TMBF: Bloom Filter Algorithms of Time-Dependent Multi Bit-Strings for Incremental Set

Mingzhong Xiao[1] Xiangzhen Kong[1] Junfei Liu[1] JuNing[2]
[1]College of Information Science & Technology, Beijing Normal University, Beijing 100875, P.R. China
[2] College of Software, Chongqing University, Chongqing 400045, P.R. China
Email: {xmz,kxz,ljf}@bnu.edu.cn

***Abstract*— Set is widely used as a kind of basic data structure. However, when it is used for large scale data set the cost of storage, search and transport is overhead. The bloom filter uses a fixed size bit string to represent elements in a static set, which can reduce storage space and search cost that is a fixed constant. The time-space efficiency is achieved at the cost of a small probability of false positive in membership query. However, for many applications the space savings and locating time constantly outweigh this drawback.**

Dynamic bloom filter (DBF) can support concisely representation and approximate membership queries of dynamic set instead of static set. It has been proved that DBF not only possess the advantage of standard bloom filter, but also has better features when dealing with dynamic set. This paper proposes a time-dependent multiple bit-strings bloom filter (TMBF) which roots in the DBF and targets on dynamic incremental set. TMBF uses multiple bit-strings in time order to present a dynamic increasing set and uses backward searching to test whether an element is in a set. Based on the system logs from a real P2P file sharing system, the evaluation shows a 20% reduction in searching cost compared to DBF.

Keywords-Bloom filter; Access locality; Incremental set; Representation and query

I. INTRODUCTION

Data representation and query are two core problems of most application systems, and representation and query often associate with each other. Representation means that organize data according to some format and mechanism, and make data be operated by corresponding method. Query means that making decision about whether an element with given attribute value belongs to a given set[1,2].

Bloom filter (BF), conceived by Bloom in 1970, is used to represent and query an element in a large set [3]. It uses a set of hash functions with uniform random distribution to map elements to a fixed size bit string. The initial value of the bit all set to 0. One can judge whether element x is a member of the set according to the bloom filter instead of the set. The time-space efficiency is achieved at the cost of a small probability of false positive in membership query. However, for many applications, the space savings and locating time constantly outweigh this drawback. The bloom filter has been widely used in different areas and has a few alternatives and extensions on classic bloom filter which target on different applications. For example, counting bloom filter allows delete operation without recreating the filter by adding 4-bits counters. It enables the applications which need cache replacement operation [4].

This paper targets on another type of applications which need a dynamic increasing set. The applications only add new elements to the set in a time window $[ts, te]$. P2P file sharing system is a good example which needs to replicate and cache files[1,2,5-8]. Let S(t) presents the increasing set: If $ts \le t1 \le t2 \le te$, then $S(t1) \subseteq S(t2)$.

Figure 1 shows a possible hierarchical P2P file sharing system scenario as in ngMaze[9], DC++[10]. The nodes in $area_i$ consist a local P2P file sharing system physically or logically. The nodes in different areas exchange data by leader nodes Li.

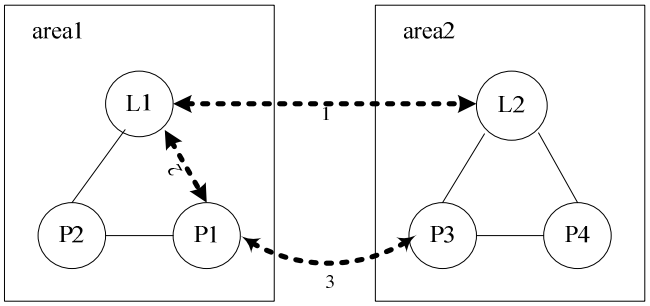

Figure 1: A Hierarchical P2P File Sharing System

Let $S_i(t)$ as the set of shared files in $area_i$ at time t. The cross area searching need to help the nodes in $area_i$ to find the file in $area_j$ when $f \notin S_i(t)$ and $f \in S_j(t), i \neq j$. Traditional implementation using leader node Lj accumulates $S_j(t)$, and broadcasts $S_j(t)$ by flooding or Gossip protocols. After Li knows $f \in S_j(t)$, the node inside $area_i$ can start to download file from $area_j$. Figure 1 shows the process of node P1 download file f from P3:

1. L1, L2 exchange sharing file meta information.
2. P1 knows $f \in P3$ after searching in L1.
3. P1 downloads file from P3.

