IEICE TRANS. INF. & SYST., VOL.E100–D, NO.10 OCTOBER 2017

PAPER Special Section on Security, Privacy and Anonymity in Computation, Communication and Storage Systems

Multi-Dimensional Bloom Filter: Design and Evaluation

Fei XU[†a)], ***Member***, **Pinxin LIU**[†b)], **Jing XU**[††], **Jianfeng YANG**[†], ***and*** **S.M. YIU**[†††], ***Nonmembers***

SUMMARY Bloom Filter is a bit array (a one-dimensional storage structure) that provides a compact representation for a set of data, which can be used to answer the membership query in an efficient manner with a small number of false positives. It has a lot of applications in many areas. In this paper, we extend the design of Bloom Filter by using a multi-dimensional matrix to replace the one-dimensional structure with three different implementations, namely OFFF, WOFF, FFF. We refer the extended Bloom Filter as Feng Filter. We show the false positive rates of our method. We compare the false positive rate of OFFF with that of the traditional one-dimensional Bloom Filter and show that under certain condition, OFFF has a lower false positive rate. Traditional Bloom Filter can be regarded as a special case of our Feng Filter.

key words: *Bloom Filter, multi-dimensional matrix, false positive rates*

1. Introduction

In many applications, storing a set of data in a data structure for efficient membership query (whether a particular item is in the dataset or not) is a common and critical problem. Bloom Filter, proposed by Burton Howard Bloom in 1970, is a simple space-efficient randomized data structure for representing a set of data that supports membership query [1]. Bloom Filter (BF) is a popular choice for the applications if a small number of false positives (i.e., an item which is *not* in the dataset, but incorrectly identified as in the dataset) is tolerable. For many applications, space and searching time are more important and outweight this drawback if the probability of having false positives can be made sufficiently small.

Initially, BF was applied to database applications, spell checkers and file operations (e.g. [2]–[4]). In recent years, BFs have received a lot of attention in networking applications, such as peer-to-peer applications, resource routing, security, and web caching [5]–[17]. A survey on the applications of Bloom Filters in distributed systems can be found in [18]. BFs are also being used in practice. For instance, Google Chrome uses a Bloom Filter to represent a blacklist of dangerous URLs. The followings show a few concrete examples in network applications. *String matching*: The core operation of deep packet inspection is to search for predefined signatures in the packet payload. A major step used for this application is string matching. BFs and their variants have been utilized to improve the efficiency of string matching algorithms. *Message authentication*: In large-scale WSNs, BF is applied in communication-efficient message authentication protocol to authenticate messages. Each sensor node is preloaded with a symmetric key and k hash functions. The sink also maintains k hash functions and n keys (n is the number of sensors). The sink constructs n message authentication codes (MACs) using the keys. These MACs are then inserted into the BF. Subsequently, BF can be used to check if a given MAC is valid or not. *Presence of mobile users in WWAN*: In instant messaging (IM) services, one needs to know quickly which mobile users are online. To reduce power consumption and wireless wide area network (WWAN) access costs, BF can be applied to store all mobiles user that are present in order to answer the query.

1.1 Bloom Filter

A traditional BF is a vector A of m bits, initially all set to 0, for representing a dataset $S = x_1, x_2, \ldots, x_n$ of n elements. The BF uses k independent hash functions $h_1, h_2, \ldots, h_k$, each with range of $\{0, \ldots, m-1\}$. A BF is constructed as follows. Each element x in S is hashed by the k hash functions. All bits at positions $h_i(x)$ in A are set to 1. A particular position in the vector A may be set to 1 multiple times, but it does not matter, i.e., it is set to 1 anyway. In the query phase, to check if an element y is in S or not, we check the bits at position $h_i(y)$ for all $i = 1, 2, \ldots, k$. If any of these bits are 0, the element is definitely not in the set. Otherwise, either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

1.2 False Positive Rate

False positive rate is an important issue for Bloom Filter. Intuitively, the more elements added to the set, the larger the probability of having false positives, on the other hand, the larger the array A, the lower the chance of having false positives. Thus, the false positive rate is determined by the following parameters:

1. Number of elements (n) added to the Bloom Filter: In most cases, this parameter is defined by the application and, thus, cannot be controlled.

