Zoned Namespace SSD를 위한 LSM-tree 최적화 기법

이영재○, 정지윤, 신동군
성균관대학교 정보통신대학
yjlee4154@gmail.com, {wjdwldbs1, dongkun}@skku.edu

Optimizing LSM-tree for Zoned Namespace SSD

Youngjae Lee○, Jeeyoon Jung, Dongkun Shin
College of Information and Communication Engineering, Sungkyunkwan University

요 약

Log-structured Merge tree(LSM-tree)는 높은 쓰기 성능을위해 최적화된 인덱스 자료구조로 key-value 데이터베이스 시스템에서 널리 사용되고 있다. LSM-tree는 sequential write의 특성으로 최근 NVMe에 제안된 Zoned Namespace 인터페이스를 활용하기 적합한 워크로드이다. 하지만 기존의 LSM-tree key-value store를 그대로 활용하는 경우 invalid data로 인한 fragmentation 때문에 space amplification이 증가하여 SSD의 용량을 충분히 활용하기 어렵다. 따라서 본 연구에서는 Zoned Namespace 인터페이스를 고려하여 LSM-tree의 compaction 알고리즘 및 zone 할당 기법을 제안하고 이를 적용하여 space amplification의 감소 효과를 분석한다.

1. 서 론

Log-structured Merge-tree(LSM-tree)[1]는 높은 쓰기 성능을 위해 최적화된 인덱스 자료구조로 LevelDB[2], RocksDB[3] 등의 많은 key-value 데이터베이스 시스템에서 사용되고 있다. LSM-tree 기반 key-value store는 새로운 데이터의 추가 및 기존 데이터의 update를 모두 새로운 record로 SSTable 단위로 batch로 기록하여 sequential write을 사용하여 쓰기 성능을 높인다.

LSM-tree의 이런 특성은 최근 NVMe에 제안된 Zoned Namespace 인터페이스에 적합하며 ZenFS와 같이 ZNS SSD를 활용하기 위한 연구가 진행되고 있다[4]. 하지만 zone의 크기가 SSTable의 크기보다 클 경우에, 여러 SSTable들이 하나의 zone에 배치되고, 그 중 일부 SSTable이 compaction 작업으로 invalid하게 되면 무효화된 공간으로 인한 space amplification 문제가 발생한다. Space utilization은 최근 SSD 환경에서 주요 최적화 대상으로 RocksDB의 space amplification을 줄이기 위한 최적화가 진행되었다[5].

본 연구에서는 Zoned Namespace SSD의 특성을 고려한 LSM-tree인 ZoneLSM을 제안하여 zone 할당 및 compaction 대상 선택 등에서 zone을 고려한 기법을 적용하고 이를 구현하여 효과를 분석한다.

2. 배경 이론

Zoned Namespace 인터페이스는 최근 제안된 NVMe 인터페이스로 기존의 블록 인터페이스와 달리 NAND 플래시 메모리의 특성과 비슷한 호스트 인터페이스를 제공하여 SSD 내부 동작을 단순화시키고 predictable한 성능을 낼 수 있다. 기존의 Open-channel SSD의 경우 NAND의 수명과 같은 하드웨어의 특성들을 host에서 모두 관리해야 하던 것과 달리 ZNS에서는 보다 추상화된 인터페이스를 제공하여 sequential write 라는 제약 조건만을 host에 요구한다.

ZNS는 기존과 같은 LBA (Logical Block Address) 방식의 주소공간을 사용하나 이를 고정 크기의 zone으로 나눈다. 각 zone 내에서는 LBA가 증가하는 순서로 쓰기가 수행되어야 하고 덮어쓰기가 불가능하며 zone을 재사용하기 위해서는 host에서 별도의 reset 명령어를 사용해야 한다.

3. 선행 연구

ZenFS[4]는 RocksDB를 위한 파일시스템으로 Host managed SMR 드라이브나 ZNS SSD와 같은 zoned block storage에서 커널 파일시스템 없이 RocksDB를 사용할 수 있도록 한다. RocksDB의 WAL(Write-ahead Log) 및 SSTable은 append-only로 쓰이기 때문에 ZNS SSD의 특성에 쉽게 적용할 수 있으며 zone을 연속된 블록인 extent 단위로 할당하여 파일의 데이터 블록으로 사용한다. 또한 파일시스템의 메타데이를 log-structured 방식으로 전용의 zone에 기록하여 crash recovery 가 가능하도록 한다.

