A Multi-attribute Data Structure with Parallel Bloom Filters for Network Services*

Yu Hua[1,2] and Bin Xiao[1]

[1] Department of Computing
Hong Kong Polytechnic University, Kowloon, Hong Kong
{csyhua, csbxiao}@comp.polyu.edu.hk
[2] School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, China

Abstract. A Bloom filter has been widely utilized to represent a set of items because it is a simple space-efficient randomized data structure. In this paper, we propose a new structure to support the representation of items with *multiple* attributes based on Bloom filters. The structure is composed of Parallel Bloom Filters (PBF) and a hash table to support the accurate and efficient representation and query of items. The PBF is a counter-based matrix and consists of multiple submatrixes. Each submatrix can store one attribute of an item. The hash table as an auxiliary structure captures a verification value of an item, which can reflect the inherent dependency of all attributes for the item. Because the correct query of an item with multiple attributes becomes complicated, we use a two-step verification process to ensure the presence of a particular item to reduce false positive probability.

1 Introduction

A standard Bloom filter can represent a set of items as a bit array using several independent hash functions and support the query of items [1]. Using a Bloom filter to represent a set, one can query whether an item is a member of the set according to the Bloom filter, instead of the set. This compact representation is the tradeoff for allowing a small probability of false positive in the membership query. However, the space savings often outweigh this drawback when the false positive probability is rather low. Bloom filters can be widely used in practice when space resource is at a premium.

From the standard Bloom filters, many other forms of Bloom filters are proposed for various purposes, such as counting Bloom filters [2], compressed Bloom filters [3], hierarchical Bloom filters [4], space-code Bloom filters [5] and spectral Bloom filters [6]. Counting Bloom filters replace an array of bits with counters in order to count the number of items hashed to that location. It is very useful

* This work is partially supported by HK RGC CERG B-Q827 and POLYU A-PA2F, and by the National Basic Research 973 Program of China under Grant 2004CB318201.

Y. Robert et al. (Eds.): HiPC 2006, LNCS 4297, pp. 277–288, 2006.

to apply counting Bloom filters to support the deletion operation and handle a set that is changing over time.

With the booming development of network services, the query based on *multiple* attributes of an item becomes more attractive. However, not much work has been done in this aspect. Previous work mainly focused on the representation of a set of items with a single attribute, and they could not be used to represent items with multiple attributes accurately. Because one item has multiple attributes, the inherent dependency among multiple attributes could be lost if we only store attributes in different places by computing hash functions independently. There are no functional units to record the multiple attributes dependency by the simple data structure expansion on the standard Bloom filters and the query operations could often receive wrong answers. The lost of dependency information among multiple attributes of an item greatly increases the false probability. Thus, we need to develop a new structure to the representation of items with multiple attributes.

In this paper, we make the following main contributions. First, we propose a new Bloom filter structure that can support the representation of items with multiple attributes and allow the false positive probability of the membership queries at a very low level. The new structure is composed of Parallel Bloom Filters (PBF) and a hash table to support the accurate and efficient representation and query of items. The PBF is a counter-based matrix and consists of multiple submatrixes. Each submatrix can store one attribute of an item. The hash table captures a verification value of an item, which can reflect the inherent dependency of all attributes for one item. We generate the verification values by an attenuated method, which tremendously reduces the items collision probability. Second, we present a two-step verification process to justify the presence of a particular item. Because the multiple attributes of an item make the correct query become complicated, the verification in the PBF alone is insufficient to distinguish attributes from one item to another. The verification in the hash table can complement the verification process and lead to accurate query results. Third, the new data structure in the PBF explores a counter in each entry such that it can support comprehensive data operations of adding, querying and removing items and these operations remain computational complexity $O(1)$ using the novel structure. We also study the false positive probability and algebra operations through mathematic analysis and experiments. Finally, we show that the new Bloom filter structure and proposed algorithms of data operations are efficient and accurate to realize the representation of an item with multiple attributes while they yield sufficiently small false positive probability through theoretical analysis and simulations.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 presents the new Bloom filter structure, which is composed of the PBF and hash table. Section 4 illustrates the operations of adding, querying and removing items. In Section 5, we present the corresponding algebra operations. Section 6 provides the performance evaluation and Section 7 concludes our paper.

