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

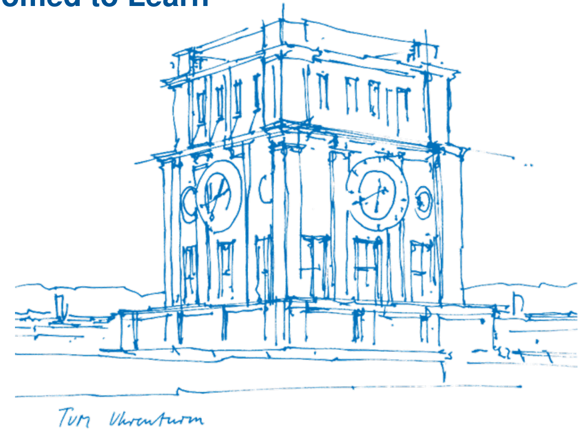
Hist-Tree

Hist-Tree: Those Who Ignore It Are Doomed to Learn

Mert Sidal

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

November 7th, 2024

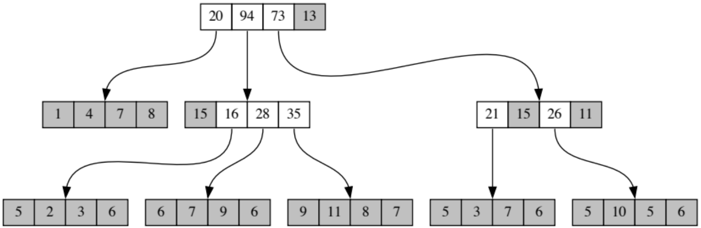


Outline

General Information

Hist-Tree: Those Who Ignore It Are Doomed to Learn

- written by Andrew Crotty – Brown University **[paper]**
- CIDR 2021 (Rank A)
- **Example** of 200 keys in the range $[0, 1000)$:



(b) Hist-Tree

0	1	2	6	187
1	0	1	5	12
2	20	3	4	5
3	35	40	42	45
4	51	57	64	73
5	79	88	99	107
6	7	135	8	176
7	114	119	122	129
8	150	155	165	170

(c) Compact Hist-Tree

Figure 1: An example dataset with the corresponding basic and compact Hist-Tree.

Outline

Motivation

- Hist-Tree tries to compete with state-of-the-art **Learned Indexes**
  - Basic assumptions: sortedness and range of data
  - $1.8 - 2.7x$ performance increase in terms of Lookup Latencies compared to RMI, PGM-Index, and RadixSpline (according to the paper)
- Potential for innovation and improvement
  - Handle Updates more efficiently
  - Parallelization of Operations
  - Memory Footprint Reduction
  - Add Cache together with Batch Updates
  - Error-Handling

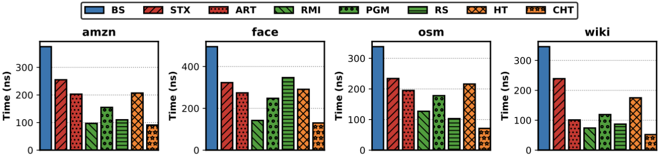


Figure 4: Lowest lookup latency achieved by each comparison point.

Outline

Overview of Project
Architecture

- Equal-sized Bins per Level (until specified threshold/error)
- Physically organized into **two** Arrays of 32-bit Integers
    1. Inner Nodes (i.e., nodes where at least one bin has a child)
    2. Leaf Nodes

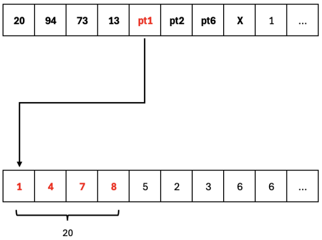


Figure Inner-Array (top) & Leaf-Array (bottom)

