Bloom Filters

Bloom, Burton H. (1970), "Space/Time Trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM* ***13*** *(7): 422–426, doi:10.1145/362686.362692*

Objective:
- Introduction to Bloom Filter,
  - a primary indexing/optimization method in Big Data
- Application to distributed query processing (joins)
  - semi-join reduction

Reading:
  - today: https://en.wikipedia.org/wiki/Bloom_filter

Slide thanks: Dan Suciu, wikipedia

Bloom filter

- A probabilistic data structure, for testing set membership.

Given a set $S = \{x_1, x_2, \ldots, x_n\}$,

With high probability, is y an element of S?

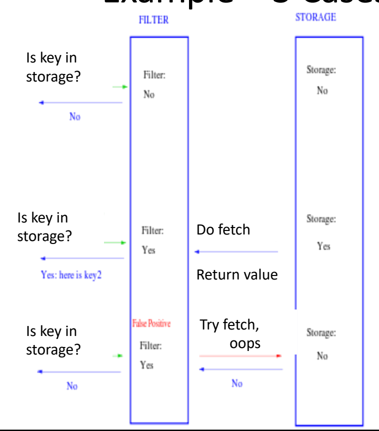
Further,

- A 'No" answer, is correct 100% of the time.
  - no *false negatives*
- A "Yes" answer, is actually "maybe".
  - *false positives* are allowed

Big Data Example

- Storage in Big Data $(key_i, value_i)$

Example – 3 Cases



https://en.wikipedia.org/wiki/File:Bloom_filter_speed.svg

What are the common requirements?

- The set elements are known **a priori,** but queries are unknown
- **Much** of the time, the answer to the membership query will be **no**
- Check **cheaply** for **set membership**
  - "More generally, fewer than 10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set (Bonomi et al. (2006))" [wikipedia]

Hash Maps

- Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of elements
- Let $m > n$
- Hash function $h : S \rightarrow \{1, 2, \ldots, m\}$

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

	1	2										m
H=	0	0	1	0	1	1	0	0	1	1	0	0

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

Hash Map = Dictionary

The hash map acts like a dictionary

- Insert(x, H) = set bit h(x) to 1
  - Collisions are possible
- Member(y, H) = check if bit h(y) is 1
  - False positives are possible
- Delete(y, H) = not supported !
  - Extensions possible, see later

Analysis

- Let $S = \{x_1, x_2, . . ., x_n\}$
- Let j = a specific bit in H ($1 \le j \le m$)
- What is the probability that j remains 0 after inserting all n elements from S into H ?
- Will compute in two steps

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

Analysis

- Recall $|H| = m$
- Let's insert only x_i into H
- What is the probability that bit j is 0 ?

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

Analysis

- Recall $|H| = m$
- Let's insert only x_i into H
- What is the probability that bit j is 0 ?
- Answer: $p = 1 – 1/m$

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

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, . . ., x_n\}$
- Let's insert all elements from S in H
- What is the probability that bit j remains 0 ?

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

Analysis

- Recall $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- Let's insert all elements from S in H

- What is the probability that bit j remains 0 ?

- Answer: $p = (1 - 1/m)^n$

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

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

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

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

- Answer: it is the probability that bit h(y) is 1, which is $f = 1 - (1 - 1/m)^n \approx 1 - e^{-n/m}$

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

Analysis: Example

- Example: $m = 8n$, then $f \approx 1 - e^{-n/m} = 1 - e^{-1/8} \approx 0.11$

- A 10% false positive rate is rather high…
- Bloom filters improve that (coming next)

Bloom Filters

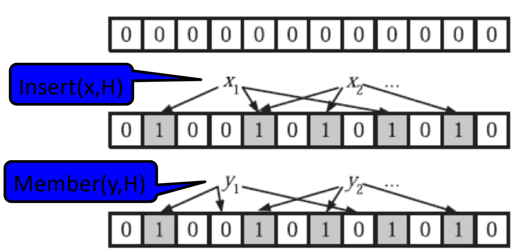
- Introduced by Burton Bloom in 1970
- Improve the false positive ratio
- Idea: use k independent hash functions

For interactive demo see:
http://www.jasondavies.com/bloomfilter/
http://billmill.org/bloomfilter-tutorial/

Bloom Filter = Dictionary

- Insert(x, H) = set bits $h_1(x), \ldots, h_k(x)$ to 1
  - Collisions between h_i and h_j are possible
- Member(y, H) = check if bits $h_1(y), \ldots, h_k(y)$ are 1
  - False positives are possible
- Delete(z, H) = not supported !
  - Extensions possible, see later

Example Bloom Filter k=3



y_1 = is not in H (why ?); y_2 may be in H (why ?)

