Bloofi: A Hierarchical Bloom Filter Index with Applications to Distributed Data Provenance

Adina Crăiniceanu

U.S. Naval Academy

Cloud Intelligence - A VLDB Workshop, August 26, 2013
Outline

- Problem
- Bloofi
  - Description
  - Search
  - Maintenance
- Distributed Data Provenance
- Experimental results
- Conclusions and future work
Problem

- Federated cloud environment
- Semi-independent clouds
- Each cloud keeps control of its data
- Data needs to be shared on demand

- Given a set of sets (the clouds)
- Find all the sets (clouds) that contain a given element X
Challenges

- Number of participants is high
- Broadcasting a query is expensive
- Processing each query at all locations is expensive
- Creating a global distributed index is not possible
  - Individual clouds maintain control of the data
  - Volume and rate of data insertion is high
Our Solution

- Each cloud maintains a Bloom filter of its data
- Bloom filters shared with a central location
- Construct a Bloom Filter Index (Bloofi) at central location
- Queries are processed first at central location, and sent only to the clouds that (might) have the answer
Outline

- Problem
- Bloofi
  - Description
  - Search
  - Maintenance
- Distributed Data Provenance
- Experimental results
- Conclusions and future work
Bloom Filters

- Bit array of size $m$
- $k$ hash functions with range $[0, m-1]$

Insert $x$: $h_1(x) = 2$
$h_2(x) = 9$
$h_3(x) = 5$

```
0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0
go 1 2 ........................................ m-1
```
Bloom Filters

- Bit array of size $m$
- $k$ hash functions with range $[0,m-1]$

Query $y$: $h_1(y) = 2$
$h_2(y) = 9$
$h_3(y) = 4$

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

0 1 2 ........................................... $m-1$
Bloom Filters Advantages

- Compact representation of sets
- Efficient insertion and testing
- Probability of false negatives = 0
- Trade-off between size of filter and false positive probability
Bloofi: A **Bloom Filter Index**

![Diagram of a Bloom Filter Index](image-url)
Bloofi Invariants

- Each Bloom filter value represents the union of all subsets in the subtree => useful for pruning during search
- Balanced tree
- Each non-root node has between $d$ and $2d$ descendants
- Each node has 1 value
Bloofi Search

Q Key: 4: \( h_1(4) = 4 \)

A: Id = 5
Bloofi Insert

Hd = 4

Hd = 3

Hd = 2

Hd = 4

Hd = 3

Hd = 3
Bloofi Insert

Hd = 4

Id: 7
1 1 1 0 0 1 0 0
Hd = 3

Id: 2
0 1 0 0 0 0 0 0
Hd = 3

Id: 1
1 0 0 0 0 0 0 0
Hd = 3

Hd = 4

Id: 8
0 0 0 0 1 0 1 0

Id: 6
0 0 0 0 0 0 1 0
Hd = 4

Hd = 3

Id: 4
0 0 0 1 0 0 0 0

Id: 5
0 0 0 0 1 0 0 0
Hd = 2

Id: 10
0 0 1 0 0 1 0 0

Id: 9
1 1 1 1 1 1 1 0
Id: 10
0 0 1 0 0 1 0 0

Id: 10
0 0 0 0 0 1 1 0
Hd = 3

Id: 10
0 0 0 0 0 1 1 0
Bloofi Split
Bloofi Properties

- Search cost is $O(d \times \log_d N)$ in general and $O(N)$ in worst case.
- Storage cost is $O(N/d)$.
- Insert cost and delete cost are $O(d \log_d N)$.
- Update cost is $O(\log_d N)$. 

Outline

- Problem
- Bloofi
  - Description
  - Search
  - Maintenance
- Distributed Data Provenance
- Experimental results
- Conclusions and future work
Distributed Data Provenance

Why not distributed index of all data?
- Some sites want data locality
- Different storage solutions at each site
- All data not created equal, not all data should go to the "headquarters"
Outline

- Problem
- Bloofi
  - Description
  - Search
  - Maintenance
- Distributed Data Provenance
- Experimental results
- Conclusions and future work
Experiments Set-up

- Simulator written in Java
  - Uses open-source Bloom filter impl [Skjegstad]
- Experiments run on a Dell Latitude E6510 2.76GHz Intel Core I7 CPU, 4 GB RAM
- Performance metrics:
  - Search cost – nb Bloom filters searched to find all matches
  - Search time – time to find all matches
  - Maintenance cost – nb of nodes accessed during insert/delete/update
- Compare with “baseline” case – no index
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range of values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N – Nb of Bloom filters</td>
<td>100-100,000</td>
<td>1000</td>
</tr>
<tr>
<td>d – Bloofi order</td>
<td>2-22</td>
<td>2</td>
</tr>
<tr>
<td>Bloom filter size (bits)</td>
<td>1000-1,000,000</td>
<td>100,992</td>
</tr>
<tr>
<td>Bloofi construction method</td>
<td>Iterative / bulk</td>
<td>Iterative</td>
</tr>
<tr>
<td>Similarity metric</td>
<td>Hamming/ Jaccard /Cosine</td>
<td>Hamming</td>
</tr>
<tr>
<td>Data distribution</td>
<td>Random/nonrandom</td>
<td>Nonrandom</td>
</tr>
</tbody>
</table>
Search Cost vs. N
Search Time vs. N

![Graph showing search time versus number of Bloom Filters Indexed](image)
Maintenance Cost vs. N
Search Cost vs. $d$
Search Cost vs. Filter Size
Search Time vs. Filter Size
Outline

- Problem
- Bloofii
  - Description
  - Search
  - Maintenance
- Distributed Data Provenance
- Experimental results
- Conclusions and future work
Related Work

- Bloom filters [Bloom70]
- Bloom filter variants [CM03, DR06, DBN12, Mitzenmacher01, SLP10]
- Bloom filter applications [FCB98, M90, BM02]
- B+trees
- Bitmap indexes [CI98]
- S-trees [Deppisch86]
Conclusions

- Bloofi – a hierarchical index structure for Bloom filters
  - Low search cost (O(N) worst case, O(logN) most cases)
  - Efficient construction and maintenance
  - Low storage cost
- Applications to cloud intelligence
Future work

- Clustering Bloom filters as a routing problem
- Compression
- Consider different Bloom filter sizes at different levels
Thank You!

Questions?