ENDURE: A Robust Tuning Paradigm for LSM Trees Under Workload Uncertainty

Andy Huynh, Harshal A. Chaudhari, Evimaria Terzi, Manos Athanassoulis
Age of Log-Structured Merge-Trees

How do we go about tuning these knobs?
LSM Trees – Mechanics

Buffer fills $\rightarrow$ Sort and Flush to disk

Compaction policy: $\pi$
Tiering (↑) or Leveling (↓)

$\mu$: Compaction Policy
$T$: Size Ratio
$\text{m}_\text{filter}$: Filter memory
$\text{m}_\text{buff}$: Buffer memory
LSM Trees – A Point Read

How do we define a workload?

Buffer

Level 1

Level 2

... 

Level L

\[ m_{\text{buff}} < k, v > \]

\[ m_{\text{filter}} \]

READ: 62
Query Types

Workload: \((z_0, z_1, q, w)\)

Empty Reads: \(z_0\)

Non-Empty Reads: \(z_1\)

Range Reads: \(q\)

Writes: \(w\)

Cool! How do we go about tuning?
The LSM-Tuning Problem

\( w : \) Workload \( (z_0, z_1, q, w) \)

\( \Phi : \) LSM Tree Design \( (m_{buff}, m_{filter}, T, \pi) \)

\( C : \) Cost

\[ \Phi^* = \arg\min_\Phi C(w, \Phi) \]
The LSM-Tuning Problem

\( w : \text{Workload} \ (z_0, z_1, q, w) \)

\( \Phi : \text{LSM Tree Design} \ (m_{\text{buff}}, m_{\text{filter}}, T, \pi) \)

\( C : \text{Cost (I/O)} \)

\[ \Phi^* = \arg\min_\Phi C(w, \Phi) \]

Define our cost function

\[ C'(\hat{w}, \Phi) = \hat{w}^\top c(\Phi) = z_0 \cdot Z_0(\Phi) + z_1 \cdot Z_1(\Phi) + q \cdot Q(\Phi) + w \cdot W(\Phi) \]
Tuning Problems

\[ w_0 : \text{Workload } (z_0, z_1, q, w) \]

Optimal configuration for the workload

Optimal tuning depends on workload

Workload uncertainty leads to sub-optimal tuning
ENDURE So Far

Introduction

LSM Trees Notation

Nominally Tuning LSM Trees

**ENDURE: Robustly Tuning LSM Trees**

The ENDURE Pipeline

ENDURE Evaluation
The LSM-Tuning Problem

\( \mathbf{w} : \) Workload \((z_0, z_1, q, w)\)

\( \Phi : \) LSM Tree Design \((m_{buff}, m_{filter}, T, \pi)\)

\( C : \) Cost (I/O)

\[ \Phi^* = \arg\min_{\Phi} C(\mathbf{w}, \Phi) \]

\( U_{\mathbf{w}}^\rho : \) Uncertainty Neighborhood of Workloads

\( \rho : \) Size of this neighborhood

\[ \Phi^* = \arg\min_{\Phi} C(\mathbf{\hat{w}}, \Phi) \]

s. t., \( \mathbf{\hat{w}} \in U_{\mathbf{w}}^\rho \)
Robust Tuning

\[ w_0 : \text{Workload} \ (z_0, z_1, q, w) \]

\[ \Phi^* = \arg\min_{\Phi} C(\hat{w}, \Phi) \]
\[ \text{s. t.,} \ \ \hat{w} \in U_w^\rho \]

- Optimal configuration for the workload
- Robust configuration for the workload neighborhood
Uncertainty Neighborhood

Neighborhood of workloads ($\rho$) via the KL-divergence

$$I_{KL}(\tilde{w}, w) = \sum_{i=1}^{m} \tilde{w}_i \cdot \log \left( \frac{\tilde{w}_i}{w_i} \right)$$

Expected $\rho$?

