RocksDB Festival

RF3_Team_WAL

Supported by IITP, StarLab.

Aug. 30, 2021

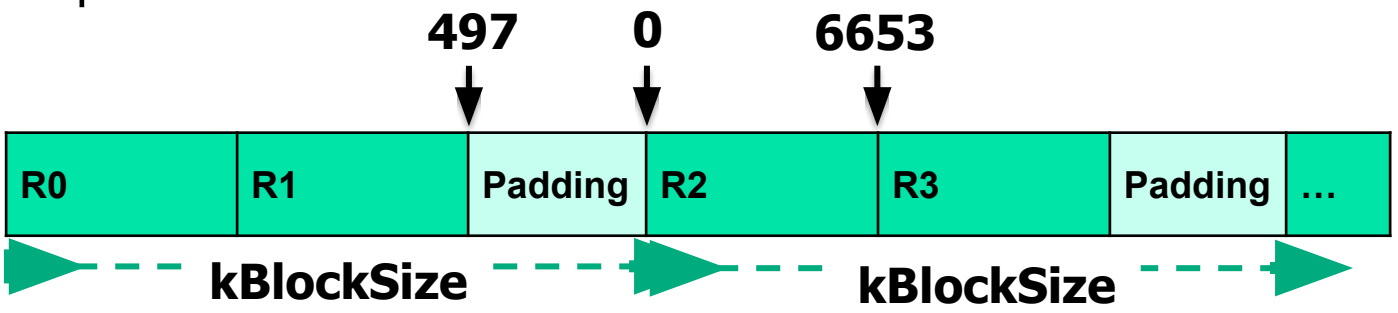
김민준, 이빈
alswnssl0528@naver.com,
32183118@dankook.ac.kr

Rocksdb : insert padding

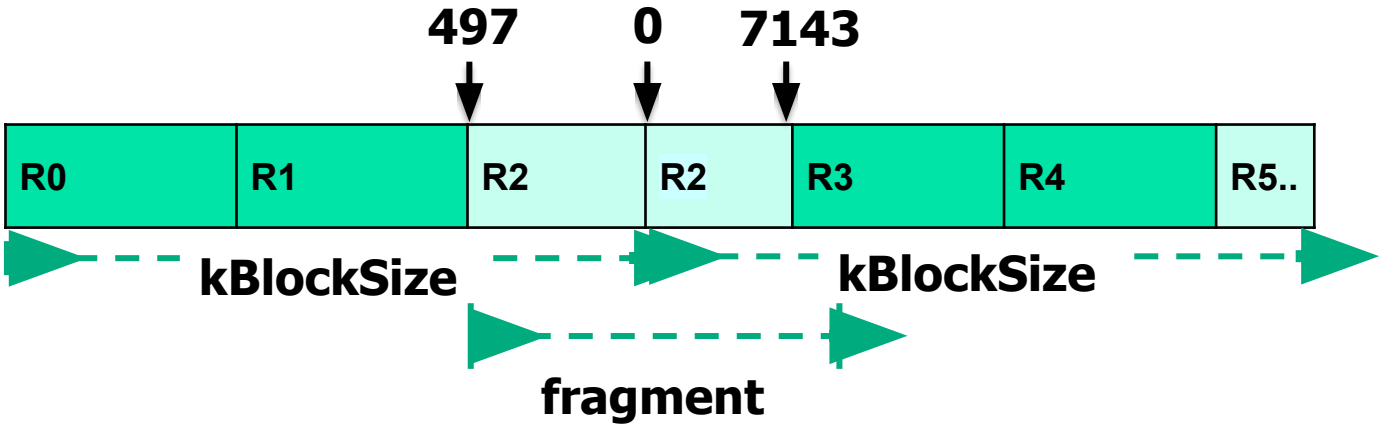
- Contents
  - ✔ Previous experiment
    - kBlockSize experiment
  - ✔ Experiment : Insert padding in WAL
    - Info
    - Code revise
    - Result
    - Discussion