4. ZoneLSM

4.1. Level Isolation

LSM-tree는 오래된 레코드를 제거하고 탐색시간을 줄이기위해 기존의 SSTable을 합쳐 새로운 SSTable로 작성하는 compaction 작업을 지속적으로 수행한다. 낮은 레벨에서는 compaction이 자주 발생하여 SSTable이 자주

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)

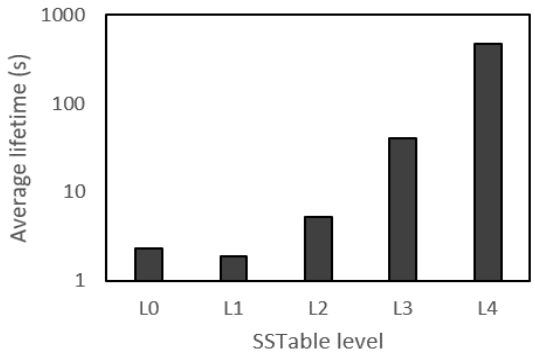
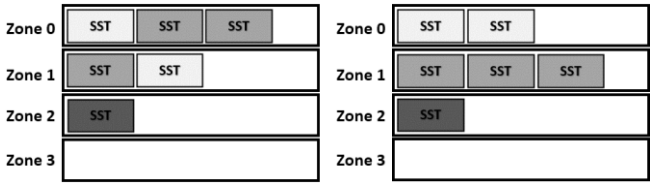


그림 1. SSTable의 수명 분포



(a) ZenFS (b) ZoneLSM

그림 2. ZoneLSM의 SSTable 배치

삭제되고 생성되며 높은 레벨의 SSTable은 상대적으로 compaction이 수행되는 주기가 길어 레벨마다 SSTable의 수명의 차이를 보인다. 그림 1은 random 쓰기 워크로드에서 각 레벨의 SSTable들의 평균수명이다.

ZenFS에서는 RocksDB의 수명 정보를 기반으로 zone 할당을 하지만, 적은 개수의 수명 단계를 사용하여 다른 레벨의 SSTable 들이 동일한 수명으로 분류될 수 있다. 또한 수명 단계 차이가 적은 SSTable이 같은 zone에 배치되는 것을 허용하므로 여러 레벨의 SSTable이 같은 zone에 배치되어 invalid SSTable에 의한 space amplification을 증가시킬 수 있다.

반면에, 우리의 ZoneLSM에서는 레벨 마다 별도의 zone을 할당하여 수명이 비슷한 SSTable이 같은 zone에 위치하도록 하여 space amplification을 줄인다. 그림 2는 기존 방식대비 ZoneLSM에서 SSTable이 zone에 배치되는 방식의 차이를 보여준다. 같은 색의 SSTable은 동일한 레벨의 SSTable을 의미하며 ZoneLSM의 경우 서로 다른 레벨의 SSTable이 동일한 zone에 배치되지 않도록 한다.

4.2. Zone-aware Compaction

Level 기반의 compaction을 사용하는 LSM-tree key-value store에서는 각 레벨마다 존재하는 SSTable의 총량을 점수로 계산하여 가장 높은 레벨을 대상으로 선택한다. 기존 기법들은 대상 레벨 내에서 다양한 기준에 따라 대상 파일을 선택하는데, ZoneLSM은 Invalid space가 가장 많은 zone을 우선 선택한 후에, 해당 zone에 포함된 SSTable을 우선적으로 compaction 함으로써 compaction 과정에서 가능한 많은 zone이 reclaim될 수 있도록 한다.

