Adaptive Extendible Hash Maps

Victor Lei, Leonhard Spiegelberg
1. Extendible Hash Maps

Key idea: Use dynamically resizing directory and buckets

Can be seen as combination of Tries + Hashing

- $h(k1) = 100100$
- $h(k2) = 010110$
- $h(k3) = 110110$
2. Experiments

Observe how running time changes for different read-write ratios as we adjust the bucket size.

**Hypothesis:** A higher ratio of reads to writes should favor a lower bucket size and vice versa.
2.1 Experimental Setup

Dedicated machine that minimizes external factors such as caching and CPU load

Java and C++ implementation of a vanilla Extendible Hash Map
2.1 Experimental Data

Generate a random set of 1 million integer reads and writes based on a specific read-write ratio (no deletion)
3.1 Results (10% reads)

More writes than reads favors a larger bucket size due to less splitting
3.2 Results (50% reads)

Equal writes compared to reads still favors a larger bucket size
3.3 Results (90% reads)

Even with many 90% reads, it still favors a larger bucket size
3.4 Results (99% reads)

At 99% reads, a lower bucket size is actually better
4. Takeaway

Hypothesis is plausible, but writes are much more expensive than reads

Use smaller bucket sizes carefully or else performance takes a big hit
Can we improve EHM by using dynamically different data structures and bucket sizes?