Adaptive Hybrid Indexes

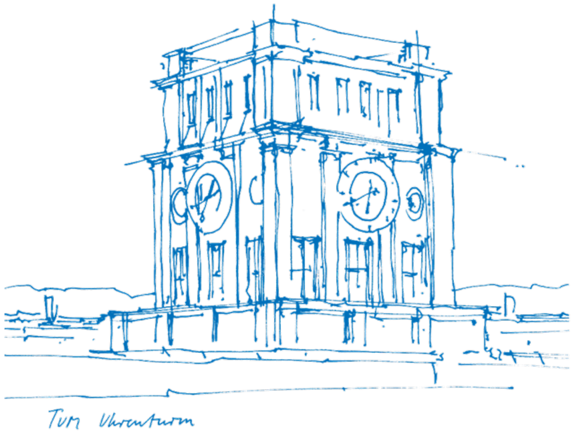
Christoph Anneser[1], Andreas Kipf[2], Huanchen Zhang[3], Thomas Neumann[1], Alfons Kemper[1]

SIGMOD, June 12 – 17, 2022

[1]Technical University of Munich, Germany

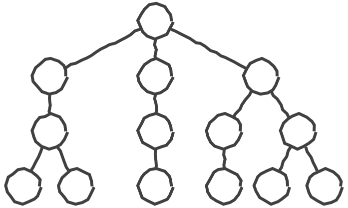
[2]Massachusetts Institute of Technology, USA

[3]Tsinghua University, China



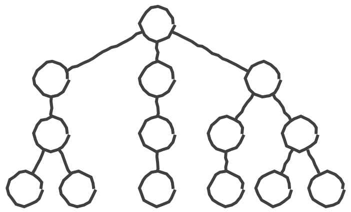
Problem

Index structures are essential for fast query processing

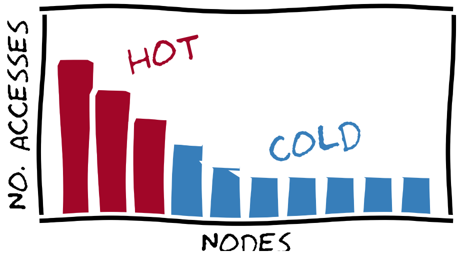


Problem

Index structures are essential for fast query processing



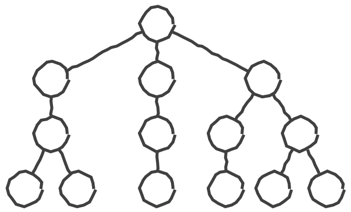
Real-world workloads have skew



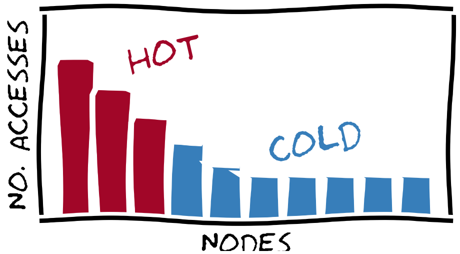
Problem

Index structures are essential for fast query processing

- Typically optimized for all operations at development time



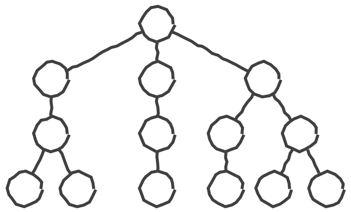
Real-world workloads have skew



Problem

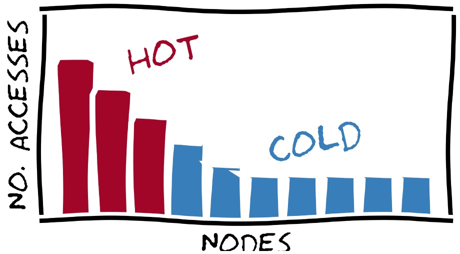
Index structures are essential for fast query processing

- Typically optimized for all operations at development time



Real-world workloads have skew

- Information is available at run-time & depends on workload



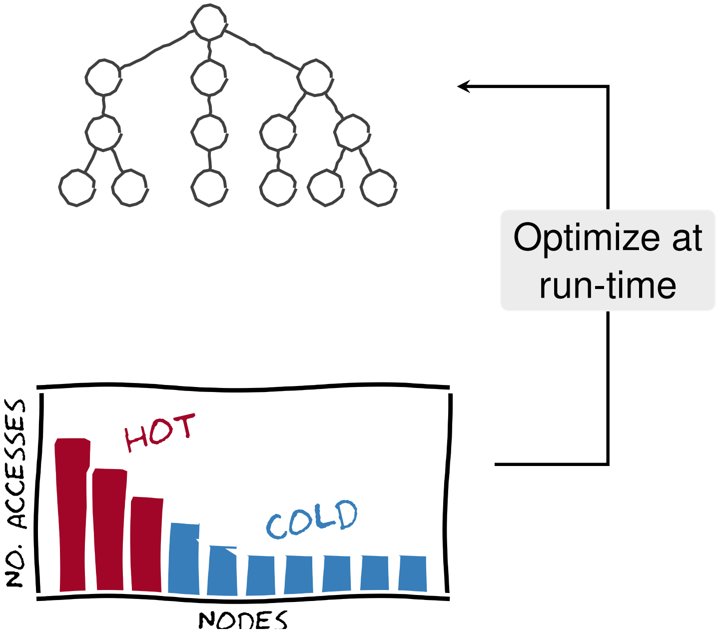
Problem

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



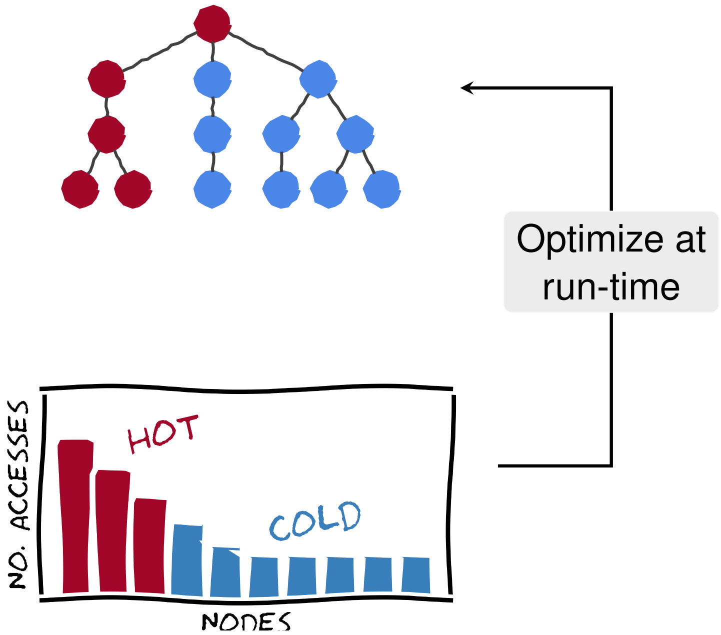
Problem

Index structures are essential for fast query processing

- Typically optimized for all operations at development time

Real-world workloads have skew

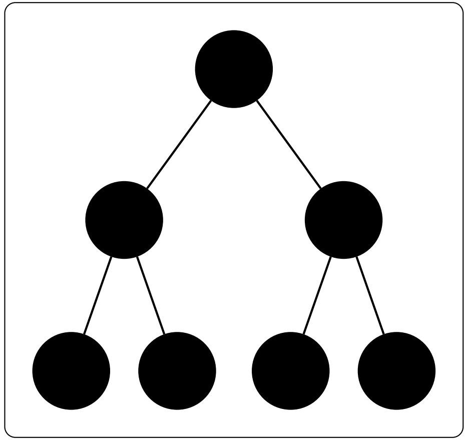
- Information is available at run-time & depends on workload



