Theory and Network Applications of Dynamic Bloom Filters

Deke Guo*, Jie Wu†, Honghui Chen*, and Xueshan Luo*
* Key laboratory of C4ISR Technology, School of Information System and Management
National University of Defense Technology, Changsha, Hu Nan, 410073, P. R. China
Email: aeronautic@126.com
† Department of Computer Science and Engineering
Florida Atlantic University, Boca Raton, FL 33431, U.S.A.
Email: jie@cse.fau.edu

***Abstract*— A bloom filter is a simple, space-efficient, randomized data structure for concisely representing a static data set, in order to support approximate membership queries. It has great potential for distributed applications where systems need to share information about what resources they have. The space efficiency is achieved at the cost of a small probability of false positive in membership queries. However, for many applications the space savings and short locating time consistently outweigh this drawback. In this paper, we introduce dynamic bloom filters (DBF) to support concise representation and approximate membership queries of dynamic sets, and study the false positive probability and union algebra operations. We prove that DBF can control the false positive probability at a low level by adjusting the number of standard bloom filters used according to the actual size of current dynamic set. The space complexity is also acceptable if the actual size of dynamic set does not deviate too much from the predefined threshold. Furthermore, we present multi-dimension dynamic bloom filters (MDDBF) to support concise representation and approximate membership queries of dynamic sets in multiple attribute dimensions, and study the false positive probability and union algebra operations through mathematic analysis and experimentation. We also explore the optimization approach and three network applications of bloom filters, namely bloom joins, informed search, and global index implementation. Our simulation shows that informed search based on bloom filters can obtain higher recall and success rate of query than the blind search protocol.**

Keywords: Bloom filters, informed search, peer-to-peer networks, resource routing

I. Introduction

Information representation and query processing are two core problems of many computer applications, and are often associated with each other. Representation means organizing information according to some format and mechanism, and making information operable by the corresponding method. Query processing means making decisions about whether an element with a given attribute value belongs to a given set.

A bloom filter (BF) is a simple, space-efficient, randomized data structure for representing a static set, in order to support an approximate membership query [1]. A bloom filter for a set S of n elements uses an array of m bits for a concise representation. Then, we can check whether an element x belongs to a given set according to its corresponding bloom filter rather than directly on the set itself. The space efficiency is achieved at the cost of a small probability of false positives in membership queries. However, for many applications the space savings and short locating time consistently outweigh this drawback.

Bloom filters have been extensively used in database applications [2] and have received widespread attention in networking literature recently. Bloom filters can be used to summarize contents to aid global collaboration in peer-to-peer (P2P) networks [3], [4], [5], to support probabilistic algorithms for routing and locating resources [6], [7], [8], [9], and to share web cache information [10]. In fact, bloom filters are a better data structure and have great potential for representing objects in memory. They have been used to summarize the contents of stream data [11] and provide a probabilistic approach for explicit state model checking of finite-state transition systems [12].

The major variations of bloom filters include compressed bloom filters [13], counter bloom filters [10], distance-sensitive bloom filters [14], bloom filters with two hash functions [15], space-code bloom filters [16], and spectral bloom filters [17]. Compressed bloom filters can improve performance in terms of bandwidth saving when bloom filters are passed on as messages. Counter bloom filters deal mainly with the element deletion operation of bloom filters. Distance-sensitive bloom filters, using locality-sensitive hash functions, try to answer queries of the form, "Is x close to an element of S?". Bloom filters with two hash functions use a standard technique in hashing to simplify the implementation of bloom filters significantly. Space-code bloom filters and spectral bloom filters are approximate representation of a multiset, which allows for querying, "How many occurrences of x are there in set M?". Both bloom filters and their variations are suitable for representing static sets whose size can be estimated before design and deployment.

Although the standard bloom filters and their variations have found their applications in different fields, there are three main

This work was conducted as part of Spatial Information Grid, supported by the National High Technology Research and Development Program of China under grants No. 2002AA104220, 2002AA131010, and 2003AA135110, and supported in part by US NSF grants ANI 0073736, CCR 0329741, CNS 0422762, CNS 0434533, and EIA 0130806.

1-4244-0222-0/06/$20.00 (c)2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the Proceedings IEEE Infocom.

obstacles listed below, together with our proposed solutions:

1) As the actual size of a data set increases, its corresponding bloom filter should scale well in order to avoid too much deviation between the actual false positive probability and the predefined threshold. In order to solve this problem, we introduce dynamic bloom filters (DBF) to support concise representation and approximate membership queries of dynamic sets.
2) How to represent dynamic sets to support queries based on multiple attributes? We propose multi-dimension dynamic bloom filters (MDDBF) to support concise representation and approximate membership queries of dynamic set in multiple attribute dimensions.
3) How to implement an efficient and scalable informed search protocol in unstructured P2P networks? We propose a framework of informed search based on bloom filters, and evaluate the positive impact of bloom filter through simulation.

The basic idea of dynamic bloom filters is to represent a dynamic set with a dynamic $s \times m$ bit matrix that consists of s standard bloom filters. We prove that DBF can control the false positive probability at a low level if DBF dynamically adjusts the number of standard bloom filters used according to the actual number of elements that belong to the given set. Furthermore, the space complexity is also acceptable if the estimation of the maximum size of the dynamic set does not deviate too much from the actual one. The most related work is split bloom filters [18] which use a constant $s \times m$ bit matrix to represent a set, where s is a constant and must be pre-defined according to the estimation of the maximum value of set size. However, split bloom filters waste too much storage space and bandwidth before the actual size of the given set reaches $(m \times \ln 2)/k$. Furthermore, a split bloom filter needs to be reconstructed when the actual size of the given set exceeds the estimation value. On the contrary, DBF naturally overcomes these disadvantages.

The rest of this paper is organized as follows. Section II surveys the theory of standard bloom filters and presents the union and intersection algebra operations on them. Section III studies the concise representation and approximate membership queries of dynamic sets, and presents dynamic bloom filters theory. Section IV studies the mechanism of dynamic set concise representation and membership queries in multiple attribute dimensions. Section V discusses the compression mechanism and three network applications of DBF. Section VI simulates the informed search protocol based on bloom filters in unstructured P2P networks. Section VII concludes this work and outlines some future work.

II. Concise representation and membership queries of static set

A. Standard bloom filters

A bloom filter is a compact data structure for probabilistic representation of a set in order to support membership queries. A bloom filter for representing a set $S = \{s_1, s_2, ..., s_n\}$ of n elements is described by a vector of m bits, with all bits initially set to 0. A bloom filter uses k independent hash functions $h_1, h_2, ..., h_k$ ranging over $\{1, ..., m\}$. These hash functions map each item in the universe to a random number uniform over the range $\{1, ..., m\}$ [19]. For each element x in S, bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. To check whether an element x belongs to S, one just needs to check whether all the $h_i(x)$ bits are set to 1. If so, then x is a member of S, although this could be wrong with some probability. Otherwise, we assume that x is not a member of S. Hence, a bloom filter may yield a false positive, for which it suggests that an element x is in S even though it is not. Each false positive is due to a filter collision, in which all bits indexed previously were set to 1 by other elements [1].

The probability of a false positive for an element not in the set can be calculated in a straightforward fashion, given our assumption that hash functions are perfectly random. Let p be the probability that a random bit of the bloom filter is 0, and let n_r be the number of elements that have been added to the bloom filters, then $p = (1 - 1/m)^{n_r \times k} \approx 1 - e^{-n_r \times k/m}$, as $n_r \times k$ bits are randomly selected, with probability $1/m$ in the process of adding each element. Let n_0 be the threshold of elements that the standard bloom filter can contain subjected to constraints m, k, and the predefined threshold of false positive probability. We use $f^{BF}(m, k, n_0, n_r)$ to denote the false positive probability caused by the $(n_r + 1)$th insertion, and we have the following expression

$$f^{BF}(m, k, n_0, n_r) = (1 - p)^k \approx (1 - e^{-k \times n_r/m})^k. \quad (1)$$

For a set X, when n_r reaches n_0, the false positive probability exceeds $f^{BF}(m, k, n_0, n_0)$. The false positive probability is also called the error rate of the given bloom filter in this paper. Expression (1) allows, for example, computing the minimal memory requirements (filter size) and number of hash functions given the error rate and number of elements in the set. From the work in [1], we know that $f^{BF}(m, k, n_0, n_0)$ is minimized exactly when $k = (m/n_0) \ln 2$.

