단국대학교 DANKOOK UNIVERSITY

RocksDB 성능 비교: Column Family 데이터 분리와 Compaction 기법

과 목 명: 오픈소스SW분석(빅데이터) 1분반
발 표 일: 2025년 05월 08일 (목)
팀　　원: 구선주 (32220207), 임수연 (32193772), 최예림 (32224684)

Contents

RocksDB 성능 비교: Column Family 데이터 분리와 Compaction 기법

01

이전 실험

- 가설 1
- 가설 2

가설 1

가설 1

WAL 설정과 Compaction 전략 간의 상호작용은 WAF에 유의미한 영향을 미칠 수 있다.

Case	Compaction	workload	WAL
Leveled	Leveled Compaction	fillrandom, overwrite	perf, stable
Universal	Universal Compaction	fillrandom, overwrite	perf, stable

perf: disable_wal=**true**, sync=false, use_direct_io_for_flush_and_compaction=**true**

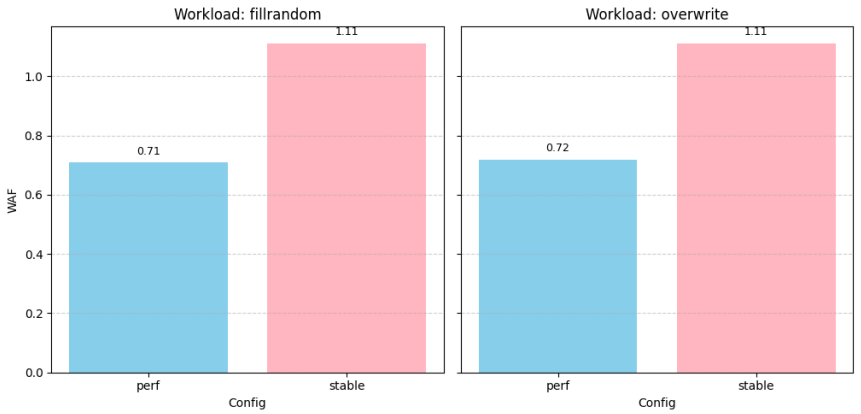
stable: disable_wal=**false**, sync=false, use_direct_io_for_flush_and_compaction=**false**

가설 1

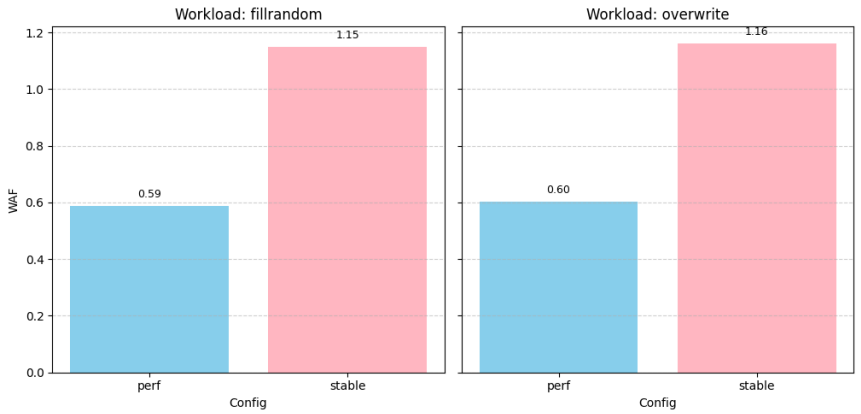
가설 1

WAL 설정과 Compaction 전략 간의 상호작용은 WAF에 유의미한 영향을 미칠 수 있다.

Leveled



Universal



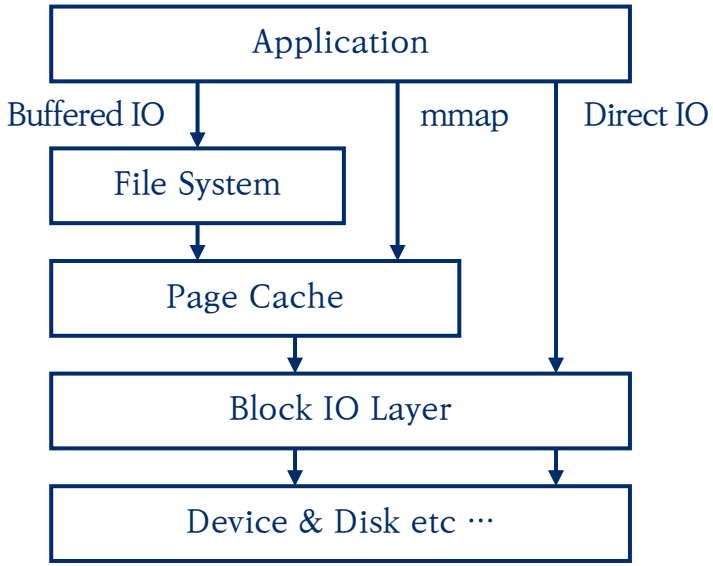
➡ performance의 경우 stable보다 WAF 감소

가설 1

가설 1 추가 실험

use_direct_reads: 읽기 작업 시 O_DIRECT 모드로 파일을 열어 운영체제 캐시를 우회

use_direct_io_for_flush_and_compaction: flush 및 compaction 작업 시 O_DIRECT 사용하여 디스크에 직접 입력



Direct IO

- Buffered IO: 데이터를 두 번 복사하여 페이지 캐시 사용
- Page Cache를 우회하여 메모리에 직접 입출력
- 복사 횟수를 줄이고, 애플리케이션이 캐시를 직접 관리
- 메모리 예측 가능성 증가, 중복 캐시 제거, I/O 성능 개선

가설 1

가설 1 추가 실험

OS	UBuntu 20.04.6 LTS (64bit)
CPU	16 * Intel(R) Core(TM) i7-10700F CPU @ 2.90GHz
RAM	31.2GiB
SSD	1.0TB
RocksDB	Version 10.2.0

가설 1

가설 1 추가 실험

$$\text{Write Amplification Factor} = \frac{\text{rocksdb.flush.write.bytes} + \text{rocksdb.compact.write.bytes}}{\text{rocksdb.bytes.written}}$$

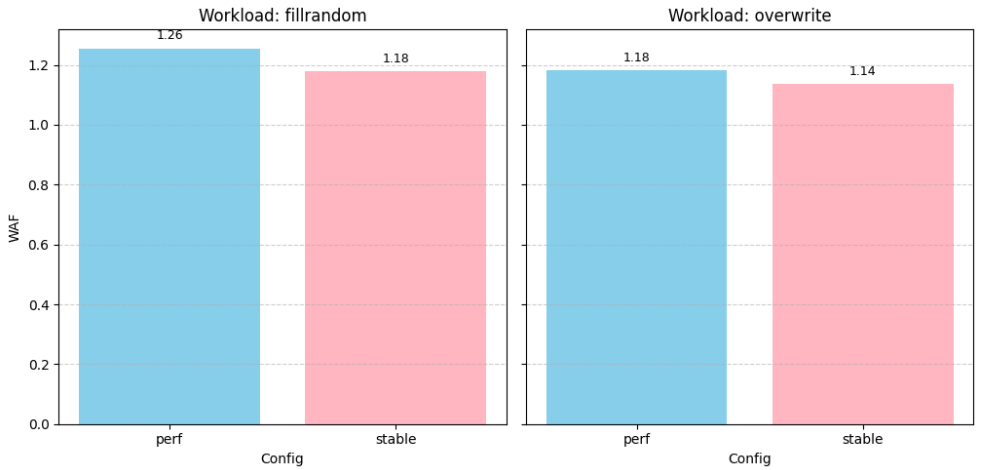
- rocksdb.flush.write.bytes: 메모리 내 데이터가 디스크에 지속되는 flush 작업 중 쓰인 바이트 수
- rocksdb.compact.write.bytes: compaction 중 쓰인 바이트 수
- rocksdb.bytes.written: rocksdb에 쓰여진 총 바이트 수

