Lethe: A Tunable Delete-Aware LSM-Based Storage Engine

Subhadeep Sarkar
Tarikul Islam Papon
Dimitris Staratzis
Manos Athanassoulis
Even years later, Twitter doesn’t delete your direct messages

TechCrunch  
Feb ’19

Small Datum  
Jan ’20

Deletes are fast and slow in an LSM

“LSM-based data stores perform suboptimally for workloads with deletes.”
log-structured merge-tree

buffer
log-structured merge-tree

buffer 2614
log-structured merge-tree
log-structured merge-tree

buffer
log-structured merge-tree
log-structured merge-tree

buffer

L1
size ratio = T

L2

exponentially larger capacity

L3
log-structured merge-tree

buffer

partial compaction

L1

L2

L3
log-structured merge-tree

buffer

partial compaction

L1
L2
L3
log-structured merge-tree

buffer

partial compaction

L1
L2
L3
log-structured merge-tree

buffer

partial compaction

L1

L2

L3
log-structured merge-tree

buffer

partial compaction

L1

L2

L3
log-structured merge-tree

buffer

partial compaction

L1

L2
NEW NEW

L3
log-structured merge-tree

buffer

partial compaction

L1
L2
L3
log-structured merge-tree

buffer

partial compaction

L1
L2
L3
log-structured merge-tree

buffer

partial compaction

L1

L2

L3
log-structured merge-tree

buffer

partial compaction

L1

L2

L3

amortized compaction cost
log-structured merge-tree

get(5)

buffer

Bloom filters

fence pointers

L1

L2

L3

fewer disk I/Os
Now, let’s talk about deletes!
deletes in LSM-tree

delete := insert tombstone
deletes in LSM-tree

```
delete := insert tombstone
```

