RocksDB Festival

Supported by IITP, StarLab.

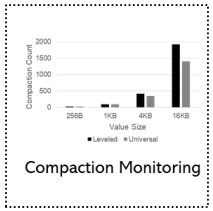
August 23, 2021

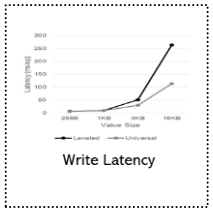
송인호, 한예진
inhoinno@dankook.ac.kr , hbb97225@naver.com
TeamName : 멘탈모델을 만들고 싶어요

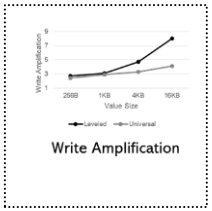
Contents

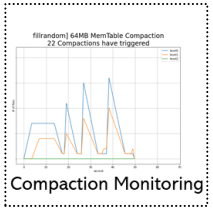
- Last Week
  - ✓ Leveled vs Universal Compaction Comparison
    - 13일 meeting feedback 정리
    - KSC 논문 그림 실험 재현
    - Read Amplification

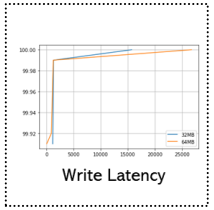
- This Week
  - 관련 논문 조사
  - Compaction 코드 리뷰
  - KSC 논문 작성 진행

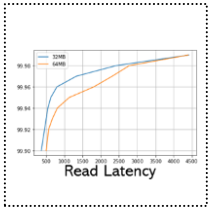












Read Amplification

- How to estimate Read Amplification Factor?

$$RA = \frac{\text{READ\_AMP\_TOTAL\_READ\_BYTES}}{\text{READ\_AMP\_ESTIMATE\_USEFUL\_BYTES}}$$

$$= \frac{\text{실제 } read\text{에 사용된 총 } byte \text{ 수}}{load\text{된 } data\ block\text{의 총 } byte \text{ 수}}$$

include/rocksdb/statistics.h

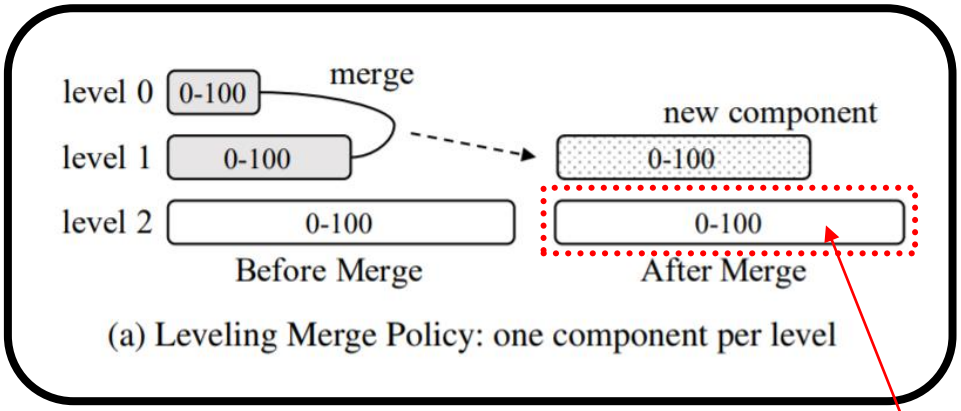
```
// Read amplification statistics.
// Read amplification can be calculated using this formula
// (READ_AMP_TOTAL_READ_BYTES / READ_AMP_ESTIMATE_USEFUL_BYTES)
//
// REQUIRES: ReadOptions::read_amp_bytes_per_bit to be enabled
READ_AMP_ESTIMATE_USEFUL_BYTES,  // Estimate of total bytes actually used.
READ_AMP_TOTAL_READ_BYTES,       // Total size of loaded data blocks.
```