Manuscript received January 18, 2017.
Manuscript revised May 25, 2017.
Manuscript publicized July 21, 2017.
[†]The authors are with Law School, Renmin University, China.
[††]The author is with Tsinghua University, China.
[†††]The author is with The University of Hong Kong, Hong Kong.
a) E-mail: xufei@iie.ac.cn
b) E-mail: liupinxin@263.net (Corresponding author)
DOI: 10.1587/transinf.2016INP0022

2. Number of bits used in a Bloom Filter (m): We can set m to be larger in order to decrease the probability of having false positives but a large m may imply more storage.
3. Number of hash functions (k) used to create the Bloom Filter: The larger the value of k, the higher the processing overhead (CPU usage) especially when the hash functions need to perform complex operations.

1.3 Contributions

The contributions of this paper can be summarized as follows:

- *Multi-Dimensional Storage Filter*: we propose to use a multi-dimensional storage Filter to extend the traditional one-dimensional Bloom Filter.
- *Four different mappings*: we propose three different kinds of implementations for our proposed multi-dimensional bloom Filter (we refer it as Feng Filter): One First Feng Filter (OFFF), Whole One Feng Filter (WOFF), and Function Feng Filter (FFF). We provide the design and details of the implementations. Users can choose one of the three Filters to fit their own application(s).
- *Analysis of false Positive rates*: We formally analyzed the false positive rates of Feng Filter with respect to the three implementations. We use OFFF as an example and compare its false positive rate with that of the traditional Bloom Filter. We show that under certain condition, OFFF has a lower false positive rate.

The rest of the paper is organized as follows. Section 2 presents the basic idea, operation and design of Feng Filter. Section 3 describes three different implementations of Feng Filter. Section 4 analyzes the positive rates of different implementations of the Feng Filter. Section 5 compares the false positive rate of OFFF (the basic implementation of the Feng Filter) with the traditional Bloom Filter. Section 6 concludes the paper.

2. Basics of Feng Filter

We first discuss the basic idea behind the multi-dimensional bloom Filter and the preconditions. Then, we provide the details of the basic operations.

2.1 Basic Idea

Assume that we have n elements in the dataset. In Feng Filter, using k hash functions, the n elements are mapped to a storage matrix M with N dimensions of a total size of m bits (i.e., the number of entries in one dimension is $m^{1/N}$. The mapping is based on the followings: k hash functions, N-dimensional storage matrix, a perfect hash location adjust function L and the given mapping relation. There are four different mapping methods, which we will discuss in the next section. In each dimension, the position being selected is based on the perfect hash location adjust function L (described below), and the k hash functions. When there is a position in M that meets the required mapping function, then this position will be set to 1 from 0. Nothing will be done otherwise. Figure 1 shows the basic idea of the Feng Filter.

