A Multi-Layer Bloom Filter for Duplicated URL Detection

Saptarsi Saha   *Shailesh Navale*   *Madhur Navandar*
2020H1030109   *2020H1030169*   *2020H1030163*

Abstract:

Reducing search time in dataset is always consider as a challenging topic of research for several years.Due to enormous increase of data,traditional search methods often takes lot of computation and more time to answer whether given element entry is present in dataset or not.Due to rapid development of internet there are large number of web-pages.In order to improve the speed,URLs are often register in web crawlers.Traditional techniques will take more time as data in crawler increases.In such scenarios Multilayer Bloom filters are considered more efficient.This paper mainly focus on implementation of Bloom filters and their more modified versions and compare them with tradional search algorithms.Along with comparison we will do analysis of how to reduce false positive probabilty in case of Bloom filter and their modified versions

1.Introduction

Bloom filter is a space-efficient probabilistic data structure used to solve membership problems.As there are millions of websites available on internet,after loading the home page of website there may be various different links available on the webpage.It will be time consuming if we extract all the request from the server instead it will be better if we have dataset which stores URL which we already visits such approach is used in web-crawler. But everytime we request a webpage of the URL we need to find whether the given URL is already present in our web crawler dataset or not. The technique will be considered efficient only when search time to query URL must be very much less than the time browser requires to extract the webpage from the server.Such approach is called as detecting duplicate URL.Though working of web-crawler is out of scope for this paper but we will consider this problem of checking if the existing url is already visted by web-crawler as problem statement and compare different search algorithms after implementing it.

1.1. Problem Statement:

Suppose there are 1 million URLs in our dataset and we have to find whether the URL we enter is present in a dataset or not?

First and very naive approach for membership problem is linear search. But if the URL entry is present at the last position of the dataset or what if the enter URL is not prsent so to reach such a conclusion, it need to make 1 million comparisons which is too high.Binary search approach is another alternative to improve the time complexity.But for binary search first need to sort the URL in alphabetical order but then also it take O(log 1000000) time.Though time complexity is reduced compare to linear search but still the insertion operation to binary search is costly as it expects sorted data. To improve more there is another approach called hashing. Hash-table has almost constant lookup time i.e O(1). It depends mostly on the quality of hash function used. Hash table basically solves the problem of time but it still need to store the items in the hash table. Thus hash requires O(n) space complexity.Though there is an improvement in time complexity but space required is more, to overcome such inefficiency of space complexity we use Bloom filter data structure. Bloom filters don’t save the entire URL/value in the hash table instead it uses bitarrays to represent them.False positive cases may arise in bloom filters as two or more items can have the same hash value. Bloom filters use more than one hash functions in order to reduce the false positive probability as it is not storing the value of the data items. We can further reduce the number of false positives if we use multi layer bloom filters as explained below.

3. Bloom filter & analysis

If we use the hashing though time efficient we have a significant concern over the space complexity as it will require O(n) space.In bloom filters we make use of the bitarray and more than one hash functions.If we use more than one different hash functions to indicate that particular item x is in the dataset then we reduce such false positive cases. As shown in the below figure we are using 3 hash functions to store the value of a key x.

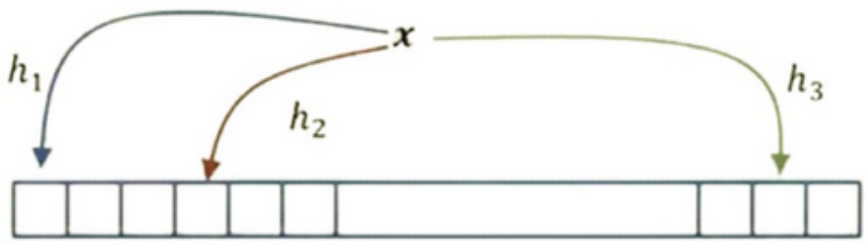

Fig 3.1: Depiction of Bloom filter data structure

In order to check if the value x is already visted, bloom filter will check if the value of the locations bit arrays generated by all hash functions is true or not. If true the value x is may already visited else it is definitely not visited. Upon querying in the bloom filters we can definitely say if whether the value is not visited/present but we can probably tell if the item is present/visited. The algorithm 1 and 2 represents the insertion and lookup operations in bloom filters.

