CS 261P Project #2 Proposal

Empirical Analysis of False Positive Rate for Two variants of Bloom Filters

Rishabh Saxena (rsaxena3) | Chaitanya Prafull Ujeniya (cujeniya)

Summary:

This project explores a new way of implementing a Bloom filter and comparing the performance to a regular bloom filter. The new bloom filter (called *k-array bloom filter*) is implemented by having different arrays for each of the k hash functions. ($h_0(x)$, $h_1(x)$, $\cdots$ $h_{k-1}(x)$)

Description:

In a regular Bloom Filter of size *N* & *k* hash functions, after n insertions, a total of *n.k* set bit operations are done. The formula for a false positive in such a bloom filter is:

$$(1-(1-\tfrac{1}{N})^{nk})^{k}$$

Therefore, each of the *k* bits in a false positive map to one of the bits set by the *n.k* set bit operations.

We define a new bloom filter (called k-array bloom filter) such that, for *n* insert operations, each of *k* hash functions sets n bits in its own individual array (Refer to Fig 2). Therefore, after n insertions, there are a total of *n* set bit operations in *k* individual arrays. **For such a bloom filter, each of the *k* bits in a false positive map to one of the bits set by *n* set bit operations.**

Finally, to ensure that the comparisons are fair, we will ensure that the 2 bloom filters being compared have the same total memory consumption i.e. we will consider the size of a regular bloom filter as *N* and the size of the k-array bloom filter as *k* arrays of *N/k* length. (Refer to Fig 1 & 2 below)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig 1. Regular Bloom Filter with N = 16, and k=4

$h_0(x)$				
$h_1(x)$				
$h_2(x)$				
$h_3(x)$				
	0	1	2	3

Fig 2. *k-array* Bloom Filter with k=4 and k arrays of length 4

Details for implementation and analysis:

The following design considerations will be made during the implementation and analysis:

1. Code implementation will be done in C++.
2. We will use tabulation hashing for both bloom filters. Each *k* hash functions will have their own random table (to make the hash functions unique). The same set of *k* hash functions will be used for both bloom filters.
3. The n elements to be inserted in the Bloom Filter will be generated randomly and logged separately. This will help us distinguish between false-positives and true-positives.
4. We will fix the value of *N/k* to 2000 (size of each array in *k-array* bloom filter). Analysis will be done on different values of *n* and *k* (*n* ranging from [50, 250] and *k* ranging from [2, 10]).
5. For *n* inserted keys, we will first perform *n* membership tests of the inserted keys to sanity test that the bloom filter identifies all true positives correctly. We will then randomly generate *T* keys (around *5.n*) and test for the false-positive rate among these keys and report the results.
6. By the above experiment, we can generate the false-positive rate (FPR) for both bloom filters for different values of *n* and *k*. We will generate 2 plots for each bloom filter of FPR *vs.* k (for fixed value of n) and FPR *vs.* n (for fixed value of k).
7. The generated graphs will allow us to compare the performance of the 2 bloom filters and draw inferences from our results.