K-Divided Bloom Filter Algorithm and Its Analysis

Xiao-Guang Liu, Jun Lee,Gang Wang, Guang-jun Xie,Jing Liu
Dep. of CS, Nankai University, Tianjin, China, 300071
e-mail: liuxg74@yahoo.com.cn, wgzwp@163.com

Abstract

By using a bit vector and a set of hash functions to represent data set, Bloom Filter can query a given data effectively. Bloom Filter can be used to determine an element belongs to data set or not. Split Bloom Filter is amelioration to the Bloom Filter, which use a \(S \times N\) bit matrix to represent data set. In distributed systems, if the number of the elements increases continually, the increasing error rate of Bloom Filter will make the representation nonsensically. Split Bloom Filter can only weaken this problem. In this paper, a new kind of Bloom Filter, named as K-Divided Bloom Filter, is presented. Compared with Split Bloom Filter, it can reduce space and time spending and has a resembling or better performance. K-Divided Bloom Filter gets better tradeoff among error rate, space and time.

1. Introduction

Bloom Filter was first described in 1970[1], but the theory of Bloom Filter has never been improved until the presentation of Random Filter in 1990[2]. The only difference between Bloom Filter and Random Bloom Filter is the probability of hash functions. The addressable bits generated by hash functions for a given key are all different in Bloom Filter, while they are independent from each other in Random Filter. Random Filter is superior to Bloom Filter in false drop probability and average testing time. The major variations of bloom filter include Segment Filter [3], Counter Bloom Filter [4], Compressed Bloom Filter [5], spectral bloom filter and Split Bloom Filter[6,7]. These are solving data deleting, compressing and increasing problems separately.

2. Related Works

Bloom Filter and Random Filter share the same basic operations. They both have two components. One is N-bits vector, named V, which represents a data set, \(A = \{a_1, a_2, \ldots, a_n\}\). In initial state, all bits in V are 0. The other is a group of hash functions. Here, the number of these functions is d. During the representation of Bloom Filter, the key of every element in A will be hashed into V by all the d functions. As results, the relevant bits will be set as 1, while other bits remain 0. Figure 1 shows the process.

![Figure 1. Representation of Bloom Filter](image)

If all n keys have been represented into V, the initialization is terminated. Now we can test a given key to determine whether it belongs to a certain data set or not. The process can be described as follows:

Firstly, using d hash functions to distribute d bits into V. Then, all the d bits are checked. If all the d bits are 1, the key will be accepted by the filter. Otherwise, if there is at least one bit appears 0, the key will be filtered out.

For a given key, the d bits are generated by different hash functions in Bloom Filter. However, in Random Filter, the d hash functions are completely independent from each other. The hash values can distribute randomly in V.

Obviously, it’s possible that some wrong keys may be regarded as a member of the data set. The probability of this error occurring is called the false drop probability. The false drop only occurs when a wrong key is accepted, the phenomenon that a right key is filtered out never happens. Another character of the Bloom and Random Filter is that deleting a key from V is forbidden, because every bit in V may be mapped by different keys.

Recently, Bloom filter was popular. It was used in P2P, distributed storage systems, etc. If the number of
elements in a node’s data set increases continually, the false drop probability will become unacceptable eventually. In order to solve this problem, a new kind of Bloom Filter, named Split Bloom Filter, is presented in reference 7. It uses a $S \times N$ bits matrix to represent data set. If the number of elements in data set exceed a quota which can be calculated in advance, then false drop probability will become unacceptable. We should recalculate the value of s in order to keep false drop probability as low as before. The new $S \times N$ bits matrix will be used to represent data set in Split Bloom Filter.

In typical distributed systems, there are many diverse resources in different nodes. According to the algorithm of Bloom Filter, the global N and d should be kept consistently. Split Bloom Filter shows a tradeoff among space, time spending and error rate. It can solve the problem of data set increasing dynamically. But Split Bloom Filter still has two limitations. Firstly, the cost of space is s times more than before. Secondly, it requires more time to process local hash computation. If the false drop probability exceeds the quota, we have to recalculate s and represent the data set again. In this paper, we present a new kind of Bloom Filter, named K-Divided Bloom Filter, to solve this problem.

3. False Drop Probability and Average Testing Time

There are two important methods to estimate the performance of Bloom Filter, false drop probability and average testing time. For the convenience of discussing, we denote the following symbols, $N$ is the size of vector(in bits), $P_{err}$ is the false drop probability, $T$ is the average testing time, $n$ is the number of keys in data set, $d$ is the number of hash functions, $p$ is the expected probability of a certain bit is 1 after $n$ keys have been mapped into V, $h_i$ is the $i^{th}$ hash function, $t$ is the average time of a hash value comparing.