2 Related Work

A Bloom filter can be used to support membership queries [7], [8] because of its simple space-efficient data structure to represent a set and Bloom filters have been broadly applied to network-related applications. Bloom filters are used to find heavy flows for stochastic fair blue queue management scheme [9] and summarize contents to help the global collaboration [10]. Bloom filters provide a useful tool to assist the network routing, such as route lookup [11], packet classification [12], per-flow state management and the longest prefix matching [13].

There is a great deal of room to develop variants or extensions of Bloom filters for specific applications. When space is an issue, a Bloom filter can be an excellent alternative to keeping an explicit list. In [14], authors designed a data structure called an exponentially decaying bloom filter (EDBF) that encoded such probabilistic routing tables in a highly compressed manner and allowed for efficient aggregation and propagation.

In addition, network applications emphasize a strong need to engineer hash-based data structure, which can achieve faster lookup speeds with better worst-case performance in practice. From the engineering perspective, authors in [15] extended the multiple-hashing Bloom filter by using a small amount of multi-port on-chip memory, which can support better throughput for router applications based on hash tables.

Due to the essential role in network services, the structure expansion of Bloom filters is a well-researched topic. While some approaches exist in the literature, most work emphasizes the improvements on the Bloom filters themselves. Authors in [16] suggested the multi-dimension dynamic bloom filters (MDDBF) to support representation and membership queries based on the multi-attribute dimension. Their basic idea was to represent a dynamic set A with a dynamic $s \times m$ bit matrix that consists of s standard Bloom filters. However, the MDDBF lacks a verification process of the inherent dependency of multiple attributes of an item, which may increase the false positive probability.

3 Analytical Model

In this section, we will introduce a novel structure, which is composed of PBF and a hash table, to represent items of p attributes. The hash table stores the verification values of items and we provide an improved method for generating the verification values.

3.1 Proposed Structure

Figure 1 shows the proposed structure based on the counting Bloom filters. The whole structure includes two parts: PBF and a hash table. PBF and the hash table are used to store multiple attributes and the verification values of items, respectively. PBF uses the counting Bloom filters [2] to support the deletion operation and can be viewed as a *matrix*, which consists of p parallel *submatrixes*

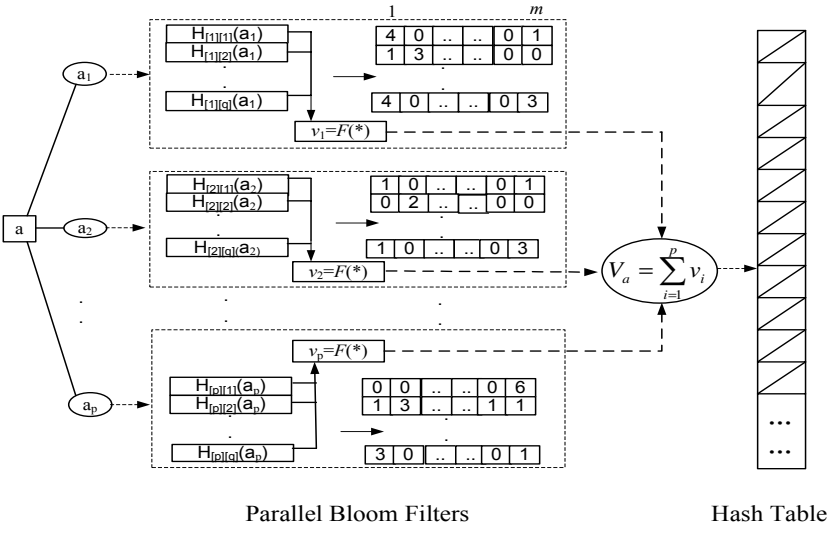


Fig. 1. The proposed structure based on counting Bloom filters

in order to represent p attributes. A submatrix is composed of q parallel *arrays* and can be used to represent one attribute. An array consists of m counters and is related to one hash function. q arrays in parallel are corresponding to q hash functions. Assume that a_i is the ith attribute of item a. We use $H_{[i][j]}(a_i)(1 \leq i \leq p, 1 \leq j \leq q)$ to represent the hash value computed by the jth hash function for the ith attribute of item a. Thus, each submatrix has $q \times m$ counters and PBF composed of p submatrixes utilizes $p \times q \times m$ counters to store the items with p attributes.