The big cardinality, overhead transmission and increasing elements of $S_j(t)$ make represent and query $S_j(t)$ a critical problem of whole system performance.

9781-4244-3941-6/09/$25.00 ©2009 IEEE

A. Bloom filter is not suitable

Bloom filter is a compact data structure for representing a set in order to support membership queries in constant time. Suppose a data set $A = \{a_1, a_2, ..., a_n\}$, the bloom filter which represents A is described by a vector of m bits, and all bits in this vector are initialized to 0. K independent hash functions $h_1, h_2, ...h_k$ are also needed. These k hash functions map each element in the data set A to a random number over the range {1,…,m}. When adding element x into bloom filter, the bits $h_i(x)$ are set to 1 for $1 \le i \le k$. And then, we can add all the elements of set A into bloom filter with the same process. When it finished, that data set A has been represented to a bloom filter.

Bloom filter may yield a false positive, with the probability that an element x is declared a member of set A even though it is not. When we do the query on element x, if all bits indexed were set to 1 by other elements previously, then the false positive occurred, which result from a filter collision.

Suppose the hash functions we choose are perfectly random, the probability of a false positive for an element not in the set can be calculated as follows.Let p be the probability that the value of a random bit in bloom filter is 0 after representation, then $p = (1-1/m)^{nk} \approx e^{-nk/m}$. Furthermore, the false positive probability is $(1-p)^k \approx (1-e^{-kn/m})^k$.

In the application, we should denote the max false positive probability we can tolerate. Let n_0 be the max number of elements that the bloom filter can contain. That is to say, when the elements added into bloom filter exceed n_0, then the false positive probability of bloom filter will exceed the max false positive probability.

The major variations of bloom filter include compressed bloom filter[11], space-code bloom filter[12], and spectral bloom filter[13]. All these bloom filters and those variations are only suitable to represent static set whose size can't be changed. However, most of the applications request for supporting the dynamic set whose size could be increasing or decreasing with the time. This has become more and more urgent.

Until now, only multi bit-strings bloom filters can support concisely representation and approximate membership queries of dynamic set, such as SBF[1], DBF[2], IBF[6],i-DBF[7], and MBF[8]. The basic idea of these bloom filters is to represent a dynamic set A with an s*m bit matrix that consists of s standard bloom filter bit strings. Here only describes the DBF, the others can be found in related references.

B. DBF for dynamic set

In DBF, the value of s is initialized to 1, but it can increase during the continuous increasing process of the set size.

In the initialization, s has been set to 1, so the first bloom filter is the current active bloom filter. Before each element addition, check whether the number of elements presented by the current active bloom filter has come up to the designed upper threshold n_0.

If not, add the attribute value of the element to the current active bloom filter by the method mentioned in the part of bloom filter. If it is true, we first set the current active bloom filter into inactive and then create a new bloom filter to be the current active bloom filter, and the value of s should be add 1. Then we add the attribute value of the element to the current active bloom filter in the same way. Based on this addition process, only the last bloom filter of DBF is always active which means in all the inactive bloom filters, the number of elements is n_0, but in the last bloom filter, it's active, and the number of elements maybe not come up to n_0.

In order to support delete or query an element operation, DBF needs to search these bloom filter bit strings from the first, which was called forward searching technology.

C. Our work and contribution

DBF can support concisely representation and approximate membership queries of dynamic set instead of static set. It has been proved that DBF not only possess the advantage of standard bloom filter, but also has better features when dealing with dynamic set[2,6].

