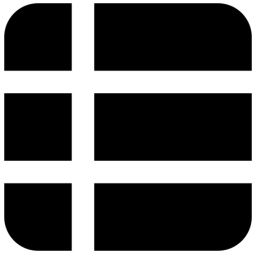
Time-to-Live (TTL) Aware Point Deletes in LSM Trees

Andrew Briasco-Stewart
Jovan Kascelan
Thura Naing

Table of Contents



Motivation - Privacy

Privacy Regulations

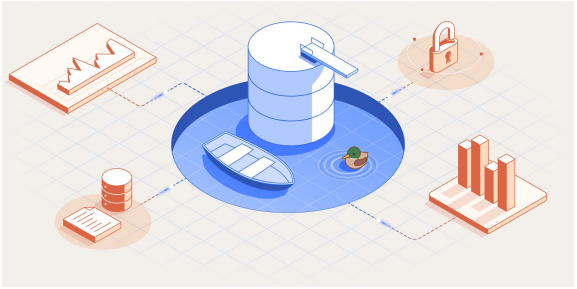
GDPR “right to erasure” - **“without undue delay”**

CCPA “right to delete” - **45 day**s to comply

Deletion Failure == Severe infringement

Fines up to €20 million or 4% of revenue

Motivation - Performance



Space amplification storage costs

- Both tombstones + physical data consumes space
- Data Lakes, billing is tied to storage volume, increased operational costs

Query/Read performance

- Reads must scan tombstones and filter
- Tombstones degrade read latency & throughput, particularly range deletes

Background: LSM Trees

Write Optimized Design

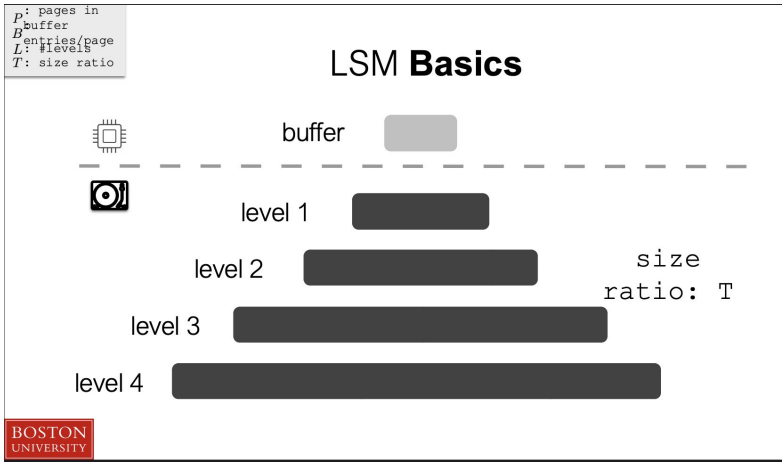
Writes into **MemTable,** the buffer in memory -> high speed ingestion

Once MemTable reaches a size limit, it is flushed into disk as a immutable **Sorted String Table (SSTables)**

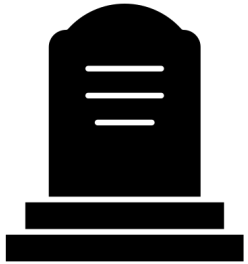
Deletes = Special marker called “tombstones”, which invalidates physically stored records logically

Many SSTables = degradation of read speed & space amp

Compaction merges older SSTables and physically deletes tombstone records



Background: LSM Trees Deletion



LSM Trees are optimized for Fast Data Ingestion

ISSUE: ***delete persistence latency***

Deleted Record is `logically deleted` with a tombstone insertion

Physical data persists on disk until compaction occurs with tombstone -> `physically deleted`