Choosing k

Two competing forces:

- If k = large
  - Test more bits for member(y,H) ➔ lower false positive rate
  - More bits in H are 1 ➔ higher false positive rate
- If k = small
  - More bits in H are 0 ➔ lower positive rate
  - Test fewer bits for member(y,H) ➔ higher rate

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

Analysis

- Recall |H| = m, #hash functions = k
- Let's insert only x_i into H
- What is the probability that bit j is 0 ?

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

Analysis

- Recall |H| = m, #hash functions = k
- Let's insert only x_i into H
- What is the probability that bit j is 0 ?
- Answer: $p = (1 – 1/m)^k$

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

Analysis

- Recall |H| = m, $S = \{x_1, x_2, . . ., x_n\}$
- Let's insert all elements from S in H
- What is the probability that bit j remains 0 ?

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

Analysis

- Recall |H| = m, $S = \{x_1, x_2, . . ., x_n\}$
- Let's insert all elements from S in H
- What is the probability that bit j remains 0 ?
- Answer: $p = (1 – 1/m)^{kn} \approx e^{-kn/m}$

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?

Probability of False Positives

- Take a random element y, and check member(y,H)
- What is the probability that it returns *true* ?
- Answer: it is the probability that all k bits $h_1(y), \ldots, h_k(y)$ are 1, which is:

$$\boxed{f = (1-p)^k \approx (1 - e^{-kn/m})^k}$$

Optimizing k

- For fixed m, n, choose k to minimize the false positive rate f
- Denote $g = \ln(f) = k \ln(1 - e^{-kn/m})$
- Goal: find k to minimize g

$$\frac{\partial g}{\partial k} = \ln\left(1 - e^{-\frac{kn}{m}}\right) + \frac{kn}{m} \frac{e^{-\frac{kn}{m}}}{1 - e^{-\frac{kn}{m}}}$$

$$\boxed{k = \ln 2 \times m / n}$$

Bloom Filter Summary

Given $n = |S|$, $m = |H|$, choose $k = \ln 2 \times m / n$ hash functions

Probability that some bit j is 1 $\boxed{p \approx e^{-kn/m} = ½}$

Expected distribution $\boxed{\text{m/2 bits 1, m/2 bits 0}}$

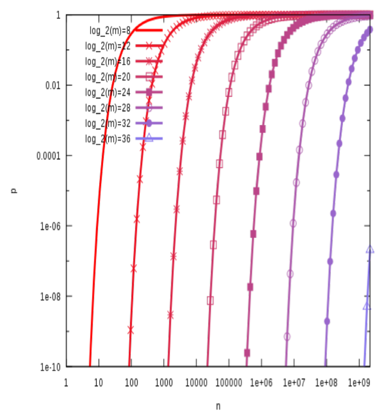
Probability of false positive

$$\boxed{f = (1-p)^k \approx (½)^k = (½)^{(\ln 2)m/n} \approx (0.6185)^{m/n}}$$

Bloom Filter Summary

- In practice one sets m = cn, for some constant c
  - Thus, we use c bits for each element in S
  - Then f $\approx$ $(0.6185)^c$ = constant
- Example: m = 8n, then
  - k = 8(ln 2) = 5.545 (use 6 hash functions)
  - f $\approx (0.6185)^{m/n} = (0.6185)^8 \approx 0.02$ (2% false positives)
  - Compare to a hash table: f $\approx 1 - e^{-n/m} = 1\text{-}e^{-1/8} \approx 0.11$

The reward for increasing m is much higher for Bloom filters



n is number of elements and *m* is filter size. k=(m/n)ln2

https://en.wikipedia.org/wiki/File:Bloom_filter_fp_probability.svg

Set Operations

Intersection and Union of Sets:

- Set S ➔ Bloom filter H
- Set S' ➔ Bloom filter H'
- How do we computed the Bloom filter for the intersection of S and S' ?

