비대칭 트래픽에 강한 데이터베이스

koreachoi96@gmail.com
Presentation by Minguk, Choi

Background

- 유저 트래픽 패턴
- LSM-tree 구조

Motivation

- 비대칭 트래픽 패턴에서의 LSM-tree 성능저하

Design

- 캐시 최적화를 통한 중복 트래픽 패턴에서의 LSM-tree 성능향상

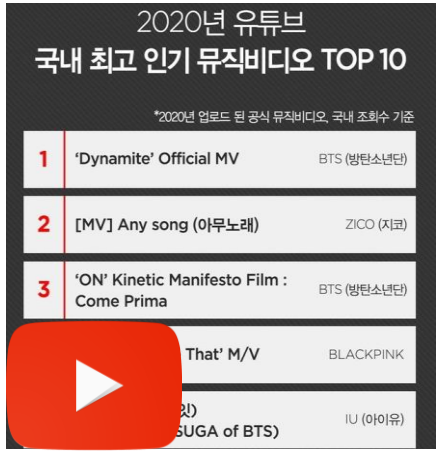
Benchmarks

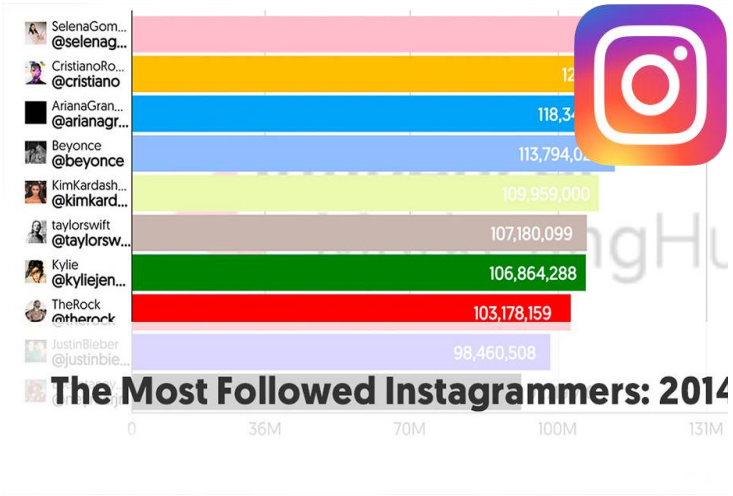
- YCSB: Zipfian Insert

Background 1.

유저 트래픽 패턴

유저 트래픽 분포





유저들은 사실
인기있는 소수의 데이터만을
좋아하고 관심 있음

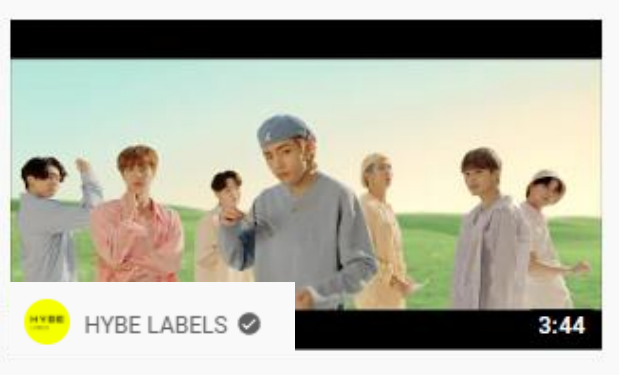
따라서 유저 트래픽은
극단적으로 소수의 인기있는
데이터에만 집중됨

유저 트래픽 분포

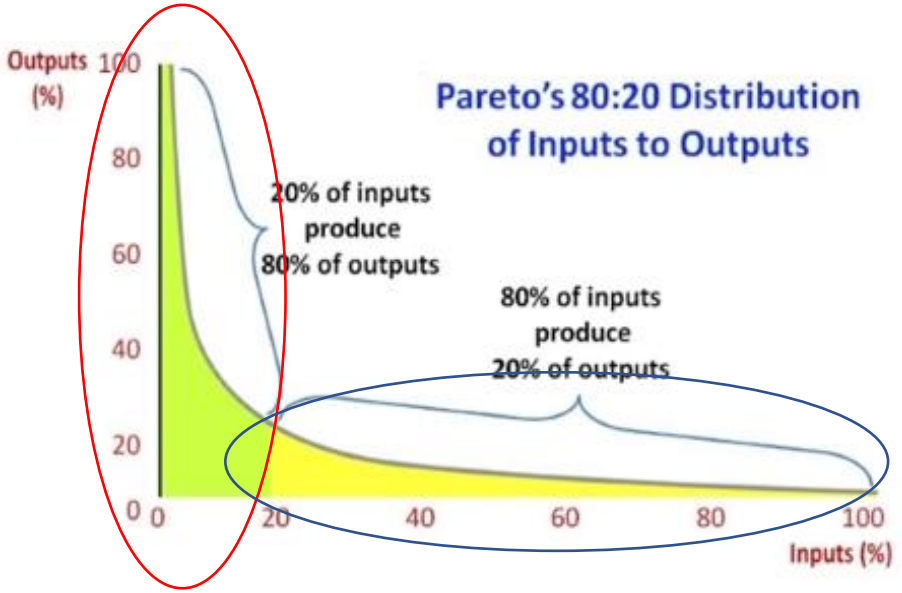
접근빈도가 높은
20%의 데이터가
80%의 트래픽를 차지

통계학
- 지피안 분포(Zipfian Distribution)
- 파레토 분포(Paretero Distribution)