가설 1

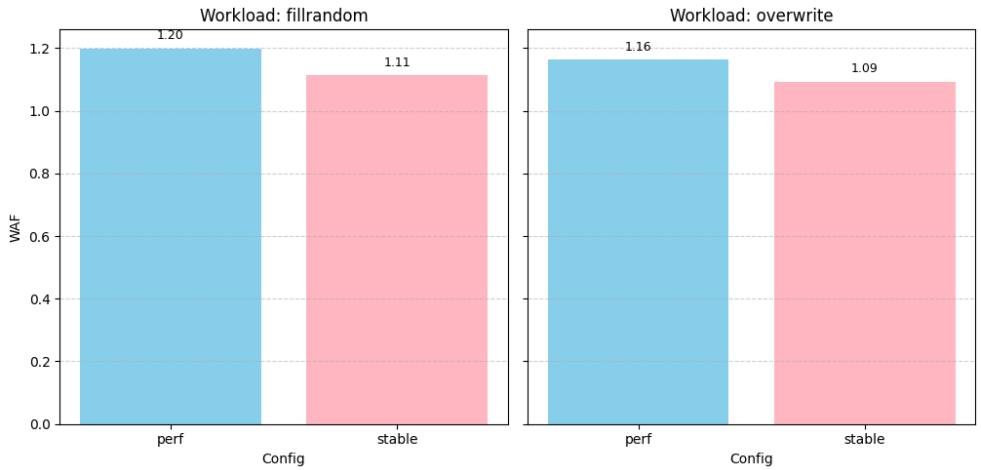
가설 1 추가 실험

Configs Option	내용
perf	disable_wal=true, sync=false, use_direct_io_for_flush_and_compaction=false
stable	disable_wal=false, sync=false, use_direct_io_for_flush_and_compaction=false

Leveled



Universal

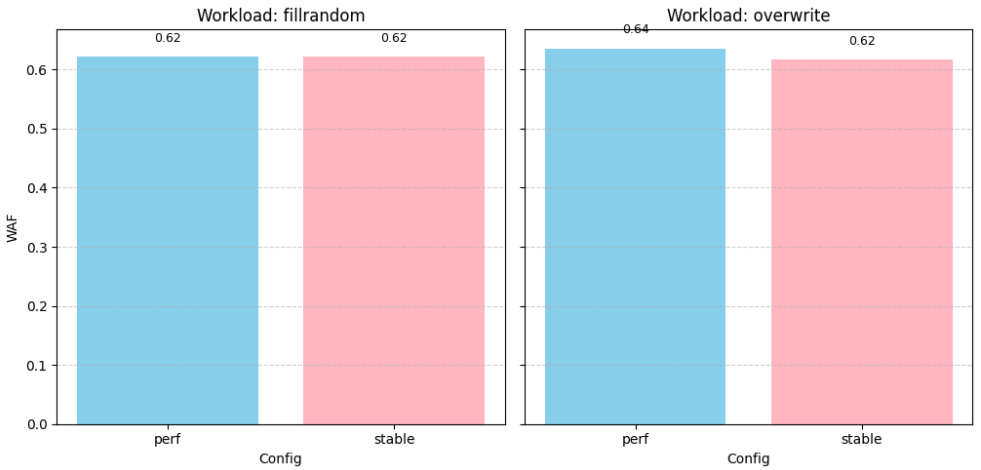


가설 1

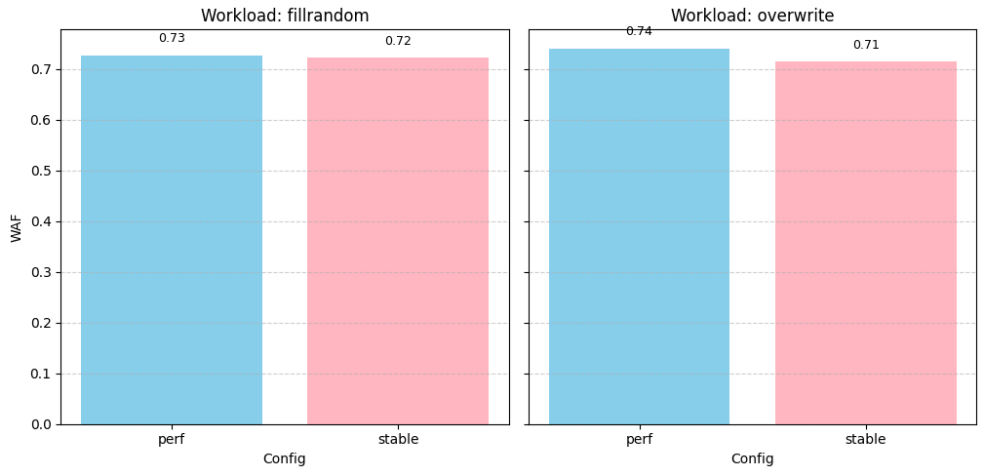
가설 1 추가 실험

Configs Option	내용
perf	disable_wal=true, sync=false, use_direct_io_for_flush_and_compaction=true
stable	disable_wal=false, sync=false, use_direct_io_for_flush_and_compaction=true