include/rocksdb/table.h

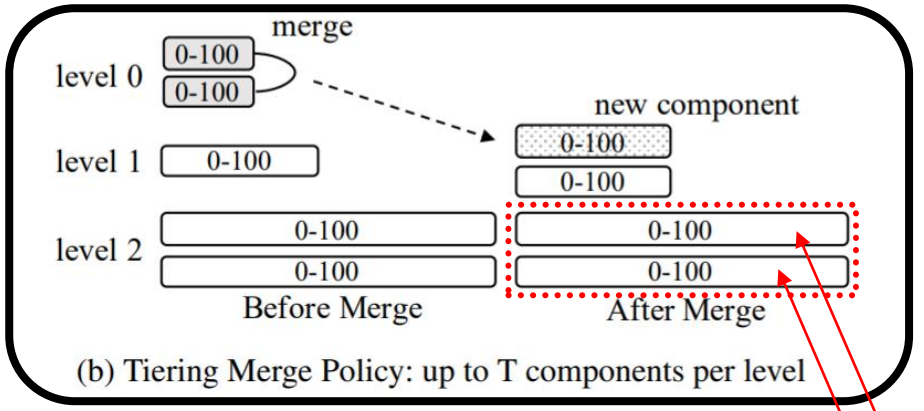
```
// If used, For every data block we load into memory, we will create a bitmap
// of size ((block_size / `read_amp_bytes_per_bit`) / 8) bytes. This bitmap
// will be used to figure out the percentage we actually read of the blocks.

// value  =>  memory usage (percentage of loaded blocks memory)
// 1      =>  12.50 %
// 2      =>  06.25 %
// 4      =>  03.12 %
// 8      =>  01.56 %        // Default: 0 (disabled)
// 16     =>  00.78 %        uint32_t read_amp_bytes_per_bit = 0;
```

