Enabling Efficient Deletes in Log-Structured KV-Stores

Subhadeep Sarkar
Enabling Efficient Deletes in Log-Structured KV-Stores

Lethe [ˈlē-thē] n: the goddess of forgetfulness
Log-Structured Merge-tree
LSM-tree
LSM-tree

The Log-Structured Merge-Tree (LSM-Tree) 1996

Patrick O'Neil¹, Edward Cheng²
Dieter Gawlick³, Elizabeth O'Neil¹
To be published: Acta Informatica
LSM-tree
O’Neil et al.
1996
LSM-tree
O’Neil et al.

1996

Bigtable

2006
Why LSM?
Why LSM?
Why LSM?
Why LSM?
Why LSM?
Why LSM ?
Why **LSM**?

- Tunable read-write performance
- Good space utilization
- Scales well
Research Trend

papers published

* data from DBLP
<table>
<thead>
<tr>
<th>Large-scale production</th>
<th>Internal db ops</th>
<th>Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZippyDB</td>
<td>table drop</td>
<td>CCPA (California)</td>
</tr>
<tr>
<td>25.2M/day</td>
<td>data migration</td>
<td>GDPR (EU, UK)</td>
</tr>
<tr>
<td>UP2X</td>
<td>cleanup /gc</td>
<td>VCPDA (Virginia)</td>
</tr>
<tr>
<td>Large-scale production</td>
<td>Internal db ops</td>
<td>Privacy</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>ZippyDB</strong> 25.2M/day</td>
<td><strong>table drop</strong></td>
<td><strong>CCPA (California)</strong></td>
</tr>
<tr>
<td><strong>UP2X</strong> 92.5M merge through deletes</td>
<td><strong>data migration</strong></td>
<td><strong>GDPR (EU, UK)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>cleanup /gc</strong></td>
<td><strong>VCPDA (Virginia)</strong></td>
</tr>
<tr>
<td>Large-scale production</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ZippyDB</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.2M/day</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>UP2X</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92.5M merge through deletes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal db ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>table drop</td>
</tr>
<tr>
<td>data migration</td>
</tr>
<tr>
<td>cleanup /gc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDPR (EU, UK)</td>
</tr>
<tr>
<td>CCPA (California)</td>
</tr>
<tr>
<td>VCPDA (Virginia)</td>
</tr>
</tbody>
</table>
on-demand

delete all data for user X within D days

rolling

keep deleting all data older than D days
What do **LSM** systems lack today?

- Supporting diverse delete types
- Time-bound deletes

**NO SUPPORT**
What do **LSM** systems lack today?

- Supporting diverse delete types
- Time-bound deletes
- Massive data movement
What do LSM systems need today?

time-bound deletes

overall performance

supporting diverse deletes
Outline

Part 1: LSM Basics

Part 2: The Problem: Deletes in LSMs

Part 3: Lethe: Enabling Efficient Deletes
LSM Basics

**key-value pairs**

<table>
<thead>
<tr>
<th>RID</th>
<th>timestamp</th>
<th>name</th>
<th>department</th>
<th>...</th>
<th>location</th>
</tr>
</thead>
</table>
LSM Basics

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>timestamp</td>
</tr>
<tr>
<td>name</td>
<td>department</td>
</tr>
<tr>
<td></td>
<td>location</td>
</tr>
</tbody>
</table>
buffer

LSM Basics
buffer 2614
LSM Basics
LSM Basics

buffer
LSM Basics

buffer

L1
LSM Basics

buffer

L1
LSM Basics

buffer

L1
LSM Basics

buffer  

L1
LSM Basics

buffer

L1

L2
LSM Basics

buffer

L1
L2
L3

compaction

exponentially larger capacity
LSM Basics

buffer

L1
L2
L3
LSM Basics

buffer  

L1
L2
L3
LSM Basics

buffer

L1
L2
L3
LSM Basics

buffer

L1

L2

L3
LSM Basics

buffer

L1
L2
L3
LSM Basics

buffer

burst of I/Os
prolonged write stalls
Partial Compaction

buffer

L1

L2

L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1

L2

L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1
L2
L3

NEW
NEW
Partial Compaction

buffer

L1
L2 NEW NEW
L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1
L2
L3
Partial Compaction

buffer

L1

L2

L3

amortized compaction cost

NEW

NEW

NEW
buffer

Lookups

L1
L2
L3
Lookups

buffer

fence
pointers

L1

L2

L3
Lookups

get(5) -> buffer

fence pointers

L1

L2

L3

1 I/O per run
get(5)

buffer

Bloom filters

fence pointers

L1

L2

L3

fewer disk I/Os

Lookups
Outline

Part 1: LSM Basics

Part 2: The Problem: Deletes in LSMs

Part 3: Lethe: Enabling Efficient Deletes
Deletes in LSMs

delete
Deletes in LSMs

delete := insert tombstone

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>TS flag</td>
</tr>
<tr>
<td>RID</td>
<td>timestamp</td>
</tr>
</tbody>
</table>
**Deletes** in LSMs

