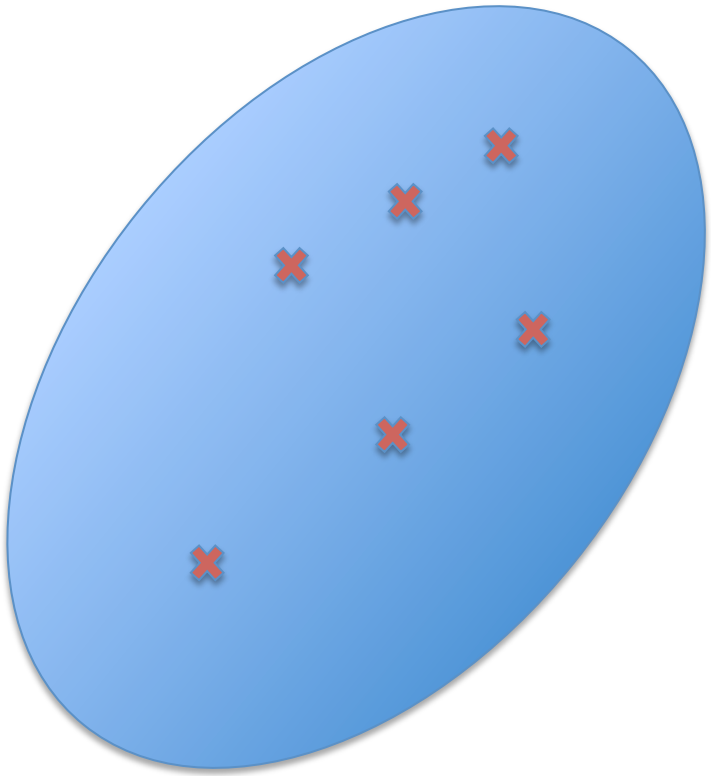
Bloom Filter & Hashing

Barna Saha

Bloom Filter

- Checks for SET MEMBERSHIP efficiently



Is element x in the set?

Motivating Example

- Spam Filtering

➢ We have a set of 1 billion email addresses that we consider to be non-spam.

➢ Each stream element is of the form (email address, email).

➢ Before accepting the email, a mail-client needs to check if this address belongs to set S.

➢ Each typical email address requires 20 bytes of storage whereas in the main memory we only have say 1 billion byte (roughly 1 Gigabyte), or 8 billion bits.

➢ We cannot store all the valid email addresses in the main memory.

Motivating Example

- Spam Filtering
  - All valid emails must be delivered
  - Number of spam emails delivered should be as low as possible

Bloom Filter

1. An array of n bits, initially all 0's.
2. A collection of hash functions $h_1, h_2, ..., h_k$. Each hash function maps "key" values to n buckets, corresponding to the n bits of the bit-array.
3. A set S of m key values

The purpose of the Bloom Filter is to allow efficient insertion of new element into the set and answer membership queries of the form "Is element e in set S?"

Bloom Filter

Initialization: Set the bit-array to all 0's. For every key $K \in S$ set

$$h_1(K), h_2(K), ..., h_k(K)$$

bits to 1.

Testing for membership: To test a key K' that arrives in the stream, check that all of

$$h_1(K'), h_2(K'), ..., h_k(K')$$

are 1's in the bit-array.

If all of them are 1, then return *YES*, else return *NO*.

Analysis of Bloom Filter

False Negative. No false negative. If a key value is in S, then the element will surely pass through the filter, and the answer will be *YES*.

False Positive. A key which is not in set S may still pass. We need to analyze the rate of false positives.

Analysis of Bloom Filter

False Positive.

