Bloom Filter Report

Spring 2016

Authors: Zahra Hosseini, Hamid Bagheri

Contents

Classes

BloomFilterDet

We used 64-bit hash function. In order to have an k distinct hash function, we combine Deterministic hash function FNV with the MD5 function in Java. MD5 is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32 digit hexadecimal number. For using Md5 in our hash function we create a 'sum' variable and add it every time by (i+1)*String.charat[i], so we give the weight to each ascii character of MD5 and finally multiply the sum by k, which is the number of hash function. We create a method H_function which return the result of kth hash function for the given string; By doing so for each call we could have a n approximately different numbers and in practice we see that 46 out of 100,000 sting we had False Positives which is a very good result.
We have the following functions:

- BloomFilterDet(int setSize, int bitsPerElement)
- add(String s)
    - Convert string s to lowercase()
    - from i=0 to k
        - call FNV Hash function
        - Set the appropriate bits of the table
- appears(String s)
    - convert string s to the lowercase()
    - for i=0 to k
        - call FNV
        - verify if for all functions all appropriate bits have been set then return true
- filterSize()
    - returns the size of the bitset
- dataSize()
    - returns the number of data that have been added to the table
- numHashes()
    - return k

BloomFilterRan

First we defined an integer RandNo array for 100 numbers. We generate 100 random number and after ax+b%p adding it with random[k%100] .
In a BllomFilterRan method we define k as k= (int) ((bitPerElement)*Math.log(2));

We have the following functions:

- BloomFilterRan(int setSize, int bitsPerElement)
- add(String s)
    - Convert input string s to lowercase()
    - from i=0 to k
        - call randomHashFunction for input s and hash function number k
        - Set the appropriate bits of the table
- appears(String s)
    - convert string s to the lowercase()
    - for i=0 to k
        - cal randomHashFunction()
        - verify if for all functions all appropriate bits have been set then return true
    -
- filterSize()
    - returns the size of the bitset
- dataSize()
    - returns the number of data that have been added to the table
- numHashes()
    - return k
- randomHashFunction: We have found a big prime number 920184149 from a website www.primos.mat.br and define a hash variable hash=(a*x+b)%p
- for a and b use a random integer number

FalsePositives

We can add MD5 algorithm multiplied by k and then add to the FNV hash function in order to have a K distinct hash function.

We use this class to test False Positive: public class MyRandomGenerator, this class create 100 random integer number.
Then we test a FalsePositives for the BloomDet and BloomRandom. We have following methods:

- FalsePositive: We generate some random string for inserting in the BloomDet and BloomRan as a testing value.
    - define number of string
    - For i=0 to numstring
        - call RandomStringGenerator
- FalsePositiveRandomBloom(): We test FalsePostive for 100,000 strings and we can see that the average is 46.8 out of 100,000 for RandomBloom the result would be positive which is a very good function(0.04%).
    - First call BloomRanFuc to create the bitSet
    - for i=0 to 100000
        - Generate a random input.
        - if for all k hash function appear=True,
            - add FalsePositive (FP++)
- FalsePositiveDeterministicBloom(): We test FalsePostive for 100,000 strings and we can see that the average is 56.7 out of 100,000 for RandomBloom the result would be positive which is a very good function(0.05%).
    - First call BloomDetFuc to create the bitSet
    - for i=0 to 100000
        - Generate a random input
        - if for all k hash function appear=True,
            - add FalsePositive (FP++)
-

We experiment the false positive for 10 times and the outcomes are mentioned in the following table:

Deterministic (out of 100,000)	76	46	54	59	65	47	55	53	58	54
Random (out of 100,000)	39	53	38	42	44	50	49	48	57	48

Average deterministic= 56.7
Average random = 46.8

BloomJoin

We join two documents as following:

1. We create BloomFilterDet class: BloomFilterDet BFD=d.CreateBloomFilteronR1() ; which create bloom filter on join column of table R1.
2. Then we call CreateNewTableR3 function: d.CreateNewTableR3(BFD); which reduce R2 to R3.
3. Join 2 documents: d.JoinDoc(); finally we join R1 and R3 to find the answer for join problem
4. Finally we call the method of Wrirefile which write the output to the file:d.WriteFile();

We have the following methods in this class:

- CreateBloomFilteronR1
- CreateNewTableR3
- JoinDoc
- WriteFile
    - create output.txt : a file for writing the result of the join R1 and R2
    - Write string1, string2 and string 3

We have implemented BloomJoin by using Arraylist in java which need about 15 minutes to give us the output.txt. This implementation is in the Distributedjoin2.java. We also implement this function by hashmap which gives us less than a minute to run.

Communication cost

Suppose we have a huge file 20GB on a distributed server, if we run join without BloomFilter and suppose we have the network bandwidths of 100 MB/S. Without Bloomfilter all these records should be send over the network which needs 200 second for transfer over the network. If the file is more than this size the transfer time would be much higher. Therefore using Bloomfilter this time is less than a second because we have the bitTable of size setSize * bitsPerElement. So it would be less than a second to transfer.

Extra

We use a 145Mb file records regarding consumer complains on this resource
http://catalog.data.gov/organization/cfpb-gov
We run our code on this data which the key for table is complainID and the other fields are company name and issues. Then we successfully created the join of these two files.

References

[1]: http://www.java2s.com/Code/Java/Development-Class/FNVHash.htm

[2]: http://www.primos.mat.br/2T_en.html

[3]: http://catalog.data.gov/organization/cfpb-gov