The Case for Learned Index Structures

June 7, 2018
Problems in the existing systems

▷ do not take advantage of more common patterns prevalent in real-world data. Continuous integer keys (e.g., the key 1 to 100M)
▷ the effort to build specialized solutions for every use case is usually too high.
Solutions

- machine learning
to learn a model that reflects the patterns in the data.
New Problems

- semantic guarantees
  - many data structures can be decomposed into a learned model and an auxiliary structure to provide the same semantic guarantees.
  - continuous functions, describing the data distribution, can be used to build more efficient data structures or algorithms.

- overfitting

- no specialization
  - rollback

- evaluate efficiency
  - learning index framework (LIF)
Outline

Range Index

Point Index

Existence Index
B-Tree
B-Trees are Models

(a) B-Tree Index

Key

BTree

pos

pos - 0 pos + pagezise

(b) Learned Index

Key

Model (e.g., NN)

pos

pos - min_err pos + max_err
What Model Complexity Can We Afford?

BTree:

- page-size 100
- traversing a single page with binary search takes 50 cycles
- a modern CPU can do 8-16 SIMD operations per cycle
  \[ \frac{1}{100} \text{ per } 50 \times 8 = 400 \text{ arithmetic operations} \]
- A single cache-miss costs 50-100 additional cycles
What Model Complexity Can We Afford?

A model will be faster as long as it has a better precision gain than $1/100$ per 400 arithmetic operations.

**GPU/TPU**

- NVIDIA Tesla V100: 60000 operations per cycle
- high latency: batching
- the number of floating/int operations per second of GPUs/TPUs will increase, the progress on increasing the performance of executing if-statements of CPUs essentially has stagnated
Range Index Models are CDF Models

\[ p = F(\text{Key}) \times N \]
A First, Naïve Learned Index

- Data: 200M web-server log records
- Environment: Tensorflow
- Model: two-layer fully-connected neural network
  - 32 neurons per layer
  - activation functions: ReLU
- Input features: timestamps
- Labels: positions
A First, Naïve Learned Index

Result:
- BTree: 300ns
- Naïve Learned Index: 80000ns

Why?
- significant invocation overhead
- the accuracy for last-mile search
- B-Trees are extremely cache-efficient
The Learning Index Framework (LIF)

- Train: Tensorflow
- Extracts weights from the model and generates efficient index structures in C++
- Running: C++
- 80000ns → 30ns
The Recursive Model Index
The accuracy for last-mile search
Hybrid Indexes

Two models:

- neural nets
  - zero to two fully-connected hidden layers
  - ReLU activation functions
  - layer width of up to 32 neurons

- B-Trees

If min-/max-error is above a threshold → Rollback to B-Trees

The worst case performance of learned indexes is the performance of B-Trees.
Search Strategies

Search record in pages

- Model Biased Search: first middle point is set to the value predicted by the model.
- Biased Quaternary Search: three middle points $\text{pos} - \sigma$, $\text{pos}$, $\text{pos} + \sigma$. 
## Results

### Integer Datasets

<table>
<thead>
<tr>
<th>Type</th>
<th>Config</th>
<th>Map Data</th>
<th></th>
<th>Web Data</th>
<th></th>
<th>Log-Normal Data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Model (ns)</td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Model (ns)</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>
<td>51.93 (4.00x)</td>
<td>276 (0.94x)</td>
<td>201 (72.7%)</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>
<td>25.97 (2.00x)</td>
<td>274 (0.95x)</td>
<td>171 (62.4%)</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>
<td>12.98 (1.00x)</td>
<td>260 (1.00x)</td>
<td>132 (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>
<td>6.49 (0.50x)</td>
<td>266 (0.98x)</td>
<td>114 (42.9%)</td>
</tr>
<tr>
<td></td>
<td>page size: 512</td>
<td>3.28 (0.25x)</td>
<td>286 (0.99x)</td>
<td>101 (35.3%)</td>
<td>3.25 (0.25x)</td>
<td>291 (0.89x)</td>
<td>100 (34.3%)</td>
</tr>
<tr>
<td>Learned</td>
<td>2nd stage models: 10k</td>
<td>0.15 (0.01x)</td>
<td>98 (2.70x)</td>
<td>31 (31.6%)</td>
<td>0.15 (0.01x)</td>
<td>222 (1.17x)</td>
<td>29 (13.1%)</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>
<td>0.76 (0.06x)</td>
<td>162 (1.60x)</td>
<td>36 (22.2%)</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>
<td>1.53 (0.12x)</td>
<td>144 (1.81x)</td>
<td>39 (26.9%)</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>
<td>3.05 (0.24x)</td>
<td>126 (2.07x)</td>
<td>41 (32.5%)</td>
</tr>
</tbody>
</table>
# Results