Set Operations

Intersection and Union:

- Set S ➔ Bloom filter H
- Set S' ➔ Bloom filter H'
- How do we computed the Bloom filter for the intersection of S and S' ?
- Answer: bit-wise AND: H $\wedge$ H'

Counting Bloom Filter

Goal: support delete(z, H)

Keep a counter for each bit j

- Insertion ➔ increment counter
- Deletion ➔ decrement counter
- Overflow ➔ keep bit 1 forever

Using 4 bits per counter:

Probability of overflow ≤ $1.37\ 10^{-15} \times m$

Semijoin Reduction

Given relations R, S

$R \bowtie_{JK1} S$

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S

PK	JK1	b
4	Dan	101
5	Bob	102
6	Dan	103
7	Jane	104

=

PK_R	PK_S	JK1	a	b
1	4	Dan	A	101
1	6	Dan	A	103
3	5	Bob	C	102

(right) Semijoin

- $R \rtimes S = \pi_{attr(S)} (R \bowtie S)$

$R \rtimes_{JK1} S = S'$

S

R join S =

PK_R	PK_S	JK1	a	b
1	4	Dan	A	**101**
1	6	Dan	A	103
3	5	Bob	C	102

S' =

PK_S	JK1	b
4	Dan	**101**
6	Dan	103
5	Bob	102

S' = subset of rows of S that will appear in R join S

All on one page

$R \rtimes_{JK1} S = S'$

S' = subset of rows of S that will appear in R join S

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S

PK	JK1	b
4	Dan	**101**
5	Bob	102
6	Dan	103
7	Jane	104

$R \bowtie_{JK1} S$

PK_R	PK_S	JK1	a	b
1	4	Dan	A	**101**
1	6	Dan	A	103
3	5	Bob	C	102

S' =

PK_S	JK1	b
4	Dan	**101**
6	Dan	103
5	Bob	102

$S' = R \rtimes S = \pi_{attr(S)} (R \bowtie S)$

Exercise: What is $R \ltimes S$?(left semijoin)

$R \ltimes_{JK1} S = R'$

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S

PK	JK1	b
4	Dan	**101**
5	Bob	102
6	Dan	103
7	Jane	104

$R \bowtie_{JK1} S$

PK_R	PK_S	JK1	a	b
1	4	Dan	A	**101**
1	6	Dan	A	103
3	5	Bob	C	102

$R \ltimes_{JK1} S = R'$

R

PK	JK1	a
1	Dan	A
~~2~~	~~Joe~~	~~B~~
3	Bob	C

S

PK	JK1	b
4	Dan	101
5	Bob	102
6	Dan	103
7	Jane	104

$R \bowtie_{JK1} S$

PK_R	PK_S	JK1	a	b
1	4	Dan	A	101
1	6	Dan	A	103
3	5	Bob	C	102

R'

PK_R	JK1	a
1	Dan	A
3	Bob	C

R' = subset of rows of R that will appear in R join S

Interesting: $R \bowtie_{JK1} S = R' \bowtie_{JK1} S')$

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S

PK	JK1	b
4	Dan	101
5	Bob	102
6	Dan	103
7	Jane	104

R'

PK_R	JK1	a
1	Dan	A
3	Bob	C

S'

PK_S	JK1	b
4	Dan	101
6	Dan	103
5	Bob	102

$R \bowtie_{JK1} S$

PK_R	PK_S	JK1	a	b
1	4	Dan	A	101
1	6	Dan	A	103
3	5	Bob	C	102

Semijoin is *lossless* wrt join

Identity 1: $R \bowtie_{JK1} S = R \bowtie_{JK1} (R \ltimes_{JK1} S)$

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S'