Tombstone deleted when compacted to bottom-most level

Our Work

RocksDB: ~2108 files, ~900K LOC, limited documentation

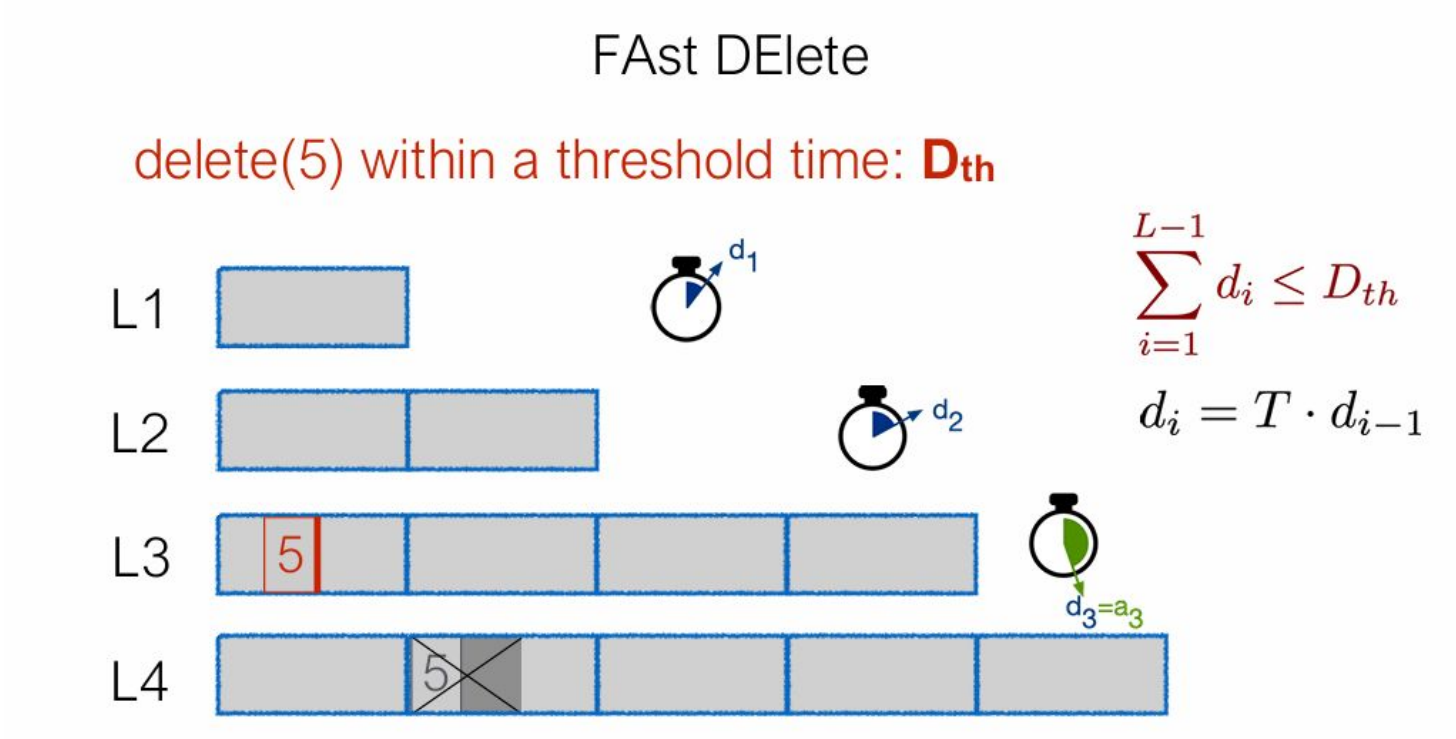
1. Update Lethe to work with current (10.10.1) version of RocksDB

   43 files changed -> 7,835 additions, 21 deletions.
2. Updated K-V-Workload Generator to create delete workloads with TTLs

   5 files changed -> 92 additions, 15 deletions.
3. Implement multiple delete threshold functionality in current version

   27 files changed -> 484 additions, 134 deletions.
4. Testing, Debugging, Verifying

Background: Original Lethe (FADE)



Our Work: Modified Lethe for Different DTs

Tombstones inserted with DT *(TTL)* values

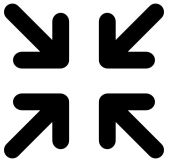
After each flush/compaction,
a collector is called to update
the min DT file metadata

T	**qFq1CHXJq6v5KSeIZuuM0GSEmEqWBVjX…**	**2000**
TTL Delete Command	Key to Delete	TTL Value (seconds)

Modified Lethe Compaction Strategy

Keep track of the smallest DT for each file 

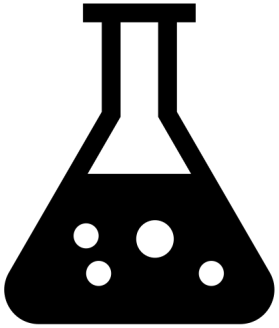
compute *value*=(tombstone insertion timestamp+Level TTL) 