## String Datasets

<table>
<thead>
<tr>
<th>Config</th>
<th>Size (MB)</th>
<th>Lookup (ns)</th>
<th>Model (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Btree</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>page size: 32</td>
<td>13.11 (4.00x)</td>
<td>1247 (1.03x)</td>
<td>643 (52%)</td>
</tr>
<tr>
<td>page size: 64</td>
<td>6.56 (2.00x)</td>
<td>1280 (1.01x)</td>
<td>500 (39%)</td>
</tr>
<tr>
<td>page size: 128</td>
<td>3.28 (1.00x)</td>
<td>1288 (1.00x)</td>
<td>377 (29%)</td>
</tr>
<tr>
<td>page size: 256</td>
<td>1.64 (0.50x)</td>
<td>1398 (0.92x)</td>
<td>330 (24%)</td>
</tr>
<tr>
<td><strong>Learned Index</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 hidden layer</td>
<td>1.22 (0.37x)</td>
<td>1605 (0.80x)</td>
<td>503 (31%)</td>
</tr>
<tr>
<td>2 hidden layers</td>
<td>2.26 (0.69x)</td>
<td>1660 (0.78x)</td>
<td>598 (36%)</td>
</tr>
<tr>
<td><strong>Hybrid Index</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t=128, 1 hidden layer</td>
<td>1.67 (0.51x)</td>
<td>1397 (0.92x)</td>
<td>472 (34%)</td>
</tr>
<tr>
<td>t=128, 2 hidden layers</td>
<td>2.33 (0.71x)</td>
<td>1620 (0.80x)</td>
<td>591 (36%)</td>
</tr>
<tr>
<td>t=64, 1 hidden layer</td>
<td>2.50 (0.76x)</td>
<td>1220 (1.06x)</td>
<td>440 (36%)</td>
</tr>
<tr>
<td>t=64, 2 hidden layers</td>
<td>2.79 (0.85x)</td>
<td>1447 (0.89x)</td>
<td>556 (38%)</td>
</tr>
<tr>
<td><strong>Learned QS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 hidden layer</td>
<td>1.22 (0.37x)</td>
<td>1155 (1.12x)</td>
<td>496 (43%)</td>
</tr>
</tbody>
</table>
Outline

Range Index

Point Index

Existence Index
Hash-Map & Learned Hash-Map

(a) Traditional Hash-Map

(b) Learned Hash-Map
## Results

<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>
Outline

Range Index

Point Index

Existence Index
Bloom filters

FPR = const & FNR = 0
Learned Bloom filters

\[ \text{FPR} = \text{const} \quad \text{and} \quad \text{FNR} = \text{const} \]
Bloom filters as a Classification Problem

FPR = const & FNR = 0
Results

- Data: blacklisted phishing URLs
- Model: a 16-dimensional Gated Recurrent Unit (GRU) with a 32-dimensional embedding for each character
Results

W: the RNN width
E: the embedding size for each character
Conclusion and Future Work

- Other ML Models
- Multi-Dimensional Indexes
- Beyond Indexing: Learned Algorithms (e.g., insertion sort)
- GPU/TPUs

“In summary, we have demonstrated that machine learned models have the potential to provide significant benefits over state-of-the-art indexes, and we believe this is a fruitful direction for future research.”
Thanks!