Algorithm 1: Insert into Bloom Filter

```
Suppose there are k hash functions
Input: S= https://$a_1$/$a_2$/$a_3$/../$a_l$
Output: void
  Initialize(B) i.e. for i∈{1..m},
  B[i] = 0
  for i =1 to l:
       for j= 1 to k
           Addr=Hj[ai]
           B[Addr]=1
       end for;
  end for;
```

Algorithm 2: Lookup in Bloom Filter

```
Suppose there are k hash functions
Input: S=https://$a_1$/$a_2$/$a_3$/../$a_l$
Output: Returns p=true if present else returns false
LOOKUP(B,S):
  for i = 1 to l:
       for j = 1 to k
           Addr=Hj[ai]
           if B[Addr]!=1
                    return false
           end if
       end for;
  end for;
```

return true;

Disadvantages of bloom filter:

Issue with bloom filter is in LOOKUP opeartion (Algorithm 2) sometime still returns true even when an element is not present in the data-set.It is called as false-positives values.The challenge is to reduce the false positives to get more accurate results from the algorithm.

Analysis of Bloom filter:

Increasing value of k (number of hash functions) possibly make it harder for false positives to happen because of ANDING(1..k) B[hi(x)], but it also increases the number of filled up position in bloom filter array.Our goal is to find the optimal value of k.

False positive analysis:

Let m = |B| and n elements are inserted.If S has not been inserted, what is the probability that Lookup(B,S) will return true?

Assume hash functions {h_1,h_2,...,h_k} are independent and for element S Pr[h_i(S)=j] = 1/m for all positions of j in a Bloom filter array B.

Suppose we have inserted n elements and we have k hash functions then

$$Pr[h_i(S) = 0] = (1 - 1/m)^{kn} \approx e^{-kn/m}$$

The expected number of zero bits = $me^{-kn/m}$ with high probability.

Now the probability that LOOKUP will return present/true is that

$$Pr[LOOKUP(B, S) = PRESENT] \approx (1 - e^{-kn/m})^k$$

We can easily observe from the above equation that when value of m increases then value of false positive reduces.

Now we need to choose k to minimize false positives,

Let p = $e^{-kn/m}$ then,

$$Log(False\ Positive) = log(1 - p)^k = klog(1 - p) = -(m/n)log(p)log(1 - p)$$

We can observe that we will get the optimal results at p = 1/2.
Thus from the above equation optimal results for the value of k which can be evaluated as below.

$$k = m * log(2)/n$$

The major advantage of bloom filter is space efficient and easily used to query. The time complexity of insertion operations is O(k) , and that of query is also O(k) Because bloom filter needs a bit-array of size m, space complexity is O(m).

Multi layer bloom filter introduction

Though Bloom filter is effective then traditionally approach but it also shows some false positive results. So another version of bloom filter was introduced; it is known as a multi-layer bloom filter in which the amount of false positives are decreased more compared to bloom filters in case of our problem based on duplicated URL detection. We can divide the entire URL in some layers and store them in the multi-layer bloom filter.

URL format which is generally used by the public is of form

S = http://a_1/a_2/a_3/../a_l
S= https://a_1/a_2/a_3/../a_l
Here we are only considering the urls with http or https protocol.

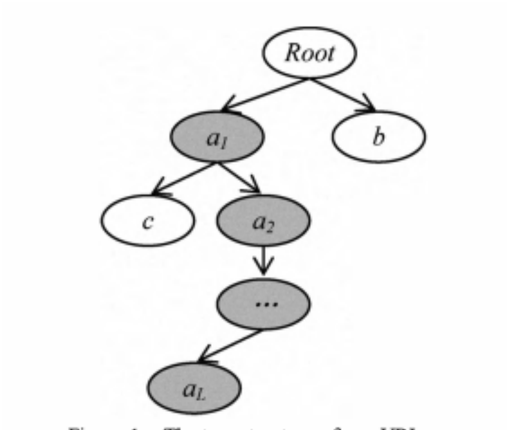

Figure 1. The tree structure of one URL

We can observe from the above figure that each string of the URL separated by delimeter are present at different layers and each layer have corresponding classic bloom filter associated with them we denote bloom filters as $BF^{k,m}$ = classic bloom filter of k hash functions and m bits array each classic bloom filter is used to record the data at each layer.

