Bloom Filter

Stefan Seelmann

Introduction

A Bloom filter is a space-efficient probabilistic data structure,
conceived by Burton Howard Bloom in 1970,
that is used to test whether an element is a member of a set.
False positive matches are possible, but false negatives are not.

https://en.wikipedia.org/wiki/Bloom_filter

Introduction

A Bloom filter is a space-efficient probabilistic **data structure**,
conceived by Burton Howard Bloom in 1970,
that is **used to test whether an element is a member of a set**.
False positive matches are possible, but false negatives are not.

https://en.wikipedia.org/wiki/Bloom_filter

Introduction

A Bloom filter is a space-efficient **probabilistic** data structure,
conceived by Burton Howard Bloom in 1970,
that is used to test whether an element is a member of a set.
False positive matches are possible, but false negatives are not.

https://en.wikipedia.org/wiki/Bloom_filter

Introduction

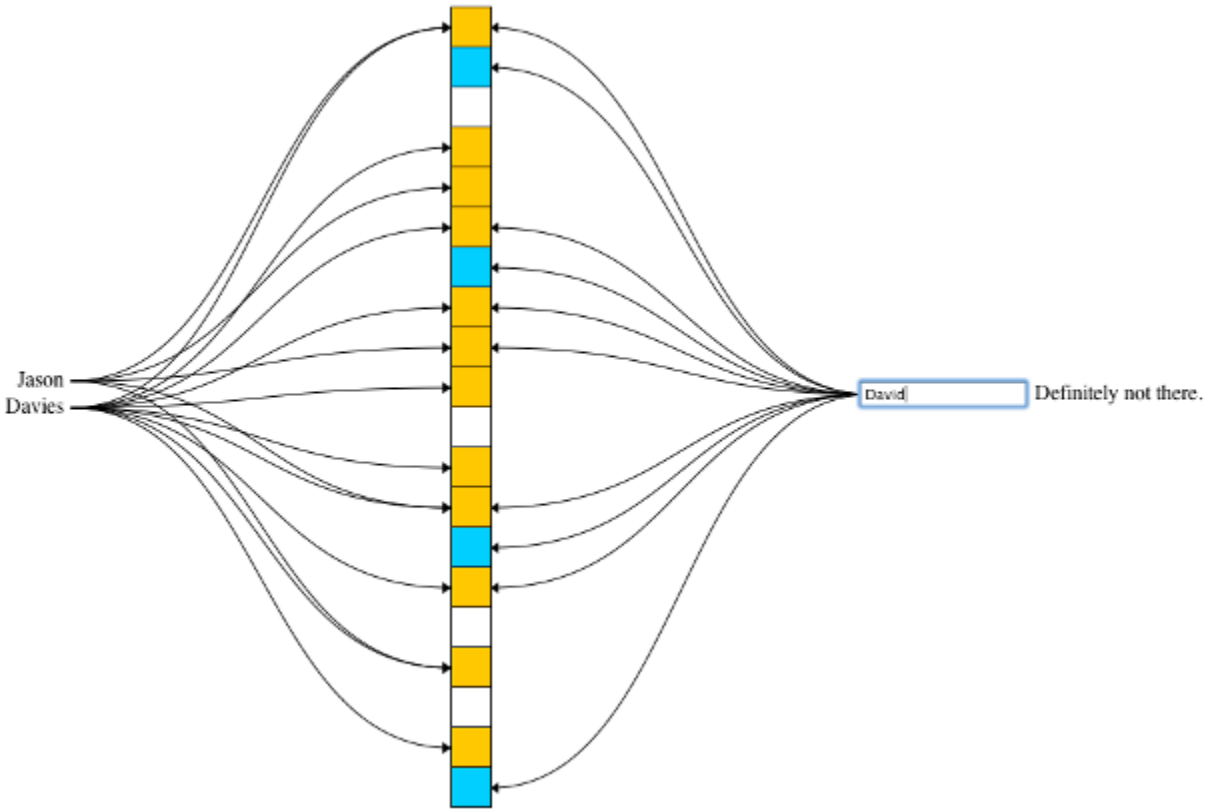
A Bloom filter is a **space-efficient** probabilistic data structure,
conceived by Burton Howard Bloom in 1970,
that is used to test whether an element is a member of a set.
False positive matches are possible, but false negatives are not.

https://en.wikipedia.org/wiki/Bloom_filter

How it works

- bit array of size m
- k hash functions to calulate array positions
- `add(e)`: set the bits
- `test(e)`: check if bits are set

Interactive demonstration



http://www.jasondavies.com/bloomfilter/

Sizing

Trade-off: memory usage vs. false positive rate

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

n: estimated number of elements
p: false positive probability
m: required bit array length

Example:

- n=1,000,000
  - FPR 10% ~= 4800000 Bit ~= 600 kByte
  - FPR 0.1% ~= 14400000 Bit ~= 1.8 MByte

Hash functions

Optimal number of hash functions:

$$k = \frac{m}{n} \ln 2,$$

m: bit array length
n: estimated number of elements
k: optimal number of hash functions

