Multidimensional Bloom Filters

Claude N. Warren, Jr

January 20, 2022

Email: claude.warren@instaclustr.com
Github: https://github.com/Claude-at-Instaclustr
LinkedIn: https://www.linkedin.com/in/claudewarren
Research Gate: https://www.researchgate.net/profile/Claude_Warren_Jr

Overview

Intro: What is a Bloom filter

- Is a probabilistic data structure defined by Burton Bloom in 1970 [2];
- Can be considered as a bit vector representing a set of objects;
- Is constructed by hashing data multiple times;
- Identifies where things are **not**;
- Will yield false positives but **never** false negatives.
- Is typically used where hash table solutions are too memory intensive and false positives can be addressed; for example a gating function to a longer operation (e.g. determine if a database lookup should be made);

Think about using a Bloom filter anytime you can frame the question as

$$A \notin B$$

Intro: How is it defined

Bloom filters are constrained by:

- p: the probability of false positives,
- n: the number of elements in the set represented by the Bloom filter,
- m: the magnitude or number of bits, and
- k: the number of hash functions.

Mitzenmacher and Upfal [8] have shown that the relationship between these properties is:

- $p = (1 - e^{-kn/m})^k$
- $n = \lceil m/(-k/\ln(1 - e^{\ln(p)/k})) \rceil$
- $m = \lceil n \ln(p)/\ln(1/2^{\ln(2)})) \rceil \approx \lceil n \ln(p)/ - 0.480453014 \rceil$
- $k = \lfloor m \ln(2)/n \rceil \approx \lfloor 0.693147181 m/n \rceil$

Thomas Hurst provides a good calculator [7] where the interplay between these values can be explored.

Intro: Example BloomFilter

Let's assume we want to put 3 items in a filter with a 1/5 probability of collision. The calculations yields a m of 11 and a k of 3.
CRC hash for "CAT" is FD2615C4 which can be interpreted as 2 unsigned ints

- FD26 = 64806
- 15C4 = 5572

Using the combinatorial hashing algorithm we generate the 3 values

Name	Calculation	Value	Value mod m
k1	5572	5572	6
k2	5572 + 64608	70180	0
k3	5572 + 64608 + 64608	134788	5

Table: Bit calculations for "CAT"

This yields a Bloom filter of $00001100001 \equiv \{0, 5, 6\}$

Intro: Example BloomFilter Pt2

Performing the same operations on "DOG" and "GUINEA PIG" yields:

Name	CRC	Bit Set	Bloom filter
CAT	FS26 15C4	$\{0, 5, 6\}$	00001100001
DOG	3560 D2EF	$\{2\}$	00000000100
GUINEA PIG	E58C A739	$\{2, 7, 10\}$	10010000100
COLLECTION		$\{0, 2, 5, 6, 7, 10\}$	10011100101

Table: Bloom filter examples

The collection is the union of the other three values. This represents the set of animals. To search for an animal, call this the target (T), generate its Bloom filter and execute $T \cap C == T$, where C is the collection of animal Bloom filters.
The Bit values for HORSE $\{2, 5, 9\}$. Testing

$$\{2, 5, 9\} \cap \{0, 2, 5, 6, 7, 10\} \Rightarrow \{2, 5\} \neq \{2, 5, 9\}$$

so HORSE is not in the collection.

Intro: Example BloomFilter Pt3

If we only put CAT and GUINEA PIG into the collection we get the same result for the collection. But when testing for DOG we get the *true* statement

$$\{2\} \cap \{0, 2, 5, 6, 7, 10\} \Rightarrow \{2\}$$

The filter says that DOG is in the collection. This is an example of a false positive result.

Bloom filters should only be used where the false positive can be filtered or otherwise accepted.

Intro: Statements about Bloom filters

- The number of bits turned on in a Bloom filter is called the cardinality or "hamming" value or weight.
- The number of bits that have to be modified to changed one Bloom filter into another is called the "hamming distance".
- If the Shape (m and k) is known then:
  - an estimate of the number of items in the filter can be calculated.
  - an estimate of the number of items in the union of two Bloom filters can be calculated.
  - an estimate of the number of items in the intersection of two Bloom filters can be calculated.
- Items can not be deleted from a standard Bloom filter, the filter must be reconstructed. Several specialized filters solve this problem.