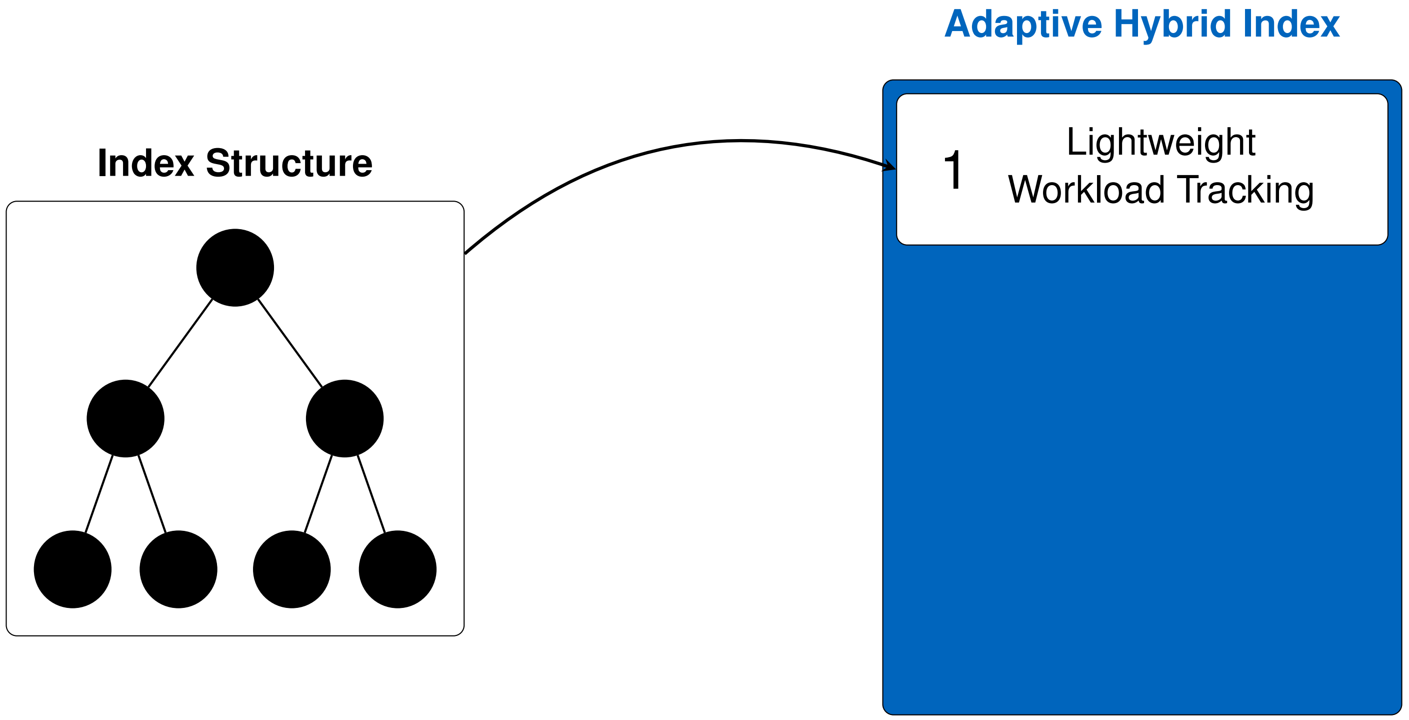
Solution

Adaptive Hybrid Index

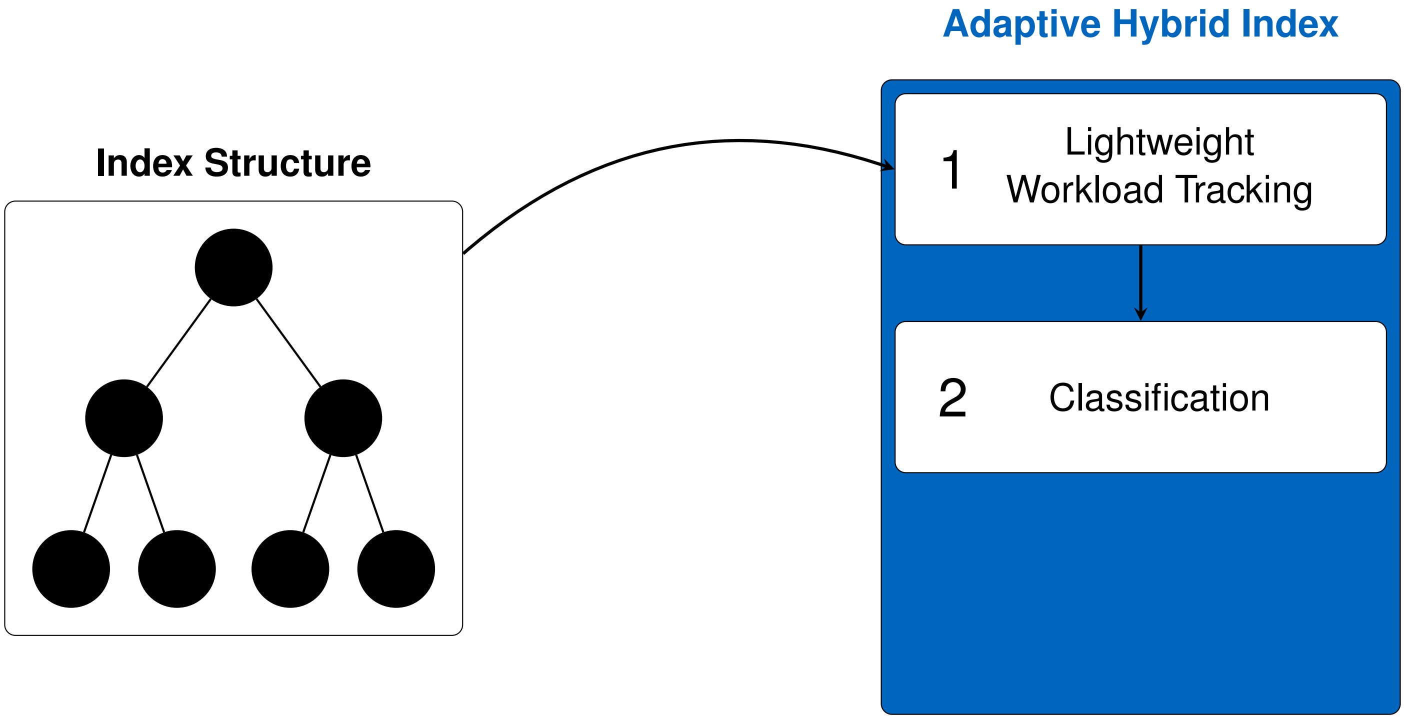
Index Structure



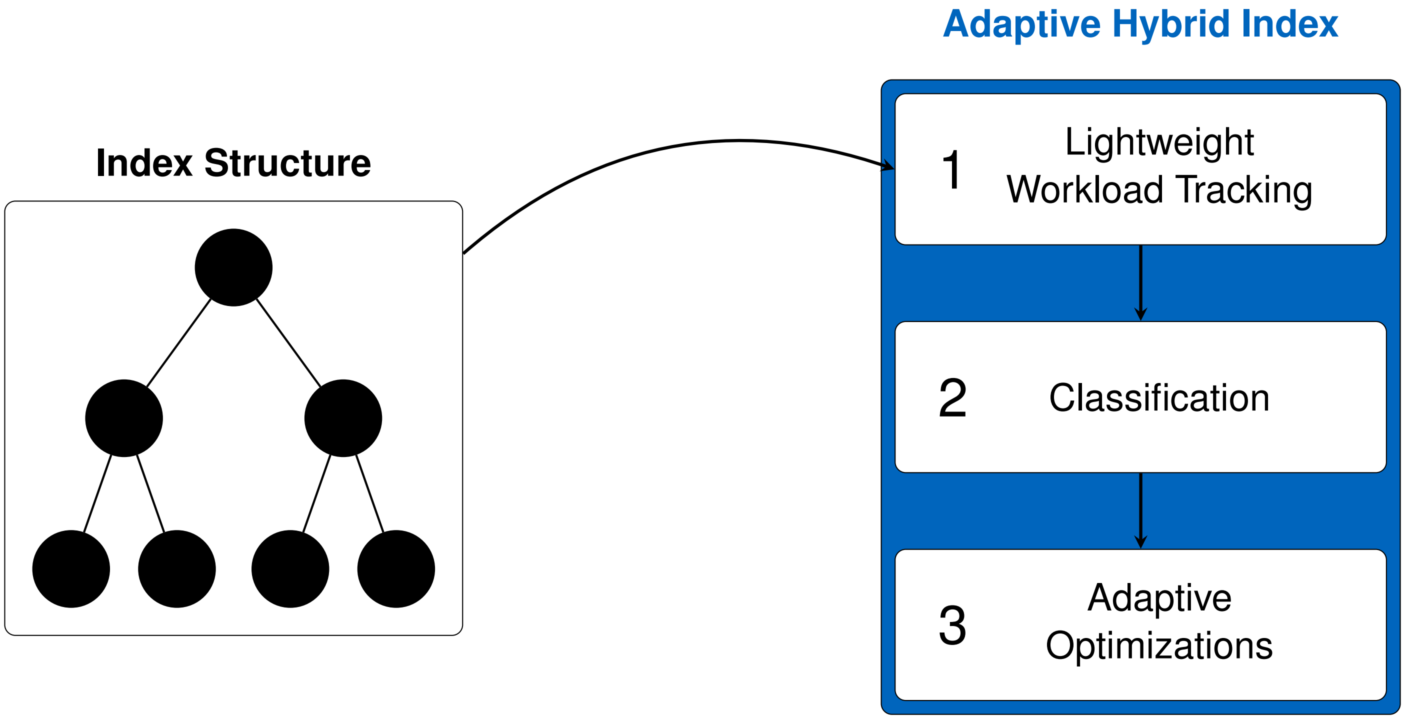
Solution


Adaptive Hybrid Index
Index Structure
1
Lightweight Workload Tracking

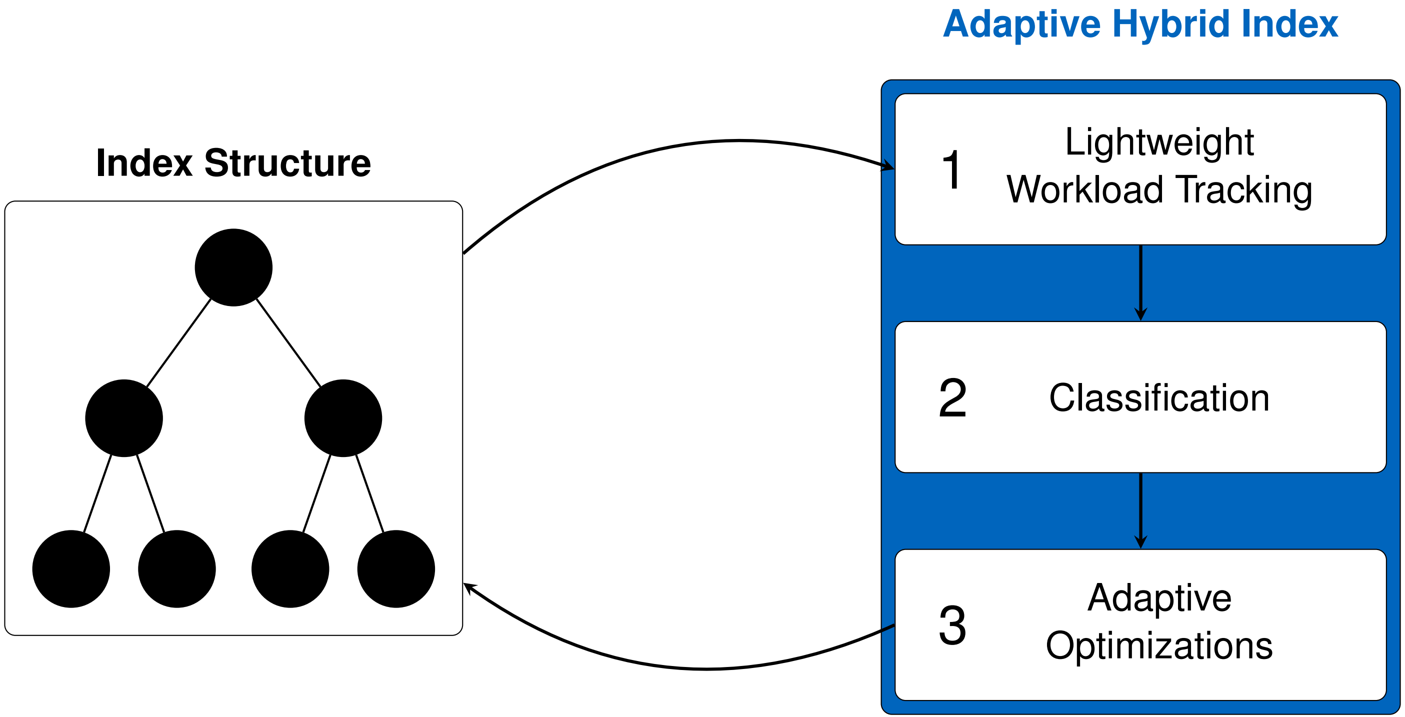
Solution


Adaptive Hybrid Index
Index Structure
1 Lightweight Workload Tracking
2 Classification

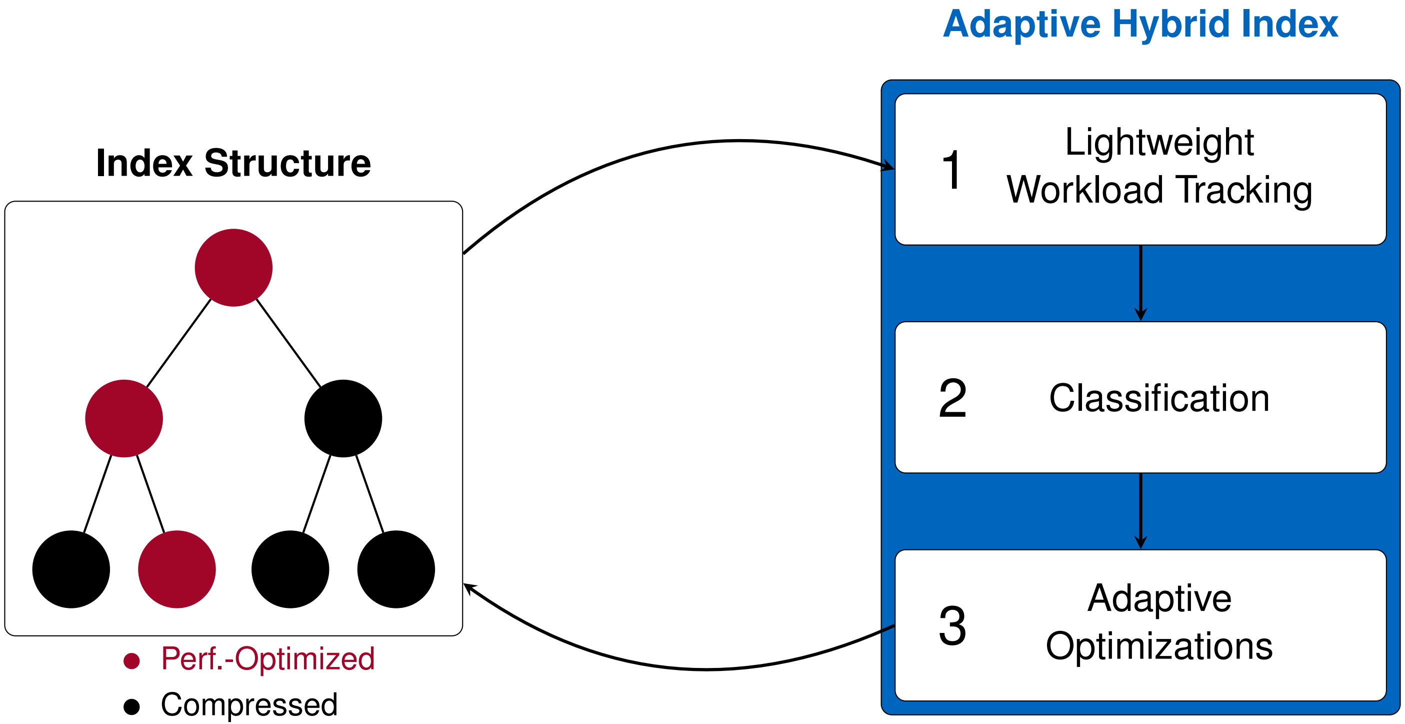
Solution


Adaptive Hybrid Index
Index Structure
1 Lightweight Workload Tracking
2 Classification
3 Adaptive Optimizations

Solution


Adaptive Hybrid Index