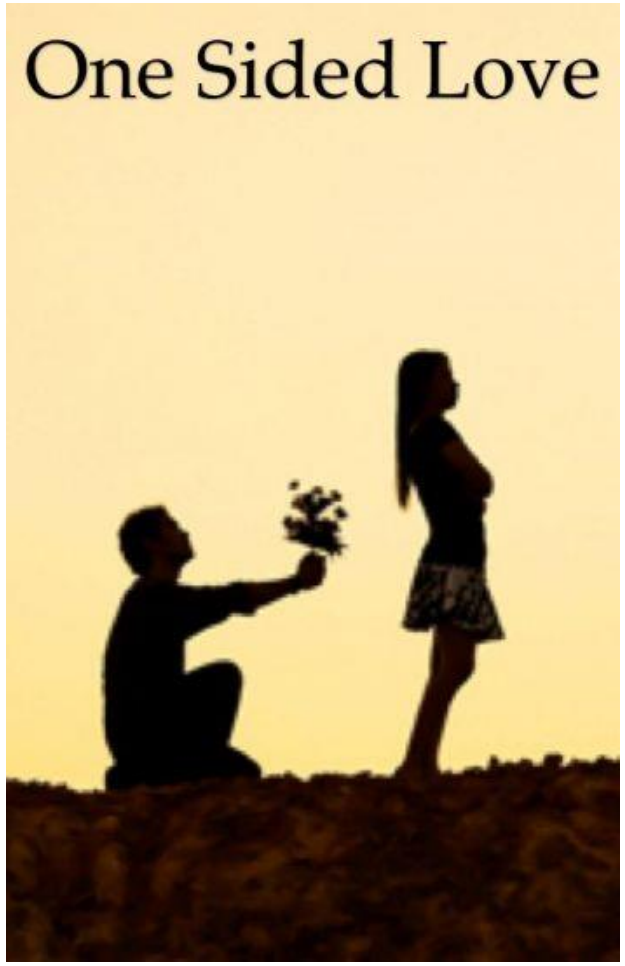
접근빈도 낮은,
80%의 데이터가
20%의 트래픽를 차지



BTS (방탄소년단)
Official MV
조회수 14억회 • 1년 전



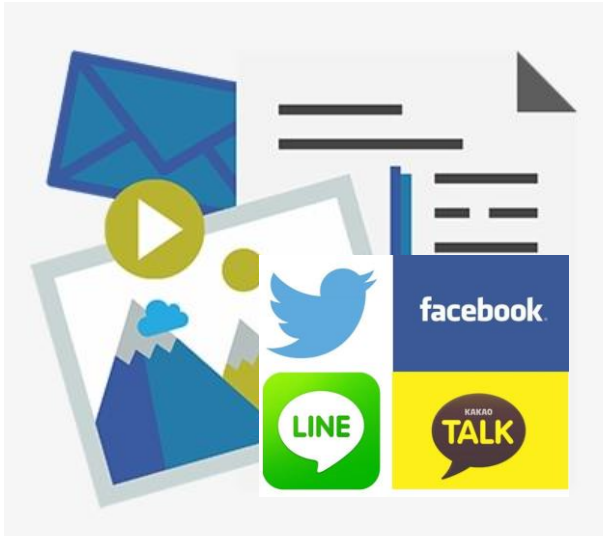
설립자 합동 추모식 및
조회수 125회 • 4개월 전



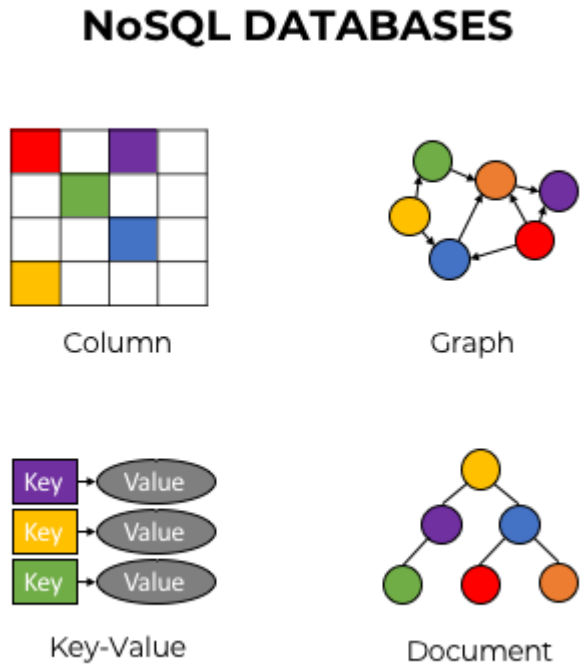
Background 2.

SSD에 최적화된
LSM-tree 구조

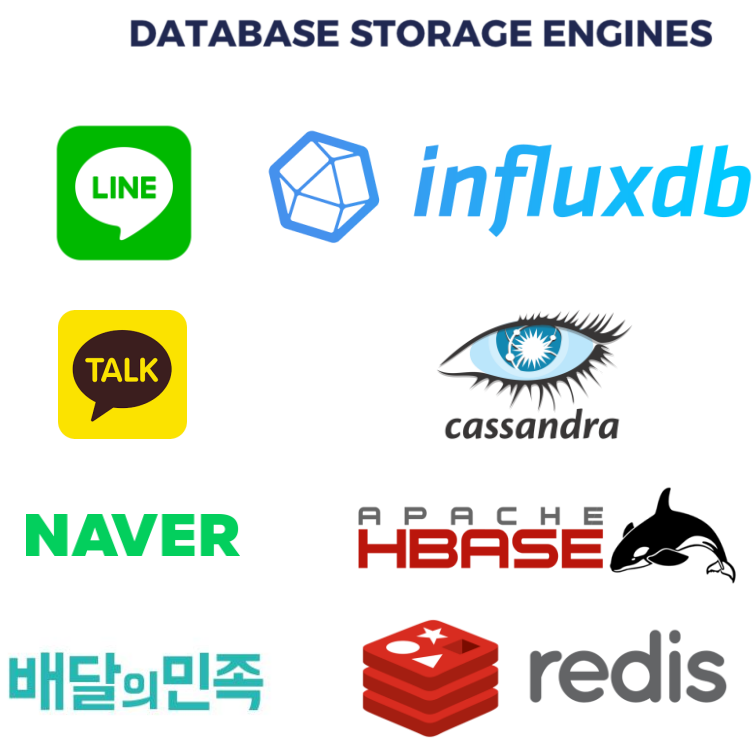
서버 DB



비정형 데이터



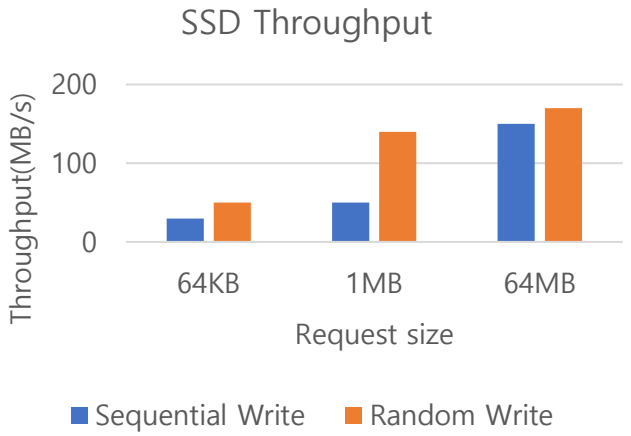
NoSQL DB



LSM-tree

LSM-tree 구조

Hardware
SSD

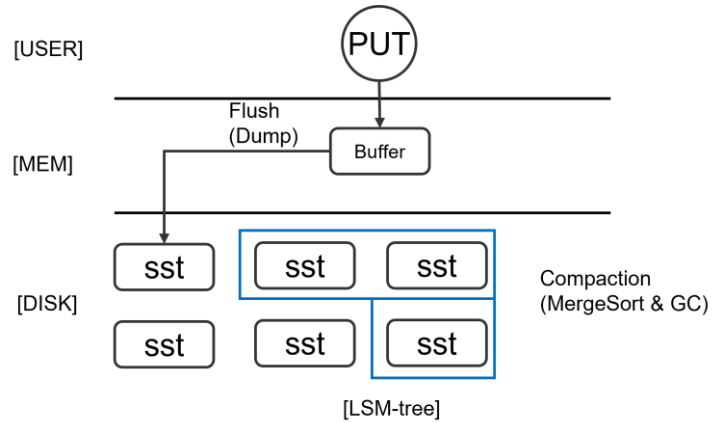


1. 최대한 한번에 모아 쓸 수록 성능이 좋음
2. 순서대로 쓸 수록 성능이 좋음
3. 업데이트 시, 덮어쓰지 않고 새로 씀
   (out-place-update, not In- place-update)

=

소프트웨어가
하드웨어의
행동양식을 따라하여
성능을 극대화

Software
LSM-tree

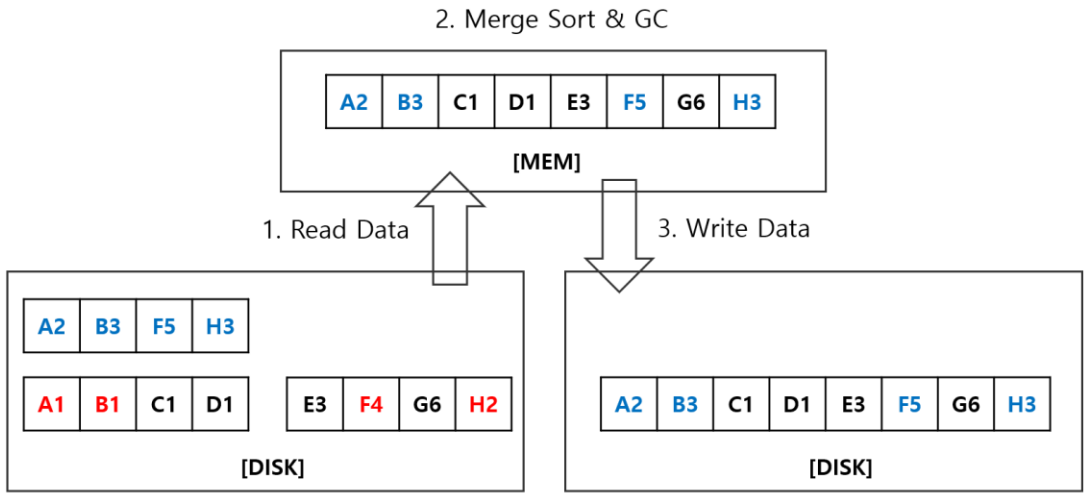


1. Flush
1) 버퍼에 데이터를 64MB씩 모았다가 한번에 쓴다.
2) 버퍼에 데이터를 순서대로 모은다.

2. Compaction
업데이트 시, 덮어쓰지 않고 새로 씀
(out-place-update, not In- place-update)

Compaction으로 인한 LSM-tree의 성능 저하

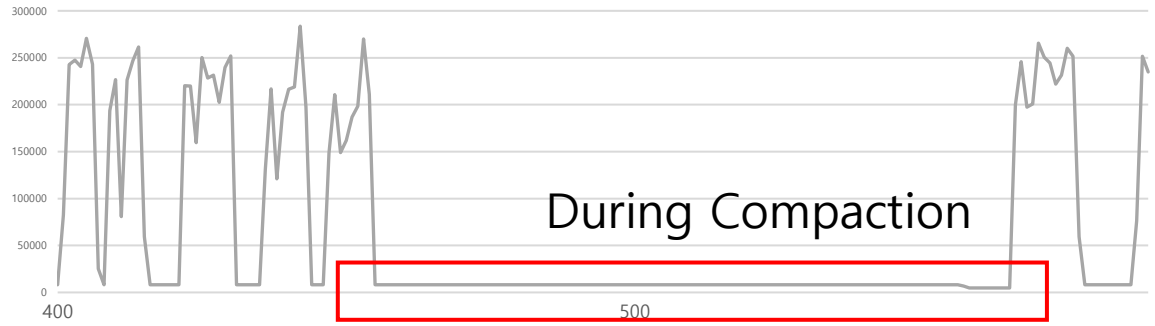
[Compaction]



