The Case for Learned Index Structures

Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N.
Background
Index Structures

• Index structures are built for efficient data access
• E.g. B-Trees
Index Structures as Models

(a) B-Tree Index

Key

BTree

pos

pos - 0  pos + pagezise
Index Structures as Models

(a) B-Tree Index

Key

\[ \text{BTree} \]

\[ \text{pos} \]

\[ \text{pos - 0} \quad \text{pos + pagezise} \]

(b) Learned Index

Key

Model (e.g., NN)

\[ \text{pos - min_err} \quad \text{pos + max_err} \]
Range Index
Range Index Models = CDF Models

True position $p^*$
$= \text{rank}(\text{key})$
$= |\{k | k \leq \text{key}\}|$
$= P(X \leq \text{key}) \times N$

$P(X \leq \text{key})$ is the CDF of keys
Range Index Models = CDF Models

Model:
\[ pos = F(key) \times N \approx p^* \]

\[ F(key) \approx P(X \leq key) \]
The Recursive Model Index (RMI)

- Prediction from previous stage chooses the next model
- Progressively refine the prediction

Figure 3: Staged models
The Recursive Model Index

• Benefits
  • Decouple execution cost & model size
  • Notion of progressively learning the shape of CDF
  • Divide the space into smaller ranges, easier to refine the final prediction

Figure 3: Staged models
The Recursive Model Index

- Worst case performance
  - If last stage models do not meet error requirement, replace by B-Trees
  - Have same worst case guarantee as B-Trees

Figure 3: Staged models
The Recursive Model Index - Training

• Loss defined as:

\[ L = \sum_{(x,y)} (f(x) - y)^2 \]

• Simple model trained in seconds, Neural Nets in minutes

Figure 3: Staged models
Experiments

• Integer Datasets
  • Weblogs dataset contains 200M log entries
  • Maps dataset indexed the longitude of ≈ 200M user-maintained features
  • Log-normal dataset synthesized by sampling 190M unique values

• Models
  • 2-stage RMI model having second-stage sizes (10k, 50k, 100k, and 200k)
  • Read-optimized B-Tree with different page sizes
## Results

<table>
<thead>
<tr>
<th>Type</th>
<th>Config</th>
<th>Map Data</th>
<th>Web Data</th>
<th>Log-Normal Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Size (MB)</td>
</tr>
<tr>
<td>Btree</td>
<td>page size: 32</td>
<td>52.45 (4.00x)</td>
<td>274 (0.97x)</td>
<td>198 (72.3%)</td>
</tr>
<tr>
<td></td>
<td>page size: 64</td>
<td>26.23 (2.00x)</td>
<td>277 (0.96x)</td>
<td>172 (62.0%)</td>
</tr>
<tr>
<td></td>
<td>page size: 128</td>
<td>13.11 (1.00x)</td>
<td>265 (1.00x)</td>
<td>134 (50.8%)</td>
</tr>
<tr>
<td></td>
<td>page size: 256</td>
<td>6.56 (0.50x)</td>
<td>267 (0.99x)</td>
<td>114 (42.7%)</td>
</tr>
<tr>
<td></td>
<td>page size: 512</td>
<td>3.28 (0.25x)</td>
<td>286 (0.93x)</td>
<td>101 (35.3%)</td>
</tr>
<tr>
<td>Learned Index</td>
<td>2nd stage models: 10k</td>
<td>0.15 (0.01x)</td>
<td>98 (2.70x)</td>
<td>31 (31.6%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 50k</td>
<td>0.76 (0.06x)</td>
<td>85 (3.11x)</td>
<td>39 (45.9%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 100k</td>
<td>1.53 (0.12x)</td>
<td>82 (3.21x)</td>
<td>41 (50.2%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 200k</td>
<td>3.05 (0.23x)</td>
<td>86 (3.08x)</td>
<td>50 (58.1%)</td>
</tr>
</tbody>
</table>

*Figure 4: Learned Index vs B-Tree*
Point Index
Point Index

• Example: hash-map
• Deterministically map keys to positions inside an array
The Hash-Model Index

• Build a hash function based on the CDF of the data ($M$ is size of hash-map):

$$h(key) = F(key) \times M$$
$$F(key) \approx P(X \leq key)$$
The Hash-Model Index

- Main objective is to reduce number of conflicts
  - Conflicts could induce high cost depending on architecture (e.g. distributed)
Experiments

- Learned models with same settings as in range index
- Compared against MurmurHash3-like hash-function

<table>
<thead>
<tr>
<th></th>
<th>% Conflicts Hash Map</th>
<th>% Conflicts Model</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Data</td>
<td>35.3%</td>
<td>07.9%</td>
<td>77.5%</td>
</tr>
<tr>
<td>Web Data</td>
<td>35.3%</td>
<td>24.7%</td>
<td>30.0%</td>
</tr>
<tr>
<td>Log Normal</td>
<td>35.4%</td>
<td>25.9%</td>
<td>26.7%</td>
</tr>
</tbody>
</table>

*Figure 8: Reduction of Conflicts*
Existence Index
Existence Index

• Example: Bloom filters
• Return whether a key exists in a dataset
• No false negatives, but has potential false positives
Bloom filters as a Classification Problem

- Binary probabilistic classification task: Whether key exists in dataset
Bloom filters as a Classification Problem

- Guarantee for no false negative
- Overflow bloom filter: remember false negatives from models
Experiments

• Data
  • 1.7M blacklisted phishing URLs
  • Negative set: random URLs + whitelisted URLs

• Comparison
  • Learned filter: RNN with GRU
  • Normal Bloom filter

Figure 10: Learned Bloom filter improves memory footprint at a wide range of FPRs. (Here \( W \) is the RNN width and \( E \) is the embedding size for each character.)
Critique
Major Contributions

1. Proposed the idea of applying machine learning in index structures
2. Solutions to offering guarantees on performance, determinism with ML models
3. Showed significant performance improvements (time and space)
4. Inspired new research direction (27 citations since June 2018)
Criticism

1. Detail of platform used for experiments not given
2. Little discussion on training time
3. Experiments on CPU only
Conclusion & Future Direction

• Proposed a new direction in database research that
  • Makes effective use of machine learning methods
  • Shows promising preliminary results
  • Inspired new research work

• Requires more details on performance evaluation

• Potentials in learned algorithms, multi-dimensional indexes