AN ANALYSIS REPORT ON "SPECTRAL BLOOM FILTERS" – A PAPER BY

Saar Cohen &Yossi Matias of Tel Aviv University

CS F211

DATA STRUCTURES AND ALGORITHMS

Submitted by:

K S SANJAY SRIVASTAV	2015A7PS102P
SREE TEJA NANDURI	2015A7PS145P

GROUP NO: 33

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI

ABSTRACT:

The paper assigned introduced an extension to Bloom Filters, **Spectral Bloom Filter**, to accommodate multi-set operations. Bloom filters involves a bit vector hence would lose the capability to check for multiple occurrences of a given element. Bloom filter only accounts for the presence of an element in a given database and not its frequency which is very essential in accounting for aggregate queries. This problem is solved by introducing Spectral Bloom Filters, which behave like a histogram, with high-granularity, where each item has its own bucket.

In this paper, existing bloom filter implementations were compared and contrasted with spectral bloom filter methodology, and few key points regarding the error reduction, memory overhead and time complexity were analysed. For the implementation of a SBF, the bit vector used in the case of a bloom filter is replaced with a vector of counters, which indicates the multiplicities of items. This was implemented using a method known as *Minimal Selection (MS)* method.

Few optimized versions of Minimal Selection method were proposed, called *Minimal Increase (MI)* and *Recurring Minimum (RM)* methods. Both these methods were better than MS in terms of time efficiency and error reduction. MI was better for insert operation whereas RM was better for delete. The method that was used for deletion is the sliding window maintenance method. This method is easy to implement assuming out-of-scope data is available.

The data structure proposed for a better implementation of SBF is String-Array Index. This data structure makes the SBF compact and increases the querying efficiency. In a string-array index, a combination of coarse vector, Offset vector and a lookup table are used in order to ensure effective random access of offset data. It was also used to handle the dynamic problem of inserting and deleting.

Some of the main applications of SBF’s include aggregate queries over specified items, Ad-hoc Iceberg queries, spectral bloom joins and bifocal sampling. Since multi-sets also have many applications in various fields, it can be used to extend and enable new and improve existing applications.

SUMMARY:

The SBF used here are compact and efficient representations of data sets which account for the multiplicities of individual items. The insert and the delete operations can be maintained in O (1) time and can be effectively built for dynamic data. Even though there may be false-positive errors, that rate is minimal and it's called bloom error, which dependent on the selection of various parameters such as the number of hash functions and the size of the array of counters.

For its implementation, using the Minimal Selection method, instead of inserting 1 as in the case of bloom filter, we increment (decrement in case of deletion) the count values of the counters obtained from the hash functions by 1. This method has the same false positive error rate as that of a normal bloom filter. This method could be optimized using minimal increase method, or recurring minimum method.

In minimal increase method, for insertion of r occurrences of an item x, we increment the smallest counter by r and set all the other hashed counters to the maximum of their old value and $m_x + r$. This method outperforms all the existing methods in the aspects of insertion of an item and its error rate is significantly lesser than all the other methods. The only flaw in this method is that it gives false-negative errors in case of deletions, which is unacceptable.

In the recurring minimum method, we use a secondary SBF to store the occurrences of items having a single minimum in the primary SBF as they would be more likely to contribute to the error cases. Also a bloom filter maybe separately used only for saving all the elements that have been moved to secondary SBF, in order to increase its accuracy, but using it requires more space wastage. Since two SBF's are being used, the error that occurs due to false-positives is more likely to get nullified as the single minimum case is likely to contribute more towards the false-positive errors. Thus, recurring minimum method, even though using up extra space, minimizes the error rate improving the performance. This method outperforms all the other methods in the aspect of deletion, as there wouldn't be any false-negative errors.

The main goal of the data structure involved here is to have a compact encoded sequence which is as close to the total number of bits for the count values of all the counters. To access such variable lengths of bit substrings, a string-array index data structure was used. This structure is implemented using two levels of coarse offset array, which provide either the exact location of the item being queried or provides an anchoring value. A threshold for the number of bits is maintained providing the constraints in either using a lookup table or providing direct access. Thus it maintains an O (1) variable length access.

For handling of updates, slack bits are used at regular intervals thus allowing for expanding counter capacity by pushing the nearby counts until the slack bits are encountered in case of insertion. Therefore, this introduces a constraint on the maximum inserts possible. An alternative method called Elias encoding was also used in representing such structure for prefix-free and compact data structure.

Experiments were also discussed in the paper, where the algorithms were tested and compared on various parameters amongst the three methods. Their lookup time, space overhead, error ratios and efficiency of deletions were analysed. This was entirely statistical and thus was more reliable.

In bifocal sampling, a structure called t-index is used for the join operation of the type sparse-any join. Here substituting this t-index with the Spectral Bloom Filter decreases the cost as t-index is more expensive. Also queries involving aggregate operations and threshold checking of the frequencies of items is more efficient through the implementation of SBF.

ANALYTICAL CRITIQUE:

The Spectral Bloom Filter has a wide range of applications as bloom filters are involved deeply in search engines, for example, that involves huge volumes of data. Handling Ad-hoc iceberg queries, which involves testing the frequency threshold based queries, is optimized with SBF while in others a rescanning of an entire file is required. It is also used in bloomjoins, which simplifies performing distributed joins.

The optimized methods discussed in the paper perform far better than the MS method, in the areas of time efficiency, space overhead and error reduction, as seen from the results of the experiments, with error reduction being the main focus, as bloom filters may have considerable false-positive errors.

From what we understand, one of the cases involving the lookup queries was being overlooked in the methods stated in the paper. Consider the scenario, where an element x, having a frequency f_x , is being queried. Let us assume that another element y is also present in the same SBF, having a frequency f_y. Also assume that the element x and element y are being hashed into the bloom filter at the same indices by all the hash functions. Thus, this lookup would've given a result of f_x+f_y , i.e. the minimum of all the hashed values of x, but the actual result desired was f_x. This is not a case of a false-positive as x is present in the bloom filter. This error has not been accounted anywhere, even though the error is more likely to occur due to biased hash functions rather than bloom filter implementation, when considering a large chunks of data, this problem is not one to be overlooked. The error probability of this maybe much less than the false-positive rate. Barring this, the entire implementation seemed efficient and pretty much better than other existing data structures for such purposes. Thus multi-sets can be efficiently handled using the Spectral Bloom Filters.

REFERENCE:

- S. Cohen and Y. Matias. Spectral Bloom Filters ,Technical Report ,

Tel Aviv University, 2003.