Intro: n and Saturation

Saturation is a measure of the number of items in a filter relative to the number specified in the shape. The table below samples 10^6 filters with $n = 3$, p to 1 in 10^6, $m = 72$, and $k = 17$. The table shows the mean ($\bar{x}$), median ($\tilde{x}$), mode (Mo), and standard deviation (σ_x) for the Hamming weights as well as the population mean (μ) and probability of false positives (p) for each saturation level. [12]

Sat	$\bar{x}$	$\tilde{x}$	Mo	σ_x	μ	p
1	34	35	36	4.35889894354067	37	0.000010
2	52	52	53	4.24264068711929	55	0.008898
3	60	61	62	3.60555127546399	64	0.115070
4	65	66	66	2.64575131106459	68	0.356832
5	68	69	69	2	70	0.606726
6	70	70	71	1.4142135623731	71	0.783301
7	70	71	71	1.4142135623731	72	0.887062
8	71	72	72	0	72	0.942790
9	71	72	72	0	72	0.971430

Intro: n and Saturation pt2

The critical point is that doubling the saturation of the graph creates a probability of collision 2 orders of magnitude larger than the requested probability. So it is critical for performance that large enough values of n are chosen when the filters are created.

Example: A gatekeeper

A "Gatekeeper" is a Bloom filter that is an "index" of all items in a collection. It is used when searching the collection may be too expensive.

- Create a "Gatekeeper" bloom filter that is large enough to hold the number of items expected in a collection.
- Every time an item is placed in the collection, union its Bloom filter with the "Gatekeeper".
- Whenever an item is requested, check if its Bloom filter is in the "Gatekeeper", if it is, execute the "expensive" collection lookup.

Examples of this usage includes determining:

- if a file is on a storage device
- if a network node offers a specific service
- if a join table contains an item

Example: A gatekeeper p2

A "Gatekeeper" may be used for sharding data across collections. In this usage each of the collections has a gatekeeper as described previously. When an item is being inserted:

- Determine the hamming distance between the item being inserted and each collection.
- Insert the item in the collection with the smallest hamming distance.

This usage has some interesting properties:

- Additional collections can be added at any time by adding the collection with an empty filter.
- "full" collections can be removed from insert consideration but still be used for query.
- redundant storage can be achieved by selecting the n lowest hamming distances for insert.

Multidimensional: What it it

Any collection of Bloom filters is a multidimensional Bloom filter. A list of gatekeeper filters to quickly locate files, or to perform sharding in a database are examples.

The question for multidimensional Bloom filters is:

> *Is there an efficient implementation of a multidimensional Bloom filter where efficient is* $< O(n)$

If there is, then "Reference" Bloom filters become interesting.

Multidimensional: What are "Reference" Bloom filters

Gateway filters are designed with the idea that a single object is hashed into a filter that will then be unioned with others to make a complete filter. So the Shape of the filter usually has a very large n and a very small p. This tends to yield filters with a very large m.

Reference filters are designed to identify a single object by unioning the hashes of multiple properties together. They are used to try to identify the specific location of the object in a collection of Reference filters. The Shape of the reference filter usually has a very small n and a very small p. This tends to yield filters with an m much smaller than for gateway filters. They are much more compact and have a higher cardinality with respect to m.

Multidimensional: Reference Uses

Reference Bloom filters can be used to:

- Index encrypted data to allow searching without decrypting. [1][6][9][11]
- Provide multi-column indexes that can query any subset of the indexed columns.
- Perform as the master index for large data sets.

Multidimensional: Write multi-column indexes

- Create a Bloom filter for each property to be indexed.
- Merge the Bloom filters together into a Reference filter
- Encrypt the fields for writing if desired.
- Write the record into the storage and get a unique identifier.
- Add the Bloom filter to the multidimensional index and associate it with the unique identifier.

Notes:

- Bloom filters are always created from plain text.

Multidimensional: Read multi-column indexes

Read Operations:

- Create a Bloom filter for each property value in the search criteria.
- Merge the Bloom filters together into a Reference filter for searching.
- Search the multidimensional Bloom filter for matching filters.
- Retrieve the associated unique identifiers
- Retrieve the associated records from storage.
- Decrypt the retrieved records if necessary
- Filter out false positives.

