Transactions on Computational and Scientific Methods | Vo. 3, No. 1, 2023
ISSN: 2998-8780
https://pspress.org/index.php/tcsm
Pinnacle Science Press

Enhancing Spam Detection: An Improved Bloom Filter Approach with Reduced False Positive Rates

Sebastian Nelson
Yeshiva University, New York City, USA
Sebastian889@gmail.com

Abstract: In the age of big data, the prevalence and impact of spam are increasing significantly. The anti-spam software integrated into many corporate email systems is becoming increasingly ineffective at managing the growing volume of spam. This paper explores an enhanced spam detection method based on the Bloom Filter. This method is straightforward, efficient, easy to implement, and exhibits a significantly lower false positive rate compared to the traditional Bloom Filter.

Keywords: Bloom Filter; Hash function; False positive rate; Spam.

1. Introduction

Bloom Filter, proposed in 1970 by Burton Howard Bloom, is a very space efficient probabilisticdata structure designed to determine whether an element exists in a set. If we want to determine whether an element is in a set or not, the traditional solution is to save all the elements and then determine by comparison. Chain table, tree and other data structure are all this kind of thinking. However, as the element in the set increases, it needs more and more storage space, and the retrieval speed is getting slower (O(n), O(logn)).

The advantage of Bloom Filter is that both spatial efficiency and query time efficiency are much better than that of ordinary algorithms. Hash tables can also be used to determine whether an element is in a set and the retrieval time efficiency is very high, but the same problem can be achieved in Bloom Filter with only 1/4 or 1/8 or less space complexity of hash tables. An element can be inserted in Bloom Filter, but an existing element cannot be deleted. Bloom Filterwon't have false negative at all, and all the elements that exist in Bloom Filter can be looked up.One drawback of Bloom Filter, however, is that it can produce false positive, and the more elements are, the greater the false positive rate will be. The algorithm proposed in this paper is to reduce the false positive rate of Bloom Filter.

2. An Improved Bloom Filter

2.1. Classic Bloom Filter

An empty bloom filter is a bit array with m binary bits, and each bit array is initialized to 0. There is a set and define that there are k different hash functions. A value greater than or equalto 1 and less than or equal to m is calculated after each element is substituted into a hash function, that is, and each element can get k value by substituting k hash functions. Of course, the perfect hash function is one of those hash functions that can randomly and uniformly hashall the elements to n different positions.

To add an element into a bloom filter, namely, to get k value by substituting the element in k hash function, and set the bit corresponding to the k value in the bit array to 1.

To query an element, namely, to determine whether it is in a set, get k value by substituting the element in k hash function. If the bits in this bit array corresponding by k value are all 1, then the element is in the set; if neither one is 1, then the element is not in the set.

Deleting an element is not allowed because the corresponding k bits will be set to 0 in that case, and there may be bits corresponding by other elements, and therefore, this is absolutely forbidden.

The algorithm code is described as follows [1]:

```
Procedure Bloom Filter (set A, hash functions, integer m)
          returns filter
     filter = allocate m bits initialized to 0
    foreach ai    in A:
      foreach hash function hj:
              filter[hj(ai)] = 1
          end foreach
    end   foreach
    return filter
```

$: X^* \{m1..m\}$.

The algorithm code design is as follows[2]:

```
Procedure BloomFilter(set A, hash functions, integer m)
             returns filter
       filter = allocate m bits initialized to 0
       foreach ai in A:
         foreach hash function hj:
               filter[hj(ai)] = 1
        end foreach
        foreach hash function fj:
              filter[fj(a*i)] = 1
        end foreach
      end foreach
      return filter
```

2.2. Mathematical Analysis

One notable feature of the classic Bloom Filter is that there is an obvious trade-off between the size of the filter and the false positive rate. Obviously, after inserting n elements into a filter with the size of m by using k hash functions, the probability of a particular bit remaining 0 is p_0, as follows:

$$p_0 = \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-\frac{kn}{m}} \tag{1}$$

Let's assume that the perfect hash function can evenly hash elements in space $\{1.m\}$. In practice, good results are achieved in virtue of MD5 and other hash functions[10]. Thus, the probability of false positive (i. e., the probability of all k-bits being placed before 1) is p_{err}, as follows:

$$p_{err} = (1 - p_0)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

$$\approx \left(1 - e^{-\frac{kn}{m}}\right)^k = e^{k\ln(1-e^{-kn/m})} \qquad (2)$$

