Compaction Optimization

Student Number: 2020-26170