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>TS flag</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>timestamp</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>department</td>
</tr>
<tr>
<td></td>
<td>location</td>
</tr>
</tbody>
</table>
```
deletes in LSM-tree

delete(5)

buffer

L1

L2

L3
deletes in LSM-tree

buffer

L1
L2
L3
deletes in LSM-tree

get(5)

buffer

Bloom filters

fence pointers

L1

L2

L3
the problems
out-of-place deletes
out-of-place deletes

L1

L2

L3

L4

space amplification
out-of-place deletes

L1

L2

L3

L4

space amplification
out-of-place deletes
out-of-place deletes

Bloom filters

L1

L2

L3

L4

poor read perf.
write amplification
space amplification
the problems

- poor read perf.
- write amplification
- space amplification
delete persistence latency
delete persistence latency

L1

L2

L3

L4
delete persistence latency

delete(5) within a threshold time: $D_{th}$
delete persistence latency

delete(5) within a threshold time: $D_{th}$
delete persistence latency

delete(5) within a threshold time: $D_{th}$
delete persistence latency

delete(5) within a threshold time: $D_{th}$
delete persistence latency

delete(5) within a threshold time: $D_{th}$

$L1$  

$L2$  5

$L3$  

$L4$  5
delete persistence latency

delete(5) within a threshold time: $D_{th}$
delete persistence latency

delete(5) within a threshold time: $D_{th}$

$D_{th} = t_1 + t_2 + t_3$
delete persistence latency

delete(5) within a threshold time: $D_{th}$

$L_1$  

$L_2$  

$L_3$  

$L_4$  

unbounded delete persistence latency
the problems

- poor read perf.
- write amplification
- space amplification

\[ \sum_{i=1}^{L-1} t_i \]

unbounded delete persistence latency
deletes on a secondary attribute
deletes on a secondary attribute

delete all entries older than: \( D \) days
deletes on a secondary attribute

delete all entries older than: D days
deletes on a secondary attribute

delete all entries older than: D days

latency spikes

superfluous I/Os
the problems

- poor read perf.
- write amplification
- space amplification

\[ \sum_{i=1}^{L-1} t_i \]

- latency spikes
- superfluous I/Os

unbounded delete persistence latency
the solution

- poor read perf.
- write amplification
- space amplification
- unbounded delete persistence latency

FADE

\[ \sum_{i=1}^{L-1} t_i \]

- latency spikes
- superfluous I/Os
FAst DElete

delete(5) within a threshold time: $D_{th}$
FAST DELETE

delete(5) within a threshold time: $D_{th}$

$L1 \hspace{1cm} L2 \hspace{1cm} L3 \hspace{1cm} L4$

$d_1 \hspace{1cm} d_2 \hspace{1cm} d_3$

$L-1 \sum_{i=1}^{L-1} d_i \leq D_{th}$

$d_i = T \cdot d_{i-1}$
Fast DElete

delete(5) within a threshold time: $D_{th}$

\[
\sum_{i=1}^{L-1} d_i \leq D_{th}
\]

\[
d_i = T \cdot d_{i-1}
\]
FAst DElete

delete(5) within a threshold time: $D_{th}$

\[ \sum_{i=1}^{L-1} d_i \leq D_{th} \]

\[ d_i = T \cdot d_{i-1} \]
Fast Delete

delete(5) within a threshold time: $D_{th}$

$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

$$d_i = T \cdot d_{i-1}$$
FAst DElete

delete(5) within a threshold time: $D_{th}$

\[ \sum_{i=1}^{L-1} d_i \leq D_{th} \]
\[ d_i = T \cdot d_{i-1} \]
FAst DElete

delete(5) within a threshold time: $D_{th}$

$L_1$

$L_2$

$L_3$

$L_4$

$L_{1-1} \sum_{i=1}^{L-1} d_i \leq D_{th}$

$d_i = T \cdot d_{i-1}$
FAst DElete

delete(5) within a threshold time: $D_{th}$

$L_{-1}$

\[ \sum_{i=1}^{L-1} d_i \leq D_{th} \]

\[ d_i = T \cdot d_{i-1} \]
FAst DElete

delete(5) within a threshold time: $D_{th}$

\[
\sum_{i=1}^{L-1} d_i \leq D_{th}
\]
\[
d_i = T \cdot d_{i-1}
\]
### Fast Delete

Delete(5) within a threshold time: $D_{th}$

