Concurrent Expandable AMQs on the Basis of Quotient Filters

Conference Presentation · June 17, 2020
Tobias Maier, Peter Sanders, Robert Williger

Institute of Theoretical Informatics · Algorithmics Group
Approximate Membership Query – Data Structures

- (approximate) **set representation**
  - insert
  - contains

  - may have **false positives** (fp rate $p^+$)
  - **no** false negatives

- constructed with capacity and false positive rate
Approximate Membership Query – Data Structures

- (approximate) **set representation**
  - insert
  - contains

  - may have **false positives** (fp rate $p^+$)
  - no false negatives

  - constructed with capacity and false positive rate
Approximate Membership Query – Data Structures

- (approximate) set representation
- insert
- contains

- may have **false positives** (fp rate $p^+)$
- **no** false negatives

- constructed with capacity and false positive rate

\[
p^+ = 1 - \frac{1}{1 - p^+}
\]
Quotient Filter

\[ f(e) = 1101011110 \]

Fingerprint quotient remainder
Quotient Filter

\[ f(e) = 1101011110 \]

Fingerprint

quotient

remainder

= Position

hash-table-like data structure:

similar to hashing with linear probing

BUT sorted by fingerprint
**Quotient Filter**

- \( e \) is hashed to \( f(e) = 1101011110 \)
- Quotient and remainder, \( = \) Position

- **run**: elements hashed to same canonical slot
- **cluster**: run at canonical slot + shifted runs behind
- **supercluster**: consecutive filled slots
Quotient Filter

\[ f(e) = 1101011110 \]

status bits:

\[
\begin{array}{cccccccccccccc}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

has a run
run start
cluster start
Quotient Filter

\[ f(e) = 1101011110 \]

Fingerprint

quotient remainder

status bits:

\[
\begin{align*}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1
\end{align*}
\]

has a run
run start
cluster start
Quotient Filter

\[ f(e) = 1101011110 \]

Fingerprint

quotient remainder

status bits:

```
1 1 1 0 0 0 1 1 1 0 0 1 0
0 1 1 0 1 0 1 0 0 0 0 1
0 1 1 1 1 1 0 1 1 0 0 1
```

has a run

run start

cluster start
### Quotient Filter – Insertion

![Insertion Diagram]

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

1. The diagram shows the insertion process into a Quotient Filter.
2. The characters `a`, `b`, `f`, `g`, and `j` are being inserted.
3. Each character is represented by a block with a corresponding bit sequence below it.
4. The process involves finding the correct position and updating the filter's state.
Quotient Filter – Insertion

1 1 0 0 0 1 1 0 0 1 0
0 1 1 0 1 0 1 0 0 0 1
0 1 1 1 1 1 0 1 1 0 1

look for cluster start

1
0
0
Quotient Filter – Insertion

- Look for cluster start
- Count runs before canonical slot
- Count run starts
Quotient Filter – Insertion

- Look for cluster start
- Count runs before canonical slot
- Count run starts
- Correct position
Quotient Filter – Insertion

- Store in correct position
- Shift elements

```
1 1 0 0 0 1 1 0 0 1 0
0 1 1 0 1 0 1 0 0 0 1
0 1 1 1 1 0 1 1 0 0 1
```

```c
1 1 0 0 0 1 1 0 0 1 0
0 1 1 0 1 0 1 0 0 0 1
0 1 1 1 1 0 1 1 0 0 1
```
Quotient Filter – Insertion

status bits of changed slots need to be updated
Quotient Filter – Growing

- limited growing

\[ f(e) = 1101011110 \]

Fingerprint

quotient remainder

\[ q \text{ bits quotient} \quad r \text{ bits remainder} \]
Quotient Filter – Growing

- **limited growing**

- **Fingerprint**

- **quotient** remainder

- $q$ bits quotient  $r$ bits remainder

- $q' = q + 1$  $r' = r - 1$
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)

- growing exponentially
- therefore logarithmic number of tables
- exponentially smaller fp rates
- overall bounded fp rate
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Growing

- limited growing
- unlimited growing (scalable quotient filter)
Quotient Filter – Trivial Locking

1. lock slot range
2. operate exclusively
3. unlock

sometimes multiple locks necessary
Quotient Filter – Trivial Locking

1. lock slot range
2. operate exclusively
3. unlock

properties

+ simple
+ growing is possible
- locking
- memory overhead
- add. lookup per operation

sometimes multiple locks necessary
Quotient Filter – Lockfree-ness seems hard
Quotient Filter – Lockfree-ness seems hard

