Mining Data Streams-2

Barna Saha

January 26, 2016

Review on Reservoir Sampling

In the class we considered the case when reservoir has size 1.

You have a stream of items of large and unknown length that we can only iterate over once. We have a reservoir that can store s items. Create a sampling algorithm such that every item has equal probability $\frac{s}{s+i}$ of being in the reservoir when the reservoir has size $s + i$.

- As long as the stream has size smaller than s, maintain the entire stream in the reservoir.
- Upon seeing the $(s + i)$th element, select it with probability $\frac{s}{s+i}$, if selected discard one item from the current reservoir with probability $\frac{1}{s}$.

Review on Reservoir Sampling

Proof by Induction.

We will show for a stream of size $s+i$, every item $j \in [1, s+i]$ has probability $\frac{s}{s+i}$ to be included in the reservoir.

Base case: $i = 0$, every item has probability $\frac{s}{s} = 1$ to be included in the reservoir.

Induction hypothesis: Suppose the statement is true for stream size $s+1, s+2, ..., s+i-1$.

Induction: Now consider the case when the stream size is $s+i$.

- Probability that the $s+i$th item is stored in the reservoir is $\frac{s}{s+i}$ [the sampling probability]
- Probability that the jth item $j \in [1, s+i-1]$ is stored in the reservoir = Probability the jth item is in the reservoir at step $s+i-1$ AND it is not replaced by the $(s+i)$th element$=\frac{s}{s+i-1} * [(1 - \frac{s}{s+i}) + \frac{s}{s+i} * (1 - \frac{1}{s})) =$ $\frac{s}{s+i-1} * (\frac{i}{s+i} + \frac{s-1}{s+i}) = \frac{s}{s+i}$

Review on Reservoir Sampling

We have just proved

If the stream size is $s + i$, every item in the stream has equal probability $\frac{s}{s+i}$ to be in the reservoir.

Exercise. Show that every subset of s items has equal probability of being in the reservoir. In other words, when the stream has size i, prove that the probability of the reservoir to contain any specific subset of s items is $\frac{1}{\binom{s+i}{s}}$.

Filtering or Selection

- **We want to accept a subset of stream elements that satisfy certain criteria.**
- Example: Accept every undergraduate student who has taken 240 and 311 to 590D–Easy!
- Example: Accept every non-spam email Not so easy!

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

- Use a hash table of size 8 billion where each table stores only one bit, initially set to 0.
- Using a perfectly random hash function, hash every email address in S to one bucket in the table and set that bit to 1.
- Roughly 1/8th of the buckets will have bits set to 1.
- When a new element (*email address*, *email*) arrives, compute the hash value of that email address. If it hashes to a bucket that contains 1 bit, then send the email, else discard it.
- No false negative–a valid email is always delivered.
- False positive is possible–but amount of spam is reduced by 7/8th.

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 $\left(1 - \frac{1}{n}\right)^{km} = \left(1 - \frac{1}{n}\right)^{n\frac{km}{n}} \approx e^{-\frac{km}{n}}$.
- 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

False Positive.
With only 1 hash function, the false positive rate is as follows:

$$n = 8 * 10^9$$

$$m = 10^9$$

- Probability that some lth bit is $0 = \left(1 - \frac{1}{8*10^9}\right)^{10^9} = e^{-1/8}$
- Probability that some lth bit is $1 = 1 - e^{-1/8} = 0.1175$. Since there is only one hash function, this is also the probability of false positive which is very close to $1/8 = 0.125$ that we had roughly calculated before.
- **Exercise.** Calculate the false positive rate when we use 3 hash functions? What happens when we use 4 hash functions?

Applications of Bloom Filter

Bloom filters have found innumerable applications in networking, in 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.

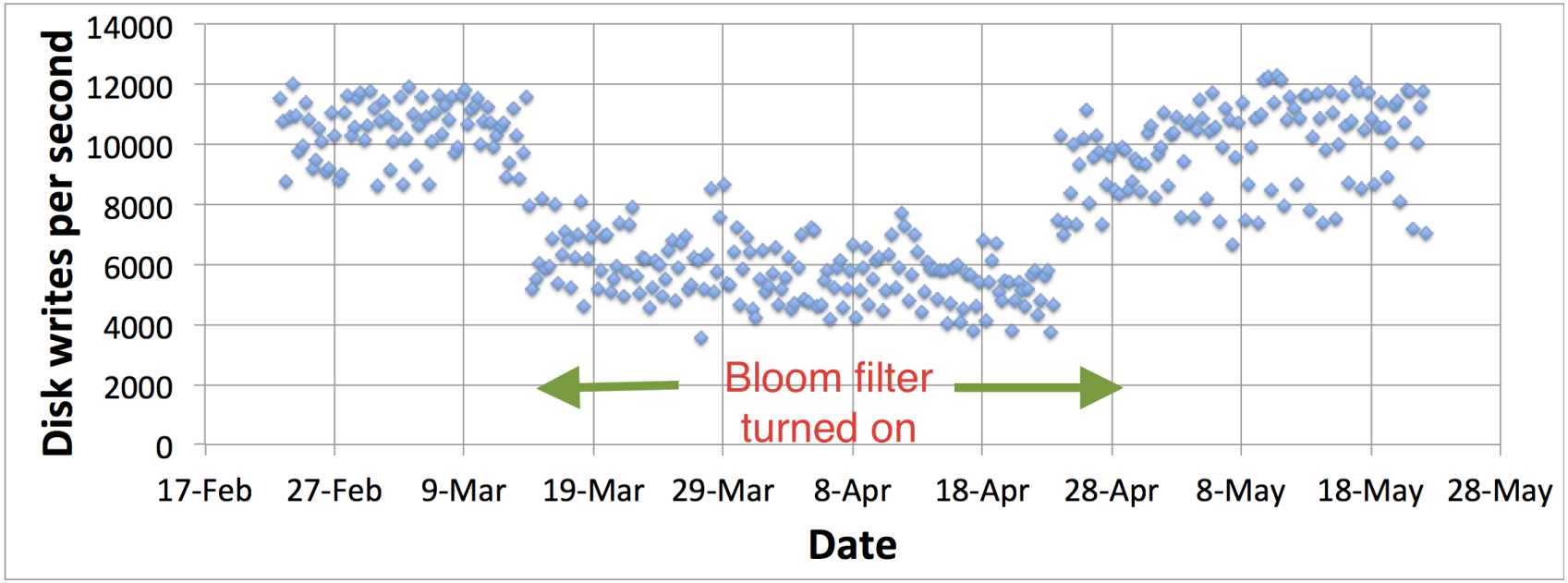

Figure: "Using a Bloom filter to prevent one-hit-wonders from being stored in a web cache decreased the rate of disk writes by nearly one half, reducing the load on the disks and potentially increasing disk performance."

"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`