Filters

Michael A. Bender

This is the advertised talk title.
Time To Change Your Filter

Michael A. Bender

This title is more to the point. (more explanation later).

Now... what's a filter.
A dictionary maintains a set $S$ from universe $U$.

A dictionary supports membership queries on $S$.

member($a$): ✔
member($b$): ✗
member($c$): ✔
member($d$): ✗
A filter is an *approximate* dictionary.

A filter supports *approximate* membership queries on $S$.

- $\text{member}(a)$: ✔️
- $\text{member}(b)$: ✗
- $\text{member}(c)$: ✔️
- $\text{member}(d)$: ✔️ ✗ false positive

$S \subseteq U$
A Filter Guarantees a False-Positive Rate $\varepsilon$

if $q \in S$, return ✔️ with probability 1 (true positive)

if $q \notin S$, return

- ✗ with probability $> 1 - \varepsilon$ (true negative)
- ✔️ with probability $\leq \varepsilon$ (false positive)

one-sided errors
False-positive rate enables filters to be compact

\[
\text{space} \geq n \log(1/\varepsilon)
\]

\[
\text{space} = \Omega(n \log |U|)
\]
Talk So Far

• Filter data structure
• Next: the **Bloom filter** [Bloom ’70]
Bloom filter: a bit array + $k$ hash functions.  

(Here $k=2$.)

\[ \begin{array}{cccccccc} 
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 
\end{array} \]
Bloom filter: a bit array + $k$ hash functions.

(Here $k=2$.)
Bloom filter: a bit array + $k$ hash functions. (Here $k=2$.)
Bloom filter: a bit array + $k$ hash functions. 

(Here $k=2$.)
Bloom filter: a bit array + $k$ hash functions. (Here $k=2$.)

$$h_1(b) = 2$$
$$h_2(b) = 5$$
Bloom filter: a bit array + k hash functions. (Here k=2.)
Bloom filter: a bit array + $k$ hash functions. (Here $k=2$.)

Bloom filters don’t support delete.

Issue: on a delete, which 1s get decremented?
Bloom filter space with false-positive rate $\varepsilon$: 
$\approx 1.44 \log (1/\varepsilon)$ bits/element.

Example:
For $\varepsilon = 2\%$, 
bits/element $\approx 8$.

Common rule of thumb: 
Bloom filters take about 1 byte/element.
Bloom filters are ubiquitous

- Computational biology
- Databases
- Networking
- Storage systems
- Streaming applications

≥ 4300 citations
Talk So Far

- Filter data structure
- The Bloom filter [Bloom ’70]
- How filters are used
Most Common Filter Use

Filter out queries to a large remote dictionary.

Only an $\varepsilon$-fraction of negative queries don’t get filtered out.

Filter
local, e.g., in RAM

Dictionary
remote, e.g., on disk
Speedup from Filter Use

Workload has $P$ positive and $N$ negative queries.

<table>
<thead>
<tr>
<th>Dictionaries w/o Filters</th>
<th>Dictionaries w/ Filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P+N$</td>
<td>$P+\varepsilon N$</td>
</tr>
</tbody>
</table>

Remote Accesses of Dictionary
Log-structured merge tree (LSM)

• An LSM tree supports fast inserts by partitioning into independent dictionaries.

[O'Neil, Cheng, Gawlick, O'Neil '96]

Point queries are slow without filters.
Talk So Far

- Filter data structure
- The Bloom filter [Bloom ’70]
- How filters are used
- **Time to change your filter**
  (the talk title)
Application must work around limited Bloom filter capabilities

<table>
<thead>
<tr>
<th>Limitations</th>
<th>Work-arounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>No deletes</td>
<td>Rebuild</td>
</tr>
<tr>
<td>No resizes</td>
<td>Guess $N$, and rebuild if wrong</td>
</tr>
<tr>
<td>No filter merging nor enumeration of elements</td>
<td>???</td>
</tr>
<tr>
<td>No values associated with keys</td>
<td>Combine with other data structure</td>
</tr>
</tbody>
</table>

Bloom filter limitations increase system complexity, waste space, and slow down application performance.
Bloom filters also have suboptimal guarantees. Bloom filter limitations increase system complexity, waste space, and slow down application performance.