Notes:

- Read operations do not support "OR" just "AND".
- Read operations do not support ranges.
- Read operations do not support inequality.
- Bloom filters are always created from plain text.

Multidimensional: Bloofi

The Bloofi [3] index mixes the gatekeeper strategy with the sharding properties to implement a B+Tree of Bloom filters. The filters are inserted in the leaf nodes and each inner node is a gatekeeper representing all the nodes below it.

In this case the root node $saturation \approx (N_{Filters} \times Average_{Saturation})$

This solution works well for indexing gatekeeper filters where the average saturation is $\approx 1/n$. It allows changing the question of the gatekeeper from "Is this object in here" into "What objects in here match this object".

Multidimensional: Byte/Nibble Trie Solutions

In the Trie solutions the each byte or nibble is used as the index for the next Trie segment. [12]

These solutions then use an expansion function to expand the search at each level of the Trie. The expansion arises because each zero (0) in the target Bloom filter represents two (2) solutions. So nibbles like "0101" have to expand to the set {"0101","0111","1101","1111"}.

Multidimensional: Hamming

This concept uses the hamming weight and estimated log_2 of the Bloom filter as an index. The critical observation here is that no filter can match a filter with a lower hamming value, nor when the filter is interpreted as a very large unsigned integer value, can it match a filter with a lower value.

the log_2 value an easily handled estimation of the very large unsigned integer and is easily estimated from the bit vector representation of the Bloom filter.

This index can be implemented as a skip list, or in a standard database as a multi-segment key.

In addition to the expansion issue noted above, this solution suffers from the distribution properties of the hamming and log_2 values. In that on average each query will search $1/2$ the filters.

Multidimensional: Flat Bloofi

Flat Bloofi [4] creates a table with one row for each bit in the filter and one column for each filter in the index. Matching is performed by performing an intersection of all the matching rows from the target Bloom filter.

BIT	CAT	DOG	GUINEA PIG
0	1	0	0
1	0	0	0
2	0	1	1
3	0	0	0
4	0	0	0
5	1	0	0
6	1	0	0
7	0	0	1
8	0	0	0
9	0	0	0
10	0	0	1

Table: Flat Bloofi layout

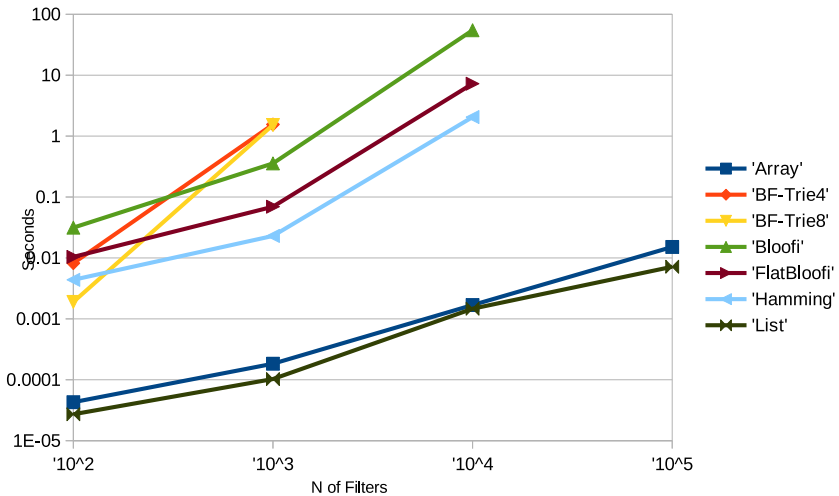
Experimental Data

A testing framework was developed [10] to capture load, query and delete timings for various type of Bloom filters using the Geonames [5] data.

