第45卷 第2期 湖南大学学报(自然科学版) Vol. 45, No. 2
2018年2月 Journal of Hunan University(Natural Sciences) Feb. 2018

文章编号:1674-2974(2018)02-0133-08 DOI:10.16339/j.cnki.hdxbzkb.2018.02.17

多路平衡型矩阵 Bloom Filter*

杨磊†,黄建智
(湖南大学 信息科学与工程学院,湖南 长沙 410082)

摘 要:海量数据的高效表示和查找成为目前存储系统面临的重要挑战.针对存储系统中大规模动态数据集的表示和查找效率问题,提出一种多路平衡型矩阵 Bloom Filter 结构(M-BMBF)及其插入和查询算法. M-BMBF 根据数据集合大小建立一个 $r \times m$ 矩阵型 Bloom Filter,设计多个定位哈希函数将该矩阵 Bloom Filter 分为多组(多路)以实现平衡插入和高效查询操作.为减缓 Bloom Filter 中比特的消耗速度,使用一种“最长位匹配”填充算法,新元素的插入将从多路备选 Bloom Filter 中选择新置为1比特个数最少的 Bloom Filter 中进行.实验结果表明,相较典型拆分 Bloom Filter, M-BMBF 能在维持算法消耗时间为常量的基础上,有效节省存储空间,降低误判率.

关键词:海量数据存储;Bloom Filter;拆分 Bloom Filter;多路平衡型矩阵 Bloom Filter
中图分类号:TP301.6 **文献标志码**:A

M-Balance Matrix Bloom Filter

YANG Lei†, HUANG Jianzhi
(College of Information Science and Engineering, Hunan University, Changsha 410082, China)

Abstract: Aiming at solving the representation and query efficiency in massive and dynamic dataset on storage system, a Multi-group Balance Matrix Bloom Filter (M-BMBF) and the algorithms on insertion and searching of data element were proposed. M-BMBF initiates a $r \times m$ matrix Bloom filter according to the size of dataset, and it introduces multiple located hash functions which can be used to divide the matrix Bloom filter into multi-group to achieve balanced insertion and efficient query operations. In order to slow down the bits consumption rate in Bloom filter when a new element is inserted, a longest-bit match filling algorithm was proposed, which selects a Bloom filter as the destination position for insertion from the candidate Bloom filters according to the rule that fewest bits will be changed due to this insertion operation. Experiment results show that compared with the classical Split Bloom Filter, M-BMBF can efficiently save storage space and decrease the misjudgment rate, while its time consume is constant.

Key words: mass data storage; Bloom Filter; split Bloom Filter; M-balance matrix Bloom Filter

---

* **收稿日期**:2017-03-13
基金项目:国家重点研发计划“高性能计算”专项资助项目(2017YFB0202901), The National Key Research and Development Program of China(2017YFB0202901);湖南省自然科学基金资助项目(2015JJ2035), National Natural Science Foundation of Hunan Province (2015JJ2035);中央高校基本科研业务费资助项目, The Fundamental Research Funds for the Central Universities
作者简介:杨 磊(1976—),男,湖南临湘人,湖南大学信息科学与工程学院副教授,博士
† 通讯联系人,E-mail:937613426@qq.com

随着企业和个人数据的迅速增长,对于数据中心的存储能力及管理要求也越来越高,如今,在计算机应用中的许多基本问题都涉及到信息的表示和查询. 检测一个元素是否属于某个集合是最困难的任务之一,尤其是当要被处理的数据量很大时.

Bloom Filter[1]是一种能够表示集合并且支持集合快速查询的简单数据结构,它能够在容忍很小误判的情况下极大地节省查询集合的存储空间. Bloom Filter 已经广泛应用于各个领域,如数据库应用[2]、P2P 网络节点交互[3-5]、资源路由[6]等. 此外,Bloom Filter 在利用较少的空间表示集合方面还有很大的应用潜力.

近些年来,针对不同的应用需求,对 Bloom Filter 做出了许多实质性的改进. 计数 Bloom Filter 解决了集合中元素的删除[7]以及多维集合元素的高效表示和查询问题[8];压缩 Bloom Filter[9]减少了 Bloom Filter 在传输中所消耗的带宽;光谱 Bloom Filter[10]、拆分 Bloom Filter[11]以及动态 Bloom Filter[12]解决了集合中元素增长的问题;SKIP Bloom Filter[13]解决了数据流的动态特性问题;Ternary Bloom Filter 在计数 Bloom Filter 的基础上降低了误判率[14].

前述方法多面向网络应用,侧重于增强集合的可扩展性、控制误判率、控制带宽以及元素删除,而缺乏对检索存储系统中大容量可变数据集时高查询性能和减少内存开销等需求的考虑. 对于大数据存储系统而言,为了能够更快地检索数据,减少系统开销,通常选择将索引表置于内存甚至 cache 中,虽然 Bloom Filter 由于其所占空间非常小而可以常驻内存,从而加快集合中数据的检索速度,但是随着数据量的增加,所需 Bloom Filter 个数也增加,最终会导致内存达到瓶颈.

