Adaptive Hybrid Indexes

Christoph Anneser\textsuperscript{1}, Andreas Kipf\textsuperscript{2}, Huanchen Zhang\textsuperscript{3}, Thomas Neumann\textsuperscript{1}, Alfons Kemper\textsuperscript{1}

SIGMOD, June 12 – 17, 2022

\textsuperscript{1}Technical University of Munich, Germany
\textsuperscript{2}Massachusetts Institute of Technology, USA
\textsuperscript{3}Tsinghua University, China
Index structures are essential for fast query processing
Problem

Index structures are essential for fast query processing

Real-world workloads have skew
Problem

Index structures are essential for fast query processing

• Typically optimized for all operations at development time

Real-world workloads have skew
Problem

Index structures are essential for fast query processing
- Typically optimized for all operations at development time

Real-world workloads have skew
- Information is available at run-time & depends on workload
Problem

Index structures are essential for fast query processing
  • Typically optimized for all operations at development time

Real-world workloads have skew
  • Information is available at run-time & depends on workload
Problem

Index structures are essential for fast query processing
• Typically optimized for all operations at development time

Real-world workloads have skew
• Information is available at run-time & depends on workload
Solution

Index Structure

Adaptive Hybrid Index

Adaptive Hybrid Indexes
Solution

Adaptive Hybrid Index

Index Structure

1. Lightweight Workload Tracking
Solution

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification

Index Structure
Solution

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations

Index Structure
Solution

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations

Index Structure

anne-ser@in.tum.de

Adaptive Hybrid Indexes
Solution

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations

Index Structure

- Perf.-Optimized
- Compressed
Sampling Parameters

Frequency

- Low frequencies reduce sampling overhead
- High frequencies allow to promptly react to changing workload
Sampling Parameters

Frequency

- Low frequencies reduce sampling overhead
- High frequencies allow to promptly react to changing workload

Size

- Small samples introduce inaccuracies
- Large samples require a longer time to be collected
Sampling Parameters

Frequency

• Low frequencies reduce sampling overhead
• High frequencies allow to promptly react to changing workload

Size

• Small samples introduce inaccuracies
• Large samples require a longer time to be collected

⇒ Adaptive Hybrid Indexes choose these parameters adaptively at runtime
Application I: Adaptive Hybrid B+-Tree

Figure: Example B+-Tree

Table: Leaf encodings storing 64-bit key-value pairs and performance implications on lookups.

<table>
<thead>
<tr>
<th>Leaf Node Encoding</th>
<th>Average Size</th>
<th>Instructions LLC Misses</th>
<th>Branch Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gapped</td>
<td>4096B</td>
<td>85</td>
<td>2.1</td>
</tr>
<tr>
<td>Packed</td>
<td>2872B</td>
<td>84</td>
<td>1.4</td>
</tr>
<tr>
<td>Succinct</td>
<td>1076B</td>
<td>341</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Application I: **Adaptive Hybrid B+-Tree**

![Example B+-Tree](image)

**Figure:** Example B+-Tree

<table>
<thead>
<tr>
<th>Encoding Type</th>
<th>Average Size</th>
<th>Instructions LLC Misses</th>
<th>Branch Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gapped</td>
<td>4096B</td>
<td>85</td>
<td>2.1</td>
</tr>
<tr>
<td>Packed</td>
<td>2872B</td>
<td>84</td>
<td>1.4</td>
</tr>
<tr>
<td>Succinct</td>
<td>1076B</td>
<td>341</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table: Leaf encodings storing 64-bit key-value pairs and performance implications on lookups.
Application I: Adaptive Hybrid B+-Tree

Figure: Example B+-Tree

<table>
<thead>
<tr>
<th>Leaf Node Encoding</th>
<th>Average Size</th>
<th>Instruction LLC Misses</th>
<th>Branch Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gapped</td>
<td>4096B</td>
<td>85</td>
<td>2.1</td>
</tr>
<tr>
<td>Packed</td>
<td>2872B</td>
<td>84</td>
<td>1.4</td>
</tr>
<tr>
<td>Succinct</td>
<td>1076B</td>
<td>341</td>
<td>1.1</td>
</tr>
</tbody>
</table>


Gapped: \[ \cdots k_0 \ k_1 \ k_2 \ \bot \ v_0 \ v_1 \ v_2 \ \bot \]

Packed: \[ \cdots k_0 \ k_1 \ k_2 \ v_0 \ v_1 \ v_2 \]

anneser@in.tum.de

Adaptive Hybrid Indexes
Application I: **Adaptive Hybrid B+-Tree**

**Figure:** Example B+-Tree

<table>
<thead>
<tr>
<th>Gapped:</th>
<th>\ldots</th>
<th>( k_0 )</th>
<th>( k_1 )</th>
<th>( k_2 )</th>
<th>( \perp )</th>
<th>( v_0 )</th>
<th>( v_1 )</th>
<th>( v_2 )</th>
<th>( \perp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packed:</td>
<td>\ldots</td>
<td>( k_0 )</td>
<td>( k_1 )</td>
<td>( k_2 )</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Succinct:</td>
<td>\ldots</td>
<td>( k_{\text{min}} )</td>
<td>( v_{\text{min}} )</td>
<td>( \Delta k_1 )</td>
<td>( \Delta k_2 )</td>
<td>( \Delta v_1 )</td>
<td>( \Delta v_2 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application I: Adaptive Hybrid B+-Tree

Figure: Example B+-Tree

Node encoding is chosen adaptively at run-time
Application I: Adaptive Hybrid B+-Tree

Figure: Example B+-Tree

Table: Leaf encodings storing 64-bit key-value pairs and performance implications on lookups.

