

RocksDB의 LSM-Tree 레벨별 특성을 고려한 적응형 압축(Adaptive Compression)

[Storage Optimization]

딥스토리 (DBStory) 팀

소프트웨어학과 이기윤
국제경영학과 홍사인
소프트웨어학과 노승아
소프트웨어학과 성진욱

목차

01

압축이란?

02

가설 1: 적응형 압축

03

가설 2: 쏠림 현상

04

결론 및 향후 계획

01 압축이란?

Storage Optimization

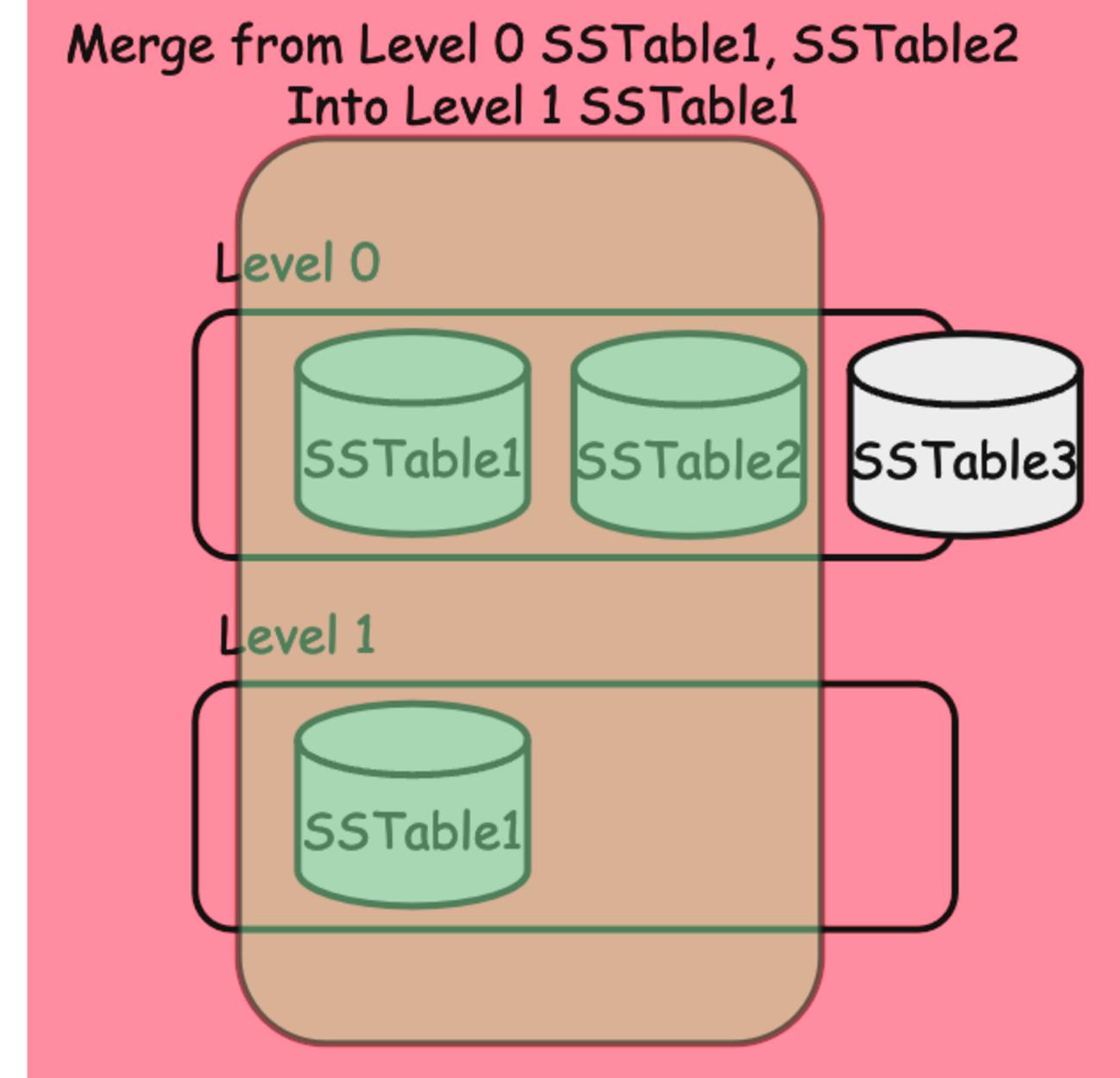
데이터의 중복을 제거(Compaction)하고
압축(Compression)하여