<table>
<thead>
<tr>
<th></th>
<th>Bloom filter</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td>$\approx 1.44 \cdot n \cdot \lg(1/\varepsilon)$</td>
<td>$\approx n \cdot \lg(1/\varepsilon) + O(1)$</td>
</tr>
<tr>
<td><strong>CPU cost</strong></td>
<td>$\Omega(\lg(1/\varepsilon))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Data locality</strong></td>
<td>$\Omega(\lg(1/\varepsilon))$ probes</td>
<td>$O(1)$ probes</td>
</tr>
</tbody>
</table>
Tons of Research on Extending/Improving/Replacing Bloom Filters

Deletes + counting
[Bonomi, Mitzenmacher, Panigrahy, Singh, Varghese 06],
[Yuan, Miao, Jia, Wang 08],
[Pandey, Bender, Patro, Johnson SIGMOD 17],

Keys
[Chazelle, Kilian, Rubinfeld, Tal 04]

Filters on SSD
[Canim, Mihaila, Bhattacharhee, Lang, Ross 10],
[Debnath, Sengupta, Lilja, Du 11], [Lu, Debnath, Du, 11], [Bender, Farach-Colton, Johnson, Kraner, Kuszmaul, Medjedovic, Montes, Shetty, Spillane, Zadok 12],
[Pandey, Singh, Bender, Berry, Farach-Colton, Johnson, Kroeger, Phillips 20]

Optimizing asymptotics
[Pagh, Pagh, Rao 05],
[Arbitman, Naor, Segev 10],
[Lovett & Porat 10],
[Bender, Farach-Colton, Goswami, Johnson, McCauley, Singh 18]

Engineering
[Bender, Farach-Colton, Johnson, Kraner, Kuszmaul, Medjedovic, Montes, Shetty, Spillane, Zadok 12],
[Fan, Andersen, Kaminsky, Mitzenmacher 14],
[Pandey, Bender, Patro, Johnson SIGMOD 17],
[Pandey, Bender, Johnson, Patro 17],
[Breslow, Jayasena 18],
[Pandey, Conway, Durie, Bender, Martin, Johnson 21]

Adaptivity
[Mitzenmacher, Pontarelli, Reviriego 18]
[Bender, Farach-Colton, Goswami, Johnson, McCauley, Singh 18]
[Bender, Das, Farach-Colton, Mo, Wang 21]

When too much stuff is growing on your filter, it’s time to change the filter.
Talk So Far

- Filter data structure
- The Bloom filter [Bloom ’70]
- How filters are used
- **Time to change your filter**
  (the talk title)
I couldn't find the right talk title.

Then I received this email.
I couldn't find the right talk title.

Then I received this email.
<table>
<thead>
<tr>
<th>From</th>
<th>Subject</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-reply</td>
<td>Product Reminder</td>
<td>Sep 17</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>Sep 14</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>Aug 15</td>
</tr>
<tr>
<td>Perform Better</td>
<td>Don’t Miss This Week’s FREE Webinars! - If you have trou...</td>
<td>Aug 23</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>Aug 1</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>Jul 11</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>May 29</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>At home? You’ll love the Home Filter Club - From your #1 ...</td>
<td>May 7</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>Apr 15</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>Mar 24</td>
</tr>
<tr>
<td>no-reply</td>
<td>Product Reminder</td>
<td>Mar 17</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>Mar 2</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>Feb 9</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>Jan 18</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>11/13/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>10/23/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>10/1/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>It’s Time to Order Your Filters - Discount for Today Only! -</td>
<td>9/17/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>9/9/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>8/19/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>7/28/19</td>
</tr>
<tr>
<td>FiltersFast.com</td>
<td>Have you been forgetting to change your filters? - The Ho...</td>
<td>7/6/19</td>
</tr>
</tbody>
</table>
And these emails.
This Talk: It’s Time to Change Your Filter

