단국대학교 DANKOOK UNIVERSITY

Column Family: 압축 알고리즘을 사용한 성능 최적화

과 목 명: 오픈소스SW분석(빅데이터) 1분반
발 표 일: 2025년 05월 22일 (목)
팀　　원: 구선주 (32220207), 임수연 (32193772), 최예림 (32224684)

Contents

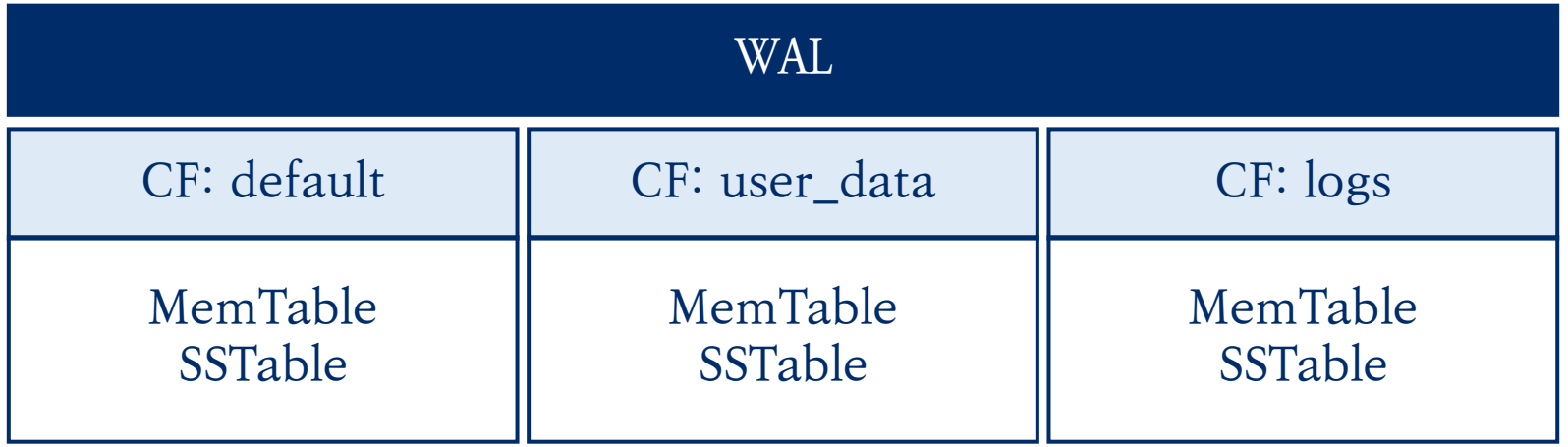
01

이전 실험

- 가설3

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

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

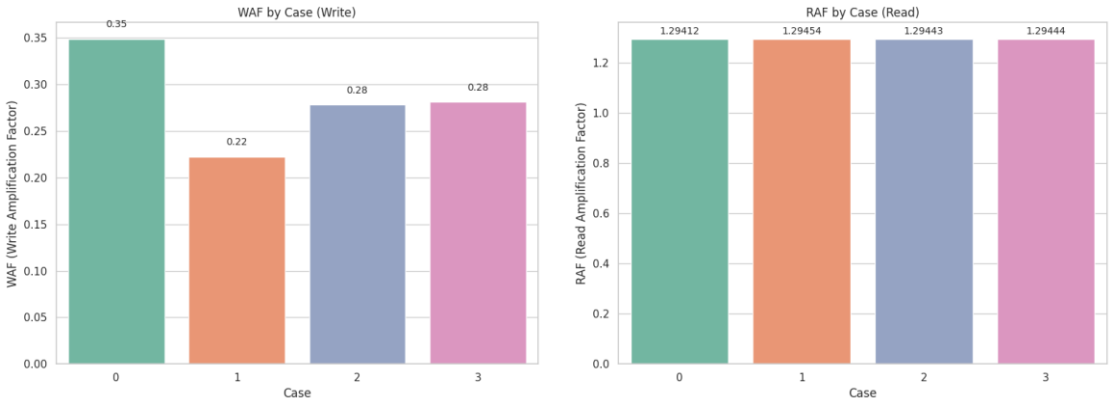
- RocksDB에서 Column Family별로 서로 다른 Compaction 방식 설정 가능
- Column Family에 저장된 데이터에 적합한 Compaction 방식 적용으로 성능 향상 기대

실험 결과

가설3 실험 결과 요약

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



Write: Case 1 〉 **Case 2** 〉 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

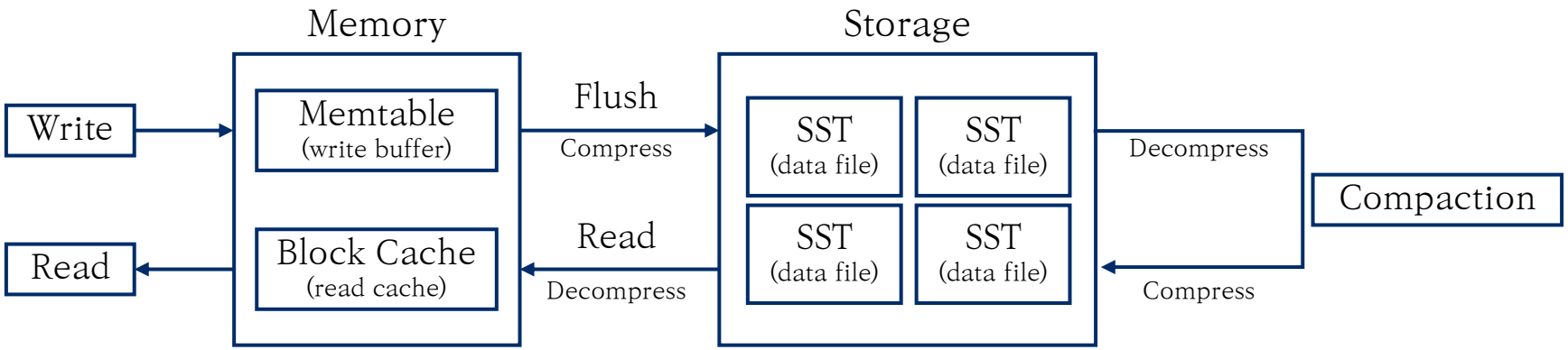
02

서론

- 배경 지식
- 연구 배경

배경 지식

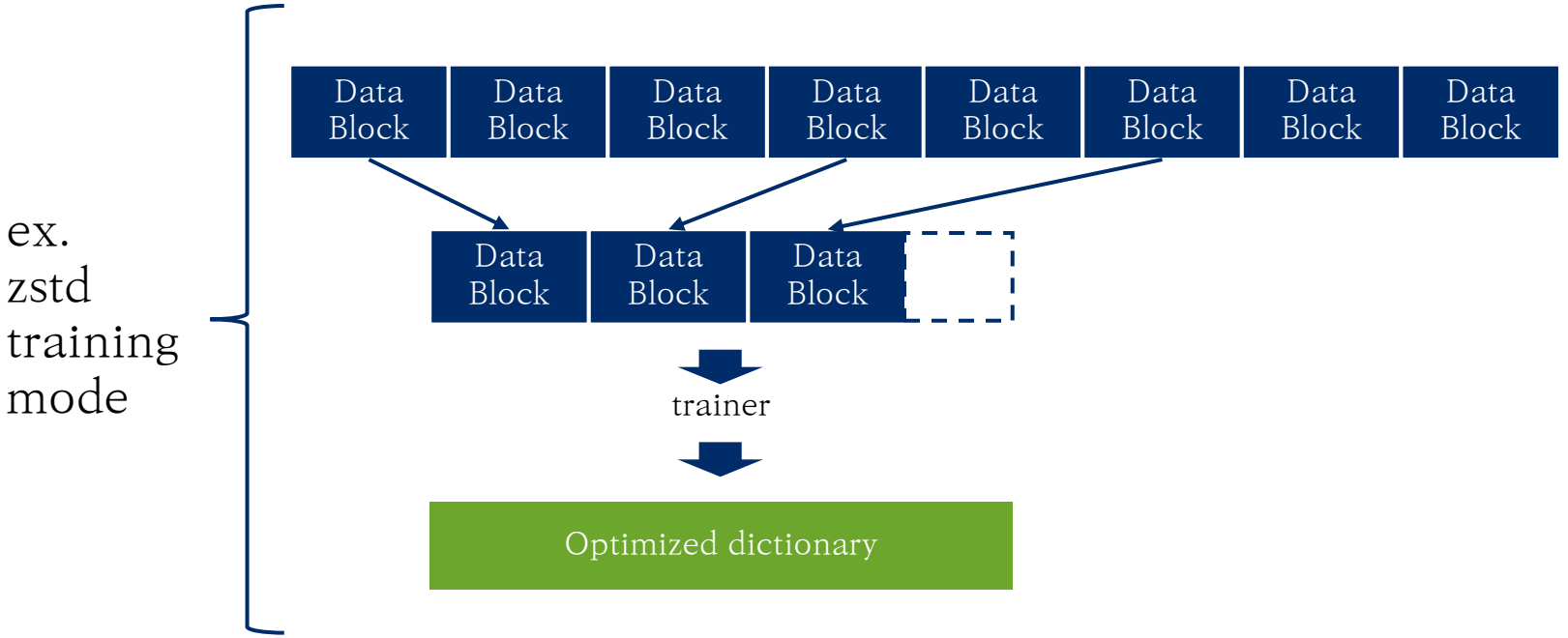
Read&Write Optimization in RocksDB



Compression	Compaction
• SSTable이 사용하는 저장공간 감소 • 저장 효율 및 read 성능 향상 • ex. Snappy, ZSTD, LZ4	• LSM-tree 구조 내에서 데이터 병합, 구성 • write 성능 향상 • ex. Leveled, Universal, FIFO

배경 지식

How compression works in RocksDB



배경 지식

압축 알고리즘 동작 방식

- LZ77 계열 알고리즘
- 반복되는 데이터는 참조로 대체
- 데이터 스트림에서 이미 나타난 부분
  → (거리, 길이) 쌍으로 인코딩

Snappy

- LZ77 변형 알고리즘
- 64KB 윈도우 내에서 반복 패턴 탐색
- 해시 테이블 통한 문자열 검색 최적화
- 일치하는 문자열
  → (오프셋, 길이) 쌍

LZ4

배경 지식

압축 알고리즘 동작 방식

Zlib

- DEFLATE 알고리즘 사용
- LZ77 알고리즘 → 반복 패턴
- 허프만 코딩:
  자주 사용되는 값 → 더 짧은 비트
- 압축 수준 조절 가능 (1-9)

ZSTD

- 향상된 LZ77 알고리즘
  + FSE 인코딩 결합
- 사전 훈련된 딕셔너리 사용 가능
- 압축 레벨 조정 가능 (-5~22)
- 컨텍스트 기반 매칭과
  휴리스틱 알고리즘

배경 지식

압축 알고리즘 특징

압축 알고리즘	특징	압축률	압축/해제 속도	CPU 사용량
NoCompression	압축 없음	없음	매우 빠름	매우 낮음
Snappy	속도 중심의 경량 압축 알고리즘	보통(~1.5x)	매우 빠름	낮음
LZ4	경량 압축 알고리즘	보통(~2x)	빠름	낮음
Zlib	높은 압축률, CPU 사용량 높음	높음(~3.5x)	느림	높음
ZSTD(Zstandard)	균형 잡힌 알고리즘	높음(~3.5x)	보통~빠름	중간

연구 배경

CF마다 다른 압축 알고리즘 적용

Hot Column	Cold Column
자주 쓰이고 읽히는 데이터 처리 속도 및 효율 최적화가 중요	저장 위주의 데이터 장기 저장 및 비용 효율성이 중요
↓	↓
빠른 처리. 낮은 지연, 낮은 CPU 사용량 요구	높은 압축률, 저장공간 절약 요구

03

본론

- 가설 설정
- 실험 설계
- 실험 결과

가설 설정

가설 설정

Column Family 별로 Hot, Cold Data를 저장하고,

각각 다른 압축 알고리즘을 적용하면 DB 성능이 더 좋아질 것이다.