For Bloom Filter, after a key is mapped into V by $d$ hash functions, the probability that a certain bit is 1 equals to $\frac{d}{N}$. When $n$ keys are all mapped,

$$p = 1 - (1 - \frac{d}{N})^n \quad (1)$$

For Random Filter, after a key is mapped into V by a hash function, the probability of a certain bit is 1 equal to $\frac{1}{N}$. After $d$ hash functions have mapped a key, the probability that a certain bit is 0 equal to $(1 - \frac{1}{N})^d$.

At last, after the $n$ keys are all mapped,

$$p = 1 - (1 - \frac{d}{N})^n \quad (2)$$

If a key which don’t belongs to the data set, named K, is accepted as a member. We can conclude that for every $i$ in $d$, $h_i(K) = 1$. So the false drop probability is defined as

$$P_{err} = p^d \quad (3)$$

The average testing time is defined as

$$T = (d \times p^d + \sum_{i=1}^{d} i \times p^{i-1} \times (1 - p)) \times t \quad (4)$$

According to conclusion in reference 2, Random Filter is superior to Bloom Filter at both false drop probability and average testing time. In latter sections, Bloom Filter, which was used as benchmark, was Random Filter in fact. In this paper, we also adopt this instance.

For Split Bloom Filter, which has a $s \times N$ bits matrix, its false drop probability is

$$S - P_{err} = 1 - (1 - (1 - \frac{1}{N})^d)^s \quad (5)$$

In reference 7, it was proofed that $S_{err}$ is better than $P_{err}$. To get better performance, Split Bloom Filter requires more local space and time spending than Bloom. The average testing time of Split Bloom Filter, which has $S \times N$ bits matrix, is

$$S - T = S \times T = \frac{1 - p^d}{1 - p} \times s \times t \quad (6)$$

4. K-Divided Bloom Filter

K-Divided Bloom Filter can be described as follows. Firstly, we use two vectors to represent the data set $A$, $V_1$ in $N$ bits and $V_2$ in $\frac{N}{k}$ bits. Then we divide the $d$ hash functions into two parts, $d_1$ for mapping into $V_1$ and $d_2$ for $V_2$. The algorithm is defined as follows:

$$V_{(i,j)} = \begin{cases} 1, & \text{if } \forall x \in A, i = 1, 2, \ldots, d_1, h_k(x) = j \\ 1, & \text{if } \forall x \in A, i = 2, \ldots, d_2, h_k(x) = j \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

Here, $V_{(i,j)}$ means the $j^{th}$ bit in the $i^{th}$ vector.
It’s obvious that K-Divided Bloom Filter consists of two common Bloom Filters actually. We can get that

\[ P_1 = 1 - \left(1 - \frac{1}{N} \right)^{\left(\frac{d}{2}\right)} \]  

\[ P_2 = 1 - \left(1 - \frac{k}{N} \right)^{\left(\frac{d}{2}\right)} \]  

The false drop probability and average testing time in this K-Divided Bloom Filter are

\[ K \_ P_{err} = p_1^d \times p_2^d \]  

\[ K \_ T = \frac{1 - p_1^d}{1 - p_1} + \frac{1 - p_2^d}{1 - p_2} \]  

We can conclude that when \( k = N \), \( V_2 \) will play no role in our algorithm. Under this condition, K-Divided Bloom Filter actually is common Bloom Filter with \( \frac{d}{2} \) hash functions and \( N \) bits vector.

5. Performance of K-Divided Bloom Filter

Figure 2 shows the contrast performance of K-Divided Bloom Filter and common Bloom Filter which has \( N \) bits vector and \( d \) hash functions. Since \( N \) is given initially, it means that we use the same size of the data set. Here we use half more spaces and get a much better false drop probability. As shown in figure 3, the number of hash functions, \( d \), will also expert an influence on the performance.

![Figure 2. The contrast on N/n between Bloom Filter and K-Divided Bloom Filter](image)

Figure 3. The contrast on \( d \) between Bloom Filter and K-Divided Bloom Filter

Certainly, false drop probability and average testing time are the most important performance criterions. Figure 4 and figure 5 show the comparison between K-Divided Bloom Filter and Split Bloom Filter, which has \( s \times N \) bits of matrix and \( d \) hash functions.

![Figure 4. The contrast on Perr between Bloom Filter and K-Divided Bloom Filter](image)

In figure 4 and 5, the horizontal axis denotes the value of \( N/n \), and the vertical axis denotes the value of \( S_{perr}/K\_Perr \) and \( S_T/K\_T \) respectively. As shown in figure 4, with the increasing of \( s \), the false drop probability of K-Divided Bloom Filter is no better than that of Split Bloom Filter. But they are close while \( s \) equals to 2. As shown in figure 5, we can conclude that average testing time of K-Divided Bloom Filter is much better than that of Split Bloom Filter, especially when \( s \) is much bigger.