LSM-tree는 성능 최적화를 위해 SSD와 동일하게
In-Place가 아닌 Out-Place 방식으로 Update

하지만 Compaction Overhead가 존재

[DB Throughput]



Compaction은 굉장히 비싸고 오래 걸리는 작업

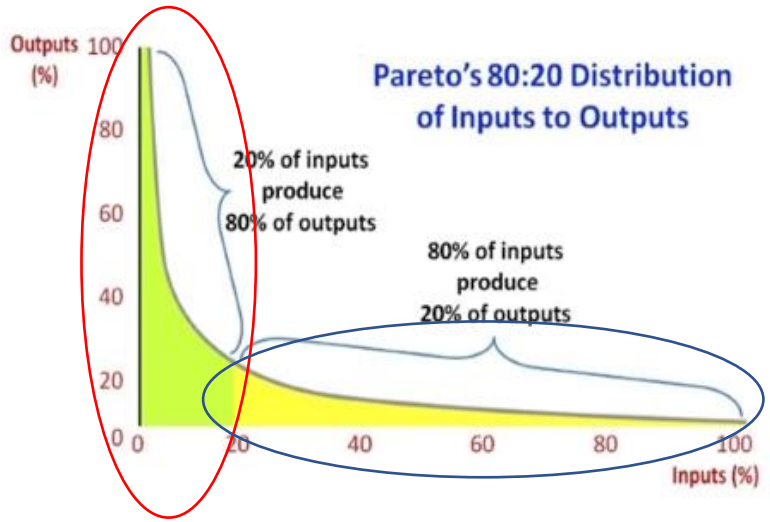
Compaction 동안 DB는 유저 Request를 응답하지 못하기도 함

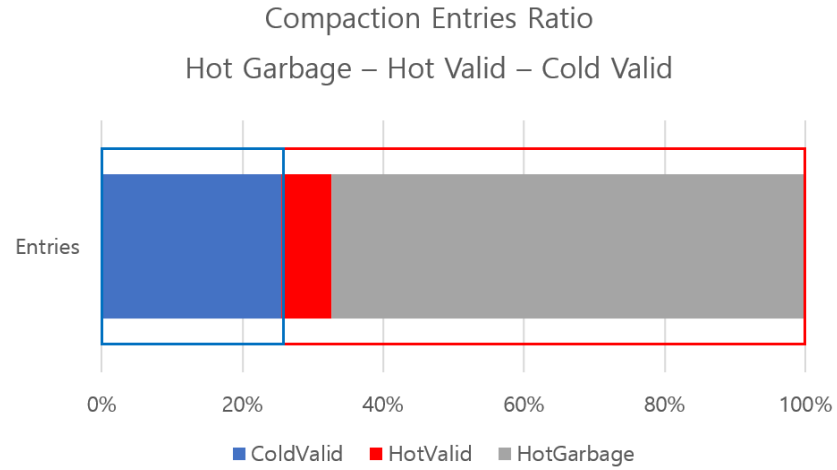
Why?
Read/Write I/O는 컴퓨터에서 가장 오래 걸리는 작업
Merge Sort, GC 또한 상당한 CPU Overhead를 일으킴

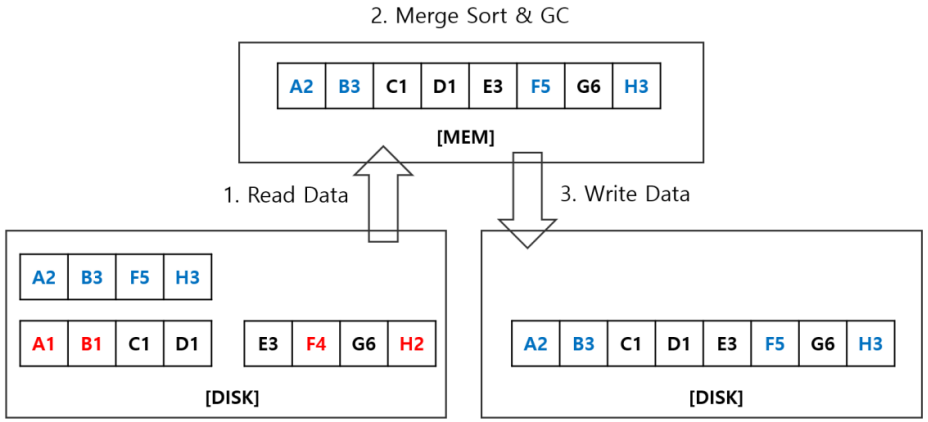
Motivation.

중복 유저 트래픽에서의
LSM-tree 성능 저하

Compaction으로 인한, Zipfian 패턴에서의 성능 저하



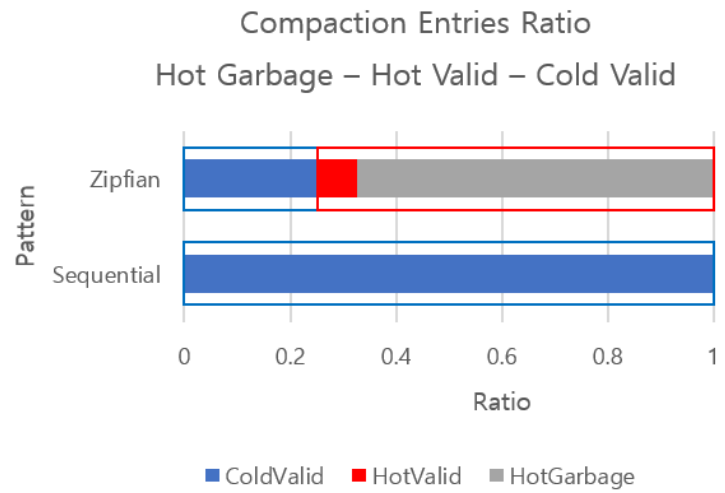


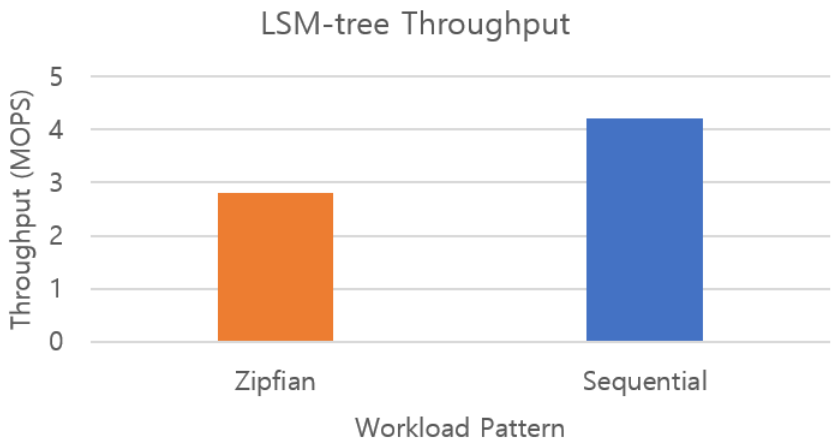


유저 트래픽은 Zipfian분포
- 20%의 데이터에 80%의 트래픽이 집중
- 하지만 80%의 트래픽 중, 90%는 의미 없는 Garbage

Compaction 대상 중 Garbage가 많을 수록
의미 없는 데이터를 더 많이 읽고, 정렬하고, 써야 하므로
Compaction Overhead가 증가