Then, **trigger** a compaction if that *value* < *current time* 

Experiment Setup - RocksDB Versioning

Version Name	RocksDB Version	Compaction Strategy
RocksDB Old	*6.11.4*	*kCompactionStyleLeveling*
Lethe Original	*6.11.4*	*Lethe kFade Compaction*
RocksDB Latest	*10.10.1*	*kCompactionStyleLeveling*
Lethe Latest	*10.10.1*	*Lethe kFade Compaction*
Lethe w/ TTL Deletes	*10.10.1*	*Modified Lethe kFade*

Experiment Setup - Delete Workloads



2 Workload Types:

1M Inserts + Deletes w/ delete percentage 2, 4, 6, 8, 10%

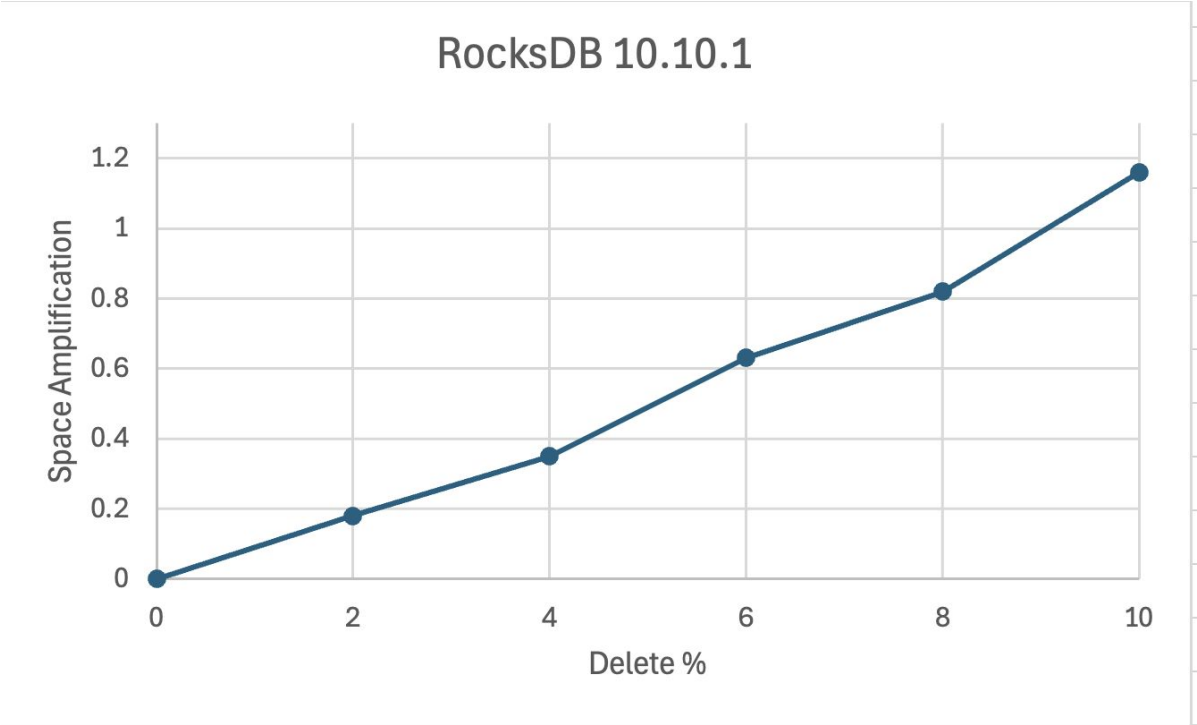
Runtime: 15 mins

