Invertible Bloom Lookup Tables

Joe Simons
presenting paper by
Goodrich and Mitzenmacher
Bloom Filter

- Key: $x$
- Hash Functions: $h_1(x), h_2(x), h_3(x), \ldots, h_k(x)$
- Bit Array: $m$ bits, each bit is 1 or 0
Bloom Filter

- Compact space – m bits
- Insertion only – No deletions
- Query
  - Possibility of false positives:
Counting Bloom Filter

Insert

key

x

Hash Functions

h1(x)
h2(x)
h3(x)

...hk(x)

Count Array of m cells

count++
count++
count++
count++
count++
count++
count++
count++
Counting Bloom Filter

Delete

key

Hash Functions

Count Array

$x$

$h_1(x)$

$h_2(x)$

$h_3(x)$

$\cdots$

$h_k(x)$

$count--$

$count--$

$count--$

$count--$

$count--$

$count--$
Counting Bloom Filter

• Insert/Delete:
  – update $k$ counters

• Query:
  – are the $k$ locations non-zero?

• Space Use:
  – 4 bits per counter is often enough
Bloom Filter Tradeoff
Space savings vs. Risk of false positives

Applications:
• White List
  – Early spell checkers when space was expensive
• Black List
  – Dictionary of insecure passwords
• Differential Files
  – Changes stored in differential file.
  – Bloom filter to check if record has changed.
Bloom Filter

• Pro:
  – Space-efficient
  – Simple

• Con:
  – False positives
  – Can’t list elements
Invertible Bloom Lookup Table

• Stores (key, value) pairs
• Supports:
  – Insert(x,y)  
  – Delete(x,y)  
  – Get(x)  
  – ListEntrees()  

Always succeeds

Small chance of error
IBLT

- $n = \# \text{ key, value pairs}$
- $t = \text{threshold \# of keys}$
- If $n \gg m$,
  - Insert and Delete still succeed
  - Space used is still $O(m)$
- If $n < t$,
  - Queries have good probability of success
IBLT

Table $T$ of $m$ cells

<table>
<thead>
<tr>
<th>count</th>
<th>keySum</th>
<th>valueSum</th>
</tr>
</thead>
</table>

3 fields per cell
1 machine word per field
IBLT

- $k$ hash functions like standard Bloom Filter
- Assume for a single key $x$, the hash functions $h_i(x)$, $i = 1...k$ give distinct locations

- Standard approach:
  - Split T into $k$ sub-tables of size $m/k$
Insert(x,y)

for each hash $h_i(x)$

$T[h_i(x)].\ count\ ++$

$T[h_i(x)].\ keySum\ +=\ x$

$T[h_i(x)].\ valueSum\ +=\ y$
Delete(x, y)

for each hash $h_i(x)$

$T[h_i(x)].\text{count} -= 1$

$T[h_i(x)].\text{keySum} -= x$

$T[h_i(x)].\text{valueSum} -= y$
Get(x)

for each hash $h_i(x)$
    if count == 0, return null
else if count == 1,
    if keySum == $x$,
        return valueSum
    else return null
return “not found”
Prob. of error for $x$ in IBLT

- Prob. that a key is not hashed to that cell:
  $$\left(1 - \frac{k}{m}\right)$$

- Prob. that $(n-1)$ keys are not hashed there:
  $$\left(1 - \frac{k}{m}\right)^{(n-1)} \approx e^{-kn/m}$$

- Prob. that get($x$) returns “not found”:
  $$\left(1 - e^{-kn/m}\right)^k$$
ListEntries()

while $\exists \ i \in [1,m] \text{ with } T[i].\text{count} == 1$
  
  output $(T[i].\text{keySum}, \ T[i].\text{valueSum})$

Delete$(T[i].\text{keySum}, \ T[i].\text{valueSum})$

Implement with Linked-List based priority queue
modify Delete$(x,y)$ so that it updates this queue
ListEntries()

• Succeeded if all cells of T are empty
• Failed if some cells still have count > 1
ListEntries()

• Same as standard “peeling process” for finding 2-core of a hypergraph

• Hypergraph = generalization of graph
  – edges are sets of vertices

• 2-core = largest sub-hypergraph with minimum degree at least 2
peeling process

- vertex = cell in table
- edge = set of cells that $x$ gets hashed to
- degree = # of incident edges = cell.count

iteratively remove vertices with degree 1.
Theorem 1

If $m > (c_k + \epsilon)n$, ListEntries() fails with prob. $O(n^{-k+2})$

where $\epsilon$ is a small constant $> 0$
and $c_k$ is a small constant $> 1$

Based on results for 2-cores of random hypergraphs
Example Thresholds:

<table>
<thead>
<tr>
<th>k</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_k$</td>
<td>1.222</td>
<td>1.295</td>
<td>1.425</td>
<td>1.570</td>
<td>1.721</td>
</tr>
</tbody>
</table>
Adding Fault Tolerance

Protect against extraneous deletions

• Add extra hash function $G: x \rightarrow [1,R]$
• Add hashKeySum field
• On insert or delete, add or subtract $G(x)$ from $T[h(x)].hashKeySum$
• allows extra check to detect extraneous deletions
Fault Tolerance Example

From the same cell:
Insert(x1)
Insert(x2)
Delete(x3)

Now cell erroneously has count == 1
Fault Tolerance Example

**Get(x)**

- check if $G(x) == \text{cell.hashKeySum}$
- Only happens if $G(x) = G(x1) + G(x2) - G(x3)$

- Probability is at most $1/R$
- Choose $R$ large enough that this never happens
- $O(\log n)$ bits for lifetimes polynomial in $n$
Adding Fault Tolerance

Protecting against multiple values

- Add extra hash function G2
- Add hashValueSum field
- On insert or delete, add or subtract $G_2(x)$ from $T[h(x)].hashValueSum$
- allows extra check for multiple values mapped to the same key
Fault Tolerance Example

• $k = 5$, $m/n = 8$, $d = 1/10$
  
  $d$ is the fraction of keys with duplicate values

• Probability that a valid key is unrecoverable:
  
  $8.6 \cdot 10^{-7}$
1) 10,000 keys, 80,000 cells, 1,000 duplicates
2) “                                         “, 2,000 duplicates
3) 100,000 keys, 800,000 cells, 10,000 duplicates

<table>
<thead>
<tr>
<th></th>
<th>Unrecovered Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Experiment 1</td>
<td>99.360</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>83.505</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>92.800</td>
</tr>
</tbody>
</table>
Applications

Database Reconciliation

• Bob and Alice want to sync databases.
  – Alice inserts into IBLT, sends it to Bob
  – Bob deletes his checksums from IBLT

• Insertion w/o deletion = Alice’s extra records

• Deletion w/o insertion = Bob’s extra records

• Size of message: only $O(t)$, the # of differences
Applications

Tracking Network Acknowledgements

• Router tracks TCP sessions in IBLT
• Supports fast insert, delete, and listing flows
• Can handle spike in # of flows, list when normal load resumes
• Can detect missing insertions or deletions
  – when flow wasn’t initialized or terminated properly