Delete \( := \) insert tombstone

```
key  value
RID  TS flag
```

```
key
RID  TS flag  timestamp  name  department  ...  location
```
Deletes in LSMs

delete(5)

buffer 5

L1

L2 5

L3
Deletes in LSMs

buffer

L1
L2
L3
Delete in LSMs

buffer → Bloom filters → fence pointers → L1 → L2 → L3

get(5)
The Problem
The Problem
Performance Roadblock
Performance Roadblock

space amplification
Performance Roadblock

L1

L2

L3

L4

space amplification
Performance Roadblock

L1: write amplification
L2: space amplification
L3
L4
Performance Roadblock

Bloom filters

L1

L2

write amplification

space amplification

L3

L4
Performance Roadblock

Bloom filters

L1

L2

L3

L4

poor read perf.
write amplification
space amplification
Performance Roadblock

Bloom filters

L1  

L2  

L3  

L4  

poor read perf.  
write amplification  
space amplification
The Problem

- poor read perf.
- write amplification
- space amplification
delete persistence latency
delete persistence latency
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: $D_{th}$
delete persistence latency

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

$L1$

$L2$

$L3$

$L4$
delete persistence latency

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

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

L1
L2
L3
L4
delete persistence latency

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

$$\sum_{i=1}^{L-1} t_i$$

unbounded delete persistence latency
The **Problem**

- poor read perf.
- write amplification
- space amplification

The persistence latency is approximated by the sum of the time differences between write and read operations, considering the unbounded delete operation:

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

The unbounded delete persistence latency is represented by the diagram on the right.
?
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: \textbf{D days}
deletes on a secondary attribute

delete all entries older than: **D days**

---

**key**
- RID
- TS flag
- timestamp
- name
- department
- ... location

**value**
- sort key
- delete key

**Diagram:**
- L1
- L2
- L3
- L4
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: \textbf{D days}

---

**Latency spikes**

**Superfluous I/Os**
deletes on a secondary attribute

delete all entries older than: \textbf{D days}

\begin{itemize}
  \item latency spikes
  \item superfluous I/Os
\end{itemize}
The Problem

- Poor read perf.
- Write amplification
- Space amplification

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

- Unbounded delete persistence latency
- Latency spikes
- Superfluous I/Os
Outline

Part 1: LSM Basics

Part 2: The Problem: Deletes in LSMs

Part 3: Lethe: Enabling Efficient Deletes
Outline

Part 1: LSM Basics

Part 2: The Problem: Deletes in LSMs

Part 3: Lethe: Enabling Efficient Deletes
poor read perf.
write amplification
space amplification

FADE

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

unbounded delete persistence latency

\[ D_{th} \]

latency spikes
superfluous I/Os
$D_{th}$

compaction trigger

compaction file picking policy
compaction trigger

- TTL
- age

compaction file picking policy

- tombstone density
- shallower level
- random

RocksDB, LevelDB, …
FAST DElete

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

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

$$
\sum_{i=1}^{L-1} d_i \leq D_{th} \quad \text{and} \quad d_i = T \cdot d_{i-1}
$$
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: $D_{th}$

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

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

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

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

\[
d_i = T \cdot d_{i-1}
\]
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: $D_{th}$

$L_1$

$L_2$

$L_3$

$L_4$

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

$$
d_i = T \cdot d_{i-1}
$$
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: $D_{th}$

$L_1$

$L_2$

$L_3$

$L_4$

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

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

L1

L2

L3

L4
FAst DElete

breaking ties in practical workloads

L1

L2

L3

L4
FAst DElete

timely delete persistence

within $D_{th}$

1M 1KB entries, 5% deletes, 1MB buffer, $T=10$
FAst DElete

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

- RocksDB
- FADE/16%
- FADE/25%
- FADE/50%

Reduced space amplification: 2.1 - 9.8x
Timely delete persistence: within $D_{th}$
improved read performance  
1.2 - 1.4 x  

reduced space amplification  
2.1 - 9.8 x  

timely delete persistence  
within $D_{th}$
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, $T=10$
- % deletes in workload
- total data written (GB)
- RocksDB
- FADE/16%
- FADE/25%
- FADE/50%
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 showing normalized bytes written over snapshot numbers for RocksDB and FADE/25% with 1M 1KB entries, 1MB buffer, T=10.](image)
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 $D_{th}$

