2011

University of Toronto
Scarborough

Kobe Sun 996003756
Shane Jiang 996022281

[CSCD43 ASSIGNMENT 2]

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

Evaluation Report

In the code, we keep track of the counts of **Total Positives**, **True Positives** and **True Negatives**.

Total Positives are all outer tuples that pass the Bloom Filter member test

True Positives are the outer tuples that pass the Bloom Filter member test and eventually find a match. (It is also actually the final output count)

True Negatives are the outer tuples that are already dropped in the first pass

Having these 3 counts, we can get:

False Positives are the tuples that do not eventually find any match but pass the Bloom Filter member test

Total Dropped are all the tuples that do not find any match

By using:

False Positive = Total Positives – True Positives

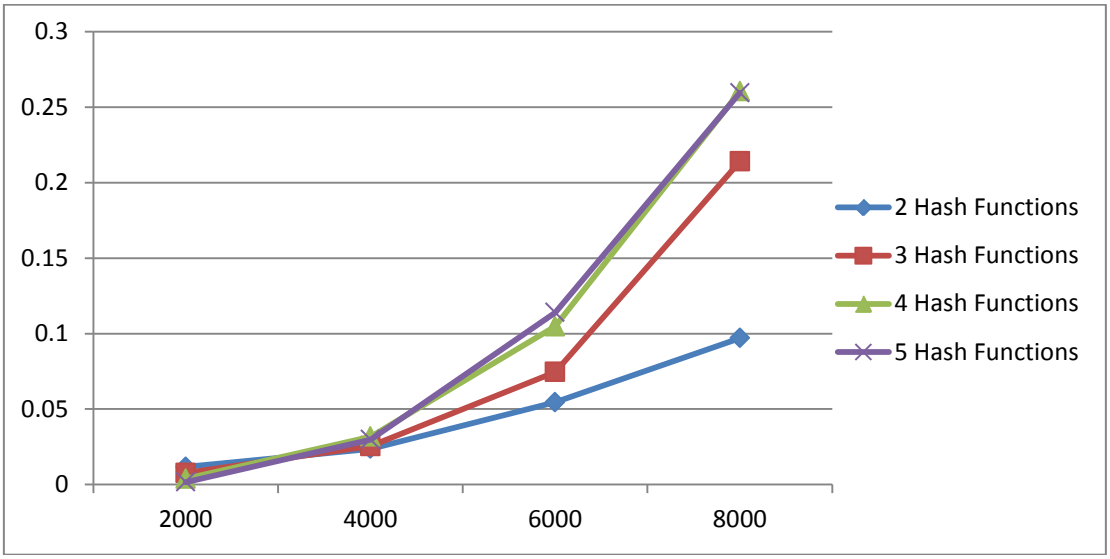
Total Dropped = False Positives + True Negatives

False Positive Rate = False Positive / Total Dropped

Based the above 3 formulas, we have the following tables and graphs

(1) Fixed Bloom Filter Size: 32768
Fixed Outer Input Size: 9000
Queries:
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID <= 2000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID <= 4000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID <= 6000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID <= 8000;

False Postive Rate		INNER INPUT			
		2000	4000	6000	8000
# of HASH FUNCTIONS	2	0.011625	0.023667	0.0545	0.097
	3	0.007625	0.0255	0.0745	0.214
	4	0.003875	0.031667	0.10475	0.2605
	5	0.001625	0.029833	0.114	0.2595