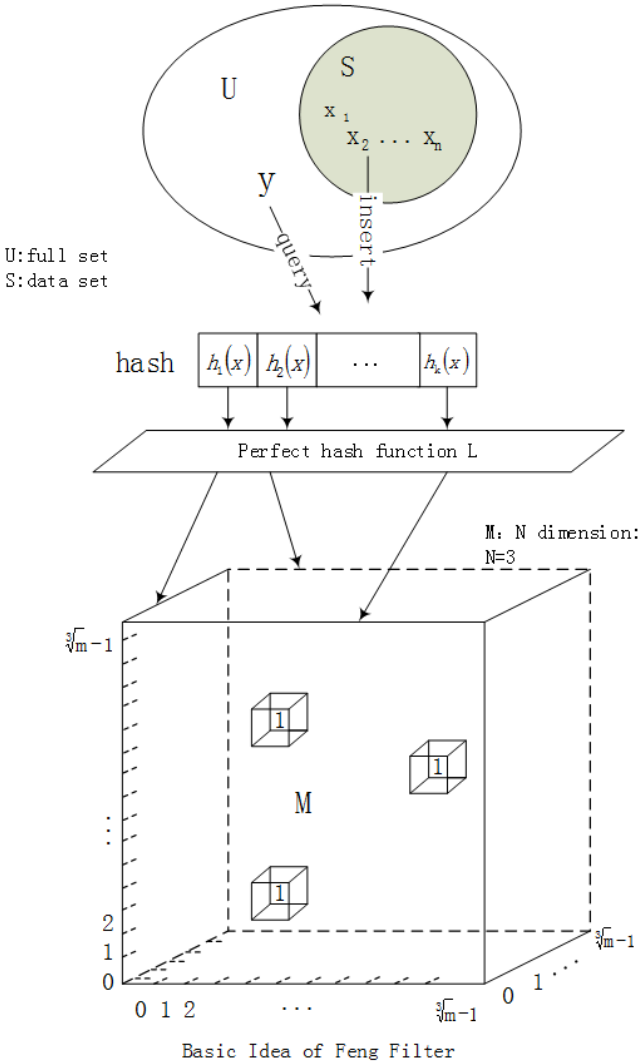

Fig. 1 Basic idea of the Feng Filter.

There are several requirements that we need to make them clear about the relation between original data set and the data in set S.

1. The relation between original data set and the data in set S. Elements are given based on the actual application and can be transformed into the domain for the hash functions.
2. Independence of the k hash functions. All given k hash functions are independent. For each of them, they are equally likely to map a given element to any value in the range $[0..m-1)]$ with equal probability.
3. Bits in the multi-dimensional storage matrix M can

only be set to 0 or 1. Before the elements in s are inserted, all bits in M are set to 1.

4. Multi-dimensional storage matrix M. In real implementation, one can determine N, the dimensions of the matrix M. In our illustration, we use a fix N, say 2.
5. Perfect hash location adjust function L. For k hash results (assuming $k \leq N$), they are divided into N groups (dimensions). For each group (dimension), there are k/N values (mapped to positions using L). The range of the results of function k, L is restricted to $[0, (m^{1/N} - 1)]$ (by performing an mod $m^{1/N}$ operation after dividing into N groups). For example, if $k = 6$ and $N = 3$, we will divide the hash values into 3 groups with 2 elements in each group as follows. We sort the k hash values. The smallest is allocated to the first group, which represents the first position in the first dimension, labelled as $L_{1,1}$. The second smallest is allocated to the second group (representing the first position of the second dimension, labelled as $L_{2,1}$). The N-th smallest is allocated to the N-th group as the first position of the N-th dimension, labelled as $L_{N,1}$. The $N + 1$ smallest will then be allocated to the first group again (representing the second position of the first dimension), and so on. Totally there will be k/N elements in each group. Note that all values will be made to fall in the range $[0, m^{1/N} - 1]$.
6. Other basic assumptions. The dimension N, hash function number k, storage matrix M with the number of entries m in each dimension, and original dataset S and element number should follow the basic requirements: (i) N, k, m, n are all positive integers; and (ii) k is a multiple of N.

2.2 Basic Operations

There are two basic operations in Feng Filter: Insert and Query. When you need to insert element x into the Feng Filter matrix M of Feng Filter, we follow the following steps.

1. Apply k hash functions on x, and get k hash results.
2. For the k hash results, use the perfect hash location adjust function L to get k values to be mapped into the positions in N dimensions.
3. Put k/N positions of the N dimensions as input, and based on the mapping function, we obtain the positions in the N dimensions.
4. Set each selected bit position of matrix M to 1 based on the results of the related function.

Query is to decide if a given element y is in the original data set or not. We can do that by checking the bit values of the corresponding positions after applying the mapping function to y in our Feng Filter. The steps are as follow.