![Figure 5. The contrast on T between Bloom Filter and K- Divided Bloom Filter](image)
It’s common that the resources required by applications will increase dynamically. Considering the tradeoff among false drop probability, average testing time and local node’s memory, we can select the feasible method between K-Divided Bloom Filter and Split Bloom Filter. If the data don’t increase rapidly in a short time, the K-Divided Bloom Filter could weaken the going up of the false drop probability. Figure 6 shows the contrast between Split Bloom Filter and K-Divided Bloom Filter, whose k increases with the size of data set. As shown in Figure 6, the increasing rate on false drop probability of the K-Divided Bloom Filter is lower than that of the Split Bloom Filter.

Cache is an efficient mechanism to improve the performance of router service. In distributed systems, such as P2P and wireless system, caches can be distributed on several routers. Bloom filter is a feasible method to improve system performance in these applications. A simulation is designed to show the application of K-Divided Bloom Filter in cache service.

<table>
<thead>
<tr>
<th>N</th>
<th>KBF</th>
<th>SBF</th>
<th>N</th>
<th>KBF</th>
<th>SBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.0064</td>
<td>0.0008</td>
<td>1000</td>
<td>0.075</td>
<td>0.0382</td>
</tr>
<tr>
<td>200</td>
<td>0.008</td>
<td>0.0118</td>
<td>1500</td>
<td>0.076</td>
<td>0.0766</td>
</tr>
<tr>
<td>300</td>
<td>0.0068</td>
<td>0.0064</td>
<td>2000</td>
<td>0.074</td>
<td>0.0662</td>
</tr>
<tr>
<td>400</td>
<td>0.0078</td>
<td>0.008</td>
<td>2500</td>
<td>0.0864</td>
<td>0.076</td>
</tr>
<tr>
<td>500</td>
<td>0.007</td>
<td>0.0056</td>
<td>3000</td>
<td>0.10006</td>
<td>0.063</td>
</tr>
</tbody>
</table>

There are two cases are considered in the simulation. One is data set is small and growth rate is low. The other is data set is big and growth rate is high. Under these two cases, the false drop probability (P_{err}) of K-Divided Bloom Filter and Split Bloom Filter are shown in figure 7 and 8. In the simulation, N is set to be 8192, and 50,000 random requests are generated to test the performance. As shown in figure 7 and 8, with the number of keys (n) increasing, the false drop probability (P_{err}) of these two methods is closed. Compared with Split Bloom Filter, K-Divided Bloom Filter requires less extra space.

6. The application of K-Divided Bloom Filter

In order to improve the performance of applications based on distributed systems, such as P2P and grid system, it requires a feasible tradeoff among global and local spaces, global communicating time and local process time. For example, Split Bloom Filter reduces the cost of global space and communicating time, which maintain the global consistency of N and d. But Split Bloom Filter need more local spaces and process time. In real environment, local nodes in distributed systems usually have limited resources. Split Bloom Filter can not work as well as expectation. K-Divided Bloom Filter can fit this situation well, because it has better tradeoff among false drop probability, spaces and time spending. If the data increases slowly and gently, K-Divided Bloom Filter can weaken this bottleneck very well.

Under the situation that the data increases rapidly and heavy, combination of Split Bloom Filter and K-Divided Bloom Filter may be a better way. Because K-Divided Bloom Filter has lower false drop probability than the common Bloom Filter and lower increasing rate on false drop probability than Split Bloom Filter. It is possible that the performance of Split Bloom Filter can be improved obviously if we use K-Divided Bloom Filters instead of common Bloom Filters in Split Bloom Filter.

7. The Conclusions

Bloom Filter is effective method to represent large-scale data set and test whether an element belongs to this data set or not. They can reduce the storage spaces and the testing times effectively with an acceptable error rate. This algorithm can only insert a data into the set. It can not delete a data from the data set. If the number of the elements in the data set increases quickly, the error rate of Bloom Filter will make the representation nonsensically a period of time. Split Bloom Filter can weaken this problem, but it brings a large spending of spaces and time to local nodes.

In this paper, a new kind of Bloom Filter is presented. It only uses one and 1/k vector and divides d hash functions into two parts. Considering the tradeoff among false drop probability, spaces and time spending according to application’s requirement, K-Divided Bloom Filter can works well under the condition that data set increases slowly and gently.

Acknowledgements

This paper is sponsored by NSF of China (No.90612001), Science and Technology Development Plan of Tianjin, (No. 043800311, 043185111-14) and Nankai University Innovation Fund and ISC.

Reference