Index Structure
1 Lightweight Workload Tracking
2 Classification
3 Adaptive Optimizations

Solution


Adaptive Hybrid Index
Index Structure
1 Lightweight Workload Tracking
2 Classification
3 Adaptive Optimizations
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

$\Rightarrow$ **Adaptive Hybrid Indexes choose these parameters adaptively at runtime**

Application I: Adaptive Hybrid B+-Tree

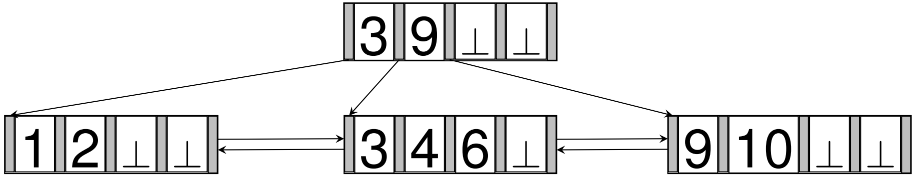


Figure: Example B+-Tree

Application I: Adaptive Hybrid B+-Tree

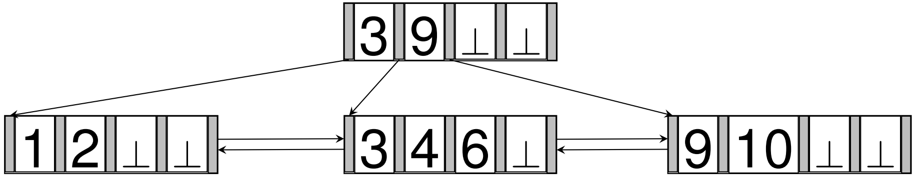
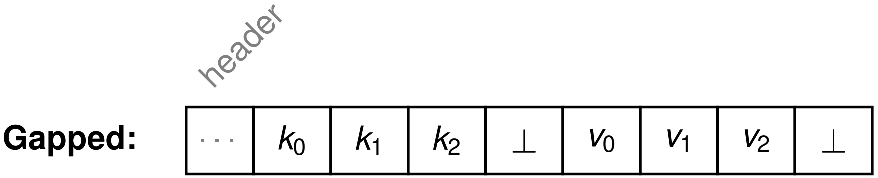