The hash table contains the verification values, which can be used to verify the inherent dependency among different attributes from one item. We measure the verification values as a function of the hash values. Let $v_i = F(H_{[i][j]}(a_i))$ be the verification value of the ith attribute of item a. The verification value of item a can be computed by $V_a = \sum_{i=1}^{p} v_i$, which can be inserted into the hash table for future dependency tests.

3.2 Role of Hash Table

The fundamental role of the hash table is to verify the inherent dependency of all attributes for an item and avoid the query collision. The main reason for the query collision in terms of multiple attributes is that the dependency among multiple attributes is lost after we insert p attributes into p independent submatrixes, respectively. Then, the PBF *only* knows the existence of attributes and cannot determine whether those attributes belong to one item. Meanwhile, the verification based on PBF itself is not enough to distinguish attributes from ne item to another. Therefore, the hash table can be used to confirm whether the queried multiple attributes belong to one item.

Thus, if a query receives answer *True*, the two-step verification process must be conducted. First, we need to check whether queried attributes exist in PBF. Second, we need to verify whether the multiple attributes belong to a single item based on the verification value in the hash table.

3.3 Verification Value

Traditionally, the hash values computed by hash functions are only used to update the location counters in the counting Bloom filters. In the proposed structure, we utilize the hash values to generate the verification values, which can stand for existing items.

The basic method of generating the verification value is to add all the hash values and store their sum in the hash table. For example, the value of variable v_i is $v_i = F(H_{[i][j]}(a_i)) = \sum_{j=1}^{q} H_{[i][j]}(a_i)$ for the ith attribute of item a. In this case, the function F is a sum operation. Then, the verification value of item a is $V_a = \sum_{i=1}^{p} \sum_{j=1}^{q} H_{[i][j]}(a_i)$. Thus, V_a can be inserted into the hash table and stands for an existing item a. However, in the basic method, the values computed by different hash functions are possible to be the same and their sums might be the same, too. Thus, different items might hold the same verification values in the hash table and this will lead to the verification collision.

The improved method utilizes the *sequential information* of hash functions to distinguish the verification values of different items. We allocate different weights to sequential hash functions in order to reflect the difference among hash functions. As for the ith attribute of item a, the value from the jth hash function in the ith submatrix is defined as $\frac{H_{[i][j]}(a_i)}{2^j}$, which is similar to the idea of the *Attenuate Bloom Filters* [17]. In attenuate Bloom filters, higher filter levels are attenuated with respect to earlier filter level and it is a lossy distributed index. Therefore, as for the item a, the verification value of the ith attribute is defined as $v_i = F(H_{[i][j]}(a_i)) = \sum_{j=1}^{q} \frac{H_{[i][j]}(a_i)}{2^j}$. The verification value of item a is $V_a = \sum_{i=1}^{p} \sum_{j=1}^{q} \frac{H_{[i][j]}(a_i)}{2^j}$. This verification value of item a can be inserted into the hash table.

4 Operations on Data Structure

Given a certain item a, it has p attributes and each attribute can be represented using q hash functions as shown in Figure 1. We denote its verification value by V_a, which is initialized to zero. Meanwhile, we can implement the corresponding operations, such as adding, querying and removing items, with a complexity of $O(1)$ in the parallel Bloom filters and the hash table.

4.1 Adding Items

Figure 2 presents the algorithm of adding items in the proposed structure. We need to compute the hash values of multiple attributes by hash functions and

then generate the verification values based on the improved method. Meanwhile, the values of corresponding location counters in PBF are incremented and corresponding operations are denoted by $PBF[H_{[i][j]}(a_i)] + +$. Finally, we insert the verification value of item a into the hash table.

Add_Item (Input: Item a)

Initialize $V_a = 0$
for $(i = 1; i \leq p; i + +)$ **do**
  for $(j = 1; j \leq q; j + +)$ **do**
    Compute $H_{[i][j]}(a_i)$
    $V_a = V_a + \frac{H_{[i][j]}(a_i)}{2^j}$
    $PBF[H_{[i][j]}(a_i)] + +$
  end for
end for
Insert V_a into the hash table

Fig. 2. The algorithm of adding an item with multiple attributes

4.2 Querying Items

Figure 3 shows the multi-attribute query algorithm, which realizes the two-step verification process. After computing the hash values of multiple attributes, we need to check whether the attributes exist in PBF. If any $PBF[H_{[i][j]}(a_i)]$ is 0 for item a, the query returns answer *False* in order to show that the queried item a does not exist. Otherwise, the hash values are added to generate the verification value V_a. If the value V_a is also in the hash table, we can determine that item a exists.