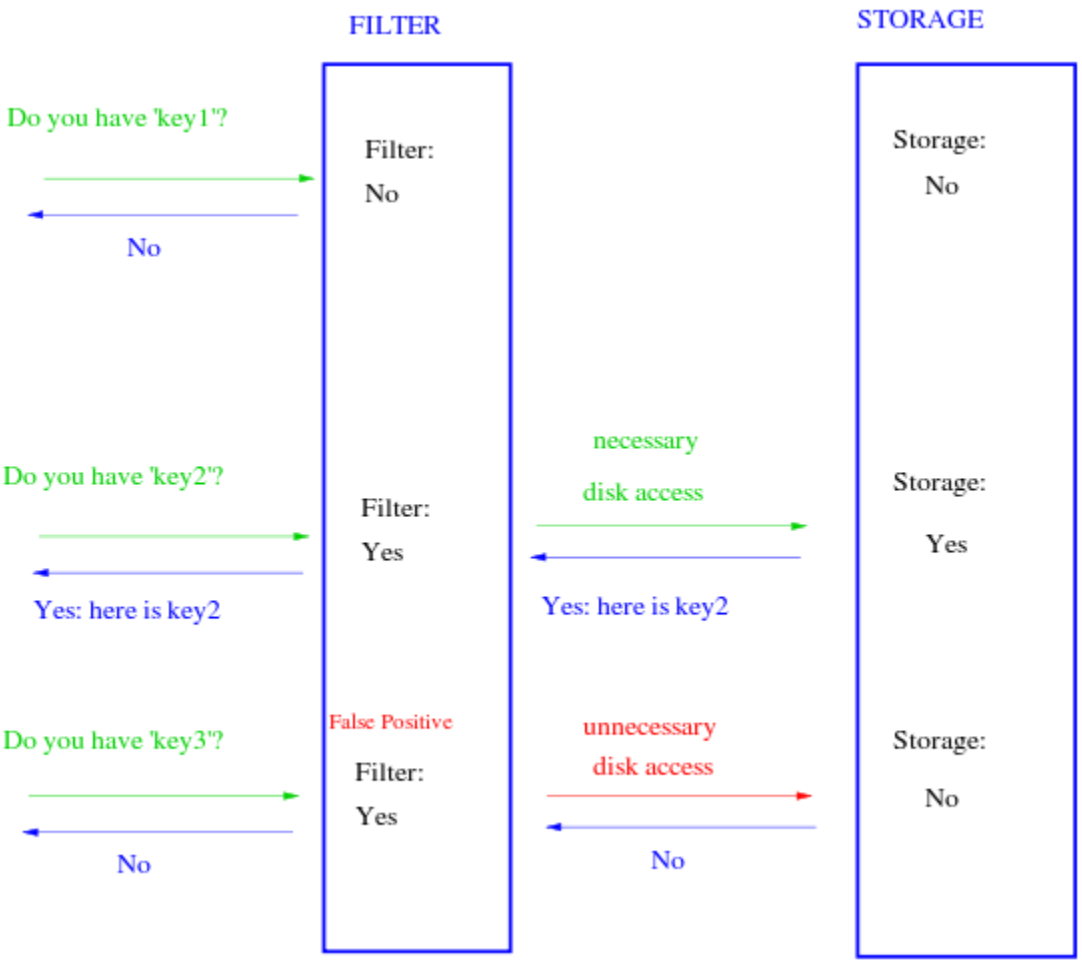
Typical 7..15

Hash function selection:

- random distribution, fast calculation
- different hash functions or same hash functions with different salts
- MurmurHash, Fowler-Noll-Vo

Use cases

- Databases
- Caches
- K/V Stores
  - HBase
  - Cassandra
- Networking
- Chrome Browser



https://en.wikipedia.org/wiki/Bloom_filter

Tricks

- Union
  - Merge 2 bloom filters
  - bitwise OR
  - precondition: 2 Bloom filters with same size and hash functions
- Intersection
  - Measure 'similarity'
  - bitwise AND
  - precondition: 2 Bloom filters with same size and hash functions
- Folding
  - halve in size
  - bitwise OR 1st and 2nd half
  - precondition: size is a power of 2

Variants

- Counting Bloom filters
  - Allow implementation of delete operation
- Scalable (dynamic) Bloom filters
  - Adapt to numbers of stored elements
- Stable Bloom filters
  - For (infinite) streaming applications, evict old state

Sources

- Wikipedia: https://en.wikipedia.org/wiki/Bloom_filter
- Original Paper: http://crystal.uta.edu/~mcguigan/cse6350/papers/Bloom.pdf
- Applications: http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf
- Interactive Demo: http://www.jasondavies.com/bloomfilter/
- Scalable Bloom filters: http://gsd.di.uminho.pt/members/cbm/ps/dbloom.pdf
- Stable Bloom filters: http://webdocs.cs.ualberta.ca/~drafiei/papers/DupDet06Sigmod.pdf