RocksDB Festival : kBlockSize

- Previous experiment
  - ✔ Expected



  - ✔ Actual result



Rocksdb : insert padding

- Experiment Info.
  - ✔ Insert padding in WAL
  - ✔ Purpose
    - Performance comparison according to existence of padding
  - ✔ Conditions
    - kBlockSize = 8KB, 16KB, 32KB
    - key_size = (16 128 512 1024)
    - value_size = (16 128 256 512 1024 2048 3072 4096 5120 6144 7168 8192)
    - benchmarks= fillrandom
    - threads=6
    - compression_type=NoCompression
    - entries = 100000 times

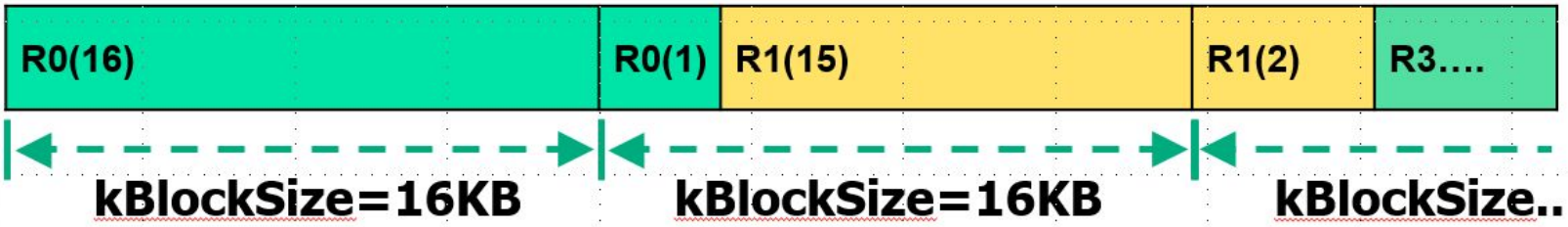
Rocksdb : insert padding

- Hardware Environment : D

D	
CPU	1 * AMD Ryzen 5 3500X 6-Core
OS	Ubuntu 20.04.2 LTS
SSD	mx500

Rocksdb : insert padding

- Previous experiment (Hypothesis)
  - ✓ If kBlockSize = 16KB, record size = 17KB, num=100



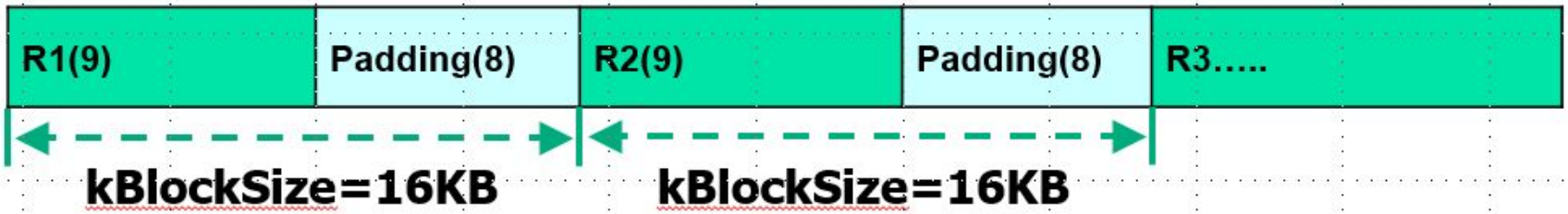
  - ✓ Padding is **not exist**, predict performance improve.
  - ✓ But fragmentation may cause **consistency** issues.

  - ✓ Padding is exist, overhead is too big.

Rocksdb : insert padding

- Previous experiment (Hypothesis)

✓ If kBlockSize = 16KB, record size = 9KB, num=100



✓ Padding is extremely high size
  - Expected performance degradation

✓ Write in DB = 900KB, Write in Storage = 1600KB
  - Write Amplification is so high

Rocksdb : insert padding

- Code revise
  - ✔ log_writer.h append code

```
#define PADDINGINSERT // if you want disable padding insert, delete this line.

  std::string repeat_;      Max Padding size(kBlockSize)

  for(int i = 0; i < n; i++) {
    repeat_ += s;
  }

  return repeat_;
}

std::string byte_padding = "\x00";        패딩 \x00 삽입
#ifdef PADDINGINSERT
  std::string slice_data_ = repeat(byte_padding, kBlockSize);
#else
  std::string slice_data_ = repeat(byte_padding, 10);
#endif
const char *slice_data__ = slice_data_.c_str();
```