디스크 공간 효율을 높이는 동시에,
읽기/쓰기 성능을 극대화하는 시스템 엔지니어링 과정

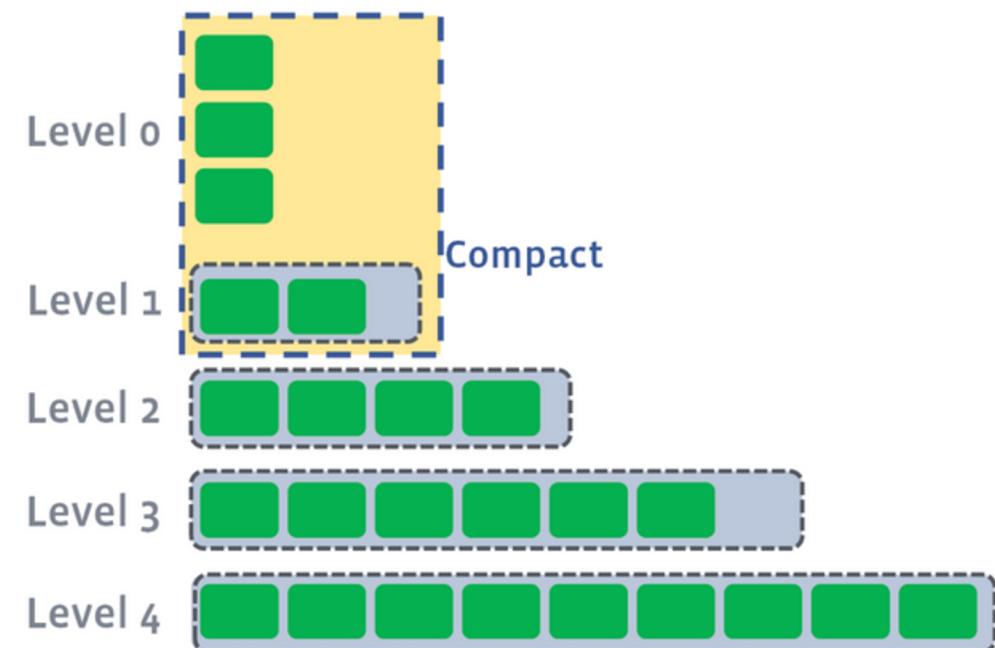
Compression

데이터의 '부피'를 줄이는 기술

SST 파일 내의 블록을 압축하여
스토리지 공간을 절약하는 것



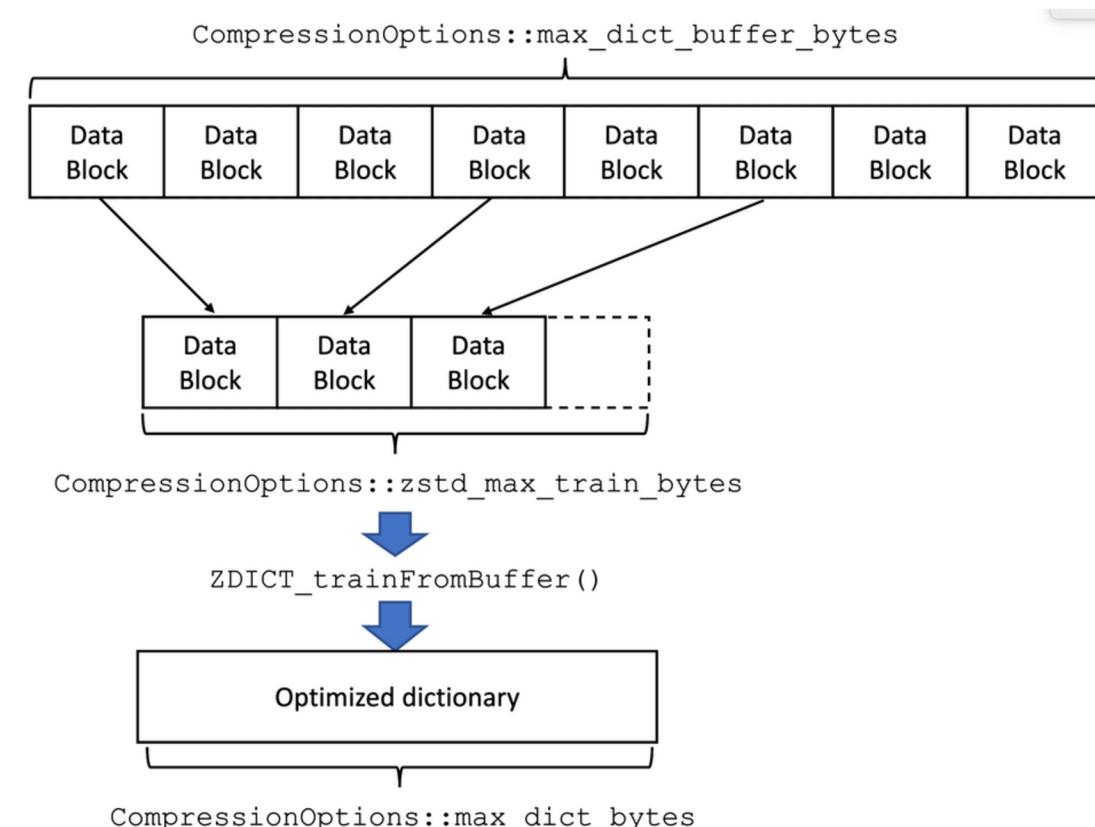
01 압축이란? Compaction vs Compression



Compaction

데이터의 '구조'를 정리하는 작업

흩어져 있는 작은 파일들을 모아서 큰 파일로 합치고,
그 과정에서 중복되거나 삭제된 데이터를 걸러내는 것



Compression

데이터의 '부피'를 줄이는 기술

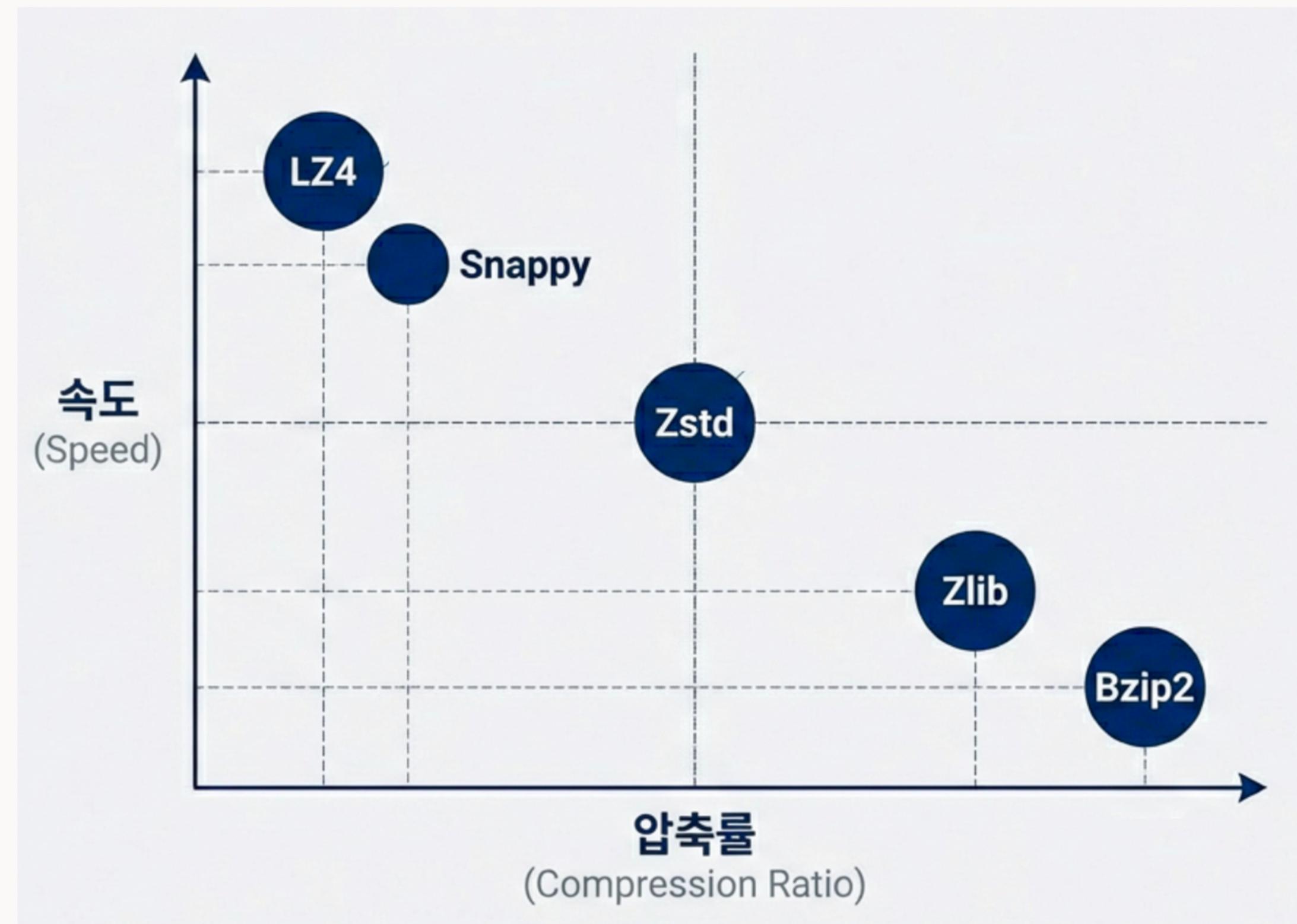
SST 파일 내의 블록을 압축하여
스토리지 공간을 절약하는 것

RESEARCH RESULTS

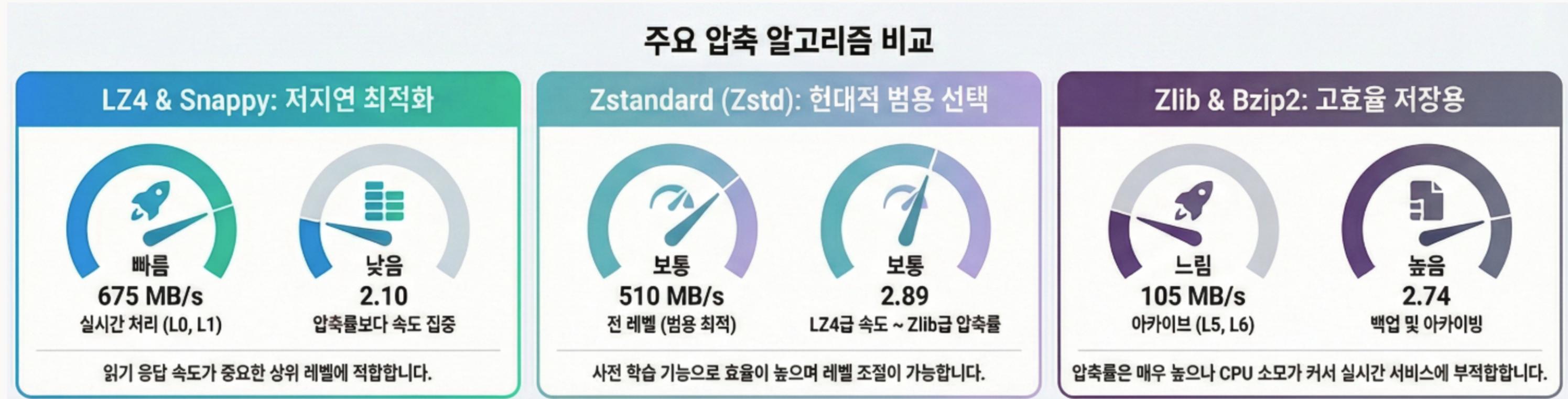
가설 1: 적응형 압축

배경 지식

02 가설 1: 적응형 압축 - Compression 알고리즘 비교



02 가설 1: 적응형 압축 - Compression 알고리즘 비교



Snappy
db_bench의 기본값

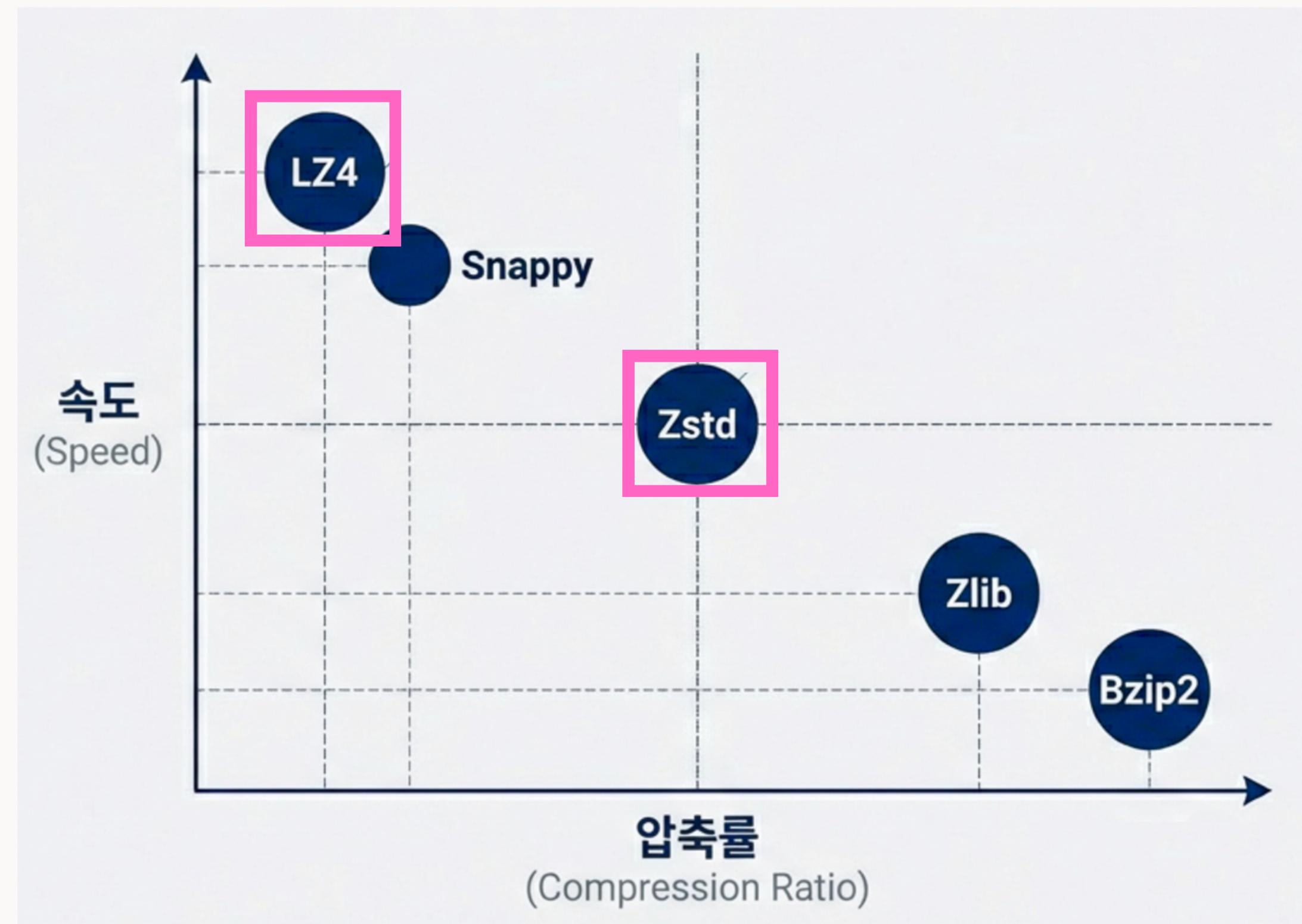
LZ4
Snappy의 대안

Zstd
최신 알고리즘

Zlib
고전 압축 방식

Bzip2
백업용 데이터

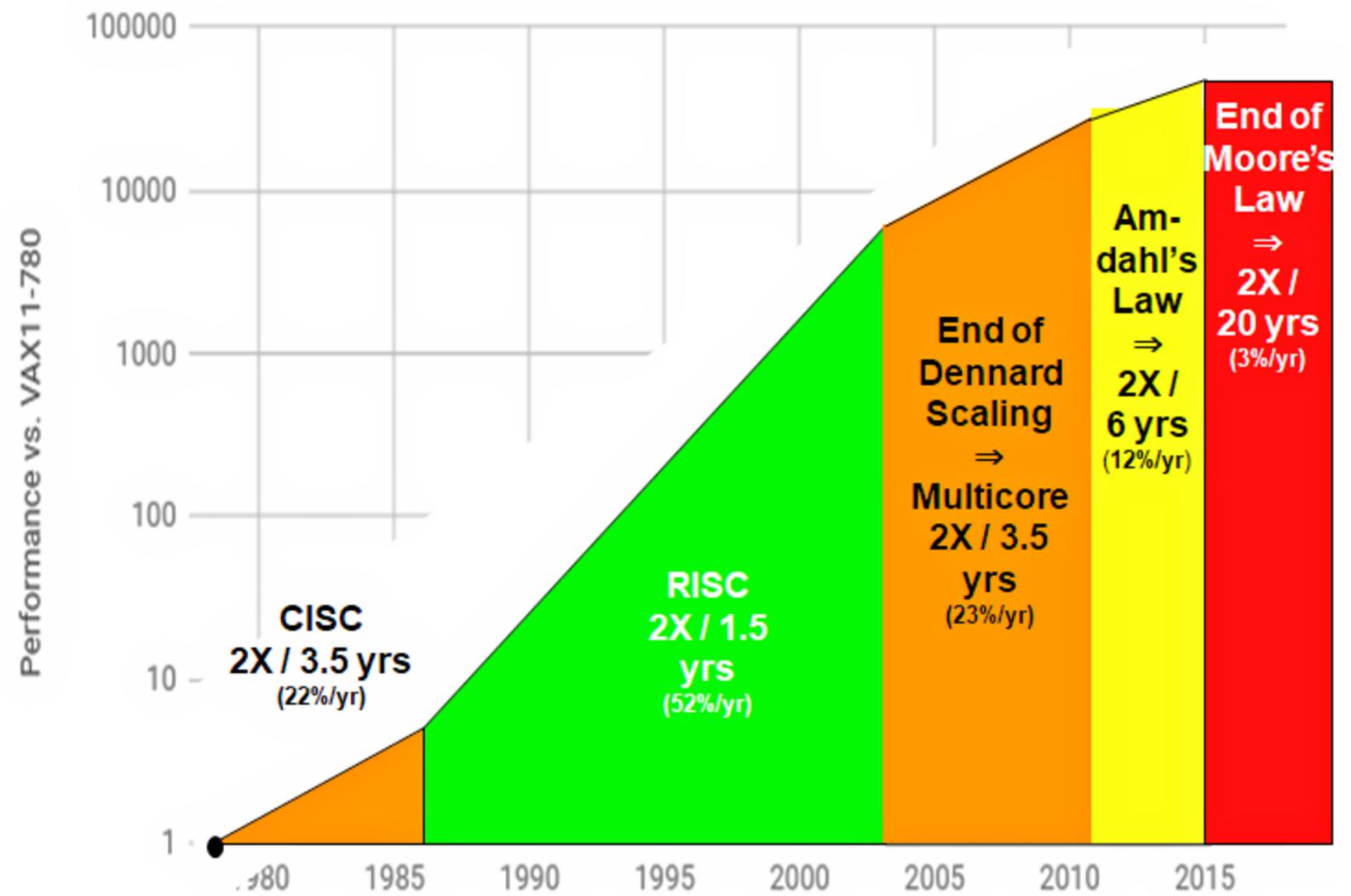
02 가설 1: 적응형 압축 - Compression 알고리즘 비교



가설 소개

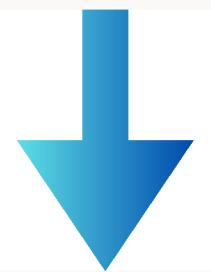
02 가설 1: 적응형 압축

40 years of Processor Performance



John Hennessy and David Patterson,