Leveled



Universal

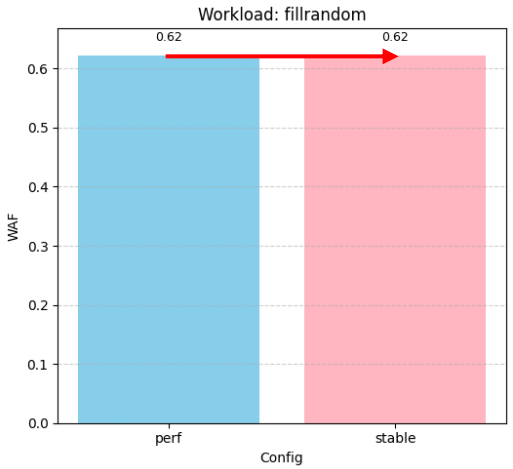


가설 1

가설 1 추가 실험

Leveled Compaction

use_direct_io_for_flush_and_compaction=**false**



Leveled Compaction

use_direct_io_for_flush_and_compaction=**true**

WAF: Performance 〉 Stable

Direct IO: true인 경우 WAF 감소

WAL 또는 다른 요인으로 인해 WAF 차이 발생

Direct IO로 flush/compaction 패턴이 유사

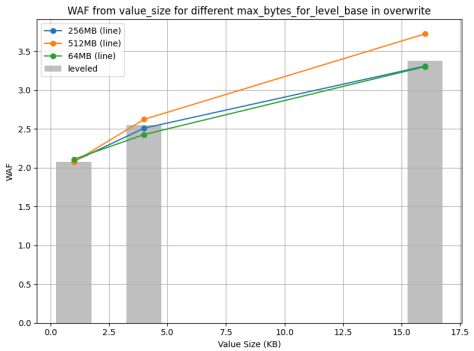
가설 2

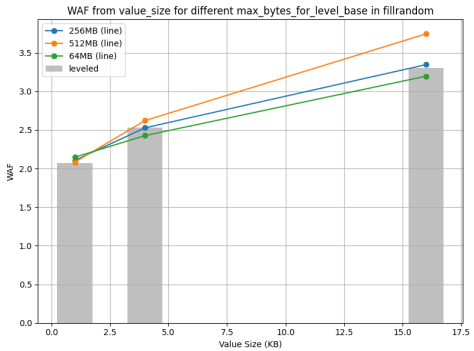
가설 2

Update 및 Delete가 빈번한 워크로드에서 Leveled Compaction의 compaction 빈도를 최적화할 경우,

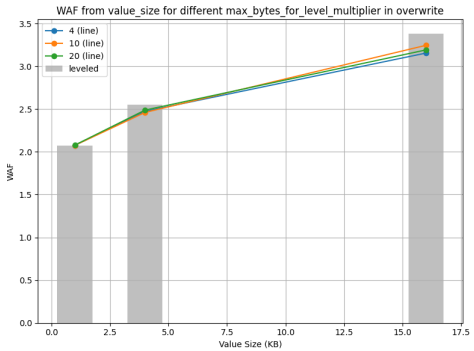
Universal Compaction과 비슷한 수준의 WAF 성능을 달성할 수 있다.

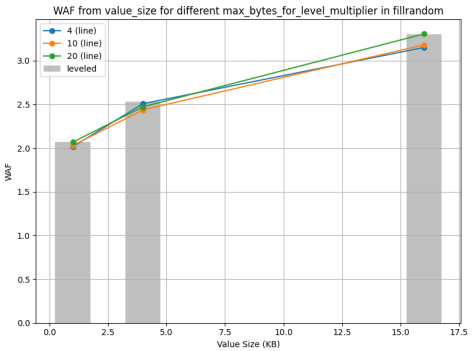
max_bytes_for_level_base





max_bytes_for_level_multiplier

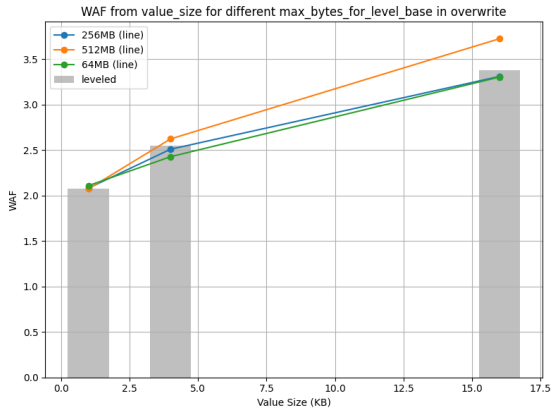


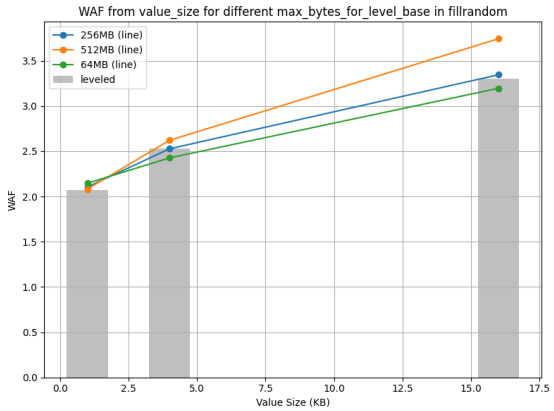


➡ max_bytes_for_level_base와 max_bytes_for_multiplier가 낮으면 WAF가 낮다

가설 2

max_bytes_for_level_base

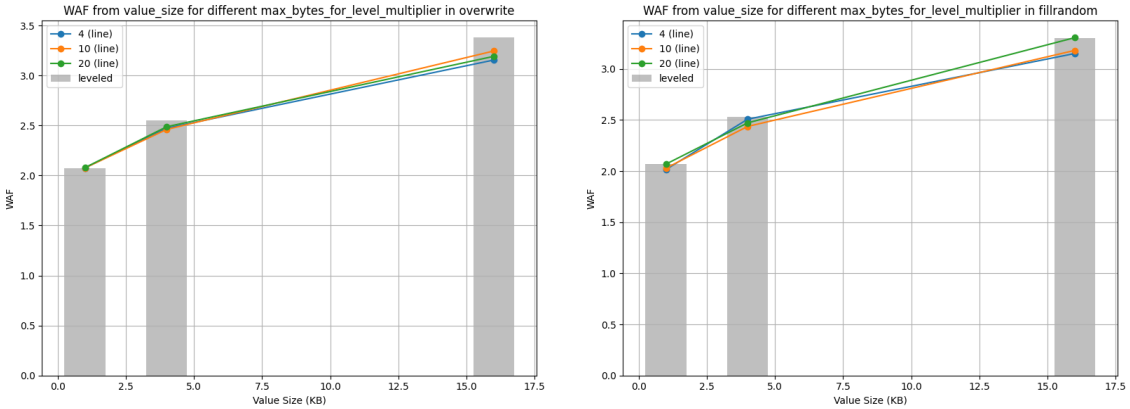




- max_bytes_for_level_base: 레벨 1의 최대 데이터 크기
- L1의 최대 크기가 증가해 compaction이 지연
- L0에 파일이 누적되다가 대량의 compaction이 한 번에 발생
- 특정 시점에 한 번에 많은 Compaction이 더 많은 쓰기 유발

가설 2

max_bytes_for_level_multiplier



- max_byes_for_level_multiplier: 다음 레벨의 용량 증가 비율
- 각 레벨이 커지며 compaction 빈도는 줄어든다
- 한 번에 처리하는 데이터 양이 증가