500K Inserts + Deletes w/ delete percentage 2, 5, 10%

Runtime: 9 mins

Latest Base RocksDB

1M Operations



Baseline for comparison

Linear increase with delete %

Better than old version

Lethe Vs Base RocksDB 1M Operations Workload

Space Amp

Original RocksDB 1.16

Lethe 0.590825

Tombstones

Original RocksDB 18493

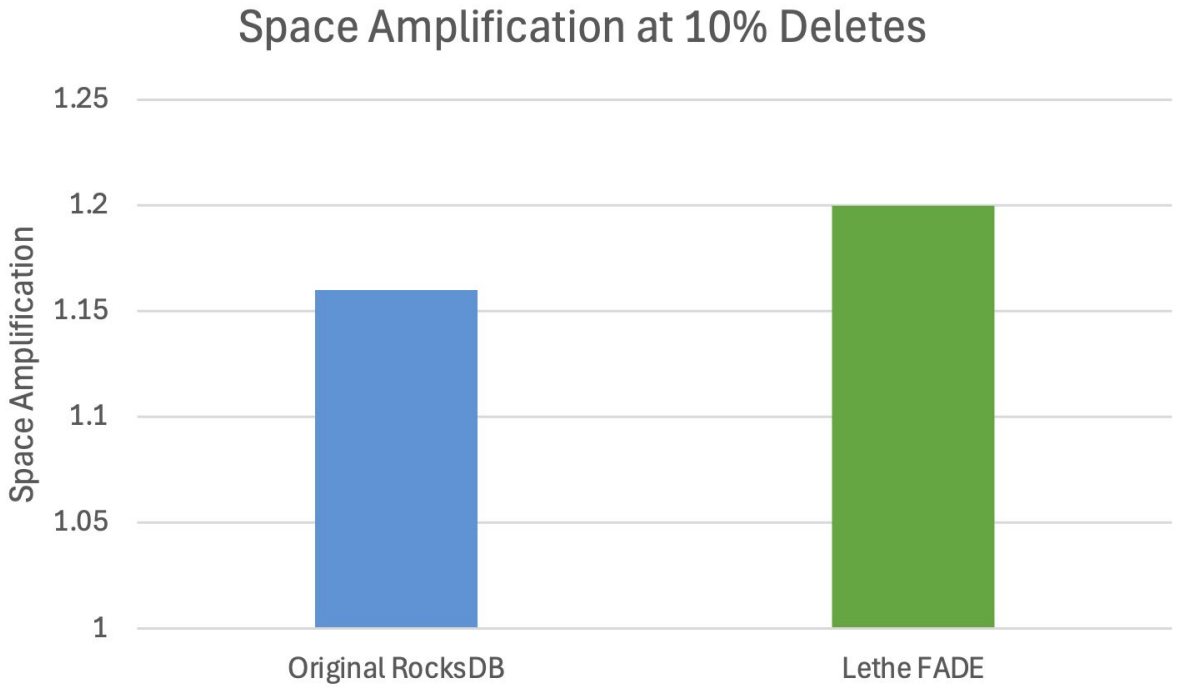
Lethe 9380

Experiment Results

Metric	kfade	kCompactionStyleLeveling
runtime	3045.182 s	3091.945 s
#cmpt	144	149
bts_rd_cmpt	15.88 GB	15.80 GB
bts_wr_cmpt	15.37 GB	15.29 GB

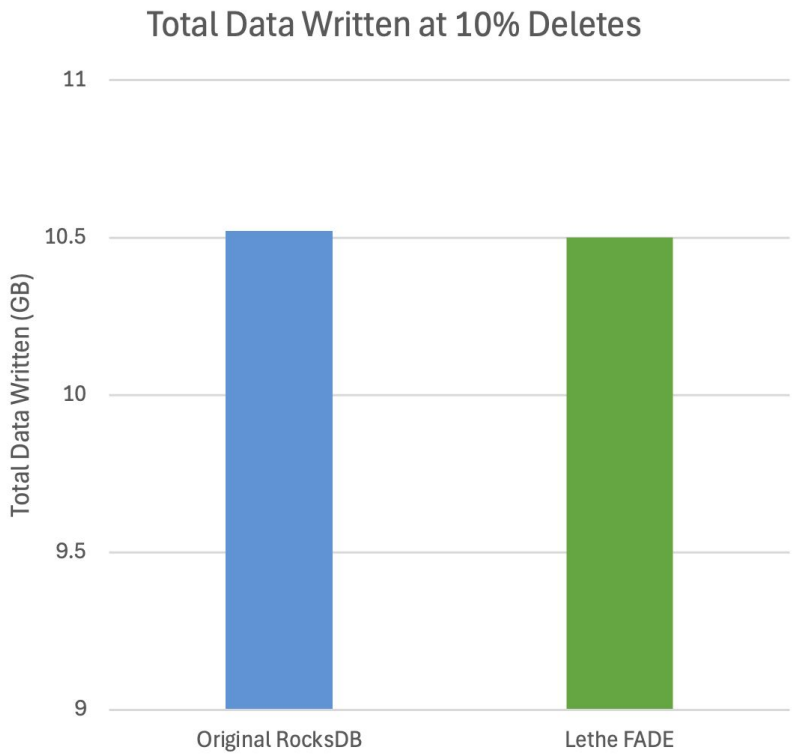
Table 1: Comparison: kfade vs kCompactionStyleLeveling