B. Algebra operations on bloom filters

A set X can be presented as a bloom filter through a mapping relation: $X \rightarrow BF(X)$. We use two bloom filters $BF(A)$ and $BF(B)$ as the concise representation of two different given sets A and B respectively.

Definition 1: **(Union of bloom filters)** Assume that $BF(A)$ and $BF(B)$ use the same m and hash functions. Then, the union of $BF(A)$ and $BF(B)$, denoted as $BF(C)$, can be represented by a logical *or* operation between their bit vectors.

Theorem 1: If $BF(A \cup B)$, $BF(A)$ and $BF(B)$ use the same m and hash functions, then $BF(A \cup B) = BF(A) \cup BF(B)$.

Proof: Assume that the number of hash functions is k. We choose an element y from set $A \cup B$ randomly, and y must also belong to set A or B. Bits $hash_i(y)$ of $BF(A \cup B)$ are set to 1 for $1 \leq i \leq k$, and at the same time, bits $hash_i(y)$ of $BF(A)$ or $BF(B)$ are set to 1, thus $BF(A)[hash_i(y)] \cup BF(B)[hash_i(y)]$ are also set to 1. On the other hand, we

chose an element x from set A or B randomly, and x also belong to set $A \cup B$. Bits $hash_i(x)$ of $BF(A) \cup BF(B)$ are set to 1 for $1 \leq i \leq k$, and at the same time bits $hash_i(x)$ of $BF(A \cup B)$ are also set to 1. Thus, $BF(A \cup B)[i] = BF(A)[i] \cup BF(B)[i]$ for $1 \leq i \leq m$. Theorem 1 is proved to be true. ■

Theorem 2: The false positive probability of $BF(A \cup B)$ is not less than that of $BF(A)$ and $BF(B)$. At the same time, the false positive probability of $BF(A) \cup BF(B)$ is also not less than that of $BF(A)$ and $BF(B)$.

Proof: Assume that the sizes of sets A, B, and $A \cup B$ are n_a, n_b, and n_{ab} respectively. According to (1), we can calculate the false positive probability for $BF(A)$, $BF(B)$, and $BF(A \cup B)$.

In fact, given the same k and m, (1) is a monotonically increasing function of n_r. It is true that $|A \cup B| \geq \max(|A|, |B|)$, thus n_{ab} is not less than n_a and n_b. We could infer that the false positive probability of $BF(A \cup B)$ is not less than that of $BF(A)$ and $BF(B)$. According to Theorem 1, we know that $BF(A \cup B) = BF(A) \cup BF(B)$, thus the false positive probability of $BF(A) \cup BF(B)$ is also not less than the value of $BF(A)$ and $BF(B)$. Theorem 2 is proved to be true. ■

Definition 2: **(Intersection of bloom filters)** Assume that $BF(A)$ and $BF(B)$ use the same m and hash functions. Then, the intersection $BF(A)$ and $BF(B)$, denoted as $BF(C)$, can be represented by a logical and operation between their bit vectors.

Theorem 3: If $BF(A \cap B)$, $BF(A)$, and $BF(B)$ use the same m and hash functions, then $BF(A \cap B) = BF(A) \cap BF(B)$ with probability $(1 - 1/m)^{k^2 \times |A - A \cap B| \times |B - A \cap B|}$.

Proof: Assume the number of hash functions is k. We can derive (2) according to Definition 1, Theorem 1, and Definition 2

$$\begin{aligned} &BF(A) \cap BF(B) = \\ &\quad (BF(A - A \cap B) \cap BF(B - A \cap B)) \cup \\ &\quad BF(A \cap B). \end{aligned} \tag{2}$$

In fact, the elements of set $A \cap B$ contribute the same bits whose value is 1 to bloom filters $BF(A \cap B)$ and $BF(A) \cap BF(B)$. According to (2), it is easy to derive that $BF(A) \cap BF(B)$ equals to $BF(A \cap B)$ only if $BF(A - A \cap B) \cap BF(B - A \cap B) = 0$.

For any element $z \in (B - A \cap B)$, the probability that bits $hash_1(z), \ldots, hash_k(z)$ of $BF(A - A \cap B)$ are 0 should be $p^k = (1 - 1/m)^{k^2 \times |A - A \cap B|}$. Thus, we can infer that the probability that $BF(B - A \cap B) \cap BF(A - A \cap B) = 0$ should be $(1 - 1/m)^{k^2 \times |A - A \cap B| \times |B - A \cap B|}$. Theorem 3 is proved to be true. ■

III. Concise representation and membership queries of dynamic set

Standard bloom filters and their variations are practical approaches to representing a static set. Given the predefined threshold n_0 of the static set and the threshold of false positive probability, it is easy to calculate the most suitable number of hash functions k and the size of bloom filters m. However, for many applications, especially large scale and distributed systems, it is impractical to foresee the threshold size for local data set hosted by every node. Thus, it is possible that the actual size of the set will exceed n_0 gradually after deployment. As a result, the actual false positive probability will exceed its threshold, and the bloom filters will become unusable under such a scenario.

Standard bloom filters do not take dynamic sets into account. Split bloom filters partially enhance standard bloom filters by using a $s \times m$ bit matrix instead of an m-bit vector to represent a dynamic set. The basic idea of spit bloom filters is to allocate more memory space and enhance the capacity of filters before their implementation and deployment. In practice, split bloom filters also need to estimate the threshold of the size of actual data set, and will encounter the same problem faced by standard bloom filters. In other words, split bloom filters also cannot support dynamic sets, and may waste storage space and bandwidth before the actual size of the set reaches $(m \times \ln 2)/k$.

We will propose dynamic bloom filters (DBF) to represent dynamic sets. Dynamic bloom filters can enhance their capacity on demand, and control the false positive probability within an acceptable range as the size of a given dynamic set increases continuously after its deployment.

A. Dynamic bloom filters

The basic idea is to represent a dynamic set A with a dynamic $s \times m$ bit matrix that consists of s standard bloom filters. The initial value of s is one. In order to construct a DBF, we must be sure that m and the threshold of the false positive probability for those standard bloom filters are set according to the application need and experiment results. Furthermore, we need to properly calculate the number of hash functions k used and the maximum number of elements n_0 contained by those standard bloom filters according to (1).

Now, we create a DBF initialized by one standard bloom filter using the above parameters, and discuss two major operations supported by DBF. First, we present the algorithm for inserting an $element$ into a DBF. After an event chain of element insertions, we can represent a dynamic set as a dynamic bloom filter. Second, we propose the membership queries algorithm based on the DBF rather than the dynamic set itself.

Before inserting an $element$ into the given DBF, according to Algorithm 1, it needs to discover an active standard bloom filter from given DBF. If there is no active standard bloom filter, Algorithm 1 must create a new standard bloom filter as the active bloom filter, then adds 1 to s. After obtaining an active bloom filter, insert the $element$ into the current active bloom filter based on the method described in Section II, then add 1 to the value of n_r for the active bloom filter. In fact, only the last bloom filter of a DBF is always active, others are inactive.

Given a dynamic set A, it is convenient to obtain the corresponding DBF according to Algorithm 1. Thus, one can

Algorithm 1 Insert ($element$)

Require: $element$ is not null

$ActiveBF \leftarrow GetActiveStandardBF()$
if ActiveBF is null **then**
  $ActiveBF \leftarrow CreateStandardBF(m, k)$
  Add ActiveBF to this dynamic bloom filter.
  $s \leftarrow s + 1$
for $i = 1$ to k **do**
  $ActiveBF[hash_i(element)] \leftarrow 1$
$ActiveBF.n_r \leftarrow ActiveBF.n_r + 1$

GetActiveStandardBF()

for $j = 1$ to s **do**
  if $StandardBF_j.n_r < n_0$ **then**
    Return $StandardBF_j$
Return null

check whether an $element$ is a member of set A according to Algorithm 2 below with the $element$ as an input parameter.

Algorithm 2 Query ($element$)

Require: $element$ is not null

for $i = 1$ to s **do**
  $counter \leftarrow 0$
  for $j = 1$ to k **do**
    if $StandardBF_i[hash_j(element)] = 0$ **then**
      break
    else
      $counter \leftarrow counter + 1$
  if $counter = k$ **then**
    Return true
Return false

The major processes of Algorithm 2 are the following: 1) For $1 \leq j \leq k$, check whether there is a standard bloom filter of DBF, and all the $hash_j(element)$ bits of it are set to 1; 2) If the result is false, we can be sure that $element \notin A$; 3) Otherwise, we believe that $element \in A$ with some false positive probability.