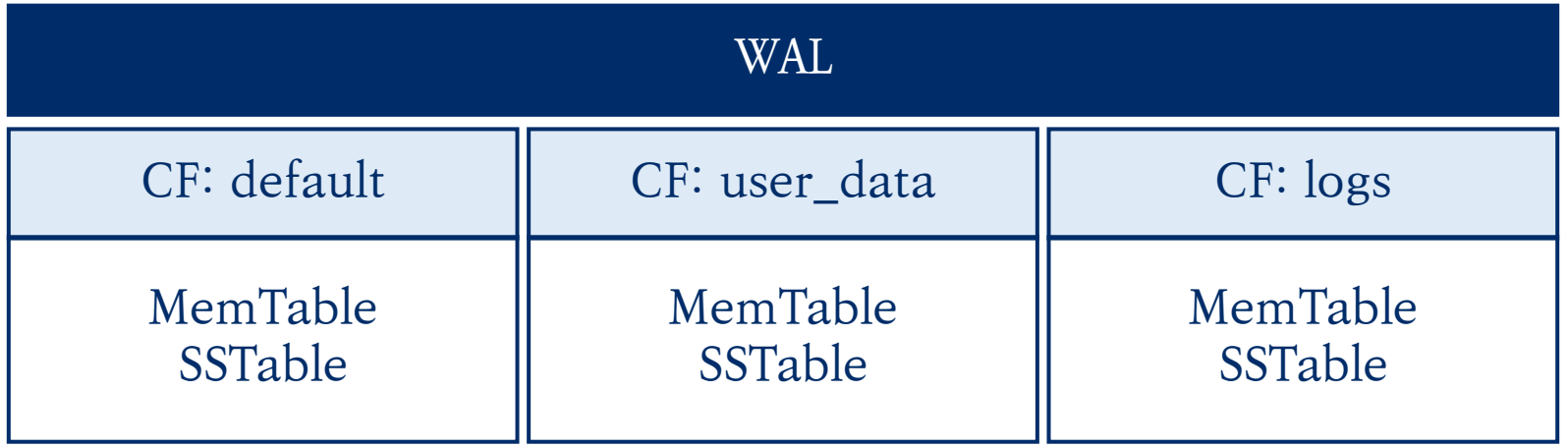
02

서론

- 배경 지식
- 연구 배경

배경 지식

Column Family



- RocksDB에서 하나의 DB 내에서 데이터를 논리적으로 key-value 분리
- 압축 설정, 블룸 필터, 캐시 크기 등의 독립적인 설정이 가능
- 서로 다른 유형의 데이터를 하나의 DB에서 효율적으로 관리 가능

배경 지식

Hot Data와 Cold Data

	Hot Data	Cold Data
접근 빈도	자주 접근됨	거의 접근되지 않음
저장 매체	고성능 저장소 사용 (SSD, 메모리 DB)	비용 효율적인 저장소 사용 (HDD, 테이프, 클라우드 아카이브)
조회 속도	빠른 조회가 필요	느린 조회도 허용
사용 사례	실시간 트랜잭션, 활성 사용자 세션, 실시간 분석	역사 기록, 백업, 오래된 파일 보관
비용	고성능 요구로 인해 일반적으로 비용 높음	느린 속도 허용으로 일반적으로 비용 낮음
데이터 관리	속도 및 효율 최적화가 중요	장기 저장 및 비용 효율성에 초점

연구 배경

Leveled Compaction vs Universal Compaction

Leveled Compaction	Universal Compaction
WAF가 높음	WAF가 낮음
Disk I/O 소모가 큼	디스크 공간 비효율
쓰기-읽기 균형이 좋음	RAF가 높을 수 있음
↓	↓
Hot Data	Cold Data

03

본론

- 가설 설정
- 실험 설계
- 실험 결과

가설 설정

가설 설정

Column Family 별로 Hot, Cold Data를 저장하고,

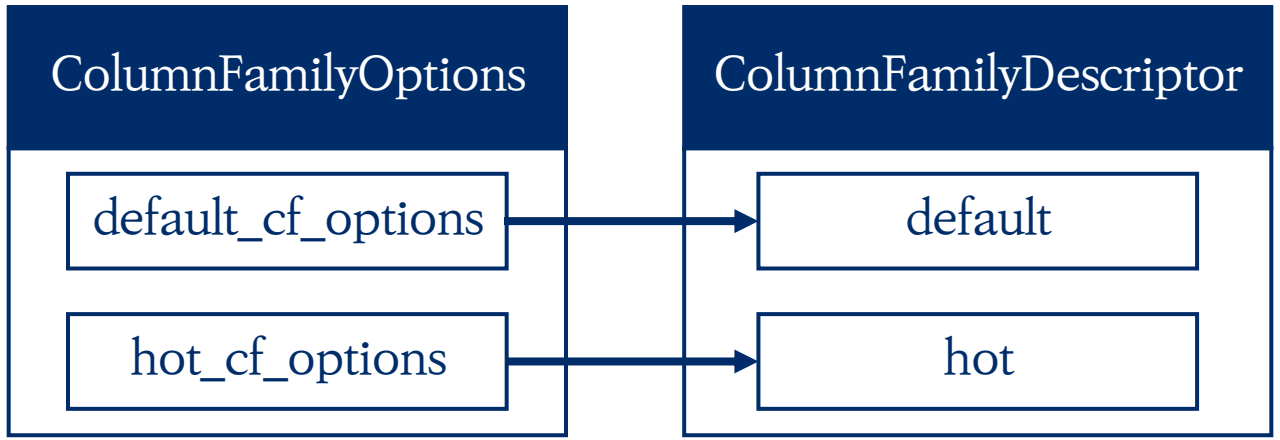
Universal Compaction과 Leveled Compaction을 나누어 사용하면 DB 성능이 더 좋아질 것이다.

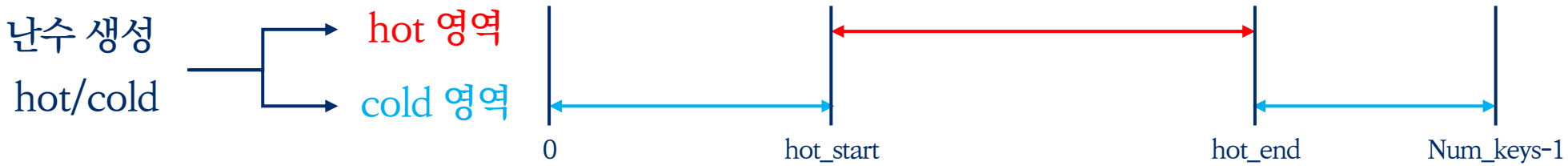
Column	Data	Compaction	Expectation
Hot Column	자주 쓰이고 읽히는 데이터	Leveled Compaction	Read Latency 향상
Cold Column	저장 위주의 데이터	Universal Compaction	Write 효율 향상

- RocksDB에서 Column Family별로 서로 다른 Compaction 방식 설정 가능
- Column Family에 저장된 데이터에 적합한 Compaction 방식 적용으로 성능 향상 기대

실험 설계

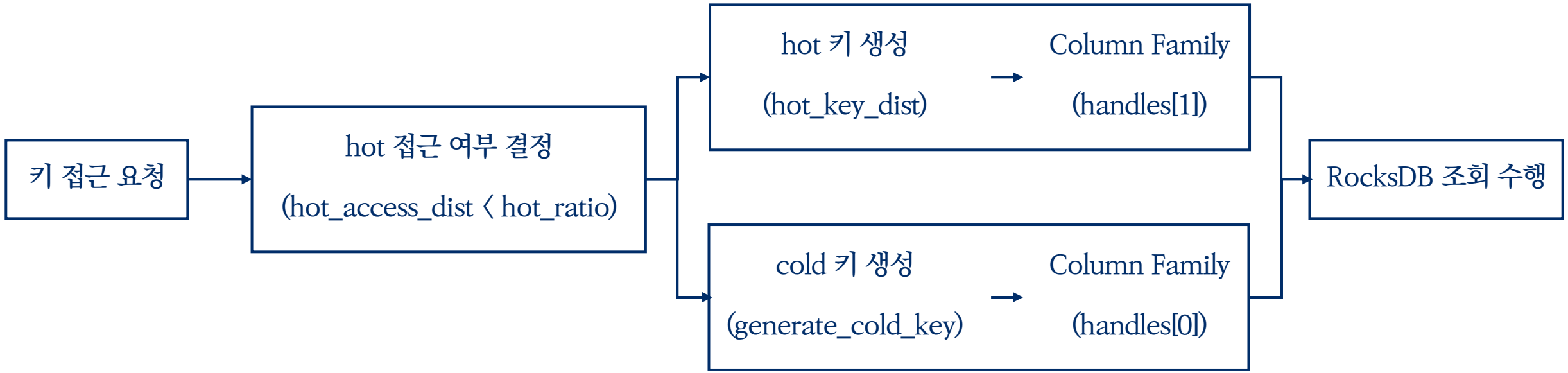
Bench mark (Write)





