IcebergHT: High-Performance Hash Tables Through Stability and Low Associativity

SIGMOD 2023

Prashant Pandey, Michael A. Bender, Alex Conway, Martin Farach-Colton, William Kuszmaul, Guido Tagliavini, Rob Johnson
Hash tables are everywhere!

Built into many languages…

Built into many software packages…

And performance is critical to many applications.
Hash table performance has a three-way trade-off between insertion speed, query speed, and space.
Hash table design mechanism

**Stability**
Items don’t move after insertion

**Low associativity**
Map each item to one a small number of locations

**Space efficiency**
Minimum overhead from pointers or over provisioning
Hash table design mechanism

Stability
Items don’t move after insertion

Low associativity
Map each item to one a small number of locations

Space efficiency
Minimum overhead from pointers or over provisioning

Fast insertion
Hash table design mechanism

**Stability**
Items don’t move after insertion

**Low associativity**
Map each item to one a small number of locations

**Space efficiency**
Minimum overhead from pointers or over provisioning

- Fast insertion
- Fast queries
Hash table design mechanism

**Stability**
Items don’t move after insertion

**Low associativity**
Map each item to one a small number of locations

**Space efficiency**
Minimum overhead from pointers or over provisioning

- Fast insertion
- Fast queries
- Low space
Achieving all three is a long-standing open problem in hash table design.
Our results:

- Stability
- Low associativity
- Space efficiency

Insertion performance:
- 50% to 3X faster on PMEM
- Up to 2X faster on DRAM

Query performance:
- 20% to 2X faster on PMEM
- Competitive on DRAM

Space efficiency:
- 17% space overhead compared to 3X for other hash tables

IcebergHT achieves stability, low associativity, and space efficiency at the same time.
Our results:

IcebergHT achieves stability, low associativity, and space efficiency at the same time.

- Stability
- Low associativity
- Space efficiency

Insertion performance:
- 50% to 3X faster on PMEM
- Up to 2X faster on DRAM

Space efficiency:
- 17% space overhead compared to 3X for other hash tables

IcebergHT also achieves:
- Almost linear scalability with increasing threads
- Fenceless crash safety on PMEM
For example: linear probing

- Stable
- Associativity $\approx \frac{\log N}{(1 - \alpha)^2}$ ($\alpha =$ load factor)
- E.g., $N = 1$Billion, $\alpha = 95\%$, associativity = 12000

Must choose between low associativity and space efficiency.
For example: cuckoo hashing

- Low associativity: queries must check only 2 cache lines
- Space efficient, load factor > 95%
- But not stable

Insertion performance drops significantly due to excessive kicking at high load factors.
Other hashing schemes:

- Other hashing schemes also lack one or more of these properties
- **Chaining**: not low associativity
- **Robin hood**: not stable and not low associativity at high load factors
- **Hopscotch**: not stable
- **Quadratic probing**: not stable and not low associativity at high load factors
Theorem: if you throw $N$ balls into $N/\log N$ bins using minimum of two choices, the fullest bin will have $\log N + \log \log N + O(1)$ balls W.H.P.

- By Berenbrink, Czumaj, Steger, Vöcking 2000
An almost solution: two choice hashing

- **2-choice hashing**: hash to two buckets and put item in emptier bucket
- Stable: no kicking
- Low associativity: $O(\log N)$
- Space efficient: load factor $1 - o(1)$
An almost solution: two choice hashing

- **2-choice hashing**: hash to two buckets and put item in emptier bucket
- Stable: no kicking
- Low associativity: $O(\log N)$
- Space efficient: load factor $1 - o(1)$

**Problem**: it does not hold when we delete items

**Opportunity**: theorem does hold with deletions if average bucket occupancy is $O(1)$
**Iceberg hashing**

- **Iceberg theorem**: if you throw $N$ balls into $N/\log N$ bins of size $\log N + o(\log N)$, the number of overflow balls will be $O(N/\log N)$.

- **Idea**: use single-choice front yard to absorb most items.

- Backyard has average occupancy of $O(1)$.
Iceberg hashing

- **Iceberg theorem**: if you throw $N$ balls into $N / \log N$ bins of size $\log N + o(\log N)$, the number of overflow balls will be $O(N / \log N)$

- **Idea**: use single-choice front yard to absorb most items

- Backyard has average occupancy of $O(1)$

**Problem**: buckets in the front yard span many cache lines, so queries must load many cache lines.
Problem: buckets in the front yard span many cache lines, so queries must load many cache lines.

Solution: store a fingerprint table.
Problem: buckets in the front yard span many cache lines, so queries must load many cache lines.

Solution: store a fingerprint table.

Use AVX512 to query metadata.
IcebergHT implementation

- **Highly concurrent** operations
- IcebergHT supports **in-place resizing**; reduces peak memory usage
  - Multi-threaded resizes are implemented using distributed reader-writer locks
- **Crash safety is trivial**
  - Using CLWB; no need for a fence between key & value writes
  - Metadata stays in DRAM and is reconstructed during recovery
Performance using 16 threads for PMEM hash tables.

Iceberg outperforms state-of-the-art hash tables across all operations.
IcebergHT offers higher space efficiency compared to Dash (extendible) and CLHT (chaining) hash tables.
Performance using 16 threads for DRAM hash tables.

Iceberg outperforms state-of-the-art hash tables for insertions and offers similar performance to CLHT for queries.

IcebergHT deletes are slower.
IcebergHT can achieve high space efficiency and maintain insertion throughput.
CLHT space efficiency drops quickly.
CuckooHT insertion throughput drops at high load factor.
Takeaways

• Stability yields:
  • Fast updates (especially on PMEM)
  • Good scalability with threads
  • Crash safety (please refer to paper)
• Low associativity yields:
  • Fast lookups
  • Small metadata
• Iceberg hashing gives both high performance and high space utilization
• Also, supports resizing without drop in instantaneous latency
• Metadata scheme is also an example of general maplet data structure

Source code: https://github.com/splatlab/iceberghashtable