We propose a time-dependent multiple bit-strings Bloom filter (TMBF), which uses multiple standard bloom filter bit strings in time order to represent a dynamic increasing set and uses backward searching to test whether an element x is in a set. The representation idea is similar to the DBF, so the evaluation will mainly focus on the forward searching and backward searching technology. Backward searching search these bloom filter strings from the end instead of the first. Intuitively, it suits to the rule of access locality property in information sharing system.

Based on the system logs from a real P2P file sharing system, called Maze, the evaluation shows the backward searching can fully utilize access locality property in the system to reduce the searching cost at least 20% compared to forward searching technology.

The rest of paper is organized as follows. Section 2 describes and analysis TMBF. Section 3 presents the experimental results. Finally, we conclude in Section 4.

II. TMBF ALGORITHMS

TMBF uses a set of fixed size bloom filter bit-string to represent $S(t)$. Each bit-string binds with a time stamp. The search algorithm tests whether the element is in the bit strings one by one with backward time order.

Let $S(t) = \{e_1, e_2, ..., e_n\}$, and $T(e_1) \le T(e_2) \le ... \le T(e_n) \le t$, $T(e_i)$ is the time of adding

element e_i to the set. Figure 2 shows how to represent $S(t)$ by a sequence of bit string: $BF(0), BF(1), ..., BF(m)$, $n_0 = |S(0)| = ... = |S(m-1) - S(m-2)| \geq |S(t) - S(m-1)|$. In case of testing whether an element is a member of the $S(t)$, we use classical bloom filter to testing this element in each bit strings in a backward time order: $BF(m), BF(m-1), ..., BF(0)$.

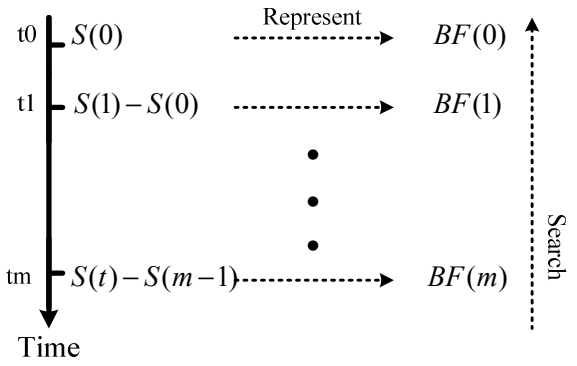

Figure 2: The Design Principle of TMBF Algorithms

We present the details of TMBF as following. *BFnumber* and *BFtime* are two global variables. The *BFnumber* records the number of bit string and the *BFtime* records the creation time of the bit string at time t. Function $T(y)$ presents the time of an element y add to increasing set $S(t)$, e.g. the creation time of a local file in P2P file sharing system. Function $Full(BF[i])$ indicates whether a bit string reaches its upper bound of capability. Function $Represent(x, BitString\ \ BF[i])$ uses k different hash function maps elements to a bit string. Function $Locate(x, BitString\ \ BF[i])$ is used to determine whether an element x belongs to BF[i] or not.

Pseudo-code of Representation Algorithm:

$At\ \ time\ \ t$:

1. $if\ (t == ts)\{i = 0; S = S(t);\}$
2. $else\{i = BFnumber; S = S(t) - S(BFtime)\}$
3. $get\ \ an\ \ element\ \ x = \min\{T(y) \mid y \in S\};$
4. $if\ \ Full(BF[i])\ \ i = i + 1;$
5. $Represent(x, BF[i]);$
6. $S = S - \{x\};$
7. $goto\ \ 3\ \ until\ \ S == NULL;$
8. $BFnumber = i; BFtime = t;$

$Represent(x, BitString\ \ BF[i])\{$

1. $for\ \ j = 1\ \ to\ \ k$
2. $BF[i][hash_j(x)] = 1;$

}

Pseudo-code of Searching an Element:

$Search\ \ x\ \ at\ \ anytime$:

1. $i = BFnumber;$
2. $while\ \ (!Locate(x, BF[i])\ \&\&(i \geq 0))\ \ i = i - 1;$
3. $if\ \ i < 0\ \ return\ \ 0; else\ \ return\ \ 1;$