Computer Architecture: A Quantitative Approach

CPU의 빠른 발전 속도



압축에 발생하는
비용의 감소를 의미

02 가설 1: 적응형 압축

```
191 // Compress blocks using the specified compression algorithm.  
192 //  
193 // Default:kSnappyCompression, if it's supported. If snappy is not linked  
194 // with the library, the default is kNoCompression.  
195 //  
196 // Typical speeds of kSnappyCompression on an Intel(R) Core(TM)2 2.4GHz:  
197 // ~200-500MB/s compression  
198 // ~400-800MB/s decompression  
199 //  
200 // Note that these speeds are significantly faster than most  
201 // persistent storage speeds, and therefore it is typically never  
202 // worth switching to kNoCompression. Even if the input data is  
203 // incompressible, the kSnappyCompression implementation will  
204 // efficiently detect that and will switch to uncompressed mode.  
205 //
```

현재 Default 압축 방식은 Snappy 알고리즘

02 가설 1: 적응형 압축

가설 설정

CPU의 성능이 향상됨에 따라 Compression을 강하게 하는 것이
전체적인 성능을 향상시킬 것이다.

CPU의 성능이 높을수록 높은 Compression을 하는 것이
전체적인 성능을 향상시킬 것이다.

실험 세팅

02 가설 1: 적응형 압축 - 실험 환경 통일 + shell로 자동화

```
docker run --rm \
--name "rocksdb-bench" \
--cpus="$cpu" \
--memory="$mem" \
--memory-swap="$mem" \
--memory-swappiness=0 \
-e BENCHMARKS="$BENCHMARKS" \
-e NUM_KEYS=$NUM_KEYS \
-e VAL_SIZE=$VAL_SIZE \
-e BLOCK_SIZE=$BLOCK_SIZE \
-e CACHE_SIZE=$CACHE_SIZE \
-e STRATEGY_NAME="$STRATEGY_NAME" \
-e COMP_LIST="$COMP_LIST" \
-v "$DATA_DIR":/work \
-v "$(pwd)/run_bench.sh":/run_bench.sh \
shinythinking/rocksdb-study:latest \
/bin/bash -c "chmod +x /run_bench.sh && /run_bench.sh > \"$LOG_FILE\" 2>&1"
```

master_bench.sh

도커 컨테이너를 생성하고,
파라미터를 조정

```
db_bench \
--db=$DB_DIR \
--num="$NUM_KEYS" \
--value_size="$VAL_SIZE" \
--benchmarks="$BENCHMARKS" \
--threads=4 \
--cache_size="$CACHE_SIZE" \
--options_file=$OPTIONS_FILE \
--use_existing_db=false \
--histogram=true \
--stats_interval=500000 \
--statistics=true
```

run_bench.sh

master_bench로부터 파라미터를 받아