Figure: Example B+-Tree



Application I: Adaptive Hybrid B+-Tree

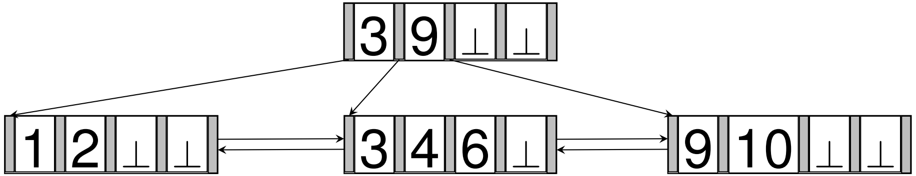
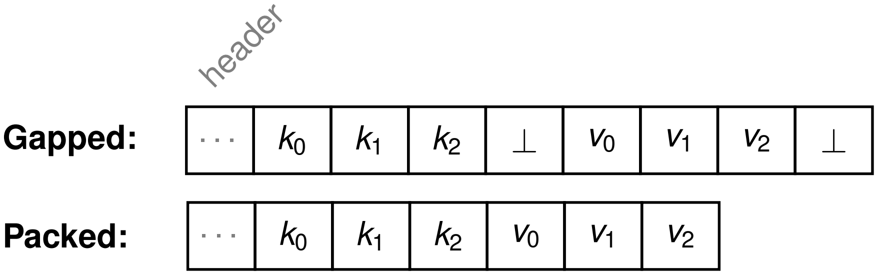


Figure: Example B+-Tree



Application I: Adaptive Hybrid B+-Tree

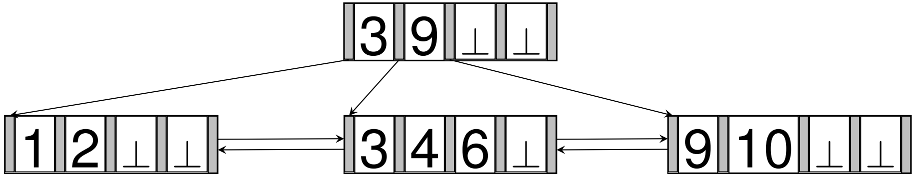


Figure: Example B+-Tree

header

Gapped:

···	k_0	k_1	k_2	$\perp$	v_0	v_1	v_2	$\perp$

Packed:

···	k_0	k_1	k_2	v_0	v_1	v_2

Succinct:

···	k_{min}	v_{min}	Δk_1	Δk_2	Δv_1	Δv_2

Application I: Adaptive Hybrid B+-Tree

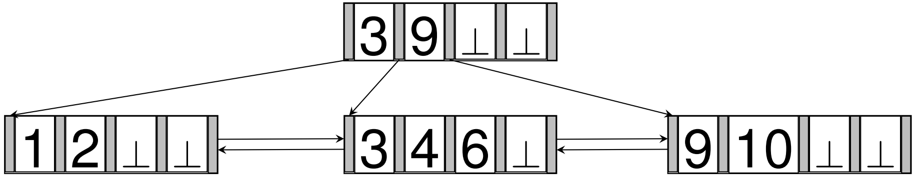


Figure: Example B+-Tree

Gapped:

header								
$\cdots$	k_0	k_1	k_2	$\perp$	v_0	v_1	v_2	$\perp$

Packed:

$\cdots$	k_0	k_1	k_2	v_0	v_1	v_2

Succinct:

$\cdots$	k_{min}	v_{min}	Δk_1	Δk_2	Δv_1	Δv_2

Node encoding is chosen adaptively at run-time

Application I: Adaptive Hybrid B+-Tree

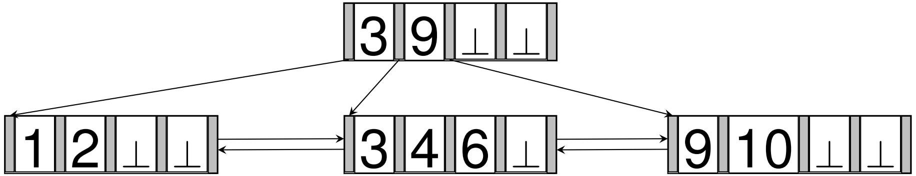


Figure: Example B+-Tree

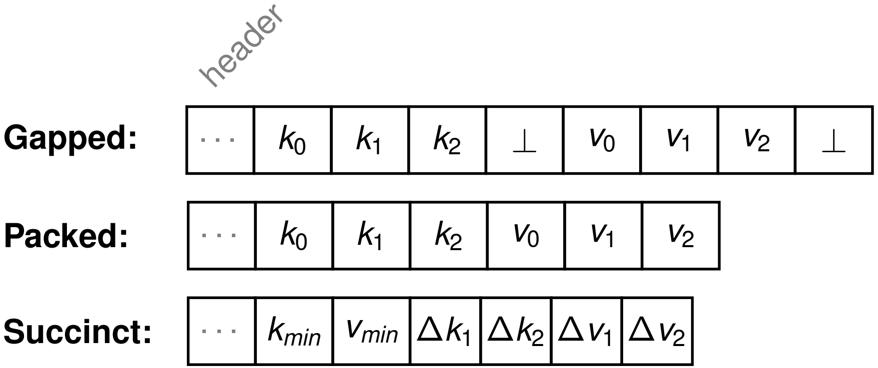


Table: Leaf encodings storing 64-bit key-value pairs and performance implications on lookups.

Leaf Node Encoding	Average Size	Instruc.	LLC Misses	Branch Misses
Gapped	4096B	85	2.1	4.44
Packed	2872B	84	1.4	4.46
Succinct	1076B	341	1.1	6.69

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

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

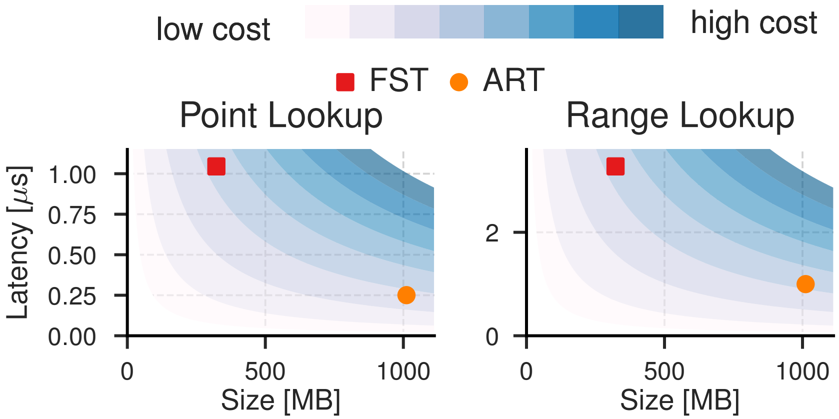
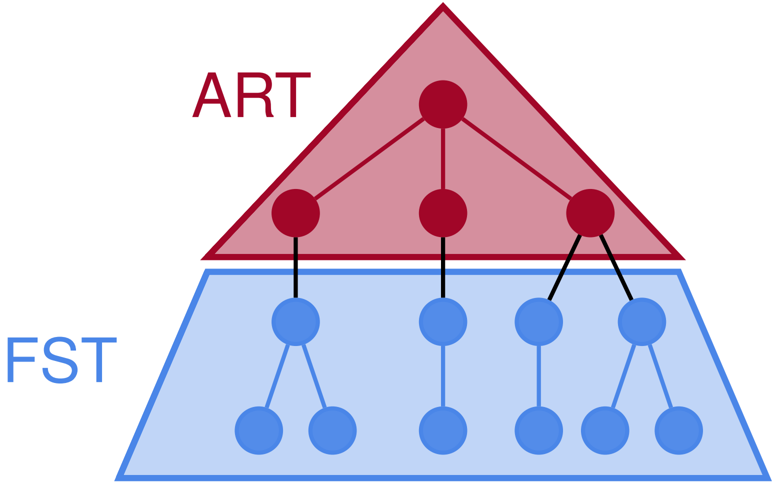


Figure: Query latency and index size of ART and FST

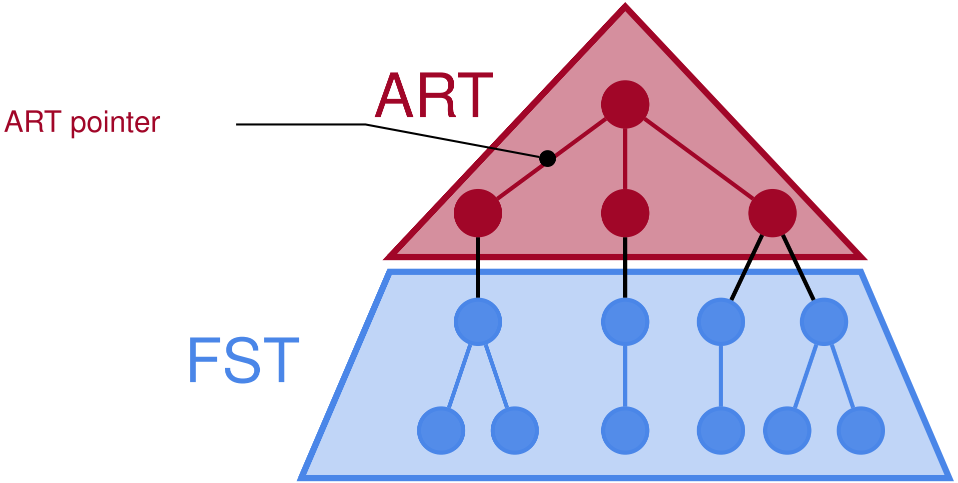
Experiment Setup:

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

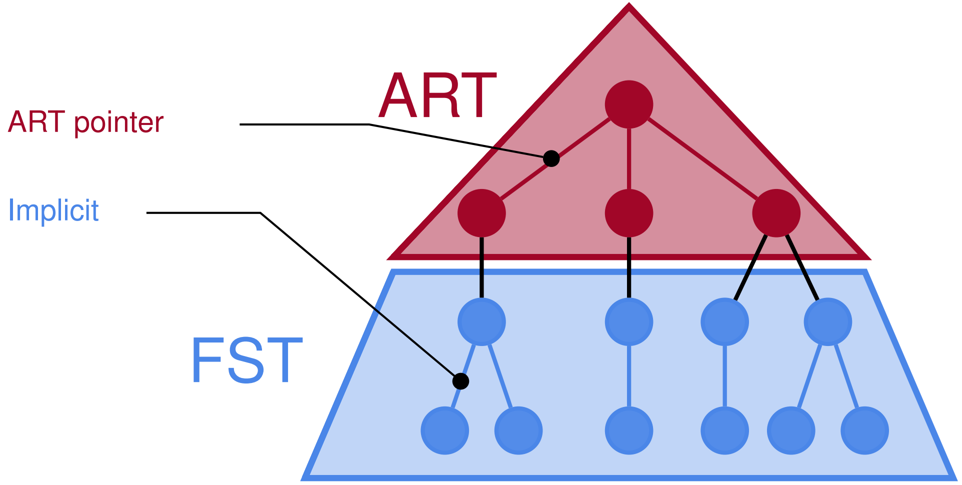
Application II: Adaptive Hybrid Trie



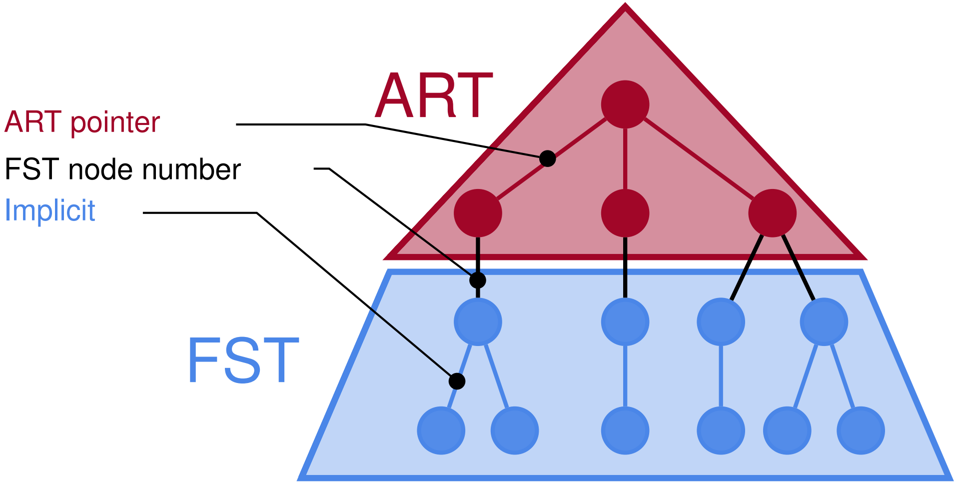
Application II: Adaptive Hybrid Trie



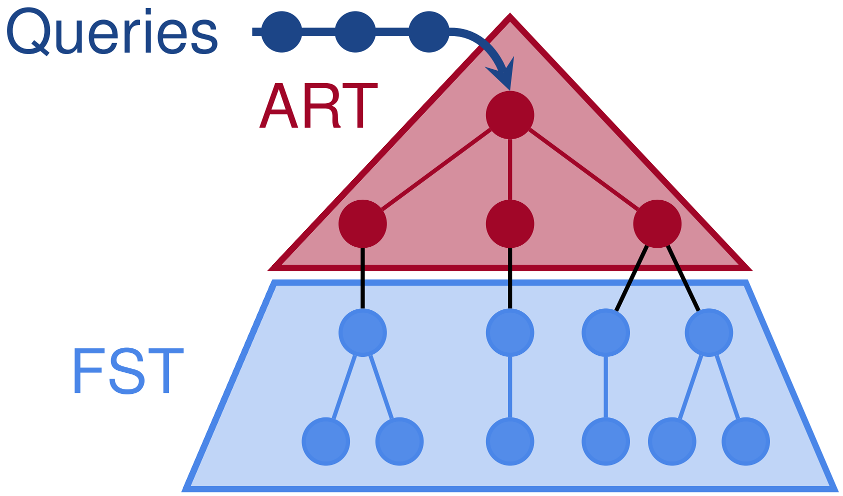
Application II: Adaptive Hybrid Trie



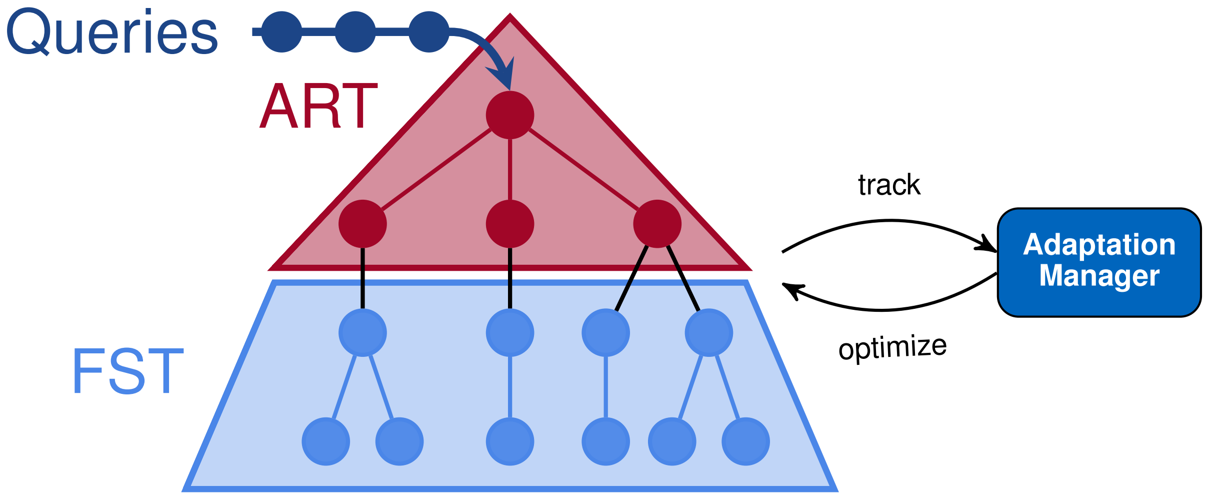
Application II: Adaptive Hybrid Trie



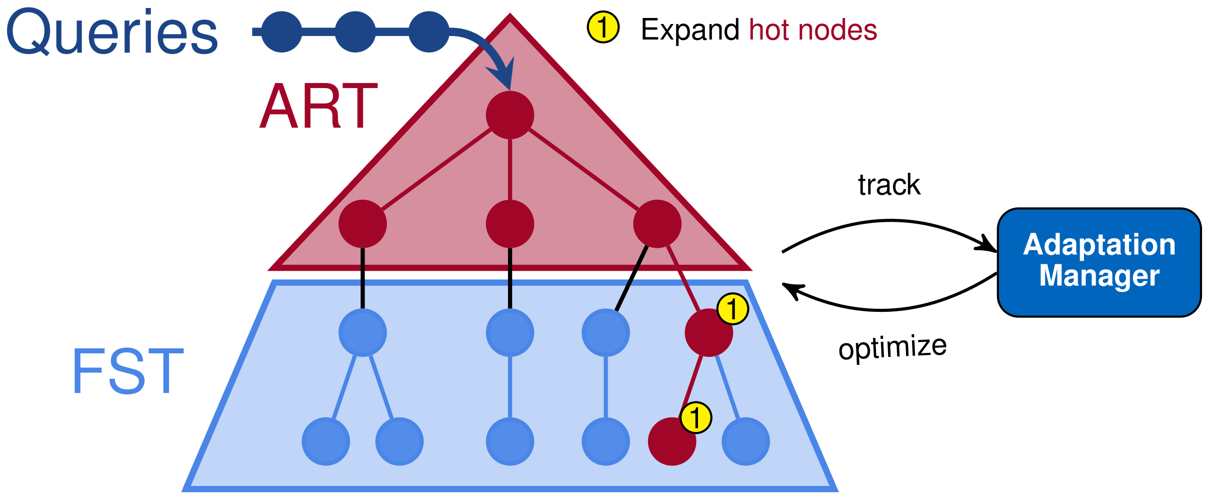
Application II: Adaptive Hybrid Trie



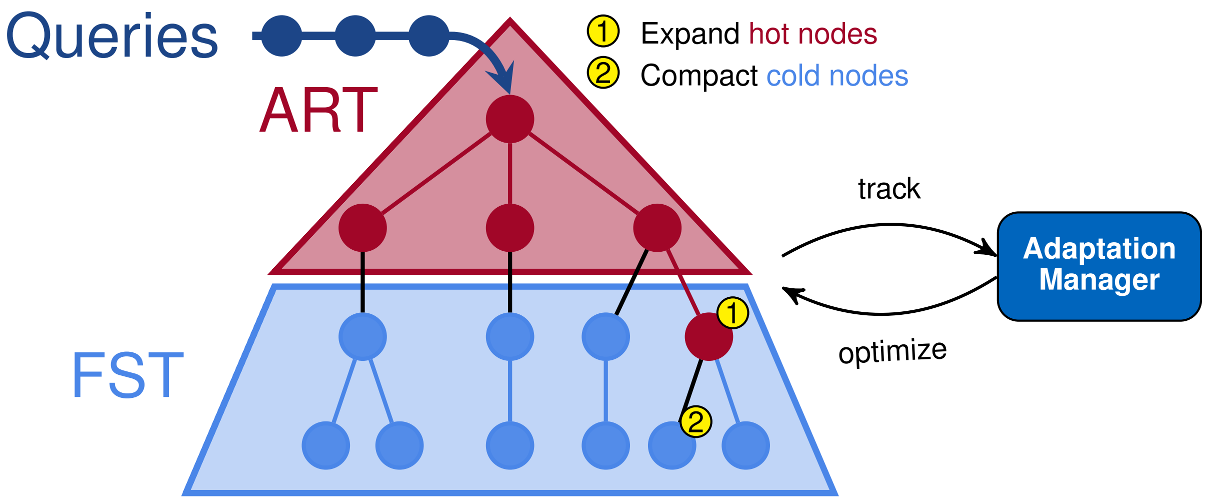
Application II: Adaptive Hybrid Trie



Application II: Adaptive Hybrid Trie



Application II: Adaptive Hybrid Trie

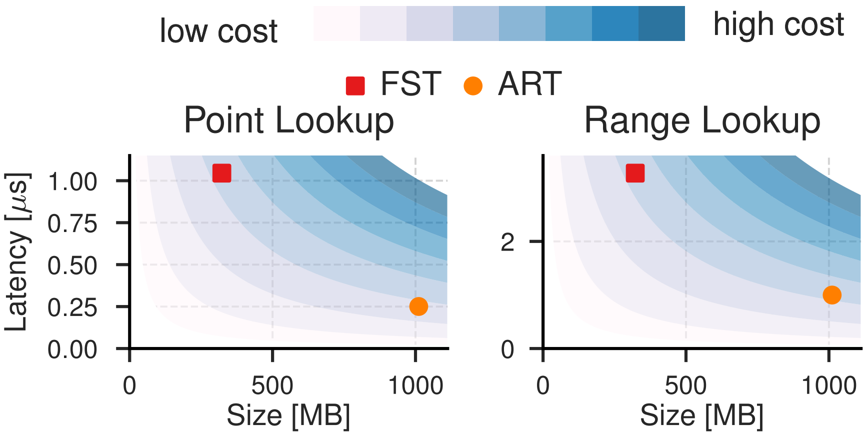


Evaluation

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