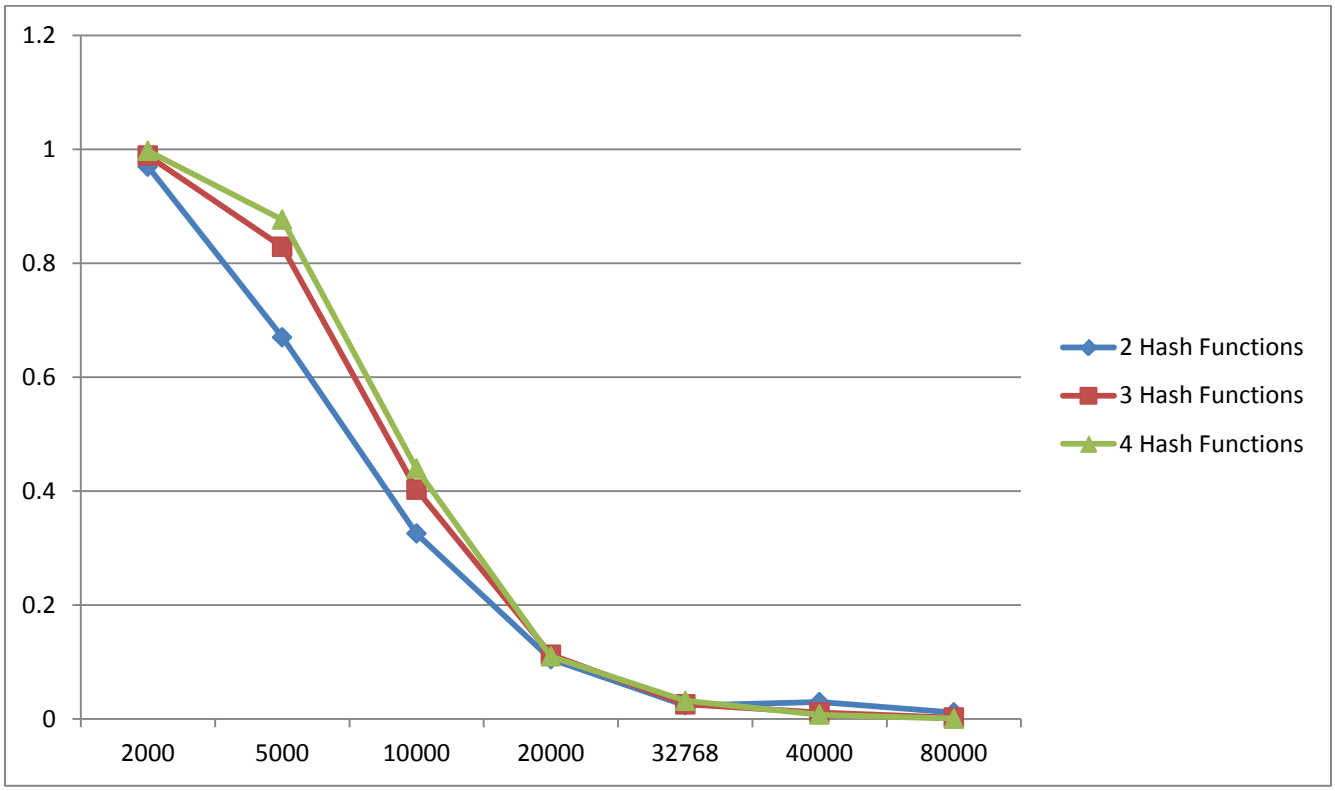
Since the size of bit array and the number of hash functions are fixed, there is a higher chance that more bits will be flipped to 1 when the size of input increases, which decreases the performance of bloom filter.

When the size of bit array and the inner input are fixed, the trend of the performance of bloom filter is not clear. In general, more hash functions using in bloom filter, better the performance will be. However, at the same time, more hash functions will “shrink” the size of bit array, which will decrease the performance. Therefore, for a large size of bit array relative to the number of input, the positive effect of hash function will dominate the performance. On the contrary, for a small size of bit array relative to the number of input, the negative effect of bit array will dominate.

(2) Fixed Inner Input Size: 4000

Query: SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID > 1000 AND S.ID <= 4000;

False Positive Rate		Bloom Filter Size						
		2000	5000	10000	20000	32768	40000	80000
# of HASH FUNCTIONS	2	0.969167	0.67	0.325333333	0.104667	0.023667	0.0295	0.011167
	3	0.988667	0.8285	0.4025	0.112667	0.0255	0.011167	0.002333
	4	0.997	0.876333	0.439333333	0.109667	0.031667	0.007333	0.000167