Compaction으로 인한, Zipfian 패턴에서의 성능 저하

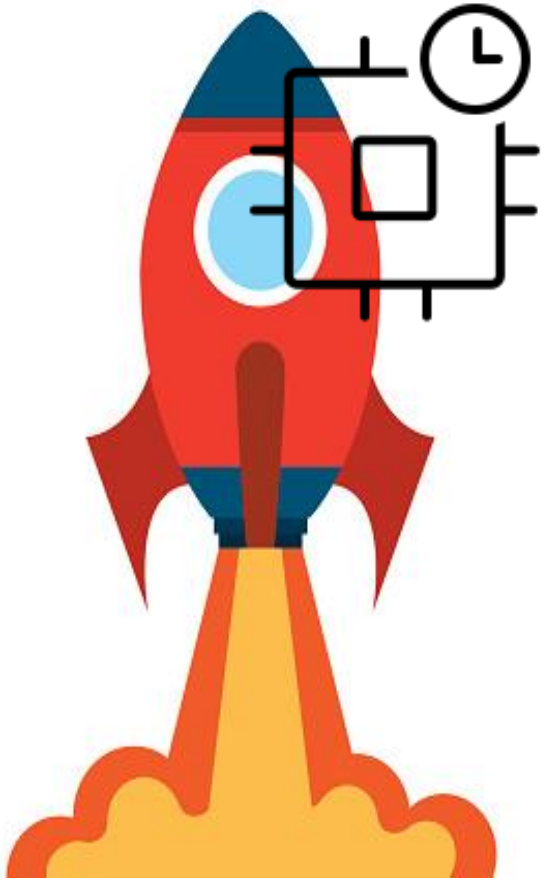




- 트래픽
  - Sequential: 중복 없음, Garbage 없음
  - Zipfian: 중복 많음, Garbage 많음

- 성능
  - Sequential > Zipfian

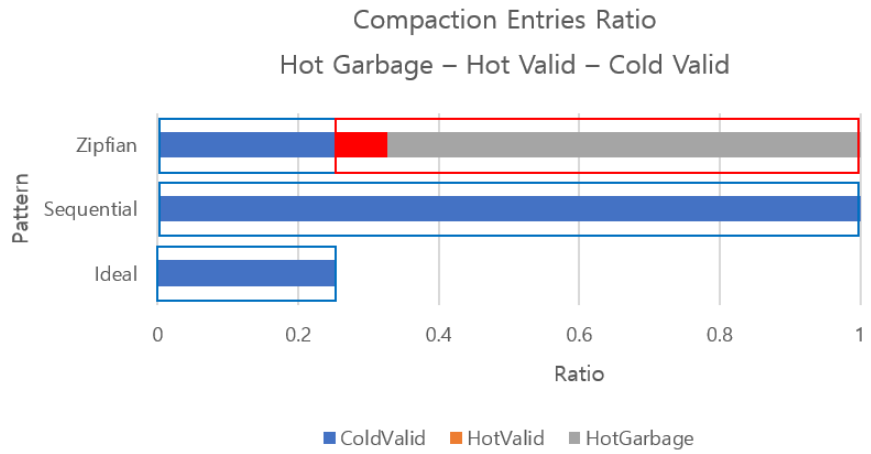
- Zipfian 패턴에서의 성능 향상 방법
  - Compaction Overhead를 일으키는 Garbage를 줄여야 함

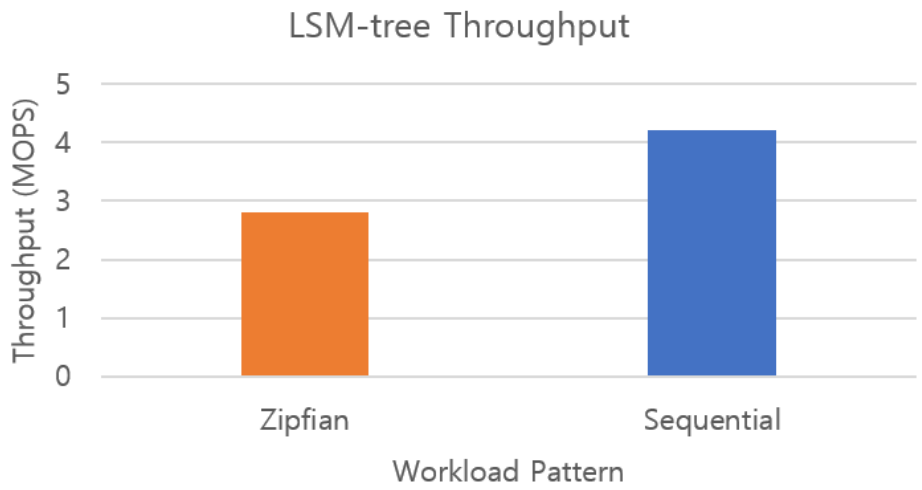


Design.

캐시를 통한
비대칭 트래픽 패턴에서의
LSM-tree 성능 향상

New Design, LSM-tree+Cache



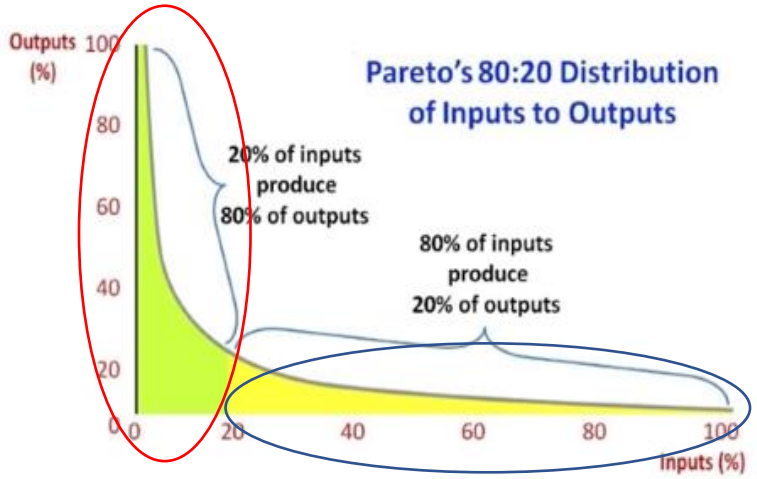


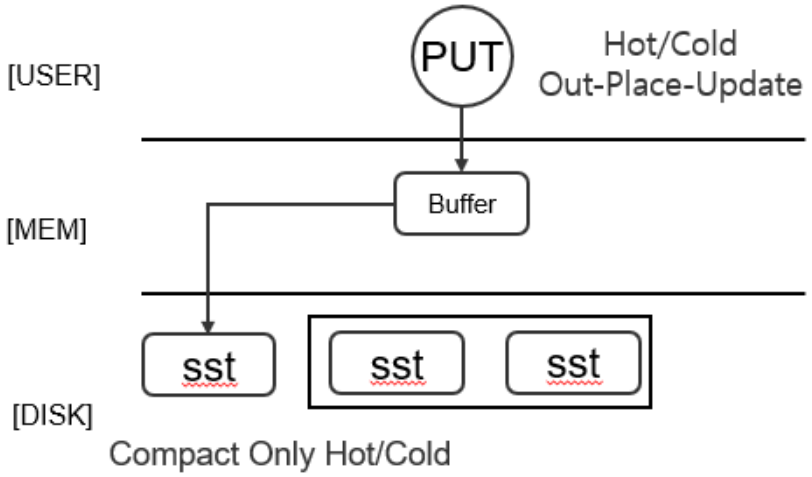
Garbage가 적을수록, Compaction Overhead가 적다.