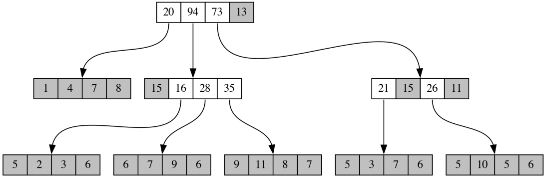


Figure Hist-Tree Example

Overview of Project

Implementation of Lookup besides Update, Delete, Bulk Load, and Build

```c
#define BINS 4
#define FLAG (1 << 31)

size_t lookup(ht_t *ht, uint64_t key)
{
    //ensure key is within range
    if (key < ht->min) return 0;
    if (key > ht->max) return ht->len;

    //initialize variables
    uint32_t next, *node = ht->node;
    int width = ht->width, done = 0;
    size_t i, bin, pos = 0;
    key -= ht->min;

    //descend tree
    do {
        //calculate bin and update running sum
        bin = key >> width;
        for (i = 0; i < bin; i++)
            pos += node[i];

        //return when done set
        if (done)
            return pos;

        //set done, next, and node based on flag
        done = node[BINS + bin] & FLAG;
        next = node[BINS + bin] & ~FLAG;
        node = done ? ht->leaf + next * BINS
                    : ht->node + next * BINS * 2;

        //adjust key and width
        key -= bin << width;
        width -= log2(BINS);
    } while (1);
}
```

Figure 2: Hist-Tree lookup function

1. **Root Node**:
   - Calculate the bin index: $567/256 = 2$.
   - Sum counts from all preceding bins: $20 + 94 = 114$.
   - Adjust the key for the next level: $567 - 256 \times 2 = 55$.
2. **Child Node**:
   - Calculate the bin index with new bin width: $55/64 = 0$.
   - Update running sum: $114 + 0 = 114$.
   - Adjust the key: $55 - 64 \times 0 = 55$.
3. **Leaf Node**:
   - Calculate the bin index for final level: $55/16 = 3$.
   - Add final bin counts to running sum: $114 + 5 + 3 + 7 = 129$.
4. **Result**:
   - Bounded search range, with 129 (the starting point)

Outline

Project Testing

- Unit Testing Framework for expected behaviour (**Google Test**)
- Test Cases for Basic Operations:
  - Insert & Delete: correct update of data structure
  - Lookup: correct results, both for found and not found elements
- Edge Cases and Exception Handling:
  - Empty Structure
  - Boundaries
  - Delete non-existing element
  - ...
- Automating Tests: **CI/CD Pipeline**

```
ASSERT_EQ(1,1);
EXPECT_EQ(1,1);
```

Outline

Benchmarking

- Focus:
  - Latencies
  - Index Size
  - Build Time
- Methodology/Tools:
  - Google Benchmark
  - Search on Sorted Data (SOSD)
- Procedure:
  1. Setup: control hardware, limit background tasks
  2. Iterations: Run multiple passes to average results
  3. Edge Cases: Test extremes (e.g., max/min keys)

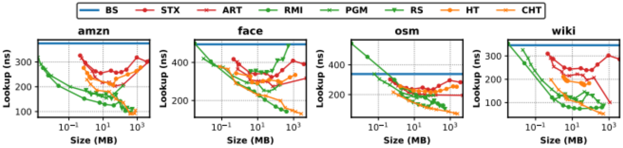


Figure 5: Index size vs. lookup latency

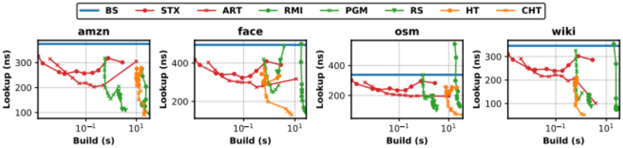


Figure 6: Build time vs. lookup latency

Outline

Milestones

Week 1	Weeks 2-5	Weeks 6-7	Weeks 8-9	Week 10
Planning	**Core Imple-mentation**	**Benchmarking Setup & Initial Testing**	**Optimization**	**Analysis & Report**
Define objectives; structure project	Build basic data structure operations; test for basic functionality across different inputs	Integrate benchmarking tools (Google Benchmark, SOSD); gather baseline performance data	Optimizations based on benchmarks	Analyze results; summarize findings; prepare report