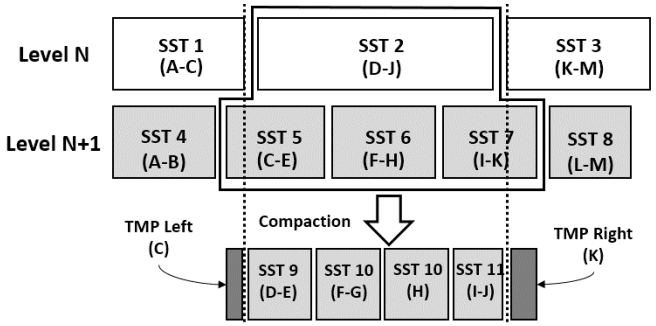


그림 3. 임시 파일 구분

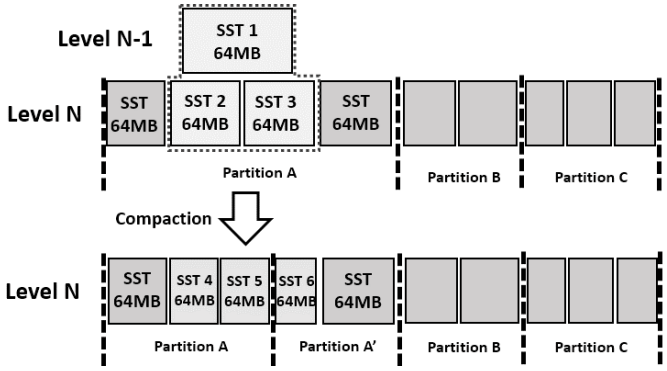


그림 4. Partitioning 예시

4.3. Temporary SSTable 분리

Compaction 과정에서 Level N의 SSTable이 선택되면 해당 SSTable과 key range가 겹치는 Level N+1의 모든 SSTable이 함께 병합되어 새로운 N+1 레벨의 SSTable들이 생성된다. 예를 들어, 그림 3에서 Level N의 2번 SSTable을 compaction 할 때 이와 range가 겹치는 5,6,7의 SSTable이 같이 병합되어 새로운 SSTable이 생성된다. 생성된 SSTable 중 양 끝에 해당하는 SSTable의 경우 Level N의 SST1이나 SST3과 key range가 겹쳐 compaction 작업시에 다시 삭제될 수 있어 매우 짧은 수명을 갖게 된다. 따라서 SST1과 SST3의 범위를 기준으로 양 끝을 TMP left, right과 같이 별도의 SSTable로 분리하여 임시파일로 구분하며 별도 zone에 배치한다.

4.3. Partitioning

Compaction 과정에서 여러 key range가 인접한 SSTable 들이 대상으로 선택되므로 동일한 레벨에서 인접한 key 범위를 갖는 SSTable들이 한 번의 compaction에서 같이 사라지게 된다. 따라서 ZoneLSM에서는 같은 레벨의 SSTable을 key 범위를 기준으로 다른 파티션에 분리하며 독립적인 zone을 할당 받아 SSTable을 기록하여 비슷한 key 범위의 SSTable이 같은 zone에 배치되도록 한다. SSTable은 지속적으로 삭제와 생성이 되며 range 마다 가변적이므로 파티션의 수를 동적으로 유지한다. 파티션 용량이 threshold를 초과하는 경우 두 개의 파티션으로 분할하여 SSTable이 많은 key range에 보다 많은 파티션을 유지한다. 그림 4는 compaction으로 Level N에 SSTable이 증가하면서 파티션이 분할되는 예시로 threshold가 300MB인 경우 1-3 SSTable이

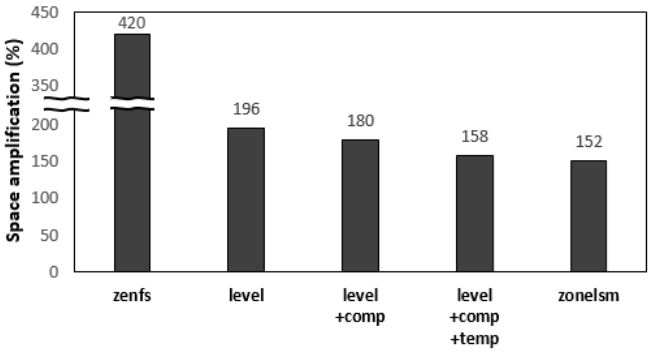


그림 5. ZoneLSM 기법에 따른 space amplification

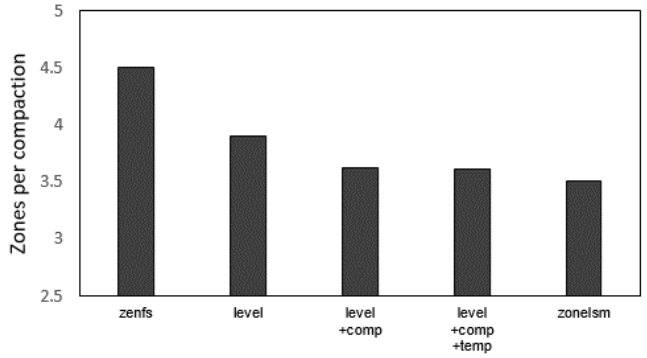


그림 6. 기법에 따른 compaction 당 zone의 분산

compaction으로 4-6 SSTable로 재생성 되며 파티션 A의 용량이 320MB가 되며threshold를 초과하여 두 개의 파티션 A와 A'으로 분리되는 경우이다.