Column	Data	Compression Algorithm	Features
Hot Column	자주 쓰이고 읽히는 데이터	NoCompression, Snappy, LZ4	빠른 처리, 낮은 지연, 낮은 CPU 사용량
Cold Column	저장 위주의 데이터	Zlib, ZSTD	높은 압축률, 저장공간 절약

- RocksDB에서 Column Family별로 서로 다른 압축 알고리즘 설정 가능
- hot/cold data에 적합한 압축 알고리즘 방식 적용으로 성능 향상 기대

선행 실험 계획

선행 실험 계획

case	Hot Column	Cold Column
0	Snappy	Snappy
1	LZ4	Zlib
2	LZ4	ZSTD
3	None	Zlib
4	None	ZSTD
5	Snappy	Zlib
6	Snappy	ZSTD

실험 환경

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 설계

측정 지표

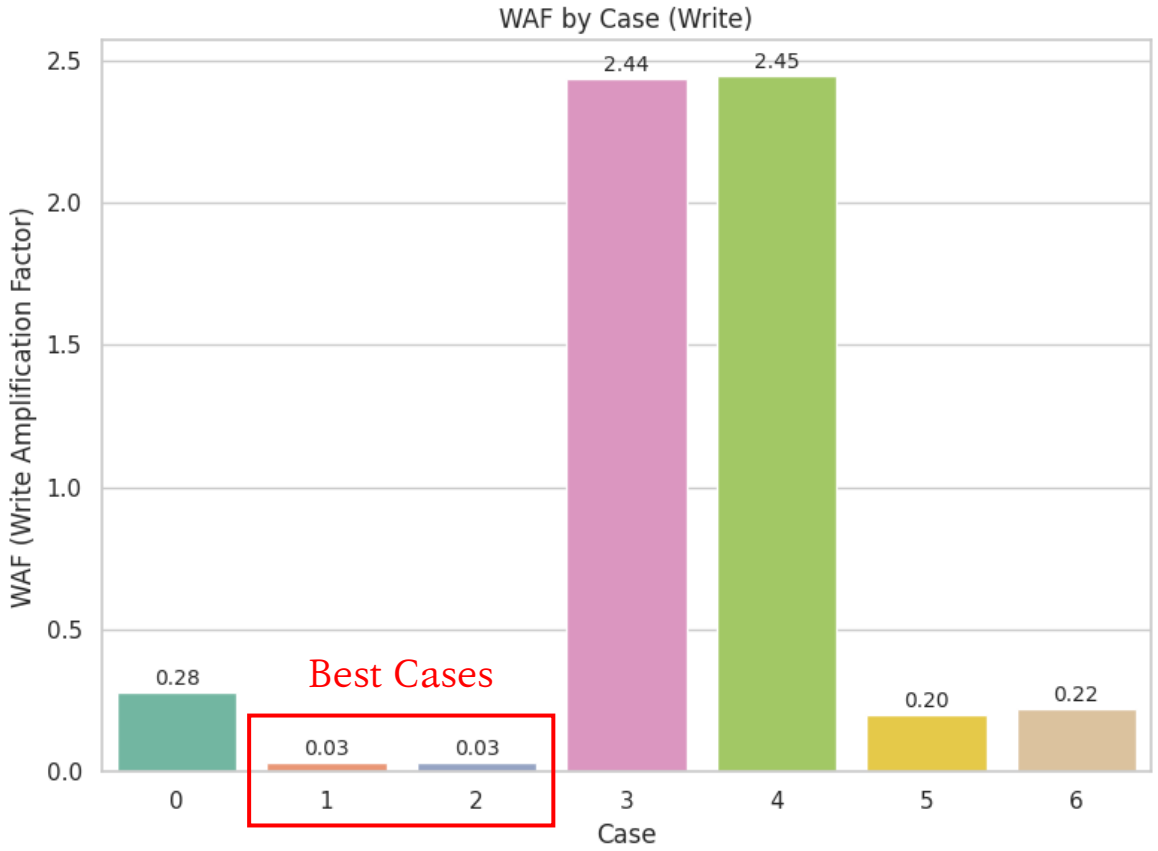
$$\text{Write Amplification Factor} = \frac{\text{rocksdb.flush.write.bytes} + \text{rocksdb.compact.write.bytes}}{\text{rocksdb.bytes.written}}$$

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

$$\text{Compression Ratio} = \frac{\text{rocksdb.bytes.compressed.to}}{\text{rocksdb.bytes.compressed.from}} \times 100$$