db_bench 실행

02 가설 1: 적응형 압축 - 파라미터 설정

```
CPUS=("2" "4" "6")
MEMS=("2g" "4g" "8g")

STRATEGY_CONFIGS=(
    "adaptive:none none lz4 lz4 zstd zstd zstd"
    "all_zstd:zstd zstd zstd zstd zstd zstd"
    "all_lz4:lz4 lz4 lz4 lz4 lz4 lz4"
    "no_comp:none none none none none none"
)

export BENCHMARKS="fillseq,compact,readrandom,stats"
export NUM_KEYS=2000000
export VAL_SIZE=4096
export BLOCK_SIZE=4096
export CACHE_SIZE=$((128 * 1024 * 1024))

ITERATIONS=1
```

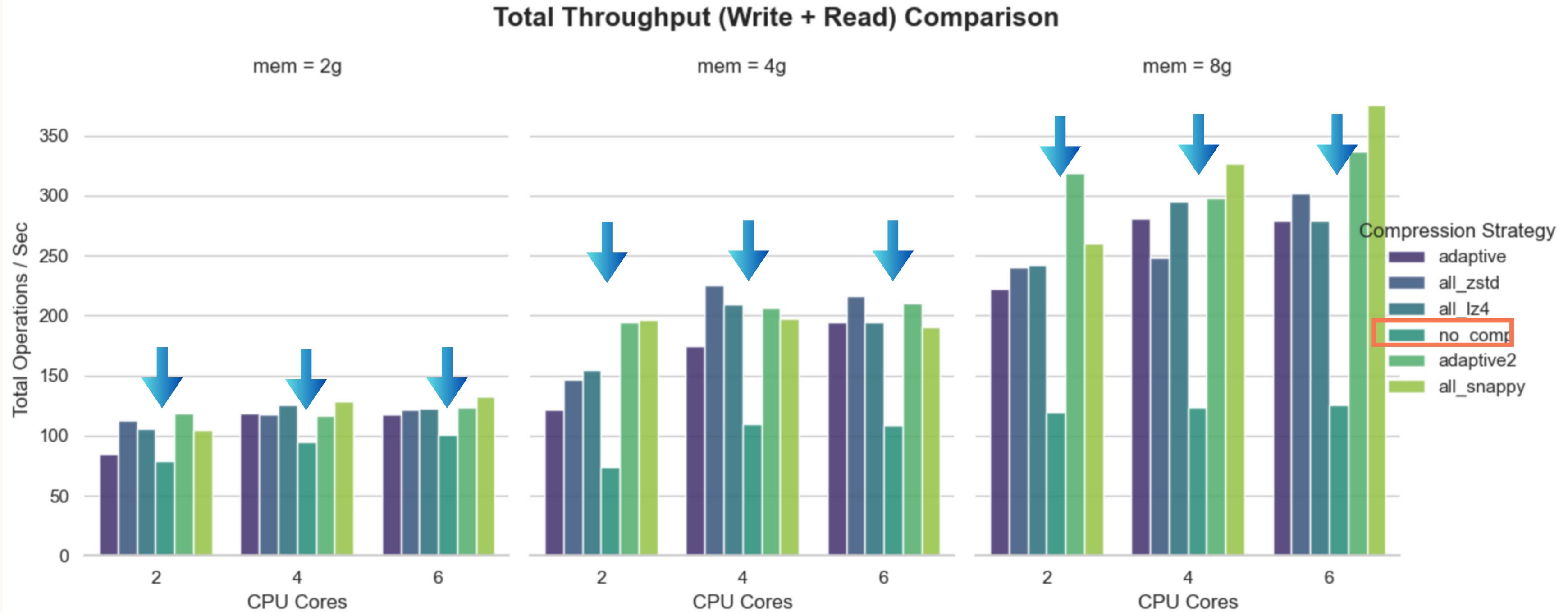
- CPUS: 진행할 코어 개수
- MEMS: 메모리 크기
- STRATEGY_CONFIGS: Compression Layer 구성
- BENCHMARKS: 벤치마크..
- NUM_KEYS: 생성할 키의 개수
- VAL_SIZE: 각 키에 저장될 값의 크기
- BLOCK_SIZE: 블록 단위
- CACHE_SIZE: 캐시 크기
- ITERATIONS: 동일 실험 반복

```
for cpu in "${CPUS[@]}"; do
    for mem in "${MEMS[@]}"; do
        for entry in "${STRATEGY_CONFIGS[@]}"; do
            STRATEGY_NAME="${entry%*:}"
            COMP_LIST="${entry#:*}"

            for ((i=1; i<=ITERATIONS; i++)); do
```