<table>
<thead>
<tr>
<th>Leaf Node Encoding</th>
<th>Average Size</th>
<th>Instruc. Misses</th>
<th>LLC Misses</th>
<th>Branch Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gapped</td>
<td>4096B</td>
<td>85</td>
<td>2.1</td>
<td>4.44</td>
</tr>
<tr>
<td>Packed</td>
<td>2872B</td>
<td>84</td>
<td>1.4</td>
<td>4.46</td>
</tr>
<tr>
<td>Succinct</td>
<td>1076B</td>
<td>341</td>
<td>1.1</td>
<td>6.69</td>
</tr>
</tbody>
</table>

Node encoding is chosen **adaptively at run-time**
Application II: Adaptive Hybrid Trie

Level-wise combination of the Adaptive Radix Tree (ART) and the Fast Succinct Trie (FST)
Application II: **Adaptive Hybrid Trie**

Level-wise combination of the **Adaptive Radix Tree (ART)** and the **Fast Succinct Trie (FST)**

- ART the default index structure in HyPer

---

Experiment Setup:
- **Dataset**: 33M unique email addresses (host-reversed order, e.g. com.foo@<username>)
- **Workload**: 50% Reads, 50% Scans, key selection follows a Zipf distribution

---

Figure: Query latency and index size of ART and FST
Application II: Adaptive Hybrid Trie

Level-wise combination of the Adaptive Radix Tree (ART) and the Fast Succinct Trie (FST)

- ART the default index structure in HyPer
- FST avoids pointers and instead calculates child node positions during traversal
Application II: Adaptive Hybrid Trie

Level-wise combination of the Adaptive Radix Tree (ART) and the Fast Succinct Trie (FST)

- ART the default index structure in HyPer
- FST avoids pointers and instead calculates child node positions during traversal

**Figure**: Query latency and index size of ART and FST

**Experiment Setup:**
- **Dataset**: 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- **Workload**: 50% Reads, 50% Scans, key selection follows a Zipf distribution
Application II: Adaptive Hybrid Trie

1. Expand hot nodes
2. Compact cold nodes

ART pointer
FST node number

Implicit Queries
Adaptation Manager
track optimize

anneser@in.tum.de Adaptive Hybrid Indexes
Application II: Adaptive Hybrid Trie

- ART pointer
- ART
- FST

1. Expand hot nodes
2. Compact cold nodes

queries
adaptation
manager
track
optimize

anneser@in.tum.de
Adaptive Hybrid Indexes
Application II: Adaptive Hybrid Trie

Implicit

ART pointer

ART

FST

1. Expand hot nodes
2. Compact cold nodes
Application II: Adaptive Hybrid Trie

ART
- ART pointer
- FST node number
- Implicit

FST

Queries
Adaptation
Manager

1. Expand hot nodes
2. Compact cold nodes
Application II: Adaptive Hybrid Trie

Queries

ART

FST

1. Expand hot nodes
2. Compact cold nodes
Application II: **Adaptive Hybrid Trie**

Queries → **ART**

**FST** → Adaptation Manager

- 1. Expand hot nodes
- 2. Compact cold nodes

optimize

track
Application II: Adaptive Hybrid Trie

Queries

- Expand hot nodes

ART

FST

Adaptation Manager

track

optimize

Expand hot nodes
Application II: Adaptive Hybrid Trie

Queries

ART

FST

1. Expand **hot nodes**
2. Compact **cold nodes**

track

optimize

Adaptation Manager
Evaluation

Setup

- 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz
- 64GB DDR4-2667 RAM
- GCC 9.3.0 with flags `-O3` and `march=native`
- CPU overhead for sampling, compacting, and expanding nodes is *included* in the plots
Evaluation: **Hybrid Trie – Space & Performance**

**Conclusions:**
For point lookups, Hybrid Trie ⇒ reduces index size by 63% comp. to ART ⇒ improves performance by 2.7x comp. to FST.

The Pre-Trained Hybrid Trie does not include tracking-related overhead.