실험 결과

실험 결과: Write 성능



Case 1, 2: LZ4

Case 3, 4: No Compression

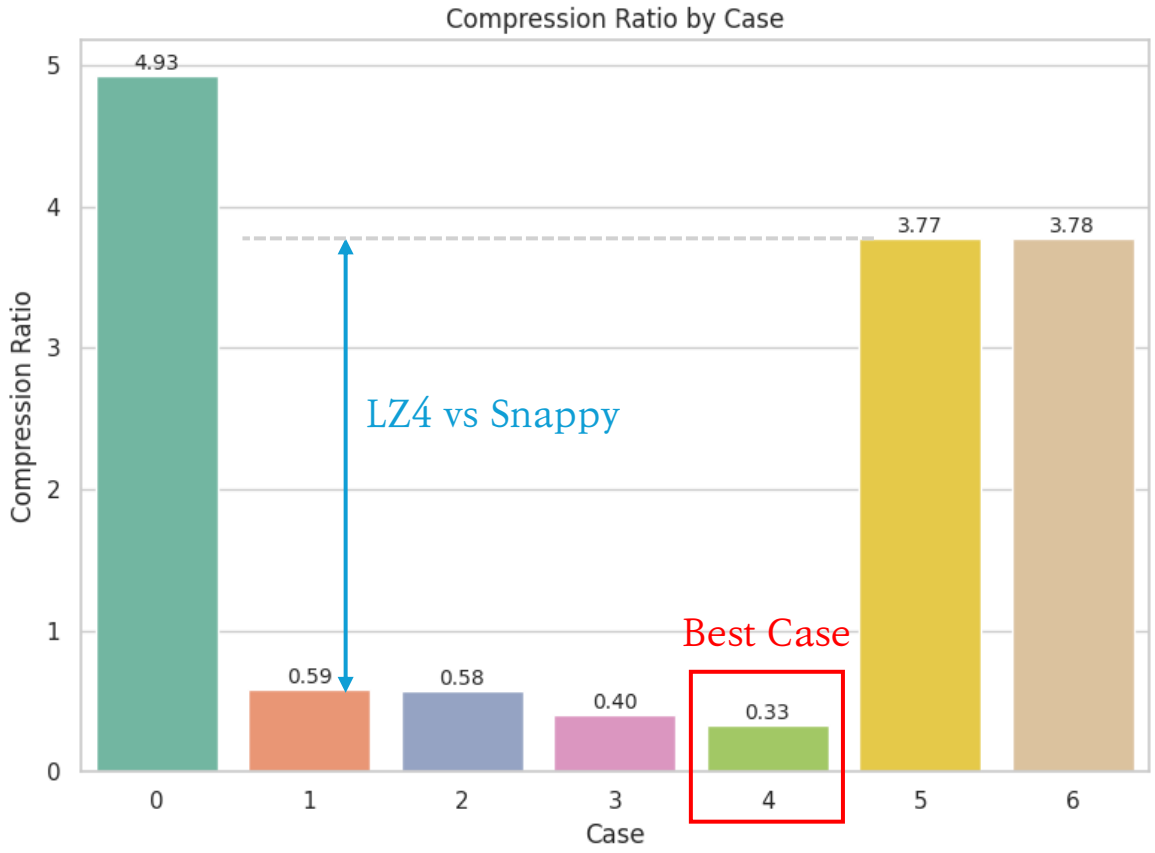
Case 5, 6: Snappy

Cold 데이터보다

Hot 데이터의 Compression 영향을 더 받는다

실험 결과

실험 결과: Write 성능



Case 1, 2: LZ4

Case 3, 4: No Compression

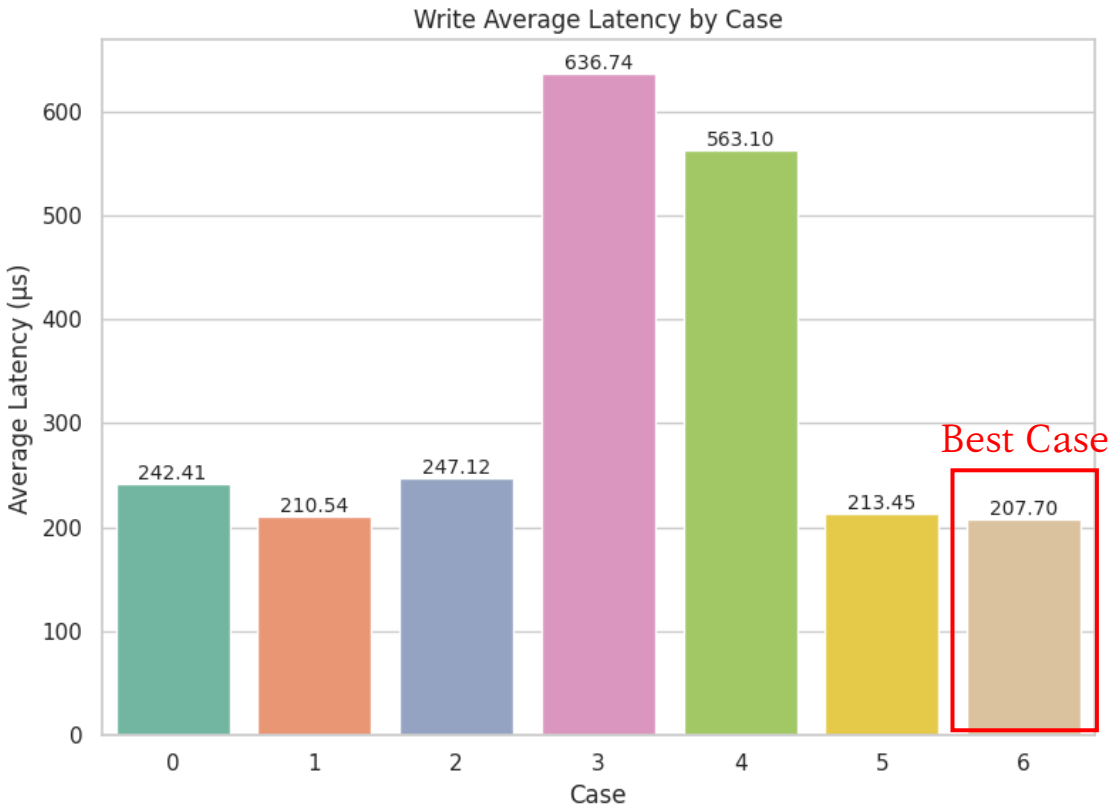
Case 5, 6: Snappy

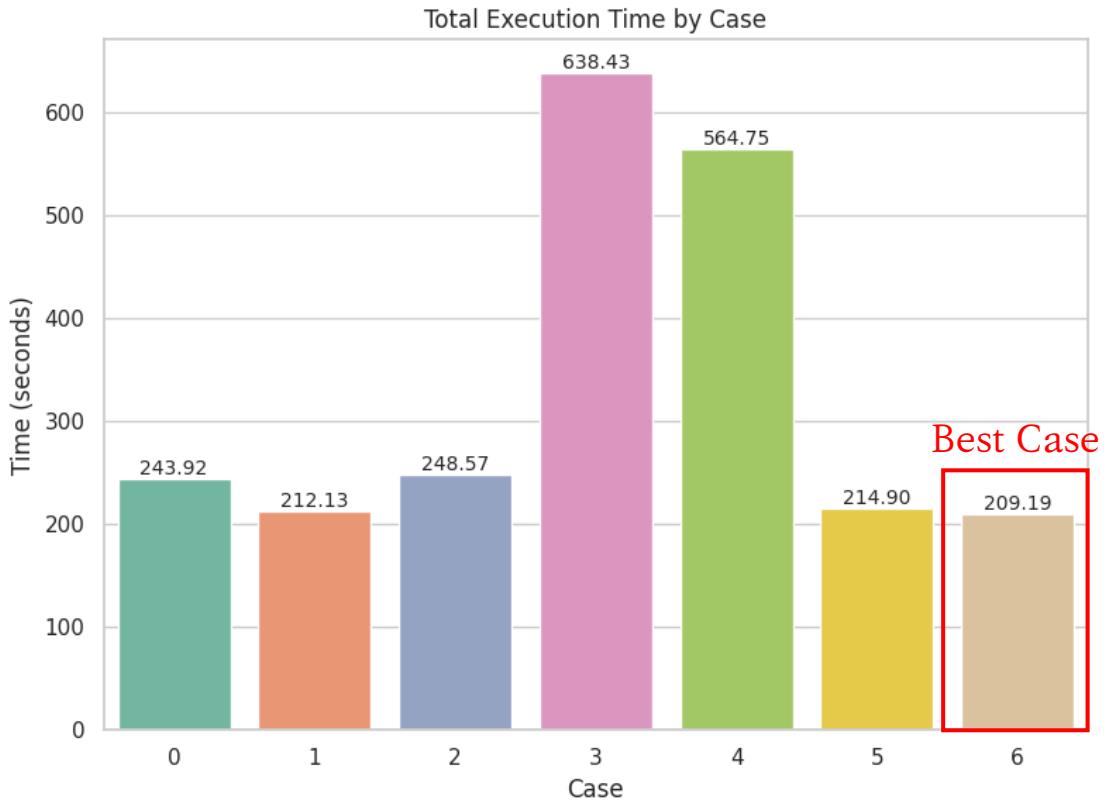
LZ4와 Snappy의 압축률 차이의 영향

Case 3, 4는 하나의 column family만 압축

실험 결과

실험 결과: Write 성능

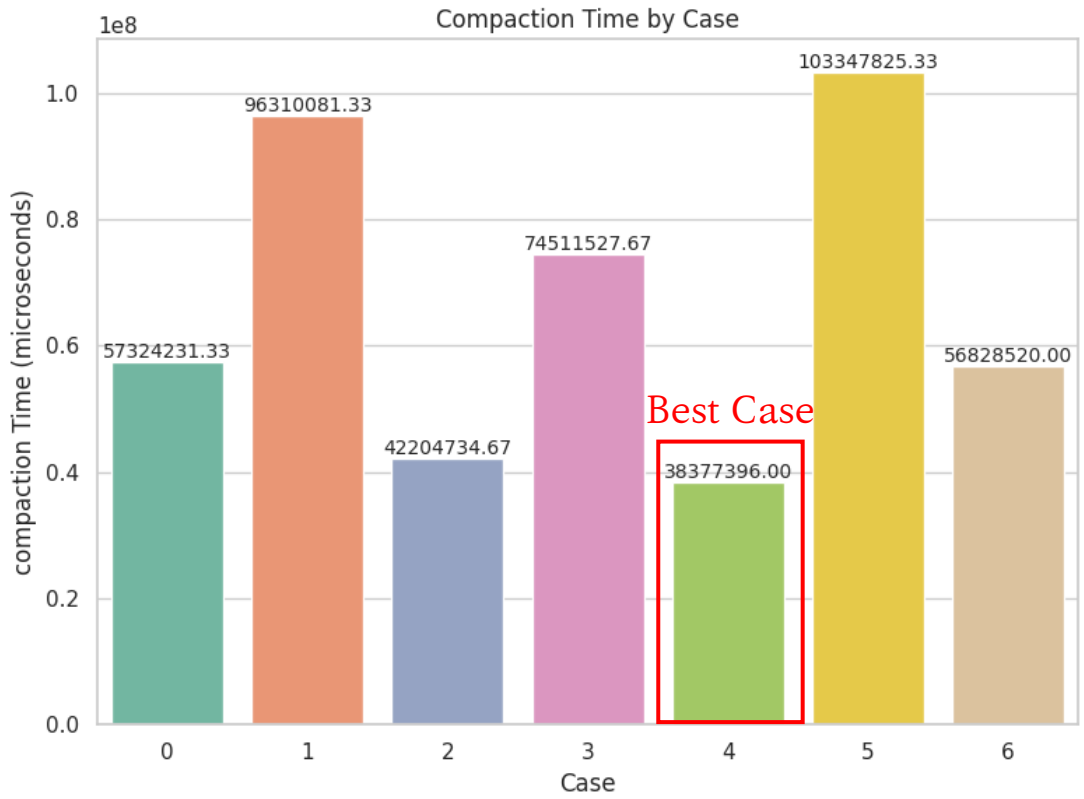




➡ hot column에 compression하면 큰 차이 없음

실험 결과

실험 결과: Write 성능



Case 1, 3, 5: Zlib

Case 2, 4, 6: ZSTD

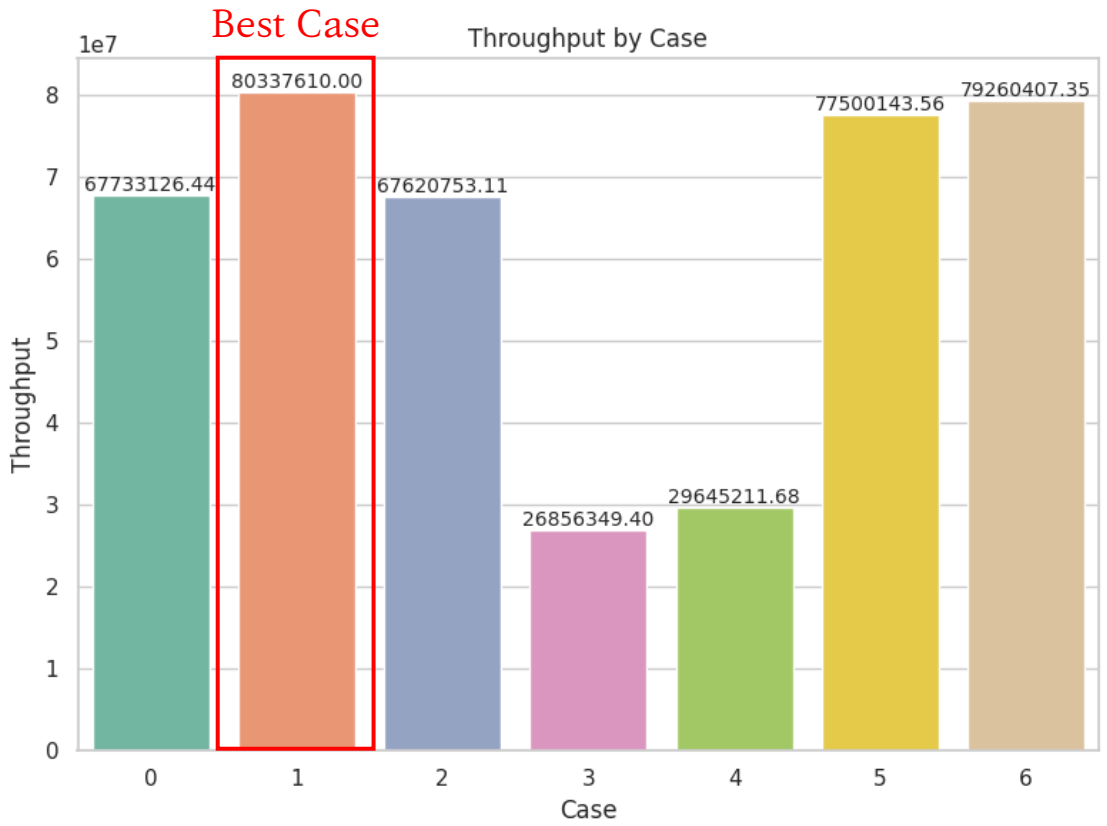
Cold Data Compression의 영향

Zlib, ZSTD의 압축/해제 속도 차이

No compression의 경우 cpu time 적음

실험 결과

실험 결과: Write 성능



Case 1

- LZ4 + Zlib
- LZ4로 인해 성능이 좋게 나온다

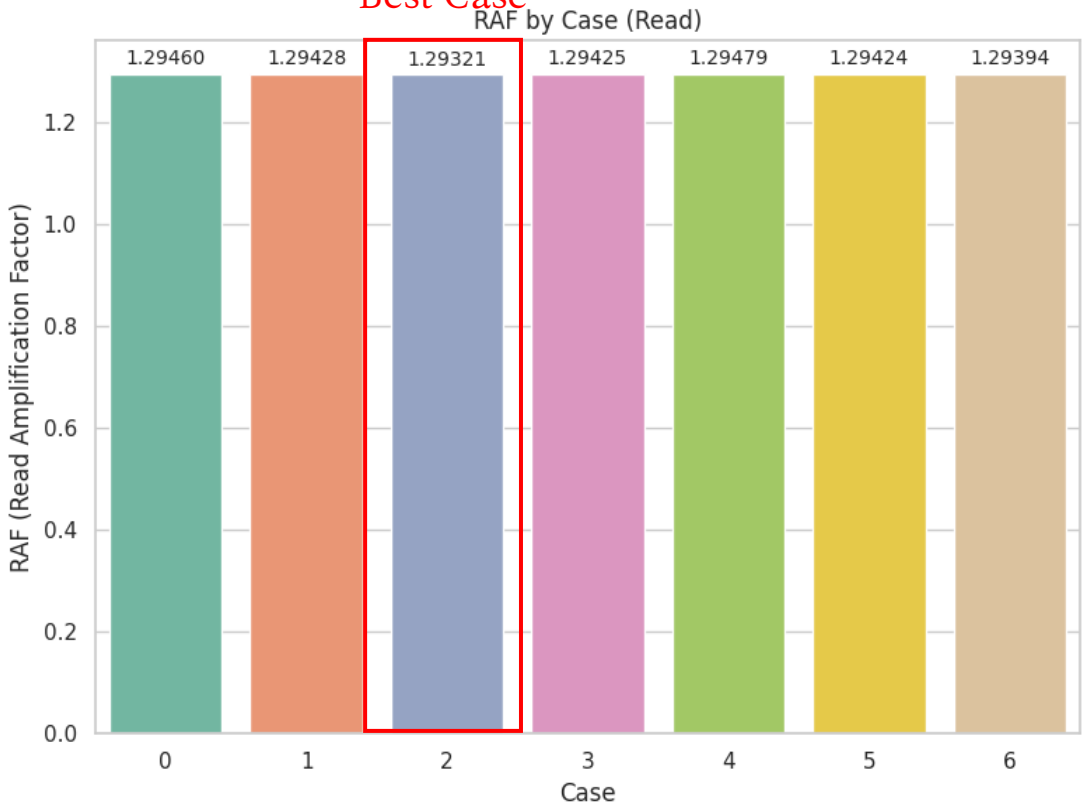
Case 3,4

- hot column에 압축 알고리즘을 사용하지 않으면
- I/O 병목이 발생하여 처리율이 저하됐을 수 있다.