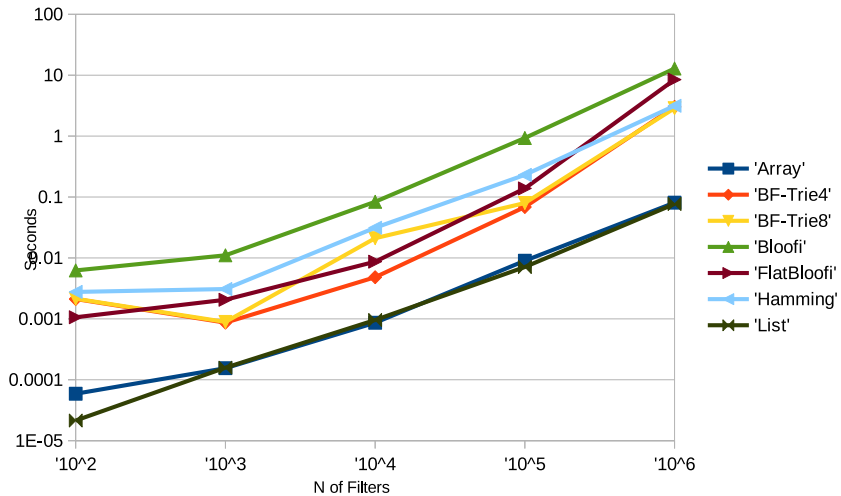
For Reference filters the name, feature_code, and country_code for each item was used to create Bloom filter. We elected to accept a p value of $1/10^6$ and using $n = 3$ the shape of our Bloom filter is $m = 72$ and $k = 17$. The index was then queried with a sample of the total population of inserted filters, with filters comprising only the feature_code (low cardinality) and with filters comprising the name (high cardinality).

For Gatekeeper data the geonameid for each item was used to create Bloom filter that were inserted in the index. The shape of the gatekeeper was specified with $n =$ the population of the test, and $p = 1/10$. The index was queried with a sample of the filters. In the Gatekeeper testing we were unable to run some implementations at desired loads due to memory constraints.