<table>
<thead>
<tr>
<th>Level</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>$d_1$</td>
</tr>
<tr>
<td>L2</td>
<td>$d_2$</td>
</tr>
<tr>
<td>L3</td>
<td>$d_3$</td>
</tr>
<tr>
<td>L4</td>
<td></td>
</tr>
</tbody>
</table>

$$\sum_{i=1}^{L-1} d_i \leq D_{th}$$

$$d_i = T \cdot d_{i-1}$$
FAst DElete

breaking ties in practical workloads
FAst DElete
breaking ties in practical workloads
FAst DElete

breaking ties in practical workloads
FAst DElete

breaking ties in practical workloads
FAst DElete

breaking ties in practical workloads
FAst DElete

breaking ties in practical workloads
FAst DElete

- **Higher write amplification**: 4 - 25%
- **Improved read performance**: 1.2 - 1.4x
- **Reduced space amplification**: 2.1 - 9.8x
- **Timely delete persistence**: within $D_{th}$

**Graph:**
- 1M 1KB entries, 1MB buffer, $T=10$
- Cumulative number of tombstones
- Age of files (s)
- $D_{th} = 16\%$, $D_{th} = 25\%$, $D_{th} = 50\%$

**Legend:**
- $x10^3$
FAst DElete

- Higher write amplification: 0.7%
- Improved read performance: 1.2 - 1.4x
- Reduced space amplification: 2.1 - 9.8x
- Timely delete persistence: within $D_{th}$

1M 1KB entries, 1MB buffer, $T=10$

-岩石DB
- FADE/25%

 normalized bytes written vs. snapshot #
the solution

**FAst DElete**
- higher write amplification
- improved read performance
- reduced space amplification
- timely delete persistence

**KiWi**
- latency spikes
- superfluous I/Os
the solution

FAst DElete

higher write amplification

FADE

improved read performance

KiWi

reduced space amplification

timely delete persistence
Key Weaving storage layout

delete all entries older than: D days

L1
L2
L3
L4

scattered occurrences
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$
Key Weaving storage layout

delete all entries with timestamp \( \leq 65_D \)

<table>
<thead>
<tr>
<th>Page</th>
<th>Key Range</th>
<th>Page Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( S_{\text{min}}=1 ), ( S_{\text{max}}=99 )</td>
<td>( D_{\text{min}}=1_D ), ( D_{\text{max}}=90_D )</td>
</tr>
<tr>
<td>2</td>
<td>( S_{\text{min}}=29 ), ( S_{\text{max}}=60 )</td>
<td>( D_{\text{min}}=9_D ), ( D_{\text{max}}=90_D )</td>
</tr>
<tr>
<td>3</td>
<td>( S_{\text{min}}=61 ), ( S_{\text{max}}=79 )</td>
<td>( D_{\text{min}}=1_D ), ( D_{\text{max}}=90_D )</td>
</tr>
<tr>
<td>4</td>
<td>( S_{\text{min}}=80 ), ( S_{\text{max}}=99 )</td>
<td>( D_{\text{min}}=7_D ), ( D_{\text{max}}=89_D )</td>
</tr>
</tbody>
</table>

partitioned on \( S \)
Key Weaving storage layout

delete all entries with timestamp \( \leq 65_D \)

partitioned on \( S \)
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

partitioned on $S$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

partitioned on $S$
Key Weaving storage layout

delete all entries with timestamp \( \leq 65_D \)

partitioned on \( D \)

partitioned on \( S \)
Key Weaving storage layout

drop page

partitioned on $D$

partitioned on $S$

delete all entries with timestamp $\leq 65_D$

$S_{\text{min}} = 1 :: S_{\text{max}} = 99$
$D_{\text{min}} = 1_D :: D_{\text{max}} = 90_D$

$S_{\text{min}} = 1 :: S_{\text{max}} = 60$
$D_{\text{min}} = 3_D :: D_{\text{max}} = 90_D$

$S_{\text{min}} = 1 :: S_{\text{max}} = 60$
$D_{\text{min}} = 3_D :: D_{\text{max}} = 64_D$

$S_{\text{min}} = 4 :: S_{\text{max}} = 56$
$D_{\text{min}} = 9_D :: D_{\text{max}} = 90_D$

$S_{\text{min}} = 4 :: S_{\text{max}} = 56$
$D_{\text{min}} = 69_D :: D_{\text{max}} = 90_D$

$S_{\text{min}} = 61 :: S_{\text{max}} = 79$
$D_{\text{min}} = 1_D :: D_{\text{max}} = 89_D$

$S_{\text{min}} = 61 :: S_{\text{max}} = 79$
$D_{\text{min}} = 7_D :: D_{\text{max}} = 85_D$