Historical workloads
maximum/average uncertainty among workload pairings

User provided workload uncertainty

$U^*_W$: Uncertainty Neighborhood of Workloads
$\rho$: Size of this neighborhood
ENDURE Pipeline

Workload Characteristic

System Information
Page Size
Memory Budget

ENDURE Solves the Robust Problem

Expected performance

RocksDB Configuration
Testing Suite

ENDURE in Python, implemented in tandem with RocksDB

Uncertainty benchmark
- 15 expected workloads
- 10K randomly sampled workloads as a test-set

Normalized delta throughput

\[ \Delta_w(\Phi_1, \Phi_2) = \frac{1/C(w,\Phi_2) - 1/C(w,\Phi_1)}{1/C(w,\Phi_1)} \]

Nominal vs Robust: > 0 is better
1 means 2x speedup

<table>
<thead>
<tr>
<th>Index</th>
<th>(z_0, z_1, q, w)</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25% 25% 25% 25%</td>
<td>Uniform</td>
</tr>
<tr>
<td>1</td>
<td>97% 1% 1% 1%</td>
<td>Unimodal</td>
</tr>
<tr>
<td>2</td>
<td>1% 97% 1% 1%</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1% 1% 97% 1%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1% 1% 1% 97%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>49% 49% 1% 1%</td>
<td>Bimodal</td>
</tr>
<tr>
<td>6</td>
<td>49% 1% 49% 1%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>49% 1% 1% 49%</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1% 49% 49% 1%</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1% 49% 1% 49%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1% 1% 49% 49%</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>33% 33% 33% 1%</td>
<td>Trimodal</td>
</tr>
<tr>
<td>12</td>
<td>33% 33% 1% 33%</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>33% 1% 33% 33%</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1% 33% 33% 33%</td>
<td></td>
</tr>
</tbody>
</table>
Relationship of Expected and Observed $\rho$

**Observed $\rho$:** distance from executed workload to expected workload

**Expected $\rho$:** workload given to tuner

**Highest throughput** when observed and expected $\rho$ **match**

**Lowest throughput** when $\rho$ is **mismatched**
RocksDB instance setup with 10 million unique key-value pairs of size 1KB

Each observation period is 200K queries, with 5 observations per session 6 million queries to the DB

Writes are unique, range queries average 1-2 pages per level
Workload Sequence

Model I/O

Nominal
- h: 8.2, T: 8.4
- \(m\): Tiering

Robust
- h: 1.0, T: 4.7
- \(m\): Leveling

\(w^*_7\) : (49%, 1%, 1%, 49%)
\(\hat{w}\) : (32%, 47%, 22%, 0%)

System I/O

\(\rho\) : 2.31
\(I_{KL}(\hat{w}, w)\) : 2.31

1. Reads
   (30%, 58%, 12%, 0%)
2. Range
   (7%, 10%, 84%, 0%)
3. Empty Reads
   (86%, 9%, 4%, 0%)
4. Non-Empty Reads
   (8%, 86%, 6%, 0%)
5. Reads
   (30%, 58%, 12%, 0%)
6. Reads
   (30%, 58%, 12%, 0%)
Workload Sequence

Model I/O
Nominal: h: 8.2, T: 8.4
Robust: h: 1.0, T: 4.7
π: Tiering
π: Leveling

System I/O
ρ: 2.31
$I_{KL}(\hat{\mathbf{w}}, \mathbf{w})$: 2.31

System Latency
Throughput: 2.6 kQPS
5.7 kQOPS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(30%, 58%, 12%, 0%)</td>
<td>(7%, 10%, 84%, 0%)</td>
<td>(86%, 9%, 4%, 0%)</td>
<td>(8%, 86%, 6%, 0%)</td>
<td>(30%, 58%, 12%, 0%)</td>
<td>(30%, 58%, 12%, 0%)</td>
</tr>
</tbody>
</table>

120% ↑
Workload Sequence

Small subset of results! Take a look at the paper for a more detailed analysis.
Thanks!

Workload uncertainty creates suboptimal tunings

ENDURE: robust tuning using neighborhood of workloads

Deployed ENDURE on RocksDB

Check out our poster tonight for more info!

disc.bu.edu/
www.ndhuynh.com/
@nd_huynh