PK	JK1	b
4	Dan	101
6	Dan	103
5	Bob	102

$R \bowtie S$

=

PK_R	PK_S	JK1	a	b
1	4	Dan	A	101
1	6	Dan	A	103
3	5	Bob	C	102

Proof by picture

Identity 1: $R \bowtie_{JK1} S = R \bowtie_{JK1} (R \ltimes_{JK1} S)$

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S'

PK	JK1	b
4	Dan	101
6	Dan	103
5	Bob	102

S

PK	JK1	b
4	Dan	101
5	Bob	102
6	Dan	103
7	Jane	104

=

PK_R	PK_S	JK1	a	b
1	4	Dan	A	101
1	6	Dan	A	103
3	5	Bob	C	102

An important general concept:

(lot's of details in a later lecture)

Semijoin reduction

Useful things computed and/or represented in $O(n^k)$

Become $O(kn^2)$

Which column(s) of R did I need to compute $R \rtimes_{JK1} S = S'$?

R

PK	JK1	a
1	Dan	A
2	Joe	B
3	Bob	C

S

PK	JK1	b
4	Dan	**101**
5	Bob	102
6	Dan	103
7	Jane	104

S' =

PK_S	JK1	b
4	Dan	**101**
6	Dan	103
5	Bob	102

Only needed column JK1

Which column(s) of R did I need to compute $R \rtimes_{JK1} S = S'$?

$\pi_{JK1} R$

JK1
Dan
Joe
Bob

S

PK	JK1	b
4	Dan	**101**
5	Bob	102
6	Dan	103
7	Jane	104

S' =

PK_S	JK1	b
4	Dan	**101**
6	Dan	103
5	Bob	102

Identity 2:

$R \rtimes_{JK} S = (\pi_{JK} R) \rtimes_{JK} S$

Summarize

- Definition of semi-join
  $R \rtimes S = \pi_{attr(S)} (R \bowtie S)$
- Identity 1: $R \bowtie_{JK} S = R \bowtie_{JK} (R \rtimes_{JK} S)$
- Identity 2: $R \rtimes_{JK} S = (\pi_{JK} R) \rtimes_{JK} S$

A very valuable optimization

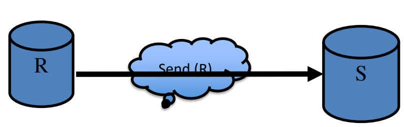
- Definition of semi-join
  $R \rtimes S = \pi_{attr(S)} (R \bowtie S)$
- Identity 1: $R \bowtie_{JK} S = R \bowtie_{JK} (R \rtimes_{JK} S)$
- Identity 2: $R \rtimes_{JK} S = (\pi_{JK} R) \rtimes_{JK} S$

Substitute id. 2 into id. 1 →
$R \bowtie_{JK} S = R \bowtie_{JK} ((\pi_{JK} R) \rtimes_{JK} S)$

Application in a Distributed Join Algorithm

Suppose need to compute R join S

R is on one computer
S is on another computer

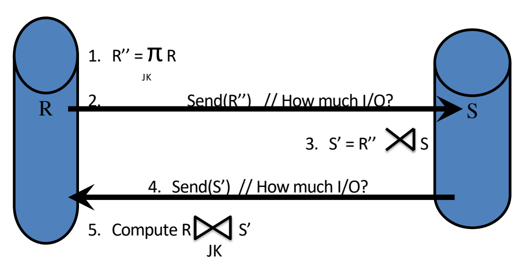


1. Send R to Computer with S
2. Compute R join S

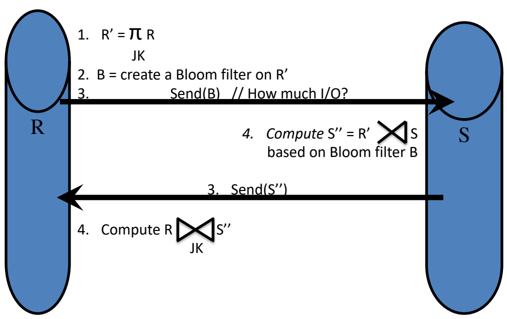
- How much I/O?