针对以上问题,本文提出了多路平衡型矩阵 Bloom Filter 算法(M-BMBF). 所谓平衡,即能够均匀地使用每个 Bloom Filter 的存储空间,为了达到平衡,M-BMBF 改变了 Bloom Filter 的判满条件,由原来的插入元素数量达到上限值时为满改为 Bloom Filter 的使用空间达到 50%时即为满. 当插入数据时,M-BMBF 引入多路(个)定位哈希函数将矩阵 Bloom Filter 分为多组,同时采用"最长位匹配"填充算法快速定位元素应该具体插入的 Bloom Filter,通过降低 Bloom Filter 空间的使用速度来提高 Bloom Filter 的利用率,从而可以存储更多的数据,提高数据的检索速度.

本文第一部分介绍相关工作,第二部分描述算法设计细节,第三部分给出实验评估结果,最后一部分进行总结.

1 相关工作

Bloom Filter 是由 Burton Bloom 于 1970 年基于数据库应用程序而提出的,Bloom Filter 的基本原理是通过一个长度为 m 的位向量来表示元素集合 $S=\{x_1,x_2,\cdots,x_n\}$,其中集合 S 包含有 n 个元素,向量中的每一位初始状态都为 0,对于集合中的每一个元素,都使用 k 个哈希函数把它映射到向量中,将哈希值相应的位置置 1,其他的位保持 0 不变. 当要查询一个元素是否属于某个集合时,分别使用 k 个哈希函数对该元素进行计算,检查每一个哈希值在向量中对应的位置是否为 1,只要某个哈希值在向量中的位置为 0,那么就可以判定该元素不属于集合,否则以一定的误判率认为该元素属于集合. Bloom Filter 的算法结构如图 1 所示.

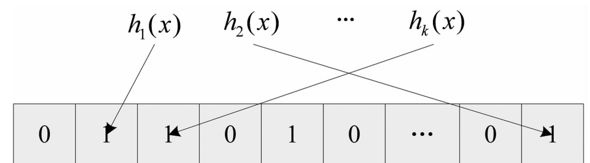

图 1 Bloom Filter 算法结构
Fig. 1 The algorithm structure of Bloom Filter

假设 m,n 和 k 给定,那么 Bloom Filter 的误判率为 $p=(1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$.

Bloom Filter 由于其简洁高效,最近在网络和存储领域上受到广泛关注[10,15-17],使用 Bloom Filter 时主要考虑误判率、内存大小、需要管理的集合数目等性能参数.

虽然 Bloom Filter 是一种高效的检索数据结构,但它还不能较好地解决诸如集合元素的动态增长和改变等问题. 随着集合中元素的增加,单个 Bloom Filter 的误判率会增大,最终将导致 Bloom Filter 失效. 近年来,针对动态集合的问题,对 Bloom Filter 作出了一些实质的改进,产生了拆分 Bloom Filter 算法[10]和动态 Bloom Filter 算法[11]以及基于这两种算法的改进算法. 拆分 Bloom Filter 和动态 Bloom Filter 的区别在于 :拆分 Bloom Filter 的基本思想是使用 r 个位向量来表示数据集合,

当集合中的元素个数增大到一定程度,影响到最初设计的误判率指标时,重新生成一个 $r \times m$ 的二维位向量,如果增加的元素并未达到预先给定的集合个数最大值,则随机选择一个位向量表示该元素. 而动态 Bloom Filter 是根据元素的增长而动态地增加 Bloom Filter,预先设计好 Bloom Filter 所能存储的元素个数,初始化 Bloom Filter 个数为 1,动态 Bloom Filter 只能在最后一个 Bloom Filter 进行元素的插入(前提是其未满,否则动态增加 Bloom Filter). 虽然拆分 Bloom Filter 和动态 Bloom Filter 解决了元素动态增长的问题,但它们并没有有效解决单个 Bloom Filter 空间利用率的问题. 同时,这些算法又带来了新的问题,在元素的查询性能方面,它们与 Bloom Filter 时间复杂度为常数相违背. 标准 Bloom Filter 的时间复杂度为 $O(k)$,而拆分 Bloom Filter 和动态 Bloom Filter 由于在插入和查询数据时需要顺序查找每个 Bloom Filter,所以其时间复杂度与 Bloom Filter 个数有关,即 $O(kr)$,其中 r 为 Bloom Filter 个数. 当 Bloom Filter 的数量很大时,查询性能较差. 矩阵 Bloom Filter[15] 基于拆分 Bloom Filter 的查询性能作了改进,通过一个特殊的哈希函数快速定位到某个 Bloom Filter,并在这个 Bloom Filter 中对元素进行查找或插入操作. Yi 等提出了 Par-BF[16],通过并行处理的方式快速匹配元素. 魏建生等人提出了动态布隆过滤器阵列(DBA)[17],DBA 由可创建的动态布隆过滤器组构成,以按需调整索引容量,通过并行计算的方式处理集合中的元素,在提高元素查询性能的同时也提高了内存空间效率. 这三种算法虽然使用不同的方法提高了元素查询效率,但是它们并未改善 Bloom Filter 的空间利用率. 为了解决 Bloom Filter 空间利用率的问题,王勇等人[18]提出了支持属性删减的布鲁姆过滤器矩阵多维元素查询算法 CBFM,具有较高的内存利用率;孙智超等人针对动态 Bloom Filter 提出了二路平衡动态 Bloom Filter (2BDBF)[19],该算法不再使用插入元素计数作为 Bloom Filter 的判满条件,而是引进了饱和度的概念,使用饱和度作为 Bloom Filter 的判满条件,并且通过向量组的方式动态增长 Bloom Filter,选择向量组中空间使用最少的 Bloom Filter 作为元素插入位置以达到插入更多元素的效果. 虽然 2BDBF 能够表示更多的元素,但其查询性能并未得到改善.