1. Apply k hash functions on y to get k hash results.
2. For the k hash results, use the perfect hash location adjust function L to get k positions, and then divide the k positions into N groups, and each group will be mapped to the first dimension, second dimension, and so on. There are k/N values in each group, which means k/N positions in each dimension.
3. Put k/N positions of the N dimensions as input, and based on the mapping function, we will get the positions of the N dimensions. Check the related position in the matrix M to see if all bits in these positions are 1, if the answer is no, that means element y is definitely not in the dataset S. If the answer if yes, that means under the reasonable false positive rate, this element is in S.

3. Detailed Mapping of Feng Filter

Recall that for each element x in S, we apply k hash functions to obtain k values, then we divide these values into N groups using the function L (the smallest one is allocated to the first group etc., see the above description): $\{L_{1,1}, L_{1,2}, \ldots, L_{1,k/N}\}$, $\{L_{2,1}, L_{2,2}, \ldots, L_{2,k/N}\}$, $\ldots$, $\{L_{N,1}, L_{N,2}, \ldots, L_{N,k/N}\}$. Feng Filter has three different implementations based on three different mapping algorithms, which are described below.

3.1 One First Feng Filter, OFFF

From the hash values, we set the bit at $(L_{1,j}, L_{2,j}, \ldots, L_{N,j})$ in M as 1 for all $j = 1, \ldots, k/N$, i.e., we set k/N bits as 1. If $N = 1$, this is the traditional Bloom Filter.

3.2 Whole One Feng Filter, WOFF

In this version, we cross over the hash values and set all the following positions into 1:
$M(L_{1,1}, L_{2,1}, \ldots, L_{N,1}), M(L_{1,2}, L_{2,1}, \ldots, L_{N,1}), \ldots,$
$M(L_{1,N}, L_{2,1}, \ldots, L_{N,1})$
$M(L_{1,1}, L_{2,2}, \ldots, L_{N,1}), M(L_{1,2}, L_{2,2}, \ldots, L_{N,1}), \ldots,$
$M(L_{1,N}, L_{2,2}, \ldots, L_{N,1})$
$\ldots$
$M(L_{1,1}, L_{2,N}, \ldots, L_{N,N}), M(L_{1,2}, L_{2,N}, \ldots, L_{N,N}), \ldots,$
$M(L_{1,N}, L_{2,N}, \ldots, L_{N,N})$

In this mapping, we set more bits to be 1. We name this type of Feng Filter as Whole One Feng Filter (WOFF). Note that in N-dimension structure, the number of bits to be set is fewer than the traditional Bloom Filter (for example, if $N = 2$, OFFF only sets $k/2$ bits while the traditional Bloom Filter will set k bits, intuitively, we have rooms to set more bits, thus we have WOFF.

3.3 Functional Feng Filter, FFF

This can be a generic scheme for N-dimensional Bloom Filter. The mapping relation of Functional Feng Filter is based on any specific function, decided by the designer or user. All the three Feng Filters we discussed above can be regarded as specific cases of FFF. We recommend the mapping relation of FFF to be a strong relation too. In this mapping

relation, Because during the setting process of setting the N dimensional storage matrix M as 0 or 1, the mapping relation is based on the specific function, So we name this type of Feng Filter as Function Feng Filter (FFF).

4. False Positive Rates

In this section, we analyze the false positive rates of the proposed Feng Filter. Take One First Feng Filter (OFFF) as an example, we will present the calculation of false positive rate. And we will give out the results of the other three implementations of Feng Filter.