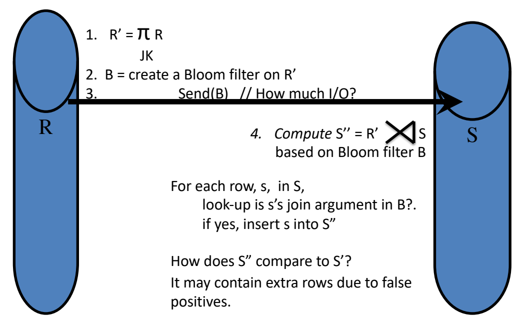
Already know a better algorithm

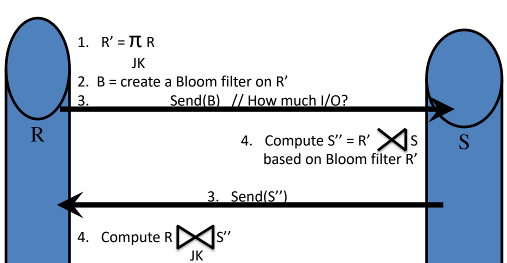
$R \bowtie S = R \bowtie ((\pi R) \rtimes S)$



Even better?
use a Bloom Filter







But S'' does not equal S' !

- Do we care?
- No,
  - the extra rows in S'' will not join with anything in R, and will not appear affect the final result

- "More generally, fewer than 10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set (Bonomi et al. (2006))" [wikipedia]

In industry…

- Google BigTable and Apache Cassandra use Bloom filters to reduce disk lookup.
- "Google Chrome web browser used to use a Bloom filter to identify malicious URLs.
  - Any URL was first checked against a local Bloom filter, and only if the Bloom filter returned a positive result was a full check of the URL performed "
- "Bitcoin uses Bloom filters to speed up wallet synchronization"

Bloom filters and MapReduce

- Pig Latin and HBASE supports it // SQL-like layers on Hadoop
- Hadoop itself has a BloomFilter class
  - http://hadoop.apache.org/docs/current/api/org/apache/hadoop/util/bloom/BloomFilter.html
- Coupled with Hadoop's **distributed cache** facility, this provides a powerful toolkit for performing joins on embarrassingly parallel architectures
  - https://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/filecache/DistributedCache.html

Conclusion

- Bloom filters are probabilistic data structures that offer an excellent tradeoff between three elements:
  - Retrieval (filter reduces time)
  - The precise size of the data structure (space)
  - Probability of false positives (effectiveness)

A Last Word: SQL and Semi-join
[Left] Semi Join

Projects

pid	pName
1	'Design Data Model'
2	'Make UML'
3	'Make DDL'

$\ltimes$

Tasks

pid	tName
1	'get requirements'
1	'digest use cases'
2	'draw UML'

=

result

pid	pName
1	'Design Data Model'
2	'Make UML'

$\Pi_{pid,pname}(\text{Projects} \bowtie \text{Tasks})$

6: Nested Queries cor. CS347 60

SQL has no explicit semijoin keyword Use the following nested query

Projects	
pid	pName
1	'Design Data Model'
2	'Make UML'
3	'Make DDL'

$\ltimes$

Tasks	
pid	tName
1	'get requirements'
1	'digest use cases'
2	'draw UML'

=

result	
pid	pName
1	'Design Data Model'
2	'Make UML'

```
SELECT *
FROM Projects
WHERE pid IN
    (SELECT pid FROM Tasks)
```


There is also an anti [semi] join, simply [Left] Anti Join

Projects	
pid	pName
1	'Design Data Model'
2	'Make UML'
3	'Make DDL'

$\overline{\ltimes}$

Tasks	
pid	tName
1	'get requirements'
1	'digest use cases'
2	'draw UML'

=

result	
pid	pName
3	'Make DDL'

result = Projects - (Projects $\ltimes$ Tasks)

// The rows not included in the semijoin

[Left] Anti Join in SQL

Projects	
pid	pName
1	'Design Data Model'
2	'Make UML'
3	'Make DDL'

$\overline{\ltimes}$

Tasks	
pid	tName
1	'get requirements'
1	'digest use cases'
2	'draw UML'

=

result	
pid	pName
3	'Make DDL'

```
SELECT *
FROM Projects
WHERE pid NOT IN
    (SELECT pid FROM Tasks)
```