分档布鲁姆过滤器的查询算法

谢 鲲[1)] 闵应骅[2)] 张大方[3)] 谢高岗[2)] 文吉刚[1)]

[1)](湖南大学计算机与通信学院 长沙 410082)

[2)](中国科学院计算技术研究所网络与普适计算研究部 北京 100080)

[3)](湖南大学软件学院 长沙 410082)

摘 要 布鲁姆过滤器是一种能够简洁地表示集合并支持集合查询的数据结构,广泛应用于数据库、网络和分布式系统中.针对现有的布鲁姆过滤器没有考虑查询失效代价这一缺陷,文中提出一种新的代价敏感的分档布鲁姆过滤器查询算法.它将元素根据不同的查询代价分为不同的子集,通过考查每档子集最低查询失效率的关系,建立由每档子集合最低查询失效假阳性概率表示的集合最低查询失效总代价目标函数,使用类目标函数梯度遗传算法获得每档的最优 Hash 函数个数 k_i,完成集合到向量的映射与查找.仿真实验结果表明,使用新结构的查询算法和标准布鲁姆过滤器算法相比,所用的查询计算时间基本相同,因为区分对待集合元素,查询失效总代价仅为标准算法的 27%.

关键词 分档布鲁姆过滤器;计算机网络;分布式计算;分布式消息系统;集合元素查询

中图法分类号 TP393

Basket Bloom Filters for Membership Queries

XIE Kun[1)] MIN Ying-Hua[2)] ZHANG Da-Fang[3)] XIE Gao-Gang[2)] WEN Ji-Gang[1)]

[1)](*College of Computer and Communication*, *Hunan University*, *Changsha* 410082)

[2)](*Networking and Ubiquitous Computing Department*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100080)

[3)](*School of Software*, *Hunan University*, *Changsha* 410082)

Abstract A Bloom Filter is a space-efficient data structure allowing membership queries over sets with allowable errors. It is widely used in databases, networks, and distributed systems. This paper presents a novel Bloom Filter, called Basket Bloom Filter (BBF). The BBF differentiates elements in a data set depending on their query invalidation cost, by clustering elements into different baskets. The total query invalidation cost function is defined. In order to minimize the total query invalidation cost, the genetic algorithm is employed to find the optimal number of hash functions for every basket. Simulation results show that, BBF's total query invalidation cost is 27% of the standard Bloom Filters' while the executing time is almost the same.

Keywords Basket Bloom Filter; computer networks; distributed computing; distributed information system; membership query

---

收稿日期:2005-09-29;修改稿收到日期:2006-12-14.本课题得到国家自然科学基金(60473031,60273070,60403031,90604015)、国家"八六三"高技术研究发展计划项目基金(2005AA121560)和湖南省科技计划项目基金(2006GK3101)资助.**谢 鲲**,女,1978 年生,博士研究生,主要研究方向为可信系统与网络、分布式计算. E-mail: xiekun@hnu.cn. **闵应骅**,男,1935 年生,IEEE Fellow,研究员,博士生导师,主要研究领域为可信计算、集成电路测试、计算机网络. **张大方**,男,1959 年生,博士,教授,博士生导师,主要研究领域为可信系统与网络、网络测试、软件容错、软件测试. **谢高岗**,男,1974 年生,博士,副研究员,主要研究方向下一代互联网、网络测试、移动计算. **文吉刚**,男,1978 年生,博士研究生,主要研究方向为对等计算、流媒体和普适计算.

1 引 言

表示和查找信息是大多数计算机应用程序的核心.近年来,随着计算机的飞速发展,数据库、网络和其他应用中的数据集合规模呈几何增长.元素查询是数据集合中最常见的操作.在集合变得越来越大,访问和表示越来越困难的情况下,如何表示大数据集合,完成大数据集合下的查询成为国内外学术界的挑战性课题.过滤器技术就是要设计精简的数据结构来表示并支持大集合的元素查询.

