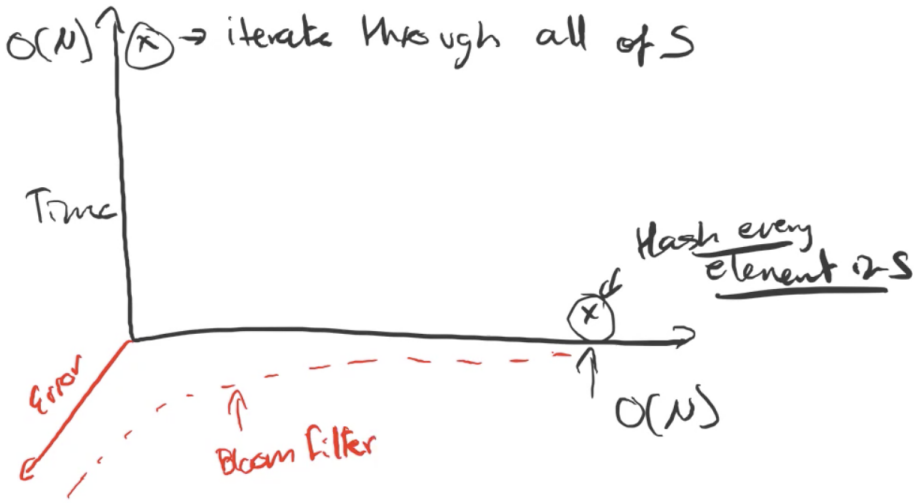
Topics in Big Data Feb 16 lecture notes

Zhiru Zhu

Review:

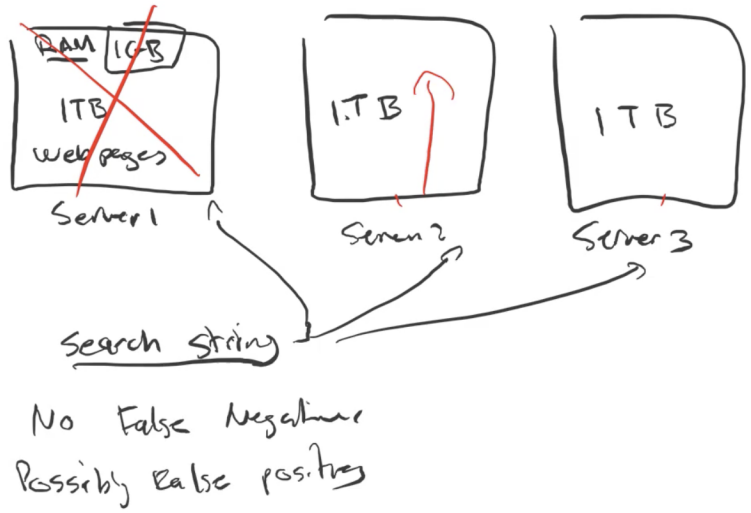
AMQ (Approximate Membership Query) Data Structures
Given a set S, and an element e, is e ∈ S?

There is a space vs time tradeoff:



AMQ adds another axis (error) - there is a spectrum of solutions we can construct if we are allowed to be slightly inaccurate. As we start requiring less space, the number of errors due to collisions starts increasing.