Regard the Formula (2) as a function of k $f(k)$, and find the best value, we can get the lowest false positive rate at k $\ln 2\frac{m}{n}$. When the false positive rate is set, the space-to-element ratio m / n is obtained by Formula (2):

$$\frac{m}{n} = \frac{-k}{\ln\left(1 - e^{\frac{\ln p_{err}}{k}}\right)} \qquad (3)$$

In practical use, there is no infinite increase in the number of hash functions, usually using a small number of hash functions. Because the computational overhead of every hash function is constant, the incremental benefit of adding a new hash function is reduced after a certain threshold, as shown in Fig. 1.

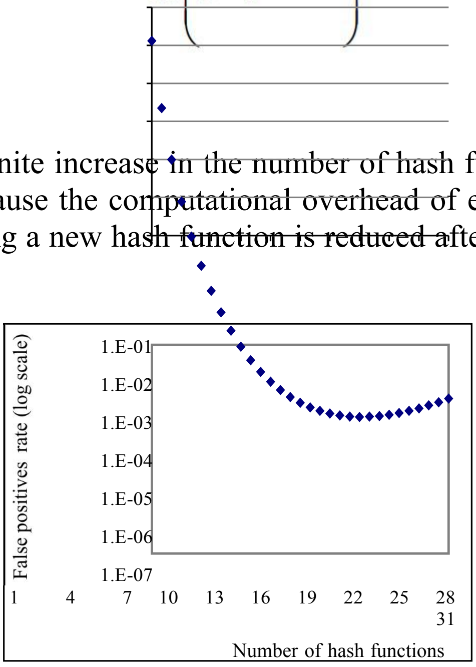

Fig 1. The Relationship between the False Positive Rate and the Number of Hash Functions. When Bloom Filter space element ratio (m/ n = 32),22 hash functions are used to minimize the false positive rate. When the number of hash functions exceeds 10, each additional hash function does not significantly reduce the false positive rate.

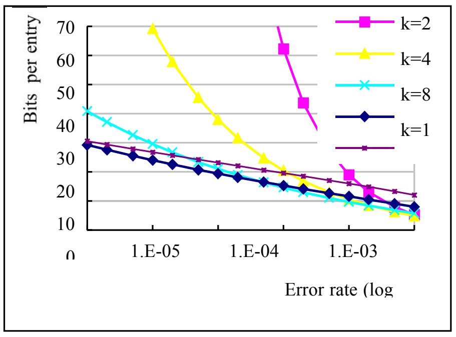

Fig 2. The Size of Bloom Filter (m / n) Acts as a Function of the False Positive Rate. Different curves represent different numbers of used hash functions. Obviously, when m / n <30, using 32 hash functions is no more beneficial than just using 8 in conflict rates.

At the same time, the relationship between Bloom Filter's memory allocation and false positive rate can be obtained by Formulas (2) (3), as shown in Fig. 3.

Obviously, as shown in Figs. 2 and 3 above, the better results will not be achieved as the hash functions get more in actual application. Therefore, we can not reduce the false positive rate by increasing the number of hash functions infinitely, and increasing the number of hash functions will not significantly reduce the false positive rate, but increase the overhead of time and space. One feasible method is to reduce the conflict rate by extracting the eigenvalues from the elements and hashing the eigenvalues in virtue of the algorithm which is put forward in this paper.

The time complexity of the improved algorithm is O (k + d), which is still constant, and there is not much increase compared with the time complexity O (k) in the classical Bloom Filter. However, the cost in space is not increased much, so we can add it properly according to the reality when we use it. Generally speaking, the number of hash function d should be set to be smaller than the number of original hash function k. If d/ k = 1/3, then increase the bit array size by about 1/3.

It is necessary to explain that, through the above algorithm description in Part 2 and Part 3, we can also see that the improved algorithm, although it greatly reduces the false positive rate, does not increase the difficulty of implementation of the algorithm, which shows that the improved algorithm is simple and efficient.

