Searching in large dataset Using Multi-layered Bloom filter

Lalit Sharma
NIT TRICHY
November 12,2021

Contents

1.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 take lot of computation and more time to answer whether given element entry is present in dataset or not. .The aim of the proposed system mainly focus on implementation of Bloom filters and their more modified versions and compare them with traditional search algorithms. Along with comparison we will do analysis of how to reduce false positive probability in case of Bloom filter and their modified versions.

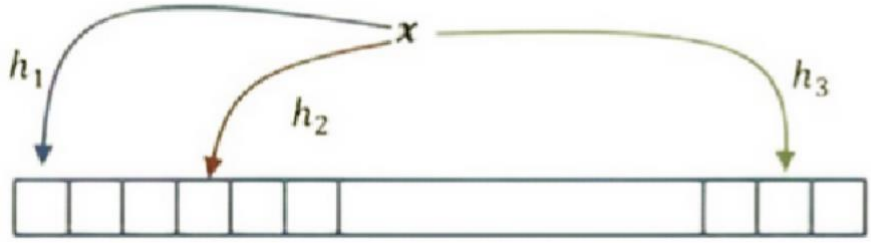
2. Problem Statement:

Suppose there are 1 million movies name in our dataset, and we have to find whether a specific movie we enter is present in a dataset or not? First and very naive approach for this problem is linear search. But if the movie entry is present at the last position of the dataset or what if the enter movie is not present so to reach such a conclusion, it takes to make 1 million comparisons to do which is very high. We can use Binary search to improve the time complexity. But in binary search first we need to sort the movies name in alphabetical order but then also it will also take O (log (1000000)) time. The time complexity is reduced compared to linear search but still the insertion operation to binary search is costly as data need to be sorted. To improve more we have another approach hashing. Hash-table has a constant lookup time of O(1). It mostly depends on the hash function used. Hash table solves the problem of time, but we still need to store the items in the hash table which is space consuming. Thus, hash requires O(n) space complexity. Though time complexity is improved but still space required is more, to overcome such inefficiency of space complexity we use Bloom filter data structure. Bloom filters don't save the entire movie/value in the hash table instead it uses bit arrays to represent them. False positive cases may occur in bloom filters as more than one items can have the same hash value. Bloom filters uses more than one hash functions to reduce the false positive probability as it is not storing the value of the data items. We can still further reduce the number of false positives if we use multi-layer bloom filters.

3.BLOOM FILTER

3.1 Bloom filter and analysis:

If we use the hashing or sets though time efficient, we've a big concern over the space complexity because it will require O(n) space. In bloom filters we make use of the bit array and more than one hash functions. If we use more than one different hash functions to point that specific item x is within the dataset then we reduce such false positive cases. As shown within the below figure we are using 3 hash functions to store the given value of a key x.



Depiction of Bloom filter data structure

In order to see if the value x is already visited, 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's 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 algorithm 2 represents the insertion and lookup operations in bloom filters.

```
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;
```

3.1.1 Algorithm 1: Insertion in bloom filters

```
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;
```

3.1.2 Algorithm 2: Lookup in bloom filters

3.2 Disadvantages of bloom filter:

The issue with bloom filter is in LOOKUP operation (Algorithm 2) sometimes still returns true even when an element is not present in the dataset. It is called false-positives values. The challenge is to reduce the false positives to get more accurate results from the algorithm 2

3.2.1 Analysis of Bloom filter:

The increasing value of k (number of hash functions) possibly makes it harder for false positives to happen, but it also increases the number of the filled-up positions in the bloom filter array. Our goal is to find the optimal value of k.