빈도 높은 데이터의 Update로 인한 Garbage를 줄여야 한다.
빈도 높은 데이터를 Compaction, LSM-tree에서 제외시키자

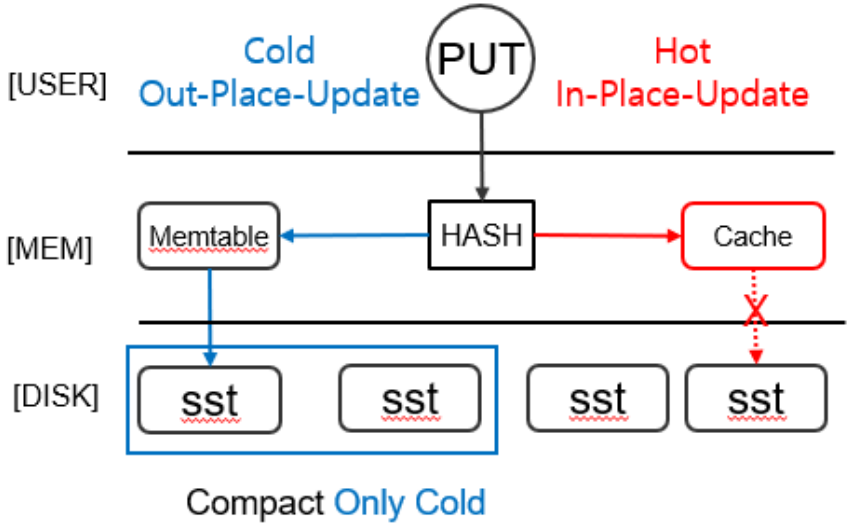
빈도 낮은 데이터는 기존대로 LSM-tree에서 관리
빈도 높은 데이터는 Cache로 관리

New Design, LSM-tree+Cache

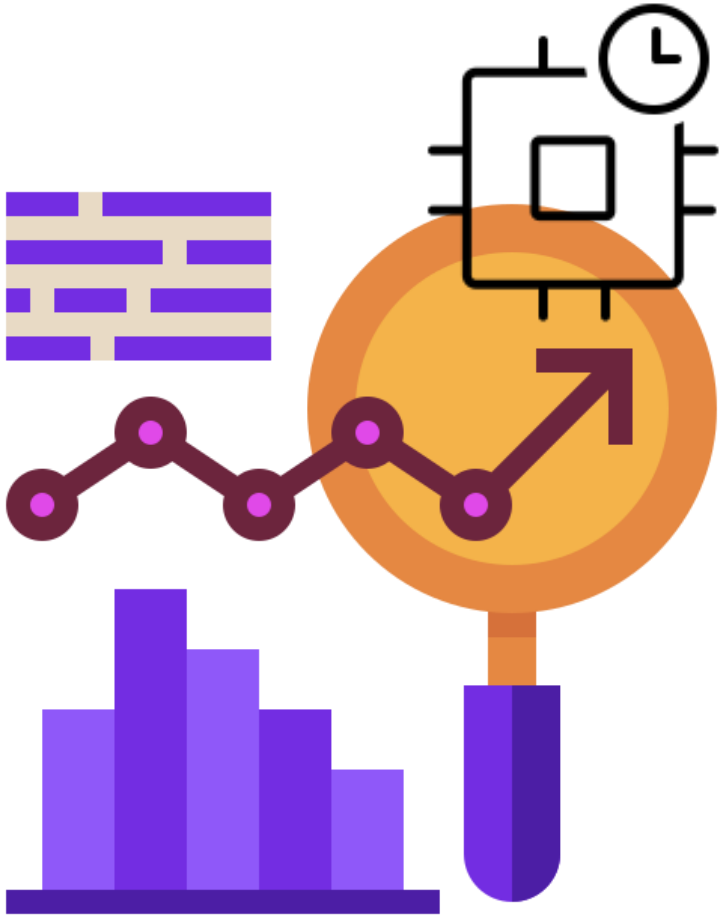




Traditional LSM-tree



Frequency Hotness Aware LSM-tree



Benchmark.

성능 분석

Benchmarks

brianfrankcooper / YCSB Public

Yahoo! Cloud Serving Benchmark

Apache-2.0 License

4.1k stars 2k forks

Star Notifications

Workload	Ratio (in %) of each operation				Dist.
	Read	Modify	Scan	RMW	
(A) Write Heavy	50	50	-	-	Zipfian
(B) Read Heavy	95	5	-	-	Zipfian
(C) Read-Only	100	0	-	-	Zipfian
(D) Read Latest	95	5	-	-	Latest
(E) Short Scans	0	5	95	-	Zipfian
(F) Read-Modify	50	0	-	50	Zipfian

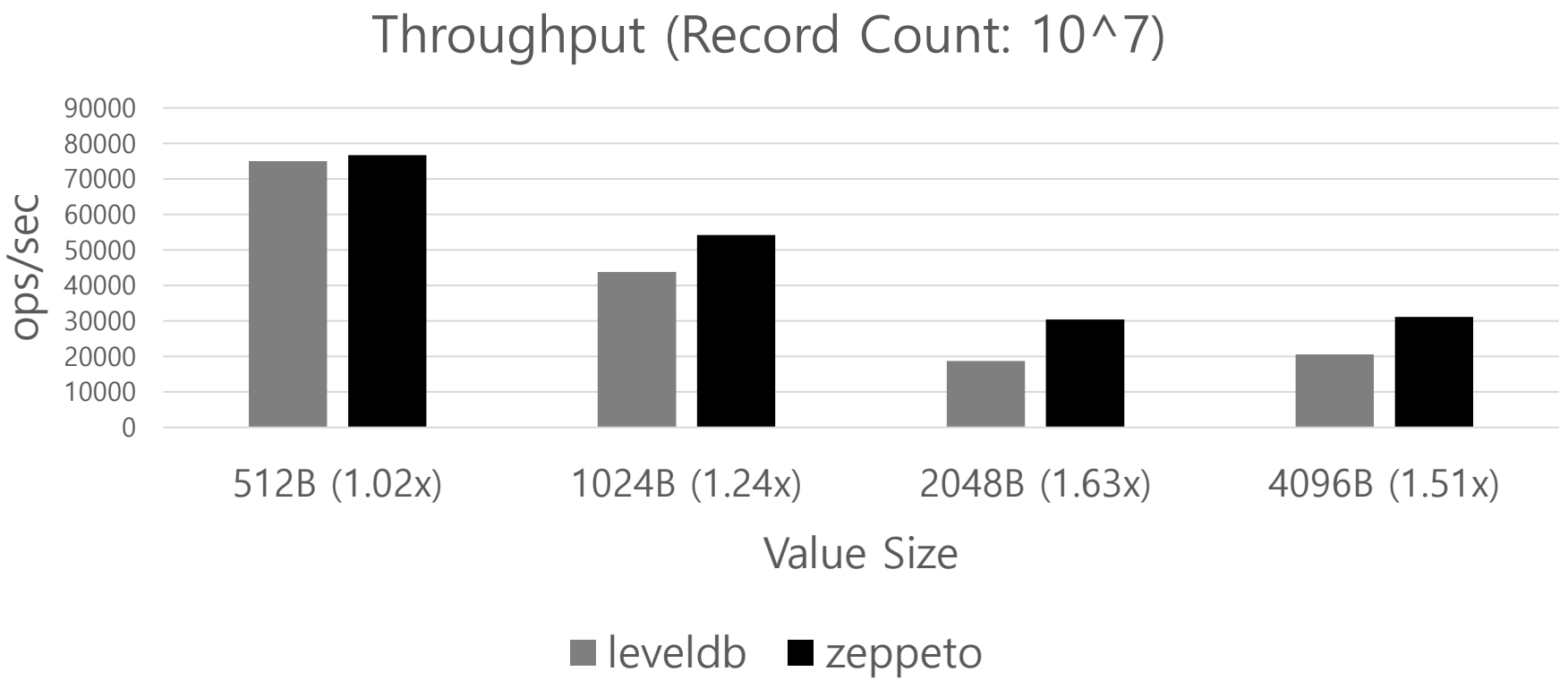
Benchmarking cloud serving systems with **YCSB**

BF Cooper, A Silberstein, E Tam... - Proceedings of the 1st ..., 2010 - dl.acm.org

... An important aspect of the **YCSB** framework is its extensibility: ... The **YCSB** framework and workloads are available in open ... In this paper, we describe the **YCSB** benchmark, and report ...