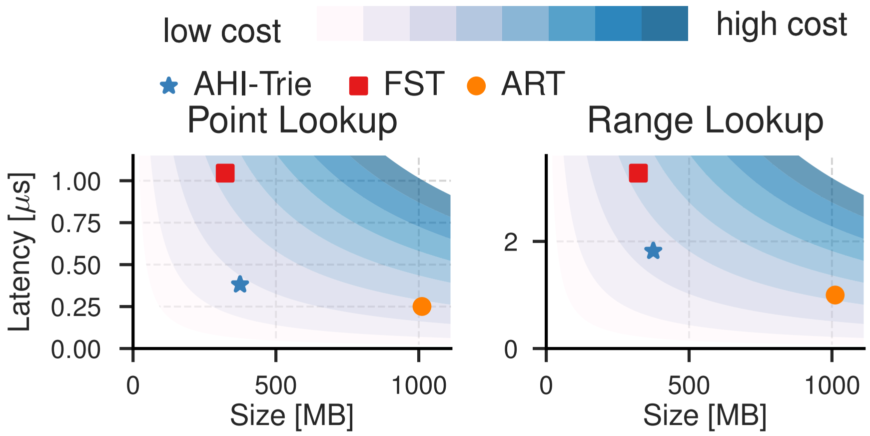
Evaluation: Hybrid Trie – Space & Performance



Experiment Setup:

- Dataset: 33M unique email adresses (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

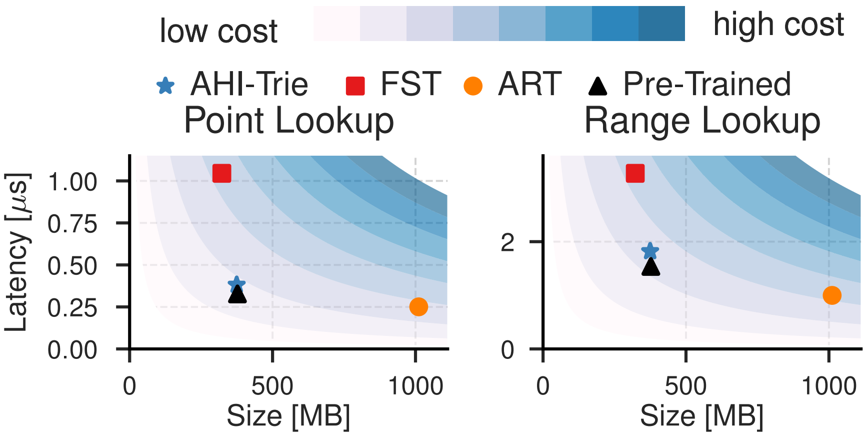
$\Rightarrow$ reduces index size by 63% comp. to ART

$\Rightarrow$ improves performance by 2.7x comp. to FST

Experiment Setup:

- Dataset: 33M unique email adresses (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

$\Rightarrow$ reduces index size by 63% comp. to ART

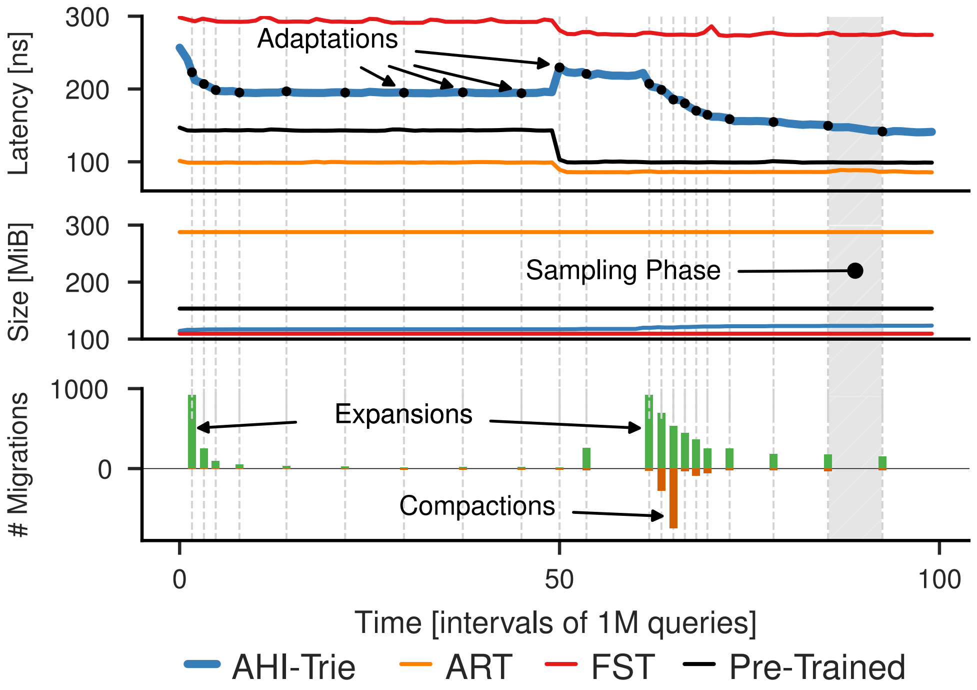
$\Rightarrow$ improves performance by 2.7x comp. to FST

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

Experiment Setup:

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

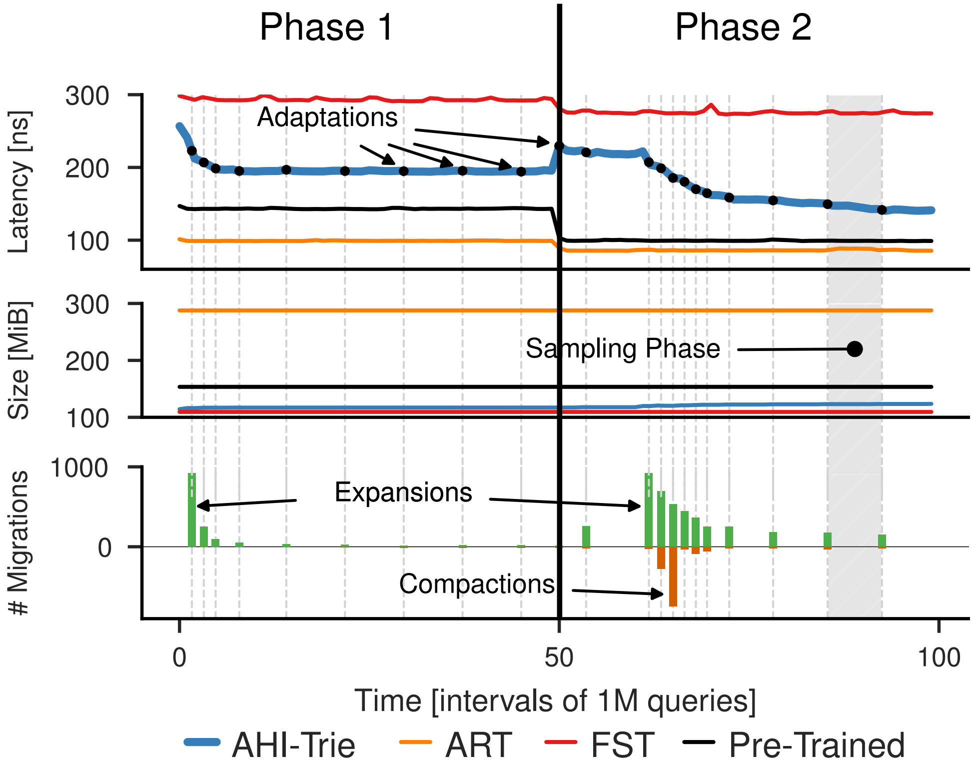
Evaluation: Hybrid Trie – Workload Adaptation



Experiment Setup:

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

Evaluation: Hybrid Trie – Workload Adaptation

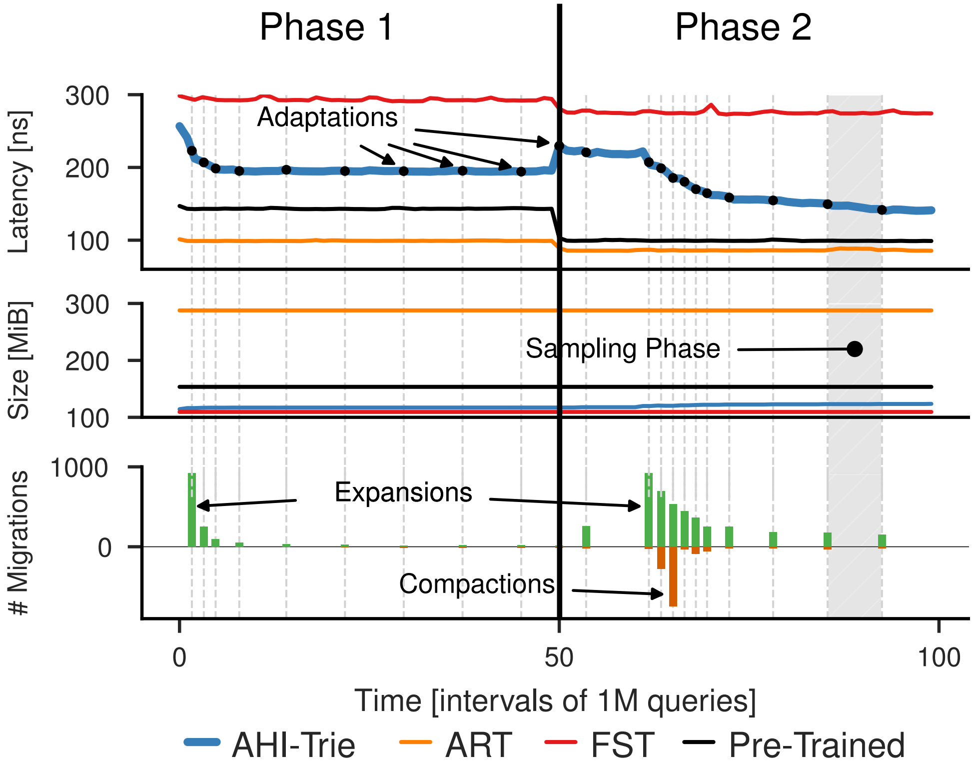


Conclusions:

Experiment Setup:

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

Evaluation: Hybrid Trie – Workload Adaptation



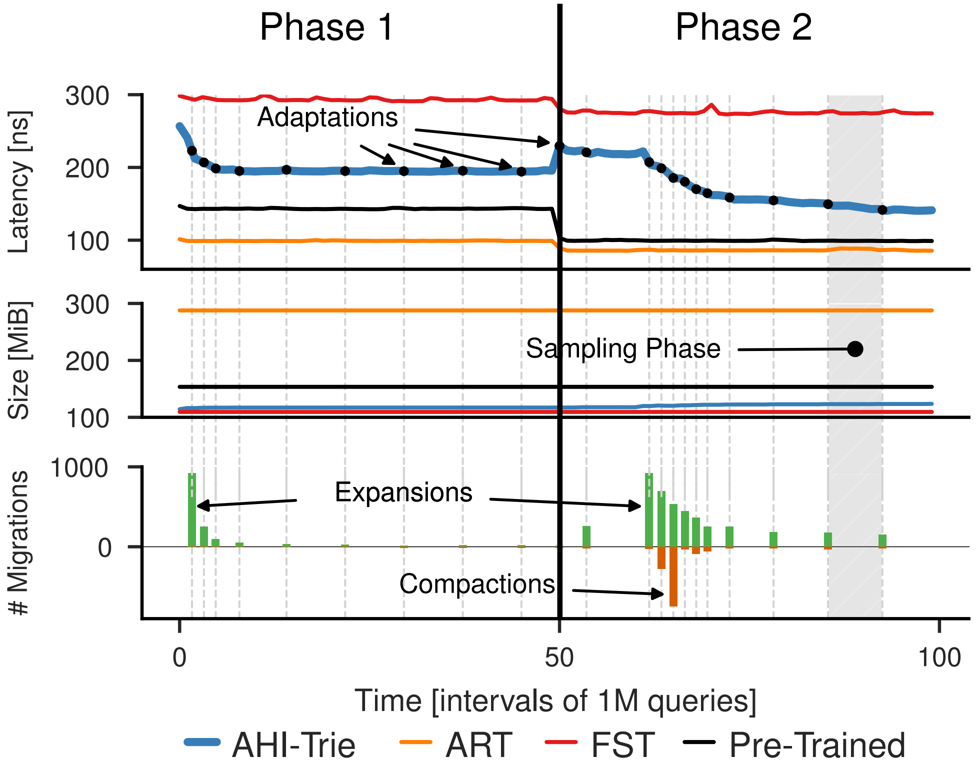
Conclusions:

$\Rightarrow$ Adaptive Encoding Optimizations improve latency

Experiment Setup:

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

Evaluation: Hybrid Trie – Workload Adaptation



Conclusions:

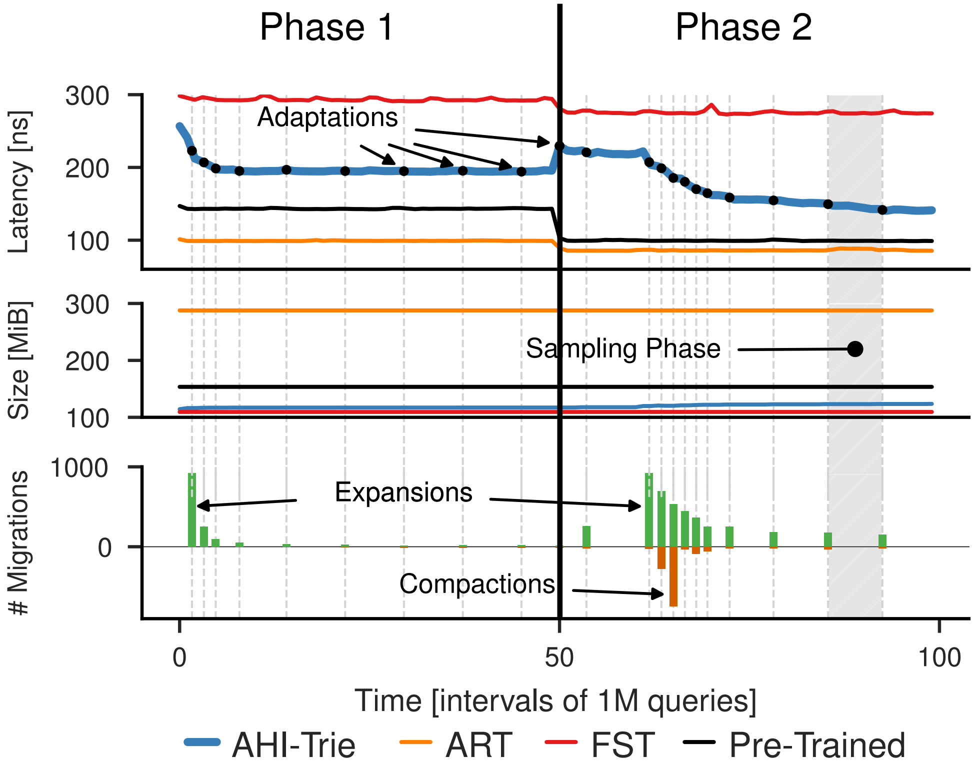
⇒ Adaptive Encoding Optimizations improve latency