4.3 Removing Items

The operation of removing items needs to remove both the attributes in PBF and the verification values in the hash table. Figure 4 shows the algorithm for removing an item. As for an item a, we compute the hash values of its attributes and subtract 1 from the values of corresponding location counters in order to remove multiple attributes in PBF. Afterwards, the verification value of item a, V_a, is also removed from the hash table.

5 Algebra Operations

The algebra operations of Bloom filters are helpful to implement the representation and membership query of items from different sets. The operations, such as union and intersection, are still applicative in the PBF structure. We first introduce the union and intersection operations of standard Bloom filters and then describe the corresponding operations of PBF and hash table. We illustrate

Membership_Query_Item (Input: Item a)

Initialize $V_a = 0$
for $(i = 1; i \leq p; i++)$ **do**
  for $(j = 1; j \leq q; j++)$ **do**
    Compute $H_{[i][j]}(a_i)$
    if $PBF[H_{[i][j]}(a_i)]==0$ **then**
      Return *False*
    end if
    $V_a = V_a + \frac{H_{[i][j]}(a_i)}{2^j}$
  end for
end for
if V_a is in the hash table **then**
  Return *True*
end if
Return *False*

Fig. 3. The algorithm for querying an item with multiple attributes

Remove_Item (Input: Item a)

Initialize $V_a = 0$
for $(i = 1; i \leq p; i++)$ **do**
  for $(j = 1; j \leq q; j++)$ **do**
    Compute $H_{[i][j]}(a_i)$
    $V_a = V_a + \frac{H_{[i][j]}(a_i)}{2^j}$
    $PBF[H_{[i][j]}(a_i)]--$
  end for
end for
Remove V_a from the hash table

Fig. 4. The algorithm of removing an item with multiple attributes

these operations in an example. Finally, we compare the false positive probability of the standard Bloom filter and our proposed structure with respect to union and intersection operations.

5.1 Standard Bloom Filter

A set S can be represented as a Bloom filter using a mapping relation: $S \rightarrow BF(S)$. We use two Bloom filters $BF(A)$ and $BF(B)$ to represent sets A and B with the same number of bits and hash functions.

Definition 1. *The union of two Bloom filters, $BF(A)$ and $BF(B)$, can be represented as $BF(A \cup B)$ by logical OR operation of their bit vectors.*

Theorem 1. *The false positive probability of $BF(A \cup B)$ is larger than that of $BF(A)$ or $BF(B)$.*

Proof. We use $|A|$, $|B|$ and $|A \cup B|$ to represent the numbers of the sets A, B and $A \cup B$. Thus, we have $|A \cup B| \geq max\{|A|, |B|\}$. The false positive probability of set $A \cup B$ is $(1 - (1 - \frac{1}{m})^{k|A \cup B|})^k$, which is larger than the false positive probability of sets A or B, $(1 - (1 - \frac{1}{m})^{k|A|})^k$ or $(1 - (1 - \frac{1}{m})^{k|B|})^k$.

Definition 2. *The intersection of two Bloom filters, $BF(A)$ and $BF(B)$, can be represented as $BF(A \cap B)$ by logical AND operation of their bit vectors.*

Theorem 2. *The false positive probability of $BF(A \cap B)$ is smaller than that of $BF(A) \cap BF(B)$ with probability*

$$(1 - (1 - \frac{1}{m})^{k|A-(A \cap B)|})(1 - (1 - \frac{1}{m})^{k|B-(A \cap B)|})$$

Proof. Intuitively, a bit is set in both filters if it is set by items in $A \cap B$, or in $A - (A \cap B)$ and $B - (A \cap B)$ [7]. In fact, we have

$$BF(A) \cap BF(B) = BF(A \cap B) \cup BF(A - (A \cap B)) \cap BF(B - (A \cap B))$$

Meanwhile, the items in $A \cap B$ produce the same bits for filters $BF(A \cap B)$ and $BF(A) \cap BF(B)$. Thus, we can conclude that $BF(A \cap B)$ is smaller than that of $BF(A) \cap BF(B)$ when $BF(A - (A \cap B)) \cap BF(B - (A \cap B)) = 1$.

