# Lecture 24: Bloom Filters

Wednesday, June 2, 2010

# Topics for the Final

- SQL
- Conceptual Design (BCNF)
- Transactions
- Indexes
- Query execution and optimization
- Cardinality Estimation
- Parallel Databases

# Lecture on Bloom Filters

Not described in the textbook !

Lecture based in part on:

- Broder, Andrei; Mitzenmacher, Michael (2005), "Network Applications of Bloom Filters: A Survey", Internet Mathematics 1 (4): 485–509
- Bloom, Burton H. (1970), "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM 13 (7): 422–42

# Example (from Pig Latin lecture)

Users(name, age)
Pages(user, url)

```
SELECT Pages.url, count(*) as cnt
FROM Users, Pages
WHERE Users.age in [18..25]
GROUP BY Pages.url
ORDER DESC cnt
```

# Example

Problem: many pages, but only a few visited by users with age 18..25

- Pig's solution:
  - MAP phase send *all* pages and *all* users to the reducers
- How can we reduce communication cost ?

# Hash Maps

- Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of elements
- Let $m > n$

- Hash function $h : S \rightarrow \{1, 2, \ldots, m\}$

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

$$\Downarrow$$

| | 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

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **1** | **0** | **1** |

# Example (cont'd)

- Map-Reduce task 1
  - Map task: compute a hash map H for **users** with age in [18..25]. Several Map tasks in parallel.
  - Reduce task: combine all hash maps using OR. One single reducer suffices.
- Map-Reduce task 2
  - Map tasks 1: map each **user** to the appropriate region
  - Map tasks 2: map each **page** in H to appropriate region
  - Reduce task: do the join

Why don't we lose any pages ?

# Analysis

- Let $S = \{x_1, x_2, \ldots, x_n\}$
- Let $j$ = a specific bit in $H$ ($1 \leq j \leq m$)
- 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 $|H| = m$, $S = \{x_1, x_2, \ldots, x_n\}$
- 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 $f = 1 - (1 - 1/m)^n \approx 1 - e^{-n/m}$

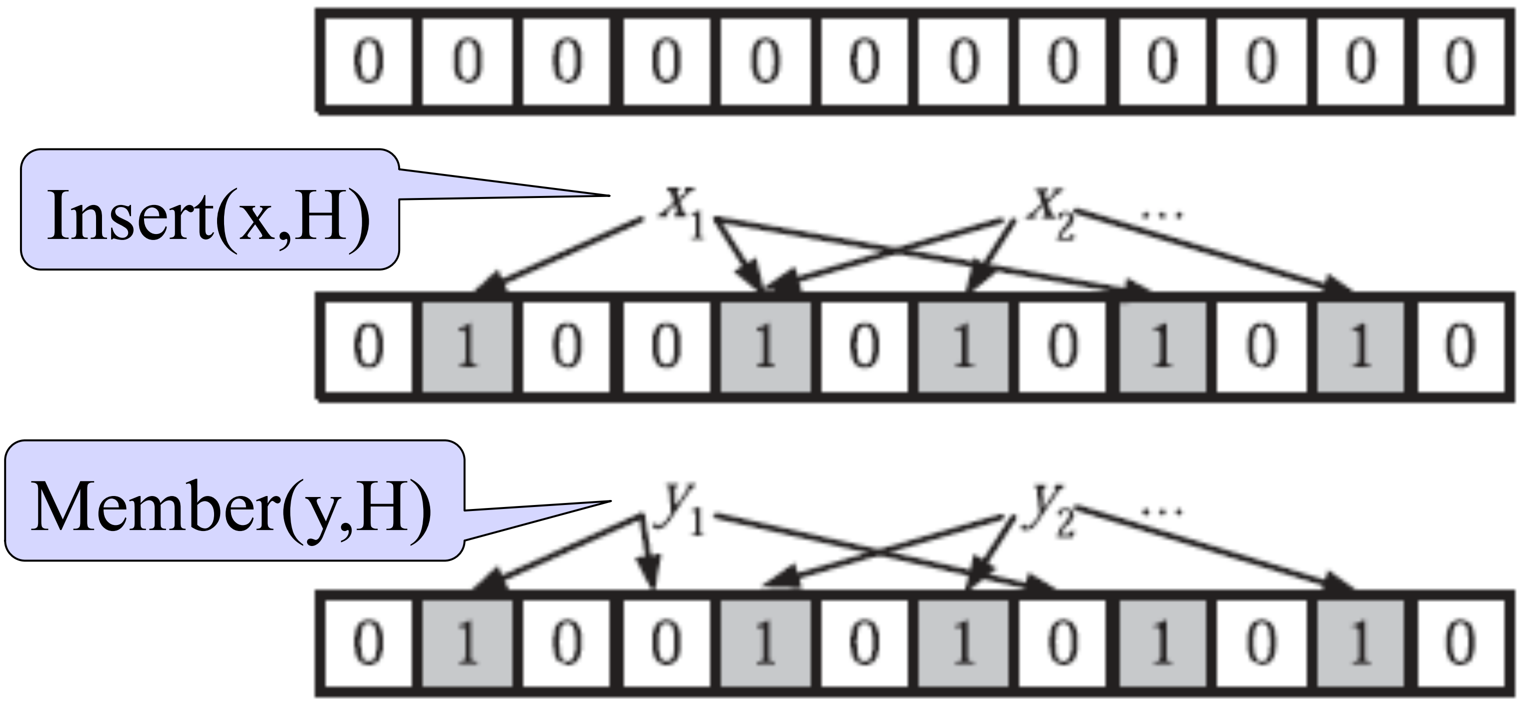
# Bloom Filters

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

# Bloom Filter = Dictionary

- Insert(x, H) = set bits $h_1(x), \ldots, h_k(x)$ to 1
  - Collisions 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 ?)

# Bloom Filter Principle

- Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated

# 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

# Analysis

- Recall $|H| = m$, #hash functions = $k$
- Let's insert only $x_i$ into H

- What is the probability that bit j is 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: $p = (1 - 1/m)^k$

# 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 ?

# 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: $p = (1 - 1/m)^{kn} \approx e^{-kn/m}$

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

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 $f \approx (0.6185)^c = \text{constant}$

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

# 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 $\leq 1.37\ 10^{-15} \times m$

# Application: Dictionaries

Bloom originally introduced this for hyphenation

- 90% of English words can be hyphenated using simple rules
- 10% require table lookup
- Use “bloom filter” to check if lookup needed

# Application: Distributed Caching

- Web proxies maintain a cache of (URL, page) pairs
- If a URL is not present in the cache, they would like to check the cache of other proxies in the network
- Transferring all URLs is expensive !
- Instead: compute Bloom filter, exchange periodically