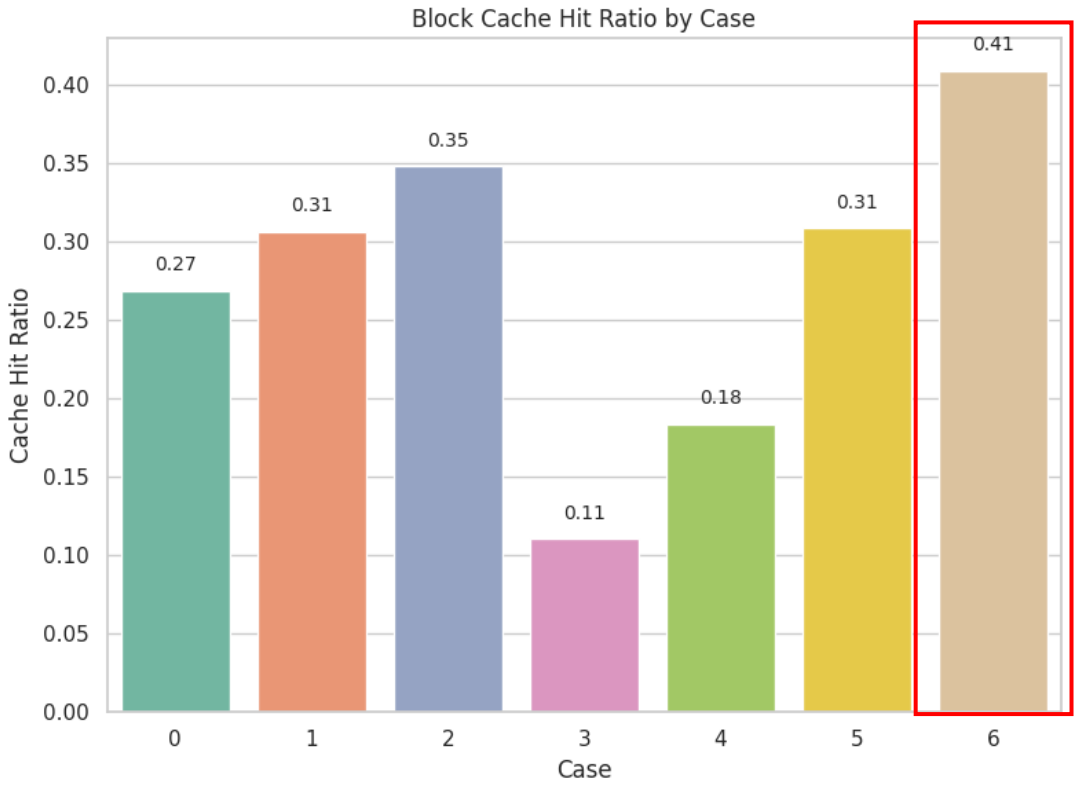
실험 결과

실험 결과: Read 성능

Best Case



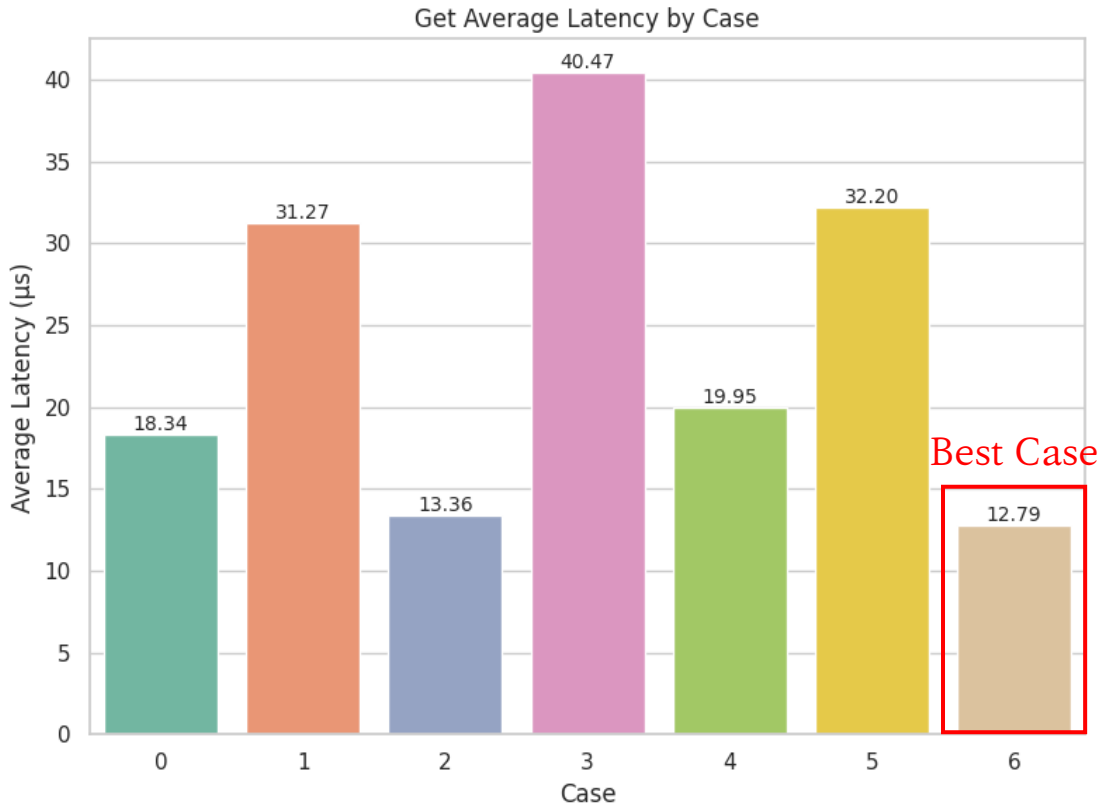
Best Case

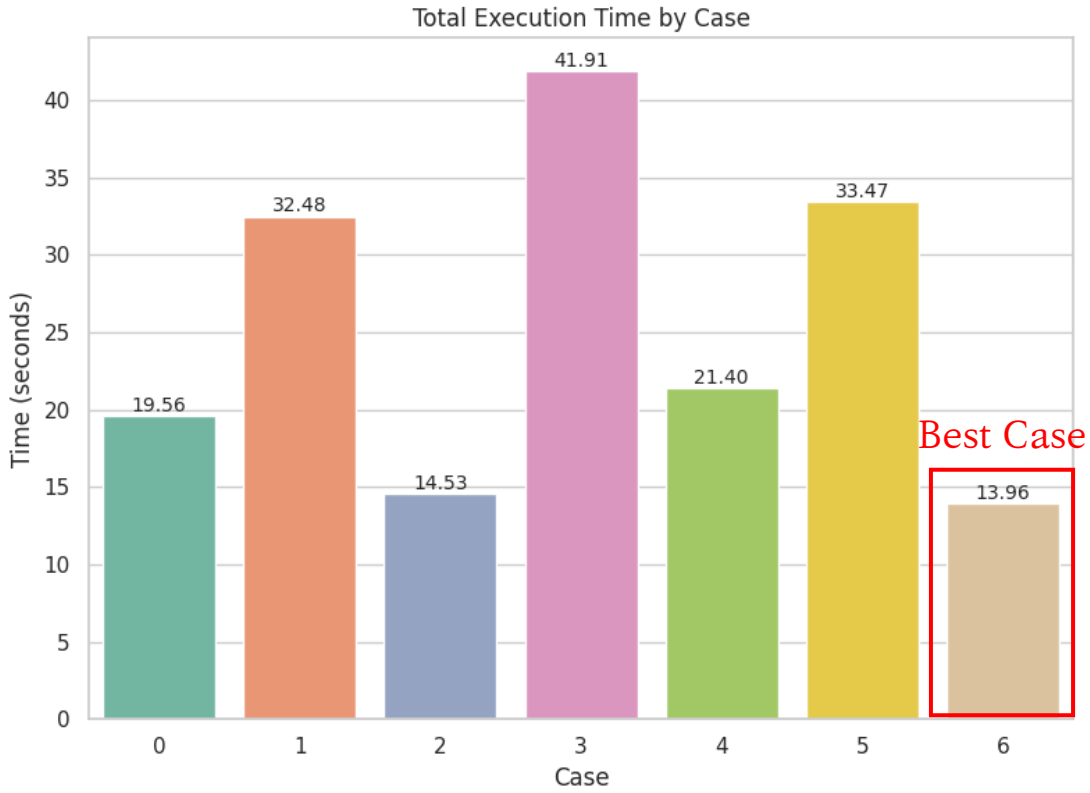


➡ Hot col에 compression하고 Cold col에 ZSTD를 사용한 경우 읽기 성능이 좋다

실험 결과

실험 결과: Read 성능

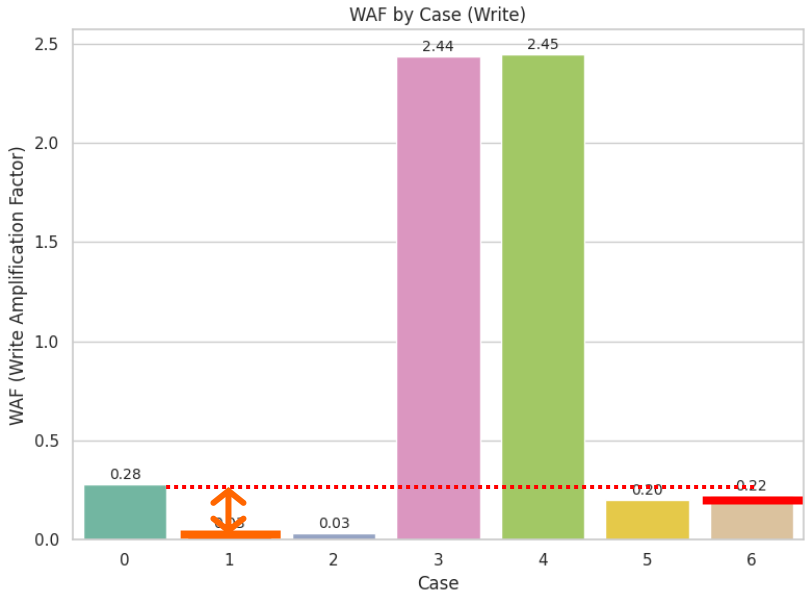


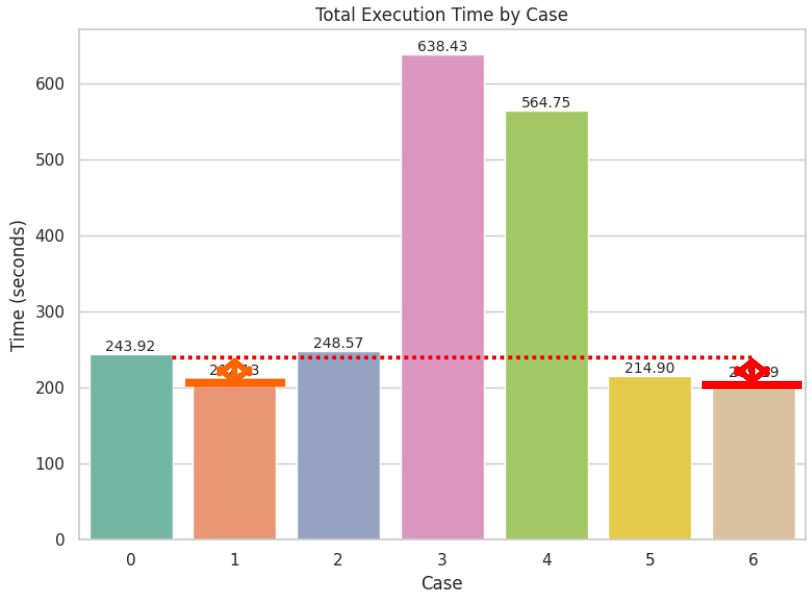


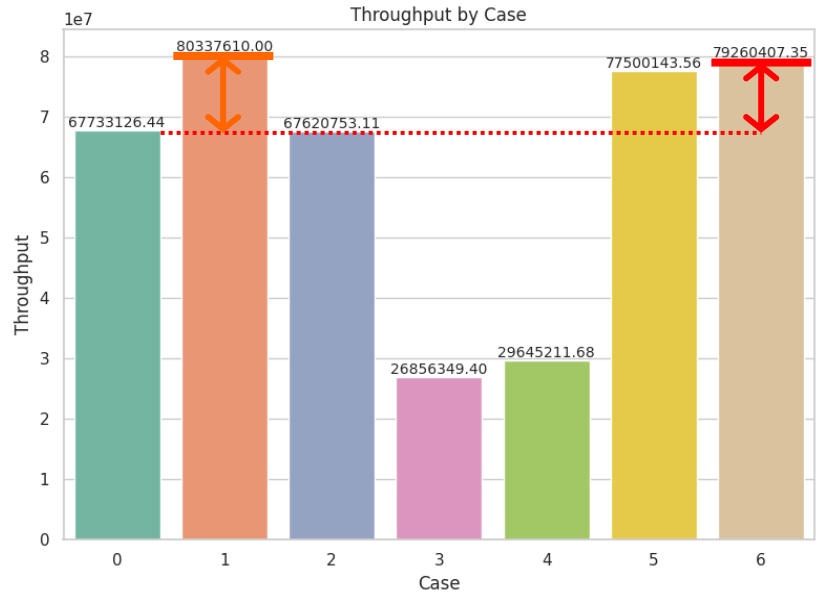
➡ Hot col에 compression하고 Cold col에 ZSTD를 사용한 경우 읽기 성능이 좋다

실험 결과 요약

실험 결과 요약: Write 성능



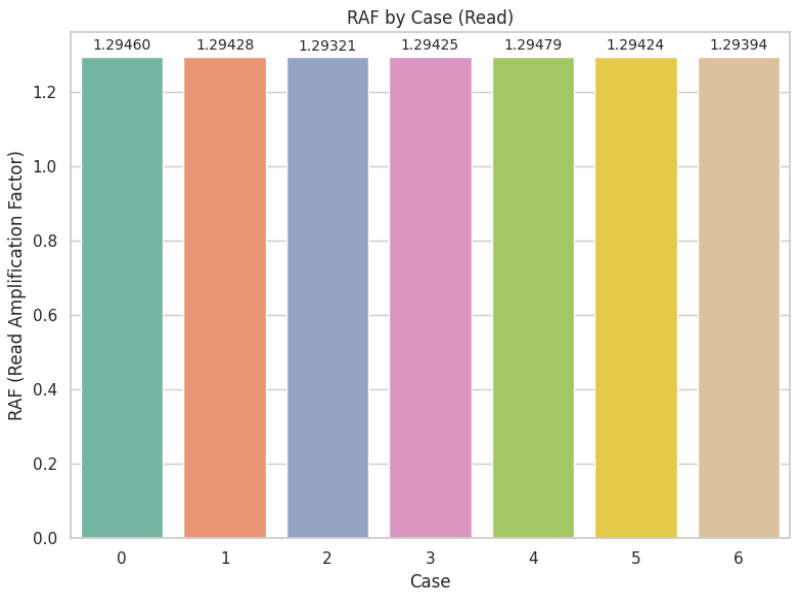


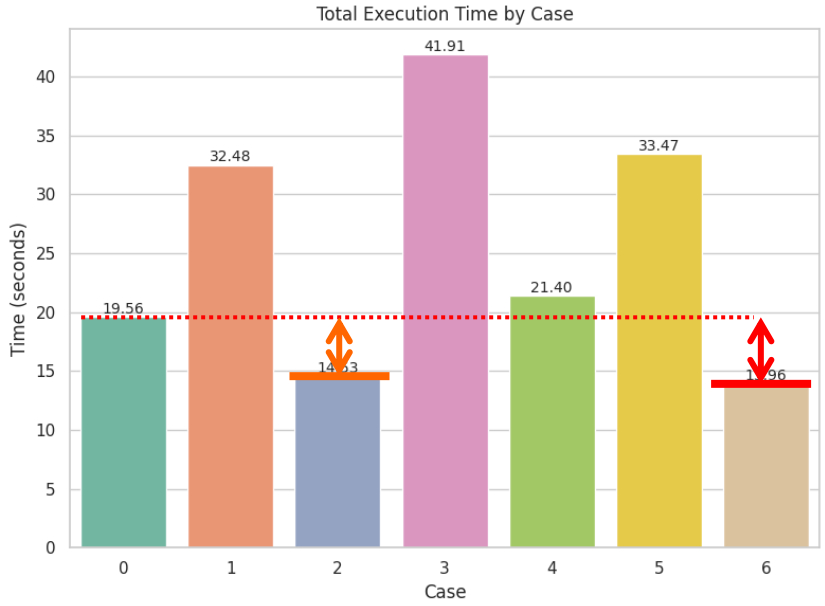


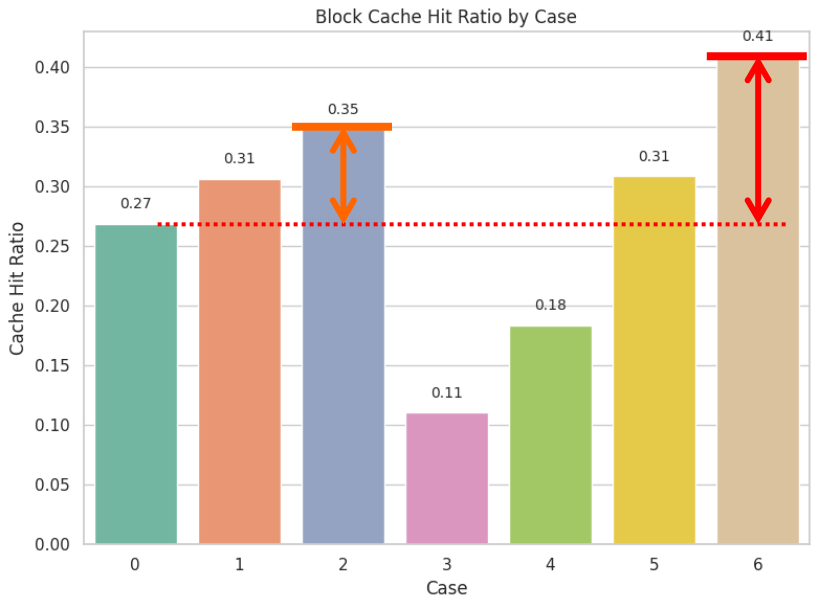
Case 1	LZ4, Zlib	WAF, compression ratio, Throughput
Case 6	Snappy, ZSTD	Latency, Total Execution Time

실험 결과 요약

실험 결과 요약: Read 성능







Case 6	Snappy, ZSTD	Total Execution Time, Block Cache Hit Ratio
Case 2	LZ4, ZSTD	RAF

원인 분석

원인 분석

[최적의 조합] Hot Column: **Snappy** + Cold Column: **ZSTD**

Hot Column	Cold Column
자주 쓰이고 읽히는 데이터 처리 속도 및 효율 최적화가 중요	저장 위주의 데이터 장기 저장 및 비용 효율성이 중요
▼	▼
압축&해제 속도가 빠르고 CPU 사용량이 적은 **Snappy**	높은 압축률로 저장공간 효율이 좋은 **ZSTD**

추가 실험 설계

추가 실험 설계

case	Hot Column	Cold Column
2	LZ4	ZSTD
6	Snappy	ZSTD
LZ4	LZ4	LZ4
Zlib	Zlib	Zlib
ZSTD	ZSTD	ZSTD