So overall result will be obtained by using the output of all the layers through upto which our URL is present.

Let L be the number of layers in URL,result of multilayer bloom filter will be like $MLBF^{L,k,m}$ = {${BF_1}^{k,m}$, ${BF_2}^{k,m}$, .. ${BF_L}^{k,m}$}

While inserting the new URL suppose insert a_i in the ith layer of bloom filter if any particular a_i is not in the ith layer then we assume S is not present else present.

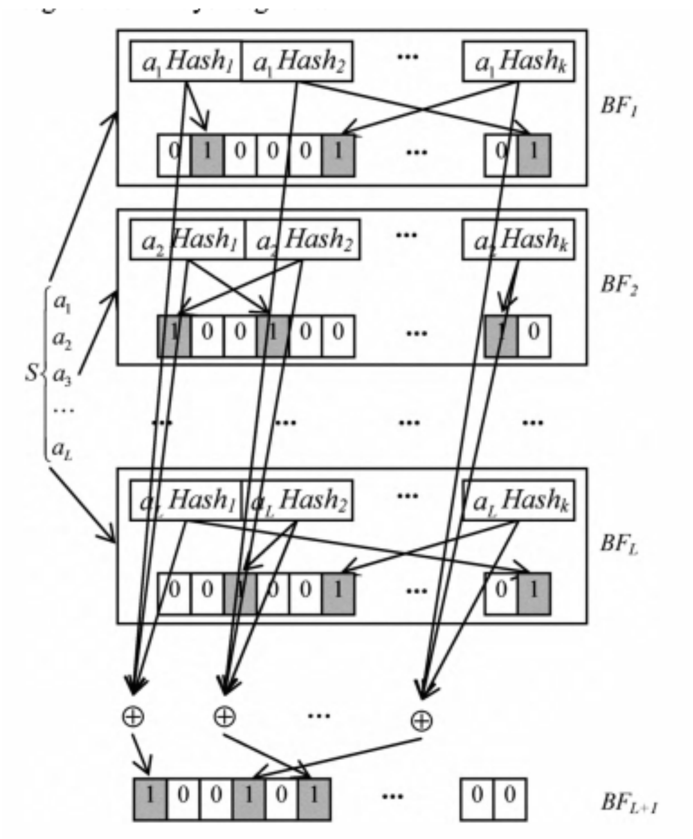

Figure 2. The principle of MLBF

- **MLBF can generate the new possibilities for the false possitives.**

  - for eg. if a_2 is the layer 2 of website S_a and b_2 is the layer 2 of website S_b.
  - if we use the present approach then bloom filter will give false positive for the website S= https:// a_1/b_2/ a_3/../a_l which is wrong.
- Hence to solve this problem an extra bloom filter is introduced which only contains the m bits array. When a new URL is inserted, the way to calculate the bit-position in ${BF_{L+1}}^{k,m}$is to perform an XOR operation among corresponding bit-positions of 1-layer segment to L-Layer segment as shown above.**

Algorithm 3: Insert into Multilayer Bloom Filter[1]

```
Input: S=https://$a_1$/$a_2$/$a_3$/../$a_L$
Output: void
  for i=1 to L:
      for j=1 to k
          Addr=$H_j$[$a_i$]
          $BF_i$[Addr]=1
          LAddr[j]=LAddr[j]$\oplus$Addr
      end for;
  end for;
  for j=1 to k
      $BF_{L+1}$[LAddr[j]]=1
  end for;
```