파티션의 크기가 지나치게 큰 경우 key range에 따른 분리효과가 적으며 크기가 작은 경우 compaction이 여러 파티션에 걸쳐 발생하여 zone reclaim 효과가 줄어들 수 있어 적절한 파티션의 크기 설정이 필요하다.

5. 실 험

5.1. 실험 환경

ZoneLSM은 RocksDB와 ZenFS를 수정하여 작성하였으며 ZNS SSD는 QEMU의 NVMe ZNS 에뮬레이션을 활용하였다. Zone의 크기는 512MB를 사용하였다. 워크로드는 db_bench의 fillrandom 벤치마크를 100,000,000 ops 만큼 수행하였고 SSTable은 64MB의 크기로 설정하였다.

5.2. 실험 결과

그림 5는 기법에 따른 space amplification의 차이를 보여준다. space amplification은 다음과 같이 계산한다.

$$\text{space amplification} = \frac{\text{used zone capacity}}{\text{valid data capacity}}$$

기존 ZenFS의 경우 수명이 다른 여러 레벨의 SSTable이 동일한 zone에 배치되어 fragmentation이 많이 발생하고 따라서 매우 높은 space amplification을 보여준다. Level Isolation 기법(level)을 적용한 경우 space amplification이 196%로 감소한 것을 볼 수 있다. 추가적으로 Zone-aware

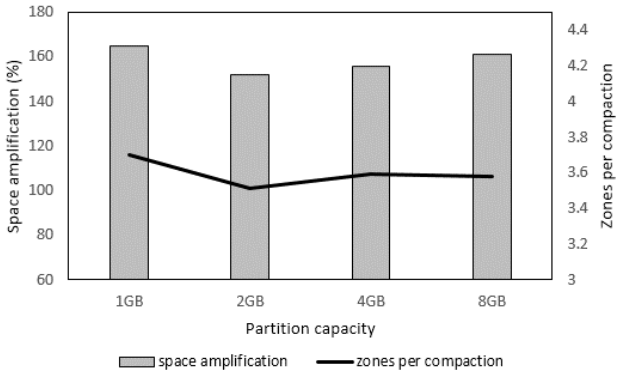


그림 7. 파티션 크기에 따른 효과 비교

compaction(comp) 기법과 임시 SSTable 구분 기법(temp)을 적용한 경우 space amplification이 추가적으로 감소하였으며 파티셔닝을 포함한 모든 기법을 적용한 ZoneLSM은 152%로 기존 ZenFS 대비 36% 정도의 낮은 space amplification을 보여준다.

그림 6는 compaction의 입력 SSTable 들이 몇 개의 zone에 분산되어 있는지 보여준다. ZoneLSM의 기법을 적용한 경우 동일한 zone에 포함된 SSTable을 위주로 compaction이 수행되어 compaction 과정에서 보다 많은 zone을 reclaim 할 수 있다.

그림 7은 파티션의 크기를 조절해가며 벤치마크를 수행했을 때 효과를 비교한 것으로 파티션의 2GB 정도의 파티션에서 가장 낮은 space amplification을 보여주었다.

6. 결론 및 향후 계획

본 연구에서는 ZNS SSD 환경을 고려한 LSM-tree key-value store의 zone 할당 및 compaction 기법을 적용하여 space amplification을 줄이는 효과를 보여주었다. ZoneLSM은 기존 ZenFS 대비 36% 정도의 낮은 space amplification을 보여 SSD를 보다 효율적으로 사용할 수 있다. 추후 적은 오버헤드로 수행할 수 있는 garbage collection 기법 등을 적용하면 보다 낮은 space amplification을 달성할 수 있을 것이다.

참고 문헌

[1] Patrick O'Neil, et al. The log-structured merge-tree (LSM-tree). Acta Informatica. 1996

[2] Sanjay Ghemawat and Jeff Dean. LevelDB. http://code.google.com/p/leveldb, 2011.

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

[4] Bjørling, M, et al. "ZNS: Avoiding the Block Interface Tax for Flash-based SSDs." In Proceedings of the 2021 USENIX Annual Technical Conference (ATC'21). 2021.

[5] Cao, Zhichao, et al. "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook." In 18th USENIX Conference on File and Storage Technologies (FAST'20). 2020.