실험 설계

Bench mark (Read)



실험 설계

Bench mark

〈DB 경로〉 〈총 키 수〉 〈핫 범위 시작〉 〈핫 범위 끝〉 〈value 크기〉 〈핫 접근 비율 (0~100)〉

〈default 컬럼 compaction〉 〈hot 컬럼 compaction〉

- NUM_KEYS: 1,000,000
- HOT START: 0
- HOT END: 199999
- VALUE SIZE: 16KB
- HOT RATIO: 70

실험 설계

실험 계획

case	Hot Column	Cold Column
0	Leveled Compaction	Leveled Compaction
1	Universal Compaction	Universal Compaction
2	Leveled Compaction	Universal Compaction
3	Universal Compaction	Leveled Compaction

각 case를 3번씩 실행하여 평균 값으로 비교

실험 설계

실험 계획

$$\text{Write Amplification Factor} = \frac{\text{rocksdb.flush.write.bytes} + \text{rocksdb.compact.write.bytes}}{\text{rocksdb.bytes.written}}$$

- rocksdb.flush.write.bytes: 메모리 내 데이터가 디스크에 지속적으로 플러시 작업 중 쓰인 바이트 수
- rocksdb.compact.write.bytes: compaction 작업 중 읽은 바이트 수
- rocksdb.bytes.written: rocksdb에 쓰여진 총 바이트 수

실험 설계

실험 계획

$$\text{Read Amplification Factor} = \frac{\text{rocksdb.number.keys.read} * \text{value\_size}}{\text{rocksdb.bytes.read}}$$

- rocksdb.number.keys.read: 읽은 key의 수
- rocksdb.bytes.read: rocksdb에서 읽혀진 총 바이트 수

실험 설계

실험 계획

$$\text{Block Cache Hit Ratio} = \frac{\text{rocksdb.block.cache.hit}}{\text{rocksdb.block.cache.hit} + \text{rocksdb.block.cache.miss}}$$

- rocksdb.block.cache.hit: 읽기 요청이 블록 캐시 에서 필요한 데이터를 찾아 disk io를 피한 횟수
- rocksdb.block.cache.miss: 데이터를 찾지 못해 disk io가 필요했던 횟수

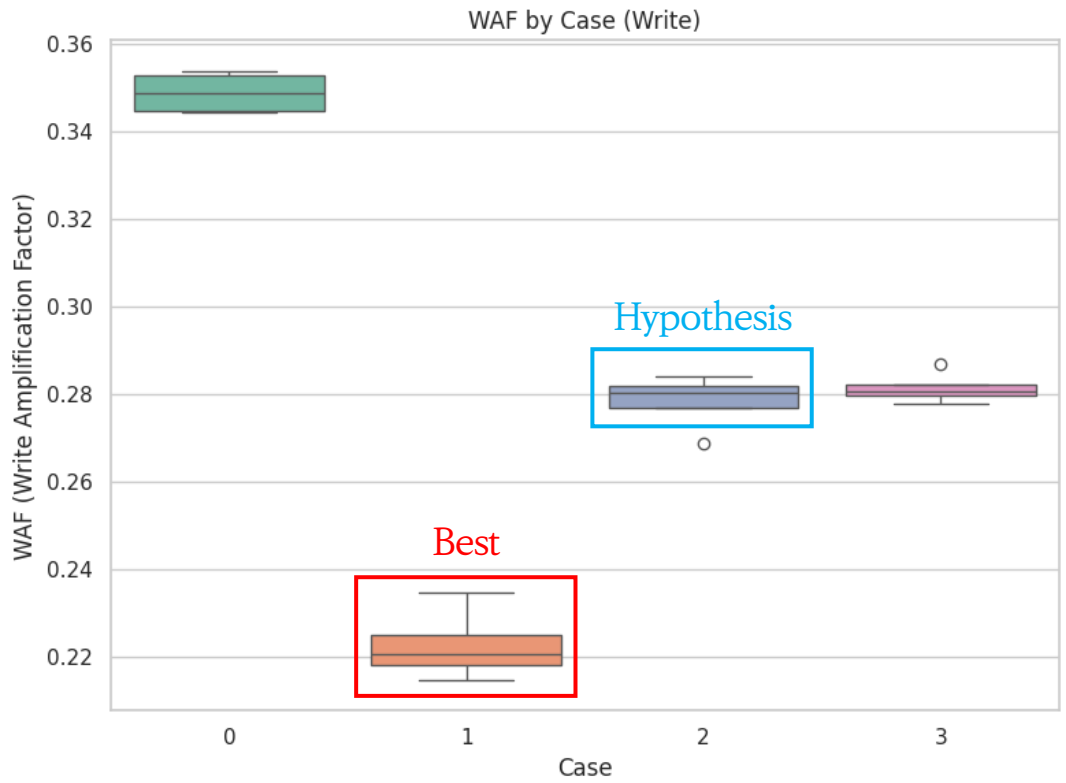
실험 설계

실험 환경

OS	UBuntu 20.04.6 LTS (64bit)
CPU	16 * Intel(R) Core(TM) i7-10700F CPU @ 2.90GHz
RAM	31.2GiB
SSD	1.0TB
RocksDB	Version 10.2.0