2　多路平衡型矩阵 Bloom Filter

2.1　算法设计思想

如上文所述,Bloom Filter 的误判率为 $p = (1-(1-\frac{1}{m})^{kn})^{k} \approx (1-e^{-\frac{kn}{m}})^{k}$,其中 m 为 Bloom Filter 的位长度,n 为插入元素个数,k 为哈希函数个数.

定理 1　Bloom Filter 的误判率函数为单调递增函数.

证　Bloom Filter 的哈希函数个数 k 和向量长度 m 确定的前提下,为了求出 Bloom Filter 的误判率最小,p 对 n 求导得 $\frac{dp}{dn} = k(1-e^{-\frac{kn}{m}})^{k}(-e^{-\frac{kn}{m}})(-\frac{n}{m})$,由于指数函数 e^{x} 在 $x \geqslant 0$ 时是递增函数,而 $\frac{kn}{m} > 0$,故 $e^{\frac{kn}{m}} > e^{0} = 1$,则 $0 < e^{-\frac{kn}{m}} < 1$,即可得 $1-e^{-\frac{kn}{m}} > 0$,又由于 $k > 0$,故 $\frac{dp}{dn} > 0$,由以上推导可得出 Bloom Filter 的误判率是单调递增函数,故当 $\frac{dp}{dn} = 0$ 时,误判率最小.

推论 1　维持误判率最小,插入元素达到最大数量时的 Bloom Filter 的位空间使用率为 50%.

证　由定理 1 可得,当误判率最小时,向量中能够表示的集合元素最大个数 n 满足 $n = \frac{m}{k}\ln 2$. 假设哈希函数服从均匀分布,在插入一个元素时,Bloom Filter 的某一个哈希函数映射到的某一位为 1 的概率为 $\frac{1}{m}$,那么当插入 n 个元素时,任意一位为 0 的概率为 $p' = (1-\frac{1}{m})^{kn}$,将公式 $n = \frac{m}{k}\ln 2$ 代入到 p' 中可得 $p' = (1-\frac{1}{m})^{m\ln 2}$,即可得 Bloom Filter 中任意一位为 0 的概率为 $\frac{1}{2}$,此时可以认为位为 1 的个数占向量总长度的 50%,即插入元素达到最大数量时的 Bloom Filter 的位空间使用率为 50%.

如前文所述,本文工作主要基于使用 Bloom Filter 表示动态数据集时的插入查询效率和空间使用效率展开,设计了一种多路平衡型矩阵 Bloom Filter 过滤器(M-BMBF),其算法设计结构如图 2

所示.

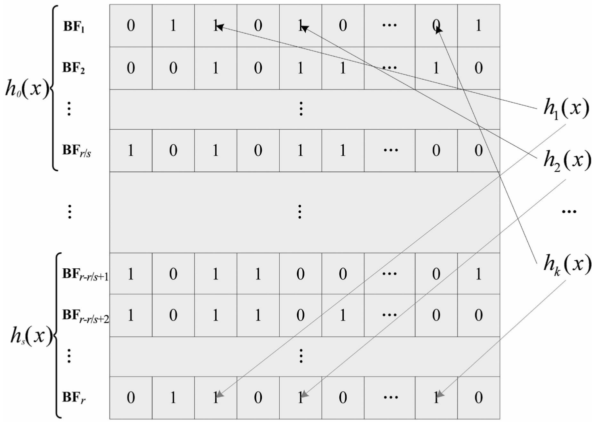

图 2 M-BMBF 算法结构

Fig. 2 The algorithm structure of M-BMBF

为解决动态数据集的插入和查询效率问题,M-BMBF 引入了多个定位哈希函数概念. M-BMBF 的基本设计思想是将动态集合表示成一个 $r\times m$ 的位矩阵. 该矩阵的行为 r,表示有 r 个 Bloom Filter;列为 m,表示每个 Bloom Filter 的向量长度为 m 比特. 在这里行为 r 是一个常量,必须按照对集合尺寸的最大值估计而预先确定. 为了提高元素的查找性能,本文设计 s 个定位哈希函数 $h_0,h_1,\cdots,h_{s-1}$,这 s 个哈希函数是经过特殊定义的并且不同于其他 k 个独立的哈希函数. 为了降低 s 个哈希函数出现冲突的概率,将 r 个 Bloom Filter 划分成 s 组,其中每组有 r/s 个 Bloom Filter,而每个定位哈希函数映射一组 Bloom Filter. 在元素添加之前,需要确定该元素应该被添加在哪一个 Bloom Filter 中,这时就可以使用这 s 个哈希函数进行定位计算,选出该元素在每一组中所对应的 Bloom Filter 作为候选插入位置.