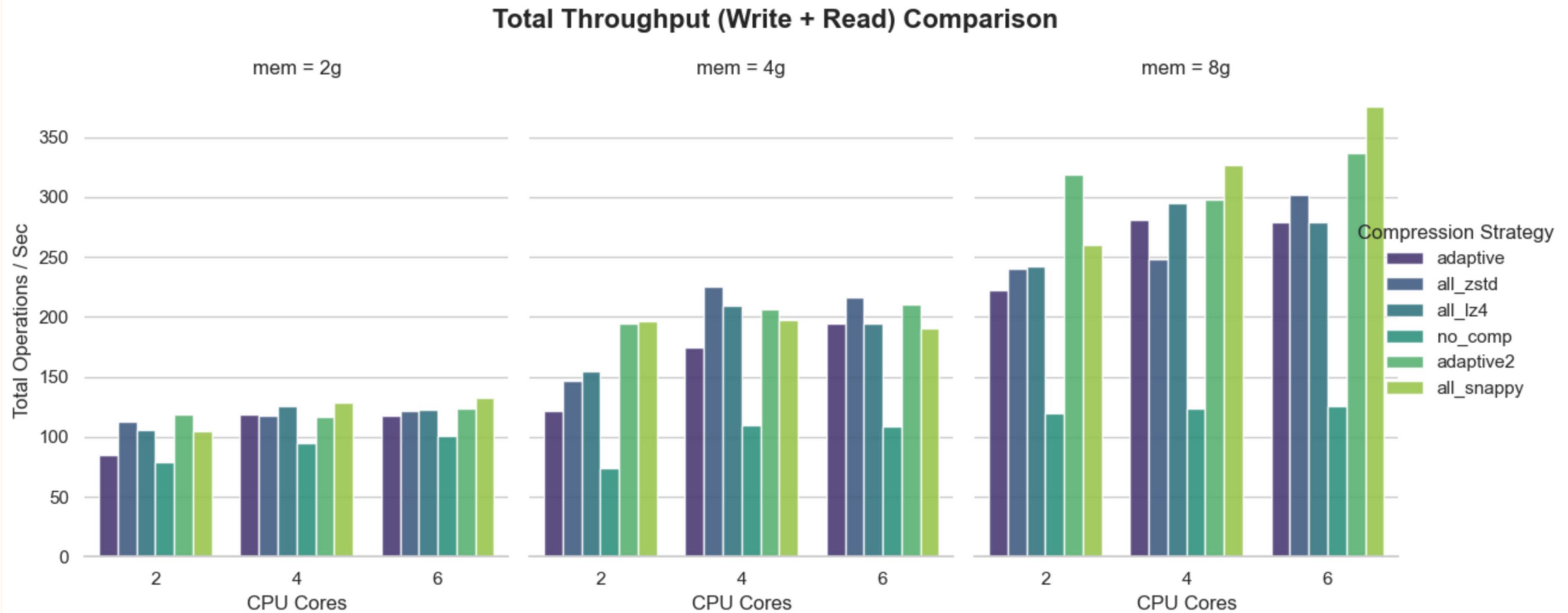
실험 결과

02 가설 1: 적응형 압축

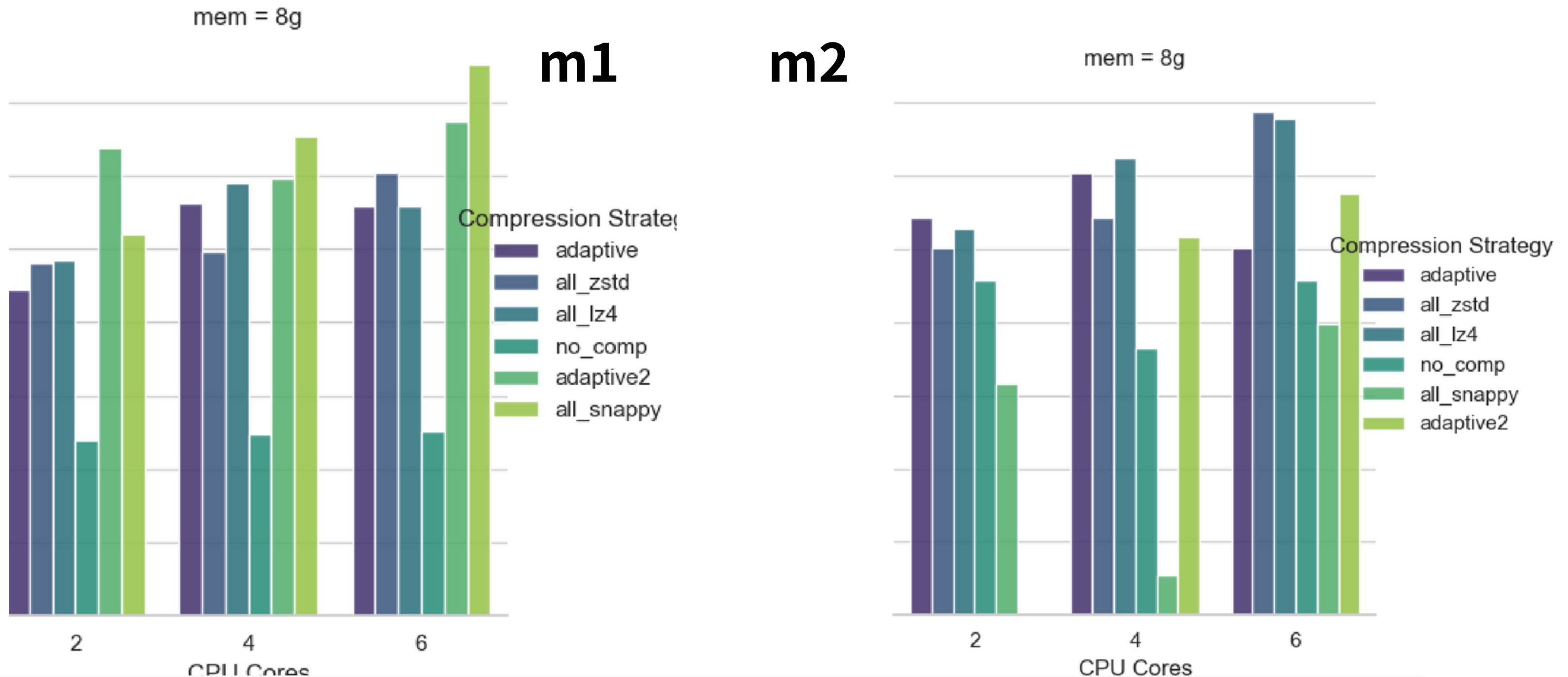


압축 O >> 압축 X

02 가설 1: 적응형 압축

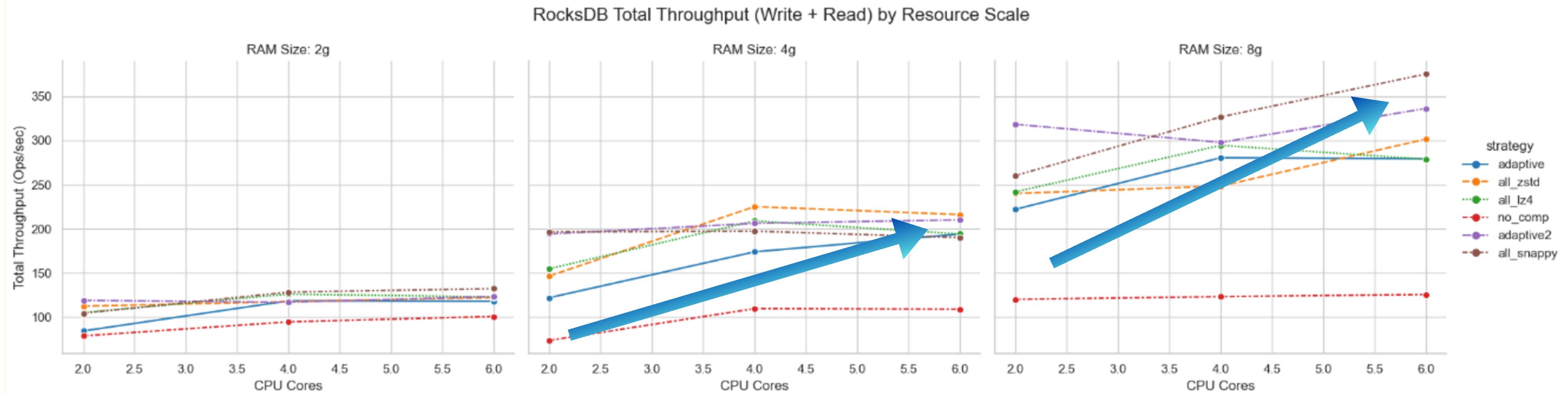


02 가설 1: 적응형 압축



CPU 성능이 높을수록 압축률이 높은 알고리즘의 성능이 높았습니다.

02 가설 1: 적응형 압축



코어의 개수와 전체적인 성능은 비례

04 결론

Discussion

가설 1 결론

CPU의 성능이 향상됨에 따라 Compression을 강하게 하는 것이
전체적인 성능을 향상시킨다

RESEARCH RESULTS

가설 2: 쏠림 현상

03 가설 2: 쓸림 현상

데이터의 레벨별 특성



Cold

- All data is available
- Low cost
- Not performance sensitive



Warm

- Most data is available
- Moderate cost
- Moderate performance



Hot

- Business-critical datasets
- Always online
- Latency extremely important

이미지 출처: <https://rk1993.tistory.com/324>

03 가설 2: 쓸림 현상

쓸림 현상이란?

Hot Data에만 아주 많은 접근이 일어나는 것

실제 서비스에서는 오래된 데이터라도 특정 인기 데이터에 트래픽이 쏠리는 '인기도 기반 쓸림'이 발생한다 (ex: 예: 10년 전 노래의 역주행)

파레토 법칙: 전체 결과의 80%가 전체 원인의 20%에서 일어난다는 '80:20 법칙'

현실 세계의 데이터 요청은 대부분
"20%의 데이터가 80%의 트래픽을 받는다"
파레토 법칙을 따른다



이미지 출처:

<https://magazine.contenta.co/2022/06/%EC%9D%B8%EB%AC%B8%ED%95%99%EC%A0%81-%EA%B4%80%EC%A0%90%EC%97%90%EC%84%9C-%EB%B0%94%EB%9D%BC%EB%B3%B8-%ED%8C%8C%EB%A0%88%ED%86%A0-%EB%B2%95%EC%B9%99/>

03 가설 2: 쓸림 현상

쓰림 현상 연구에 대한 가설

실제 서비스에서는 오래된 데이터라도 특정 인기 데이터(Key)에 트래픽이 쓸리는 '인기도 기반 쓸림(Zipfian Skew)'이 발생하고

이 데이터들은 이미 디스크 최하단에 고압축 상태로 저장됨
따라서 반복적인 압축 해제 비용이 발생

→ Adaptive 전략이 이 상황에서도 성능 저하 없이 버틸 수 있을 것인가?

03 가설 2: 쓸림 현상

연구 방법

vim mixed_compression.ini

```
#설정 파일 만들기  
vim mixed_compression.ini
```

```
[Version]  
rocksdb_version=10.11.0  
  
[DBOptions]  
  
[CFOptions "default"]  
compression_per_level=kNoCompression:kNoCompression:kLZ4Compression:kLZ4Compression:kLZ4Compression:kZSTD:kZSTD:kZSTD
```