Load Times



(a) Gateway Index Load



(b) Reference Index Load

Search

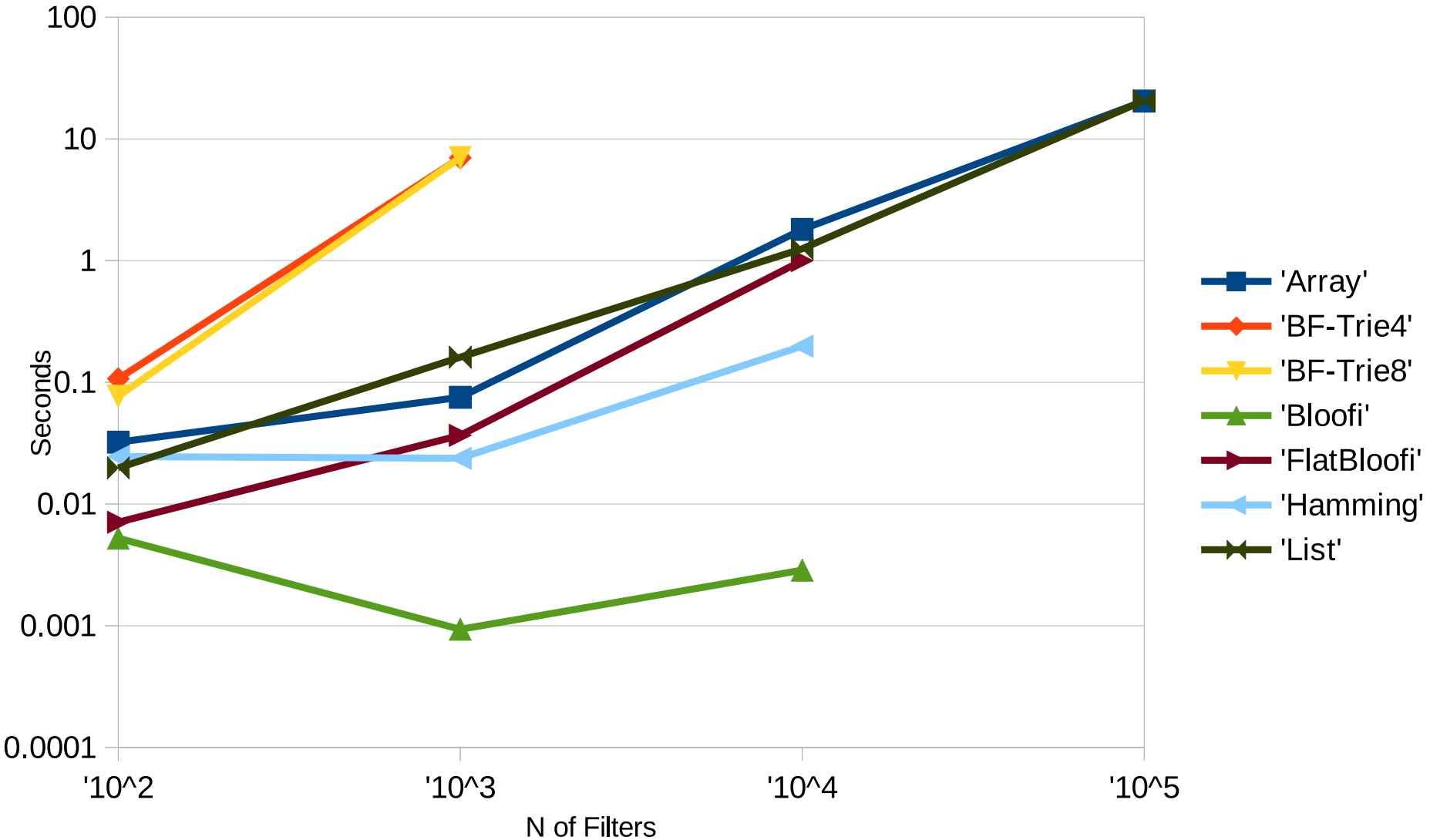
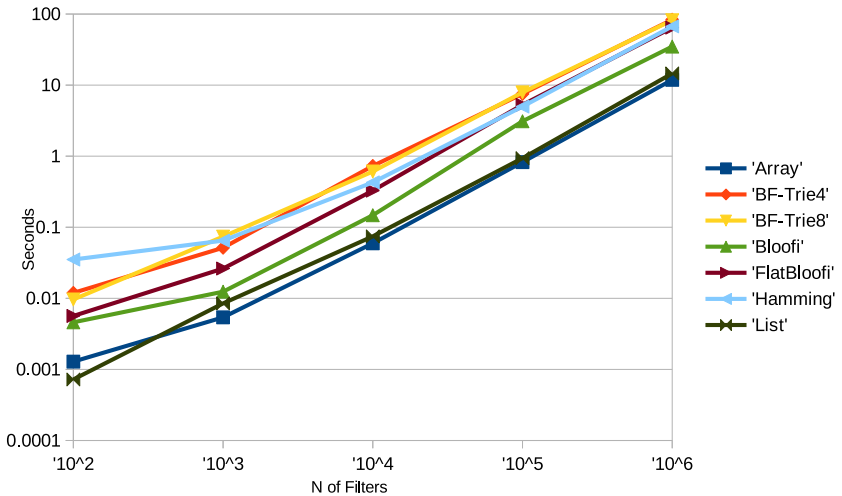
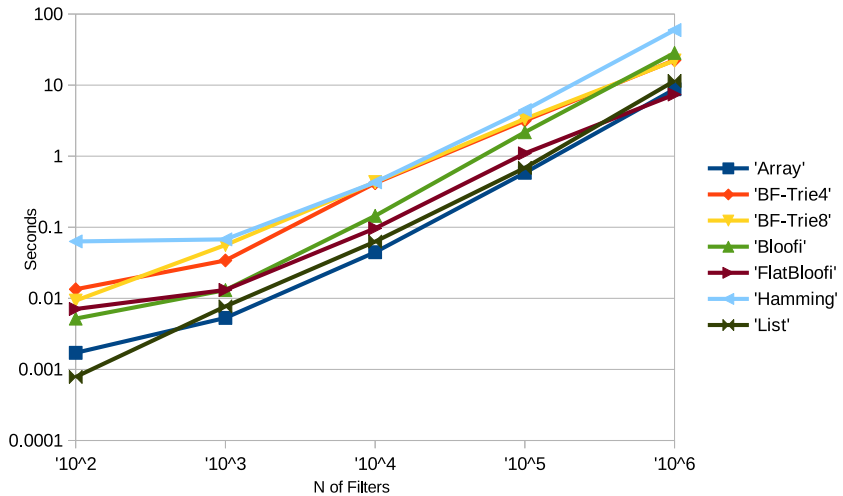