由推论 1,我们可以将 Bloom Filter 在最低误判率下的最大插入元素个数转化为其位空间的使用状况,据此我们使用了“最长位匹配”填充算法以提高 Bloom Filter 的空间利用效率. “最长位匹配”填充算法的主要设计思想是如果在插入元素时,能把元素的哈希函数值尽量多地映射到 Bloom Filter 向量中已为 1 的位置上,则可能在保持 p 不变的前提下插入比 n 更多的元素.

具体来说,插入时,利用 k 个哈希函数计算元素 x 的 k 个哈希函数值后,将它们与候选 Bloom Filter 的对应地址位置进行比较,如果发现有其中一个 Bloom Filter 所对应的 k 个哈希函数的地址都为 1,则认为该元素已存在,不需要作插入处理;否则选择这 s 个候选 Bloom Filter 中对应地址值为 1 的位置与 x 的 k 个哈希地址重合最多的作为其插入位置. 在 M-BMBF 中,每个定位哈希函数的哈希范围为 $\{0,1,2,\cdots,r/s\}$,而其他 k 个哈希函数的范围为 $\{0,1,\cdots,m-1\}$. 因此,在构建 M-BMBF 之前,哈希函数的个数 k 并不是真正用在算法中的个数,实际上,将要使用 $k+s$ 个哈希函数. 假设定位哈希函数是完美随机的,这样每一组中所能容纳的元素个数应该是相同的,为 n/r. M-BMBF 的算法结构如图 2 所示.

2.2 算法实现

如算法 1 所示,为了表示和查找动态集合中元素,首先要根据对集合尺寸最大值的估计而预先建立好一个 $r\times m$ 的矩阵 Bloom Filter,并将所有的位初始化为 0(参见伪代码中的第 1 行). 定义 s 个定位哈希函数,然后根据上一节所描述的算法思想,将 r 个 Bloom Filter 划分成 s 组,每组有 r/s 个 Bloom Filter,每个定位哈希函数通过对第一组哈希函数的映射再偏移的方式对应映射到相应的组 Bloom Filter 中(参见伪代码中的第 5 行).

对于元素的插入过程,首先计算 s 个哈希函数,找出每一组中具体映射到的 Bloom Filter 作为备选插入位置,然后对该元素进行 k 个相互独立的哈希函数计算(参见伪代码中的第 6 行),将该元素的 k 个哈希函数地址与备选的 Bloom Filter 的对应地址位置进行比较,找出 k 个哈希函数的计算中与向量中已经为 1 的位置重合最多的 Bloom Filter,将该元素插入到此 Bloom Filter 中(参见伪代码 8－11 行),并将该 Bloom Filter 的 k 个哈希函数映射中剩余的未被置 1 的位置置为 1 即可(参见伪代码 12－13 行).

算法 1 将集合中元素插入到 M-BMBF

Input:输入整数 r、m、n、k,其中 r 为给定的矩阵行,m 为给定的矩阵列,n 为集合元素个数,k 为 Bloom Filter 的映射哈希函数个数;

Output:矩阵 M;

```
1   CreatbitArray M[0,1,…,r−1][0,1,…,m−1]←0
2   ”for(int i=1;i<=n;i++)
3   for(int d=1;d<=s;d++)
4     for(int count=0,j=1;j<=k;j++)do
5       h_d(x_i) = h_1(x_i)+(d−1)*r/s ;// h_1(x_i) 为第一组哈希函数的行映射
6       count=count+ M[h_d(x_i)][h_j(x_i)]
7       end
8       if(count>max&&M[h_d(x_i)] 未满)then
```

```
9          max←count;
10           v = h_d(x_i)
11         end if
12      for(int j=1;j<=k;j++)do
13           M[v][h_j(x_i)] = 1
14      end
```

当要查询一个元素时，其过程如算法 2 所示.

对于元素 x_i，首先利用哈希函数 $h_0,h_1,\cdots,h_{s-1}$ 依次计算这个元素在每一个 Bloom Filter 组中可能存在位置，然后在这些 Bloom Filter 中，对于 $1 \leqslant j \leqslant k$，检查是不是其中有一个 Bloom Filter 中所有的 $h_j(x_i)$ 都被置为 1(参见伪代码 5－8 行)，如果找到则返回 true，并结束查找(参见伪代码 9－12 行)，如果找不到则继续查找下一个 Bloom Filter(参见伪代码 13－14 行)，查找到最后一个 Bloom Filter 中仍然未发现，可以确定该元素不是集合 S 中的一个元素，返回 false 并结束查找(参见伪代码 16－17 行).