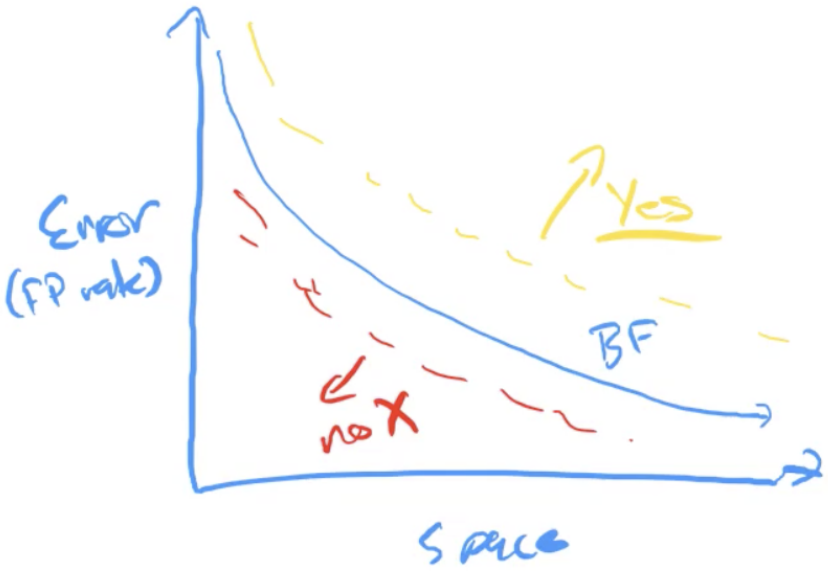
What are the applications of bloom filter?



If we have three servers, each has 1TB of webpages, and we want to search a string in those webpages, we can fit a bloom filter inside the available RAM and test whether the string is present. Since the bloom filter does not have false negatives, we can rule out an entire server if it tells us the string is not present, and we will have to scan the webpages if it tells us the string might be present.

Questions:

1. Is a bloom filter optimal? Given the space vs time tradeoff, is there an optimal curve?



   No. When the code space (N) is close to the size of the total number of elements inserted into the bloom filter (m), then this bloom filter is not optimal.

2. False positive rate: $g(k) = (1 - e^{-k\,N/m})^{k}$
   Optimal choice of: $k^{*} = m/N\ ln2$
   Optimal false positive rate at optimal k: $g(k^{*}) = (0.5)^{m/N\ ln2}$

3. m/N is the compression ratio
   $g(k^{*}) = ((0.5)^{ln2})^{\gamma}$ where γ is the compression ratio m/N
   $g(k^{*}) = (0.618)^{\gamma}$
   This is not optimal because the optimal false positive rate should be 0

Approximate Counting (AC): How many times have we seen an element before?

If we want to count exactly, we need to construct a hashmap of O(D) space, where D bedding the number of distinct elements.
Can we do this with less than O(D) space?
A set of independent hash functions $\{h_i\}_{i=1}^{k}$ (k hashes)
hi: maps s -> {0,...,,m-1}

Suppose we have an m-dimensional vector.
For each s:
  hash s
  increment the bucket
Each bucket has a count of all of the "s"'s with the same hash code
Suppose we get a new "q", we hash q and look it up in the vector, the bucket is the upper bound for its count.

The expected count of a bucket:
$E[bucket] = total / m$

The expected count given that we know element e is in the bucket:
$E[bucket \mid e] = total(e) + total(not\ e) / m = total(e) + \epsilon_e$

$\epsilon_e = total(not\ e) / m$

Where ϵ_e is the expected error (how much we overestimate the count by)

Relative error $R = \epsilon_e / total(e) = total(not\ e) / m / total(e)$

e -> occurs γ times above the average count
$R = total(not\ e) / m / (\gamma \cdot avg(count)) = \frac{N}{m\gamma}$
As γincreases, we can get better estimates (if we focus on the heavy hitters).

Count-Min Sketch
We have k m-dimensional vectors
At query time we get k buckets
We take the minimum of the k buckets because each bucket is an overestimate of the count.
$R \approx \frac{N}{m\gamma k}$
Count-Min Sketch allows you to count with m-bounded memory.
It works well when the queries are skewed to the top, and it might give poor estimates to less frequent elements.

The worst case is when the data is evenly distributed.

Hash-based methods like AMQ (bloom filter) or AC do not consider relationships between elements when assigning them to buckets.
What should you use for counting numbers? Histogram
One-dimensional approximate counting data structure is similar to a histogram, except that it randomly groups data together instead of exploiting any correlations among data.