The average time complexity of adding an $element$ to a standard and dynamic bloom filter is the same $O(k)$, where k is the number of hash functions used by them. The average time complexity of membership queries for standard and dynamic bloom filters are $O(k)$ and $O(k \times (S + 1)/2)$ respectively, where s is the number of standard bloom filters used by this dynamic bloom filter.

B. False positive probability of dynamic bloom filters

Given the standard bloom filter with parameters m, k, n_0, and error rate, one dynamic set A could be represented by a dynamic bloom filter using the standard bloom filter mentioned above. In other words, there is a mapping relation $A \rightarrow DBF(A)$. The corresponding dynamic bloom filter uses $s = \lceil n_r/n_0 \rceil$ standard bloom filters. Then, one can check whether an element x is a member of the dynamic set A according to Algorithm 2, if the result is true, we believe $x \in A$ even though it may be false with a certain probability. Hence, dynamic bloom filters also may yield a false positive, and each false positive is due to a filter collision, in which all the bits indexed by k independent hash functions of any one standard bloom filter of $DBF(A)$ are set to 1 by the insertion operation by other elements previously.

If $n_r \leq n_0$, $DBF(A)$ is just a standard bloom filter, and the false positive probability of $DBF(A)$ can be calculated according to (1). Otherwise, the false positive probability of $DBF(A)$ can be calculated in a straightforward way. For $1 \leq i \leq s - 1$, the false positive probability of those standard bloom filters coming from $DBF(A)$ is $f^{BF}(m, k, n_0, n_0)$, and the false positive probability of the last standard bloom filter coming from $DBF(A)$ is $f^{BF}(m, k, n_0, i)$, with $i = n_r - n_0 \times \lfloor n_r/n_0 \rfloor$. Then, the probability that not all the bits indexed by k independent hash functions of all standard bloom filters belonged to $DBF(A)$ set to 1 is $(1 - f^{BF}_{m,k,n_0,n_0})^{\lfloor n_r/n_0 \rfloor}(1 - f^{BF}_{m,k,n_0,i})$. Thus, the probability of all the bits indexed by k independent hash functions of at least one standard bloom filter of $DBF(A)$ being set to 1 can be denoted as

$$\begin{aligned} f^{DBF}_{m,k,n_0,n_r} &= 1 - (1 - f^{BF}_{m,k,n_0,n_0})^{\lfloor n_r/n_0 \rfloor}(1 - f^{BF}_{m,k,n_0,i}) \\ &= 1 - (1 - (1 - e^{-k \times n_0/m})^k)^{\lfloor n_r/n_0 \rfloor} \\ &\quad (1 - (1 - e^{-k \times (n_r - n_0 \times \lfloor n_r/n_0 \rfloor)/m})^k). \quad (3) \end{aligned}$$

In the following discussion, we will use the dynamic set A to represent both standard bloom filters and dynamic bloom filters, and observe the change trend of $f^{DBF}(m, k, n_0, n_r)$ and $f^{BF}(m, k, n_0, n_r)$ as n_r increases continuously.

For $1 \leq n_r \leq n_0$, false positive probability of $DBF(A)$ equals to the corresponding value of $BF(A)$, and both are less than or equal to $f^{BF}(m, k, n_0, n_0)$. In this case, a dynamic bloom filter degenerates as a standard one, and (3) also degenerates as (1). For $n_r > n_0$, the false positive probability of $DBF(A)$ increases gradually with n_r. The false positive probability of $BF(A)$ increases quickly to a high value, and then, slowly increase to almost one. For example, when n_r reaches $10 \times n_0$, $f^{BF}(m, k, n_0, 10 \times n_0)$ becomes about one hundred times of $f^{BF}(m, k, n_0, n_0)$, but $f^{DBF}(m, k, n_0, 10 \times n_0)$ is about ten times of $f^{BF}(m, k, n_0, n_0)$. We can draw a conclusion from (1), (3), and Figure 1 that dynamic bloom filters scale better than standard bloom filters after the actual size n_r of dynamic set exceeds the predefined threshold n_0.

Furthermore, we use multiple different dynamic and standard bloom filters to represent the same dynamic set A, and study the trend of $f^{BF}(m, k, n_0, n_r)/f^{DBF}(m, k, n_0, n_r)$ as n_r increases continuously. In the experiment, we choose four kinds of DBF using four different standard bloom filters with different m. For all four standard bloom filters, the number of hash functions used is 7, and the predefined threshold of false positive probability is 0.0098. The experiment results are illustrated in Figure 2, and it is obvious that all the four curves follow a similar trend. The ratio of the false positive probability of standard bloom filters (noted as BF_{error}) to

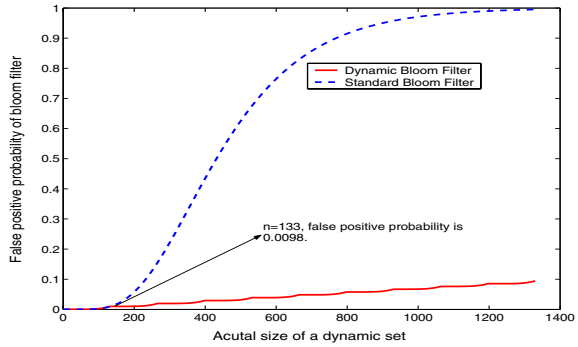

Fig. 1. False positive probability of dynamic and standard bloom filters are functions of the actual size n_r of a dynamic set, where $m = 1280$, $k = 7$, and $n_0 = 133$.

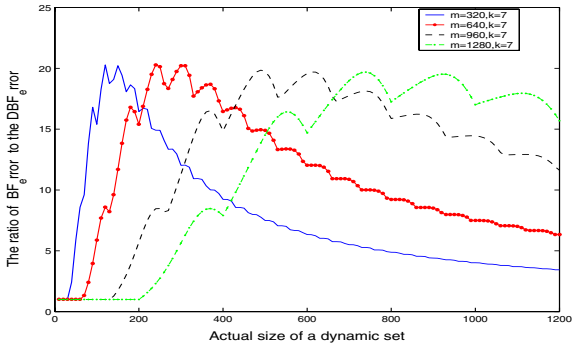

Fig. 2. The ratio of false positive probability of a standard bloom filter to the value of a DBF is a function of the actual size n_r of a dynamic set.

that of DBF (noted as DBF_{error}) is a function of the actual size n_r of dynamic set A. For $1 \leq n_r \leq n_0$, the ratio equals to 1. For $n_r > n_0$, the ratio quickly increases to the peak because of the slow increase in DBF_{error} and the quick increase in BF_{error}, and then decreases slowly because of the slow increase in DBF_{error} and the very slow increase in BF_{error}. After the actual size n_r of dynamic set A exceeds n_0, the dynamic bloom filter with different parameter m scale better than corresponding standard one. In fact, the value of m has no effect on the curve trend of $f^{BF}(m,k,n_0,n_r)/f^{DBF}(m,k,n_0,n_r)$.

$f^{BF}(m,k,n_0,n_r)$ and $f^{DBF}(m,k,n_0,n_r)$ are monotonically decreasing functions of m according to (1) and (3). In other words, $f^{DBF}(m_1,k,n_0,n_r) < f^{DBF}(m_2,k,n_0,n_r)$ for $m_1 > m_2$, this means that the curve of $f^{DBF}(m_1,k,n_0,n_r)$ is always lower than the curve of $f^{DBF}(m_2,k,n_0,n_r)$ as n_r increases. In fact, so does $f^{BF}(m,k,n_0,n_r)$. We also conduct experiments to confirm this conclusion, and illustrate the result in Figure 3. Thus, we can conclude that both standard and dynamic bloom filters which possess larger m can represent larger set and control the false positive probability at an acceptable level.

C. Algebra operations on dynamic bloom filters

We use two dynamic bloom filters $DBF(A)$ and $DBF(B)$ as the concise representation of two given different dynamic sets A and B.

Definition 3: **(Union of dynamic bloom filters)** Given the same standard bloom filter, we assume that $DBF(A)$ and $DBF(B)$ use $s_1 \times m$ and $s_2 \times m$ bit matrix, respectively. $DBF(A) \cup DBF(B)$ could result in a $(s_1 + s_2) \times m$ bit matrix. The ith line vector equals to the ith line vector of $DBF(A)$ for $1 \leq i \leq s1$, and the $(i - s_1)$ th line vector of $DBF(B)$ for $s_1 < i \leq (s_1 + s_2)$.