以分布式网络监测系统为例,为了管理大量的网络监测数据,各个分布式采集节点需要交互包括网络流量、利用率和不同层次级别的统计信息、告警信息等海量网络监测数据.如果直接从一个节点到另一个节点传输如此海量的网络管理信息,将占用大量带宽,可能会造成网络拥塞.如果能够设计一种简洁的结构,存储采集节点的网络健康状态信息,可以大大减少由于采集节点之间的信息交互和查询占用的网络带宽.高速发展的计算机系统中,数据膨胀时,如何借助简单的结构表示集合并支持集合查询,成为分布式信息共享系统中最常见的问题.

布鲁姆过滤器对数据集合采用一个位串表示并能有效支持集合元素的 Hash 查找,是一种能够表示集合、支持集合查询的简洁数据结构,它能够有效地过滤掉不属于集合的元素,因其是由 Bloom 于 1970 年[1] 提出,因此叫布鲁姆过滤器(Bloom filter).布鲁姆过滤器结构的实质是将集合中的元素通过 k 个 Hash 函数映射到位串向量中,对于一个元素只需要保存几个比特.布鲁姆过滤器作为一种集合查询的数据结构,在达到其高效简洁表示集合的同时,却存在某元素不属于数据集合而被指称属于该数据集合的可能性,即假阳性误判,而不存在假阴性误判(属于集合中的元素而误判为不属于集合中)[1].

虽然布鲁姆过滤器结构在元素查询时存在少量假阳性误判,但是由于它对存储空间的高效性,使得该数据结构有很好的实用价值.早期的应用主要集中在数据库操作[2] 和字典查询操作[3],最近,随着网络研究的发展以及新的覆盖网和 P2P 网络应用技术的涌现,布鲁姆过滤器查询算法已经越来越广泛地应用于网络中,包括覆盖网和 P2P 网节点协作交互[4-6]、资源路由[7]、数据帧路由标签[8]、网络测量管理[9-10].

目前布鲁姆过滤器查询算法主要有:标准的布鲁姆过滤器算法[1]、计数器布鲁姆过滤器算法[11]、压缩布鲁姆过滤器算法[12] 以及光谱布鲁姆过滤器算法[13] 等.Brooder 和 Mitzenmacher 在文献[14]中指出,虽然布鲁姆过滤器在 1970 年就已经产生,且应用广泛,但还有很大的研究改进空间.现有的布鲁姆过滤器算法没有考虑集合的查询代价,它们认为集合元素在查询时所付出的 I/O 操作代价相同,而在实际中,集合中的元素由于查询失效(假阳性发生),元素的查询失效所额外付出的 I/O 操作代价由于元素在集合中和作用和地位不同而不同.以往的算法因为没有考虑到查询代价,它们对集合中的元素都均一对待,为每个元素分配同样多的 Hash 映射函数,每个元素查询失效率相同,导致集合的查询总代价比较高.缺乏查询代价考虑并且缺乏元素的区分对待,使当前布鲁姆过滤器算法在进一步节约资源方面有改进的可能.

本文提出一种新的代价敏感的分档布鲁姆过滤器查询算法(以下简称 BBF 算法).新算法将元素根据不同的查询代价分为不同的子集,通过考查每档子集最低查询失效率的关系,建立由每档子集合最低假阳性误判率表示的集合最低查询失效总代价目标函数,使用类目标函数梯度遗传算法获得每档的最优 Hash 函数个数 k_i,再完成集合到向量的映射与查找.事实上,当查询出现假阳性失效时,不同的元素由于其在集合中的地位和作用不同,查询失效时所需要承担的额外 I/O 操作代价也不尽相同.如果将查询代价高的元素降低其失效率,增加代价低的元素的失效率,此时的集合查询失效总代价也可能降低,这是本文算法的出发点.仿真实验表明,新算法可以降低集合查询失效总代价 40% 以上,大大减少用布鲁姆过滤器算法时的资源消耗.

文章第 2 节介绍布鲁姆过滤器的相关研究;第 3 节提出分档布鲁姆过滤器查询算法,得出集合最低查询失效总代价目标函数;第 4 节给出利用类目标函数梯度遗传算法优化分档布鲁姆过滤器的设计,确定每档应分配的 Hash 函数个数;第 5 节比较分档布鲁姆过滤器算法和标准布鲁姆过滤器算法性能;第 6 节给出分档布鲁姆过滤器算法的应用实例仿真;第 7 节总结全文.