算法 2　在 M-BMBF 中查询元素

Input：位矩阵 $M(r\times m)$，待查元素 x，定位哈希函数个数 s，Bloom Filter 映射哈希函数 k，集合元素个数 n；

Output：true or false；

```
1    d←1;
2    j←1;
3    flag←false;
4    while(d<=s)do
5       while(j<k)do
6       if( M[h_d(x)][h_j(x)] == 1 )then
7       j++;
8       end
9      if(j==k)then
10          flag←true;
11          return flag;
12          break;
13       else
14          d++;
15       end if
16    if(d>s) then
17    return flag;
```

2.3　性能分析

2.3.1　时间复杂度(元素检索时间)

当要查询一个元素是否属于某个集合时，拆分 Bloom Filter 和 2BDBF 是对每个 Bloom Filter 按顺序依次查询，直至查到为止，而对每一个 Bloom Filter 的查询时间为 $O(k)$，其中 k 为哈希函数个数，那么拆分 Bloom Filter 和 2BDBF 最坏情况下的平均查询时间为 $\mathrm{O}(kr)$. 本文算法(如算法 2)首先用 s 个定位哈希函数定位到相对应的 Bloom Filter 组中的某一个 Bloom Filter，其时间复杂度为 $\mathrm{O}(s)$，然后在这 s 个 Bloom Filter 中进行 k 个哈希函数的映射，其时间复杂度为 $\mathrm{O}(k)$，因此，本文算法总的查询时间复杂度为 $\mathrm{O}(ks)$. 由于 s 一般远小于 r，本文算法能够有效解决查询效率的问题.

2.3.2　误判率分析

就 Bloom Filter 而言，在查询某个元素时，如果该元素本不属于动态集合，但在查询过程中却返回 true，便发生误判. 由文献[10]可知，拆分 Bloom Filter 的误判率为：

$$F = 1-(1-(1-\mathrm{e}^{-\frac{kn}{mr}})^k)^r \tag{1}$$

而对于本文所提算法，当 $0 \leqslant d \leqslant s-1$，假设第 d 个定位哈希函数对应的 Bloom Filter 发生误判的概率为 f_d，那么在算法的所有定位行中都不发生误判的概率为 $(1-f_\mathrm{d})^s$. 因此，至少有一个 Bloom Filter 发生误判的概率为 $1-(1-f_\mathrm{d})^s$，故可得 M-BMBF 的误判率为：

$$F_{\text{M-BMBF}} = 1-(1-(1-\mathrm{e}^{-\frac{kn}{mr}})^k)^s \tag{2}$$

相较拆分 Bloom Filter，M-BMBF 的误判率与定位哈希函数个数 s 有关，在插入元素相同的情况下，由于 $s \leqslant r$，故 $F_{\text{M-BMBF}} \leqslant F$.

2.3.3　空间性能

Bloom Filter 本身是一种具有高效空间表现的数据结构，它利用位数组很简洁地表示一个集合，相比 B 树和哈希表等其他与数据大小相关的数据结构而言，所占存储空间非常小，故可以常驻内存. 本文在多个候选 Bloom Filter 中选择元素插入位置时采用“最长位匹配”填充算法，在 Bloom Filter 的位空间使用率达到 50%之前可以插入更多的元素，从而节省 Bloom Filter 的存储空间. 同时，从公式(1)和公式(2)的分析可知，在变量参数 m、k 和 r 相同时，如果误判率也控制在相等的情况下，由于 $s \leqslant r$，故 M-BMBF 插入的元素多于拆分 Bloom Filter，从而可以达到节省空间的效果.

3　实验结果与分析

如前文所述，拆分 Bloom Filter 算法为动态经典算法，2BDBF 与本文算法部分思路相近，下面通过实验对比 M-BMBF 算法、2BDBF 算法以及拆分 Bloom Filter 三种算法性能. 本实验各算法使用 Java 语

言编写,在 eclipse 的编译工具下编译运行,使用的运行环境是 Intel(R) Core(TM) Duo CPU,2.00 GB 内存和32 位的 Windows7 操作系统. 为了进行实验测试,搜集30 万个文本文件,并对这些文件进行去重处理,取出其中的 20 万个文件作为数据集进行实验. 在整个实验过程中 Bloom Filter 的长度 m 和哈希函数的个数 k 是固定的,其中 $m=131\ 072$,$k=10$,由于插入元素个数 n 和 Bloom Filter 个数 r 以及定位哈希函数个数 s 是用户可以自定义的三个参数,本文首先充分利用计算机多核 CPU 的计算资源,并行执行 s 个定位哈希函数的计算并通过实验比较和分析了三种算法在元素插入个数,算法消耗时间以及误判率三个典型指标上的性能.

本文首先测试了取不同定位哈希函数个数 s 时对 M-BMBF 的性能影响,主要从三个方面进行分析. 如表 1 所示,在并行执行多个定位哈希函数的计算时,当定位哈希函数 s 发生变化,对插入元素个数以及算法的消耗时间并没有明显影响. 之所以对算法消耗时间没有产生影响,是因为,在多核环境下,只要 CPU 的核数足以支撑每个定位哈希函数的独立计算,那么元素的多个定位哈希函数计算就可以并行执行. 从表中我们还可以知道,当定位哈希函数的个数增加时,误判率也会随着增加,这是因为随着定位哈希函数的增加,元素的查找范围也会增大,这就会导致误判的概率增大. 综合考虑,本文后续实验均取 $s=2$.