실험 결과

Write 실험 결과



Universal Compaction

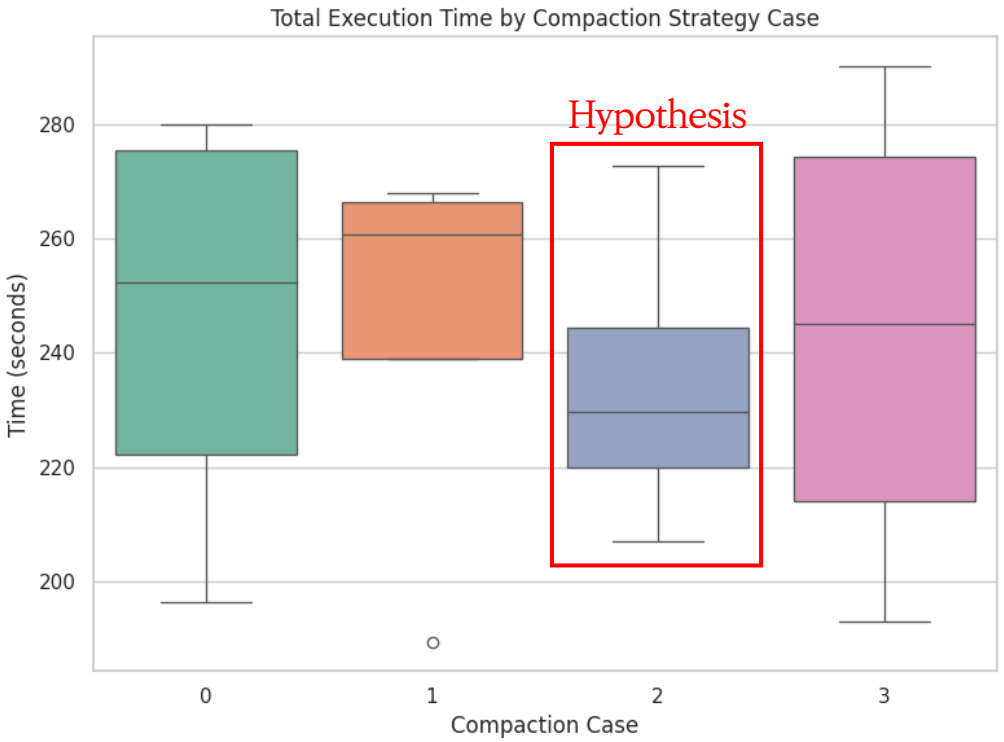
WAF가 낮다

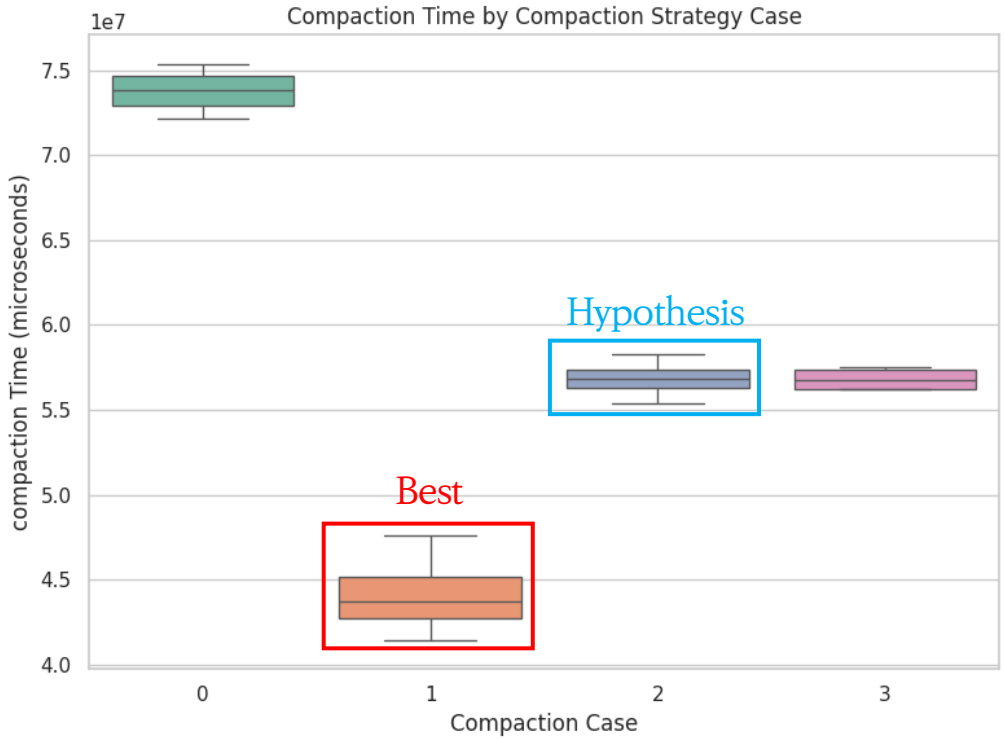
Universal Compaction을 사용하면 WAF가 감소

Trade-Off가 존재할 것으로 예상

실험 결과

Write 실험 결과

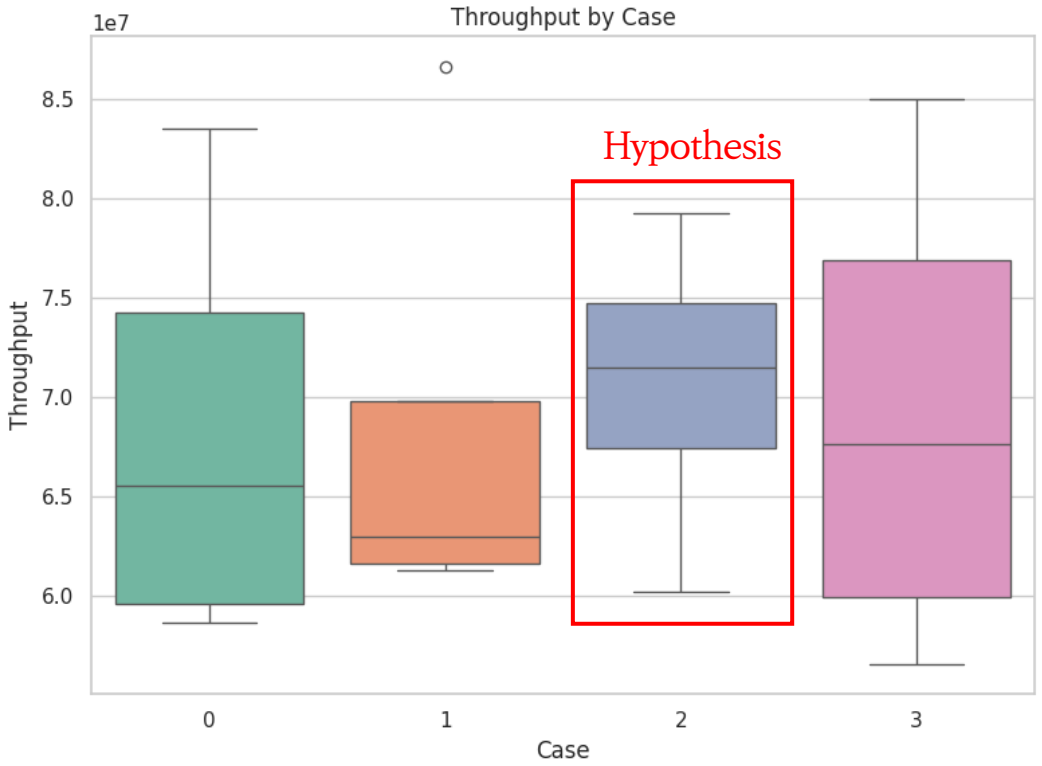




rocksdb.compaction.total.time.cpu_micros: rocksdb의 compaction 작업에 사용된 총 CPU 시간

실험 결과

Write 실험 결과



WAF: user가 쓴 데이터 대비 실제 디스크에 쓴 데이터