In addition, it is important to note that the total length of Bloom Filter and the length of each part can be calculated and estimated. All can be obtained based on the Formula (3). As long as the error rate is fixed, the length of m can be obtained from the Formula (3). The improved algorithm can obtain the length of {1..m1} and {m1..m}, respectively.

3. The Improved Bloom Filter for Is Used for Spam System

In the field of e-mail systems, there is a huge amount of spams. The main anti-spam technologies used in today's e-mail system are reverse DNS resolution (PTR), blacklist, white list, real-time blacklist (RBL), gray list (GrayList), Bayesian intelligent analysis, SPF of e-mail sender address technology and similar Microsoft-provided anti-spam Send ID scheme and so on. These technical solutions are often used synthetically, and an e-mail system uses all of the above schemes to achieve the best effect of filtering spams. Due to the adoption of many schemes, anti-spam system will take up a large number of system resources, a letter has to pass 7 to 8 different solutions analysis filtering intercept. Many large-scale e-mail systems send and receive large amounts of e-mails every day, so the efficiency of anti-spam e-mail system is being highly valued increasingly[3].

Common e-mail system uses black-and-white list mechanism to manage e-mail, and the normal e-mails are put into the white list, while the spams are marked into the blacklist. Some e-mail systems also use real-time blacklists (RBL) or gray lists (GrayList) technology schemes. Whether it's a blacklist, a white list, or a real-time blacklist (RBL) or GrayList, the lists in these technical solutions need to be stored in the database[4].

Retrieve a database to look up if the e-mail received belongs to the list set by the system. However, the time efficiency of the database is extremely low, and because the large-scale e- mail system will keep receiving new mail, it will cause too much pressure on the database, and the resource occupancy is extremely high. E-mail systems require rapid identification of e-mail addresses to filter out spams[5]. Consequently, it has become increasingly impractical to determine whether the mail received belongs to the list set up by the system or not by directly querying the database, and many e-mail systems apply the Bloom Filter to query the list. Bloom Filter accesses much faster than a database. The main reasons why the Bloom Filter accesses much faster than a database are as follows:

First, differences in storage modes and access modes make Bloom Filter access faster than a database. Database is usually stored in the hard disk or the disk array composed by hard disks, and the access efficiency of hard disk is much lower than that of memory. In addition, large databases often require cloud storage. Cloud storage requires remote access over the network, while the remote access is always delayed. These have led to time-consuming and inefficient access to the database. Bloom Filter is a compact data structure with much less storage than a database and can generally be stored in memory. This ensures that Bloom Filter not needs to access both the hard disk and the cloud when it is stored in local memory, greatly increasing access speed.

Second, from the analysis of lookup algorithms, the different search algorithms make Bloom Filter access faster than a database. Bloom Filter uses hash lookup, and the lookup efficiency of the algorithm is very high. While databases are mostly indexed, they are much less efficient.

The access speed of the classical Bloom Filter is relatively fast, but there is weakness of false positives. If you can reduce the false positive rate of Bloom Filter, it will effectively improve the user sensitivity of e-mail system. The improved Bloom Filter proposed in this paper can reduce the false positive rate effectively. In particular, since the algorithm presented in this paper extracts eigenvalues, it will effectively reduce the false positive rate in the case of many similar elements. However, people who send spams often send a lot of spams, and the similarity of these e-mails is very high, which will greatly increase the false positive rate of the classic Bloom Filter. If adopt the improved Bloom Filter algorithm proposed in this paper in the spam systems, there will be lower false positive rates.

In practice, the improved Bloom Filter algorithm proposed in this paper can be used to judge spams. The spams are stored in Bloom Filter, and the spams are quickly identified with the improved Bloom Filter algorithm. For all e-mail addresses which are identified as spam, as shown in Table 1, each of these elements can be hashed into the area of {1..m1} of a bit array ofm bits using the k hash function, as the same in the classic Bloom Filter algorithm, and meanwhile, the eigenvalues are simultaneously extracted, as shown in Table 1.