3.2.2 False-positive analysis:

Let m = |B| and n elements are inserted. If S has not beeninserted, what is the probability that Lookup (B, S) will return true? Assume hash functions {h1,h2……..hk } are independent and for element S

$P_r[h_i(S) = j] = 1/m$ 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 likelihood 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 the value of m increases, the false positive value reduces. Now we need to choose k to minimize false positives, Let p = $e^{-kn/m}$, then.

$$Log(FalsePositive)$$
$$= log(1-p)^k = klog(1-p)$$

$$= -(m/n)log(p)log(1-p)$$

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

$$k = mlog(2)/n$$

The significant advantage of bloom filters over a traditional time efficient HashMap or set-based approach is improved space efficiency and ease of query. Even though HashMap's and set-based approaches can provide O (1) time complexity, the bloom filter is perfect for scenarios involving a large dataset with tight memory requirements since bloom filters require approximately 10 bits to store a character compared to a byte in traditional set-based approaches. Insertion operations are always O(k) in time, and the query is also O(k) in time. The space complexity of bloom filters is O(m) because they require a bit-array of size m. A typical bloom filter, on the other hand, has its own set of drawbacks. Furthermore, because a bloom filter has no recollection of which bits were changed by which objects, we can only obtain a yes or no answer, and even a yes answer is not always right. As a result, we implemented a layered approach to the bloom filter, as discussed in the following sections.

4 MULTILAYER BLOOM FILTER:

4.1Analysis and design:

Bloom filters do not save the objects themselves and consume less space than the theoretical minimum of one byte per character required to store the data accurately, hence they have an error

rate. They have false positives.[15] So another version of the bloom filter was introduced; it is known as a multi-layer bloom filter in which the number of false positives is decreased more than bloom filters.
In the instance of our multi-keyword search problem, we may partition the full set of keywords into layers and store them in the multi-layer bloom filter.

Each keyword of the strings separated by delimiter is present at different layers, and each layer has a corresponding classic bloom filter associated with them. We denote bloom filters as $BF^{k,m}$ = traditional bloom filter ofk hash functions and m bits array each standard bloom filter
is used to record the data at each layer. So, the overall result will be obtained by using the output of all the layers up to which our keyword is present. Let L be the number of layers, the 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 keyword, suppose insert ai in the ith layer of bloom filter. If any particular ai is not in the ith layer, then we assume S is not present else present.

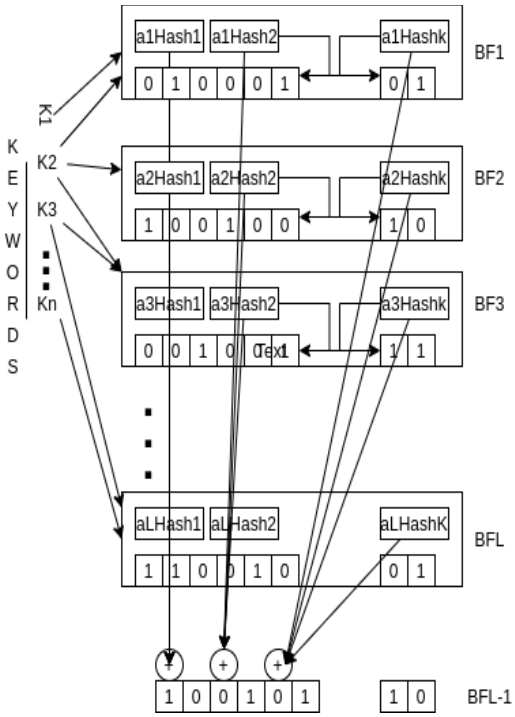


Fig .The principle of MLBF

When a new keyword 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 the 1-layer segment tothe L-Layer segment, as shown in Fig 1.2 above.
Algorithms 3 and 4 describe the steps to Insert and Lookup in a Multilayer Bloom Filter.

Let “L” be the number of layers in the multilayer bloom filter. Let “k” be the number of hash functions.

```
for i=1 to L:
    for j=1 to k
        Addr=H_j[a_i]
        BF_i[Addr]=1
        LAddr[j]=LAddr[j]⊕Addr
    end for;
end for;
for j=1 to k
    BF_{L+1}[LAddr[j]]=1
end for;
```

4.1.1 Algorithm 3: Insert into Multilayer Bloom Filter:

```
//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]⊕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;
```

4.1.2 Algorithm 4.: Lookup in Multilayer Bloom Filter:

1. Get the unique dataset(A) and split it into equal numbers of rows. In proposed system the parts will comprise n appx rows. Let's call them B and C.
2. Make a new dataset consisting of n rows which comprise shuffled rows from the original

dataset. Let's call this T.