Theorem 4: The false positive probability of $DBF(A) \cup DBF(B)$ is larger than that of $DBF(A)$ and $DBF(B)$.

Proof: Assume that $DBF(A) \cup DBF(B)$, $DBF(A)$, and $DBF(B)$ use the same standard bloom filter with parameters m, k, n_0, and the actual size of dynamic set A and B are n_a and n_b respectively. The false positive probability of $DBF(A) \cup DBF(B)$ is

$$f^{DBF}_{m,k,n_0,n_a+n_b} = 1-(1-f^{BF}_{m,k,n_0,n_0})^{(\lfloor n_a/n_0 \rfloor + \lfloor n_b/n_0 \rfloor)} \times (1-(1-e^{-k\times(n_a-n_0\times\lfloor n_a/n_0\rfloor)/m})^k) \times (1-(1-e^{-k\times(n_b-n_0\times\lfloor n_b/n_0\rfloor)/m})^k). \quad (4)$$

The false positive probability of $DBF(A)$ and $DBF(B)$ are $f^{DBF}(m,k,n_0,n_a)$ and $f^{DBF}(m,k,n_0,n_b)$ respectively. In fact, given the same k, m, and n_0, the value of (4) minus $f^{DBF}(m,k,n_0,n_a)$ is larger than 0, and the value of (4) minus $f^{DBF}(m,k,n_0,n_b)$ is also larger than 0. Thus, we can easily derive that the false positive probability of $DBF(A) \cup DBF(B)$ is larger than that of $DBF(A)$ and $DBF(B)$. ■

Theorem 5: If the size of sets A and B is not zero and less than n_0, the false positive probability of $DBF(A) \cup DBF(B)$ is less than that value of $BF(A) \cup BF(B)$.

Proof: The false positive probability of $DBF(A) \cup DBF(B)$ is denoted as $f^{DBF}(m,k,n_0,n_a+n_b)$, and that of $BF(A) \cup BF(B)$ is denoted as $f^{BF}(m,k,n_0,n_a+n_b)$. Because the size of sets A and B is less than n_0, (4) can be simplified as (5). Let $x = e^{-k\times n_a/m}$ and $y = e^{-k\times n_b/m}$, we can obtain (6) which denotes $f^{BF}(m,k,n_0,n_a+n_b)$ minus $f^{DBF}(m,k,n_0,n_a+n_b)$ according to (1) and (5).

$$f^{DBF}_{m,k,n_0,n_a+n_b} = 1-(1-(1-e^{-k\times n_a/m})^k)(1-(1-e^{-k\times n_b/m})^k) \quad (5)$$

$$f(x,y) = (1-xy)^k + ((1-x)(1-y))^k - (1-x)^k - (1-y)^k \quad (6)$$

$$f(a) - f(d) = f(d)(a-d) + \ldots + f^{k-1}(d)(a-d)^{k-1}/(k-1)! + f^k(\xi)(a-d)^k/k!,\ d < \xi < a \quad (7)$$

$$f(c) - f(b) = f(c)(c-b) + \ldots + f^{k-1}(b)(c-b)^{k-1}/(k-1)! + f^k(\xi)(c-b)^k/k!,\ b < \xi < c \quad (8)$$

Let $a = 1 - xy$, $b = (1-x) \times (1-y)$, $c = 1 - x$, and $d = 1 - y$. Thus, $b < c < a$, $b < d < a$ because of $0 < x < 1$

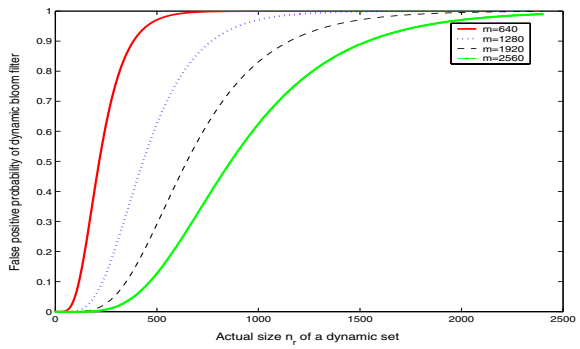

Fig. 3. False positive probability of four kinds of DBF are functions of the actual size n_r of a dynamic set, where $k = 7$, and the predefined threshold of false positive probability of each DBF is 0.0098.

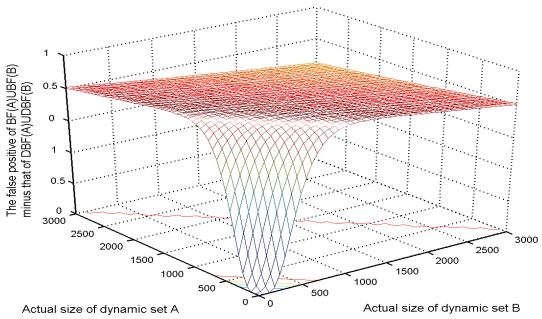

Fig. 4. False positive probability of $BF(A) \cup BF(B)$ minus that of $DBF(A) \cup DBF(B)$ is a function of size n_a of dynamic set A and n_b of set B, where $m = 1280$, $k = 7$, and $n_0 = 133$.

and $0 < y < 1$. If $c < d$, then we obtain formulas (6) and (7) according to the Tailor formula. $f(z) = z^k, 0 < z < 1$, is a monotonically increasing function of z and has a continuous k-rank derivative, and the ith derivative is a monotonically increasing function for $1 < i \leq k$. It is obvious that $a - b = c - d, d < c < a, b < d < a$. Thus, each item of $f(a)$ is larger than the corresponding item of $f(c)$, then (6) is larger than 0. If $c > d$, the result is the same. Theorem 5 is proved to be true. ■

On the other hand, we used MATLAB 6.5 to calculate the result of $f^{BF}(m, k, n_0, n_a + n_b)$ minus $f^{DBF}(m, k, n_0, n_a + n_b)$, and the result is illustrated in Figure 4. We discover that the false positive probability of $DBF(A) \cup DBF(B)$ is also less than that of $BF(A) \cup BF(B)$, even though the size of both A and B exceeds n_0.

D. Performance analysis

We compare the space of the standard and dynamic bloom filter used to represent the same dynamic set A with the same false positive probability. Then, we compare the false positive probability of dynamic and standard bloom filters, but we allow a standard bloom filter to expand its size to $s = \lceil n_r/n_0 \rceil$ instead of keeping the filter size as a constant.

First, we discuss the case where the ratio of actual size n_r to predefined size n_0 is an integer. The false positive probability of DBF can be simplified as (9) under this situation. Then, we use a standard bloom filter to present this dynamic set, and wish to obtain the same false positive probability. Let m_1 be the actual size of a standard bloom filter, and x be the ratio of actual size n_r to predefined size n_0 of dynamic set A. Finally, we establish the relationship between (1) and (9), and obtain the expression of the ratio of space used by a standard bloom filter to space used by a dynamic bloom filter as (10).

$$f^{DBF}_{m,k,n_0,n_r} = 1 - (1 - (1 - e^{-k \times n_0/m})^k)^{n_r/n_0} \quad (9)$$

$$m_1/x \times m = -k \times n_0/(m \times ln(1 - \sqrt[k]{1 - (1 - y)^x})) \quad (10)$$

We can draw the following conclusions according to (10) and Figure 5. In order to obtain the same false positive probability, standard and dynamic bloom filters use the same bits to represent dynamic set A for $n_r \leq n_0$, and standard bloom filters always use fewer bits than dynamic bloom filter for $n_r > n_0$. But, if the estimation of the maximum size of dynamic set does not deviate too much (i.e., x is not too large), then the size difference between standard and dynamic bloom filters is small. Thus, choosing DBF to represent a dynamic set will not cause much of a space complexity when compared to a standard bloom filter.