⇒ 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



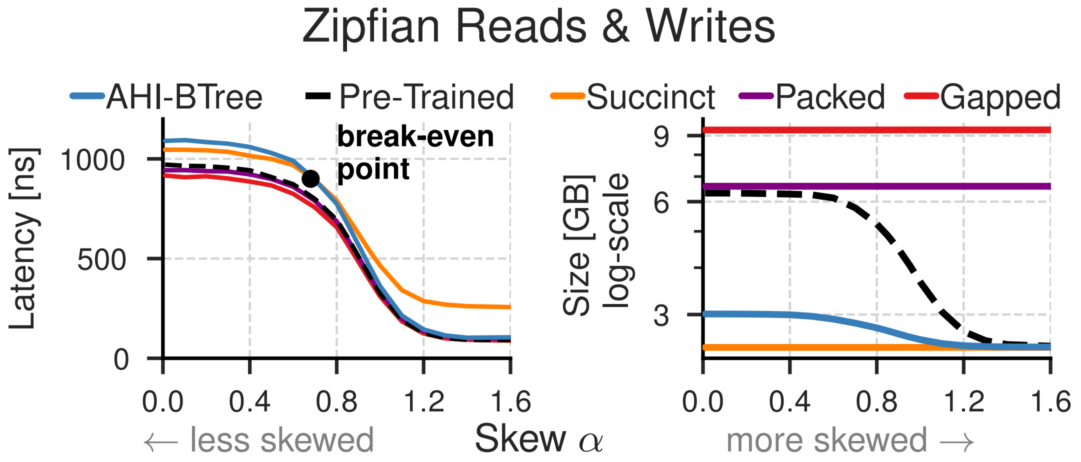
Conclusions:

- ⇒ 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

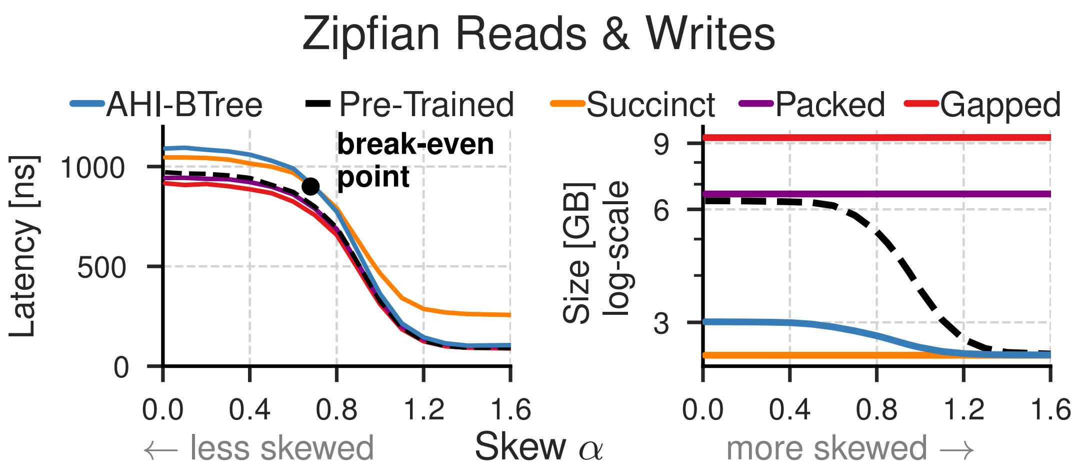
Evaluation: Hybrid B+-Tree – Skewed Workloads



Experiment Setup:

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

Evaluation: Hybrid B+-Tree – Skewed Workloads



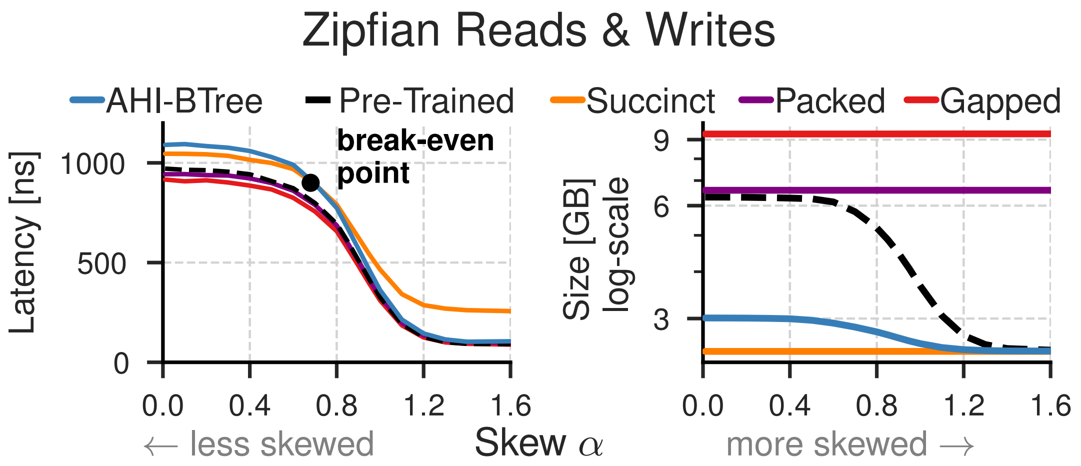
Conclusions:

- Adaptive Hybrid Indexes perform best under skewed workloads

Experiment Setup:

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

Evaluation: Hybrid B+-Tree – Skewed Workloads



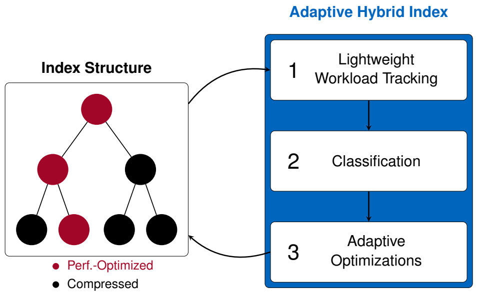
Conclusions:

- 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

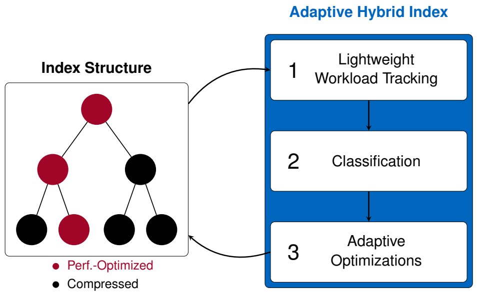
Conclusions

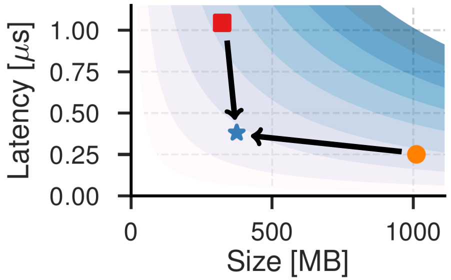


Generic framework to create Adaptive Hybrid Indexes

Conclusions

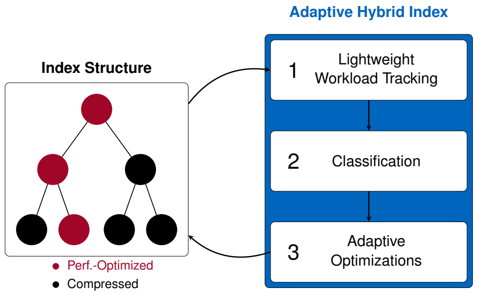
Generic framework to create Adaptive Hybrid Indexes



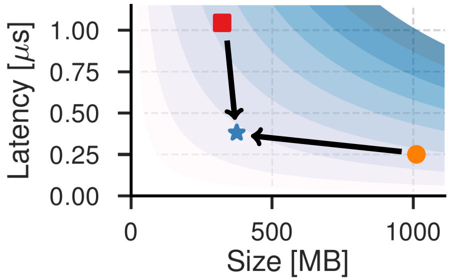


Reduce storage overheads while retaining high performance

Conclusions



Generic framework to create Adaptive Hybrid Indexes



Reduce storage overheads while retaining high performance

Evaluated the framework using B+-trees and prefix trees