Rocksdb : insert padding 2

- Code revise
  - ✔ log_writer.cc revise code

```
bool begin = true;
do {
  const int64_t leftover = kBlockSize - block_offset_;
  assert(leftover >= 0);
  if (leftover < header_size) {
    // Switch to a new block
    if (leftover > 0) {
      // Fill the trailer (literal below relies on kHeaderSize and
      // kRecyclableHeaderSize being <= 11)
      assert(header_size <= 11);
      s = dest_->Append(Slice("\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00",
                              static_cast<size_t>(leftover)));
      if (!s.ok()) {
        break;
      }
    }
    block_offset_ = 0;
  }
```

```
bool begin = true;
do {
  const int64_t leftover = kBlockSize - block_offset_;

// revised============================
  #ifdef PADDINGINSERT
    ispadding = (leftover < static_cast<int64_t>(left) && leftover != kBlockSize) || leftover < header_size ? true : false;
  #else
    ispadding = leftover < header_size ? true : false;
  #endif
// ================================

  // fprintf(stdout,"leftover : %ld\n", leftover);
  assert(leftover >= 0);

// revised============================
  if (ispadding) {
// ================================

    // Switch to a new block
    if (leftover > 0) {
      // Fill the trailer (literal below relies on kHeaderSize and
      // kRecyclableHeaderSize being <= 11)
      assert(header_size <= 11);

// revised============================
      // fprintf(stdout,"padding insert\n");
      s = dest_->Append(Slice(slice_data__, static_cast<size_t>(leftover)));
// ================================
```

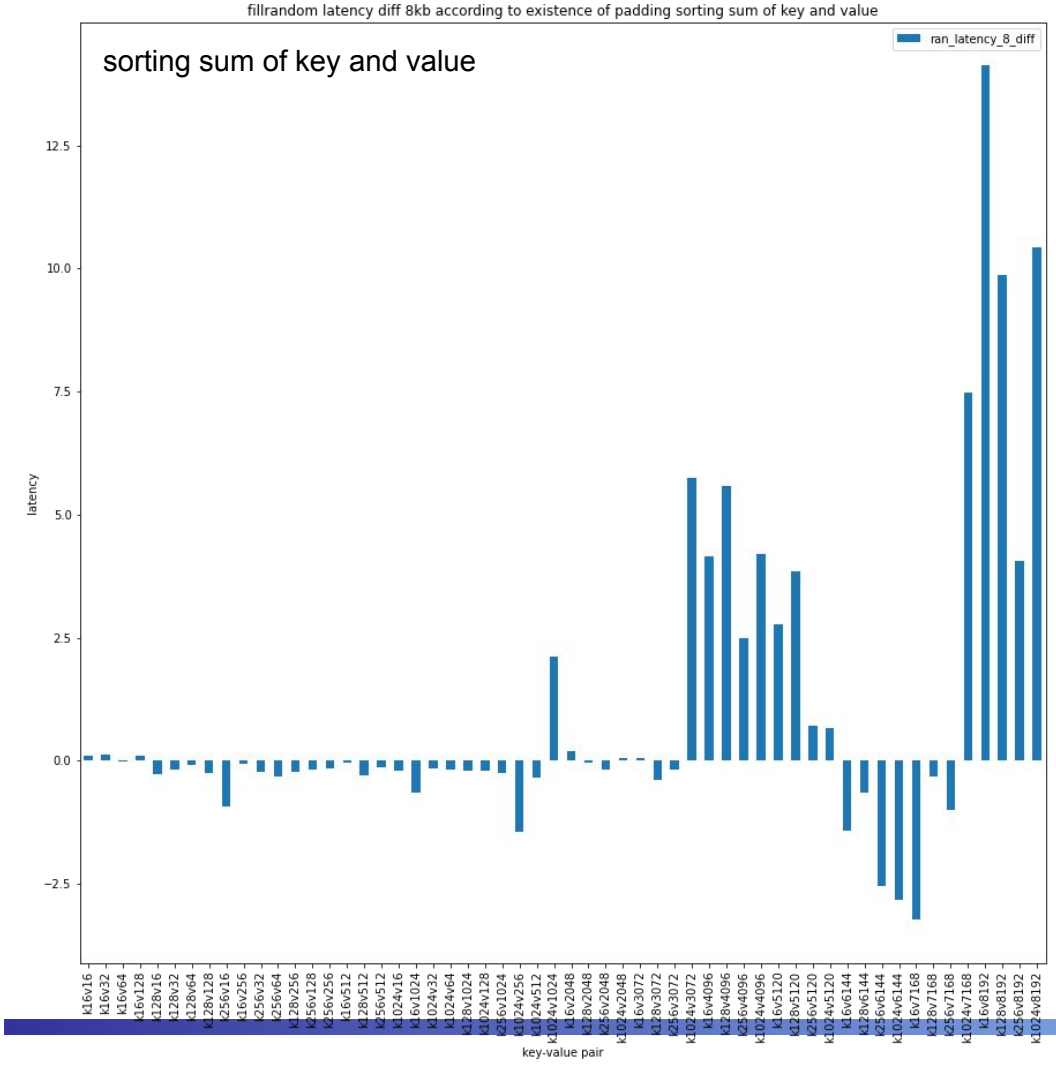
If padding exist
kBlock의 남은 크기가 **record**의 남은 크기보다 작고 **kBlock**이 비어있지 않을때

