LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data

Xingbo Wu, Yuehai Xu, Zili Shao, and Song Jiang
Wayne State University, \{wuxb,yhxu,sjiang\}@wayne.edu
The Hong Kong Polytechnic University, cszlshao@comp.polyu.edu.hk
Presenter: Xuan Wang
Main Point

• LSM-trie is designed to manage a large set of small data.

• It reduces the write-amplification by an order of magnitude.

• It delivers high throughput even with out-of-core metadata.
1. “The indices and Bloom filters in a KV store can grow very large.” Use an example to show that these **metadata** in LevelDB may **have to be out of core**.

- **Metadata** in LevelDB includes indices and bloom filters.

- **Out of core** means not on the memory.

- Why memory cannot handle all of the indices and bloom filters?
1. “The indices and Bloom filters in a KV store can grow very large.” Use an example to show that these metadata in LevelDB may have to be out of core.

- 10TB Hard Drive
- Each KV pair suppose to take 50B space
  - 10TB/50B = 20 Billion
- Each KV pair require 10 bit-per-key bloom filter
  - 20 Billion * 10 bit is around 250 GB bloom filter
- Each KV pair require 1~2 bit index
  - 20 Billion * 1 bit is around 25 GB indices
2. “Therefore, the Bloom filter must be beefed up by using more bits.” Use an example to show why the Bloom filters have to be longer?

- **False Positive** will increase the disk read

<table>
<thead>
<tr>
<th>bits/key</th>
<th>50 Levels</th>
<th>100 Levels</th>
<th>150 Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>40.95%</td>
<td>81.90%</td>
<td>122.85%</td>
</tr>
<tr>
<td>12</td>
<td>15.70%</td>
<td>31.40%</td>
<td>47.10%</td>
</tr>
<tr>
<td>14</td>
<td>6.00%</td>
<td>12.00%</td>
<td>18.00%</td>
</tr>
<tr>
<td>16</td>
<td>2.30%</td>
<td>4.59%</td>
<td>6.89%</td>
</tr>
<tr>
<td>18</td>
<td>0.88%</td>
<td>1.76%</td>
<td>2.64%</td>
</tr>
</tbody>
</table>

*Table 1: Bloom filter false-positive rate.*
2. “Therefore, the Bloom filter must be beefed up by using more bits.” Use an example to show why the Bloom filters have to be longer?

- For LSM-trie (32MB Htables and Amplification Factor is 8)
  - For a 10TB hard disk.
    - The first four level has 32-sublevels and the fifth level require 80 sublevels
  - Total would be 112 sublevels.

<table>
<thead>
<tr>
<th>bits/key</th>
<th>50 Levels</th>
<th>100 Levels</th>
<th>150 Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>40.95%</td>
<td>81.90%</td>
<td>122.85%</td>
</tr>
<tr>
<td>12</td>
<td>15.70%</td>
<td>31.40%</td>
<td>47.10%</td>
</tr>
<tr>
<td>14</td>
<td>6.00%</td>
<td>12.00%</td>
<td>18.00%</td>
</tr>
<tr>
<td>16</td>
<td>2.30%</td>
<td>4.59%</td>
<td>6.89%</td>
</tr>
<tr>
<td>18</td>
<td>0.88%</td>
<td>1.76%</td>
<td>2.64%</td>
</tr>
</tbody>
</table>

Table 1: Bloom filter false-positive rate.
3. What’s the difference between **SSTable in LevelDB** and **HTable in LSM-trie**?

- Sorted by index
- Index is needed for locating a block
3. What’s the difference between SSTable in LevelDB and HTable in LSM-trie?

- Each block is considered as a bucket for receiving KV items whose keys are hashed into it.
- No index

![Diagram showing the structure of an HTable.]

Figure 4: The structure of an HTable.
3. What’s the difference between **SSTable in LevelDB** and **HTable in LSM-trie**?

- **Structure:**
  - **LevelDB:**
  - Exponential growth each level
  - **LSM-trie:**
  - Linear growth sublevel and exponential intra level

(a) Exponential growth pattern in LevelDB.

