Benchmarking RocksDB: Exploring Compaction Options

David Shen
Xiaoyan Ge

Outline

1. What is RocksDB
2. Project Motivation
3. Background Info
4. Methodology
5. Results and Discussion
6. Conclusion
7. References

RocksDB

- RocksDB is a key-value store based on a log-structured merge-tree (LSM tree) data structure
- Developed by Facebook and based off of Google’s LevelDB.

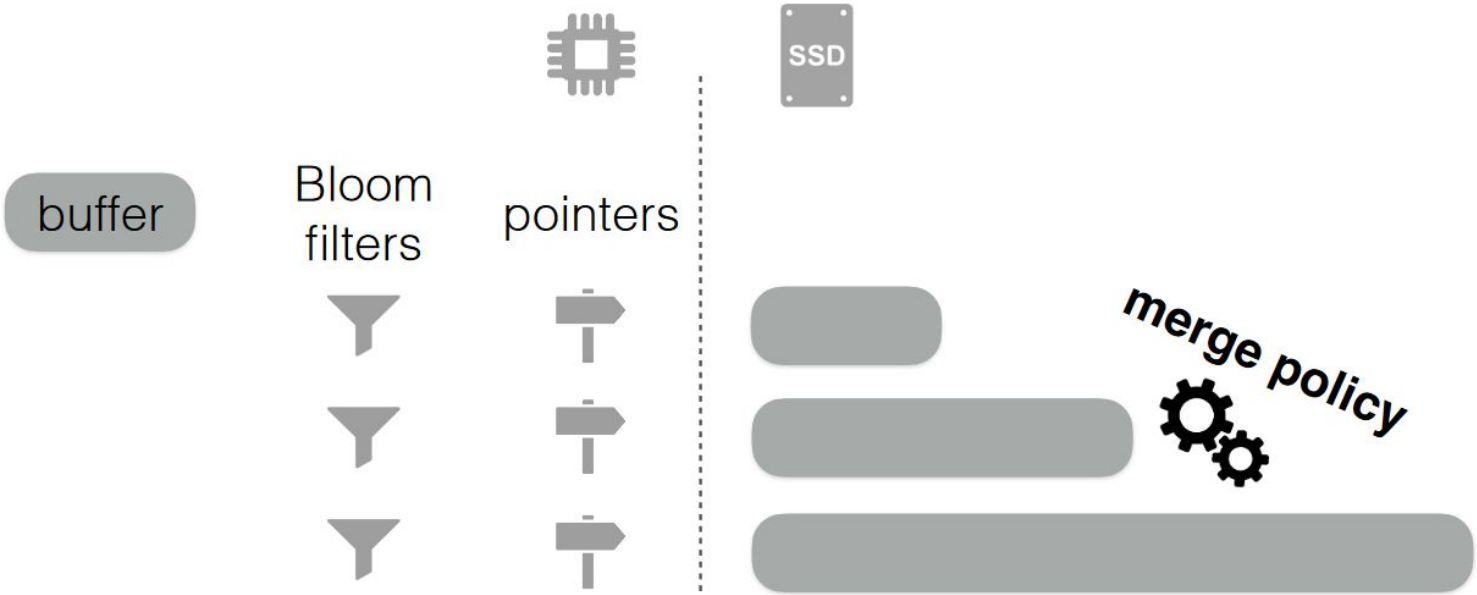
Motivation

Why benchmark a DB?

- Tuning a DB to a workload is a difficult problem
- Due to the sheer volume of configuration options, a lot of them often go undocumented

> Unfortunately, configuring RocksDB optimally is not trivial. Even we as RocksDB developers don't fully understand the effect of each configuration change. If you want to fully optimize RocksDB for your workload, we recommend experiments and benchmarking, while keeping an eye on the three amplification factors. Also, please don't hesitate to ask us for help on the RocksDB Developer's Discussion Group.