$S_{\text{min}} = 61 :: S_{\text{max}} = 79$
$D_{\text{min}} = 7_D :: D_{\text{max}} = 85_D$

$S_{\text{min}} = 61 :: S_{\text{max}} = 79$
$D_{\text{min}} = 7_D :: D_{\text{max}} = 85_D$

$S_{\text{min}} = 61 :: S_{\text{max}} = 79$
$D_{\text{min}} = 7_D :: D_{\text{max}} = 85_D$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_{10}$

---

- Partitioned on $S$: $S_{\text{min}} = 1 :: S_{\text{max}} = 99$
  - $D_{\text{min}} = 1_{10} :: D_{\text{max}} = 90_{10}$

- Partitioned on $D$: $D_{\text{min}} = 3_{10} :: D_{\text{max}} = 64_{10}$

- Sorted on $S$: $S_{\text{min}} = 1 :: S_{\text{max}} = 60$
  - $D_{\text{min}} = 3_{10} :: D_{\text{max}} = 64_{10}$

- Sorted on $D$: $D_{\text{min}} = 69_{10} :: D_{\text{max}} = 90_{10}$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

<table>
<thead>
<tr>
<th>page 1</th>
<th>page 2</th>
<th>page 3</th>
<th>page 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 9 15 20 24 33 44 60</td>
<td>4 14 19 29 32 40 52 56</td>
<td>67 79 84 86 87 91 95 99</td>
<td>61 63 71 72 73 78 80 94</td>
</tr>
</tbody>
</table>

1 I/O drop page

drop page

<table>
<thead>
<tr>
<th>partitioned on S</th>
<th>partitioned on D</th>
<th>sorted on S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{\min}=1 :: S_{\max}=99$</td>
<td>$D_{\min}=1_D :: D_{\max}=90_D$</td>
<td>$S_{\min}=1 :: S_{\max}=99$</td>
</tr>
<tr>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
</tr>
<tr>
<td>$S_{\min}=1 :: S_{\max}=60$</td>
<td>$S_{\min}=1 :: S_{\max}=60$</td>
<td>$S_{\min}=67 :: S_{\max}=99$</td>
</tr>
<tr>
<td>$D_{\min}=9_D :: D_{\max}=90_D$</td>
<td>$D_{\min}=9_D :: D_{\max}=90_D$</td>
<td>$D_{\min}=1_D :: D_{\max}=62_D$</td>
</tr>
<tr>
<td>$S_{\min}=67 :: S_{\max}=99$</td>
<td>$S_{\min}=67 :: S_{\max}=99$</td>
<td>$S_{\min}=61 :: S_{\max}=94$</td>
</tr>
<tr>
<td>$D_{\min}=65_D :: D_{\max}=89_D$</td>
<td>$D_{\min}=65_D :: D_{\max}=89_D$</td>
<td>$D_{\min}=65_D :: D_{\max}=89_D$</td>
</tr>
<tr>
<td>$S_{\min}=61 :: S_{\max}=60$</td>
<td>$S_{\min}=61 :: S_{\max}=60$</td>
<td>$S_{\min}=61 :: S_{\max}=60$</td>
</tr>
<tr>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
<td>$D_{\min}=3_D :: D_{\max}=64_D$</td>
</tr>
</tbody>
</table>

$S_{\min}=1 :: S_{\max}=60$

$D_{\min}=3_D :: D_{\max}=64_D$

$S_{\min}=1 :: S_{\max}=99$

$D_{\min}=1_D :: D_{\max}=90_D$

$S_{\min}=67 :: S_{\max}=99$

$D_{\min}=1_D :: D_{\max}=62_D$

$S_{\min}=61 :: S_{\max}=94$

$D_{\min}=65_D :: D_{\max}=89_D$
Key Weaving storage layout

- Reduced latency spikes
- Full page drops reduce superfluous I/Os

1M 1KB entries, buffer = file = 256 pages

% full page drops vs. fraction of deleted entries (%)
Key Weaving storage layout

- higher lookup cost
- reduced latency spikes
- full page drops reduces superfluous I/Os

1M point lookups, buffer = file = 256 pages, T=10

Graph: avg lookup cost (I/Os)
- Non–zero result lookup
- Zero result lookup

RocksDB

Full page drops reduces superfluous I/Os
the solution

**FAst DElete**

- amortized write
- reduced space
- improved read
- timely delete persistence

**KiWi**

- higher lookup cost
- reduced latency spikes
- full page drops reduces superfluous I/Os
the solution

FADE

KiWi

Lethe
delete tile size

lookup cost

secondary range delete cost
delete tile size

lookup cost

secondary range delete cost
delete tile size

WL-2
WL-1  WL-3

lookup cost

secondary range

delete cost

WL-1  WL-3
suboptimal state of the art design for workloads with deletes

FADE persists deletes timely using latency-driven compactions

KiWi supports efficient secondary range deletes by key-interweaved data layout

Lethe strikes balance between cost, performance, and latency