- Tutorial-like introduction to filters.
- General techniques for solving filter problems.

fingerprinting  quotienting  collision resolution
This Talk: It’s Time to Change Your Filter

- Tutorial-like introduction to filters.
- General techniques for solving filter problems.

fingerprinting  quotienting  collusion collision resolution
This Talk

• Tutorial-like introduction to filters.
• General techniques for solving filter problems.

fingerprinting
quotienting
collision resolution
Fingerprinting

Filter $F$ of set $S$: $\{h(x) \mid x \in S\}$

$F = \{h(x_1), h(x_2), \ldots, h(x_n)\}$

$S = \{x_1, x_2, \ldots, x_n\}$

$F$ can be stored more compactly than a Bloom filter.
Filter $F$ of set $S$:

$$\{ h(x) \mid x \in S \}$$

$$F = \{ h(x_1), h(x_2), \ldots, h(x_n) \}$$

$$S = \{ x_1, x_2, \ldots, x_n \}$$

$F$ can be stored more compactly than a Bloom filter.

$F$ is also just dictionary—just a more compact one.
Filter $F$ of set $S$:

$\{ h(x) \mid x \in S \}$

$$F = \{ h(x_1), h(x_2), \ldots, h(x_n) \}$$

$$S = \{ x_1, x_2, \ldots, x_n \}$$

$\text{member}(x) = \begin{cases} 
\checkmark & \text{if } h(x) \in F \\
\times & \text{if } h(x) \notin F
\end{cases}$
Filter $F$ of set $S$:

\[ F = \{ h(x_1), h(x_2), \ldots, h(x_n) \} \]

\[ S = \{ x_1, x_2, \ldots, x_n \} \]

$F$ can be stored more compactly than a Bloom filter.
False-Positive Analysis for Fingerprinting

\[ F = \{ h(x_1), h(x_2), \ldots, h(x_n) \} \]

\[ S = \{ x_1, x_2, \ldots, x_n \} \]

\[
\text{member}(y) = \checkmark \quad \text{if } h(y) \in F.
\]

\[
y \text{ is a false positive if } \exists x_i \ h(y) = h(x_i), \text{ but } y \notin S.
\]

**Fingerprint collisions:** only source of false positives.
False-Positive Analysis for Fingerprinting

\[ x \rightarrow h(x) \]

\[ \geq \log(|U|) \text{ bits} \rightarrow \log(n/\epsilon) \text{ bits} \]

\[ \Pr[x \text{ and } y \text{ collide}] = \frac{1}{2^{\log(n/\epsilon)}} = \frac{\epsilon}{n} \]
False-Positive Analysis for Fingerprinting

- $x$ \rightarrow h(x)
  - $\geq \log(|U|)$ bits
  - $\log(n/\epsilon)$ bits

$Pr[x \text{ and } y \text{ collide}] = \frac{1}{2^{\log(n/\epsilon)}} = \frac{\epsilon}{n}$

$Pr[y \notin S \text{ is a false positive}] \leq \epsilon$
False-Positive Analysis for Fingerprinting

\[ h(x) \]

\[ \geq \log(|U|) \text{ bits} \]

\[ \log(n/\epsilon) \text{ bits} \]

\[ \Pr[x \text{ and } y \text{ collide}] = \frac{1}{2^{\log(n/\epsilon)}} = \frac{\epsilon}{n} \]

\[ \Pr[y \not\in S \text{ is a false positive}] \leq \epsilon \]

\[ n \text{ fingerprints can be stored compactly:} \]

\[ \log(1/\epsilon) + O(1) \text{ bits/element} \]
False-Positive Analysis for Fingerprinting

\[ h(x) \]

\[ \geq \log(|U|) \text{ bits} \]

\[ \log(n/\varepsilon) \text{ bits} \]

\[ \Pr[y \notin S \text{ is a false positive}] \leq \varepsilon \]

\[ \Pr[x \text{ and } y \text{ collide}] = \frac{1}{2^{\log(n/\varepsilon)}} = \frac{\varepsilon}{n} \]

\[ n \text{ fingerprints can be stored compactly:} \]

\[ \log(1/\varepsilon) + O(1) \text{ bits/element} \]
Compact Storage Using Quotienting [Knuth]