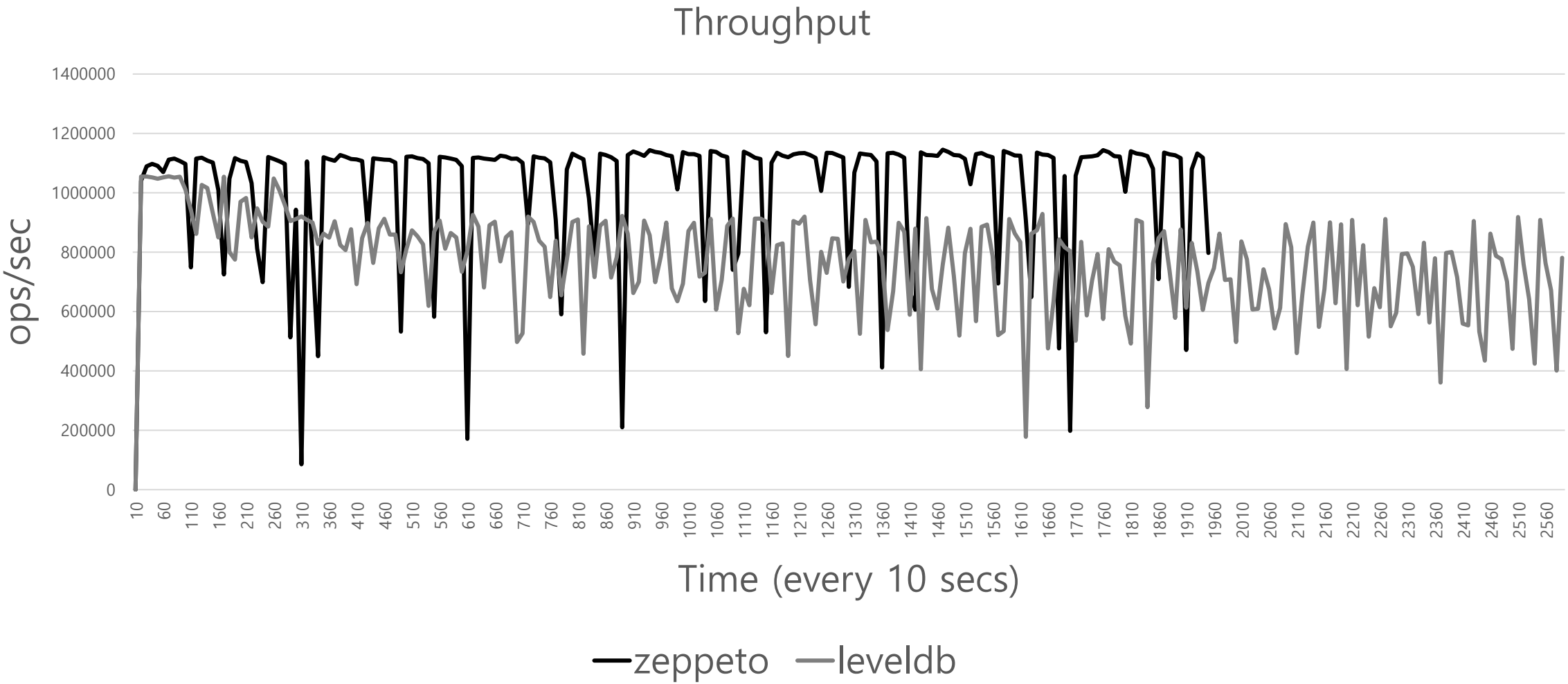
Save Cite Cited by 3754 Related articles All 22 versions