Second, if a standard bloom filter can expand its bloom filter size m to $s \times m$ ($s = \lceil n_r/n_0 \rceil$) instead of keeping its size as a constant as the dynamic set grows, false positive probability of standard bloom filters should be calculated according to (11). The false positive probability of DBF should still be (3). It is necessary to compare $f^{DBF}(m, k, n_0, n_r)$ and $f^{NBF}(m, k, n_0, n_r)$ again under this situation, and we have

$$f^{NBF}_{m,k,n_0,n_r} = (1 - e^{-k \times n_r/(m \times \lceil n_r/n_0 \rceil)})^k \quad (11)$$

We used MATLAB 6.5 to compare (3) and (11), the result is illustrated in Figure 6. We can draw the following conclusions from the result. For $n_r \leq n_0$, $f^{DBF}(m, k, n_0, n_r) = f^{NBF}(m, k, n_0, n_r)$, and both are not more than $f^{BF}(m, k, n_0, n_0)$. For $n_r > n_0$, false positive probability of DBF will increase continuously with the actual size of the given dynamic set, and that of updated standard bloom filters will fluctuate between $i \times n_0$ and $(i + 1) \times n_0$, where i is any non-negative integer. Let $n_x < n_0$ be any non-negative integer, and k be a positive integer, thus $f^{NBF}(m, k, n_0, n_x + (k - 1) \times n_0)$ is not larger than $f^{NBF}(m, k, n_0, n_x + k \times n_0)$. In fact, $f^{NBF}(m, k, n_0, n_r)$ increases as n_r increases in the whole range, but the increase rate is slower than that of $f^{DBF}(m, k, n_0, n_r)$.

The filter size m and the number of hash functions k must be consistent among all nodes, only this can guarantee the stability and inter-operability of applications in a distributed environment. If one uses standard bloom filters to represent dynamic sets among all nodes, once the size of a dynamic set exceeds its predefined threshold it needs to readjust parameters of corresponding bloom filters locally and performs a consistency operation through the whole network. The consistency operation includes propagating the new parameters of bloom filters to other peers, reconstructing the bloom filter again at

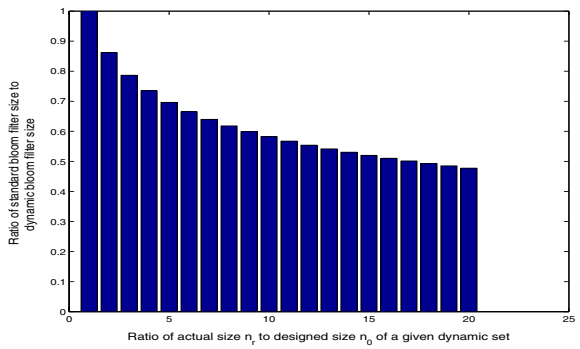

Fig. 5. The ratio of size of a standard bloom filter to that of a DBF is a function of a non-negative integer, which denotes the ratio of n_r to n_0. The experiment condition is the same as that in Figure 4.

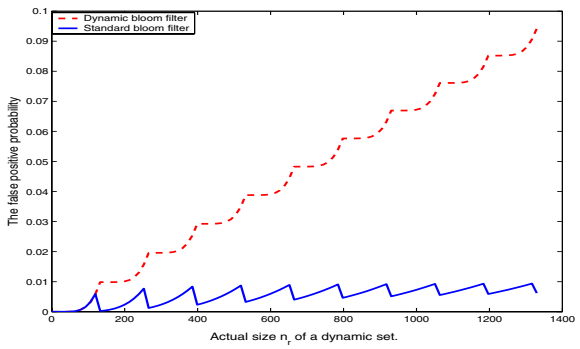

Fig. 6. False positive probability of dynamic and standard bloom filters are functions of the actual size of a dynamic set. Standard bloom filters can expand the filter size m to $\lceil n_r/n_0 \rceil \times m$. $m = 1280$, $k = 7$, and $n_0 = 133$.

each node, and gossiping each new bloom filter to other nodes. However, the overhead of consistency operation is often huge. On the other hand, it is reasonable that few nodes own large amounts of data while the most of the nodes own a small amount of data, thus it is unsuitable to use standard bloom filters with the same configuration to represent these data sets among all nodes.

Dynamic bloom filters are suitable when dealing with the above problem. Initially, one can represent a dynamic set for each node as a DBF using an appropriate number of standard bloom filters with same configuration. Once any dynamic set exceeds its threshold, the DBF just needs to adjust the number of standard bloom filters used without performing the consistency operation among all nodes. Furthermore, DBF can also control the false positive probability at a low level, and the space complexity is also acceptable if the estimation of the maximum size of the dynamic set does not deviate too much according to Figure 5.

IV. Concise Representation and Membership Queries of Multi-Attribute Dynamic Set

A. Multi-dimension dynamic bloom filters

Standard and dynamic bloom filters just focus on representing sets consisted of single attribute objects, and supporting approximate membership queries based on a single attribute. In reality, it is common to describe and represent a given object using multiple attributes in many applications. In order to deal with this situation, we propose multi-dimension standard bloom filters (MDBF) and multi-dimension dynamic bloom filters (MDDBF). The basic idea is to represent sets consisted of multi-attribute objects from each attribute dimension using standard and dynamic bloom filters. In the following discussion, we first explain the details of adding objects with multi-attribute to a MDDBF in Algorithm 3. Then add all the objects of a dynamic set A to the MDDBF according to Algorithm 3.

In order to represent multi-dimension information of a given object, we first obtain the DBF for each attribute dimension according to the attribute name from current MDDBF. Then, add the value of each attribute to the corresponding DBF by calling Algorithm 1. It is necessary to initialize every DBF for each attribute dimension before processing the first addition.

Algorithm 3 Insert($element$)

Require: $element$ with multi-attribute is not null

1: Get all attribute names of the $element$, and store them to a string array $attributes$
2: **for** $i = 0$ to $attributes.length$ **do**
3: $DynamicDBF \leftarrow GetDynamicDBF(attributes[i])$
4: **if** $DynamicDBF$ is null **then**
5: $DynamicDBF \leftarrow CreateDynamicDBF(m, k)$
6: SetDynamicBF(attribute[i], DynamicDBF)
7: DynamicDBF.Insert(element.GetValue(attribute[i]))

Once dynamic set A has been represented as an MDDBF, we check whether an $element$ is a member of set A according to the MDDBF instead of the set A itself. We present the details of the algorithm of supporting membership queries based on the value of multi-attribute in Algorithm 4.

Algorithm 4 Query($element$)

Require: $element$ with multi-attribute is not null

1: Get all attribute names of $element$, and store them to a string array $attributes$
2: **for** $i = 0$ to $attributes.length$ **do**
3: $DynamicDBF \leftarrow GetDynamicDBF(attributes[i])$
4: **if** DynamicDBF.Query(element.GetValue(attributes[i])) is false **then**
5: Return false
6: Return true

The major process of Algorithm 4 is as follows. First, find the corresponding DBF for each attribute dimension of an $element$. Second, check whether the value of $element$ for each attribute dimension is presented by corresponding DBF by invoking Algorithm 2. If the responses for all attribute dimensions are true, one can assume that $element \in A$ with some false positive probability. Otherwise, one can be sure that $element \notin A$.

The time complexity of adding an $element$ to MDDBF is $O(l \times k)$, where l denotes the number of attribute dimensions

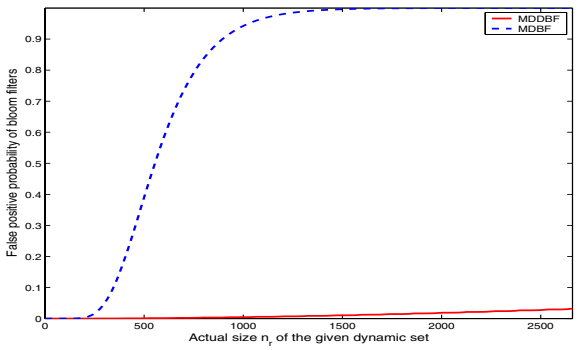

Fig. 7. The false positive probability of MDBF and MDDBF are functions of the actual size n_r of a given dynamic set, where $m = 1280$, $k = 7$, and $n_0 = 133$. The number of the attribute dimensions is 2.

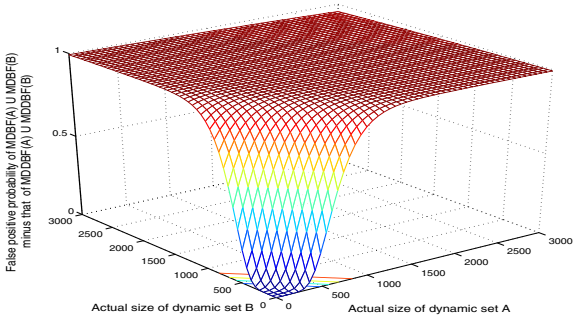

Fig. 8. False positive probability of MDBF(A)∪MDBF(B) minus that of MDDBF(A)∪MDDBF(B) is a function of the size of set A and B where $m = 1280$, $k = 7$, $n_0 = 133$, and the attribute dimension of set A and B are 3.

used to describe the full information of a given object, and k denotes the number of hash functions used by dynamic bloom filters. The average time complexity of querying an object from a given MDDBF based on multi-attribute is $O(l \times k \times (s+1)/2)$, where s is the number of standard bloom filters used by the DBF for each attribute dimension. Algorithms of multi-attribute set representation and membership queries are similar between MDBF and MDDBF, so these algorithms for MDBF are omitted here.