Table 1. Element Hash and Eigenvalue Extraction. (The e-mail addresses are the element, and some ASCII codes are the eigenvalue)

a	value of a	eigenvalues	hash value ($h_i : X \quad \{1..m1\}$)
a1	x1x2x3.x4x5x6.x7x8x9.@xx...x.com	x1,x4,x7	h1(a1), h2(a1), …,hj(a1),... ,hk(a1)
a2	y1y2y3.y4y5y6.y7y8y9.@yy...y.net	y1,y4,y7	h1(a2), h2(a2), …,hj(a2),... ,hk(a2)
a3	z1z2z3.z4z5z6.z7z8z9.@zz...z.org	z1,z4,z7	h1(a3), h2(a3), …,hj(a3),... ,hk(a3)
a4	u1u2u3.u4u5u6.u7u8u9.@uu...u.net	u1,u4,u7	h1(a4), h2(a4), …,hj(a4),... ,hk(a4)
a5	http://www.v1v2v3.v4v5v6.v7v8v9.com	v1,v4,v7	h1(a5), h2(a5), …,hj(a5),... ,hk(a5)
…	…	…	…

For each element's eigenvalue $a^*,_i$a separate d hash function is used to hash to the area of {m1..m}, as shown in Table 2.

In querying, first compute the e-mail address using the k hash function and get k value of {1..m}, if all the bits corresponding to k value are 1, then the d hash functions corresponding to their eigenvalues are calculated to get the bit corresponding by d value, and if all the bits corresponding to d value are 1, then the e-mails are judged as spams.

By doing so, the spam can be determined more accurately and quickly, and the accidental damage to the normal email similar to the spam can be avoided as much as possible.

Table 2. Hash Eigenvalue Again

a*	value of a*	*hash value(*$f_j : X \quad \{m1..m\}_*$*)*
a*1	x1x4x7	f1(a1), f2(a1), …,fj(a1),... ,fd(a1)
a*2	y1y4y7	f1(a2), f2(a2), …,fj(a2),... ,fd(a2)
a*3	z1z4z7	f1(a3), f2(a3), …,fj(a3),... ,fd(a3)
a*4	u1u4u7	f1(a4), f2(a4), …,fj(a4),... ,fd(a4)
a*5	v1v4v7	f1(a5), f2(a5), …,fj(a5),... ,fd(a5)
...	...	...

In practice, there are a variety of methods for eigenvalues. For example, take the domain name ASCII code or Unicode code, or take the length of the domain name, or use some arithmetic operations such as midsquare method, division method, or simply use the random function to get the random value.

4. Summary

In practice, the classical Bloom Filter can't reduce the false positive rate by adding hash function infinitely, but it needs to make appropriate trade-off between space, time and false positive rate. At the same time, increasing the number of hash function largely does not significantly reduce the effect of false positive rate after a certain threshold, but increases the cost of time and space. In fact, the false positives are mostly caused by the similar elements, and the similarity between the similar elements can be greatly reduced by extracting the eigenvalues. Therefore, the improved Bloom filter algorithm of hashing eigenvalues again proposed in this paper can reduce the false positive rate effectively.

At the same time, the improved Bloom Filter algorithm inherits the advantages of time and space efficiency of the classical Bloom Filter. The implementation of the algorithm is also very concise.

Moreover, this paper also introduces the application of improved algorithm in the query of spams. Common e-mail systems use blacklist and white list schemes, and some also use real- time blacklist (RBL) or gray list (GrayList) schemes. If the improved Bloom Filter algorithm proposed in this paper is used, it can not only have the running efficiency of the classical Bloom Filter, but also have the lower false positive rate than that of the classical Bloom Filter.

References

[1] B. H. Bloom, Space/time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, vol. 13 (7). 1970, pp. 422-426.

[2] M. Jiang, C. Zhao, G. Xiang, A modified algorithm based on the bloom filter. The 6rd International Congress on Image and Signal Processing (CISP 2013), Hangzhou, June 2013, p. 1087–1091

[3] Li Zhuang, John Dunagan, Daniel R. Simon,etc. Characterizing Botnets from Email Spam Records[C] First USENIX Workshop on Large-Scale Exploits and Emergent Threats, April 15, 2008, San Francisco, CA, USA, Proceedings. 2008.

[4] Mengjun Xie, Heng Yin, Haining Wang. An effective defense against email spam laundering[C] Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006. ACM, 2006.

[5] El-Sayed M. El-Alfy, Radwan E. Abdel-Aal. Using GMDH-based networks for improved spam detection and email feature analysis [J]. Applied Soft Computing, 11(1):477-488.