If padding not exist
kBlock의 남은 크기가 **header**의 크기보다 작을 때

kBlock의 남은 크기만큼 **max Padding size**를 **slicing**해서 **padding(\x00 -> 1byte) slicing**된 크기만큼 삽입한다.

Rocksdb : insert padding

- Result
  - ✔ kBlockSize = 8KB, latency = padding insert – not padding insert (s)

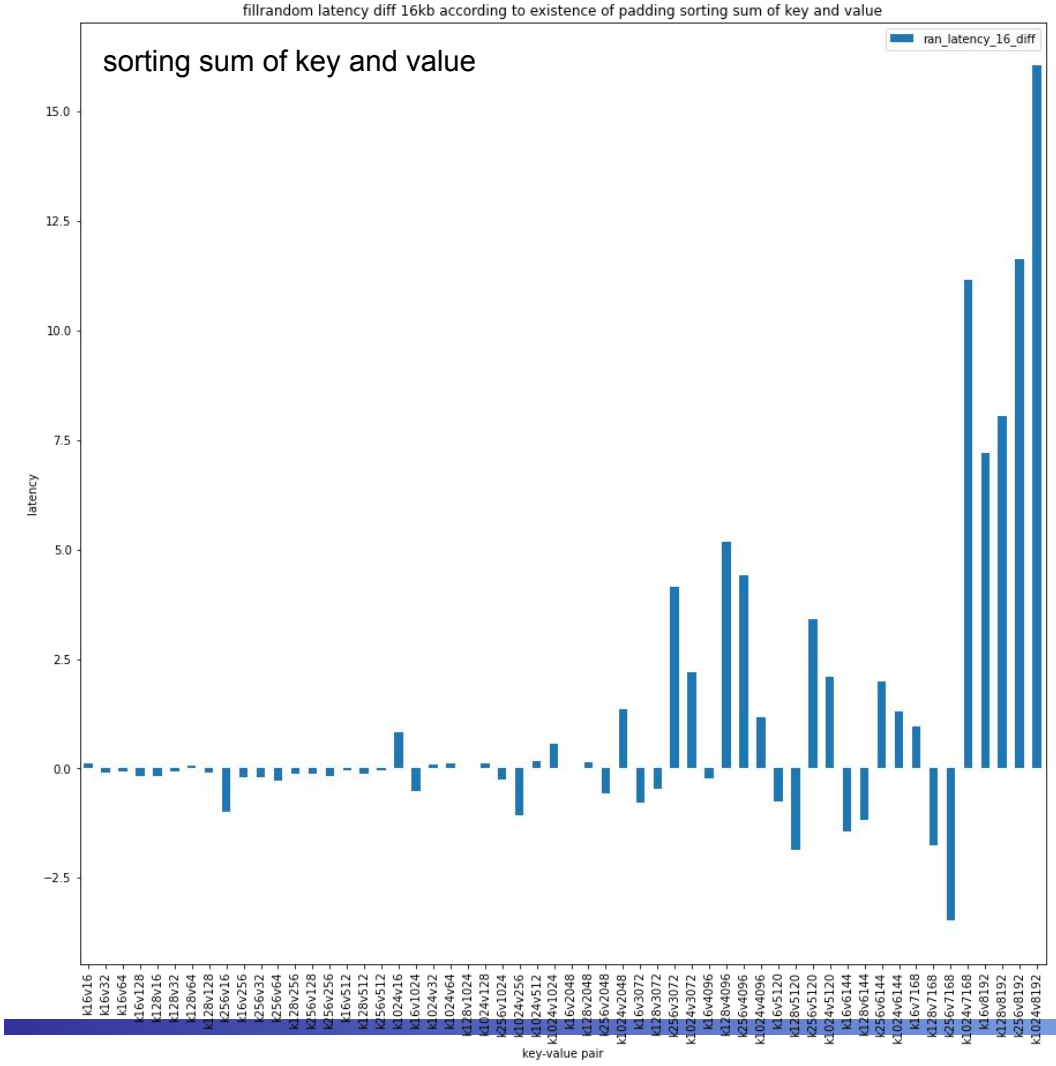


payload = key + value + log + dummy(15)

k16v16 : 48	k16v2048 : 1934
k16v32 : 2	k128v2048 : 1598
k16v64 : 32	k256v2048 : 1214
k16v128 : 58	k1024v2048 : 2004
k128v16 : 58	k16v3072 : 1972
k128v32 : 2	k128v3072 : 1748
k128v64 : 60	k256v3072 : 1492
k128v128 : 130	k1024v3072 : 4074
k256v16 : 254	k16v4096 : 4058
k16v256 : 254	k128v4096 : 3946
k256v32 : 132	k256v4096 : 3818
k256v64 : 336	k1024v4096 : 3050
k128v256 : 72	k16v5120 : 3034
k256v128 : 72	k128v5120 : 2922
k256v256 : 182	k256v5120 : 2794
k16v512 : 492	k1024v5120 : 2026
k128v512 : 248	k16v6144 : 2010
k256v512 : 292	k128v6144 : 1898
k1024v16 : 758	k256v6144 : 1770
k16v1024 : 758	k1024v6144 : 1002
k1024v32 : 646	k16v7168 : 986
k1024v64 : 422	k128v7168 : 874
k128v1024 : 1148	k256v7168 : 746
k1024v128 : 1148	k1024v7168 : 8214
k256v1024 : 380	k16v8192 : 8154
k1024v256 : 380	k128v8192 : 8004
k1024v512 : 402	k256v8192 : 7726
k1024v1024 : 1982	k1024v8192 : 6680