B. False positive probability of multi-dimension bloom filters

Given the standard bloom filter with parameters m, k, n_0, and predefined threshold of false positive probability, the dynamic set A consisting of objects with multi-attribute could be presented as a MDDBF using l DBFs through a mapping relation: $A \rightarrow MDDBF(A)$, where l is the number of attribute dimensions for set A.

If we denote the actual size of A as n_r, the corresponding DBF for each attribute dimension should use $s = \lceil n_r/n_0 \rceil$ standard bloom filters to represent it. False positive of MDDBF for an $element$ not in the set A means that there are filter collisions in each DBF for each attribute dimension, and the false positive probability of each DBF can be calculated by (3). Thus the false positive probability of a given MDDBF can be denoted as

$$f^{MDDBF}_{m,k,n_0,n_r,l} = \prod_{i=1}^{l} f^{DBF_i}_{m,k,n_0,n_r} = (f^{DBF}_{m,k,n_0,n_r})^l. \qquad (12)$$

Indeed, the dynamic set A also can be represented by a MDBF using l standard bloom filters through a mapping relation: $A \rightarrow MDBF(A)$. The false positive of MDBF for an $element$ not in set A means that there are filter collisions of BF for each attribute dimension, and the false positive probability of BF can be calculated according to (1) mentioned above. Thus, the false positive probability of a given MDBF can be denoted as

$$f^{MDBF}_{m,k,n_0,n_r,l} = \prod_{i=1}^{l} f^{BF_i}_{m,k,n_0,n_r} = (f^{BF}_{m,k,n_0,n_r})^l. \qquad (13)$$

For $1 \leq n_r \leq n_0$, MDDBF becomes MDBF, and (12) degenerates as (13). It implies that the MDBF and MDDBF representation of a set whose actual size does not exceed the largest capacity is identical. For $n_r > n_0$, the false positive probability of MDBF is larger than that of MDDBF, we also illustrate this fact in Figure 7.

The false positive probability of MDDBF increases slowly with n_r, but the corresponding value of MDBF quickly increases to a high value, and then, slowly increases to almost one. We can infer from (12), (13), and Figure 7 that MDDBF scales better than MDBF when the actual set size exceeds the predefined value.

C. Algebra operations on multi-dimension bloom filters

We assume that there are random dynamic sets A and B with identical attribute dimensions.

Definition 4: **(Union of multi-dimension dynamic bloom filters)** Assume that sets A and B are represented as MDDBF(A) and MDDBF(B) respectively. MDDBF(A)∪ MDDBF(B) result in a new MDDBF(C), and the DBF of MDDBF(C) for each attribute dimension equals to the union result on DBF of MDDBF(A) and MDDBF(B) for the same attribute dimension. The detailed union operation on DBF has been presented in Section III.

Theorem 6: The false positive probability of MDDBF(A)∪ MDDBF(B) is larger than that of MDDBF(A) and MDDBF(B).

Proof: We use MDDBF(C) to denote the result of MDDBF(A)∪ MDDBF(B). For $1 \leq i \leq l$, according to the definition of union operation on MDDBF, we can conclude that $f^{DBF_i}_{m,k,n_0,n_r}(C) = f^{DBF_i}_{m,k,n_0,n_r}(A) \cup f^{DBF_i}_{m,k,n_0,n_r}(B)$. Furthermore, it is easy to see that $f^{DBF_i}_{m,k,n_0,n_r}(A) \cup f^{DBF_i}_{m,k,n_0,n_r}(B)$ is larger than $f^{DBF_i}_{m,k,n_0,n_r}(A)$ and $f^{DBF_i}_{m,k,n_0,n_r}(B)$ according to Theorem 4. We can infer that $f^{DBF}_{m,k,n_0,n_r}(C)$ is larger than $f^{DBF}_{m,k,n_0,n_r}(A)$ and $f^{DBF}_{m,k,n_0,n_r}(B)$. Theorem 6 is proved to be true. ■

Definition 5: **(Union of multi-dimension bloom filters)** If set A and B are represented as MDBF(A) and MDBF(B), the union of MDBF(A) and MDBF(B) will result in a new MDBF(C). The standard bloom filter of MDBF(C) for each attribute dimension equals to the union result on standard bloom filters of MDBF(A) and MDBF(B) for the same attribute dimension. The union operation on standard bloom

filters has been presented in Section II.

Theorem 7: If the size of sets A and B is not zero and less than n_0, then the false positive probability of MDDBF(A)∪MDDBF(B) is less than that of MDBF(A)∪ MDBF(B).

Proof: We use MDDBF(C) and MDBF(C) to denote the result of MDDBF(A)∪MDDBF(B), MDBF(A)∪MDBF(B). Then the false positive probability of MDDBF(C) and MDBF(C) can be calculated according to (12) and (13) respectively. According to the definition of union operation on MDDBF and MDBF, for $1 \leq i \leq l$, we have

$$\begin{aligned} f^{DBF_i}_{m,k,n_0,n_r}(C) &= f^{DBF_i}_{m,k,n_0,n_r}(A) \cup f^{DBF_i}_{m,k,n_0,n_r}(B) \\ f^{BF_i}_{m,k,n_0,n_r}(C) &= f^{BF_i}_{m,k,n_0,n_r}(A) \cup f^{BF_i}_{m,k,n_0,n_r}(B) \end{aligned}$$

If the size of both sets A and B is not zero and less than n_0, $f^{DBF_i}_{m,k,n_0,n_r}(C) < f^{BF_i}_{m,k,n_0,n_r}(C)$ according to Theorem 5 for $1 \leq i \leq l$. Therefore, $f^{MDDBF}_{m,k,n_0,n_r}(C) < f^{MDBF}_{m,k,n_0,n_r}(C)$. Theorem 7 is proved to be true. ■

In fact, $f^{DBF_i}_{m,k,n_0,n_r}(C) < f^{BF_i}_{m,k,n_0,n_r}(C)$ for $1 \leq i \leq l$ according to Theorem 5 and Figure 4, even when both A and B are larger than n_0. Thus, we can infer that the false positive probability of MDDBF(A)∪MDDBF(B) is less than that of MDBF(A)∪MDBF(B) for random sets A and B. We also conducted an experiment using MATLAB 6.5 to validate Theorem 7, and illustrate the results in Figure 8.

V. Optimization and Applications of Dynamic Bloom Filters

A. Compressed multi-dimension dynamic bloom filters

Some applications that use bloom filters need to communicate these filters across the network. In this case, besides the three performance metrics we have seen so far: (1) the computational overhead to lookup a value (related to the number of the hash functions used), (2) the size of the filter in memory, and (3) the error rate, a fourth metric can be used: the size of the message used to transmit the filter across the network. The compressed bloom filters might save significant bandwidth at the cost of larger uncompressed filters and some additional computation to compress and decompress the filter sent across the network. In the idealized setting, using compression always reduces the false positive probability by adopting larger bloom filter size and less number of hash functions than standard bloom filters. We do not detail here all theoretical and practical issues about compressed bloom filters analyzed in [13].

DBF uses standard bloom filters as foundation. It is reasonable to compress DBF by using compressed bloom filters instead of standard bloom filters. Thus, compressed DBF can reduce transmission size and false positive probability of original DBF at the cost of higher memory requirements and additional computation at each node. On the other hand, it is reasonable to compress MDDBF by using compressed DBF instead of original DBF. Compressed MDDBF can reduce transmission size and false positive probability at the cost of higher memory requirements and additional computation at each node.

B. Applications of dynamic bloom filters

Bloom filters have a great potential for distributed protocols where systems need to share information about what data they have. A survey of network applications of bloom filters has been presented in [19]. Moreover, bloom filters as a better data structure has great potential for representing objects in memory [11], [12]. DBF is also suitable for various applications mentioned in those papers, and has some better characteristics than standard bloom filters.

