The Dynamic Cuckoo Filter

Hanhua Chen, Liangyi Liao, Hai Jin, Jie Wu

Huazhong University of Science and Technology

Temple University
Two Core Problems

• **Set representation**
  • organize the information of the elements of a set using some data structure
  • make the information of the set elements operable by corresponding methods

• **Set membership testing**
  • determine whether an element with a given attribute value belongs to a given set
Emergence of Dynamic Sets

• Real world big data applications
  • Cloud storage
  • Stream applications

• Stringent requirements for dynamic sets
  • Flexibly capacity extending or reducing
  • Reliable delete operation
## Existing Designs

<table>
<thead>
<tr>
<th>Existing Designs</th>
<th>Elastic capacity</th>
<th>Reliable delete</th>
<th>Space cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloom filter</td>
<td>✗</td>
<td>✗</td>
<td>1</td>
</tr>
<tr>
<td>Counting Bloom filter</td>
<td>✗</td>
<td>✓</td>
<td>N</td>
</tr>
<tr>
<td>Cuckoo filter</td>
<td>✗</td>
<td>✓</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic Bloom filter</td>
<td>✓</td>
<td>✗</td>
<td>N</td>
</tr>
</tbody>
</table>
Bloom filter (BF)

1 bit
0 1 0 1 0 0 1 0 1 0 1 0
Cuckoo Filter (CF):

- **An array of buckets**
- Each bucket has \( b \) entries
- **Fingerprint** based matching
- **Relocation** when collision

\[
\begin{align*}
  h_1(x) &= \text{hash}(x) \\
  h_2(x) &= h_1(x) \oplus \text{hash}(\xi_x)
\end{align*}
\]

Drawback: fixed capacity

“Cuckoo filter: Practically better than bloom,” in *CoNEXT*, 2014
Dynamic Bloom Filter

• Linked list of \( s \) Counting Bloom Filters (CBF)
• Extends capacity by appending new building blocks of CBFs
• Drawback: unreliable deletion

Multiple Address Problem in DBF

- Give up deleting the element found in multiple building blocks
- Raises up false positives
Our Design: Dynamic Cuckoo Filter (DCF)

- DCF uses Cuckoo Filter (CF) as building block
- Extends capacity by appending new building blocks
- Monopolistic fingerprint avoid multiple address problem

Insert & Query & Deletion
Dynamic Cuckoo Filter (DCF)

Compaction

- Leverage greedy strategy
- Empty a building block with the least fingerprint movements
Testing & Analysis

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Maximum set cardinality</th>
<th>False positive rate</th>
<th># of building blocks</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Dataset</td>
<td>46,080</td>
<td>1.17x 10^{-2}</td>
<td>6~96</td>
<td>I/Q/D speed</td>
</tr>
</tbody>
</table>

Baseline: the Dynamic Bloom Filter (DBF)

Two sets of experiment

- Varying set cardinality $N$ under optimized building block number $s$
- Varying $s$ under a fixed set cardinality $N$
Space Optimization

Space optimization under 5 different set size distribution

![Graph showing space optimization results under different set size distributions. The graph illustrates the ratio of DCF's memory to CF against set cardinality. The legend indicates different distributions: uniform, normal, minimum Zipf, maximum Zipf, and random Zipf. A 25% decrease is marked on the graph.]
Space Optimization

Space optimization under 5 different set size distribution:

75%↓
Insert Time

50%↑

Changes slightly
Query Time

- **DCF**
- **DBF**

Query time (s) vs. Set cardinality

80% ↑

Positive correlation

Query time (s) vs. Value of s
Application in File Backup System

- **DRAM**
  - Chunk Digest

- **SSD**
  - Chunk Cache

- **HDD**
  - Chunk Index
  - Chunk Sequence
  - Chunk Physical Storage

- Version 2
- Version 1
## Data Sets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Maximum set cardinality</th>
<th>False positive rate</th>
<th># of building blocks</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel Dataset</td>
<td>2,371,618</td>
<td>$1.17 \times 10^{-2}$</td>
<td>1~50</td>
<td>175</td>
</tr>
</tbody>
</table>

Memory Cost Reduction

![Graph showing memory cost reduction](image)

- Reduction by 75%
Disk I/O Reduction

Number of disk I/O

Kernel version

DCF
Baseline

62.5%↓
Conclusion

• A novel DCF design for approximate set representation and membership testing for a dynamic set

• Support reliable element deletion and flexible expending

• Memory reduced by 75% compared to DBF
Thank you!

Download DCF Toolkit
https://github.com/CGCL-codes/DCF