START – Self-Tuning Adaptive Radix Tree

Philipp Fent, Michael Jungmair, Andreas Kipf, Thomas Neumann

Technische Universität München

April 20, 2020
Learned indexes

• Small and efficient
• Adapt to data distribution
• Fast for read-only workloads
Practical Database Indexes

Learned indexes
- Small and efficient
- Adapt to data distribution
- Fast for read-only workloads

ART – Adaptive Radix Tree
- Used in practice
- Well understood
- Fast for various workloads
- But: limited adaption to data distribution
- and slower than a well-trained machine learning model
Learned indexes

- Small and efficient
- Adapt to data distribution
- Fast for read-only workloads

**START Self Tuning ART**

- Bridges the gap
- Adapts to data: 85% faster!
- Keep robustness

**ART – Adaptive Radix Tree**

- Used in practice
- Well understood
- Fast for various workloads
- But: limited adaptability to data distribution
- and slower than a well-trained machine learning model
Self-Tuning ART

Byte 1

Byte 2

Byte 3

Byte 4

ART
Self-Tuning ART

Byte 1

Byte 2

Byte 3

Byte 4

ART

START
Self-Tuning ART

Byte 1

Byte 2

Byte 3

Byte 4

ART

START

introduce multilevel nodes

cost model

optimizer
Rewired memory nodes

- Dense regions with many keys
- Subtrees of full nodes
- Keep data continuous
Multilevel Nodes

Rewired memory nodes
- Dense regions with many keys
- Subtrees of full nodes
- Keep data continuous

Sparse regions
- Already use path compression
- Still: improve common prefixes
- Reduce chains of small nodes
Rewired Memory Nodes

Node256
256 children

---

Philipp Fent et al. START – Self-Tuning Adaptive Radix Tree 5 / 12
Rewired Memory Nodes

Node256
256 children

Node16M
64 K Node256
Rewired Memory Nodes

A diagram showing the rewiring process of memory nodes. The diagram includes:
- Virtual pages with the labels 'a', 'b', and '∅'.
- Memory file with corresponding entries for 'a', 'b', and '∅'.
- Physical pages with 'a' and 'b'.

The diagram illustrates how certain nodes are rewired to optimize memory access.
Rewired Memory Nodes

page 1: virtual pages
page 2: virtual pages
page 3: virtual pages
page 4: physical pages
Rewired Memory Nodes

![Diagram showing rewired memory nodes with virtual and physical pages.](image)
Rewired Memory Nodes

virtual pages

physical pages

page 1

page 2

page 3

page 4

∅
Rewired Memory Nodes

- Virtual pages
- Physical pages

Page 1

Page 2

Page 3

Page 4

access(11)
Multilevel Node4

- Helps reduce sparse affix nodes
- Use all 64 Bytes of a cacheline
Cost Model for Node Lookup

- Minimize the average child lookup cost
- Consider individual node cost

<table>
<thead>
<tr>
<th>[ns/lookup]</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node4</td>
<td>1</td>
</tr>
<tr>
<td>Node16</td>
<td>1</td>
</tr>
<tr>
<td>Node48</td>
<td>1</td>
</tr>
<tr>
<td>Node256</td>
<td>1</td>
</tr>
<tr>
<td>Rewired64K</td>
<td>2</td>
</tr>
<tr>
<td>Rewired16M</td>
<td>3</td>
</tr>
<tr>
<td>MultiNode4</td>
<td>2-4</td>
</tr>
</tbody>
</table>
Cost Model for Node Lookup

- Minimize the average child lookup cost
- Consider individual node cost

<table>
<thead>
<tr>
<th>[ns/lookup]</th>
<th>Levels</th>
<th>Cached</th>
<th>Header</th>
<th>Cached</th>
<th>Uncached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node4</td>
<td>1</td>
<td>7</td>
<td></td>
<td>7</td>
<td>68</td>
</tr>
<tr>
<td>Node16</td>
<td>1</td>
<td>5</td>
<td></td>
<td>77</td>
<td>162</td>
</tr>
<tr>
<td>Node48</td>
<td>1</td>
<td>2</td>
<td></td>
<td>165</td>
<td>168</td>
</tr>
<tr>
<td>Node256</td>
<td>1</td>
<td>2</td>
<td></td>
<td>88</td>
<td>92</td>
</tr>
<tr>
<td>Rewired64K</td>
<td>2</td>
<td>6</td>
<td></td>
<td>87</td>
<td>162</td>
</tr>
<tr>
<td>Rewired16M</td>
<td>3</td>
<td>6</td>
<td></td>
<td>88</td>
<td>165</td>
</tr>
<tr>
<td>MultiNode4</td>
<td>2-4</td>
<td>6</td>
<td></td>
<td>6</td>
<td>68</td>
</tr>
</tbody>
</table>
Bottom-up Optimization

```
20 [60, \ldots ] \rightarrow ... \\
5 [20, \ldots ] \rightarrow ...
```

```
10 [30, \ldots ] \rightarrow ...
5 [25, \ldots ] \rightarrow ...
```

```
40 \cdot 3 + 135
40 \cdot 2 + 215
130 + 80 + 55
15 + 85 + 55
```

1-level:

2-level:
Bottom-up Optimization
Bottom-up Optimization

![Diagram of a bottom-up optimization process with nodes labeled 25, 20, 5, 10, 5 and ranges [130, 80, ...], [60, ...], [20, ...], [30, ...], [25, ...], [85, 55, ...]].
Bottom-up Optimization

25 + 15
40

40 · 2 + 215
2-level: [295, 215, 135, ...]

1-level: [130 + 85, 80 + 55]

25
[130, 80, ...]

2
20
[60, ...]

15
[85, 55, ...]

10
[30, ...]

5
[25, ...]

5
[20, ...]

...
Performance

https://learned.systems/sosd

Latency [ns / op]

Datasets

Indexes

- ART
- START
- RS
- RMI

Philipp Fent et al. START – Self-Tuning Adaptive Radix Tree
**Conclusion**

- START – Best of both worlds?
- Adapt to real-world data distribution
- Still with robust underpinning of ART
- Inserts still possible and efficient
- But: might degrade multilevel nodes

github.com/jungmair/START

fent@in.tum.de