Algorithm 4: Lookup in MultiLayer Bloom Filter[1]

```
Input: S=https://$a_1$/$a_2$/$a_3$/../$a_L$
Output: if S in MLBF, return true, else return false
//first, check $BF_1$ to $BF_L$
  for i=1 to L:
      for j=1 to k:
          Addr=$H_j$[$a_i$]
          if $BF_i$[Addr]!=1
                   return false
          LAddr[j]=LAddr[j]$\oplus$Addr
      end for;
  end for;
  //second, check $BF_{L+1}$
  for j=1 to k
      if $BF_{L+1}$[LAddr[j]]!=1
         return false;
      end if
  end for;
  return true;
```

The URL query algorithm based on MLBF is shown in Fig. 4. When querying the URL string S= a_1/a_2/a_3/../a_l, check whether all segments of URL are in the set from ${BF_1}^{k,m}$ to ${BF_L}^{k,m}$. After this, it calculates the bit position for the ${BF_{L+1}}^{k,m}$. If it is also present for all the hash functions then S is considered to be in set else not.

Analysis

As we oberved URL are divided into layers and each layer have classic bloom filter so overall result will be the combination of result of all the bloom filters use in layers i.e from level 1 to level L and the result of overall bloom filter will be store in number bit array which we consider is present at level L+1. As in the bloom filter analysis we observed the false positive probability

P[false positive of classic bloom filter]$\approx (1 - e^{-kn/m})^k$
So for L classic bloom filter it will be multilayer bloom filter(MLBF)

P[false positive of MLBF]$\approx (1 - e^{-kn/m})^{kl}$

We can observe clearly from above equation, that false positive of MLBF is less than classic bloom filter.

Experiment and Results:

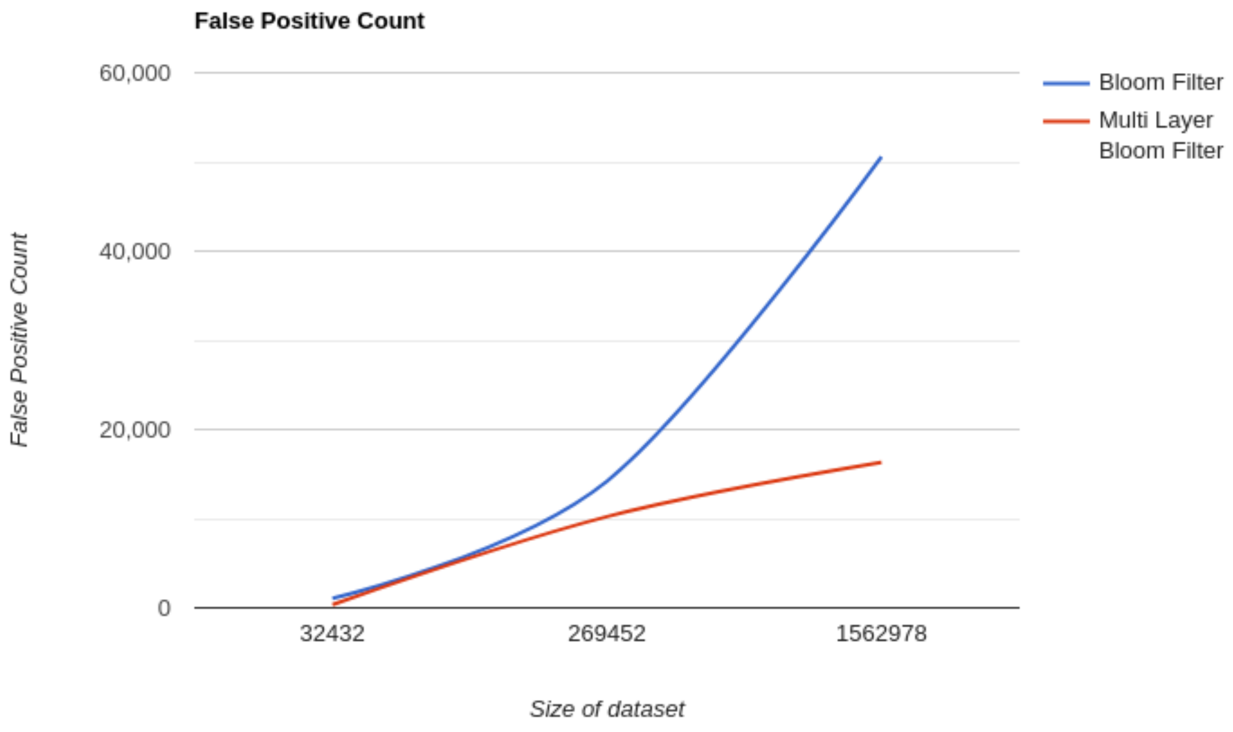

Fig 3. False positve comparison between bloom filter and multilayer bloom filter

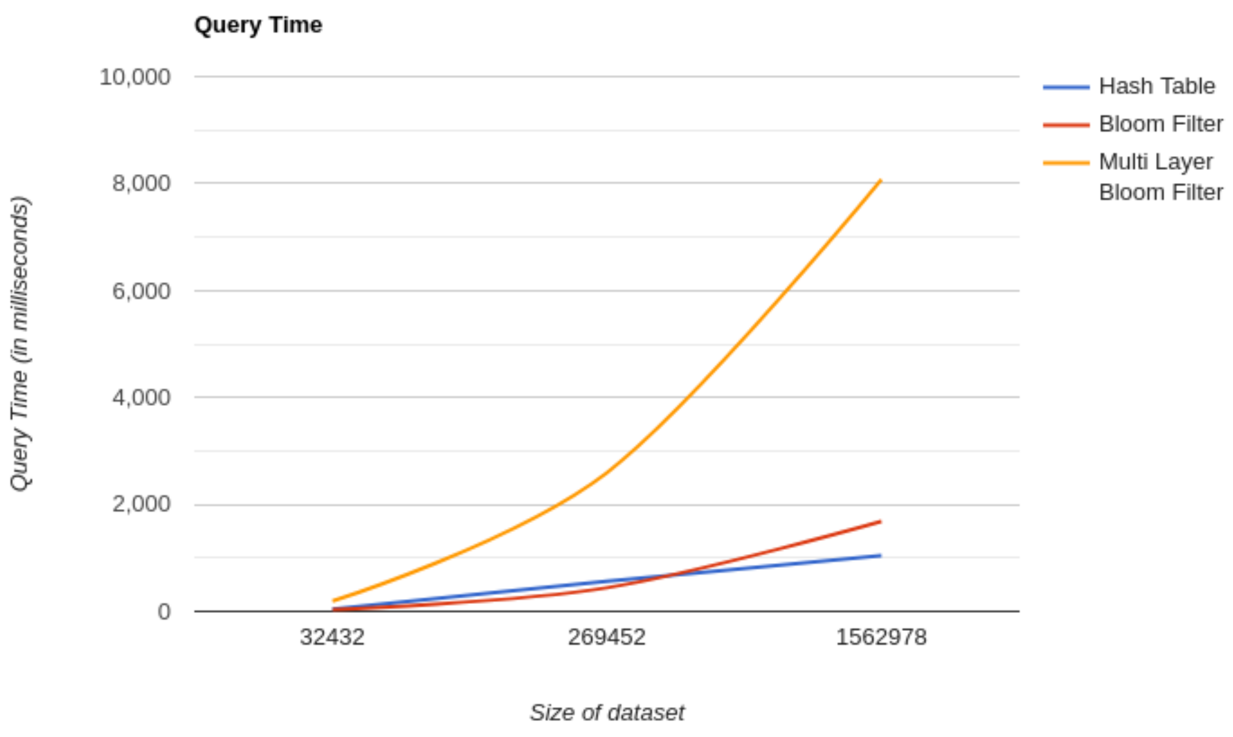

Fig 4. Query Time Comparison hash,bloom filter and multilayer

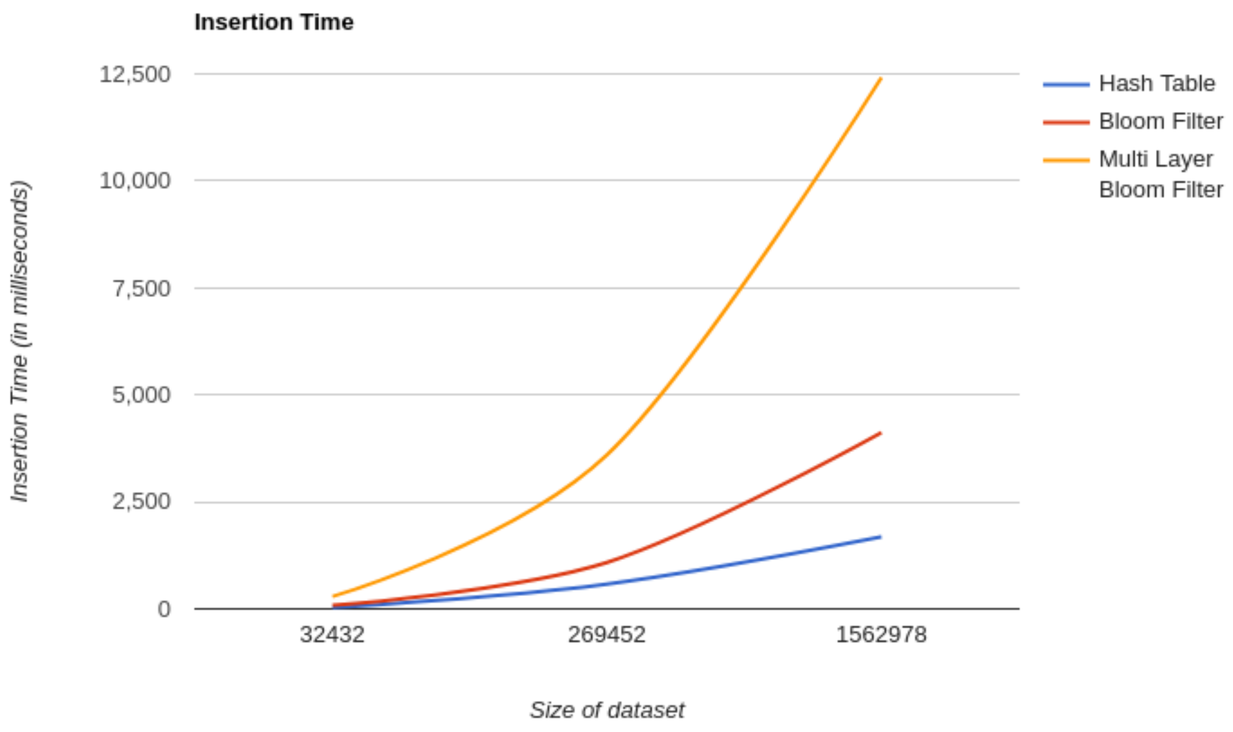

Fig 5. Insertion Time Comparison hashing,bloom filter and multilayer bloom filter

Above figures show the experimental results on the datasets of the different count sizes of the URL. As infered by the figure 3 the number of false positive counts for the multilayer bloom filters is far less than the false positives obtained for the bloom filters. Though the query time and insertion time of the multilayer bloom filter will be more as there are more number of hash functions involved as inferred by figure 4 and 5 respectively.

Conclusion:

In this paper we compare traditional search algorithm with bloom filters and multilayer bloom filter by implementing duplicated URL detection algorithm using both the techniques.It was observed that both version bloom filters are efficient in term of search time and space in comparison with traditional approaches but also have some small probability of error.Comparison between bloom filter and multilayer bloom filter it was oberve that multilayer bloom filter takes more space compare to classic bloom filter but have lower false positives.Both multilayer bloom filter and classic takes same time for insertion and for query.As multilayer bloom filter have least false positives compare to classic bloom filter it is consider as most effective algorithm for duplicated URL detection overall and can be use in implementation of web crawler.

References

[1] A Multi-Layer Bloom Filter for Duplicated URL Detection, Cen Zhiwang, Xu Jungang, Sun Jian, 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), Aug. 2010,
IEEE
[2] https://github.com/saptarsi96/Algo-Term-Project