Throughput: 단위 시간당 처리 가능한 쓰기 요청

일반적인 경우

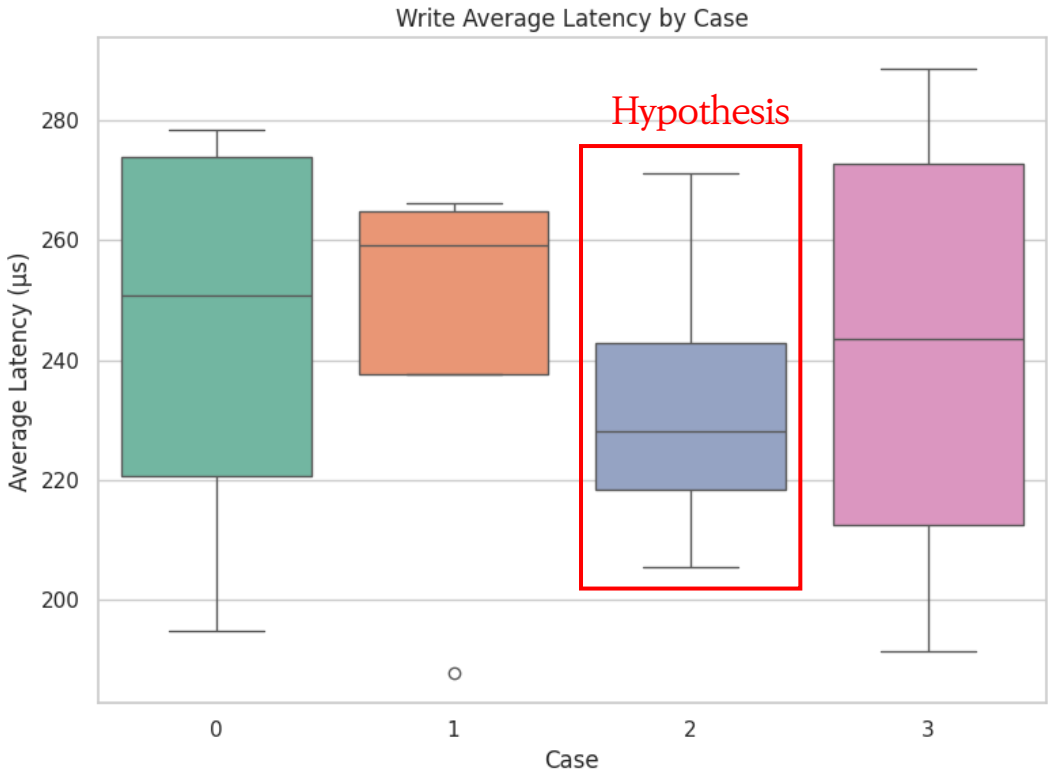
- WAF가 높으면 Throughput이 낮음
- WAF가 낮으면 Throughput이 높음

Case 1의 경우

- WAF가 가장 낮음
- Case 2와 Case 3에 비해 Throughput이 낮음

실험 결과

Write 실험 결과



WAF: user가 쓴 데이터 대비 실제 디스크에 쓴 데이터

Write Latency: 단일 쓰기 요청에 대한 응답 시간

일반적인 경우

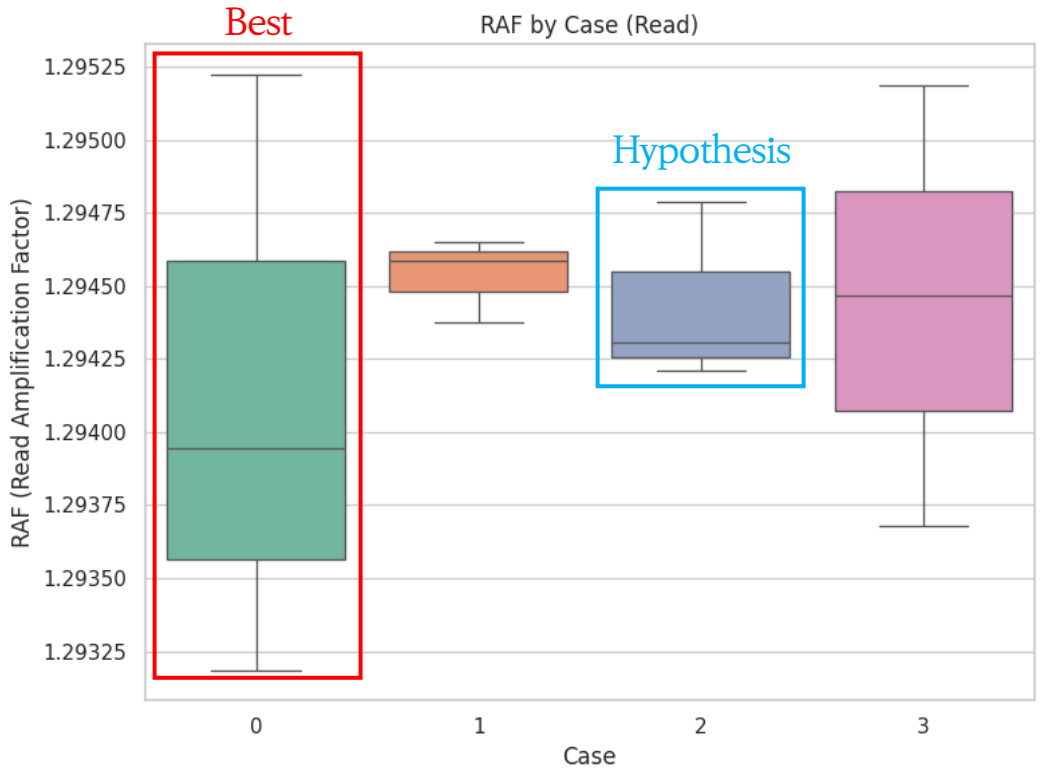
- WAF가 높으면 Latency가 높음
- WAF가 낮으면 Latency가 낮음

