Systems Project: Implementation of an LSM Tree
What if I told you we built...

**Read-Optimized**
Log(n)!

**Write-Optimized**
Instant Writes

**Small Memory Footprint!**
Only a block in memory at a time!
We would be **Lying!**

*But now we have your attention...*
Introduction to LSM trees
Design Goals
System Specifications
Experimentation
Introduction
What we actually built

**LSM - Tree:**

Data Structure that performs well in reads and vastly outperforms B-trees in writing.

Why? Because avoids constant dispersed update operations

**Intuition:** Multiple levels of sorted runs, with external log to speed up reads.
Design Goals

Full Functionality
Support operations:
- Write
- Update
- Delete
- Point Read
- Range Read

Merging Policies
- Tiering:
  - Flush Now, Merge tomorrow
- Leveling:
  - Flush now, Merge now.

Performance
The trinity of Data:
- Fast Writes (the whole reason behind LSM)
- Decent Reads (specially using leveling)
- Low Memory Footprint
The Smallest unit

- Unique key: Duplicates are interpreted as updates
- Value: String
- Tombstone: Supports deletion
The **Buffer** Class

- Tunable size by user
- Sorted inserts
- No duplicates: On time Deletions and Updates
- Merging-policy specific flushing

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>value</td>
<td>False</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>47</td>
<td>value</td>
<td>False</td>
</tr>
<tr>
<td>76</td>
<td>value</td>
<td>False</td>
</tr>
</tbody>
</table>

...
**System Description**

- Metadata, Fence Pointers
- Key, Value pairs

The **LSM Class**
- Policy Specific Drivers
- Reads Support – Fence Pointers
- Writes – Overview
  - Flushing Buffers
  - Flushing Levels
  - Merging Levels/Tiers
3 Experimentation
What we asked:

**Scalability: System performance**
- How does performance scale with the number of operations?
- How does the workload impact the scalability?

**Tuning: Buffer Size and Ratio**
- What is the impact of changing this values on our system?
- What values are best for read intensive workloads?
- What values are best for write intensive ones?
Results - Scalability

- Total Seconds per Workload - Write Heavy
- Total Seconds per Operation - Update Heavy
- Total Seconds per Workload - Point Read Heavy
- Total Seconds per Workload - Range Read Heavy
Results - Knobs

- **Total Seconds Depending on Knobs - Writes**
  - Q = 50, T = 6
  - Q = 50, T = 3
  - Q = 25, T = 6
  - Q = 25, T = 3

- **Total Seconds Depending on Knobs - Point Reads**
  - Q = 50, T = 6
  - Q = 50, T = 3
  - Q = 25, T = 6
  - Q = 25, T = 3

- **Total Seconds Depending on Knobs - Update**
  - Q = 50, T = 6
  - Q = 50, T = 3
  - Q = 25, T = 6
  - Q = 25, T = 3

- **Total Seconds Depending on Knobs - Range Reads**
  - Q = 50, T = 6
  - Q = 50, T = 3
  - Q = 25, T = 6
  - Q = 25, T = 3
Thanks!

Any questions?
### Appendix

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of Operations</th>
<th>$Q = 50, T = 6$</th>
<th>$Q = 50, T = 3$</th>
<th>$Q = 25, T = 6$</th>
<th>$Q = 25, T = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Write</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.137487</td>
<td>0.217731</td>
<td>0.204025</td>
<td>0.275852</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>1.723890</td>
<td>2.118570</td>
<td>2.450830</td>
<td>2.943550</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>10.784900</td>
<td>13.165600</td>
<td>14.901100</td>
<td>18.458500</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>23.005700</td>
<td>28.546000</td>
<td>28.834700</td>
<td>35.180900</td>
</tr>
<tr>
<td><strong>Tests - Update/Delete He</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.148235</td>
<td>0.180082</td>
<td>0.219480</td>
<td>0.268841</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>2.042160</td>
<td>2.279290</td>
<td>2.646050</td>
<td>3.083380</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>10.719600</td>
<td>13.261000</td>
<td>14.004200</td>
<td>17.611300</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>21.879200</td>
<td>29.099300</td>
<td>31.239100</td>
<td>37.327200</td>
</tr>
<tr>
<td><strong>Point Reads</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.147790</td>
<td>0.188584</td>
<td>0.169152</td>
<td>0.226547</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>1.716690</td>
<td>2.134470</td>
<td>1.999680</td>
<td>2.515850</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>8.860930</td>
<td>11.631600</td>
<td>11.857000</td>
<td>13.502600</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>20.908600</td>
<td>24.368100</td>
<td>25.018700</td>
<td>30.041700</td>
</tr>
<tr>
<td><strong>Range Reads</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.147096</td>
<td>0.166553</td>
<td>0.168165</td>
<td>0.199130</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>1.485910</td>
<td>2.692720</td>
<td>1.764910</td>
<td>2.865070</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>8.652140</td>
<td>10.179600</td>
<td>10.640000</td>
<td>11.784300</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>15.519000</td>
<td>20.028000</td>
<td>23.126100</td>
<td>25.806000</td>
</tr>
</tbody>
</table>