LVL vs Univ Read Amplification



Leveled Compaction

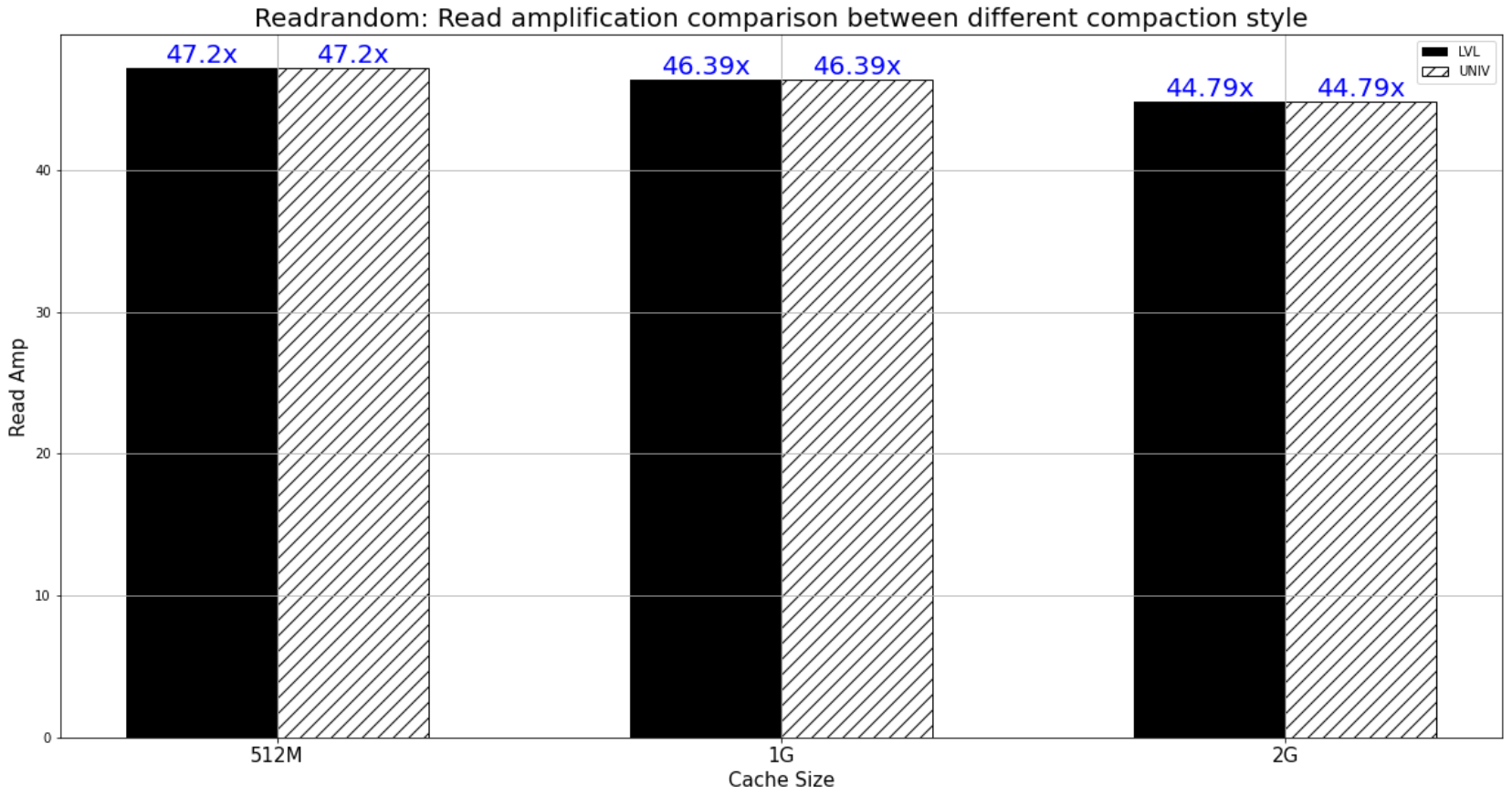


Tiered Compaction
(≠ ***RocksDB Universal Compaction***)

- Readrandom
  - ✓ No difference…. (→ Due to Bloom filter)
- Range query
  - ✓ Big difference !!!