**Space:** $O(\log (1/\varepsilon))$ bits per element

$x \rightarrow h(x) = q(x) \quad r(x)$

$q(x) = \text{location in hash table}$

$r(x) = \text{data stored in hash table}$
Compact Storage Using Quotienting

**Space:** $O(\log (1/\varepsilon))$ bits per element

$x \rightarrow h(x) = q(x) + r(x)$

$q(x) = \text{location in hash table}$

$r(x) = \text{data stored in hash table}$

**How to deal with collisions in the hash table?**

*Isn’t this a solved problem?*

*What’s wrong with out-of-the-box linear probing?*
Hash Collisions in Quotienting

Ex: 6 bit hash. 3 bits for address, 3 for data.

The hash is stored implicitly based on location. So how can we change its location?
Hash Collisions in Quotienting

Ex: 6 bit hash. 3 bits for address, 3 for data.

The hash is stored implicitly based on location. So how can we change its location?
# Talk Structure

- Filters + how filters are used + Bloom limitations

## Quotient filters variants
- [Bender, Farach-Colton, Johnson, Craner, Kuszmaul, Medjedovic, Montes, Shetty, Spillane, Zadok 12]
- [Pandey, Bender, Johnson, Patro 17]
- [Pandey, Bender, Conway, Farach-Colton, Johnson 21]

## Cuckoo filter variants
- [Fan, Andersen, Kaminsky, Mitzenmacher 14]
- [Breslow, Jayasena 18]

## Miscellaneous
- [Pagh, Pagh, Rao 05]
- [Arbitman, Naor, Segev 10]
- [Bender, Farach-Colton, Goswami, Johnson, McCauley, Singh 18]

<table>
<thead>
<tr>
<th>Quotient filters variants</th>
<th>Cuckoo filter variants</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bender, Farach-Colton, Johnson, Craner, Kuszmaul, Medjedovic, Montes, Shetty, Spillane, Zadok 12], [Pandey, Bender, Johnson, Patro 17], [Pandey, Bender, Conway, Farach-Colton, Johnson 21]</td>
<td>[Fan, Andersen, Kaminsky, Mitzenmacher 14], [Breslow, Jayasena 18]</td>
<td>[Pagh, Pagh, Rao 05], [Arbitman, Naor, Segev 10], [Bender, Farach-Colton, Goswami, Johnson, McCauley, Singh 18]</td>
</tr>
</tbody>
</table>

- uses linear proving and Robinhood hashing
  - [Celis, Larson, Munro 85]
- Uses Cuckoo hashing
  - [Pagh, Rodler 01]
- Balls and bins + more advanced hashing
2 metadata bits per slot let us recover original location.

000 0 001 001 001 0
001 1 010 111 111 1
010 1 010 010 1 0
011 0 111 111 1 0
100 0 010 010 0 0
101 0 001 001 0 0
110 1 011 011 1 1
111 0

1 = “something is hashed here” 1 = “I’m hashed to the same place as the element before me”
Quotient Filters

2 metadata bits per slot let us recover original location.

Recall: no element is stored before its target position.