Figure: Gateway Search

Cardinality Search



(a) Reference Low Cardinality Search



(b) Reference High Cardinality Search

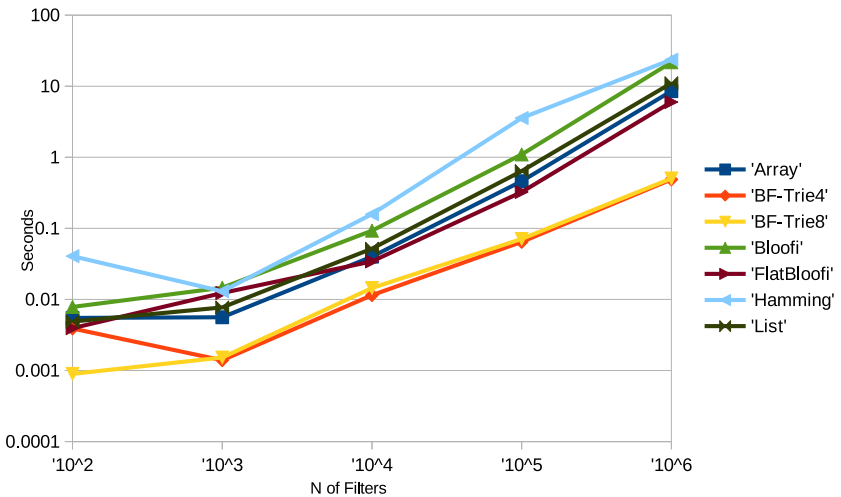
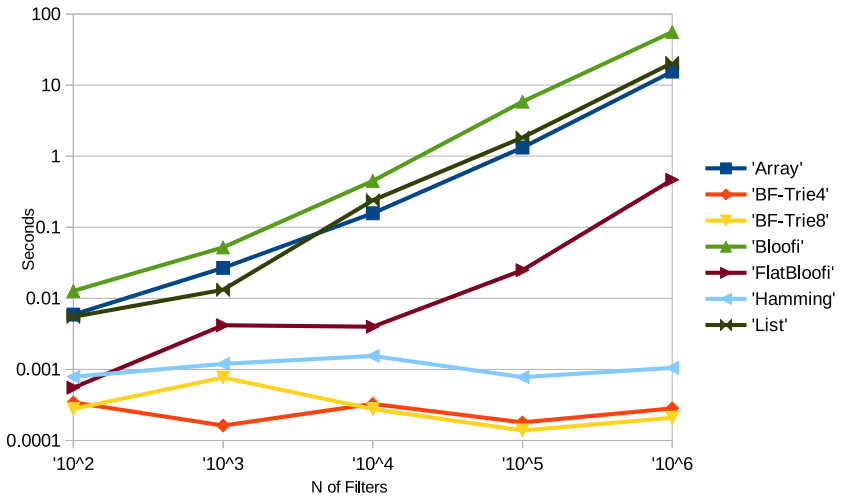
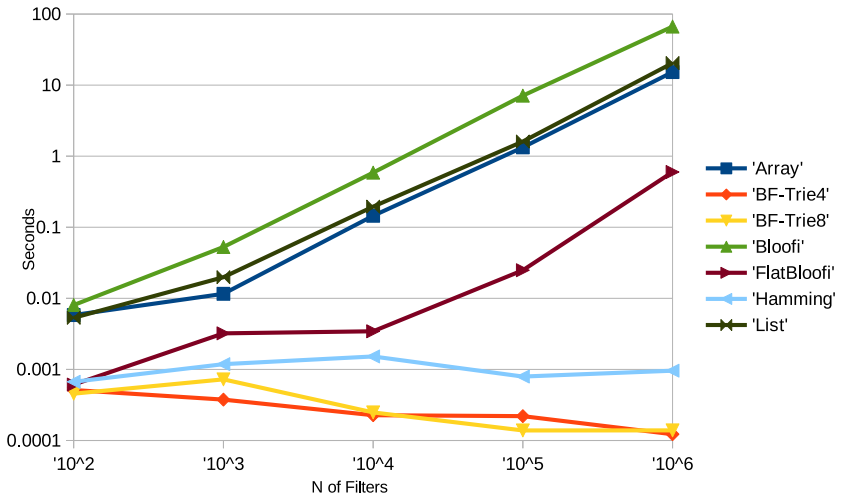