In distributed applications, some peers own large amount of data while most of the nodes own a small amount of data. If we set relevant parameters of standard bloom filters according to the largest amount of data, it would result in huge waste of space and bandwidth. By adjusting the number of standard bloom filters used according to the actual number of data at each node, DBF can overcome this problem. Furthermore, DBF can tolerate the data increase without reconstructing a new bloom filter at each node. If a distributed application desires to distribute DBF of each peer among part of or all other peers, it also needs to keep the consistency among replications for each DBF. In reality, the data insertion just affects the active BF of DBF, and for keeping consistency it is enough to gossip the active BF instead of the whole DBF.

1) Bloom joins: Bloom joins [20], [21] is a method for performing a fast join between two distributed data sets R_1 and R_2 based on a attribute a: R_1 in site 1 and R_2 in site 2. The bloom joins includes the following steps. First, site 1 represents R_1 as a $BF(R_a)$ in the attribute dimension a and sends it to site 2. Second, site 2 sends tuples of R_2 with a match in $BF(R_a)$ to site 1, noted as R_{12}. Third, site 1 performs a join operation between R_1 and R_{12}, and produces the final result. The first transmission only sends a summarization of a projection of the tuples, and the second transmission usually contains a small fraction of the tuples. So this method is economical in network usage.

DBF is also suitable to perform single attribute distributed bloom joins between data sets as the number of tuples increases. Furthermore, MDDBF can be used to perform multi-attribute distributed bloom joins between data sets as the number of tuples increases. In the following, we will give an example.

SELECT R.a, R.b, R.c, S.d, S.e FROM R, S
WHERE R.a = S.a and R.b=S.b

First, Site 1 represents data sets R as a $BF(R_{a,b})$ in the attribute dimensions a and b, and sends it to site 2. Second, site 2 sends tuples of data set S with a match in $BF(R_{a,b})$ to site 1, denoted as $R_{r,s}$. Third, at site 1, performs a join operation between R and $R_{r,s}$, and produces the final result.

2) Informed routing: The searching strategy in unstructured P2P systems is either blind search or informed search [22]. In a blind search such as iterative deepening [23] and random walker [24], no node has information about the location of the desired data. In an informed search [25], [26], each node keeps some information about the data location.

Bloom filters are an alternative method to implement informed resource routing for distributed applications, and

many literatures have recently presented different approaches to utilize bloom filters for different scenarios [6], [7], [8], [9], [27]. The common assumption in those literatures is to represent local resource using bloom filters and gossip it to other peers according to different control mechanisms. Thus, each peer can possess individual bloom filters coming from related peers, then re-construct them according to the distance and/or direction between the local peer and other peers, and obtain a set of union results of individual bloom filters at each relative distance and/or relative direction.

A dynamic bloom filter is still suitable to support informed routing, and has more advantages than the standard one as the resource at each peer increases. Furthermore, a dynamic bloom filter is more suitable to support the necessary union operation than the standard one according to Theorem 5. As mentioned above, dynamic bloom filters, standard bloom filters, and their variations just represent objects and support approximate membership queries in a single attribute dimension. On the other hand, it often requires to route multi-attribute queries in reality. Both MDDBF and MDBF can satisfy this need. The former has the advantage over the latter as the resource at each peer increases, and supports the union operation better than the latter according to Theorem 7. Thus, DBF and its variations are better alternatives than standard bloom filters to implement informed routing in some scenarios.

3) Implementation of global index: We will refer to the globally replicated index as the global index, while the more detailed index that describes only the resources hosted locally by a peer will be denoted as the local index. Global index can be implemented in a number of ways. We define bloom filters in such a way that each peer summarizes the set of terms in its local index as a bloom filter. The cost of replicating the global index can be reduced by simply decreasing the gossiping rate; updating the global index with a new bloom filter requires constant time, regardless of the number of changes introduced. Furthermore, bloom filters can be compressed to achieve a single bit per word average ratio. Memory-constrained peers can also independently trade accuracy for storage by combining several filters into one.

When the global index has been established and propagated to the whole network, each peer uses a copy of global index hosted at local storage to find the desired peers and appropriate resources within one hop. In order to support queries that contain a set of queries based on different attribute dimensions, we can adopt MDDBF to summarize local content index and construct global content index by a periodic gossiping update operation.

VI. Simulation

In this section, we present a simulation-based evaluation of the informed search protocol and compare its performance with other blind search protocols.

We use PeerSim to design and implement our experimentations. PeerSim is delivered by the BISON project [28], and is an open source, Java based, P2P simulation framework aimed to develop and test any kind of P2P algorithm in a dynamic environment. It supports both cycle based and event based simulation. Our experiment is cycle based, which means that the simulation runs in a sequential order and in each cycle each protocol can run its behavior independently. It is easy for PeerSim to simulate more than one protocol in the same running context, and to compare many performance metrices between different protocols.

A. Informed search protocol based on bloom filters

Basically, the informed search protocol is a forward-based routing protocol. It has two major components, the construction and maintenance of a routing table, and a query forward mechanism using the routing table. In our informed protocol, the routing table is a set of dynamic bloom filters or multi-dimension dynamic bloom filters, each corresponding to a link. When a peer needs to forward a query, bloom filters corresponding to each link will be scanned and desired links will be filtered out as the forwarding directions.

In order to construct a routing table, Kumar et al have presented a novel method in [29]. Each peer first constructs the local bloom filter and sends a routing advertisement (in the form of a dynamic or multi-dimension dynamic bloom filter) to the neighbor during a connection setup. Then, the neighbor can construct a routing entry for the link from itself to the new peer. The initial advertisement is created by taking the decay union of all advertisements received from neighbors other than the target neighbors and the union of local bloom filters. It would be better for the method in [29] to adopt the gossiping protocol [30] to exchange advertisements between the source and sink peer instead of push or pull. The experiment shows that the convergent speed of the gossiping protocol is faster than that of a single push protocol.

However, the mechanism in [29] alone is not enough to ensure that each routing entry contains whole summary information of the reachable data along the corresponding link direction. In fact, the majority of early arriving peers have little information about the later peers, although the later peers have enough information about the early peers. Thus, we should pay more attention to update the routing table. Kumar et al have presented a push protocol in which each peer constructs and pushes the update advertisement for each neighbor during a given interval. In our experiment, we found that it is not necessary to update all link directions. We also adopt the asynchronous gossiping update protocol, and each peer creates an update advertisement for a random link direction at each gossiping round, and exchanges update advertisements in that direction.

The query forward mechanism is tightly coupled with the bloom filter corresponding to each link. Before the bloom filter of a given peer has propagated through the whole P2P network without any information loss, queries with payload contained by the peer may be issued at any time from any other peer. In reality, the search protocol may be designed to decay the bloom filter of a given peer during the propagation process in order to save bandwidth and make the protocol more scalable, such as the protocol mentioned in [29].

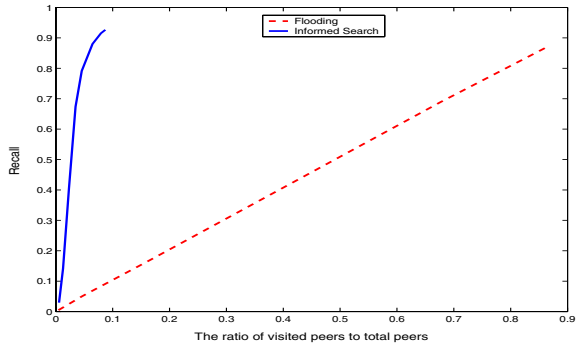

Fig. 9. The ratio of visited peers for one query to total peers vs. recall.

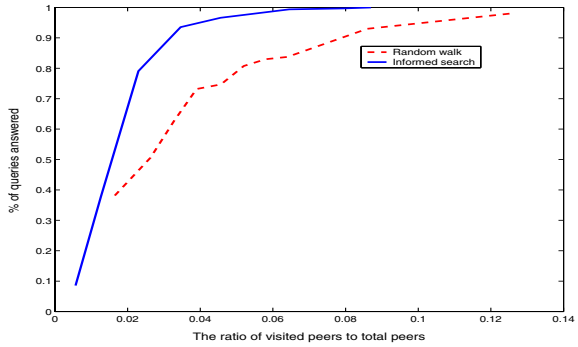

Fig. 10. The ratio of visited peers to total peers vs. % of queries.

In order to overcome information uncertainty, we combine the informed search protocol based on bloom filters with the k random walker protocol. After a peer receives a query, it will process the query and check whether to terminate the query. If the check result is true, the peer does not forward the query to any neighbor. Otherwise, the peer will forward the query to part of or all neighbors selected according to its routing table and Algorithm 2(or Algorithm 4). If there is no satisfied neighbor, the k random walker will be used as the assistant query forward protocol. This policy is also suitable when a peer initiates a query.