$Locate(x, BitString\ \ BF[i])\{$

1. $for\ \ j = 1\ \ to\ \ k$
2. $if\ \ (BF[i][hash_j(x)]! = 1)\ \ \{return\ \ 0;\ \ break;\}$
3. $return\ \ 1;$

}

During adding an element x, if the number of element reaches the upper bound of a bit string (n_0), a new bit string will be created. All bits in the new bit string set to 0. And all the bits at those position calculated by k hash functions set to 1. Obviously, there exists some probability: $\exists y, y \in S(t), T(y) < T(x), \wedge_{j=1}^{k} BF[i][hash_j(y)] = 1$, which means that the positions in bit string of an element has been set by former adding operation. i.e., the representation algorithm is not one-to-one mapping.

According to TMBF, testing whether x belongs to S(t) is based on a set of bit string. Obviously, there also exists some probability: $x \notin S(t), but \wedge_{j=1}^{k} BF[i][hash_j(x)] = 1$, i.e., TMBF will return a error result, which is usually defined as the probability of false positive.

If we assume that hash function has uniform random distribution, and p presents the probability of a certain bit is still 0, then: $p = (1 - 1/L)^{n_0 k} \approx e^{-n_0 k / L}$. L is the length of bit string, k is the number of hash function, and n_0 is the number of maximum elements represent by a bit string. We can conclude the probability of false positive of BF and TMBF are $(1-p)^k \approx \left(1 - e^{-kn_0/L}\right)^k$, and $1 - \left(1 - \left(1 - e^{-kn_0/Ls}\right)^k\right)^s$ respectively, where s is the number of bit string, and we assume each bit string represent equal number of elements. Using appropriate parameters in a special probability of false positive, TMBF is valuable because it can reduce storage space and network communication cost.

III. EVALUATION

Compare to other works which also based on multiple bit string [1,2,6-8], TMBF conceives reverse time searching. In this section, we compare the performance results focusing on search with DBF [5] which based on forward searching by using Maze system log [9].

Maze is a peer-to-peer file-sharing application that is developed and deployed by an academic research team. Maze is in its 5th major software release, and is currently deployed across a large number of hosts inside China's education and research network. As of October 2004, Maze includes a user population of about 410K register users and supports searches on more than 150 million files totaling

over 200TB of data. At any given time, there are over 10K users online simultaneously, and over 200K transfers occurring per day. Maze provides an excellent platform to observe many important activities inside the network. More details of the Maze and its measurement results are available in http://net.pku.edu.cn.

The experimental data set chosen randomly from Maze includes peers downloading log in a sequential 7 days (a week) in 2004. We create six bloom filter bit strings with indexing 0 to 5 based on the log in first 6 days, and randomly select some requests from the log at day 7.

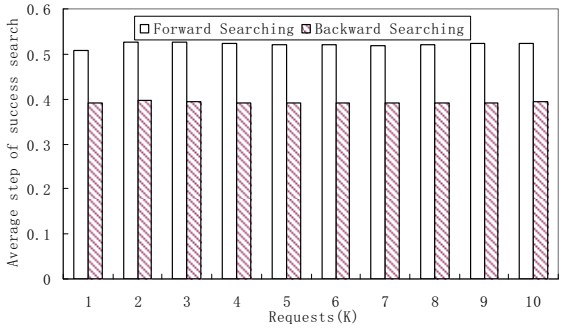

Figure 3: Comparison of Forward and Backward Searching

Figure 3 shows the comparison of forward searching and backward searching. The axis-x is the number of search requests chosen randomly at day 7. The axis-y is the average cost of finding the result. It equals to the number of bit string before we find the result. If we hit the result in the first bit string, the cost equals to 0. For example, if the number of requests is 3, the first one is found in BF[0], the second one is found in BF[1] and the third one is not found in BF[0]~BF[5], then the forward searching cost (i.e. average step length of success search) is (0+1)/2=0.5. We can see from Fig.3 that backward finding reduces the cost by at least 20% in any number of requests. For instance, if the number of requests is 1K, the cost of forward searching and backward searching are about 0.5 and 0.4 respectively.