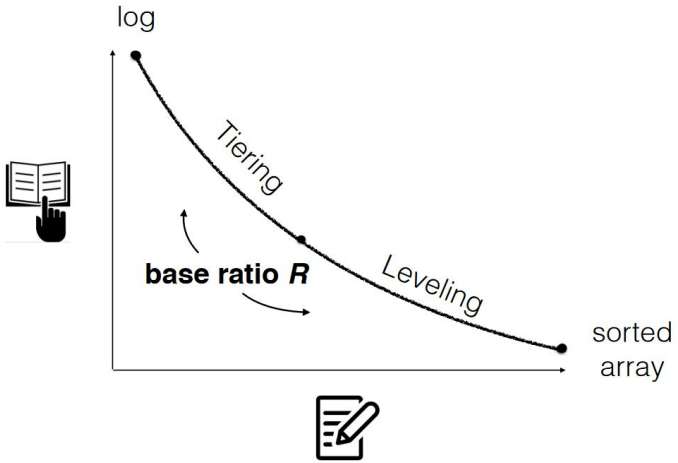
LSM-Tree and Merge Policies



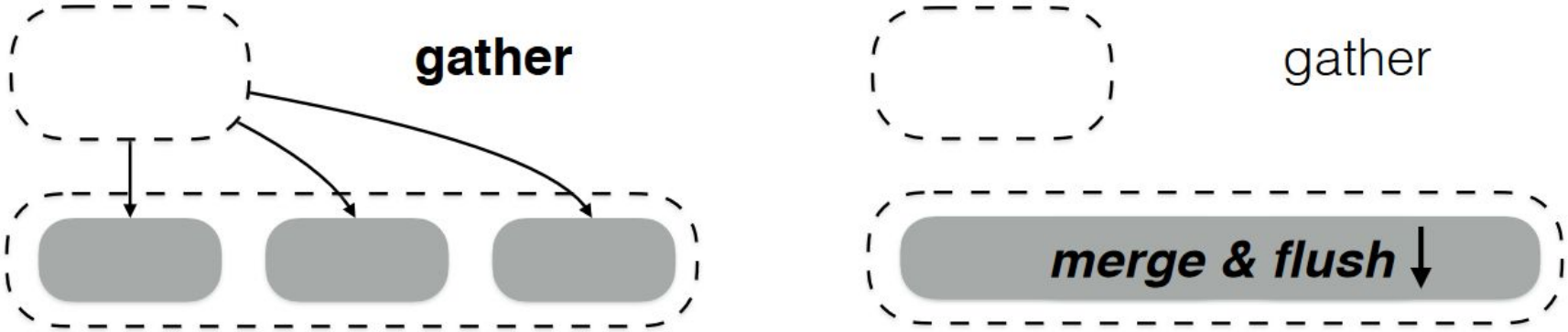
Merge Policies/Compaction Methods

Tiering (write optimized)

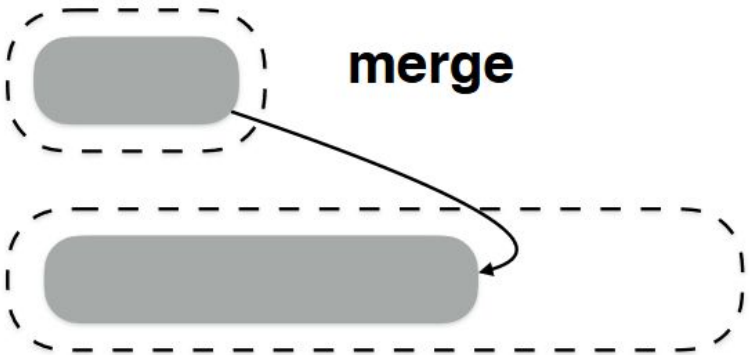
Leveling (read optimized)