Rocksdb : insert padding

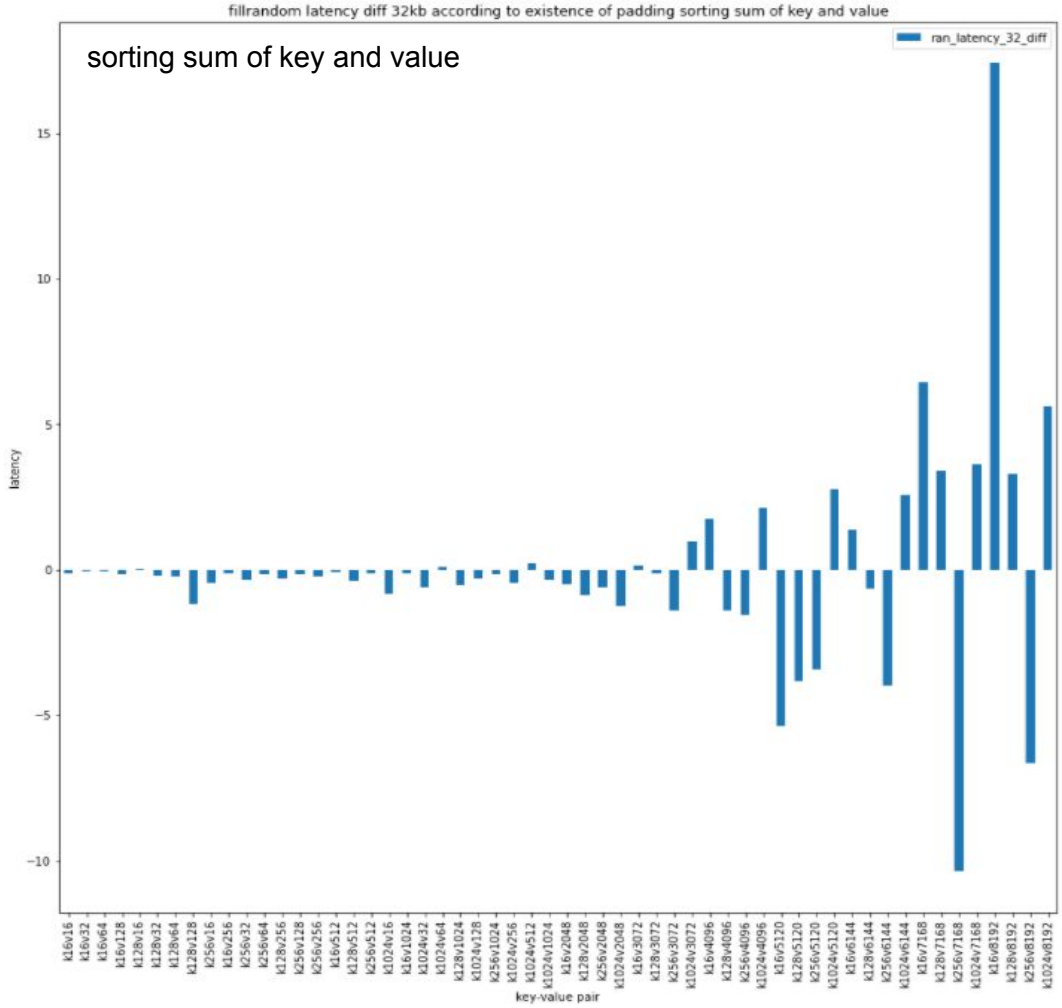
- Result
  - ✔ kBlockSize = 16KB, latency = padding insert – not padding insert (s)



k16v16 : 22	k16v2048 : 1782
k16v32 : 4	k128v2048 : 998
k16v64 : 64	k256v2048 : 102
k16v128 : 116	k1024v2048 : 914
k128v16 : 116	k16v3072 : 834
k128v32 : 4	k128v3072 : 274
k128v64 : 120	k256v3072 : 2984
k128v128 : 274	k1024v3072 : 4030
k256v16 : 214	k16v4096 : 3982
k16v256 : 214	k128v4096 : 3646
k256v32 : 264	k256v4096 : 3262
k256v64 : 310	k1024v4096 : 958
k128v256 : 144	k16v5120 : 910
k256v128 : 144	k128v5120 : 574
k256v256 : 364	k256v5120 : 190
k16v512 : 434	k1024v5120 : 4052
k128v512 : 496	k16v6144 : 4020
k256v512 : 584	k128v6144 : 3796
k1024v16 : 454	k256v6144 : 3540
k16v1024 : 454	k1024v6144 : 2004
k1024v32 : 214	k16v7168 : 1972
k1024v64 : 844	k128v7168 : 1748
k128v1024 : 1122	k256v7168 : 1492
k1024v128 : 1122	k1024v7168 : 8170
k256v1024 : 760	k16v8192 : 8154
k1024v256 : 760	k128v8192 : 8042
k1024v512 : 804	k256v8192 : 7914
k1024v1024 : 1894	k1024v8192 : 7146