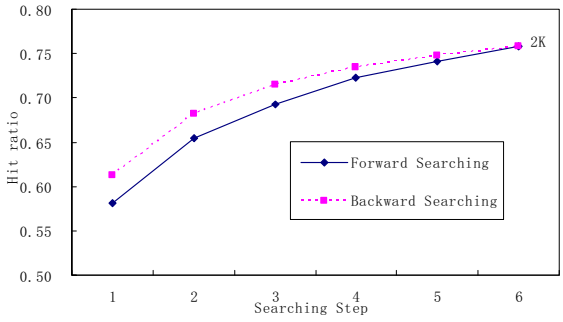

Figure 4: Verification of Access Locality

Figure 4 shows the access locality will benefit backward searching a lot. The axis-x is the searching step length, and the axis-y is the hit ratio. For example, when the step length is 2, forward searching only in BF[0] and BF[1] while backward searching only in BF[5] and BF[4], the hit ratio are 0.65 and 0.68 respectively. Figure 4 shows in any step length, backward searching is better than forward searching. And the hit ratio is the same at length 6 because both of them have to traverse all six Bloom filter bit strings. Figure 4 only shows the case of the number of requests is 2K, but the others have also the same conclusion.

IV. CONCLUSION

Bloom filter and its variations are simple space-efficient randomized data structure for concisely representing a set in order to support approximate membership query, and has a great deal of potential for distributed applications where systems need to share information about what data they have available.

Using DBF can reduce space and network communication cost for dynamic incremental data set. Based on the access locality, this paper presents a new backward searching technical to improve DBF which is called TMBF. Compared to the forward searching, our work reduces the query cost at least 20%.

ACKNOWLEDGMENT

The authors would like to thank Qi Zhang, Lijuan Liu for their suggestion and help.

REFERENCES

[1] M.Xiao, Y.Dai, and X.Li, “Split Bloom Filter”, Chinese Journal of Electronic. Vol.32, No.2, pp.241-245, 2004.

[2] Deke Guo, Honghui Chen, and Xueshan Luo, “Theory and network application of dynamic Bloom Filters”, Proc. of IEEE INFOCOM conference, 2006.

[3] B.Bloom, “Space/time tradeoffs in hash coding allowable errors”, Communications of the ACM, 1970, 13(7):422-426.

[4] L.Fan, P.Cao, J.Almeida, and A.Broder, “Summary cache: a scalable wide-area web cache sharing protocol”, IEEE/ACM transactions on networking, 2000, 8(3).

[5] Q.Lv, P.Cao, E.Chen, K.Li, and S.Shenker, “Search and replication in unstructured Peer-to-Peer networks”, Proc. of the 16th ACM International conference on supercomputing (ICS’02), NewYork, USA, June 2002.

[6] F.Hao, M.Kodialam, and T.V.Lashman, “Incremental Bloom Filters”, Proceedings of IEEE INFOCOM conference, 2008.

[7] J.Wang, M.Xiao, J.Jiang, and B.Min, “i-DBF: an improved Bloom Filter representation method on dynamic set”, gccw, pp. 156-162, Fifth International Conference on Grid and Cooperative Computing Workshops, 2006.

[8] J.Wang, M.Xiao, and Y.Dai, “MBF: a real matrix Bloom Filter representation method on dynamic set”, Proc. of IEEE NPC conference, 2007.

[9] Maze/ngMaze web site, http://maze.pku.edu.cn

[10] DC++ web site, http://sourceforge.net/dcplusplus

[11] M.Mitzenmacher, “Compressed bloom filters”, IEEE/ACM Trans. On Networking, vol.10, no.5, pp.604-612, 2002.

[12] A.Kumar, J.Xu, J.Wang, O.Spatschek, and L.Li, “Space-Code bloom filter for efficient per-flow traffic measurement”, Proc. of the conference on computer communications (IEEE INFOCOM) 2004,pp.1762-1773.

[13] C.Saar,M.Yossi, “Spectral bloom filters”, Proc. of the ACM SIGMOD international conference on management of data, 2003, pp.241-252.