Figure: Reference Complete Search

Deletion



(a) Reference High Cardinality Delete



(b) Reference Low Cardinality Delete

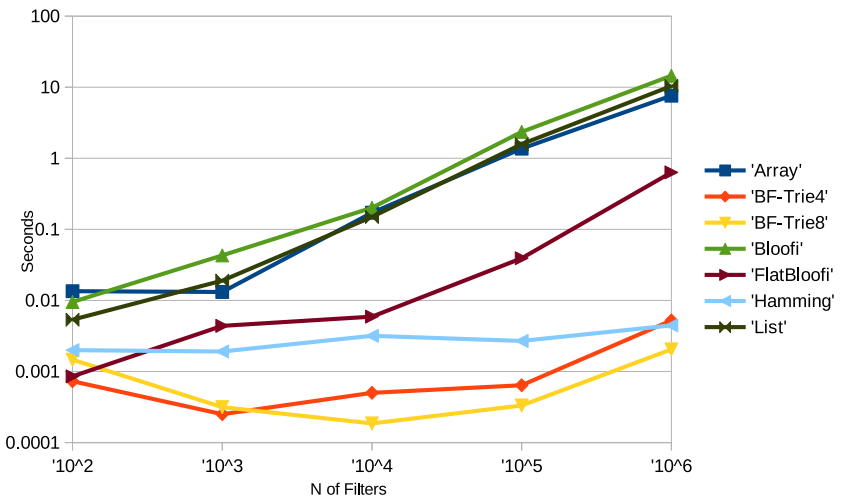


Figure: Reference Complete Delete

Recommendations

- If your application continuously builds and drops multidimensional Bloom filters (e.g. for join processing) use a List or Array, the load overhead in other solutions will probably negate any query performance benefits.
- Determine which type of Bloom filter your application is using. Do not use Trie multidimensional filters for Gateway applications, consider Bloofi.
- If you have Reference filters and you are generally searching for the complete set of data consider using Trie implementations.
- If you have Reference filters and are using partial Reference filters for querying consider using List, Array, Bloofi or Flat Bloofi.
- If you have Reference filters and are updating them consider using one of the strong deletion implementations, like Trie, Hamming or Flat Bloofi.

References I

[1] Steven M Bellovin and William R Cheswick". *Privacy-Enchanced Searched Using Encrypted Bloom Filters*". 2004. URL: https://mice.cs.columbia.edu/getTechreport.php?techreportID=483.

[2] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors"". In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426.

[3] Adina Crainiceanu and Daniel Lemire. *Bloofi: Multidimensional Bloom Filters*. Accessed on 11-Jan-2020. 2016. URL: https://arxiv.org/pdf/1501.01941.pdf.

[4] Adina Crainiceanu and Daniel Lemire. "Bloofi: Multidimensional Bloom filters". In: *Information Systems* 54.C (Dec. 2015), pp. 311–324. ISSN: 0306-4379. DOI: 10.1016/j.is.2015.01.002. URL: http://dx.doi.org/10.1016/j.is.2015.01.002.

References II

[5] Geonames. *GeoNames*. 2019. URL: http://www.geonames.org/.

[6] Eu-Jin Goh. *Secure Indexes*. 2004. URL: https://crypto.stanford.edu/~eujin/papers/secureindex/secureindex.pdf.

[7] Thomas Hurst. *Bloom filter calculator*. 2019. URL: https://hur.st/bloomfilter/.

[8] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge, Cambridgeshire, UK: Cambridge University Press, 2005, pp. 109–111, 308. ISBN: 9780521835404.

[9] Arisa Tajima, Hiroki Sato, and Hayato Yamana. *Privacy-Preserving Join Processing over outsourced private datasets with Fully Homomorphic Encryption and Bloom Filters*. 2018. URL: https://db-event.jpn.org/deim2018/data/papers/201.pdf.

References III

[10] Claude N. Warren Jr. *A Library to Test Bloom Filter Indexing Strategies*. 2019. URL: https://github.com/Claudenw/BloomTest.

[11] Claude N. Warren Jr. "Indexing Encrypted Data Using Bloom Filters". In: Free and Open Source Developers European Meeting. 2020. URL: https://www.researchgate.net/publication/338544506_Indexing_Encrypted_Data_Using_Bloom_Filters.

[12] Claude N. Warren Jr. et al. "Naught For All: An Investigation of Bloom Filter Indexing Strategies". unpublished paper. 2020.