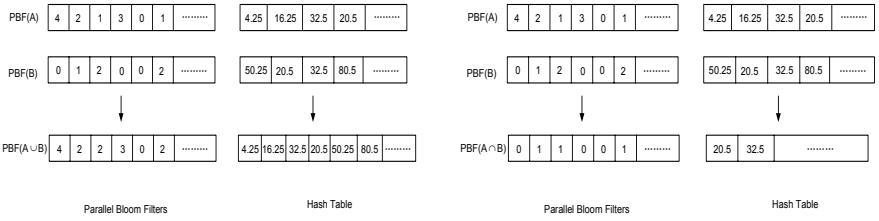
Given a standard Bloom filter and from the conclusion in [7], we know $P(BF(A - (A \cap B)) = 1) = 1 - (1 - \frac{1}{m})^{k|A-(A \cap B)|}$, and $P(BF(B - (A \cap B)) = 1) = 1 - (1 - \frac{1}{m})^{k|B-(A \cap B)|}$. Thus, the event that the false positive probability of $BF(A \cap B)$ is smaller than that of $BF(A) \cap BF(B)$ occurs with probability

$$(1 - (1 - \frac{1}{m})^{k|A-(A \cap B)|})(1 - (1 - \frac{1}{m})^{k|B-(A \cap B)|})$$

5.2 Practical Operations for PBF

Although the union and intersection operations of PBF are similar to those of standard Bloom filters, they are different because PBF is counter-based filters. The counter-based Bloom filters utilize the one-way hashed computation. Because we cannot accurately know the actual relationship between two data sets represented by two Bloom filters, the union operation result is possible not to exhibit the actual effects very accurately. Hence, we consider the conservative viewpoint as the policy of our union operation in order to statistically display the approximate result. The union operation in PBF obtains the bigger counter values from two arrays in the corresponding positions. On the contrary, the intersection operation in PBF obtains the smaller counter values.

Given the new structure to represent items with multiple attributes, the union and intersection operations will get an updated hash table to store verification values. The union operation integrates the verification values of two hash tables into a new one. The intersection operation maintains the verification values, which appear at both hash tables, in the new hash table. Figure 5(a) and (b) show an example to realize the union and intersection operations of PBF and hash tables respectively.

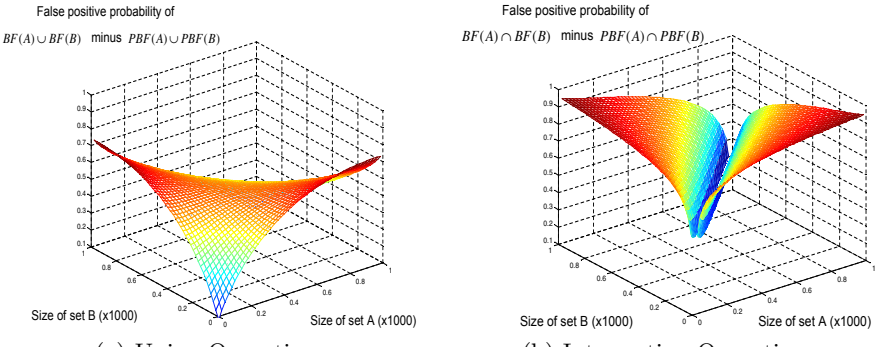


(a) Union Operation (b) Intersection Operation

Fig. 5. The union and intersection operations of PBF for multiple attributes

5.3 Comparisons of False Positive

We compare the false positive probability applying the union and intersection operations in both the standard Bloom filter and the newly proposed structure. We can compute the false positive probability of union and intersection operations for multiple attributes, which are shown in Figure 6 (Note that PBF refers to the whole structure including PBF and the auxiliary hash table). We carry out the comparison by the false positive probability of $BF(A) \cup BF(B)$ minus that of $PBF(A) \cup PBF(B)$ in Figure 6(a) and $BF(A) \cap BF(B)$ minus $PBF(A) \cap PBF(B)$ in Figure 6(b) respectively. We set the parameters as $m = 1280$ and $k = 6$.



