Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes

Huanchen Zhang

Co-authors:
David G. Andersen
Andrew Pavlo
Michael Kaminsky
Lin Ma
Rui Shen
You are running out of memory
You are running out of memory
You are running out of memory

Buy more

Yes
No
TPC-C on H-Store

Transactions Executed

Memory Limit = 5GB

Throughput

Memory (GB)

Disk tuples
In-memory tuples
Indexes
I GOT STUCK
SO I WENT TO SLEEP
The better way:

Use memory more efficiently
Indexes are **LARGE**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>% space for index</th>
<th>Hybrid Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-C</td>
<td>58%</td>
<td>34%</td>
</tr>
<tr>
<td>Voter</td>
<td>55%</td>
<td>41%</td>
</tr>
<tr>
<td>Articles</td>
<td>34%</td>
<td>18%</td>
</tr>
</tbody>
</table>
Our Contributions

1. The hybrid index architecture
2. The Dual-Stage Transformation
3. Applied to 4 index structures
   - B+tree
   - Masstree
   - Skip List
   - Adaptive Radix Tree (ART)

Performance ≈ 30 – 70%
Did we solve this problem?

TPC-C on H-Store

Transactions Executed

Stay tuned
How do hybrid indexes achieve memory savings?

Static
Hybrid Index: a dual-stage architecture

dynamic stage

static stage
Inserts are batched in the dynamic stage

dynamic stage  static stage

write  merge
Reads search the stages in order
A Bloom filter improves read performance

① read

dynamic stage

② read

static stage
Dynamic stage

Memory-efficient

Skew-aware

Static stage

1. Read
2. Read

Write
Merge
The Dual-Stage Transformation

1. dynamic stage

2. static stage

3. merge
The Dynamic-to-Static Rules

- Compaction
- Reduction
- Compression
Compaction: minimize # of memory blocks
Compaction: minimize # of memory blocks
Reduction: minimize structural overhead
Reduction: minimize structural overhead

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram with nodes labeled 1 to 12, and connections indicating relationships or dependencies.
The Dual-Stage Transformation

1. dynamic stage
2. merge
3. static stage
The Dual-Stage Transformation

Merge Questions:
1. Partial? 🍀🍀?
2. When? 🕒?
3. Blocking? 🗝?

dynamic stage

static stage
Did we solve this problem?

Transactions Executed
Yes, we improved the DBMS’s capacity!

TPC-C on H-Store

Throughput (txn/s)

Transactions Executed

B+tree

Hybrid
TPC-C on H-Store

Transactions Executed

Throughput (txn/s)

Memory (GB)

TPC-C on H-Store

B+tree

Hybrid

B+tree

Hybrid

Disk tuples

In-memory tuples

Indexes

Transactions Executed

21
Transactions Executed Throughput (txn/s)

Memory (GB)

TPC-C on H-Store

B+tree

Hybrid

Disk tuples

In-memory tuples

Indexes

Transactions Executed
Transactions Executed

Throughput (txn/s)

<table>
<thead>
<tr>
<th>Memory (GB)</th>
<th>B+tree</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B+tree

Hybrid

Transactions Executed

TPC-C on H-Store

Disk tuples

In-memory tuples

Indexes
Take Away:

Memory saved by indexes → Larger working set in memory → Higher throughput

<table>
<thead>
<tr>
<th>Memory (GB)</th>
<th>Transactions Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2M</td>
</tr>
<tr>
<td>2M</td>
<td>4M</td>
</tr>
<tr>
<td>4M</td>
<td>6M</td>
</tr>
<tr>
<td>6M</td>
<td>8M</td>
</tr>
<tr>
<td>8M</td>
<td>10M</td>
</tr>
</tbody>
</table>

B+tree

Hybrid

Indexes

Disk tuples

In-memory tuples

Take Away: Larger working set in memory leads to Higher throughput.
This is just the **BEGINNING**
Conclusions

1. The hybrid index architecture
2. The Dual-Stage Transformation
3. Applied to 4 index structures
   - B+tree
   - Masstree
   - Skip List
   - Adaptive Radix Tree (ART)
Toll-Free Hotline:

1-844-888-CMUDB