Rocksdb : insert padding

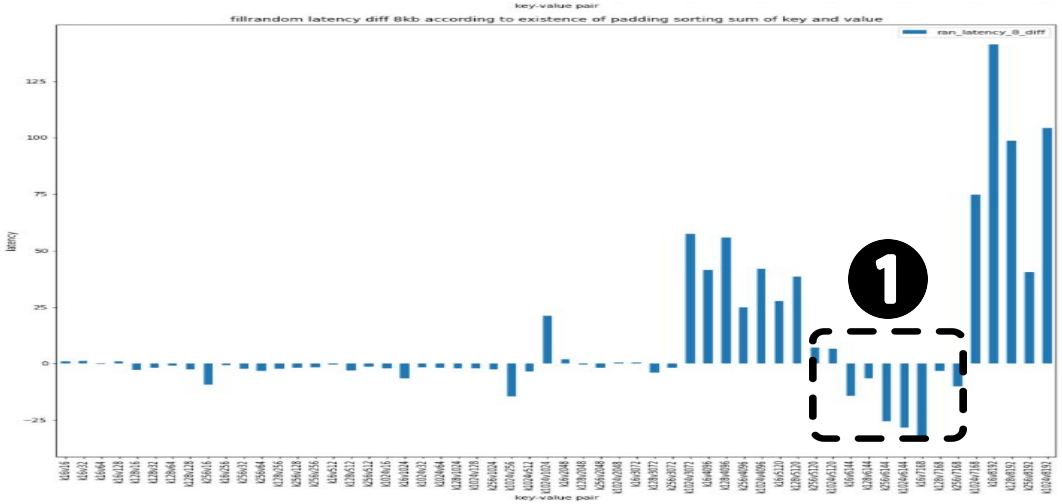
- Result
  - ✔ kBlockSize = 32KB, latency = padding insert – not padding insert (s)



k16v16 : 42	k16v2048 : 1478
k16v32 : 8	k128v2048 : 1996
k16v64 : 26	k256v2048 : 204
k16v128 : 66	k1024v2048 : 1828
k128v16 : 66	k16v3072 : 1668
k128v32 : 8	k128v3072 : 548
k128v64 : 26	k256v3072 : 2618
k128v128 : 242	k1024v3072 : 3942
k256v16 : 134	k16v4096 : 3830
k16v256 : 134	k128v4096 : 3046
k256v32 : 218	k256v4096 : 2150
k256v64 : 278	k1024v4096 : 1916
k128v256 : 288	k16v5120 : 1820
k256v128 : 288	k128v5120 : 1148
k256v256 : 194	k256v5120 : 380
k16v512 : 318	k1024v5120 : 1938
k128v512 : 330	k16v6144 : 1858
k256v512 : 378	k128v6144 : 1298
k1024v16 : 908	k256v6144 : 658
k16v1024 : 908	k1024v6144 : 4008
k1024v32 : 428	k16v7168 : 3944
k1024v64 : 578	k128v7168 : 3496
k128v1024 : 1070	k256v7168 : 2984
k1024v128 : 1070	k1024v7168 : 8126
k256v1024 : 218	k16v8192 : 8078
k1024v256 : 218	k128v8192 : 7742
k1024v512 : 50	k256v8192 : 7358
k1024v1024 : 1718	k1024v8192 : 5054

Rocksdb : insert padding

- Discussion
  - ✔ kBlockSize = 8KB

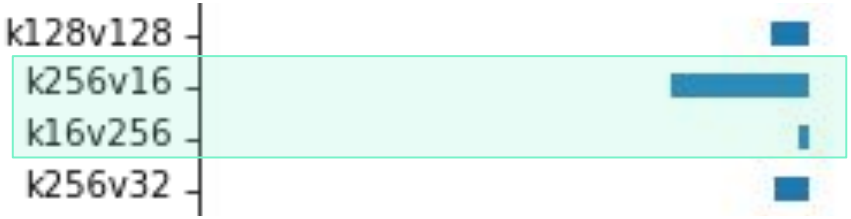


1. Despite insert padding, latency is lower than no padding result (performance better). Why?

2. The larger the padding, the worse the performance(latency) but exception(compression, memtable, etc….) is exist

Rocksdb : insert padding

- Discussion
  - ✔ kBlockSize = 8KB



latency graph

k128v128 : 242
k256v16 : 134
k16v256 : 134
k256v32 : 218

padding size

  - ✔ Padding sizes are same, different performance

Discussion