(b) Alternating use of linear and exponential growth patterns.
3. What’s the difference between **SSTable in LevelDB** and **HTable in LSM-trie**?

- **Lookup:**
  - **SSTable:**
    - Searching in the index
    - Check with bloom filter
    - Retrieve data
  - **HTable:**
    - Generate the **hashkey** by SHA-1
    - Check with **cluster bloom filter**
    - Retrieve data
3. What’s the difference between SSTable in LevelDB and HTable in LSM-trie?

- HashKey generated by SHA-1:
  - Prefix is used for check the location of the KV pair in which HTable of the LSM-trie
  - Suffix is used for check the location of the KV pair in which bucket of the HTable

![SHA-1 value (160 bits)]

**Figure 8**: Use of a 160-bit SHA1 key. (1) The prefix is used for trie encoding. (2) The infix is used for sorting KV items in a bucket. (3) The suffix is used for locating the KV items in an HTable.
3. What’s the difference between SSTable in LevelDB and HTable in LSM-trie?

- Cluster bloom filter in LSM-trie:
  - One bloom filter check for one level

![Diagram showing clustering of bloom filters](image)

**Figure 9: Clustering Bloom filters**
3. What’s the difference between SSTable in LevelDB and HTable in LSM-trie?

- Compaction:
  - LevelDB:
    - Compact the L0 into L1
    - WA = 11 if each level is 10 times larger than previous level
3. What’s the difference between SSTable in LevelDB and HTable in LSM-trie?

- Compaction:
  - LSM-trie:
    - Compact L0 into L1
      - WA = 1
4. “However, a challenging issue is whether the buckets can be load balanced in terms of aggregate size of KV items hashed into them” Why may the buckets in an HTable be load unbalanced? How to correct the problem?

- According to Zipf’s law, although we randomly generate the data. It still would be standard normal distribution

![Normal Distribution Diagram](image)

**Figure 5**: Distribution of bucket load across buckets of an HTable with a uniform distribution of KV-item size and an average size of 100 B (a), 200 B (b), and 300 B (c). The keys follow the Zipfian distribution. For each plot, the buckets are sorted according to their loads in terms of aggregate size of KV items in a bucket.
4. "However, a challenging issue is whether the buckets can be load balanced in terms of aggregate size of KV items hashed into them." Why may the buckets in an HTable be load unbalanced? **How to correct the problem?**

- Sort the buckets according to the load of the KV pairs
- Move from the most overloaded to the most underloaded

**Three concerns:**
- How to know an KV item has been moved
- How to reduce the chance one item keep moving
- How to deal with the large item that cannot be moved

(a) KV items are assigned to the buckets by the hash function, causing unbalanced load distribution.

(b) Buckets are sorted according to their loads and balanced by using a greedy algorithm.
4. “However, a challenging issue is whether the buckets can be load balanced in terms of aggregate size of KV items hashed into them” Why may the buckets in an HTable be load unbalanced? **How to correct the problem?**

- First concern:
  - HashMark set
  - Bloom Filter would not change

Figure 5: Distribution of bucket load across buckets of an HTable with a uniform distribution of KV-item size and an average size of 100 B (a), 200 B (b), and 300 B (c). The keys follow the Zipfian distribution. For each plot, the buckets are sorted according to their loads In terms of aggregate size of KV items in a bucket.

Figure 7: Bucket load distribution after retaining.
4. “However, a challenging issue is whether the buckets can be load balanced in terms of aggregate size of KV items hashed into them” Why may the buckets in an HTable be load unbalanced? **How to correct the problem?**

- Second concern:
  - Infix is used to move the overflowed:

![Diagram](image)

**Figure 8**: Use of a 160-bit SHA1 key. (1) The prefix is used for trie encoding. (2) The infix is used for sorting KV items in a bucket. (3) The suffix is used for locating the KV items in an HTable.
4. “However, a challenging issue is whether the buckets can be load balanced in terms of aggregate size of KV items hashed into them” Why may the buckets in an HTable be load unbalanced? How to correct the problem?

• Third concern:
  • Every bucket load until 95%
    • Some of the overflown cannot be moved to another bucket
    • Create a special bucket with fully indexed with HTable file.
Question?