B. Simulation result analysis

In this section, we present simulation results using Gnutella0.4, k random walk, and informed search based on bloom filters in a random P2P network with 5,000 nodes. There are multiple replications of some objects at different locations. The model we use for replication of content is based on the zipf distribution, frequently used to model the replication of objects on the web. The ith most popular elementary object of a space will have $1/i^a$ times as many replicas as the most replicated object. In our experiment, the size of the entire object space is $50,000$, the size of elementary object space is $5,000$, and the parameter a used by the zipf law is set to 0.5. The total number of queries is $10,000$, and the distribution of query's payload also obeys the zipf law, and the parameter a is set to 0.5.

Performance issues in actual P2P networks are extremely complicated. In addition to issues such as load on the network, load on network participants, delay and success rate of query, there is a host of other metrices. In our experiment, we focus on efficiency aspects of algorithms, and use the following simple metrics.

- Pr(success): the probability of finding the queried object before the search terminates. Different algorithms have different criteria for terminating the search; it depends on the search semantics.
- Recall: the ratio of the number of relevant documents presented to the user to the total number of relevant documents in the P2P network.
- Nodes visited: the number of peers that a query's search message travel through. This is an indirect measure of the impact that a query generates on the whole network.

The simulation result of search for all copies (i.e., to get all the copies of a given object) under protocol Gnutella0.4 and our informed search based on bloom filters is illustrated in Figure 9. For any query, informed search protocol can obtain high recall without visiting a large portion of the whole P2P network in order to process the query, while the Gnutella-like protocol can obtain relatively lower recall with the cost of visiting a large portion of the whole P2P network. It is shown that the informed search based on bloom filters can avoid the message flooding problem.

The simulation result of search for one copy (i.e., to get at least one copy of a given object) under the random walker protocol and our informed search based on bloom filters is illustrated in Figure 10. For any query, the informed search protocol can obtain high Pr(success) compared to random walker with the same ratio of visited peers in the whole network. When there are multiple replications distributed randomly among the whole P2P network for any object, the Pr(success) of both protocols can almost reach 1 after visiting less than 10% of the whole network. It also explains that informed search based on bloom filters possesses advantages over blind search.

The overhead of our informed search protocol is the need to exchange information between peers at a given gossiping rate. This operation can be merged with the stabilization operation, which is used to manage the neighbor relationship and maintain the P2P network. Furthermore, the transitive size can become small by adopting bloom filters and compressed bloom filters.

VII. Conclusion

A bloom filter is a simple, space-efficient, randomized data structure for concisely representing a static data set in order to support approximate membership queries. As the actual size of the set increases continuously after deployment, a bloom filter should scale well in order to avoid too much deviation between the actual false positive probability and the predefined threshold. In order to deal with this problem, we present dynamic bloom filters to support concise representation and approximate membership queries of dynamic sets. It has been proved that dynamic bloom filters not only

possess the advantage of standard bloom filters, but also have better features than standard bloom filters when dealing with dynamic sets. False positive probability of dynamic bloom filters can be controlled at a low level, and space complexity is also acceptable if the estimation of the threshold of the dynamic set does not deviate too much. In addition, we present multi-dimension dynamic bloom filters to support concise representation and approximate membership queries of dynamic sets from multiple attribute dimensions.

We have explored three kinds of representative applications of dynamic bloom filters: bloom joins, informed search, and implementation of global index. These applications also illustrate that dynamic bloom filters and their variations scale well and are practical for representing dynamic sets. Finally, we have simulated the informed search protocol based on bloom filters in unstructured P2P networks. Our simulation shows that informed search based on bloom filters can obtain high recall and success rate of query than the blind search protocol.

In future work, we will further enhance dynamic bloom filters in order to support the removal operation, and compare the space/time trade-off of both dynamic and standard bloom filters.

ACKNOWLEDGMENT

The authors would like to thank Yongbo Wu for the help to prove a theorem, Kang Chen for the help to design related experiments, and Yang Ren and Yanli Hu for their constructive comments.

REFERENCES

[1] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[2] J. K. Mullin. Optimal semijoins for distributed database systems. *IEEE Trans. Software Eng.*, 16(5):558–560, 1990.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, and D. Geels. Oceanstore: An architecture for global-scale persistent storage. *ACM SIGPLAN Notices*.

[4] J. Li, J. Taylor, L. Serban, and M.Seltzer. Self-organization in peer-to-peer system. In *Proc. the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[5] F. M. Cuena-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. Plantp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *Proc. the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 236–249, Seattle, WA, USA, June 2003.

[6] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proc. IEEE INFOCOM*, pages 1248–1257, New York, NY, United States, June 2004.

[7] T. D. Hodes, S. E. Czerwinski, and B. Y. Zhao. An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2-3):213–230, 2002.

[8] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proc. ACM International Middleware Conference*, pages 21–40, Rio de Janeiro, Brazil, June 2003.

[9] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel. Bringing efficient advanced queries to distributed hash tables. In *Proc. IEEE Conference on Local Computer Networks*, pages 6–14, Tampa, FL, United States, November 2004.

[10] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Networking*, 8(3):281–293, 2000.

[11] C. Jin, W. Qian, and A. Zhou. Analysis and management of streaming data: A survey. *Journal of Software*, 15(8):1172–1181, 2004.

[12] C. D. Peter and M. Panagiotis. Bloom filters in probabilistic verification. In *Proc. the 5th International Conference on Formal Methods in Computer-Aided Design*, pages 367–381, Austin, Texas, USA, November 2004.

[13] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Networking*, 10(5):604–612, 2002.

[14] A. Kirsch and M. Mitzenmacher. Distance-sensitive bloom filters. http://www.eecs.harvard.edu/ michaelm/postscripts/lsbf.ps, January 2006.

[15] A. Kirsch and M. Mitzenmacher. Building a better bloom filter. http://www.eecs.harvard.edu/ michaelm/postscripts/tr-02-05.pdf, January 2006.

[16] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Proc. IEEE INFOCOM*, pages 1762–1773, Hongkong, China, March 2004.

[17] S. Cohen and Y. Matias. Spectral bloom filters. In *Proc. ACM International Conference on Management of Data (SIGMOD)*, pages 241–252, San Diego, CA, United States, June 2003.

[18] M. Xiao, Y. Dai, and X. Li. Split bloom filters. *Chinese Journal of Electronic*, 32(2):241–245, 2004.

[19] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

[20] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *Proc. the 12th International Conference on Very Large Data Bases (VLDB)*, pages 149–159, Kyoto, Jpn, August 1986.

[21] Z. Li and K. A. Ross. Perf join: An alternative to two-way semijoin and bloomjoin. In *Proc. International Conference on Information and Knowledge Management*, pages 137–144, Baltimore, MD, USA, November 1995.

[22] X. Li and J. Wu. Searching techniques in peer-to-peer networks. In J. Wu, editor, *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks*. Auerbach, New York,USA, 2006.

[23] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. the 22th IEEE International Conference on Distributed Computing*, pages 5–14, Vienna, Austria, July 2002.

[24] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. the 16th ACM International Conference on Supercomputing*, pages 84–95, Marina Del Rey, CA, United States, June 2002.

[25] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. the 22th International Conference on Distributed Computing*, pages 23–32, Vienna, Austria, July 2002.

[26] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search in peer-to-peer networks. In *Proc. the 3th International Conference on Peer-to-Peer Computing*, pages 102–109, Sweden, September 2003.

[27] K. Shanmugasundaram, H. Bronnimann, and N. Memon. Payload attribution via hierarchical bloom filters. In *Proc. the 11th ACM Conference on Computer and Communications Security*, pages 31–41, Washington, DC, United States, October 2004.

[28] G. D. Caro, F. Ducatelle, P. Heegaard, M. Jelasity, R. Montemanni, and A. Montresor. Evaluation of basic services in ahn,p2p and grid networks. http://www.cs.unibo.it/bison/deliverables/D07.pdf, February 2005.

[29] A. Kumar, J. Xu, and E. W. Zegura. Effcient and scalable query routing for unstructured peer-to-peer networks. In *Proc. IEEE INFOCOM*, pages 1162–1173, Miami, FL, United States, March 2005.

[30] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *Proc. IEEE INFOCOM*, pages 1653–1664, Miami, FL, United States, March 2005.