School of Computation, Information and Technology
Decentralized Information Systems and Data Management
Technical University of Munich

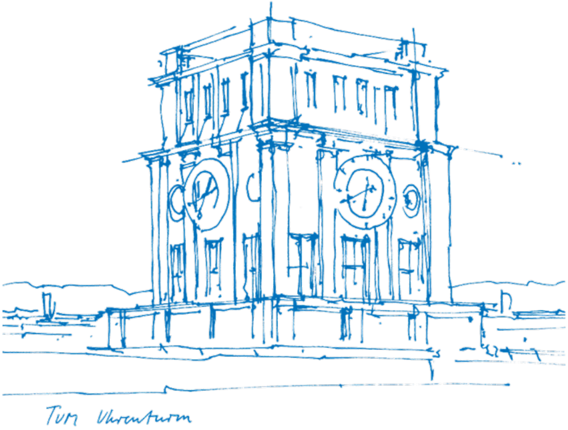
Hist-Tree

Status Update

Mert Sidal

School of Computation, Information and Technology
Decentralized Information Systems and Data Management
Technical University of Munich

December 19th, 2024



Outline

1 Recap of the Hist-Tree

2 Main Challenge

3 Implementation

4 Demonstration

5 Next Steps

Recap of the Hist-Tree

- **Hist-Tree** tries to compete with state-of-the-art Learned Indexes
- **Basic assumptions**: sortedness and range of data
- **Idea**: histogram to partition data into equal-sized bins
- Physically organized into **two** Arrays of 32-bit Integers
  1. Inner Nodes **with** Child Pointers
  2. Leaf Nodes

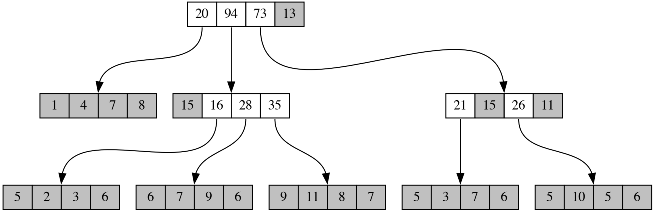


Figure 1 Example Hist-Tree with 200 keys in the range $[0, 1000)$

Outline

1 Recap of the Hist-Tree

2 Main Challenge

3 Implementation

4 Demonstration

5 Next Steps

Main Challenge

Understanding the Build Process

Context:

- The paper provided only a brief explanation of the build process
- This lack of detail made it difficult to fully comprehend the intended implementation

My Solution:

- I designed my own **bit-vector-based approach** to manage the data structure efficiently

Outcome:

- The bit-vector approach has proven to be effective, enabling:
  - □ Clear partitioning of keys → optimal structure
  - □ Efficient use of resources during the build phase
  - □ Optimization through SIMD

Outline

1 Recap of the Hist-Tree

2 Main Challenge

3 Implementation

4 Demonstration

5 Next Steps

Implementation Concepts

Builder.h:

- Keys: $\{0, 2, 3, 5, 6, 7\}$
  - □ `createBitVector()`

    $$10110111$$

  - □ `partitionVector()`

    $$10 \| 11 \| 01 \| 11$$

  - □ `countBinElements()`

    $$1 \| 2 \| 1 \| 2$$

- Based on the mentioned functions `build()` returns the `HistTree`

HistTree.h:

- `getSearchBound()` returns a range of *threshold* size, which can then be scanned, e.g. Binary Search

common.h:

- Contains utilities like the `SearchBound` struct and the class `Visualizer`:

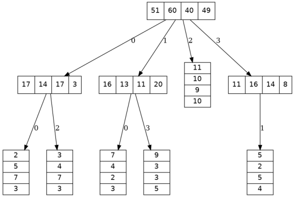


Figure 2 Automated image from random tree

Outline

1. Recap of the Hist-Tree

2. Main Challenge

3. Implementation

4. Demonstration

5. Next Steps

Demonstration

Figure 3 Placeholder

Outline

1 Recap of the Hist-Tree

2 Main Challenge

3 Implementation

4 Demonstration

5 Next Steps

Next Steps

- **Implement**: `remove(key)`, `insert(key)`
- Expand Google Test cases
- Benchmark `read(key)` with Search On Sorted Data (SOSD)
- Benchmark with Google Benchmark
- Optimization ideas:
  - □ Memory optimization via dynamic resizing and reordering (avoid fragmentation)
  - □ More SIMD during building phase
  - □ Parallel build (with merging?) and query execution
  - □ Cache
  - □ Adaptive parameter optimization before construction based on data size and distribution

⇒ Identify **Hot Paths** and focus on them