03 가설 2: 쓸림 현상

연구 방법

test_adaptive.sh

```
#실행파일 생성  
vim test_adaptive.sh
```

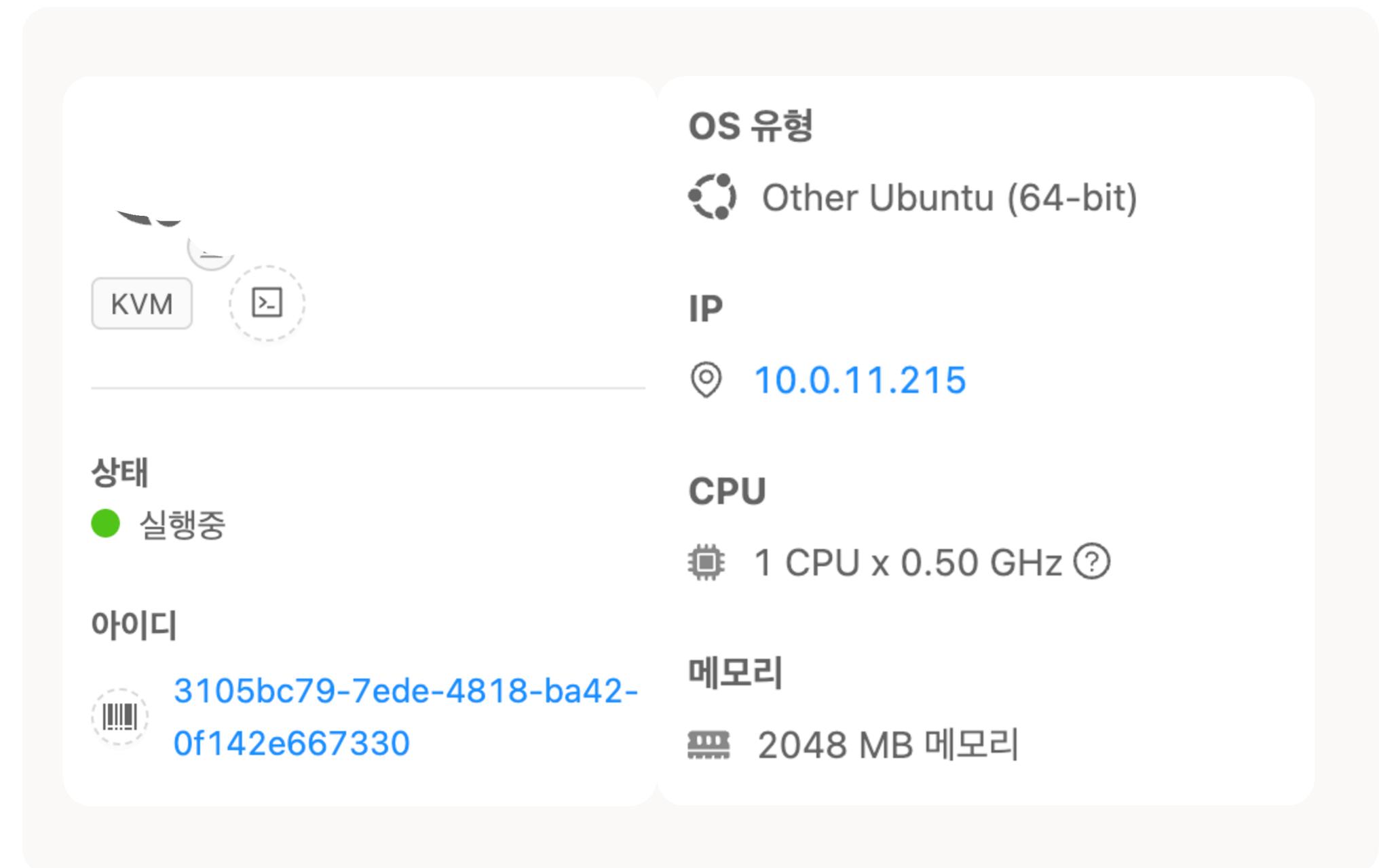
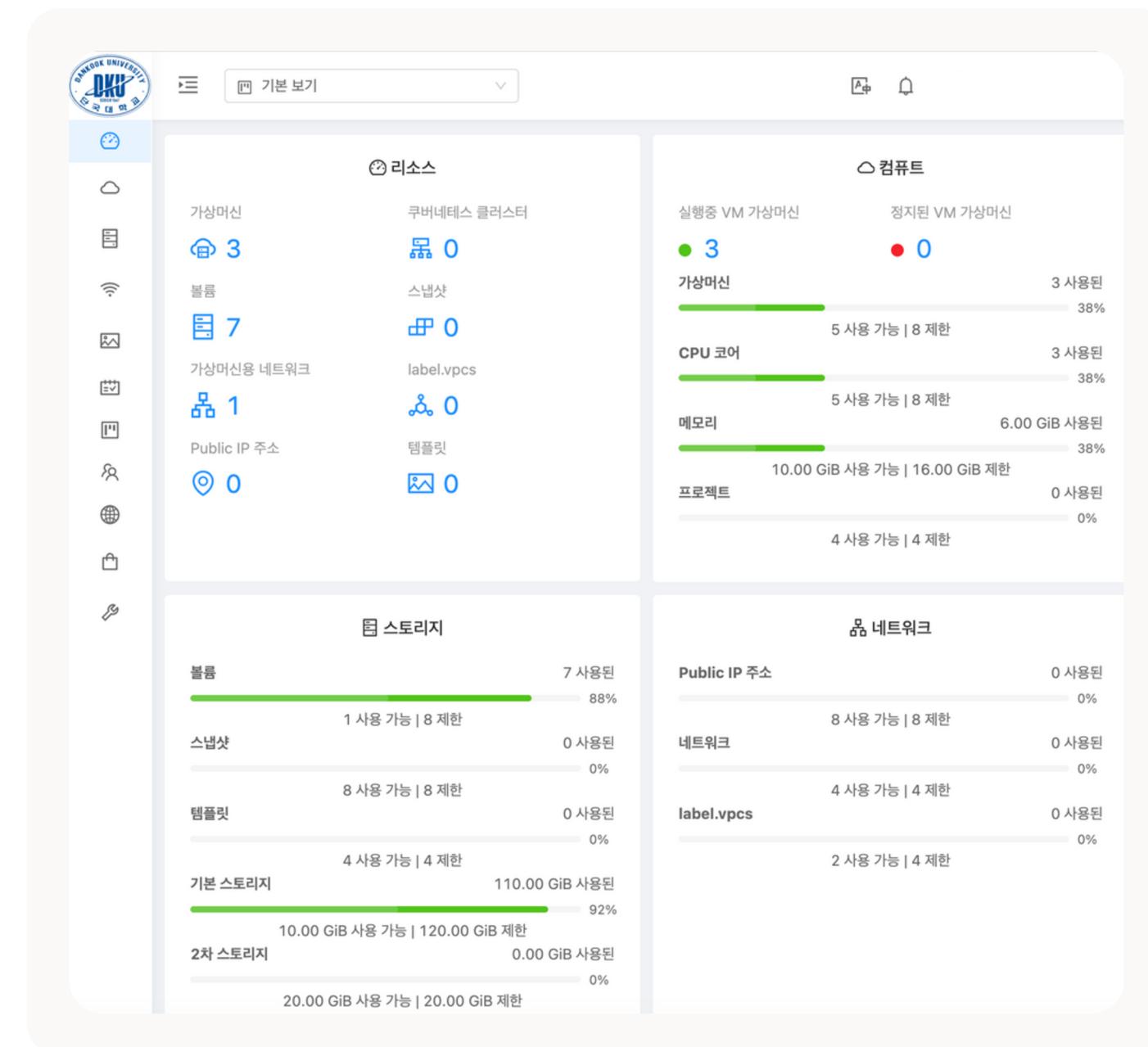
```
echo "start_Adaptive test"  
  
. /db_bench --benchmarks="fillseq,readrandom" \  
--options_file=./mixed_compression.ini \#설정파일로 압축방식 지정  
--num=1000000 \  
--value_size 1024 \  
--statistics=1 \  
--read_random_exp_range=200000 \  
--cache_size=268435456 \  
| grep -E 'readrandom|P50|P99'
```