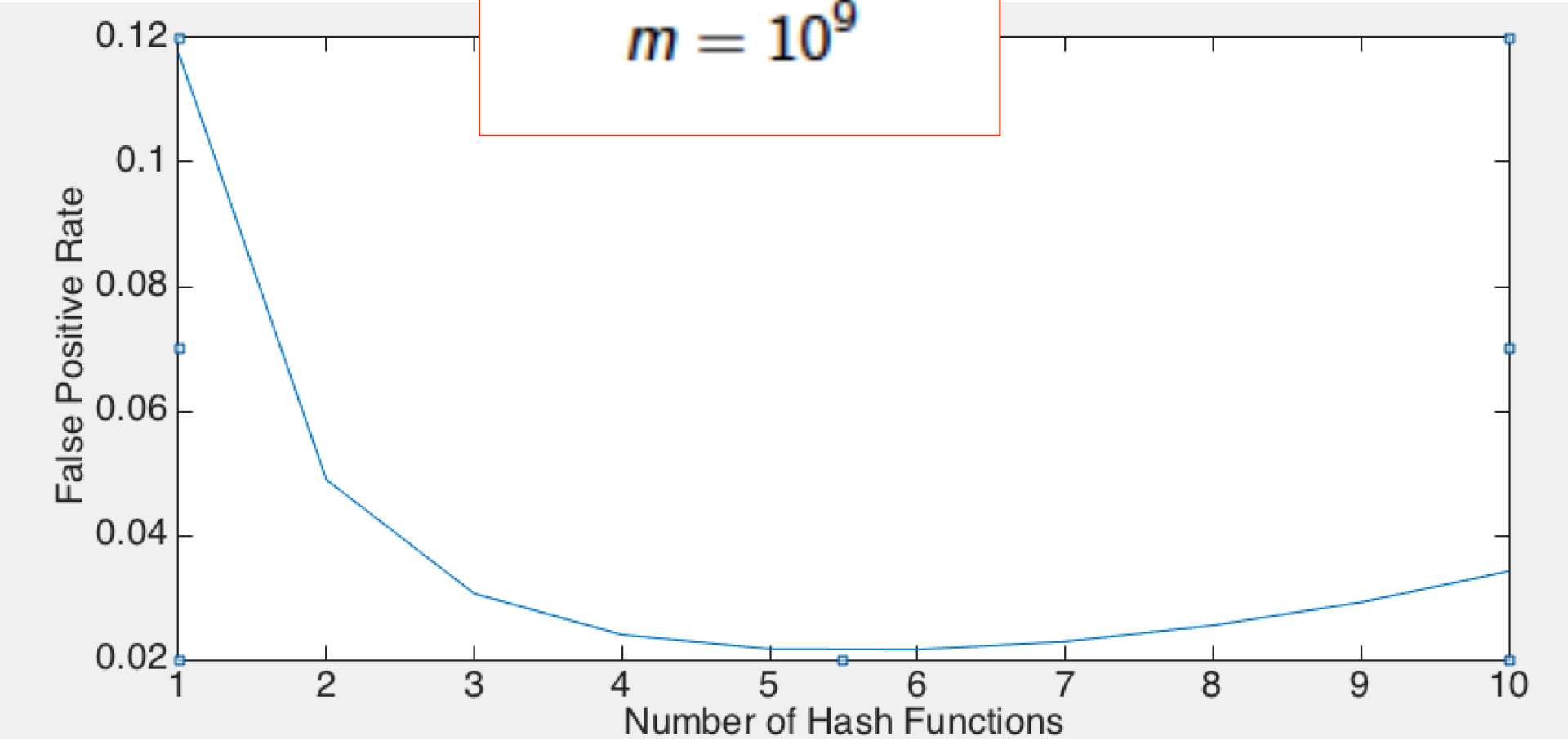
- For simplicity, we assume that for every key K and hash function h_i, $h_i(K)$ is distributed independently and uniformly over the range of values 1 to n.
- For any $1 \leq l \leq n$, let us calculate the prob of the lth bit to remain 0 after inserting all the m elements. It is $(1-\frac{1}{n})^{km} = (1-\frac{1}{n})^{n\frac{km}{n}} \approx e^{-\frac{km}{n}}$.
- Therefore, the probability that the lth bit is 1 is simply $1-e^{-\frac{km}{n}}$.
- Probability of false positive$=(1-e^{-\frac{km}{n}})^k$

Spam Filtering Example

- We have

$$n = 8 * 10^9$$

$$m = 10^9$$



Optimum Value of k

- As the number of hash functions increase, higher is the chance of finding a 0 bit cell
- Also with increasing number of hash functions, the number of cells with 0 bits decreases
- Optimum value obtained by differentiation

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

Applications of Bloom Filter

- Bloom Filter has found innumerable applications in networking and web technology

Akamai Content Distribution Network.

"Akamai's web servers use Bloom filters to prevent **one-hit-wonders** *from being stored in its disk caches. One-hit-wonders are web objects requested by users just once, something that Akamai found applied to nearly three-quarters of their caching infrastructure. Using a Bloom filter to detect the second request for a web object and caching that object only on its second request prevents one-hit wonders from entering the disk cache, significantly reducing disk workload and increasing disk cache hit rates."*

Accessing objects from cache is must faster. Bloom filter allows the detection of objects that are requested for the second time, rather than wasting cache space for one-hit wonders.

Reference: "Bruce M. Maggs and Ramesh K. Sitaraman, Algorithmic nuggets in content delivery, ACM SIGCOMM Computer Communication Review (CCR), July 2015." (PDF).

Akamai Content Distribution Network.