2 布鲁姆过滤器算法原理

1970 年 Bloom[1] 提出了标准的布鲁姆过滤器算法(以下简称 SBF 算法).其具体描述如下:集合

$S=\{s_1,s_2,\cdots,s_n\}$共有 n 个元素通过 k 个 Hash 函数 $h_1,h_2,\cdots,h_k$映射到长度为 m 的向量 $\boldsymbol{V}$ 中. 通常说布鲁姆过滤器表示的摘要信息就是指这个向量 $\boldsymbol{V}$. 每一个 Hash 函数相互独立且函数的取值范围为 $\{0,1,2,\cdots,m-1\}$. 集合到向量的映射过程如下：

(1) 向量 $\boldsymbol{V}$ 初始化，将向量 $\boldsymbol{V}$ 所有 bit 位置 0.

(2) 对于每一个元素 s_i，计算 $h_j(s_i)(1\leq j\leq k)$，将向量中的这 k 个位置置位，如图 1. 此时发现向量中任何一个位置可能会多次置位，但仅第一次置位有效.

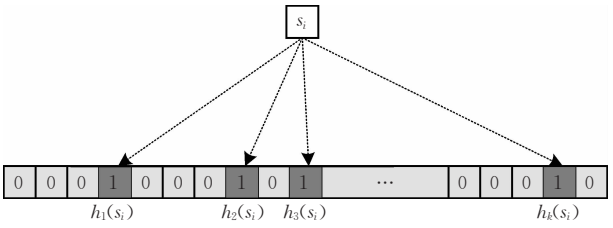


图 1　布鲁姆过滤器算法集合元素到向量的映射

集合元素查询：进行元素是否属于集合中的查询判断时，对于给定的元素 x，检查向量 $\boldsymbol{V}$ 的 k 个位置$(h_1(x),h_2(x),\cdots,h_k(x))$. 如果 k 个位置全部为 1，则 x 可能在集合中；如果有一个位置为 0，则 x 一定不在集合中.

布鲁姆过滤器结构在进行元素查询时存在假阳性误判，即将不属于集合的元素误判断成属于集合中. 如图 2 所示，假设元素 x 不属于集合，属于集合的元素 a,b,c,d 分别使得 $h_1(x),h_2(x),\cdots,h_k(x)$ 置位，x 就被错误地判断属于集合中，假阳性误判发生. 为了在下文中说明方便，我们用三元组$\{n,m,k\}$形式化表示标准布鲁姆过滤器算法，n 为集合 S 的元素个数，m 为向量 $\boldsymbol{V}$ 的长度，k 为 Hash 函数个数.

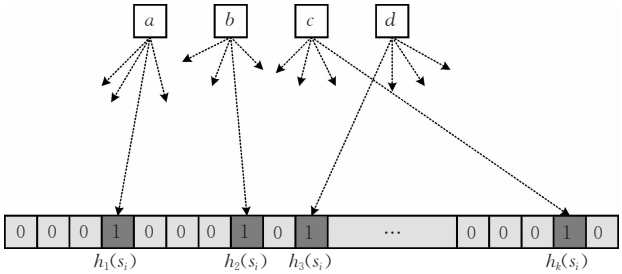


图 2　布鲁姆过滤器出现的假阳性误判

为了分析方便，我们用 p 表示向量 $\boldsymbol{V}$ 中位为 0 的概率，用 $1-p$ 表示为 1 的概率. 同时，在本文的讨论中，假设 Hash 函数取值服从均匀分布，当集合中所有元素都映射完毕后，$\boldsymbol{V}$ 向量任一位为 0 的概率为

$$p'=\left(1-\frac{1}{m}\right)^{kn}\approx \mathrm{e}^{-kn/m}=p \tag{1}$$

那么元素出现假阳性的概率为[12]

$$f=(1-p)^k=\mathrm{e}^{k\ln(1-\mathrm{e}^{-kn/m})} \tag{2}$$

令 $g=k\ln(1-\mathrm{e}^{-kn/m})$，可知函数 g 和 f 可以同时达到最小值，对 g 取 k 的导数：

$$\frac{\mathrm{d}g}{\mathrm{d}k}=\ln(1-\mathrm{e}^{-kn/m})+\frac{kn}{m}\frac{\mathrm{e}^{-kn/m}}{1-\mathrm{e}^{-kn/m}} \tag{3}$$