--read_random_exp_range=200000 \

03 가설 2: 쓸림 현상

실험 환경

서버: <https://dku.kloud.zone>



03 가설 2: 쓸림 현상

가설 2: 쓸림 현상

실험 과정

진행한
압축 전략

#1. All none

#2. All zstd

#3. All LZ4

#4. none none LZ4 LZ4 LZ4 ZSTD ZSTD

#5. none none ZSTD ZSTD ZSTD ZSTD ZSTD

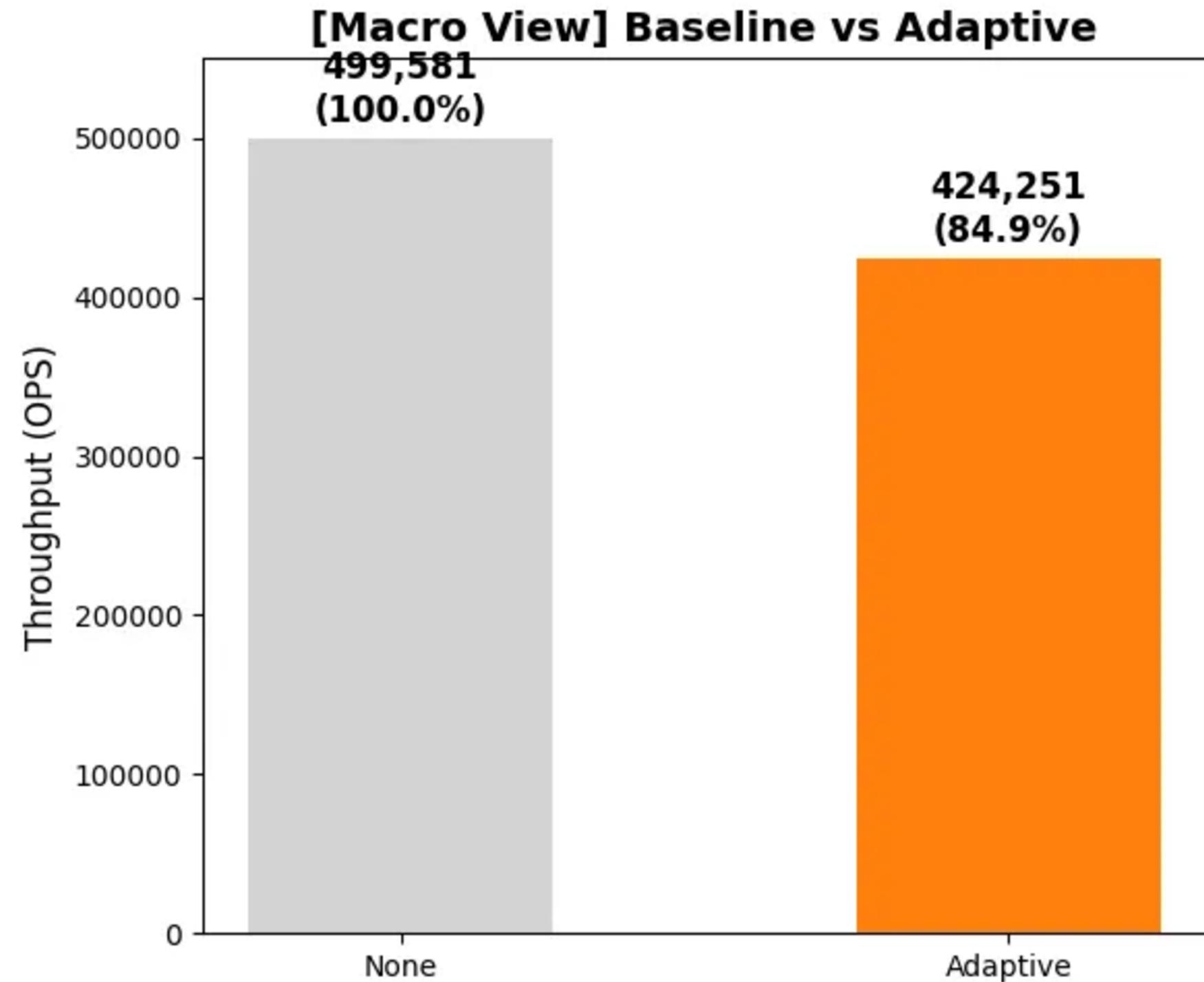
실험 결과

03 가설 2: 쓸림 현상

실험결과

none 압축과 adaptive 압축 비교

압축 전략	처리량 (OPS)	지연 시간 (Latency)	성능 비율 (vs None)
None	499,581	2.00 μ s	100% (Baseline)
Adaptive (L0~1:None, L2~:Zstd)	424,251	2.36 μ s	85%

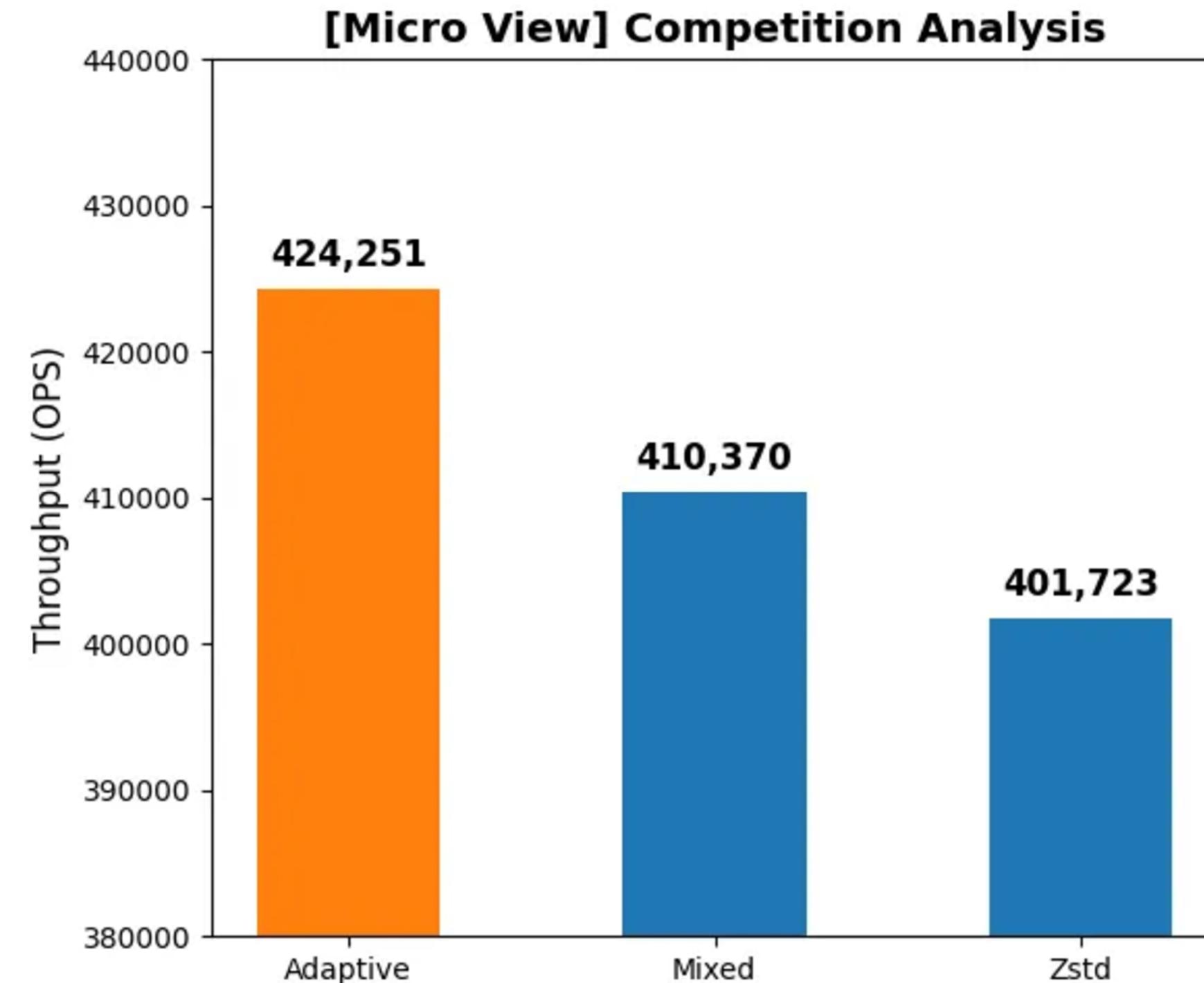


03 가설 2: 쓸림 현상

실험결과

Adaptive와 Mixed(adaptive), Zstd 비교

압축 전략	처리량 (OPS)	지연 시간 (Latency)	성능 비율 (vs None)
Adaptive (L0~1:None, L2~:Zstd)	424,251	2.36 μ s	85%
Mixed (LZ4 + Zstd)	410,370	2.44 μ s	82%
All Zstd (전체 Zstd)	401,723	2.49 μ s	80%



03 가설 2: 쓸림 현상

Discussion

Adaptive 전략의 유효성 검증

읽기 성능은 Block Cache를 활용해 None
방식과 대등한 최고 속도를 낸다

캐시가 밭쳐주는 한,
hot 데이터에 대해 마음껏 압축을 해도 성능 손실이 없다

04 결론 및 향후 계획

향후 계획

단기적 과제: 시스템 내부 효율 최적화

압축 방식에 따른 캐시 효율성 분석 :

Compressed VS Uncompressed 블록 캐시의 성능 트레이드오프
검증

메모리 리소스 배분 최적화 :

write_buffer_size와 인덱스 블록 크기 비율에 따른 처리율 최적 지점 도출

장기적 과제: 클라우드 네이티브 환경으로의 확장

Qos 기반 Compaction 스케줄링 :

K8s 등 공유 인프라 환경에서 컨테이너별 리소스 제한과 연동된 적응형
컴팩션 연구

감사합니다

RocksDB의 LSM-Tree 레벨별 특성을
고려한 적응형 압축(Adaptive Compression)

[Storage Optimization]

딥스토리 (DBStory) 팀

소프트웨어학과 이기윤
국제경영학과 홍사인
소프트웨어학과 노승아
소프트웨어학과 성진욱