SHEARS: Persisting Deletes in LSM Trees

Megan Fantes, Ketill Guðmundsson, Nikhilesh Murugavel, Allison Weaver
Introduction

- Unoptimized deletes in LSM trees
- Deleted value is logically hidden
- **Problem?**
  - Increased tree size
  - Compromise in security
- **Goal?**
  - Faster persistent deletes
Background

**Mem-table:** Data structure (skip-list) in memory

**SSTs:** Sorted Sequence Table in disk

**Out-of-place updates:** When data is updated, it is added as a new key-value pair in the mem-table

**Tombstones:** Used to mark a key as deleted

**Persistent Deletes:** When a deleted key-value pair is removed from the tree

**Compaction/Partial Merging:** Some of level-L SSTs merged with level-(L+1) SSTs
Our Solution
SHEARS Design

Our additions:

- Sequencer
- Sequence Number
- Tombstone buffer
- Tombstone Group

What it does:

- Three Way merge
- Inserts deletes
- Does not guarantee persistent deletes
- Better read performance and increases storage space
How it works:

In-memory:
- Tombstone Buffer
  - Added to forming file in mem-table
- Mem-table
  - Deletes are copied into buffer
  - File flushed to L0
  - Sequencer assigns sequence number to file

On the tree:
- L0
- L1
- L2
- LN

Sequencer gives number of 5-1 = 4 to tombstone

Transition between L1 and L2 results in partial merge

Incoming key-value pair
Merge Policy

LSM-tree

L

... 68 45 70 ...

L+1

... 10 37 92 ...

Tombstone Buffer

... 90 92 94 96 ...

Merge Policy

**LSM-tree**

<table>
<thead>
<tr>
<th>L</th>
<th>...</th>
<th>68</th>
<th>...</th>
</tr>
</thead>
</table>

| L+1 | ... | 94 | 94 | 94 | ... |
K-way Merge

- Merge many tombstone groups in a compaction
- Makes system more fluid
- Downside is higher compaction cost
- Possible with the use of priority cue
- Additional cost is now $O(n \log k)$
K-way Merge

Tombstone Buffers

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>10</td>
<td>12</td>
<td>54</td>
<td>72</td>
<td>...</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>26</td>
<td>30</td>
<td>40</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>58</td>
<td>82</td>
<td>94</td>
<td>...</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>60</td>
<td>66</td>
<td>...</td>
</tr>
</tbody>
</table>

Diagram:

- Node 10
- Node 26
- Node 12
- Node 60

Connections:
- 10 → 26
- 12 → 60
Deleting Tombstone Groups

- Keep track of lowest sequence numbers in LSM tree
  - If any tombstone group has lower number then delete
  - Costly operation
  - Runs in background periodically

- Backup systems in place of overflow
  - Deletions happening too slow
  - Discard oldest tombstone groups
Experiments
Experiments: Storage, Latency, CPU Load

- Hypothesis: SHEARS uses more memory, less disk space
- Hypothesis: SHEARS increases write latency, decreases read latency
- Hypothesis: sorting, merging increases CPU load
Experiments: Bloom filters, SeqNum Distribution

- Hypothesis: SHEARS decreases false positives by pruning LSM tree
- Insight: tracking the min. SeqNum to define delete policy
Experiments: Force deletes, delete persistence

- Trade-offs: increased CPU/IO cost, memory used, delete persistence
- Hypothesis: SHEARS persists deletes faster (that is the point)
Thank You

Questions?