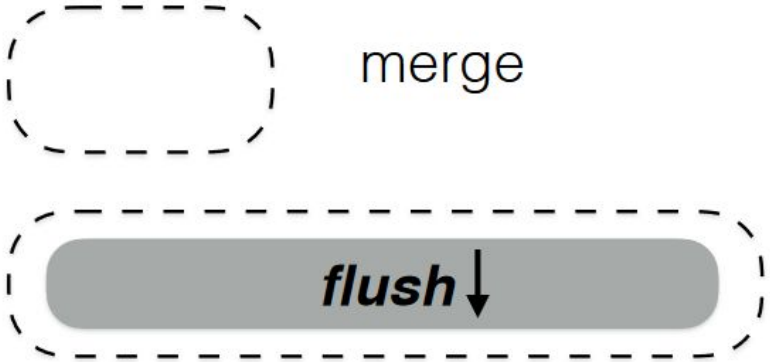


Tiering



Leveling





RocksDB’s Compaction methods

- RocksDB implements 2 main compaction methods
  - Tiered Compaction (“Universal”)
  - Leveled Compaction
- RocksDB offers various tuning knobs for compaction
  - Target_file_size_base
  - Target_file_size_multiplier
  - Compaction priority for leveled compaction

Methodology

Wanted to focus on the impact of compaction methods within RocksDB

- Create and run workloads using built in RocksDB tools
- Measure statistics using RocksDB tools and Linux command line tools (time, iostat)
- Benchmarking metrics
  - Response time
  - Throughput
  - Amplification factors

Experiment Setup

RocksDB: version 6.8

CPU: 8 * Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz

Keys: 20 bytes each

Values: 400 bytes each (200 bytes after compression)

Entries: 10 million entries

Tries: 5 tries, take average

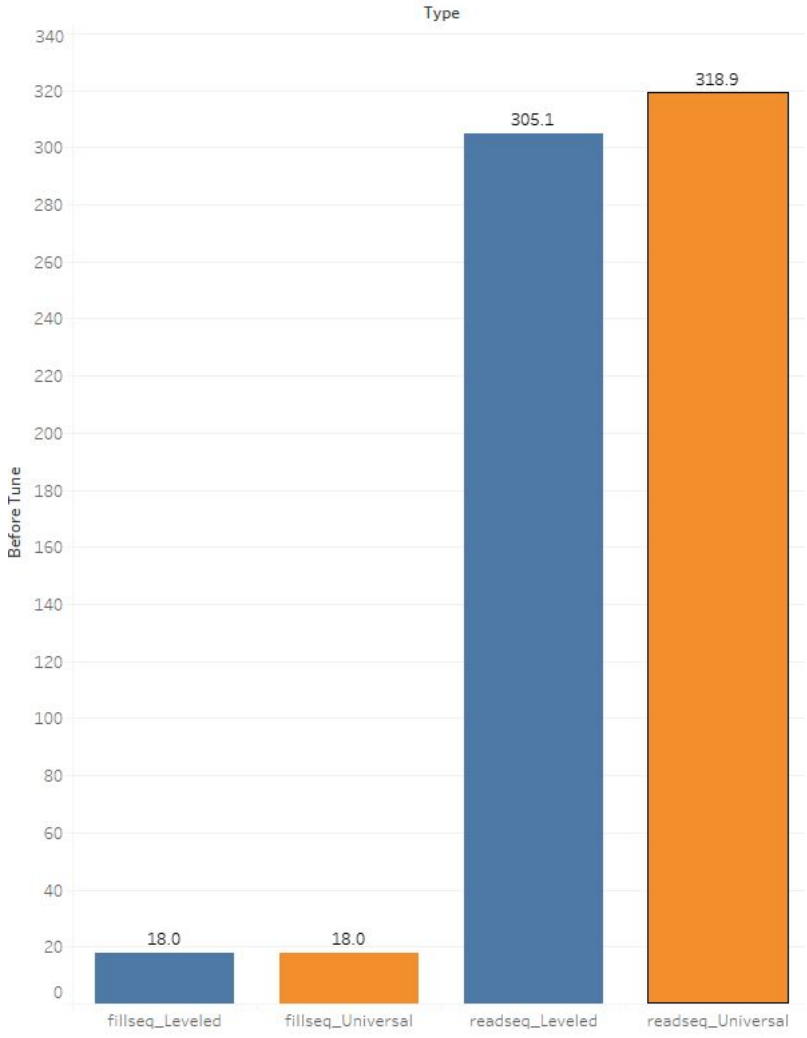
Tuning for write heavy workloads

Trading read performance for write performance

Workload:

- Fill a database with 10,000,000 key-value pairs in sequential order (fillseq)
- Read the data sequentially (readseq)
- Run each workload against both compaction methods

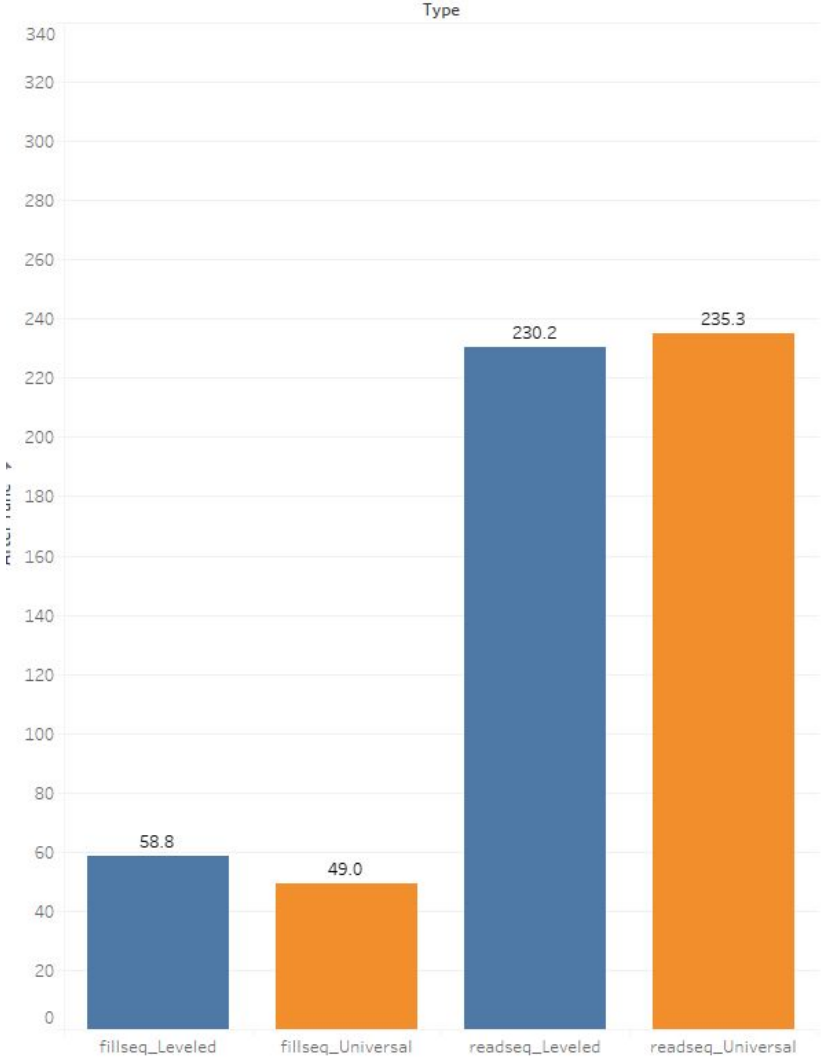
MB/s

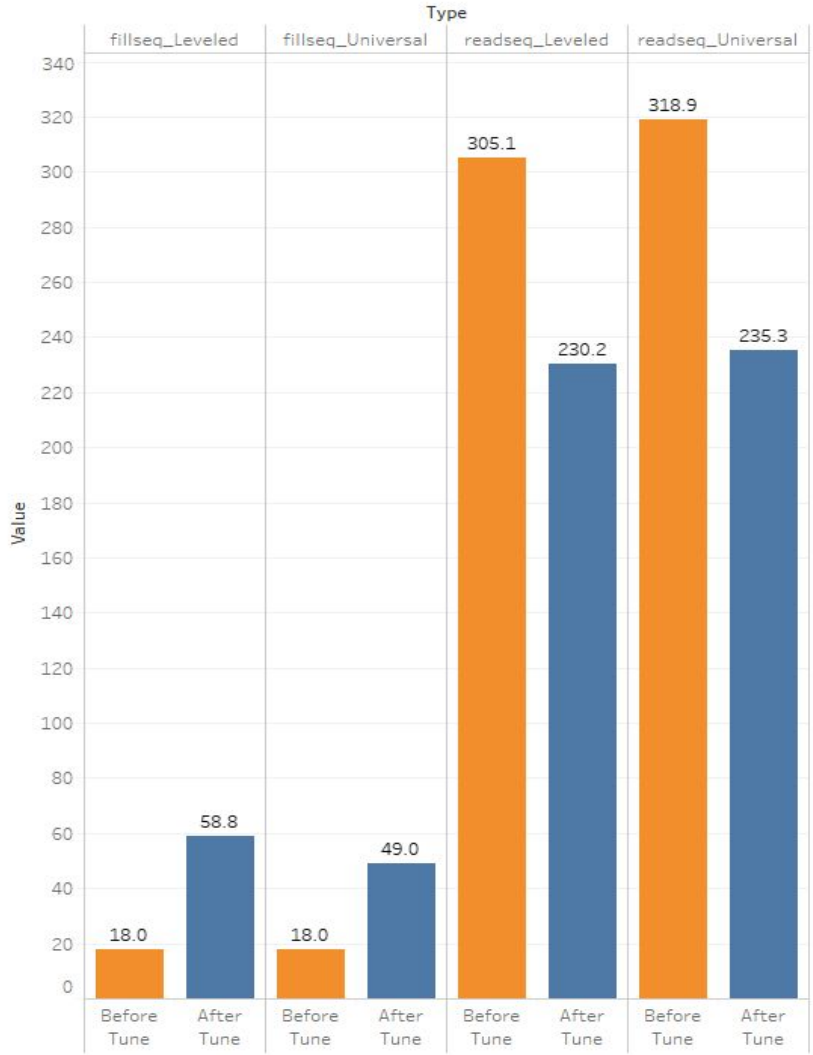

Type
Before Tune
340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
18.0
18.0
305.1
318.9
fillseq_Leveled
fillseq_Universal
readseq_Leveled
readseq_Universal

Trading Writing and Reading

- Doubled Target file based at level 1
- Doubled write_buffer_size before compaction
- Double max bytes for level 1 and multiplier to 8
- Pin_l0_filter_and_index_blocks_in_cache = 1
- Bytes_per_sync: sync SST files to disk while they are being written
- Bloom_bits = 10 bits for each key

After Tuning




Type
fillseq_Leveled
fillseq_Universal
readseq_Leveled
readseq_Universal
Value
340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
18.0
58.8
18.0
49.0
305.1
230.2
318.9
235.3
Before Tune
After Tune
Before Tune
After Tune
Before Tune
After Tune
Before Tune
After Tune

Compaction Priority

- Used in Level-based compaction
- Determines how level compaction chooses which files to be compacted in each compaction
- Four options in RocksDB
    - kByCompensatedSize (prioritize files with the most tombstones)
    - kOldestLargestSeqFirst (for workloads that update some hot keys in small ranges)
    - kOldestSmallestSeqFirst (for uniform updates across the key space)
    - kMinOverlappingRatio (looks at ratio between overlapping size in next level and its size)