4.1 OFFF

The probability for one bit in the N-dimensional matrix M to be set is $1/m$. The probability of it being 0 is $1-1/m$. For each element, we will set k/N bits. Thus, for each element inserted, the probability of a bit being 0 is $(1-1/m)^{k/N}$. If we have inserted n elements, then the probability of a bit being 0 is $(1-1/m)^{nk/N}$. Thus, the probability that this bit is set to 1 randomly is $1-(1-1/m)^{nk/N}$. Finally, if all k/N bits corresponding to an element y to be all set is:

$$f_{OFFF}(m,n,k,N)=\left(1-\left(1-\frac{1}{m}\right)^{\frac{nk}{N}}\right)^{\frac{k}{N}}$$

4.2 Other Implementations

The following probabilities are stated without derivations.

$$f_{WFFF}(m,n,k,N)=\left(1-\left(1-\frac{1}{m}\right)^{n\left(\frac{k}{N}\right)^{N}}\right)^{\left(\frac{k}{N}\right)^{N}}$$

$$f_{FFF}(m,n,k,N)=\left(1-\left(1-\frac{1}{m}\right)^{nt}\right)^{t},$$

where t is the number of bits to be set.

5. Comparison of Feng Filter with Bloom Filter

We will use One First Feng Filter as an example to compare the false positive rate of Feng Filter with that of the one-dimensional traditional Bloom Filter. For One First Feng Filter, we claim that the false positive rate is smaller than that of Bloom Filter provided

$$0<\left(1-\frac{1}{m}\right)^{nk}<\frac{1}{2}.$$

Recall that the false positive rate of OFFF is

$$f_{OFFF}(m,n,kN)=\left(1-\left(1-\frac{1}{m}\right)^{\frac{nk}{N}}\right)^{\frac{k}{N}}$$

When $N\in[1,k]$, we want to make sure that $f_{OFFF}(m,n,k,N)$ is monotonic. Here, we set $t=k/N$ (the number of bits to be set per element) and $b=\left(1-\frac{1}{m}\right)^{n}$. Note that $t\in[1,k]$ and $b\in(0,1)$. Then, we have:

$$f_{OFFF}(m,n,k,N)=(1-b^t)^t \text{ and}$$

$$f'_{OFFF}(m,n,k,N)\mid_N= f'_{OFFF}(b,t)\mid_t \times\frac{dt}{dN}$$

And $\frac{dt}{N}=-\frac{k}{N^2}$. Differentiating f_{OFFF} to get

$$f'_{OFFF}=e^{t\ln(1-b^t)}\left(\ln\left(\ln(1-b^t)\right)-\frac{tb^t\ln b}{1-b^t}\right)$$

Because, $e^{t\ln(1-b^t)}>0$, we set

$$u(b,t)=\ln(1-b^t)-\frac{tb^t\ln b}{1-b^t}$$

$$u'(b,t)\mid_t=\frac{b^t\ln b\times(2b^t-2-t\ln b)}{(1-b^t)^2}$$

Now, put $t=k$ into $u(b,t)$, we have:

$$u(b,k)=\ln(1-b^k)-\frac{kb^k\ln b}{1-b^k}$$

Finally, it is clear that:

$$u(b,k)=\begin{cases} > & 0, \quad b^k\in(0,\frac{1}{2}) \\ = & 0, \quad b^k=\frac{1}{2} \\ < & 0, \quad b^k\in(\frac{1}{2},1) \end{cases}$$

We can see that, if $b^k\in(0,\frac{1}{2})$, then $f_{OFFF}(m,n,k,N)$ is decreasing when $t=k$, $N=1$. That is, under this condition, the false positive rate of OFFF is lower than that of the traditional bloom Filter.

6. Conclusions

In this paper, based on the basic idea of Bloom Filter, we extend it to use multi-dimensional matrix instead of using only one-dimensional array. We call our Filter the Feng Filter. From the perspective of the storage structure, the Bloom Filter can be regarded as a special case of the Feng Filter with only one dimension. Our paper gives the definitions and details of this proposed multi-dimensional Filter. Depending on the mapping relationship, we propose three kinds of implementations for Feng Filter, namely OFFF, WOFF, FFF. We also compared the false positive rates of OFFF with that of the traditional Bloom Filter and show that under certain conditions, OFFF is better than the traditional Bloom Filter. We believe that our three variations can provide more flexibility to users depending on the applications.

Acknowledgments

This work is supported by Beijing Natural Science Foundation (4164089) and the Major Project of Key Research Base for Humanities and Social Sciences, Education Ministry of China, "Criminal Rule of Law in Internet Security" (15JJD820011).