3. Insert the values from the dataset B&C into two different bloom filters BF_B & BF_C .
4. Insert the values from the dataset B&C into two different multi layer bloom filters & MBF_B & MBF_C .
5. For every E ∈ T
   if E ∈ BF_B and E ∈ BF_C
   false_postiive_bloom++
   if E ∈ MBF_B and E ∈ MBF_C
   false_postive_multi_bloom++
6. Return false_positive_bloom,false_positive_multi_bloom

4.1.3 Algorithm 5: False positives detection:

Since all the elements are unique, if a value is present in B and C simultaneously, it is evident that it's a false positive.

5.PROPOSED SCHEME:

Although we have different searching methods like linear search, binary search, hashing etc. To improve performance, we can use bloom filter data structure. Bloom filters don't save the entire value in the hash table instead it uses bit arrays to represent them. . Although the Bloom filter outperforms the traditional approaches, it does produce some false-positive results. So another version of bloom filter is proposed which is multi-layer bloom filter in which the number of false positives is decreased more than bloom filters in case of our problem based on multi-keyword.
The following steps are followed to perform the operations in our proposed scheme.

- Get the entire keywords dataset from the user or by extraction from the document.
- Each string of the input keywords is separated by a delimiter which will be split accordingly to a list of words.
- The words from the list will be inserted into a multilayered bloom filter in a round-robin fashion.
- Each word from the list is present at different layers, and each layer has a corresponding classic bloom filter associated with them.
- We denote bloom filters as= traditional bloom filter of k hash functions and m bits array.
- Each standard bloom filter is used to record the data at each layer.
- So, the overall result will be obtained by using the output of all the layers up to which our keyword is present as in algorithm 3 and algorithm 4 given.
- Let L be the number of layers, the result of multilayer bloom filter will be like $MLBF^{L,k,m} = BF_1^{k,m}, BF_2^{k,m}, ..BF_L^{k,m}$
- The multilayer bloom filter is indexed accordingly and ready to be searched upon.

6.Accessing the dataset and processing the input:

- The initial dataset used from Kaggle named tmdb_5000 and Netflix titles comprised a set

of keywords of movies in .csv file unique values respectively.

- To determine the false positives in an ideal scenario, we run the following steps through a traditional bloom filter approach with a 0.05 error rate (the ideal lowest error rate for a bloom filter) and a multilayered bloom filter with n traditional bloom layers
- where n is determined uniquely by dividing the number of keywords with the hash count dynamically.

7.Platforms and tools used:

- Python is used for developing the system.
- Python libraries such as csv, time, math, mmh3, os, pandas are used.
- System is being run on Linux environment.
- Dataset from .csv excel file and writing output to .csv excel spreadsheet.

8.Expected outcome:

Proposed plan has the potential to improvethe precision of the existing scheme by a factor of 15-30%. Implementing a dynamic layered approach of a Bloom filter to determine the relevant documents to improve search efficiency. Because of the Bloom filter's properties, this new scheme can be implemented in very lessmemory space compared to traditional approaches and dynamic in its approach to maximize the precision of query. Moreover, the layered approach helps to reduce the false positives count to a greater extent than any traditional approach. The memory usage in this approach is far less compared to any traditional approaches.

9.References:

[1] J. Kim, "On the False Positive Rate of the Bloom Filter in Case of Using Multiple Hash Functions," 2014 Ninth Asia Joint Conference Journal on Information Security, 2014, pp. 26-30, DOI: 10.1109/AsiaJCIS.2014.32.

[2] Thomas Mueller Graf and Daniel Lemire. 2020. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. ACM J. Exp. Algorithmics 25, Article 1.5 (2020), 16 pages. DOI:https://doi.org/10.1145/3376122.

[3] Pranav Byali, Md Zaid S Bevinahalli, Vaishnavi Chavan, 2020, Bloom Filter, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCAIT – 2020 (Volume 8 – Issue 15).

[4] (Géraud, Lombard-Platet, & Naccache, 2019) a hash table data structure for detecting duplicate on data streams.