Benchmarks: Zipfian Insert

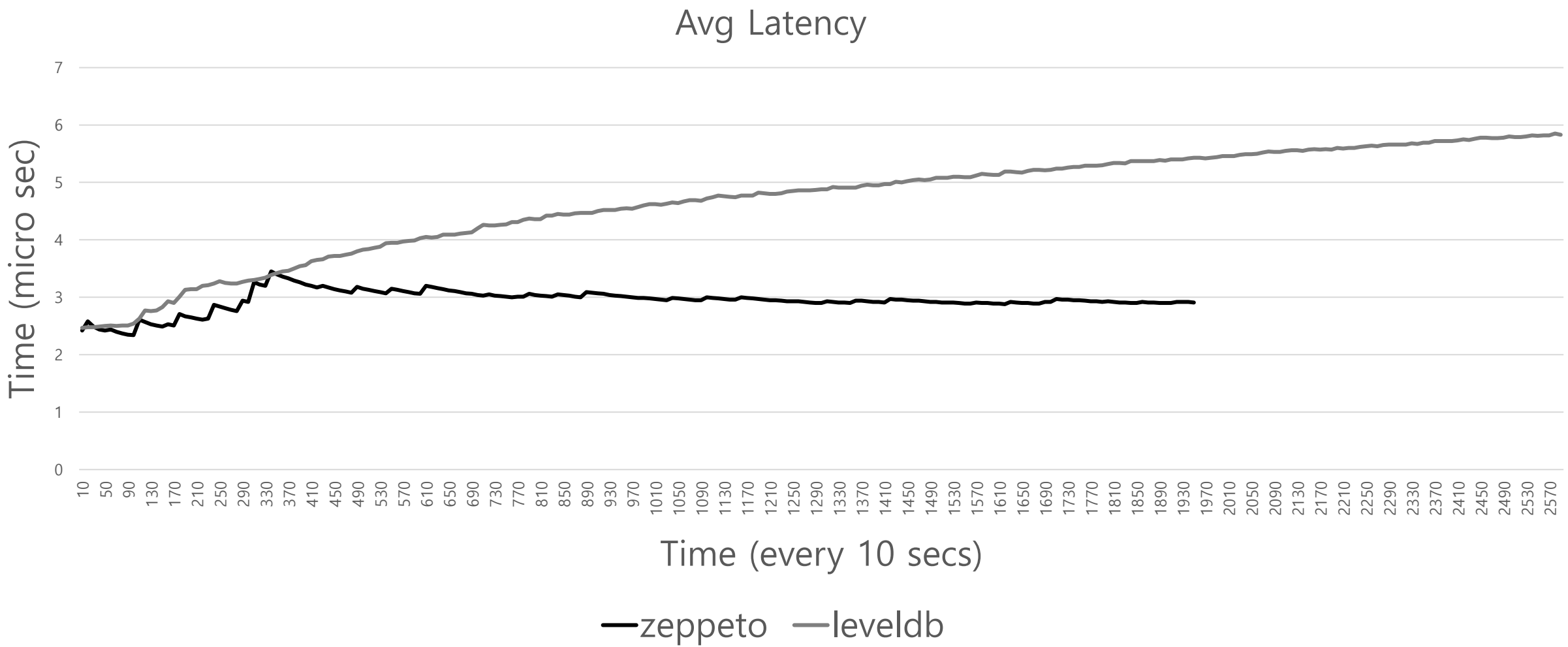


Value Size	512B	1024B	2048B	4096B
Throughput	1.02x	1.24x	1.63x	1.51x

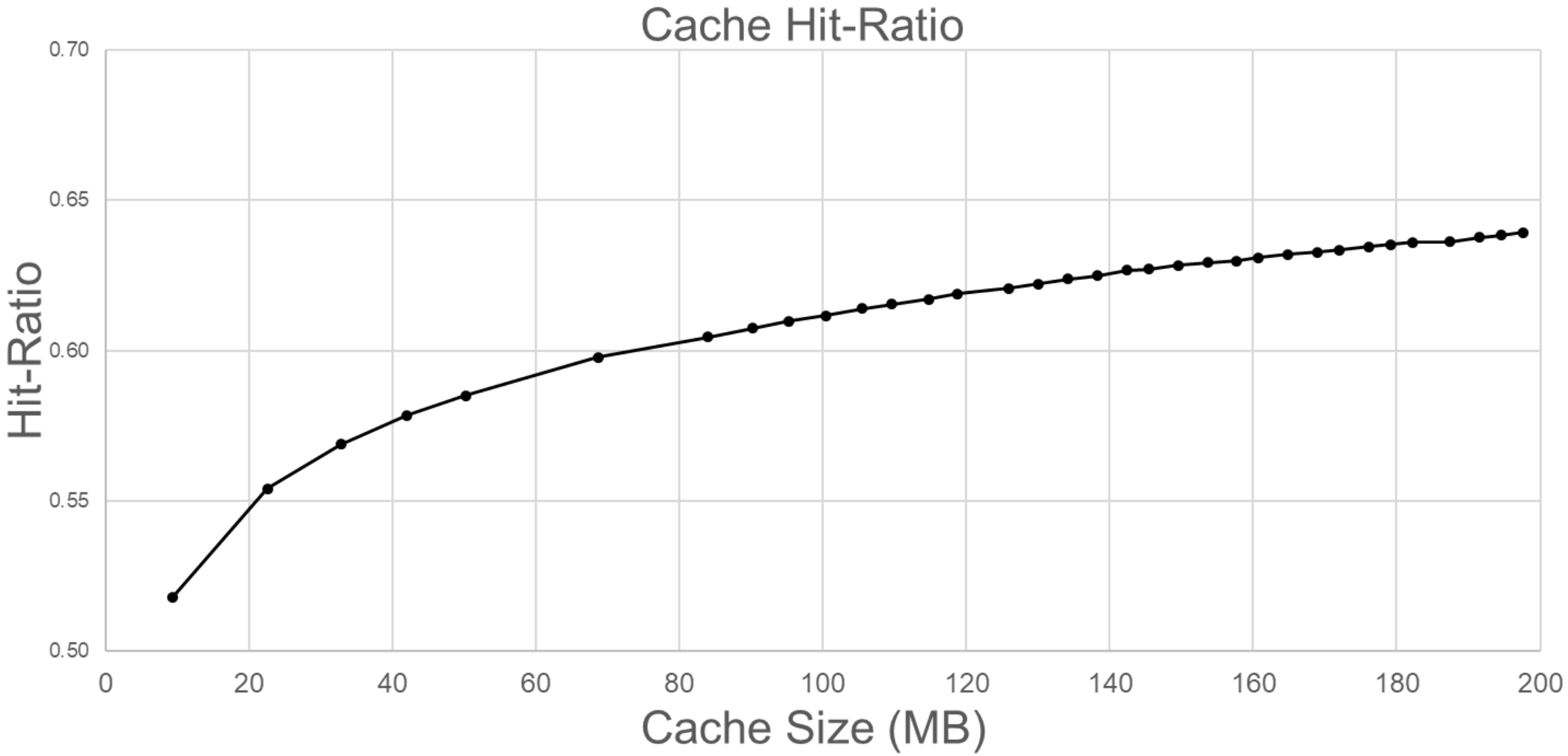
Benchmarks


Throughput
ops/sec
1400000
1200000
1000000
800000
600000
400000
200000
0
10 60 110 160 210 260 310 360 410 460 510 560 610 660 710 760 810 860 910 960 1010 1060 1110 1160 1210 1260 1310 1360 1410 1460 1510 1560 1610 1660 1710 1760 1810 1860 1910 1960 2010 2060 2110 2160 2210 2260 2310 2360 2410 2460 2510 2560
Time (every 10 secs)
zeppeto
leveldb