References

[1] J.K. Mullin, "Optimal semijoins for distributed database systems," IEEE Trans. Softw. Eng., vol.16, no.5, pp.558–560, 1990.
[2] J.K. Mullin, "Estimating the size of a relational join," Information Systems, vol.18, no.3, pp.189–196, 1993.
[3] U. Manber and S. Wu, "An algorithm for approximate membership checking with application to password security," Information Processing Letters, vol.50, no.44, pp.191–197, 1994.
[4] D. Sarang, K. Praveen, and E.T. David, "Longest prefix matching using bloom filters," Proc. ACM SIGCOMM, pp.201–212, 2003.
[5] L.L. Gremilion, "Designing a bloom filter for differential file access," Commun. ACM, vol.25, no.9, pp.600–604, 1982.
[6] F. Bonomi, M. Mitzenmacher, R. Panigraphy, S. Singh, and G. Varghese, "Beyond bloom filters: From approximate membership checks to approximate state machines," Proc. ACM SIGCOMM, pp.315–326, 2006.
[7] J. Ledlie, J.M. Taylor, L. Serban, and M. Seltzer, "Self-organization in peer-to-peer systems," Proc. 10th European SIGOPS Workshop, pp.125–132, 2002.
[8] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen, "PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities," Proc. 12th IEEE International Symposium on High Performance Distributed Computing, pp.236–246, 2003.
[9] S.C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," Proc. INFOCOM 2002, pp.1248–1257, 2002.
[10] A. Whitaker and D. Wetherall, "Forwarding without loops in Icarus," Proc. Open Architectures and Network Programming, pp.63–75, 2002.
[11] C. Estan and G. Varghese, "New directions in trace measurement and accounting," Proc. ACM SIGCOMM, pp.323–336, 2002.
[12] D.C. Suresh, Z. Guo, B. Buyukkurt, and W.A. Najjar, "Automatic compilation framework for bloom filter based intrusion detection," Proc. ARC, pp.413–418 2006.
[13] P. Gross, J. Parekh, and G. Kaiser, "Secure "selecticast" for collaborative intrusion detection systems," Proc. International Workshop on Distributed Event-based Systems, pp.50–55, 2004.
[14] M.E. Locasto, J.J. Parekh, A.D. Keromytis, and S.J. Stolfo, "Towards collaborative security and P2P intrusion detection," Proc. Information Assurance Workshop 2005, pp.30–36, 2005.
[15] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," IEEE J. Sel. Areas Commun., vol.23, no.4, pp.839–850, 2005.
[16] W.B. Jaballah, A. Meddeb, and H. Youssef, "An efficient source authentication scheme in wireless sensor network," Proc. IEEE/ACS International Conference on Computer Systems and Applications, pp.1–7, 2010.
[17] Y.-S. Chen and C.-L. Lei, "Filtering false messages en-route in wireless multi-hop networks," Proc. IEEE Wireless Communications and Networking Conference, pp.1–6, 2010.
[18] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," Internet Mathematics, vol.1, no.4, pp.485–509, 2005.

Fei Xu received her Ph.D. degree in Network and Information Security at Beijing Institute of Technology in 2012. She has been working as an Assistant Professor at Institute of Information Engineering, Chinese Academy of Sciences since then. She is also with Law School, Renmin University of China.

Pinxin Liu received his Ph.D. degree in Law at Renmin University of China in 2003. He is currently a Professor at the School of Law, and a Researcher at Research Center of Criminal Jurisprudence, Renmin University of China.

Jing Xu received her Ph.D. degree in Network and Information Security at Beijing Institute of Technology in 2016. She is currently a postdoc in Tsinghua University.

Jianfeng Yang received his master degree in Network and Information Security from Peking University in 2015.

S.M. Yiu received his Ph.D. degree in Computer Science at the University of Hong Kong in 1997. He is currently an Associate Professor in the Department of Computer Science of the same university.