Graph:
- 1M 1KB entries, 1MB buffer, T=10
- 0.0 - 1.5 normalized bytes written
- SN1, SN2, SN3, SN4, SN5 snapshot #
- Lines:
  - RocksDB
  - FADE/25%
FADE

KiWi

higher write amplification

improved read performance

reduced space amplification

timely delete persistence

latency spikes

superfluous I/Os
FASTDELETE

KiWi

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

latency spikes

reduced space amplification
Key Weaving storage layout

delete all entries older than: \textbf{D days}

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 $\leq 65_D$

SST file

partitioned on $S$
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

SST file

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 $\leq 65_D$

SST file

partitioned on $S$

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

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

$S_{\text{min}}=1 :: S_{\text{max}}=24$

$D_{\text{min}}=3_D :: D_{\text{max}}=80_D$

$S_{\text{min}}=29 :: S_{\text{max}}=60$

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

$S_{\text{min}}=1 :: S_{\text{max}}=24$

$D_{\text{min}}=3_D :: D_{\text{max}}=80_D$

$S_{\text{min}}=29 :: S_{\text{max}}=60$

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

$S_{\text{min}}=61 :: S_{\text{max}}=79$

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

$S_{\text{min}}=80 :: S_{\text{max}}=99$

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

page 1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>14</th>
<th>15</th>
<th>19</th>
<th>20</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>34_D</td>
<td>69_D</td>
<td>3_D</td>
<td>79_D</td>
<td>8_D</td>
<td>80_D</td>
<td>23_D</td>
<td>24_D</td>
</tr>
</tbody>
</table>

page 2

<table>
<thead>
<tr>
<th></th>
<th>29</th>
<th>32</th>
<th>33</th>
<th>40</th>
<th>44</th>
<th>52</th>
<th>56</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>88_D</td>
<td>90_D</td>
<td>28_D</td>
<td>74_D</td>
<td>9_D</td>
<td>76_D</td>
<td>81_D</td>
<td>64_D</td>
</tr>
</tbody>
</table>
Key Weaving storage layout

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

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

SST file

partitioned on $S$

partitioned on $D$

drop page
Key Weaving storage layout

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

partitioned on \(S\)

sorted on \(S\)

drop page
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

SST file

page 1

page 2

page 3

page 4

partitioned on $D$

sorted on $S$

drop page

delete tile 1

delete tile 2
Key Weaving storage layout

delete all entries with timestamp $\leq 65_D$

SST file

partitioned on $S$

sorted on $S$

partitioned on $D$

drop page

1 I/O

drop page

1 I/O
Key Weaving storage layout

Internals of a file in KiWi

% reduction in disk I/Os

fraction of deleted entries (%)
Key Weaving storage layout

Internals of a file in KiWi

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

% reduction in disk I/Os

fraction of deleted entries (%)
Key Weaving storage layout

Internals of a file in KiWi

8 pages/delete tile

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

% reduction in disk I/Os

fraction of deleted entries (%)
Key Weaving storage layout

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

% reduction in disk I/Os

fraction of deleted entries (%)
Key Weaving storage layout

Internals of a file in KiWi

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

% reduction in disk I/Os

fraction of deleted entries (%)
Key Weaving storage layout

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

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

% reduction in disk I/Os vs. fraction of deleted entries (%)

- h=1
- h=4
- h=8
- h=16
- h=32
- h=64
- h=128
- h=256

Reduced latency spikes and full page drops reduce superfluous I/Os.
Key Weaving storage layout

- higher lookup cost
  - increased lookup cost

- reduced latency spikes
  - decreased latency spikes

- full page drops reduces superfluous I/Os
  - reduced superfluous I/Os

---

Graph:

- X-axis: delete–tile granularity (log scale)
- Y-axis: avg lookup cost (I/Os)

Legend:

- Non–zero result lookup
- Zero result lookup

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

FADE

amortized write

improved read

timely delete persistence

higher lookup cost

reduced space

reduced latency spikes

full page drops reduces superfluous I/Os

KiWi
FADE → KiWi

Lethe
delete tile size

lookup cost

secondary range delete cost
delete tile size

lookup cost

secondary range
delete cost

P
delete tile size

lookup cost

secondary range delete cost
delete tile size

lookup cost

secondary range
delete cost
delete tile size

lookup cost

secondary range

delete cost
suboptimal state-of-the-art design for workloads with deletes

FADE persists deletes timely using latency-driven compactions

KiWi supports efficient secondary range deletes using key-interweaved data storage
Lethe strikes balance between cost, performance, and latency

- FADE persists deletes timely using latency-driven compactions
- KiWi supports efficient secondary range deletes by key-interweaved data layout

Thank You!

disc-projects.bu.edu/lethe/