Write: Case 1 / Case 6

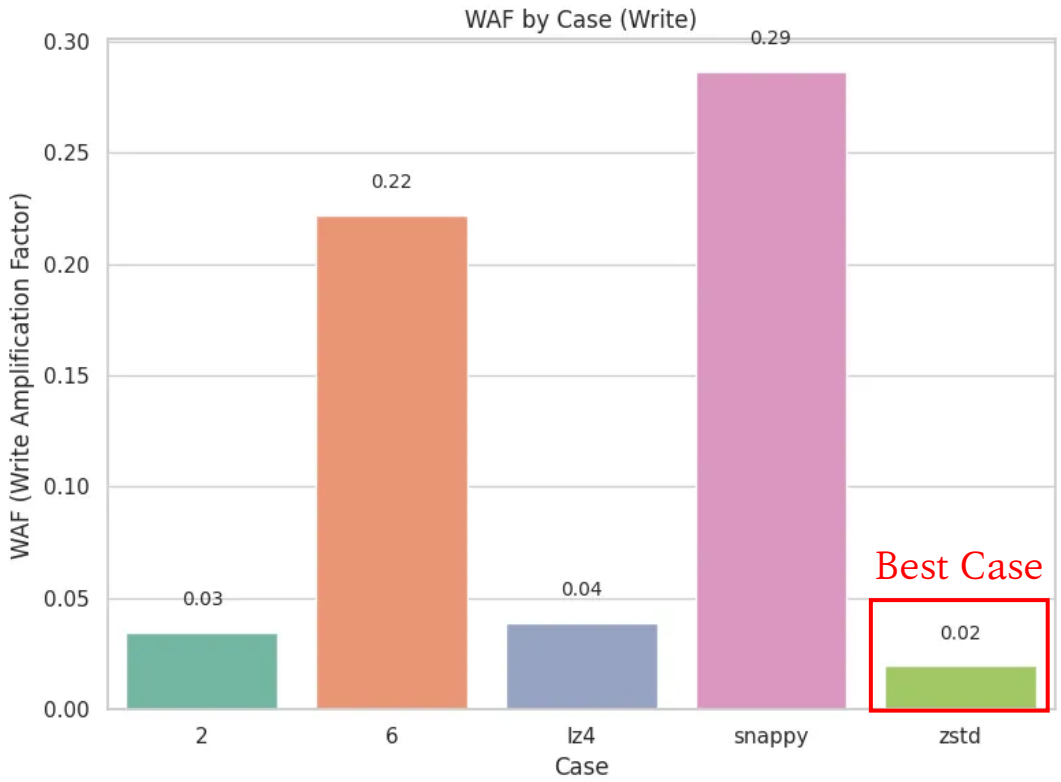
Read: Case 2 / Case6

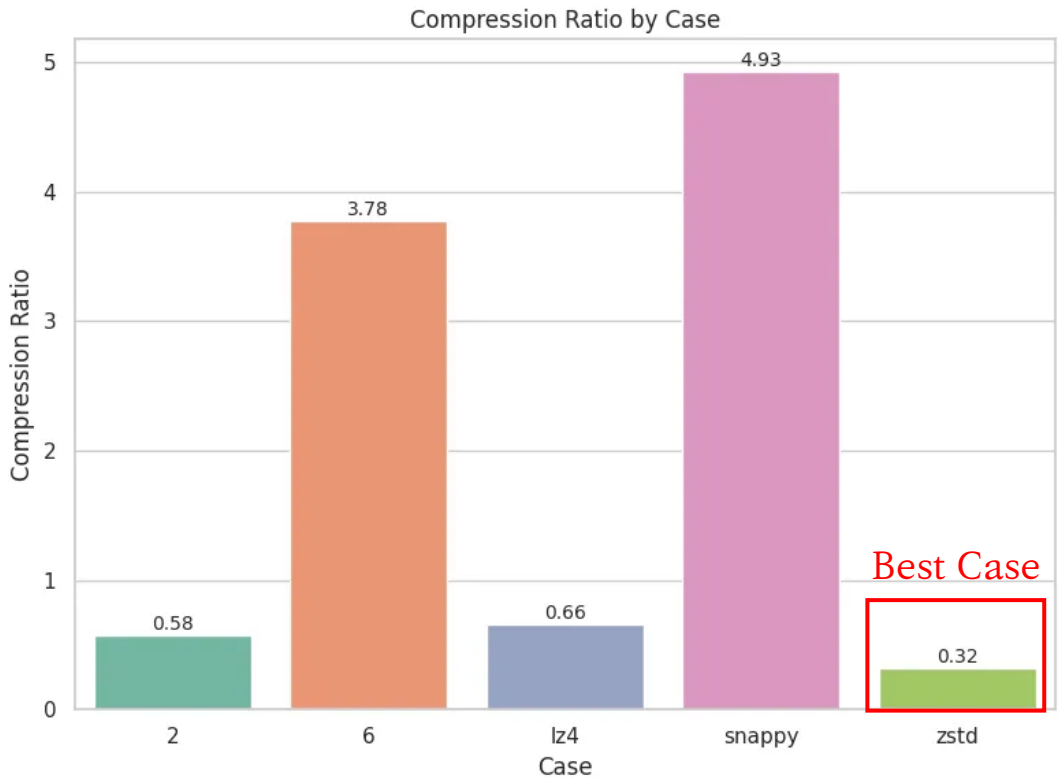
(Case 1과 Case2는 Write에서 큰 차이 X)