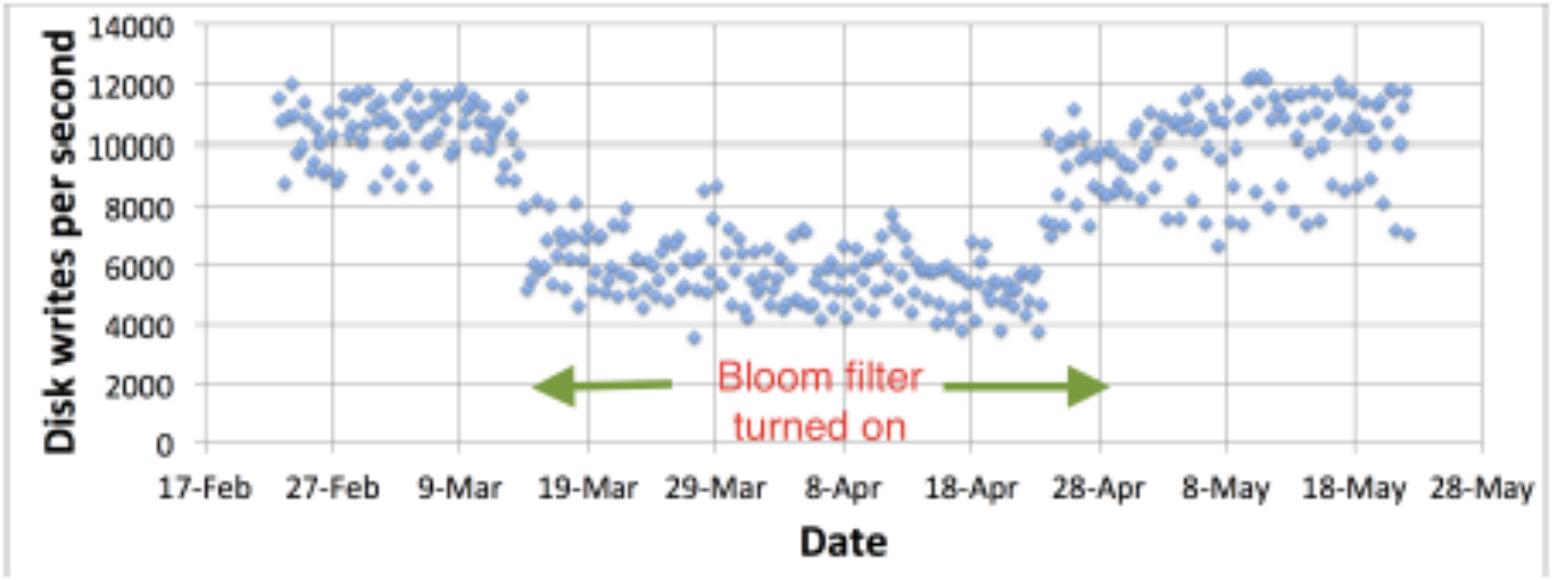


Figure: "Using a Bloom filter to prevent one-hit-wonders from being stored in a web cache decreased the rate of disk writes by nearly one half, reducing the load on the disks and potentially increasing disk performance."

"BloomFilterDisk" by Ramesh K. Sitaraman - https://people.cs.umass.edu/ ramesh/Site/PUBLICATIONS.html. Licensed under CC BY-SA 4.0 via Commons - https://commons.wikimedia.org/wiki/File:BloomFilterDisk.png

Google

" Google BigTable, Apache HBase and Apache Cassandra use Bloom filters to reduce the disk lookups for non-existent rows or columns in SSTables. Avoiding costly disk lookups considerably increases the performance of a database query operation."

Reference: Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson; Wallach, Deborah; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber, Robert (2006), "Bigtable: A Distributed Storage System for Structured Data", OSDI.

Google

"The 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 (and the user warned, if that too returned a positive result). "

Reference: Wikipedia.

Learn more about Bloom Filter

Video link: `https://www.youtube.com/watch?v=947gWqwkhu0`

Analysis of Bloom Filter

False Positive.

- For simplicity, we assume that for every key K and hash function h_i, $h_i(K)$ is distributed independently and uniformly

Analysis uses fully random hash functions—difficult to obtain with high space and computing requirements

- Therefore, the probability that the ith bit is 1 is simply $1 - e^{-\frac{km}{n}}$.
- Probability of false positive$=(1 - e^{-\frac{km}{n}})^k$

Strongly 2-wise Universal Hash Function

- Mapping set of keys U=[0,1,2,…,m-1] to range R=[0,1,2,…,n-1]
  - H={$h_{a,b}$=[(ax+b) mod p] mod n}
- p >=m is a prime, 1 <= a <=p-1, 0<=b <=p-1
- Easy to compute and store: O(1)
- Satisfies (almost) for all $x, y, r, s,\ x \neq y$

$$Prob_{h \in H}(h(x) = r \wedge h(y) = s) = \frac{1}{n^2}$$

Strongly 3-wise Universal Hash Function

- Mapping set of keys U=[0,1,2,…,m-1] to range R=[0,1,2,…,n-1]
  - $H=\{h_{a,b}=[(ax^2+bx+c) \bmod p] \bmod n\}$
- p >=m is a prime, 1 <= a <=p-1, 0<=b,c<=p-1
- Easy to compute and store: O(1)
- Satisfies (almost) $x, y, z, x \neq y \neq z$

$$Prob_{h \in H}(h(x) = r \wedge h(y) = s \wedge h(z) = t) = \frac{1}{n^3}$$

Strongly 2-Universal

- Mapping set of keys U=[0,1,2,…,p-1] to range R=[0,1,2,…,p-1]
  - H={$h_{a,b}$=(ax+b) mod p}, 0<= a,b <=p-1
- Fix $x, y, x \neq y.$
  - What is $Prob_{h \in H}(h(x) = r \wedge h(y) = s)$?
  - Number of hash functions p^2
  - Number of solutions for “a” and “b”=1