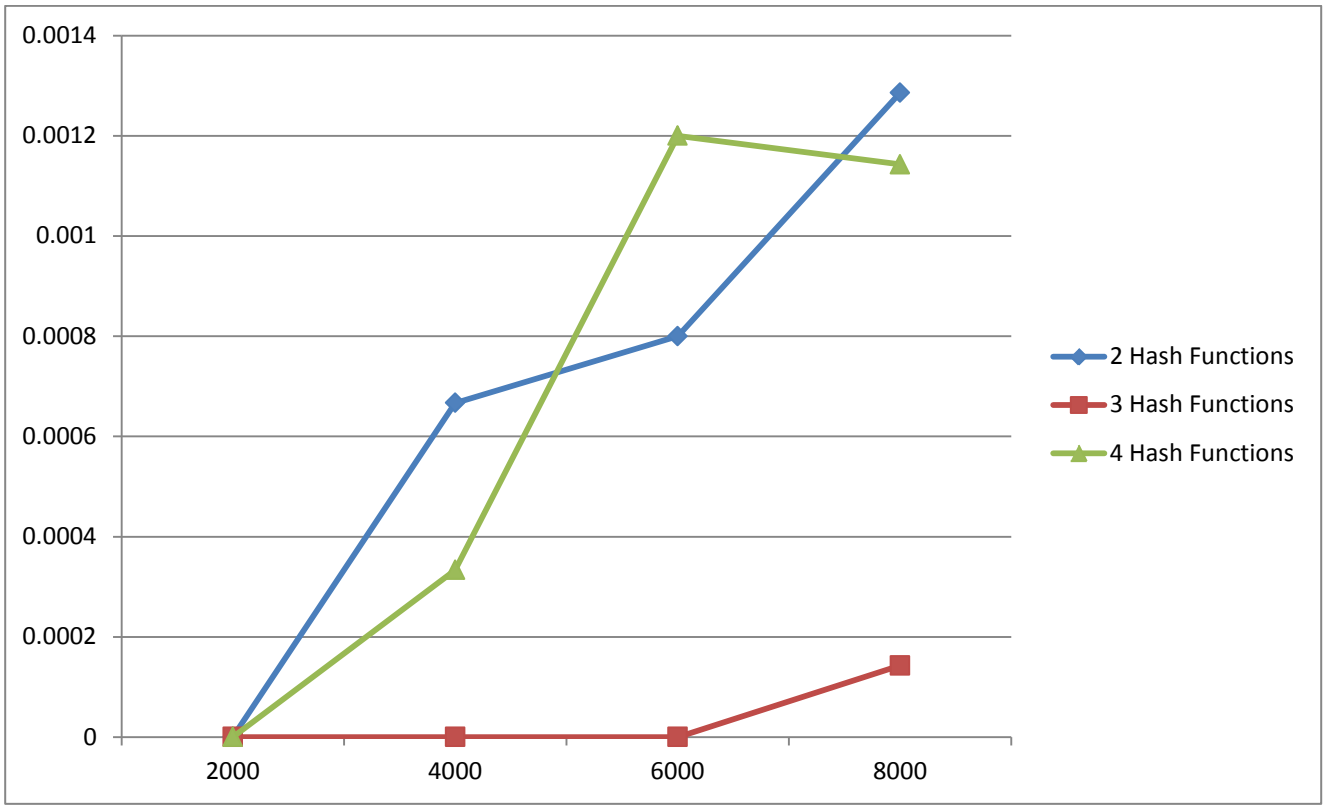
When the inner input size and the number of hash functions are fixed, along with increasing size of bit array, the performance of bloom filter improves, because there are more bits available for the hashing output.

Let the inner input size and the size of bit array become fixed and changes the number of hash functions. Similar to the reason I explained in the 1st graph, the performance determined by the “dominator”.

(3) Fixed Inner Input Size: 1000
Fixed Bloom Filter Size: 32768
Queries:

```
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID <= 2000 AND S.ID <= 1000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID <= 4000 AND S.ID <= 1000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID <= 6000 AND S.ID <= 1000;
SELECT COUNT(*) FROM R,S WHERE R.ID = S.ID AND R.ID <= 8000 AND S.ID <= 1000;
```

False Positive Rate		Outer Input Size			
		2000	4000	6000	8000
# of HASH Funtions	2	0	0.000667	0.0008	0.001286
	3	0	0	0	0.000143
	4	0	0.000333	0.0012	0.001143



Since the inner input size, the size of bit array and the number of hash functions are fixed, thus the bit array using to test membership are always the same. During testing membership process, along with the increasing size of outer input size, there will be a bigger chance that an outer relation tuple misses the match because of the “reliability”. For each tuple, the probability of causing fault positive is very small, for example, 0.000001. Less than 100000 tuples will not cause 1 false positive, because 100000 * 0.000001 = 1. The hash function effect is the same as what I illustrated in above graphs.

How many hash functions would you use?

According to the third graph, using 3 hash functions will give us the lowest false positive rate when we use 32768 as the size of the Bloom Filter.

What would be the size of your bit array?

According to the second graph, we would choose 32768 to be the size of the Bloom Filter bit array, because approximate from that point, the false positive rate is small enough and becomes relative steady, and 32768 would not use too much space.

Describe the strategy you finally choose, and implement it in your code.

As we know, the false positive rate highly depends on the number of tuples in the outer relation n, the number of hash functions k and the size of the bit array in the bloom filter m, we can get some idea from the following relationship among them:

$$\left(1 - e^{-\frac{kn}{m}}\right)^k$$

The above formula illustrates the approximate false positive rate relative to n, k, and m. Thus I set the expected false positive rate to 0.02 and get the value of m from the above. The performance of this kind of dynamic size of bit array is pretty good. However, since the number of tuples we retrieve from the plan is the estimated value and not accurate, suppose there should be a parameter that can make it approach the actual data, but we ignore this parameter in our implementation as it doesn't make too much difference on the false positive rate.