表 1 不同定位函数数量对 M-BMBF 插入元素数量、消耗时间以及误判率的影响($k=10$,$m=131\ 072$)

Tab. 1 The element number for insertion, the time-consuming cost and the misjudgment rate of M-BMBF with different located Hash function numbers

定位哈希函数个数 s	插入元素个数		算法消耗时间/ms		误判率	
	$r=8$	$r=16$	$r=8$	$r=16$	$r=8$	$r=16$
2	71 638	144 346	292	565	0.001 79	0.001 84
3	71 825	144 980	301	551	0.002 76	0.002 85
4	72 051	144 512	344	607	0.003 64	0.003 76
5	72 238	144 178	296	575	0.004 21	0.004 51
6	72 339	144 566	321	586	0.005 65	0.005 63
7	72 506	144 482	333	547	0.006 29	0.006 71
8	72 825	144 387	332	602	0.007 22	0.007 57

为了进一步验证 M-BMBF 算法的其它性能,取定位哈希函数个数固定为 $s=2$,在整个实验过程中分别改变元素插入个数 n 和 Bloom Filter 个数 r,在相同的条件下与拆分 Bloom Filter、2BDBF 作比较,分析三种算法的元素插入数量、算法消耗时间以及误判率如下.

图 3 给出了三种算法在存储空间相同的情况下分别取不同的 Bloom Filter 个数时所能插入的元素个数情况. 由图可知,随着 Bloom Filter 个数 r 的增加,当 Bloom Filter 的空间使用达到饱和度时,拆分 Bloom Filter、2BDBF 和 M-BMBF 的元素插入个数均呈稳定增长. 从图中同时可以看出,相较拆分 Bloom Filter 和 2BDBF,由于 M-BMBF 算法每次都选择新置为 1 的个数增加最少的 Bloom Filter 作为元素的插入位置,减缓了达到饱和度的速度,因此在 Bloom Filter 个数相同的情况下,M-BMBF 能够插入的元素更多,这种增加趋势在 Bloom Filter 越多时则越显著.

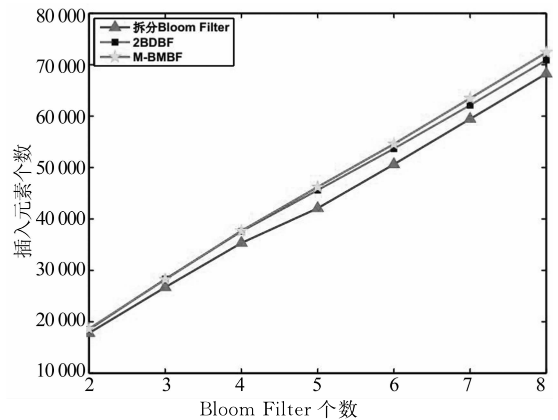

图 3 元素插入个数

Fig. 3 The number of the elements which can be inserted

图 4、图 5 给出了分别改变 Bloom Filter 个数和元素插入个数时三种算法的消耗时间情况. 图 4 为取 $n=10\ 000$ 时不同 Bloom Filter 个数所带来的算法消耗时间结果. 由图可知,当 Bloom Filter 个数较少时,执行 M-BMBF 算法所消耗的时间比拆分 Bloom Filter 和 2BDBF 多,这是由于当要查询一个元素时,M-BMBF 需要计算 2 个额外的哈希函数,随着 Bloom Filter 个数 r 的增加,M-BMBF 算法所消耗的时间仍然趋于稳定,而拆分 Bloom Filter 和 2BDBF 所消耗的时间却随之增加并超过 M-BMBF,而且这种趋势会随着 Bloom Filter 个数的增多越来越大. 这是因为拆分 Bloom Filter 和 2BDBF 的元素查找过程是逐个 Bloom Filter 顺序查找,而采用定位哈希函数的 M-BMBF 却只需要映射到相对应的 2 个 Bloom Filter,并对这两个 Bloom Filter 进行查找即可. 图 5 为固定取 Bloom Filter 个数 $r=8$ 时三种算法随着插入元素个数增加时的算法消耗时间结果. 在 Bloom Filter 大小 m 和哈希函

数个数 k 固定的情况下,随着集合元素个数的增加,算法消耗的时间也逐渐增多,使用拆分 Bloom Filter 算法所消耗的时间与 2BDBF 一致,比 M-BMBF 算法所消耗的时间多,并且趋势越来越大. 这是因为查询一个元素时,使用 M-BMBF 的时间复杂度为 $O(2k)$,而使用拆分 Bloom Filter 和 2BDBF 时最坏情况下的时间复杂度为 $O(kr)$,并且随着集合中元素个数的增加,前两种算法与本文所提算法之间的时间消耗差必然也会越来越大.

综合图4和图5的结果和分析可知,M-BMBF 的算法消耗时间优于拆分 Bloom Filter 和 2BDBF 的算法消耗时间.