가장 성능이 좋은 Case 2, Case 6 추가 비교

추가 실험 결과

추가 실험 결과: Case 2,6과 Write 성능 비교

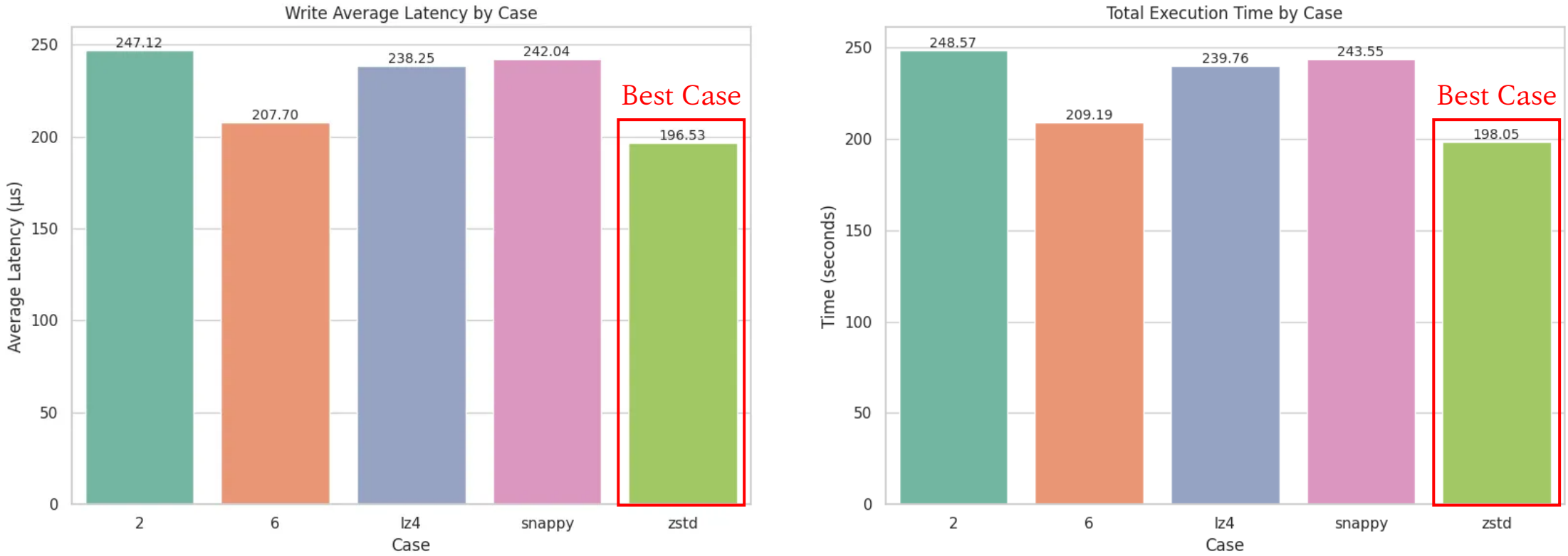




➡ Case 2, 6는 ZSTD를 사용하지만, ZSTD를 단독으로 사용하는 것보다는 쓰기 성능이 낮다

추가 실험 결과

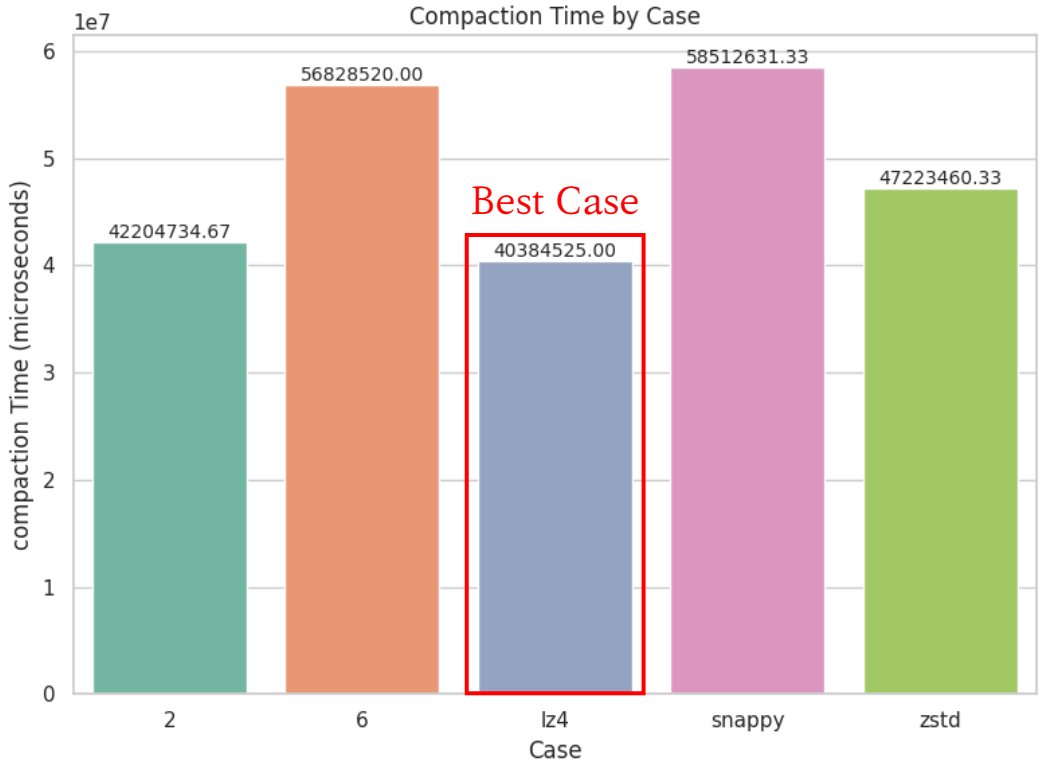
추가 실험 결과: Case 2,6과 Write 성능 비교

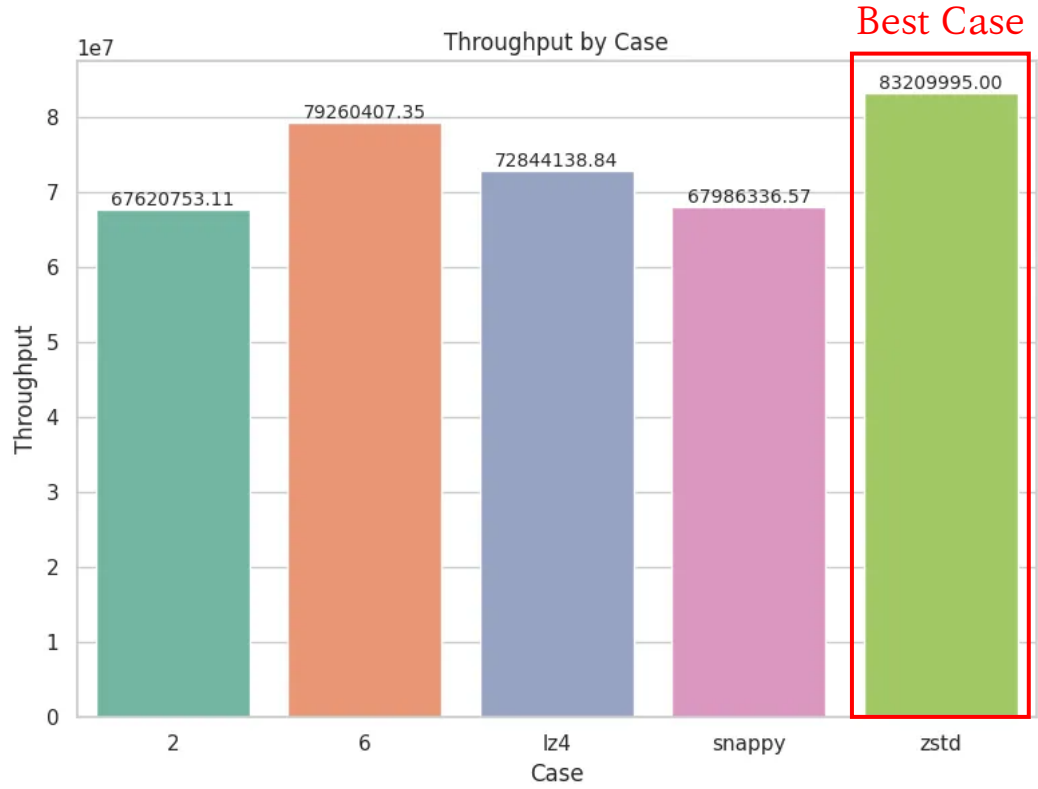


➡ Case 2, 6는 ZSTD를 사용하지만, ZSTD를 단독으로 사용하는 것보다는 쓰기 성능이 낮다

추가 실험 결과

추가 실험 결과: Case 2,6과 Write 성능 비교



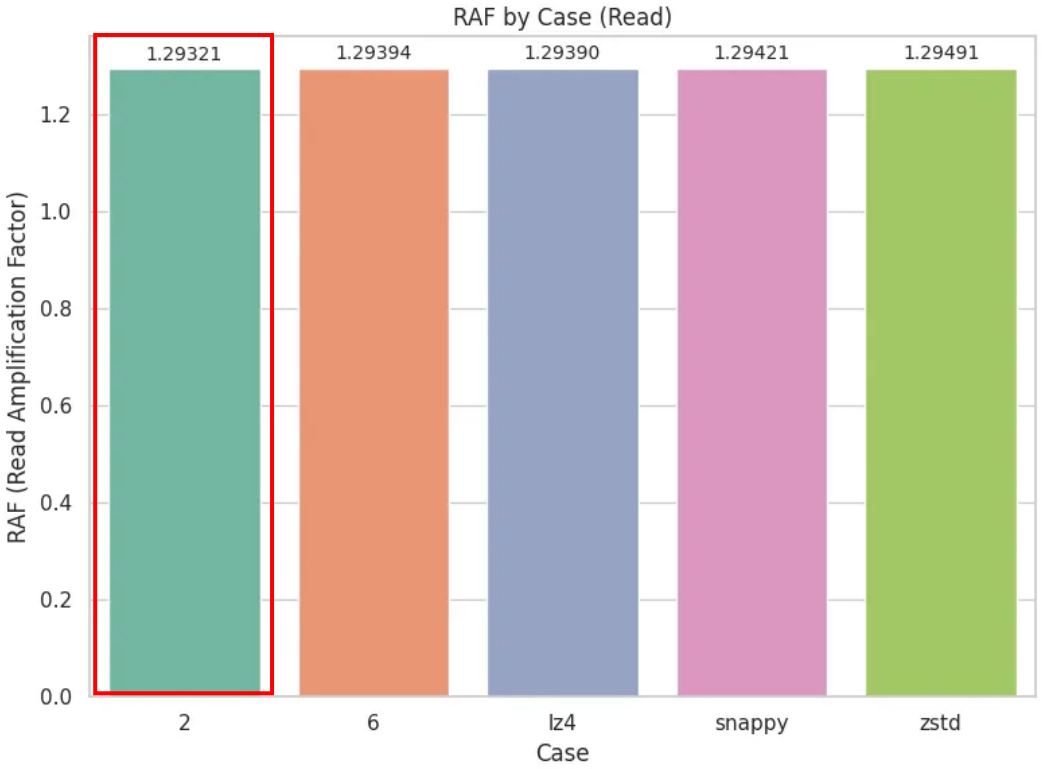


➡ Compaction time의 경우 LZ4 단독 사용 시 가장 우수한 성능을 보인다.