thrd-1 find x
Quotient Filter – Lockfree-ness seems hard

thrd-1 find x

x must be in 2nd run
Quotient Filter – Lockfree-ness seems hard

thrd-1 find x

x must be in 2nd run

thrd-2 insert y
Quotient Filter – Lockfree-ness seems hard

thrd-1 find x
x must be in 2nd run

thrd-2 insert y
Quotient Filter – Lockfree-ness seems hard

thrd-1 find $x$

$x$ must be in 2nd run

thrd-2 insert $y$

2nd run

$x \neq y$

false negative
Linear Probing Quotient Filter

- no run reconstruction ➔ no sorted order
- linear probing like insertion
- more comparisons = more false positives?
  \[ E[fp \ rate] = \frac{1}{2} \left( 1 + \frac{1}{1 - \frac{n}{m}} \right) \cdot \frac{1}{2^{r+3} - 1} \]
- counteract that with 3 more fingerprint bits (same slot size)

without status bits  assuming no remainder = 0
Linear Probing Quotient Filter

without status bits assuming no remainder = 0

- no run reconstruction ➔ no sorted order
- linear probing like insertion
- more comparisons = more false positives?

\[ E[fp\ rate] = \frac{1}{2} \left( 1 + \frac{1}{1 - \frac{n}{m}} \right) \cdot \frac{1}{2^{r+3} - 1} \]

- counteract that with 3 more fingerprint bits (same slot size)
Linear Probing Quotient Filter

- a b a a b f g f c j j

without status bits assuming no remainder = 0

- no run reconstruction ➔ no sorted order
- linear probing like insertion
- more comparisons = more false positives?

\[ E[fp \ rate] = \frac{1}{2} \left( 1 + \frac{1}{1 - \frac{n}{m}} \right) \cdot \frac{1}{2^{r+3} - 1} \]

- counteract that with 3 more fingerprint bits (same slot size)
Linear Probing Quotient Filter

properties

- super simple
- lockfree (using CAS)
- fast
- no growing, no deletion
- worse over 70% fill ratio
- counteract that with 3 more fingerprint bits (same slot size)

without status bits assuming no remainder = 0
no run reconstruction no sorted order
linear probing like insertion
more comparisons = more false positives?

\[ E[fp\ rate] = \frac{1}{2} \left( 1 + \frac{n}{m} \right) \cdot \frac{2^{r+3} - 1}{2^r + 3 - 1} \]
Locally Locking Quotient Filter – Using Status Bits

\[\begin{array}{cccccccccccc}
a & a & a & b & b & f & f & g & j & j \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array}\]

1 0
1 and 1 are impossible status-bit-combinations
0 0 i.e., new cluster but no new run

We use them as \textit{read-} and \textit{write-lock}
Locally Locking Quotient Filter – Using Status Bits

1. write-lock next empty slot
   reserves the supercluster

1 0
1 and 1 are impossible status-bit-combinations
0 0 i.e., new cluster but no new run

⇒ We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

1. Write-lock next empty slot reserves the supercluster

1 and 1 are impossible status-bit-combinations
0 and 0 i.e., new cluster but no new run

⇒ We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

2. read-lock cluster start

<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>f</th>
<th>f</th>
<th>g</th>
<th>j</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1 and 1 are impossible status-bit-combinations

0 0 i.e., new cluster but no new run

We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

We use them as read- and write-lock

1 0
1 1 are impossible status-bit-combinations
0 0 i.e., new cluster but no new run

We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

3. execute operation

1 0
1 1 and 1 are impossible status-bit-combinations
0 0 i.e., new cluster but no new run

=> We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

\[\begin{array}{ccccccccccc}
\text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{f} & \text{f} & \text{g} & \text{j} & \text{j} \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
\end{array}\]

1 and 1 are impossible status-bit-combinations
0 and 0 i.e., new cluster but no new run

We use them as read- and write-lock
Locally Locking Quotient Filter – Using Status Bits

4. release both locks

1 0
1 and 1 are impossible status-bit-combinations
0 0 i.e., new cluster but no new run

➡️ We use them as *read-* and *write-lock*
Locally Locking Quotient Filter – Using Status Bits

We use them as read- and write-lock.
Experimental Results

![Graph showing performance (MOPS) vs. fill degree (%) for different filters: LP Q-Filter, loc. locked Q-Filter, ext. locked Q-Filter, Bloom Filter, and counting Q-Filter. The graph is executed with 80 threads on a 4-socket Intel Xeon Gold 6138 (20 cores per socket).]
Experimental Results

![Graph showing performance (MOPS) vs. fill degree (%)]

successful contains

- LP Q-Filter
- loc. locked Q-Filter
- ext. locked Q-Filter
- Bloom Filter
- counting Q-Filter

executed with 80 threads
4-socket Intel Xeon Gold 6138 (20 cores per socket)
Experimental Results

![Graph showing performance and fill degree for different filters.]

- LP Q-Filter
- loc. locked Q-Filter
- ext. locked Q-Filter
- Bloom Filter
- counting Q-Filter
Conclusion

- trivial (external) locking is not enough
- **local locking** with inherent status bits
  previously unused combinations
- lock-free **linear probing quotient filter**
  more bits counteract more comparisons
- growing implementations for status bit variants
- unlimited growing
  combines migration and multi table