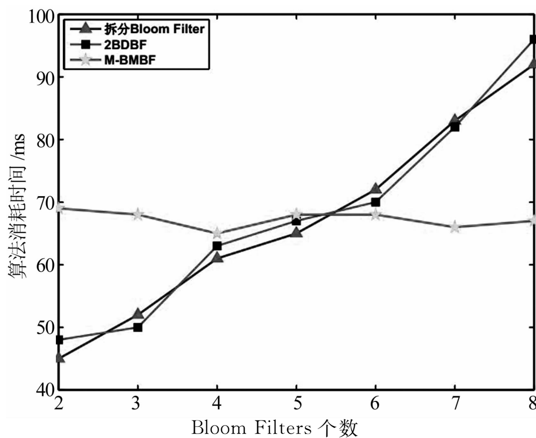

图4　当 $n=10\,000$ 时,拆分 Bloom Filter、2BDBF 和 M-BMBF 的算法消耗时间对比

Fig. 4　When $n=10\,000$, the algorithm time-consuming cost of the Split Bloom Filter, 2BDBF and M-BMBF

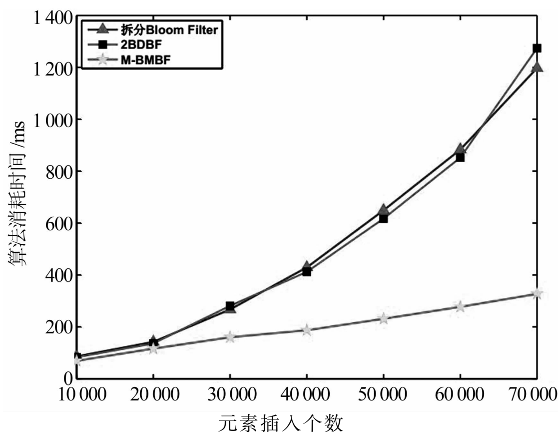

图5　当 $r=8$ 时,拆分 Bloom Filter、2BDBF 和 M-BMBF 的算法消耗时间对比

Fig. 5　When $r=8$, the algorithm time-consuming cost of the Split Bloom Filter, 2BDBF and M-BMBF

误判率是 Bloom Filter 的一个重要衡量指标,图6给出了取 Bloom Filter 个数 r 为8时(2BDBF 的组基为8)随着插入元素的增加使用三种算法时误判率的变化情况. 如图6所示,随着插入元素的增加,三种算法的误判率均随之增加,但 M-BMBF 的误判率在三种算法中是最小的,而拆分 Bloom Filter 是最大的,这是因为将三种算法的误判率控制在相同的情况下,使用 M-BMBF 和 2BDBF 算法能够插入更多的元素,反过来说,当插入元素相同的情况下,M-BMBF 和 2BDBF 的误判率低于拆分 Bloom Filter 的误判率. 同时,M-BMBF 缩小了元素的查找范围,故 M-BMBF 的误判率低于 2BDBF.

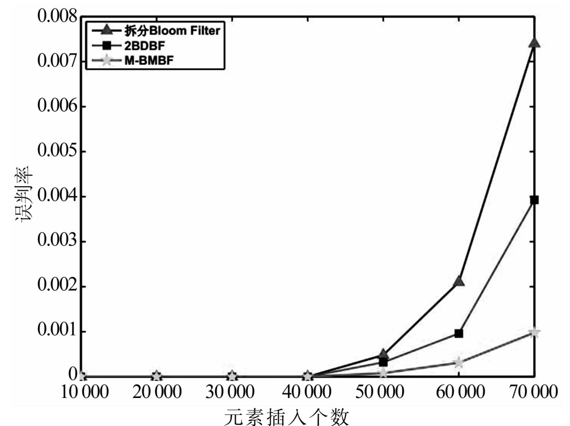

图6　误判率分析

Fig. 6　The analysis of misjudgment rate

4　结　论

本文改变了传统 Bloom Filter 的判满条件,由原来的统计插入元素个数改变为统计 Bloom Filter 中已置1的位置是否达到饱和(即比例占50%),从而能够使判满条件更加适应过滤器长度的变化. 本文提出的 M-BMBF 算法在改变 Bloom Filter 的判满条件的同时利用多个定位哈希函数进行集合元素的插入和查找操作. 为了减少哈希映射的冲突,将 Bloom Filter 进行分组,每个定位哈希函数映射到一组 Bloom Filter. 理论分析和实验结果表明,在存储空间有限的情况下,M-BMBF 既能够减缓空间的利用率,又能够节省算法的消耗时间,同时能够保持较低的误判率. 随着存储元素的不断增加,当 M-BMBF 设定空间无法满足元素的表示时,需要动态地增加 Bloom Filter,如果动态地增加一个 M-BMBF,当需要表示的元素数量很少时,会较大地浪费 M-BMBF 的存储空间,如果在原有的 M-BMBF 基础上增加一组 Bloom Filter,为保证负载均衡,需要重新计算 s,从而增加了计算的时间复杂性,因此,如何动态增加 M-BMBF 的存储空间是下一步需

要研究的问题.M-BMBF 的设计同时可以保证在并行环境中的高效运行,由于高效的检索技术对于大数据环境下的重复数据删除是必不可少的,故接下来的工作是将所提算法应用于典型集群文件系统中以实现重复数据检测.