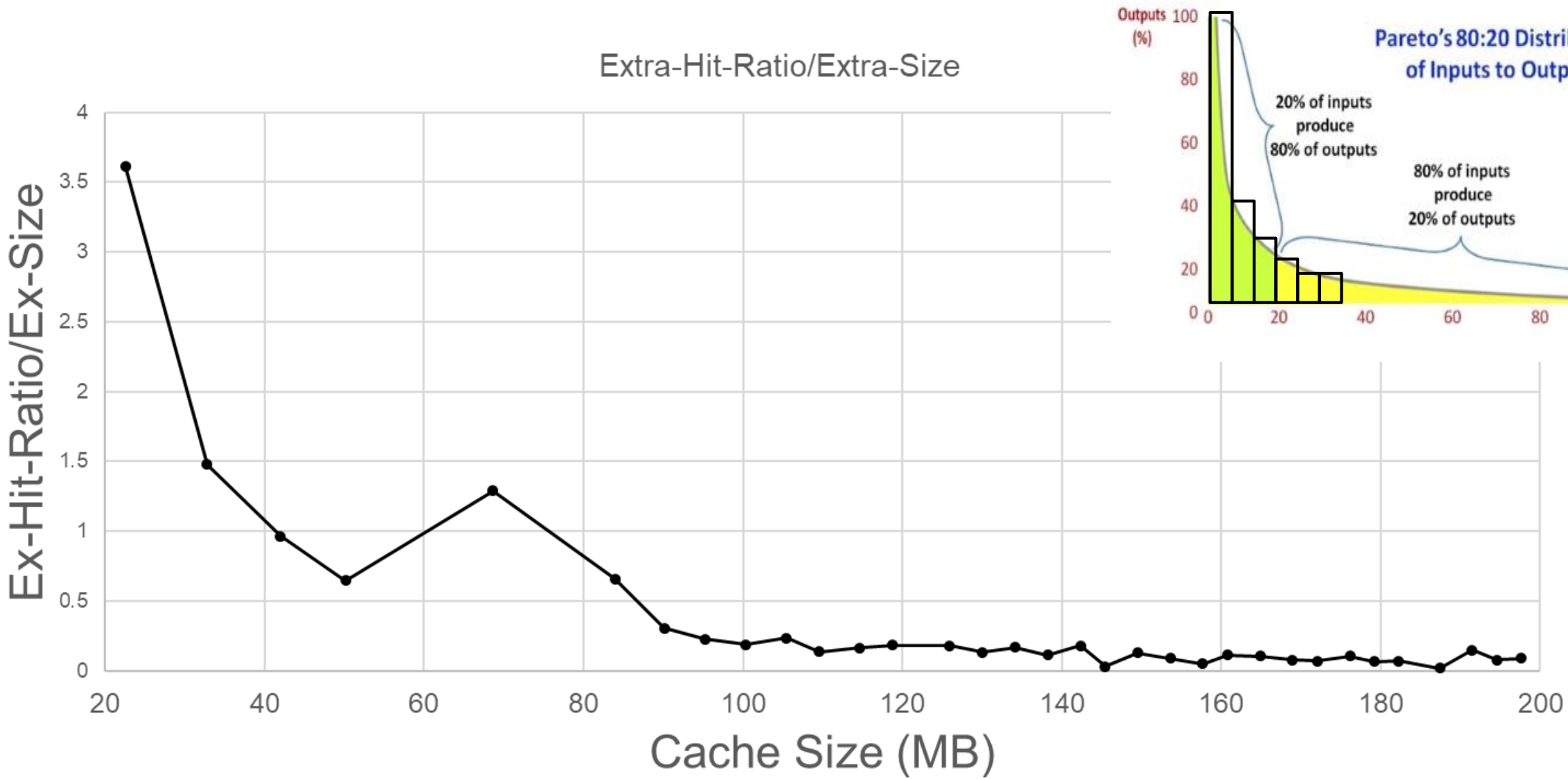
Benchmarks

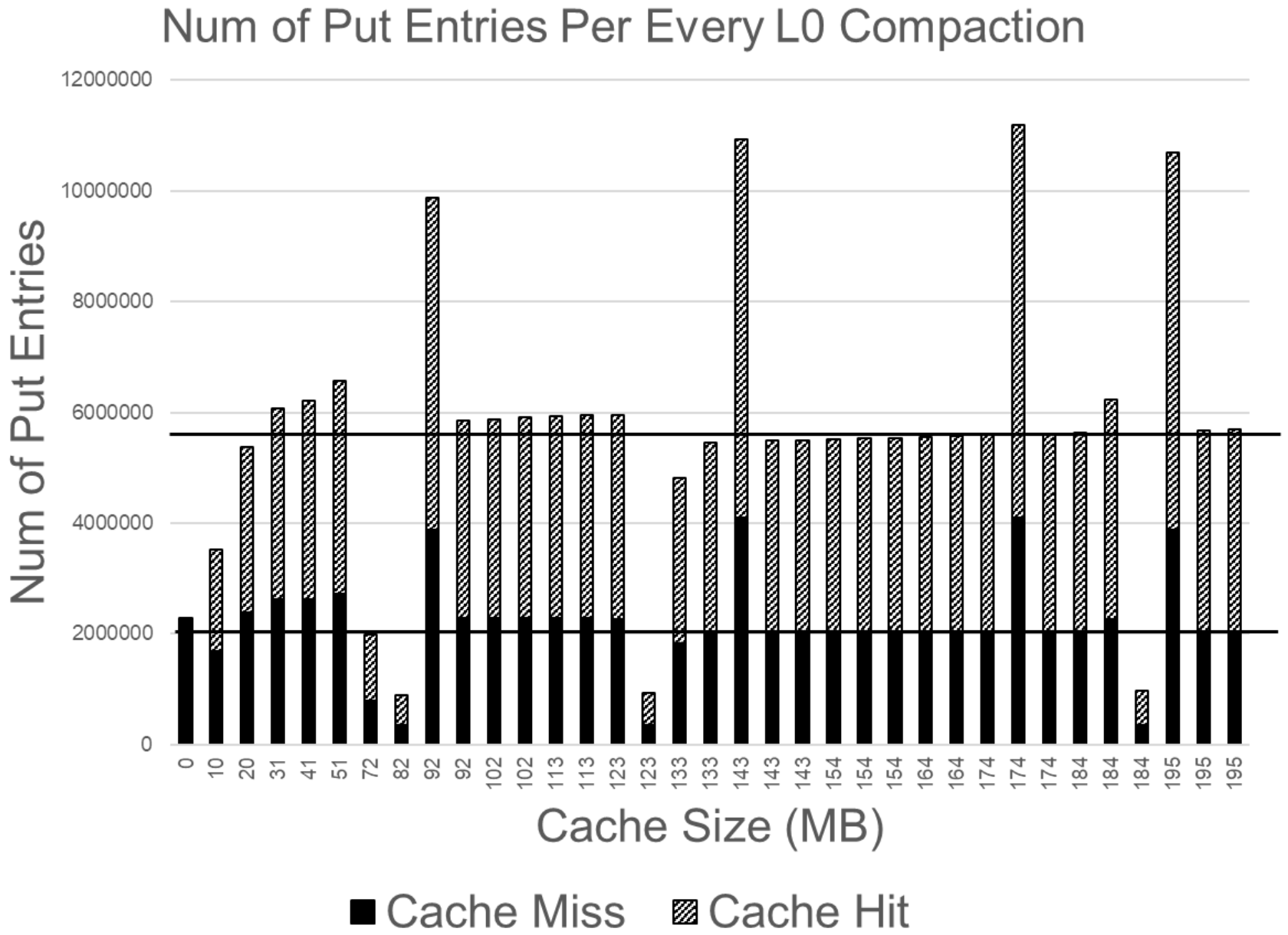

Avg Latency
Time (micro sec)
0
1
2
3
4
5
6
7
10 50 90 130 170 210 250 290 330 370 410 450 490 530 570 610 650 690 730 770 810 850 890 930 970 1010 1050 1090 1130 1170 1210 1250 1290 1330 1370 1410 1450 1490 1530 1570 1610 1650 1690 1730 1770 1810 1850 1890 1930 1970 2010 2050 2090 2130 2170 2210 2250 2290 2330 2370 2410 2450 2490 2530 2570
Time (every 10 secs)
zeppeto
leveldb

Benchmarks

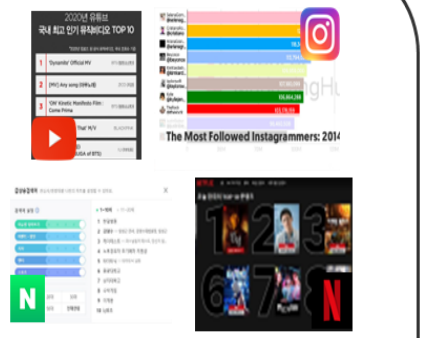

Cache Hit-Ratio
Hit-Ratio
0.70
0.65
0.60
0.55
0.50
0
20
40
60
80
100
120
140
160
180
200
Cache Size (MB)

Benchmarks

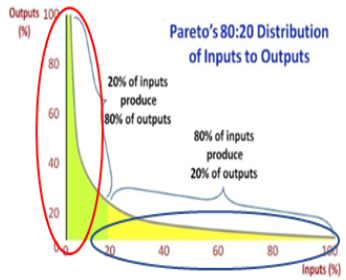

Extra-Hit-Ratio/Extra-Size
Ex-Hit-Ratio/Ex-Size
Cache Size (MB)
0
0.5
1
1.5
2
2.5
3
3.5
4
20
40
60
80
100
120
140
160
180
200
Outputs (%)
Inputs (%)
Pareto's 80:20 Distribution of Inputs to Outputs
20% of inputs produce 80% of outputs
80% of inputs produce 20% of outputs


Num of Put Entries Per Every L0 Compaction
Num of Put Entries
12000000
10000000
8000000
6000000
4000000
2000000
0
0 10 20 31 41 51 72 82 92 92 102 102 113 113 123 123 133 133 143 143 143 154 154 154 164 164 174 174 174 184 184 184 195 195 195
Cache Size (MB)
Cache Miss
Cache Hit

요약



[유저 트래픽]

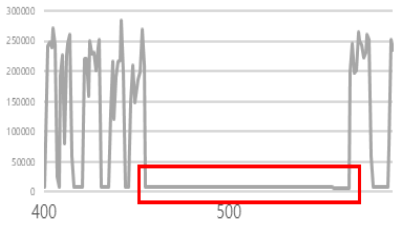


[지피안 분포]

1. 유저 트래픽

- Zipfian분포
- 20%의 데이터에 80%의 트래픽이 집중

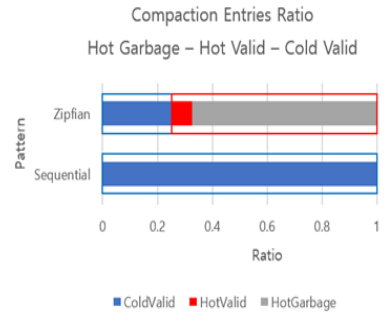
[Compaction]



[Compaction Overhead]

2. LSM-tree

- Write Optimized
- SSD Optimized
- Compaction Overhead

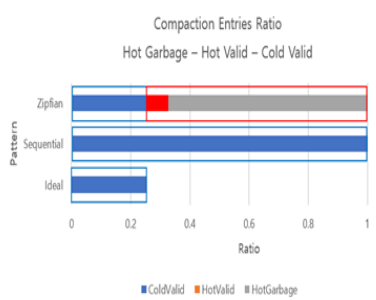


[Garbage 비율]

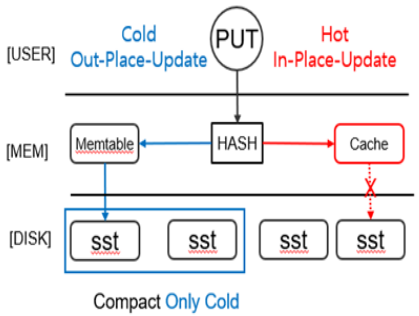
[낮은 지피안 성능]

3. Zipfian 분포에서의 LSM-tree 성능 저하

- Hot Garbage로 인한, Compaction Overhead 증가

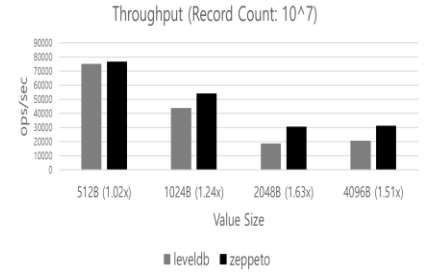


[캐시 후, Garbage 비율]

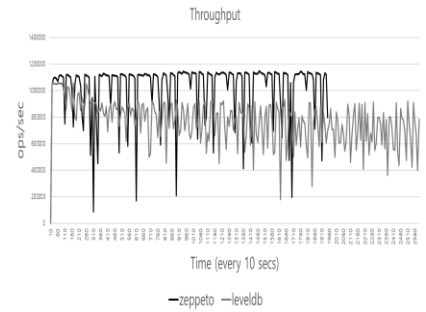


4. Frequency Hotness Aware LSM-tree

- Hot: Cache
- Cold: LSM-tree



[Value Size별 성능]



[Throughput Spike]

5. Benchmark

- Write Performance
- Throughput Spike