(a) Union Operation (b) Intersection Operation

Fig. 6. The false positive probability of union and intersection operations for multiple attributes

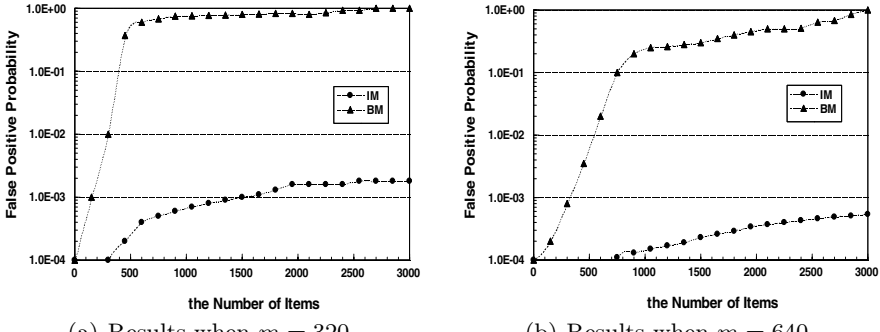
Figure 6(a) displays that the false positive probability of $PBF(A) \cup PBF(B)$ is less than that of $BF(A) \cup BF(B)$, especially when the set size becomes larger. Figure 6(b) displays that the false positive probability of $PBF(A) \cap PBF(B)$ is also much lower than that of $BF(A) \cap BF(B)$. The PBF structure fully supports algebra operations and maintains low errors. For example, we can realize the intersection operation based on PBF in order to know the common items of two data sets. Similarly, we use the union operation to get the total information of two data sets.

6 Performance Evaluation

We simulate the basic and improved methods of generating verification values and compare the false positive probability for Standard Bloom Filter (SBF) and PBF in terms of increasing number of items. In order to make the multi-attribute operations feasible in the SBF, we can concatenate multiple attributes into one parameter as the input to MD5 hash functions. Thus, the SBF in this paper uses the concatenated multiple attributes as an input of hash functions and the approach is an extension to the Bloom filters in [7] for items with multiple attributes.

6.1 Verification Values

We compare the false positive probability of *Basic Method (BM)* with that of *Improved Method (IM)* based on the same hash functions and available space sizes. Each item has four attributes and each attribute is computed by six hash functions, *i.e.*, $p = 4$ and $q = 6$, respectively. Figure 7 illustrates the simulation results. Compared with the basic method, the improved method can obtain the smaller false positive probability under different space sizes. As a result of considering the sequential information of hash functions, the improved method can distinguish the hash values of attributes of items very well. Thus, the average false positive probability of *IM* can be bounded by 0.002, which is much smaller than *BM*.

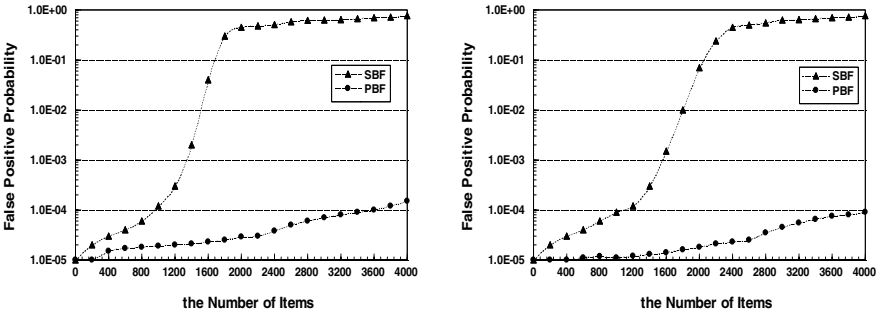


(a) Results when $m = 320$ (b) Results when $m = 640$

Fig. 7. The false positive probability of Basic and Improved Methods

6.2 Parallel Bloom Filters

In this simulation, we use the MD5 as the hash function for its well-known properties and relatively fast implementation. The value of an attribute can be hashed into 128 bits by calculating the MD5 signature. Then, we divide the 128 bits into four 32-bit values and utilize the modulus of each 32-bit value by the filter size m.



(a) Results when $m = 1280$ (b) Results when $m = 2560$

Fig. 8. The results of comparisons between SBF and PBF

Figure 8 shows the false positive probability of SBF and PBF in terms of different space sizes. Note that PBF in this figure stands for the new data architecture, which consists of PBF and the hash table. We set three attributes for each item and each attribute is computed by seven hash functions, *i.e.*, $p = 3$ and $q = 7$. Meanwhile, the space sizes available are m=1280, 2560 counters, respectively. It can be seen that given a certain number of items, the bounds on the PBF are always smaller than the bounds on the SBF. The upper probability of PBF is much smaller than that of SBF. Meanwhile, the variation trends of PBF are smooth in terms of the increasing number of items. The main reason is that the verification step based on the hash table can enhance the accuracy and efficiency of PBF, which can support the operations with multiple attributes. Therefore, although we use the simple method of attributes concatenation to realize the attributes-based operations in SBF, the PBF shows that its false positive probability is much lower than that of SBF.

