Learned Indexes

Sumer Rathinam
Traditional Indexes

- B-Trees: For range requests
- Hash-maps: Single key lookups
- Bloom Filters: Check for record existence
Problem

- Traditional indexes are general purpose data structures
- Assume nothing about the data distribution
- Doesn't take advantage of common prevalent patterns in real world data
Example

Goal: Index all integers from 1 to 100M

1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ... 100M

B-Tree?

B-Tree of Order 4
Key Insight

Knowing the exact data distribution allows for instance based optimization
Real World Data

- Real data doesn’t follow perfectly known pattern
- Engineering cost to build specialized solution is too high
Machine Learning

- ML can learn a model to reflect data patterns
- Creates specialized index structures
- Low engineering costs
- Cannot provide semantic guarantees
- Traditionally high compute costs
Disclaimers

• Learned indexes are not meant to completely replace existing indexes
• Complement existing work
• Most data structures can be broken down into a learned model and an auxiliary structure
• Continuous functions describing data distribution are used to build efficient data structures and algorithms
3 Key Learned Indexes

- Learned indexes using B-Trees
- Learned indexes using Hash-maps
- Learned indexes using Bloom filters
Range Index

- Only index every nth key where n is page size
- Min error of 0, max error of the page size
- ML model only needs to provide these error guarantees

Figure 1: Why B-Trees are models
ML models

• Have same guarantees as B-Trees
• B-Trees are rebalanced with new data
• ML models retrain to do the same
• Linear regression or neural net are common models that could replace B-Trees
New Challenges

• B-Trees have bounded insert and lookup costs
• Takes advantage of the cache
• Can map keys to pages that are not continuously mapped to memory or disk

* Assumption: we only index an in-memory dense array that is sorted by key
Model Complexity

• Needs to match the same number of operations it takes to traverse B-Tree
• Precision of model needs to be more efficient than a B-Tree

Assumption: B-Tree that indexes 100M records with a page size of 100

With this assumption a model needs to have a better precision gain than $1/100$ per 400 arithmetic operations
(50 cycles per b-tree page traversal * 8 CPU SIMD operations per cycle)

*This is with all B-Tree pages in cache
CDF Models

• Model that predicts the position of a key inside a sorted array approximates the cumulative distribution function

\[ p = F(\text{Key}) \times N \]

• \( p \) is the position estimate
• \( F(\text{Key}) \) is the estimated CDF for the data to estimate the likelihood to observe a key smaller or equal to the look-up key \( P(X \leq \text{Key}) \)
• \( N \) is the total number of keys
Key Takeaways

- B-Tree learns the distribution by creating a regression tree
- ML model can do the same by minimizing the squared error of a linear function
- CDF will play a key role in optimizing other types of index structures
Naïve Learned Index

- Used 200M web server log records
- Built secondary index over the timestamps
- Trained a two-layer fully connected neural network with 32 neurons per layer
- Timestamps are input features
- Positions in sorted array are the labels
- Took 80,000 nano-seconds to execute
- B-Tree took 300 nano-seconds
Recursive Model Index

- Takes key as input
- Predicts position with certain error
- Selects another model based on error of prediction
- Final stage gives position
Hybrid Indexes

• Recursive model allows for a mixture of models depending on the stage
• Top layers are more likely to use small Neural Nets so they can learn a wide range of data
• Bottom layers can use thousands of simple linear regression models as they are inexpensive in space and execution time
• Paper replaces NN models with B-Trees if absolute min-/max-error is above a predefined threshold

* Hybrid indexes bind the worst case performance of learned indexes to the performance of B-Trees.
### Results

- Learned index dominates B-Tree
- Most configurations 1.5 - 3 times faster
- Up to 2 orders of magnitude smaller in size

<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>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>
</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>255 (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*
Indexing Strings

- Tokenize input string into input vector
- Treated the same as real valued keys except with a vector instead of single value
- Linear models scale the number of multiplications and additions linearly with regards to input length
Results For Strings

- 10M non continuous document IDs of a large web index
- Learned QS is a non hybrid recursive model index using quaternary search
- Best performance for strings, while normal learned index did not perform as well

Figure 6: String data: Learned Index vs B-Tree
Point Index

• Hash-maps traditionally used
• Key is to prevent too many conflicts
• Example:
  o 100M records
  o Hash-map size of 100M
  o Uniformly random keys
  o Leads to 33% or 33M conflicts

*Machine learning models can reduce conflict
ML Models

• Using learned models as a hash-function already exists
• Existing solutions don’t take advantage of underlying data distribution
• Machine learning models can provide a more customized solution
Comparison

- \( H(K) = F(K) \times M \), \( M \) is the size of the hash-map
- Scales the CDF by targeted size of \( M \)
- If we perfectly learn the CDF of keys, no conflicts would occur
- Uses the same recursive model index as before

Figure 7: Traditional Hash-map vs Learned Hash-map
Hash Model

• Tradeoff between size of index and performance
• Benefits of learned model depend on
  o How accurately the model represents the CDF
  o Hash map architecture

Example:
• With small keys and little to no values, traditional hash functions will perform well
• With larger payloads learned models will perform better
Results

- Used the same 3 sets of data from b-tree evaluation
- 2 stage recursive model index used
- 100k models on the second stage
Existence Index

- Traditional bloom filters are space efficient, but still can occupy a lot of memory
- False negative rate of 0
- Specific false positive rate
- Learned model can achieve these requirements
Existence Index Model

- Learn a model $f$ that predicts whether query $x$ is a key or non-key.
- Use Recurrent NN or Convolutional NN to do this.
- Will need an overflow bloom filter to keep false negative rate at 0.
- Still has a certain false positive rate.
Results

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.)
Future Work

• Using other ML models i.e. not just linear models and NN
• Multidimensional indexes i.e. position of all records filtered by any combination of attributes
• Beyond indexing: learned algorithms
  o Learning the CDF model could speed up sorting and joins, not just indexes
• GPU/TPU improvements and speedups
Overall Thoughts

• Does a great job of putting complex concepts into simple terms
• The mapping between traditional indexes and learned models is great
• Experiments were well thought out and covered worst cases
• Could've talked more on how these new findings will impact the industry
• How can we get learned indexes into some sort of commercial system