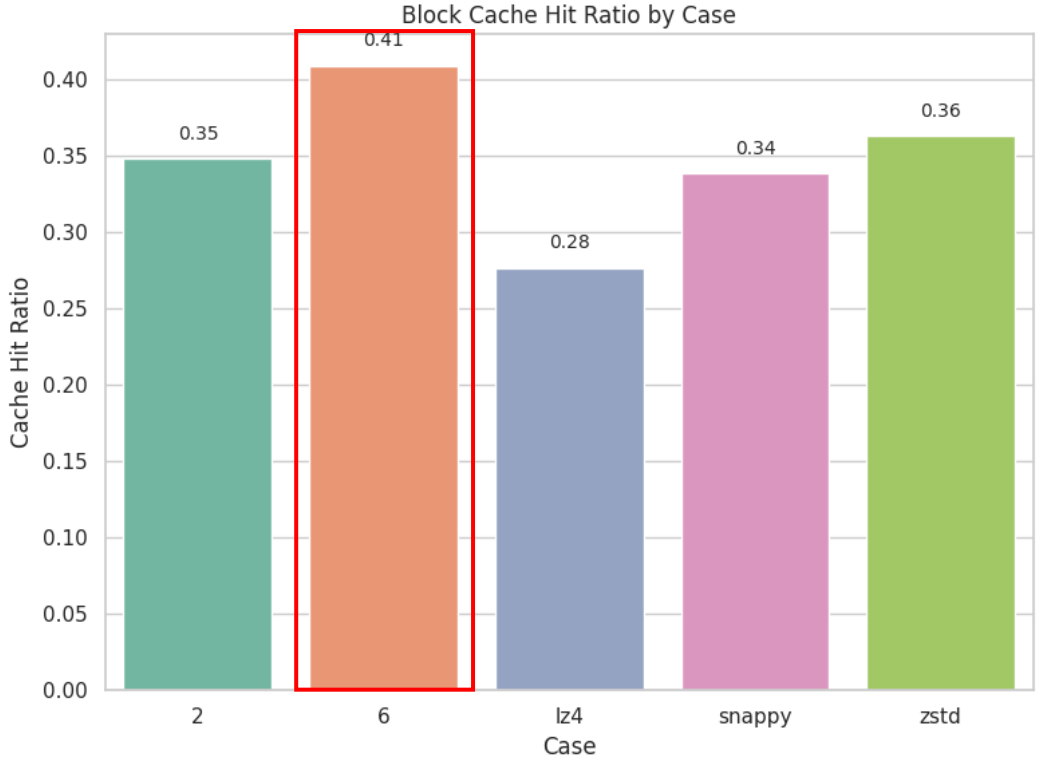
추가 실험 결과

추가 실험 결과: Case 2,6과 Read 성능 비교

Best Case



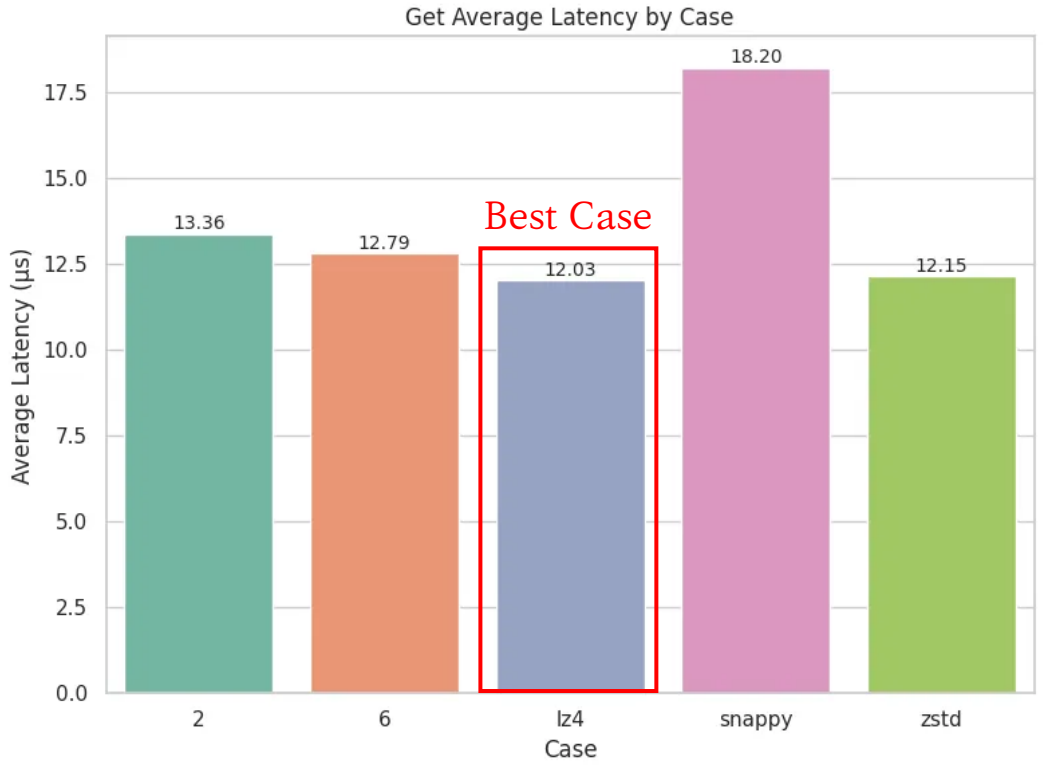
Best Case

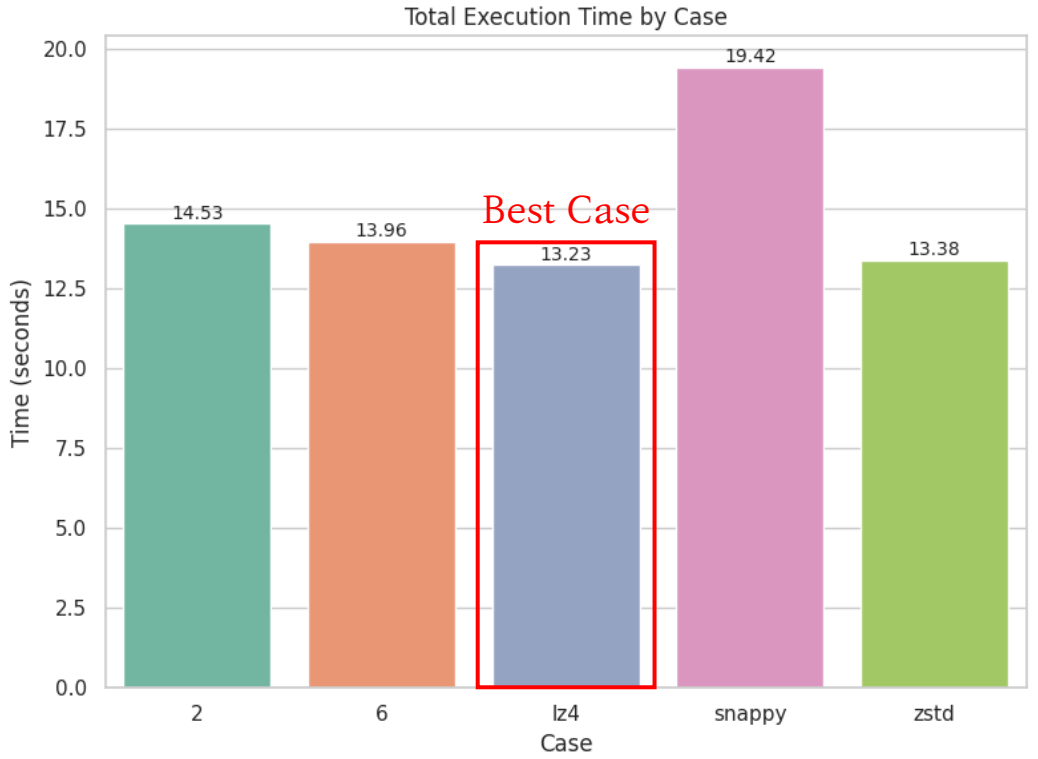


➡ Block Cache Hit Ratio의 경우 Snappy + ZSTD를 사용한 Case 6가 가장 우수한 성능을 보인다

추가 실험 결과

추가 실험 결과: Case 2,6과 Read 성능 비교





➡ Case 2는 LZ4를 사용하지만, LZ4를 단독으로 사용하는 것보다는 읽기 성능이 낮다

04

결론

- 가설 검증
- 실험 고찰

가설 검증

가설 검증

Column Family 별로 Hot, Cold Data를 저장하고,

각각 다른 압축 알고리즘을 적용하면 DB 성능이 더 좋아질 것이다.

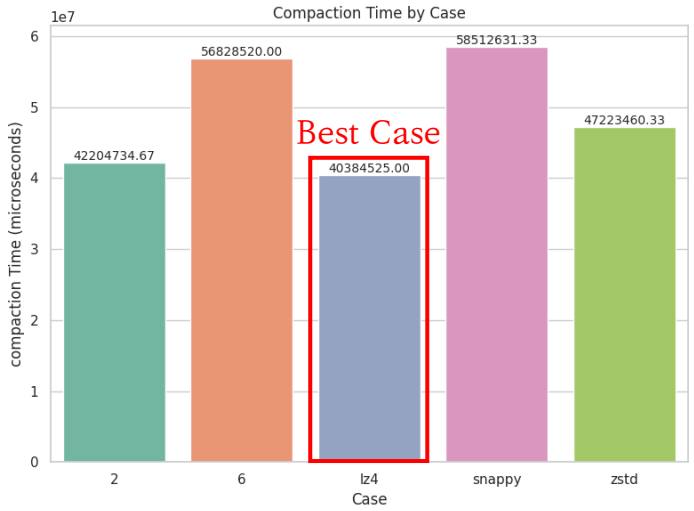
워크로드 유형	hot	cold	특징
쓰기 중심	LZ4	Zlib	낮은 WAF, 높은 throughput
읽기 중심	Snappy	ZSTD	빠른 응답, 높은 cache hit
읽기/쓰기 균형	Snappy	ZSTD	다양한 지표에서 고른 성능
	LZ4	LZ4	latency와 압축 효율의 균형 (단, CPU 성능이 보장된 경우)
저장 공간 민감	ZSTD	ZSTD	저장 최적화, compaction 관리 필요

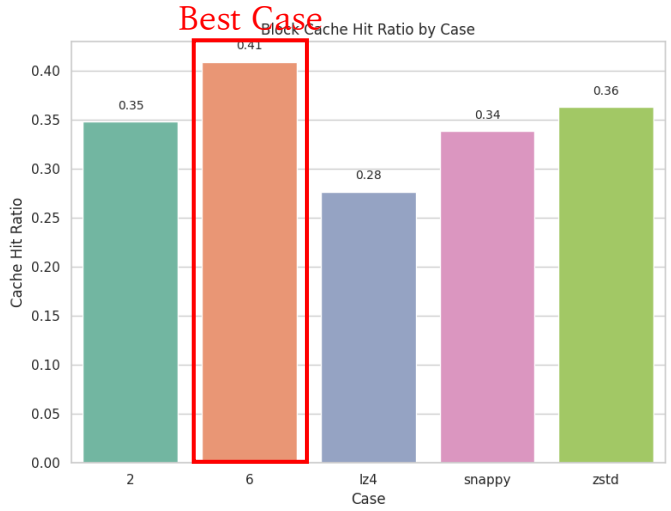
가설 검증

가설 검증

Column Family 별로 Hot, Cold Data를 저장하고,

각각 다른 압축 알고리즘을 적용하면 DB 성능이 더 좋아질 것이다.





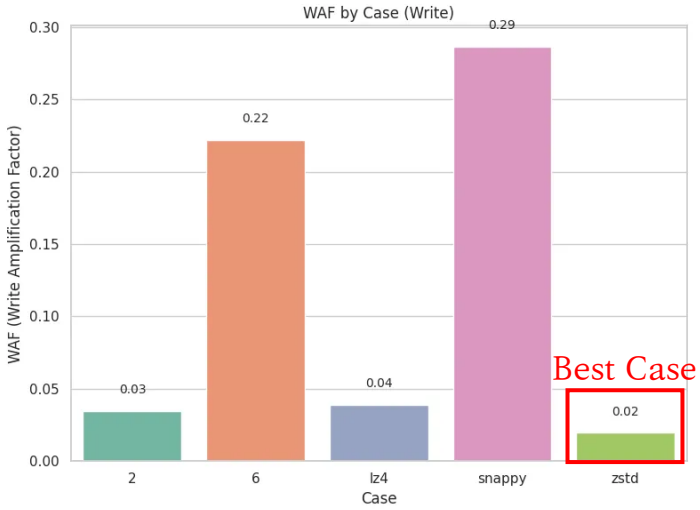
➡ 워크로드 특성에 따라 압축 알고리즘을 달리 적용하여 성능 개선에 효과를 볼 수 있다.

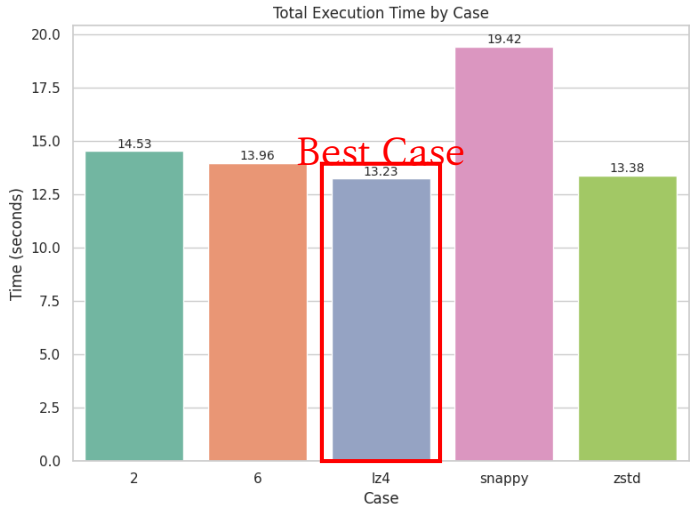
가설 검증

가설 검증

Column Family 별로 Hot, Cold Data를 저장하고,

각각 다른 압축 알고리즘을 적용하면 DB 성능이 더 좋아질 것이다.





➡ Write의 경우 ZSTD, Read의 경우 LZ4를 단독으로 사용하는 것이 성능이 높은 경우가 많다

실험 고찰

실험 고찰

- 워크로드 특성에 따라 압축 알고리즘을 구분 적용한 시도는 유의미했다.
- 하지만 ZSTD or LZ4 압축 알고리즘을 단일 적용한 케이스가
  각각 쓰기와 읽기에서 더 높은 성능을 보인 경우가 많았다.
- 성능 최적화는 직관과 항상 일치하지 않기 때문에,
  다양한 조합과 단일 설정을 모두 실험해보는 것이 중요하다.

05

참고 문헌

참고문헌

[1] BZip2(2019). https://sourceware.org/bzip2/

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

[3] Compression (2022). https://github.com/facebook/rocksdb/wiki/Compression

[4] Introduction to Data Compression(2020).

[5] LZ4(2024). https://github.com/google/lz4/lz4

[6] Preset Dictionary Compression(2021). https://rocksdb.org/blog/2021/05/31/dictionary-compression.html

[7] Snappy(2024). https://github.com/google/snappy

[8] Tuning Guide for RocksDB Compression and Decompression with Intel®(2022). https://www.intel.com/content/www/us/en/developer/articles/guide/rocksdb-compression-decompression-iaa-4th-gen-xeon.html#:~:text=After%20enabling%20the%20Intel%20IAA%20compress%2C%20it,improve%20compaction%20performance%20and%20reduce%20CPU%20use

[9] Zlib(2024). https://www.zlib.net/

[10] ZSTD(2017). https://github.com/facebook/zstd

Thank you

Thank you for listening to my presentation

단국대학교 DANKOOK UNIVERSITY