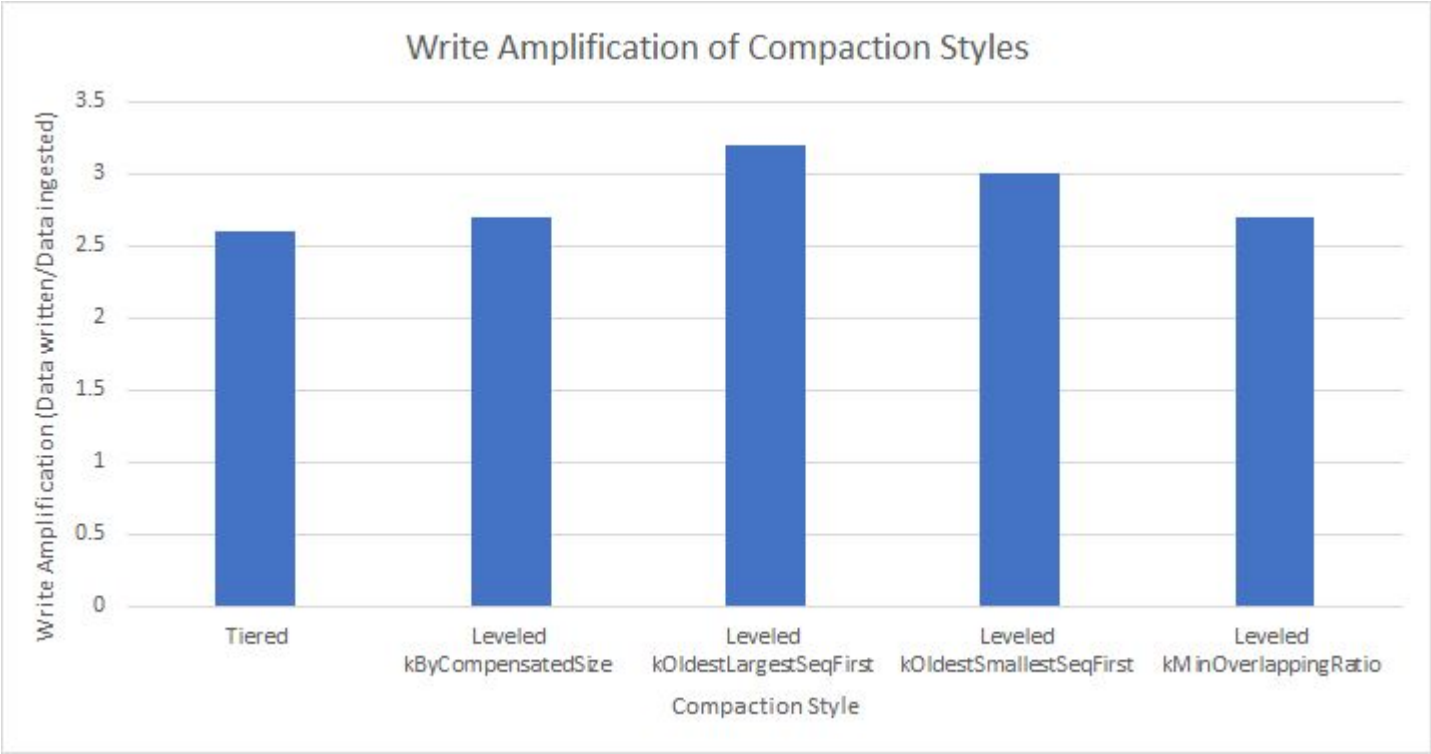
Tuning Priority

Tiered vs Leveled, varying compaction priority in Leveled

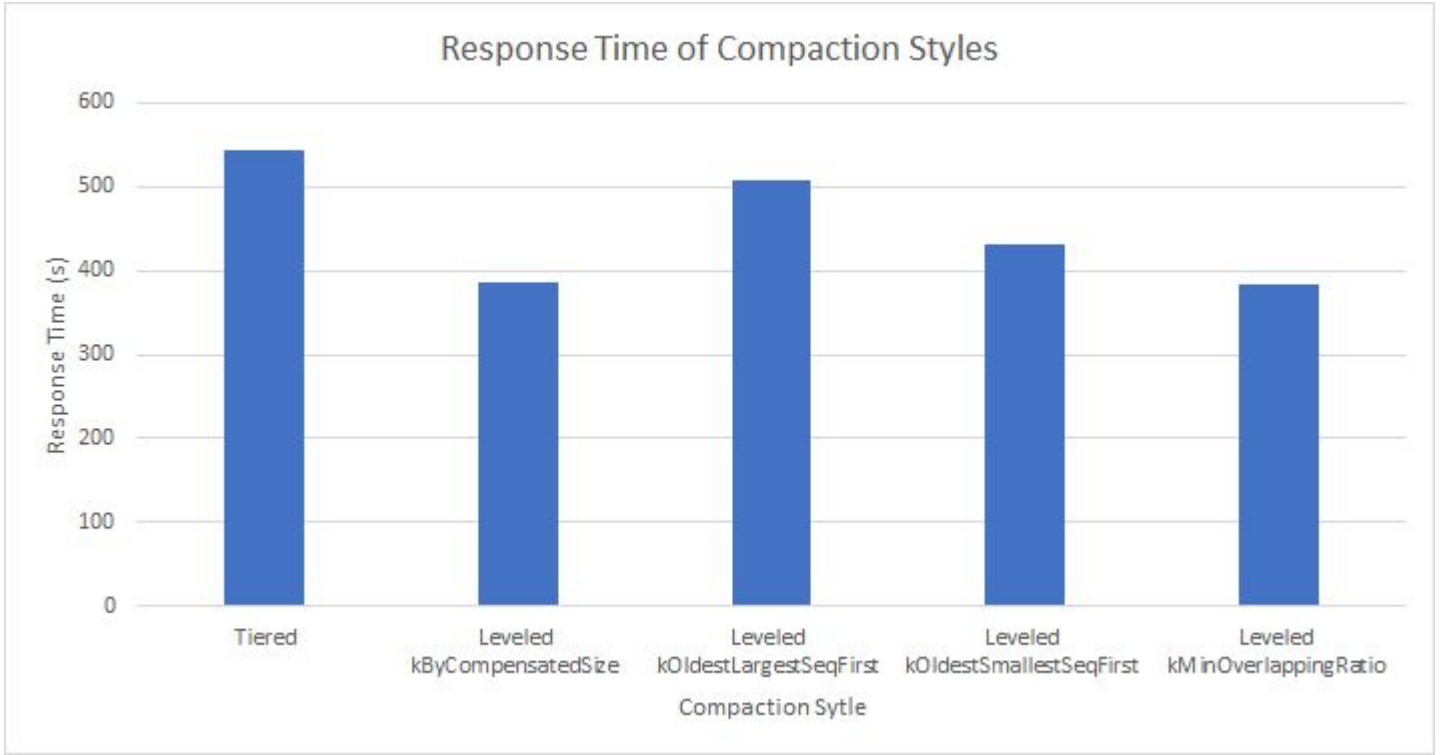
- Response time
- Write Amplification

We wanted to see if we could find a write-heavy workload that runs better using a specific priority in leveled compaction vs tiered compaction. Measure performance in terms of response time and write amplification.

Write Amplification



Response Time



Conclusion

Reinforces that finding an optimal tuning is a difficult problem.

- What is generally true may no longer hold
- Difficult to document what happens when a knob is turned
    - Have to either run the experiment or look at the source code

References

- https://github.com/facebook/rocksdb
- https://rocksdb.org/
- Dayan, Niv; Idreos, Stratos: The Log-Structured Merge-Bush & the Wacky Continuum, SIGMOD 2019. https://doi.org/10.5446/42955 (LSM-Tree background information)