参考文献

[1] BLOOM B H. Space/time trade-offs in hash coding with allowable errors[J]. Communication of the ACM, 1970,13(7):422－426.

[2] MULLIN J K. Optional semijoins for distributed data system [J]. IEEE Trans Software Eng,1990,16(5):558－560.

[3] 朱桂明,郭得科,金士尧. ODBF:基于操作型衰落 Bloom Filter 的 P2P 网络弱状态路由算法[J]. 计算机学报,2012,35(5):911－917.
ZHU G K, GUO D K, JIN S R. ODBF: A P2P weak state routing scheme based on operative decaying Bloom filter[J]. Chinese Journal of Computers, 2012, 35(5): 911－917. (In Chinese)

[4] LI J, TAYLOR J, SERBAN L, *et al*. Self-organization in peer-to-peer system[C]// Proceedings of the 10th Workshop on ACM SIGOPS European Workshop. New York: ACM, 2002:125－132.

[5] CUENA-ACUNA F M, PEERY C, MARTIN R P, *et al*. PlantP: using gossiping to build content addressable peer to peer information sharing communities[C]//Proc 12th IEEE International Symposium on High performance Distributed Computing. Washington: IEEE, 2003:236－249.

[6] RHEA S C, KUBIATOWICZ J. Probabilistic location and routing[C]//Proceedings of INFOCOM 2002. New York: IEEE Computer Society, 2002:1248－1257.

[7] FAN L, CAO P, J ALMEIDA J, *et al*. Summary cache: a scalable wide-area web cache sharing protocol[J]. IEEE/ACM Trans on Networking, 2000,8(3):281－293.

[8] 李纬,张大方,黄昆,等. 面向大数据处理的高精度多维计数布鲁姆过滤器[J]. 电子学报,2015,43(4):652－657.
LI W, ZHANG D F, HUANG K, *et al*. Accurate multi-dimension counting Bloom filter for big data processing[J]. Acta Electronica Sinica, 2015,43(4):652－657. (In Chinese)

[9] MITZENMAEHER M. Compressed Bloom filters[J]. IEEE/ACM Trans on Networking, 2002,10(5):604－612.

[10] SAAR C, YOSSI M. Spectral Bloom filters[C]// Proceedings of ACM SIGMOD International Conference on Management of Data. San Diego: ACM Press, 2003:241－245.

[11] 肖明忠,代亚菲,李晓明. 拆分型 Bloom Filter[J]. 电子学报,2004,32(2):241－245.
XIAO M Z, DAI Y F, LI X M. Split Bloom filter[J]. Acta Electronica Sinica, 2004,32(2):241－245. (In Chinese)

[12] GUO D, WU J, CHEN H, *et al*. Theory and network applications of dynamic Bloom filters[C]//Proceedings of 25th IEEE International Conference on Computer Communications. Barcelona, Catalunya: Joint Conference of the IEEE Computer and Communications Societies, 2006:23－29.

[13] 唐海娜,林小拉,韩春静. 基于移动指针的数据流冗余消除算法[J]. 通信学报,2012,33(2):7－14.
TANG H N, LIN X L, HAN C J. Duplicate elimination algorithm for data streams with SKIP Bloom filter[J]. Journal on Communications, 2012,33(2):7－14. (In Chinese)

[14] LIM H, LEE J, BYUN H, *et al*. Ternary Bloom filter replacing counting Bloom filter[J]. IEEE Communications Letters, 2017,21(2):278－281.

[15] 肖明忠,王佳聪,闵博楠. 针对动态集的矩阵型 Bloom Filter 表示与查找[J]. 计算机应用研究,2008,25(7):2001－2022.
XIAO M Z, WANG J C, MIN B N. Matrix Bloom filter on dynamic set[J]. Application Research of Computers,2008,25(7): 2001－2022. (In Chinese)

[16] LIU Y, GE X Z, DU D H C, *et al*. Par-BF: a parallel partitioned Bloom filter for dynamic data sets[C]//Proceedings of 2014 International Workshop on Data Intensive Scalable Computing Systems. New Orleans,Piscataway, NJ: IEEE, 2014: 1－8.

[17] WEI J, HONG J, ZHOU K, *et al*. Efficiently representing membership for variable large data sets [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(4): 960－970.

[18] 王勇,云晓春,王树鹏,等. CBFM:支持属性删减的布鲁姆过滤器矩阵多维元素查询算法[J]. 通信学报, 2016,37(3):139－147.
WANG Y, YUN X C, WANG S P, *et al*. CBFM: cutted Bloom filter matrix for multi-dimensional membership query [J]. Journal on Communications, 2016,37(3):139－147. (In Chinese)

[19] 孙智超,徐蕾. 二路平衡动态布隆过滤器[J]. 数学的实践与认识,2014,44(5):199－205.
SUN Z C, XU L. 2-balance dynamic Bloom filter[J]. Mathematics in Practice and Theory, 2014,44(5): 199－205. (In Chinese)