**Experiment Setup:**
- **Dataset:** 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- **Workload:** 50% Reads, 50% Scans, key selection follows a Zipf distribution
Evaluation: Hybrid Trie – Space & Performance

Conclusions:

For point lookups, Hybrid Trie
- reduces index size by 63% comp. to ART
- improves performance by 2.7x comp. to FST

Experiment Setup:
- **Dataset**: 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)
- **Workload**: 50% Reads, 50% Scans, key selection follows a Zipf distribution
Evaluation: Hybrid Trie – Space & Performance

Conclusions:
For point lookups, Hybrid Trie
⇒ reduces index size by 63% comp. to ART
⇒ improves performance by 2.7x comp. to FST
The Pre-Trained Hybrid Trie does not include tracking-related overhead

Experiment Setup:
- Dataset: 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution
Evaluation: Hybrid Trie – Workload Adaptation

![Graph showing latency, size, and migrations over time for different phases.](image)

**Experiment Setup:**
- **Dataset:** 172M user ids (each 8B)
- **Workload:** Prefix Random
- Prefix Ranges randomly assigned to two phases

---

**Conclusions:**
- Adaptive Encoding Optimizations improve latency
- Limited size overhead
- Sampling frequency changes adaptively with # migrations
Evaluation: Hybrid Trie – Workload Adaptation

Phase 1
- Adaptations
- Expansions
- Compactions

Phase 2
- Adaptations
- Expansions

Latency [ns]
- AHI-Trie
- ART
- FST
- Pre-Trained

Size [MiB]

# Migrations

Time [intervals of 1M queries]

Conclusions:
- Adaptive Encoding Optimizations improve latency
- Limited size overhead
- Sampling frequency changes adaptively with # migrations

Experiment Setup:
- Dataset: 172M user ids (each 8B)
- Workload: Prefix Random
- Prefix Ranges randomly assigned to two phases
Evaluation: Hybrid Trie – Workload Adaptation

Phase 1
- Adaptations
- Expansions
- Compactions

Phase 2
- Sampling Phase
- Expansions

Conclusions:
⇒ Adaptive Encoding Optimizations improve latency
⇒ Limited size overhead
⇒ Sampling frequency changes adaptively with # migrations

Experiment Setup:
- Dataset: 172M user ids (each 8B)
- Workload: Prefix Random
- Prefix Ranges randomly assigned to two phases
Evaluation: Hybrid Trie – Workload Adaptation

Phase 1

Phase 2

Conclusions:
⇒ Adaptive Encoding Optimizations improve latency
⇒ Limited size overhead

Experiment Setup:
• Dataset: 172M user ids (each 8B)
• Workload: Prefix Random
• Prefix Ranges randomly assigned to two phases
Evaluation: Hybrid Trie – Workload Adaptation

Phase 1
- Adaptaions
- Expansions
- Compactions

Phase 2
- Adaptaions
- Expansions

Latency [ns]
- AHI-Trie
- ART
- FST
- Pre-Trained

Size [MiB]
- AHI-Trie
- ART
- FST
- Pre-Trained

# Migrations
- AHI-Trie
- ART
- FST
- Pre-Trained

Time [intervals of 1M queries]

Conclusions:
⇒ Adaptive Encoding Optimizations improve latency
⇒ Limited size overhead
⇒ Sampling frequency changes adaptively with # migrations

Experiment Setup:
- Dataset: 172M user ids (each 8B)
- Workload: Prefix Random
- Prefix Ranges randomly assigned to two phases
Evaluation: **Hybrid B+-Tree – Skewed Workloads**

**Zipfian Reads & Writes**

![Graph showing latency and size vs skew](image)

- **Latency [ns]**
- **Size [GB]**

**Experiment Setup:**
- **Dataset:** 400M Open Street Map Cell IDs
- **Workload:** 49% Reads, 49% Scans, 2% Inserts

---

**Conclusions:**
- Adaptive Hybrid Indexes perform best under skewed workloads.
- Tracking overhead & performance improvements through adaptive optimizations equalize at the break-even point.
Evaluation: Hybrid B+-Tree – Skewed Workloads

Zipfian Reads & Writes

Conclusions:
- Adaptive Hybrid Indexes perform best under skewed workloads

Experiment Setup:
- Dataset: 400M Open Street Map Cell IDs
- Workload: 49% Reads, 49% Scans, 2% Inserts
Evaluation: Hybrid B+-Tree – Skewed Workloads

Conclusions:
• Adaptive Hybrid Indexes perform best under skewed workloads
• Tracking overhead & performance improvements through adaptive optimizations equalize at the break-even point

Experiment Setup:
• Dataset: 400M Open Street Map Cell IDs
• Workload: 49% Reads, 49% Scans, 2% Inserts
Conclusions

Generic framework to create Adaptive Hybrid Indexes
Conclusions

Generic framework to create Adaptive Hybrid Indexes

Reduce storage overheads while retaining high performance
Conclusions

**Generic framework** to create Adaptive Hybrid Indexes

- **Lightweight Workload Tracking**
- **Classification**
- **Adaptive Optimizations**

Reduce storage **overheads** while retaining high performance

Evaluated the framework using **B+-trees and prefix trees**