- 000: 0
- 001: 1
- 010: 1
- 011: 0
- 100: 0
- 101: 0
- 110: 1
- 111: 0

1 = “something is hashed here”
1 = “I’m hashed to the same place as the element before me”
Quotient Filter Capabilities

$\log n \quad 1/\varepsilon$

Bloom limitations

- No deletes or counting
- No resizes
- No element enumeration or merging of filters
- Keys have no values

QF Capabilities:
- Counting and deletes
- Resizing
- Element enumeration + filter merging
- Values allowed
Quotient Filter Capabilities

\[ k \xrightarrow{h(k)} q(k) \oplus r(k) = q(k) \oplus r(k) = \log n - \frac{1}{\epsilon} \]

-Bloom performance

- Space: \( \approx 1.44n \lg(1/\epsilon) \)
- CPU cost: \( \Omega(\lg(1/\epsilon)) \)
- Data locality: \( \Omega(\lg(1/\epsilon)) \) probes

QF performance

\( (1+\text{tiny})n \approx 1.44n(\lg(1/\epsilon) + 2) \)

O(1) expected

1 probe + scan

QF has practical + theoretical performance advantages.
• **Fingerprinting:** key \( k \in S \rightarrow h(k) \in F \).
  - False-positives only come from fingerprint collisions.

• **Quotienting:** Store fingerprints in a hash table implicitly.
  - \( \lg n \) bits of fingerprint depend on hash location.

• **Collision resolution:**
  - E.g., using linear probing/Robinhood hashing.
  - Use metadata bits to recover each \( h(k) \).

• **Designs alternatives:** cuckoo, Morton, broom
  - Use a different hash table but the same general approach
Cuckoo hash table has 2 hash functions $h_1$ and $h_2$.

Each hash bucket has 4 slots.
Cuckoo hash table has 2 hash functions $h_1$ and $h_2$. Each hash bucket has 4 slots.

If there’s no space in any of the 8 slots: **kick out** an element, and move it to the alternative location (which may cause other kicks).
Cuckoo hash table has 2 hash functions \( h_1 \) and \( h_2 \).

Each hash bucket has 4 slots.

If there’s no space in any of the 8 slots: **kick out** an element, and move it to the alternative location (which may cause other kicks).
Obstacles for Cuckoo Filters

Q: If \( f(x) \) is kicked, how to find an alternative location when we don’t store \( x \)?

A: We give up on independent hash functions. The alternate location only depends on \( r(x) \).

We also give up on asymptotic correctness.

Amazingly, it works for (practical) \( n \) not too large.

Cuckoo hashing seemingly doesn’t have metadata bits.

But because there are 4 slots per cell, 2 more fingerprint bits are stored explicitly.

[Fan, Andersen, Kaminsky, Mitzenmacher 14] [Breslow, Jayasena 18] [Pagh, Rodler 01]
<table>
<thead>
<tr>
<th>Quotient filter</th>
<th>Cuckoo filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>good space</td>
<td>good space</td>
</tr>
<tr>
<td>very good locality</td>
<td>ok locality</td>
</tr>
<tr>
<td>some degradation at high load factors</td>
<td>degradation at high load factors</td>
</tr>
<tr>
<td>good searches</td>
<td>very good searches</td>
</tr>
<tr>
<td>fast inserts</td>
<td>fast inserts</td>
</tr>
<tr>
<td>supports counting, multisets, values, deletes</td>
<td></td>
</tr>
<tr>
<td>complicated code</td>
<td>code is easier</td>
</tr>
<tr>
<td>good for all $n$.</td>
<td>pre-asymptotic guarantees. fails w.h.p. large enough $n$</td>
</tr>
</tbody>
</table>
Theorem: There is an optimal filter with

- **Space:** \( (1 + o(1)) \, n \log(1/\varepsilon) + O(n) \)
- **Error rate:** \( \leq \varepsilon \)
- **Operations:** \( O(1) \) insert, delete, query.

- [Pagh, Pagh, Rao 05]
- [Arbitman, Naor, and Segev 10]
- [Lovett & Porat 10]
- [Bender, Farach-Colton, Goswami, Johnson, McCauley, Singh 18] adaptive & worst case
Empirical Performance of a Vector Quotient Filter

Vector Quotient Filters: Overcoming the Time/Space Trade-O

(a) Insertion (Higher is better.)

(b) Deletion throughput (Higher is better.)