7 Conclusion

Bloom filter is a kind of space-efficient data structure and can be widely used for information representation and membership query in current network environments. The space efficiency is achieved with certain false positive probability in membership query. The standard Bloom filter cannot efficiently support the representation and query of *multiple* attributes for the burgeoning and higher-level network services.

In this paper, we have presented a novel structure and practical algorithms which outperform the conventional standard Bloom filters algorithms by using the two-step verification process. Our proposed architecture extends the standard Bloom filters to efficiently support the membership query with multiple attributes. By using the verification values in the hash table, we illustrate how the false positive probability of the proposed structure can be reduced significantly. Meanwhile, the operations on both Bloom filters and the hash table have the complexity of $O(1)$. Hence, the total complexity of our proposed structure

is of the order $O(1)$. Through theoretical analysis and simulations, we further show that the novel architecture can be efficiently applied in network services for its small space requirement and very low false positive probability.

References

1. Bloom B.: Space/time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, **13** (1970) 422–426
2. Fan L., Cao P., Almeida J., Broder Z.A.: Summary cache: a scalable wide area web cache sharing protocol. IEEE/ACM Transaction on Networking, **8** (2000) 281–293
3. Mitzenmacher M.: Compressed Bloom filters. IEEE/ACM Transaction on Networking, **10** (2002) 604–612
4. Zhu Y.F., Jiang H., Wang J.: Hierarchical Bloom Filter Arrays (HBA): A Novel, Scalable Metadata Management System for Large Cluster-based Storage. Proceedings of the 5th IEEE International Conference on Cluster Computing (Cluster), (2004) 165–174
5. Kumar A., Xu J., Wang J., Spatschek O., Li L.: Space-Code Bloom filter for efficient per-flow traffic measurement. Proceedings of the IEEE INFOCOM, **3** (2004) 1762–1773
6. Saar C., Yossi M.: Spectral Bloom filters. Proceedings of the ACM SIGMOD, (2003) 241–252
7. Broder A., Mitzenmacher M.: Network applications of Bloom filters: a survey. Internet Mathematics, **1** (2005) 485–509
8. Xiao B., Chen W., He Y.X., Sha E.H.M.: An active detecting method against SYN flooding attack. Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS), **1** (2005) 709–715
9. Feng W.C., Kandlur D.D., Saha D., Shin K.G.: Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. Proceedings of the IEEE INFOCOM, **3** (2001) 1520–1529
10. Cuenca-Acuna F.M., Peery C., Martin R.P., Nguyen T.D.: PlantP:Using gossiping to build content addressable peer-to-peer information sharing communities. Proceedings of the 12th IEEE High Performance Distributed Computing, (2003) 236–246
11. Broder A., Mitzenmacher M.: Using multiple hash functions to improve IP lookups. Proceedings of the IEEE INFOCOM, **3** (2001) 1454–1463
12. Baboescu F., Varghese G.: Scalable packet classification. Proceedings of the ACM SIGCOMM, (2001) 199–210
13. Dharmapurikar S., Krishnamurthy P., Taylor D.E.: Longest Prefix Matching Using Bloom Filters. Proceedings of the ACM SIGCOMM, (2003) 201–212
14. Kumar A., Xu J., Zegura E.W.: Efficient and scalable query routing for unstructured peer-to-peer networks. Proceedings of the IEEE INFOCOM, **2** (2005) 1162–1173
15. Song H.Y., Dharmapurikar S., Turner J., Lockwood J.: Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. Proceedings of the ACM SIGCOMM, (2005) 181–192
16. Guo D.K., Wu J., Chen H.H., Luo X.J.: Theory and Network Application of Dynamic Bloom Filters. Proceedings of the IEEE INFOCOM, (2006)
17. Rhea S.C., Kubiatowicz J.: Probabilistic location and routing. Proceedings of the IEEE INFOCOM, **3** (2002) 1248–1257