Michael Koopmann
Com S 435
PA1 report
10/18/21

For BloomFilterFNV I have a FNV hashing function that converts a string to bytes then goes over the bytes and for each byte xor the result and then multiply by FNV_64_PRIME value that I got from the link on the BloomFilters.pdf. Then when I'm adding it to the filter, I add hash number to the string and then hash it to give me the k hash functions where the difference is just the value I add to the string before hashing it. This works because the FNV hash function xor the result in the function, so you'll get a widely different value by just adding 1 to the string before hashing.

For BloomFilterRan I have a Random hashing function that gets a Prime number that is greater than the filter size or M. I do this by going up by 1 or 2 depending on the situation and checking if that number is prime if not then repeat. Once I have my prime number I get 2 random numbers that are within its range I then take the string convert it to an int with hashCode and then I use the ((a*x)+b)%p formula and return the result as the key. Then I just create an array of those functions each with their own random a and b.

I have a function that creates a random string of a given length by randomly adding a randomly picked char from a list of approved chars until it reaches required length. Then when I start, I go through and add a set number of random strings to each of the bloom filters and a list that I'll use to check with. I then go and create a certain number of test where I create another random string and then check if it's in the list if it then I continue otherwise I go through and check each of the bloom filters and if they return positive then I increase a counter that tracks how many false positives they've had then once I'm done, I get the rate by taking the number of false positives by the number of test.

4 bits:	FP Rate Observed:	FP Rate Theoretical:
BloomFilterFNV	14.90%	14.58%
BloomFilterRan	12.80%	14.58%
MultiMultiBloomFilter	15.10%	14.58%
NaiveBloomFilter	15.50%	14.58%
8 bits:		
BloomFilterFNV	1.70%	2.12%
BloomFilterRan	1.90%	2.12%
MultiMultiBloomFilter	2.60%	2.12%
NaiveBloomFilter	1.90%	2.12%
10 bits:		
BloomFilterFNV	0.70%	0.81%
BloomFilterRan	0.90%	0.81%
MultiMultiBloomFilter	0.60%	0.81%
NaiveBloomFilter	1.00%	0.81%

Because as we increase the number of bits that represent each element in our bloom filter, we have less probability of 2 elements sharing the same bits for their marks, so a higher number of bits decreases our chance of false positive (see equation below). BloomFilterFNV usually got the smallest FP rate for me. The only one that had a considerable difference as the bits were increased was MultiMultiBloomFilter I'd say this is because with just those few arrays it wasn't able to get as many hash functions as it's able to when you increase the number of bits. They're close to the theoretical values which is as expected because this is estimated, and the actual results are a few degrees above or below but generally around the given mark.

I got a 0.1% accuracy for the estimateSetSize function, but I got a 100% accuracy for the estimateIntersectSize I'm not sure what I calculated wrong or if I messed up the equation for the estimateSetSize function.