Results: 1M operation workload



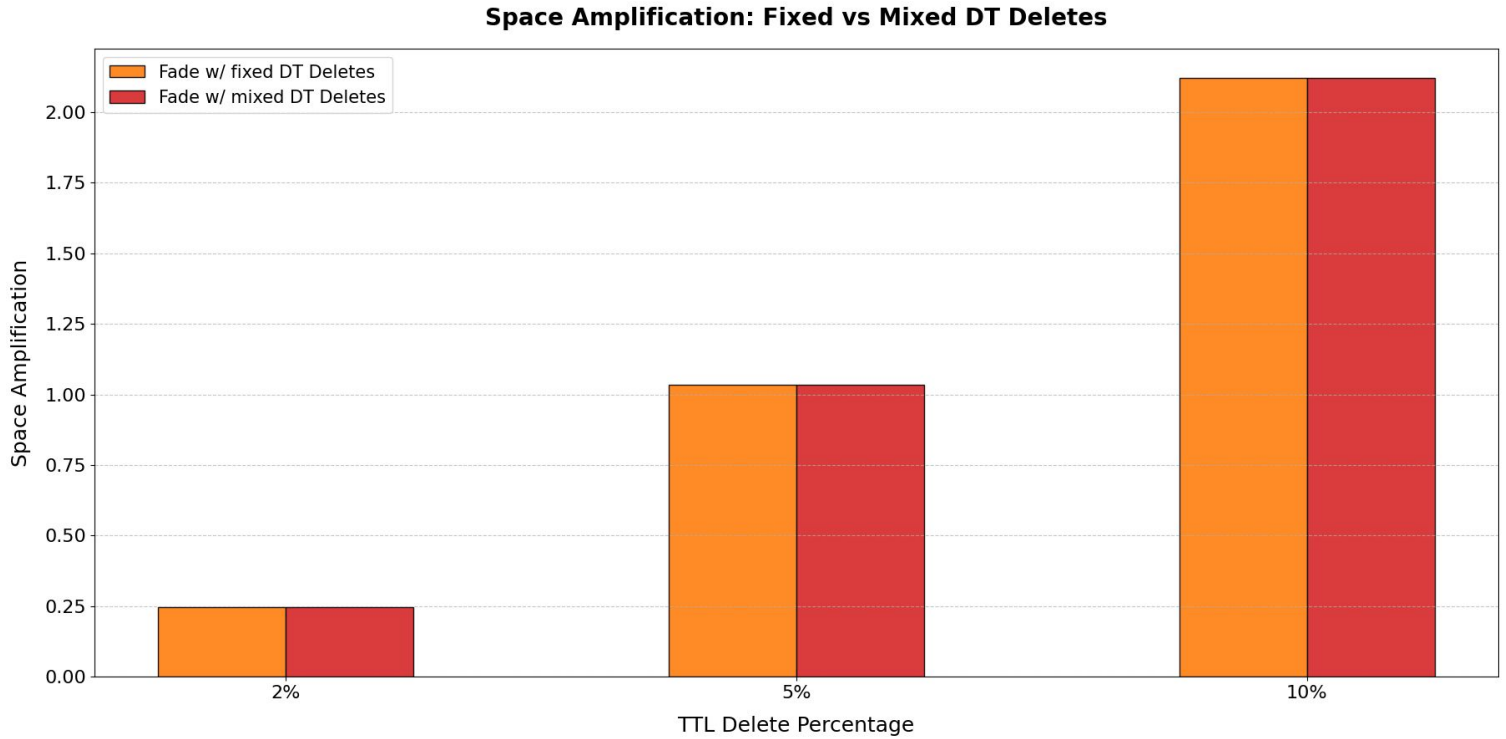
Slight Increase in Space Amplification

Results: 1M operation workload

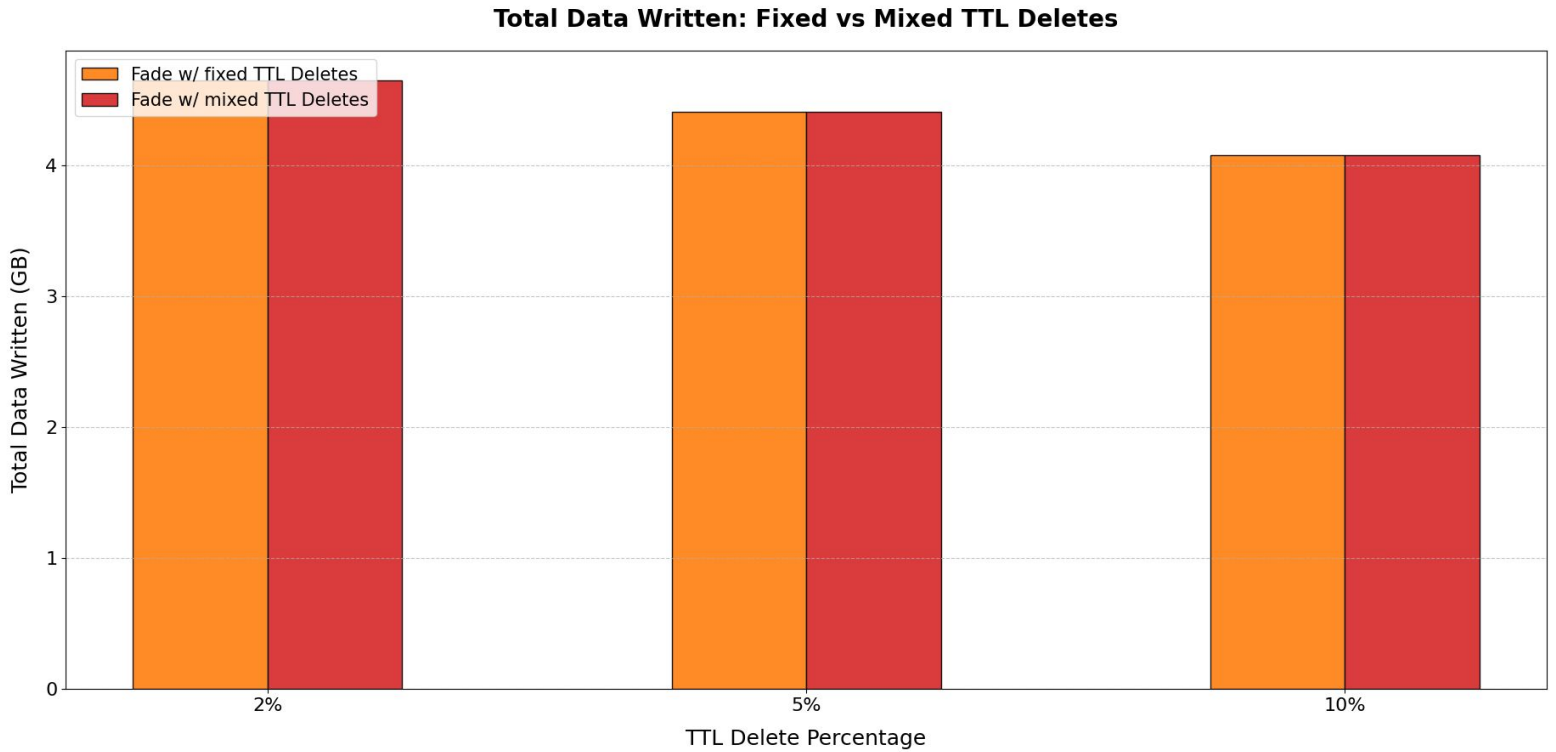


Similar or slightly less data written

Results: 500K operation workload



Results: 500K operation workload



Analysis: Why Similar Performance?

New RocksDB optimization that removes tombstones early reduces the impact of Fade

Mixed/variable DT implementation disables this optimization

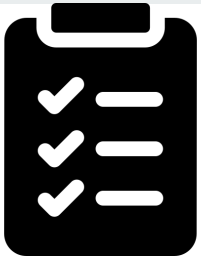
Mixed/variable DT logic regresses to fixed DT because most files contains a tombstone with the min DT

Additional Results (ongoing)

Lethe with variable TTL Deletes gets 1.07x smaller space amplification over regular RocksDB when the tombstone optimization is removed.

They achieve a nearly identical total amount of data written in this scenario.

Future Work



Goal: Demonstrate the advantage of having a mixed/variable DT approach

Next steps:

- Try mixed DT workload where the minimum DT is very rare
- Change the workload generator to prioritize generating tombstones with the min DT earlier

Additional goal: Add verification that confirms that the tombstones are getting removed on time

Thank You

Questions?