(c) Successful lookup (Higher is better.)

(d) Random lookup (Higher is better.)

Figure 4: Insertion, deletion, and lookup performance of different filters in RAM. VQF (no sc) means that no shortcuts during insertion. VQF (insert sc) means shortcut during insertion. Note that in Figure 4(d), the line for VQF (no sc) is hidden behind the line for VQF (insert sc), as their performance was almost identical. VQF (insert sc) and VQF (no sc) only show throughput up to 90% load factor because we can only fill the vector quotient filter to 93% capacity.

In order to isolate the performance differences between the data structures, we do not count the time required to generate the random inputs to the filters. Application workload. We also measure the performance of the data structures on workloads consisting of equal portions of insertions, removals, and lookups when the data structure is maintained at a high load factor (90%). This workload is characterized as a write heavy (WH) workload [23] because it involves inserting and removing items from the data structure when it is almost full. This type of workload is often seen in real-world applications and the performance of the data structure at a high load factor and under a write heavy workload is critical for applications to scale.

For the application workload, we first fill up all the data structures to 90% load factor. We then perform operations from a mixed workload and compute the aggregate throughput of the data structure to execute the set of operations.

The Morton filter supports a batch API for insertions and queries [13]. Nonetheless, we use the one-at-a-time API for two reasons. First, this makes an apples-to-apples comparison with the other filters. Second, many applications cannot use batching, and we want our benchmarks to reflect the performance that such applications would see.

7.2 In-RAM performance

Figure 4 shows the in-RAM performance of data structures. The vector quotient filter has the highest insertion throughput compared to other data structures. It is 2× and 2.5× faster than the Morton filter and cuckoo filter, respectively. Aggregate throughput of different operations are shown in Figure 6a.

The insertion throughput of the vector quotient filter with short-circuit optimization stays consistent across different load factors. With the short-circuit optimization, the insertion throughput is π1.25× higher until π75% load.

Quotient filter plus minimum of 2 choice retains good performance even when almost full.

[Pandey, Conway, Durie, Bender, Farach-Colton, Johnson 21]
Figure 4: Insertion, deletion, and lookup performance of different filters in RAM. VQF (no sc) means that no shortcuts during insertion. VQF (insert sc) means shortcut during insertion. Note that in Figure 4(d), the line for VQF (no sc) is hidden behind the line for VQF (insert sc), as their performance was almost identical. VQF (insert sc) and VQF (no sc) only show throughput up to 90% load factor because we can only fill the vector quotient filter to 93% capacity.

In order to isolate the performance differences between the data structures, we do not count the time required to generate the random inputs to the filters.

Application workload. We also measure the performance of the data structures on workloads consisting of equal portions of insertions, removals, and lookups when the data structure is maintained at a high load factor (90%). This workload is characterized as a write heavy (WH) workload [23] because it involves inserting and removing items from the data structure when it is almost full. This type of workload is often seen in real-world applications and the performance of the data structure at a high load factor and under a write heavy workload is critical for applications to scale.

For the application workload, we first fill up all the data structures to 90% load factor. We then perform operations from a mixed workload and compute the aggregate throughput of the data structure to execute the set of operations.

7.2 In-RAM performance

Figure 4 shows the in-RAM performance of data structures. The vector quotient filter has the highest insertion throughput compared to other data structures. It is $2 \times$ and $2.5 \times$ faster than the Morton filter and cuckoo filter, respectively. Aggregate throughput of different operations are shown in Figure 6a.

The insertion throughput of the vector quotient filter with short-circuit optimization stays consistent across different load factors. With the short-circuit optimization, the insertion throughput is $\approx 1.25$ times higher until $\approx 75\%$ load. Cuckoo still kicks butt on queries.

(c) Successful lookup (Higher is better.)

Cuckoo still kicks butt on queries.
It’s time to change your filter.

Applications should demand a richer set of operations from their filter.

Fingerprinting + quotienting + collision resolution is unifying theme in theory and practice.