Case 1의 경우

- WAF가 가장 낮음
- Case 2와 Case 3에 비해 Latency가 높음

실험 결과

Read 실험 결과



Universal Compaction

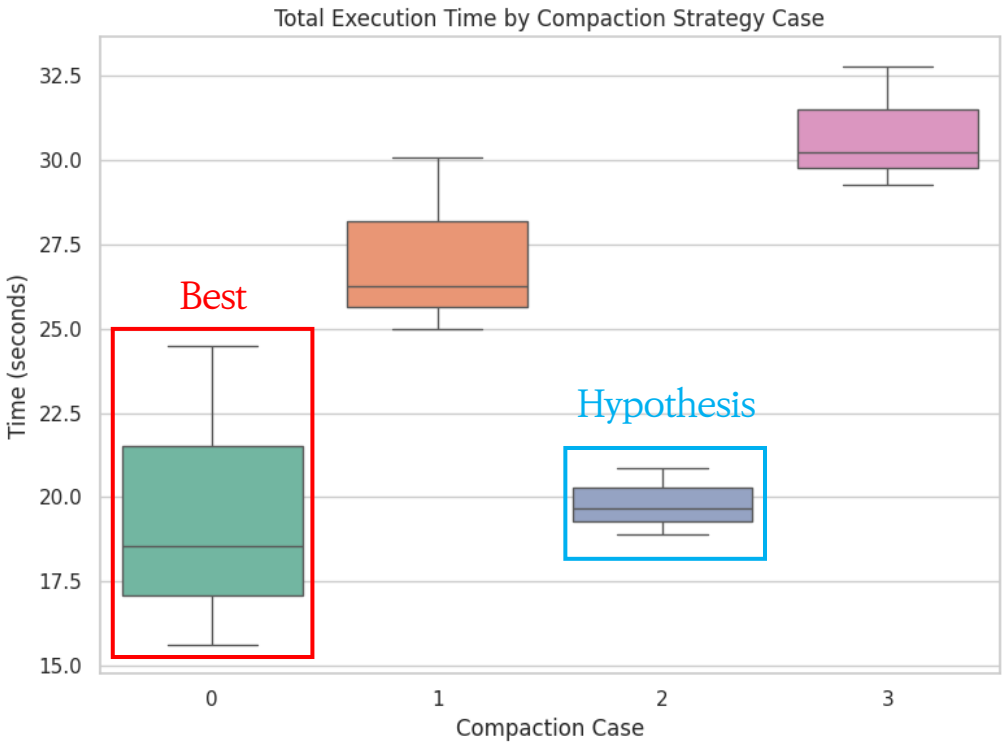
RAF가 높다

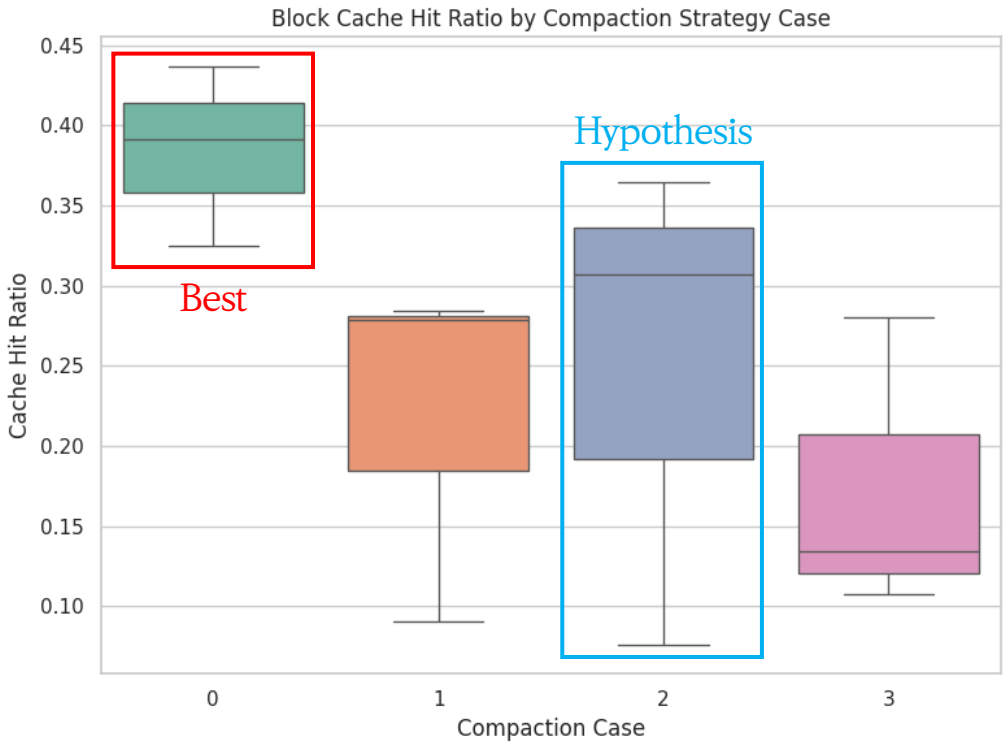
Case 0의 RAF가 가장 낮음

Case1의 RAF가 가장 높음

실험 결과

Read 실험 결과





➡ Universal Compaction을 사용하면 Read 성능이 Leveled Compaction보다 낮다

04

결론

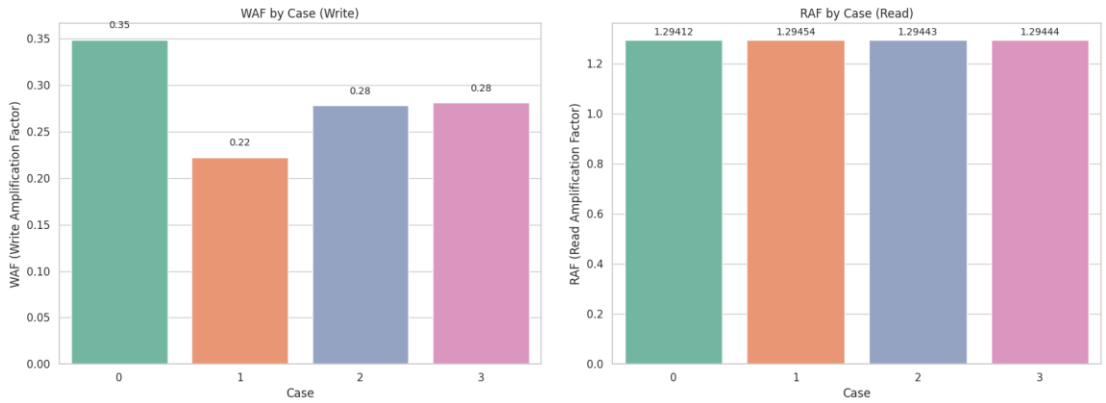
- 실험 결과 요약
- 고찰 및 향후 연구

실험 결과 요약

실험 결과 요약

Column Family 별로 Hot, Cold Data를 저장하고,

Universal Compaction과 Leveled Compaction을 나누어 사용하면 DB 성능이 더 좋아질 것이다.



Write: Case 1 〉 Case2 〉 Case 3 〉 Case 0

Read: Case 0 〉 Case 3 〉 Case 2 〉 Case 1

Case 0, Case1보다 균형 잡힌 성능

Read/Write 성능을 더 높일 수 있는 가능성

고찰 및 향후 연구

고찰

변인 통제

- direct_io가 WAL 설정보다 WAF에 더 큰 영향을 미친다
- 통제변인을 더 철저히 관리해야 할 필요성을 높임

추가 실험

- 각 case의 결과들이 큰 차이를 보이지 않는다
- hot_ratio나 hot_key의 범위를 더 극단적으로 조정

고찰 및 향후 연구

향후 연구

독립적인 설정

- 압축 설정, 블룸 필터, 캐시 크기 등
- Workload 특성에 맞는 최적화 가능

- Column Family에는 Compaction 방법 외 다른 옵션들도 존재
- hot-cold 데이터 워크로드 특성에 맞게 다른 옵션들을 조정하여 최적화

05

참고 문헌

참고문헌

[1] Column Families. (2021). https://github.com/facebook/rocksdb/wiki/Column-Families.

[2] Differences between Hot Data and Cold Data - System Design. (2024). https://www.geeksforgeeks.org/differences-between-hot-data-and-cold-data-system-design/.

[3] Direct-IO.md. (2017). https://github.com/EighteenZi/rocksdb_wiki/blob/master/Direct-IO.md.

[4] Leveled Compaction. (2023). https://github.com/facebook/rocksdb/wiki/Leveled-Compaction.

[5] RocksDB in Microsoft Bing. (2021). https://blogs.bing.com/Engineering-Blog/october-2021/RocksDB-in-Microsoft-Bing.

[6] statistics.h. (2022). https://github.com/facebook/rocksdb/blob/main/include/rocksdb/statistics.h.

[7] Universal Compaction. (2023). https://github.com/facebook/rocksdb/wiki/universal-compaction.

Thank you

Thank you for listening to my presentation