LVL vs Univ Read Amflification: Readrandom

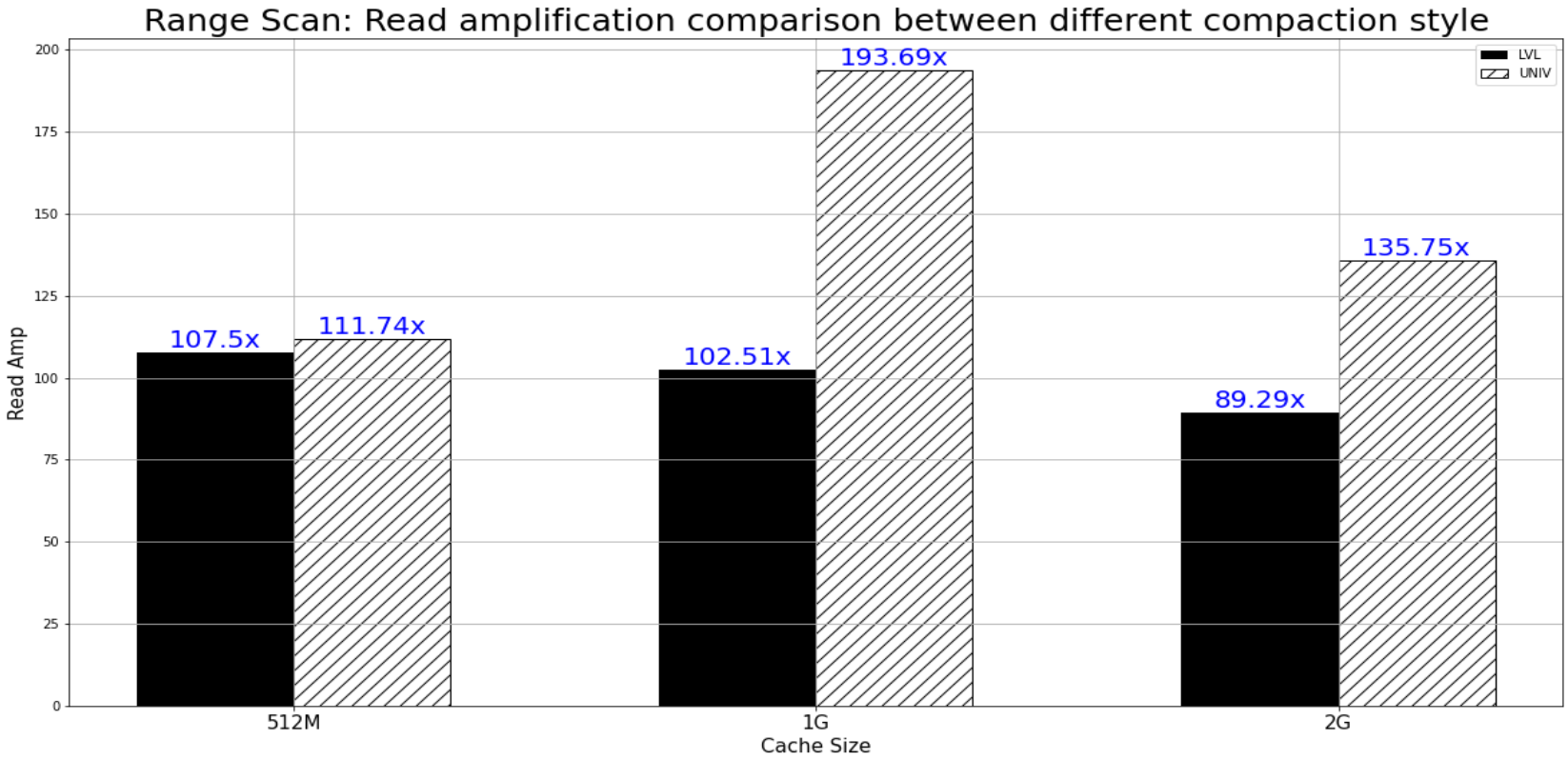
- Readrandom result :: Read amplification
  - ✓ (K16, V64 readrandom, ~10GB)



☞ Bloom filters are useful for single-key lookups (“Is key 42 in SSTable?”)
☞ No difference between LVL and Univ compaction style.

LVL vs Univ Read Amplification: Rangequery

- Range scan result :: Read amplification
  - ✓ (K16, V64, seek random, ~10GB)

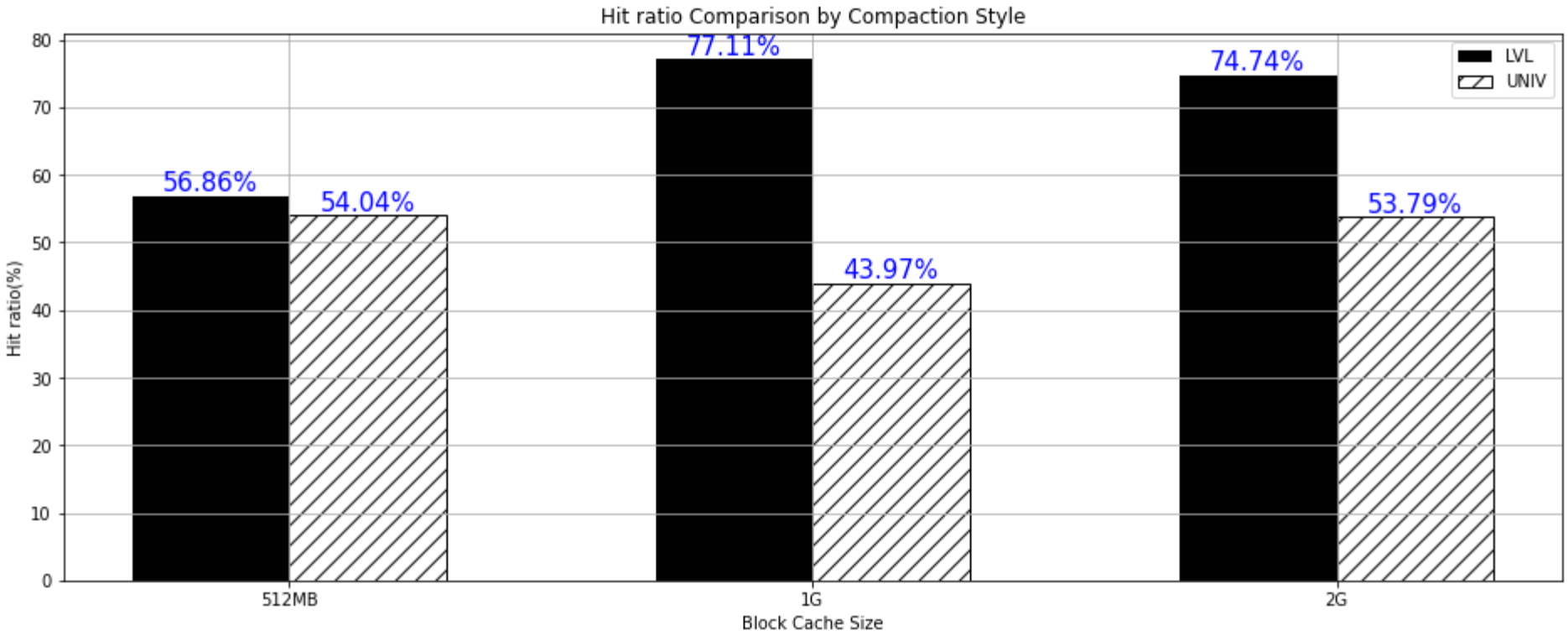


☞ Bloom filters cannot handle range queries
("Are there keys between 42 and 1000 in the SSTable?")

☞ Big difference between LVL and Univ compaction style

LVL vs Univ Read Amplification: Rangequery

- Range scan result :: Hit ratio
  - ✓ (K16, V64, seek random, ~10GB)



- ☞ Bloom filters cannot handle range queries
  ("Are there keys between 42 and 1000 in the SSTable?")
- ☞ Big difference between LVL and Univ compaction style

LVL vs Univ Space Amplification

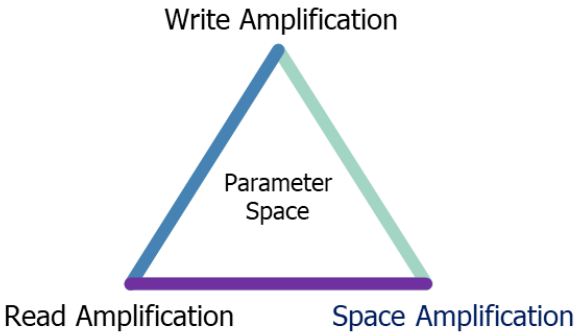
- How to estimate Space Amplification Factor?

Analyzing Space-Amplification. We define space-amplification as the factor *amp* by which the overall number of entries N is greater than the number of unique entries *unq*: $amp = \frac{N}{unq} - 1$.

Space-Amplification. Levels 1 to $L-1$ contain a fraction of $\frac{1}{T}$ of the dataset, and so they may render up to this fraction of entries obsolete at the largest level. In Level L, at most $Z-1$ of the runs may be completely filled with obsolete entries. We model space-amplification as the sum of these terms in Equation 13.

$$amp = Z - 1 + \frac{1}{T} \tag{13}$$

Niv Dayan, Stratos Idreos, Dostoevsky: BetterSpace-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging, **SIGMOD '18**



Discussion