令$\frac{\mathrm{d}g}{\mathrm{d}k}=0\Rightarrow k_{\min}=\ln 2\left(\frac{m}{n}\right)$，当 $k=k_{\min}$时，那么元素出现假阳性率最小值为

$$f(k_{\min})=(1/2)^{k_{\min}}=(0.6185)^{m/n} \tag{4}$$

k 是 Hash 函数的个数，应为整数，即

$$k=\lceil \ln 2(m/n)\rceil \tag{5}$$

使用布鲁姆过滤器完成集合存储，只需要为每个元素平均保存 m/n 位，十分简洁. 正因为此，布鲁姆过滤器已经广泛应用于分布式系统中，尤其是能容忍少量查询假阳性误判且存储空间受限的应用.

3　分档布鲁姆过滤器查询算法

公开发表的文献表明，目前存在的布鲁姆过滤器各种扩展算法，都有一个共同的弊端：每一个元素都被一致考虑，因此每个元素出现查询失效假阳性率相同，使得集合查询总代价较高. 但在实际应用中，不同的元素出现查询失效所承担的额外操作代价并不相同，对于那些额外代价高的元素，降低查询失效率，对于代价低的元素，相应少许增加查询失效率，就可能大大降低集合查询失效总代价，这正是本文工作的出发点.

本文从查询代价出发来研究集合查询算法，提出了分档布鲁姆过滤器查询算法. 分档布鲁姆过滤器算法按照集合元素在实际应用中的作用及查找失效的代价区别，为不同代价元素分配不同个数 Hash 函数，然后按照 Hash 运算完成集合映射和元素查找. 其思路是将元素按照不同的查询失效代价分为不同的档(子集合)，通过对高代价子集合分配多数目的 Hash 函数，降低查询失效率；对低代价子集合分配少数目 Hash 函数，稍许增加查询失效率，使集合查询总代价最小.

设集合 S 包含 L 档：

$$S=\{\{S_1\},\{S_2\},\cdots,\{S_L\}\},$$

每档子集合的元素个数为 $n_i=|S_i|$，子集合 S_i中的元素发生查询失效时，所需付出的额外 I/O 操作代价为 $c_i(1\leq i\leq L)$，子集合 S_i的 Hash 映射函数个数为 $k_i(1\leq i\leq L)$，对应的假阳性率为 $f_i(1\leq i\leq L)$.

定义 1. 集合查询失效总代价为各子集合查询失效代价之和：

$$F = n_1 f_1 c_1 + n_2 f_2 c_2 + \cdots + n_L f_L c_L \quad (6)$$

为了下文表述方便，我们用$\{\{n_1, n_2, \cdots, n_L\}, \{c_1, c_2, \cdots, c_L\}, m, \{k_1, k_2, \cdots, k_L\}\}$形式化表示分档布鲁姆过滤器算法.

定理 1. 分档布鲁姆过滤器算法集合最低查询失效总代价表达式为

$$F_{\text{baskets}}(L) = \sum_{i=1}^{L} n_i c_i f_i = \sum_{i=1}^{L} n_i c_i \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_i m}{\sum_{j=1}^{L} n_j r_j}\right)} \quad (7)$$

其中 $r_i = \ln(f_i)(1 \le i \le L)$，而且每档元素需要映射的 Hash 函数个数为

$$k_i = \ln 2 \left(\frac{r_i m}{\sum_{j=1}^{L} n_j r_j}\right) \quad (8)$$

下面进行定理证明.

3.1 两档布鲁姆过滤器($L=2$)

$BBF = \{\{n_1, n_2\}, \{c_1, c_2\}, m, \{k_1, k_2\}\}$，集合中所有元素映射到向量 $\boldsymbol{V}$ 后，向量 $\boldsymbol{V}$ 中任一位为 0 的概率为

$$\left(1-\frac{1}{m}\right)^{k_1 n_1}\left(1-\frac{1}{m}\right)^{k_2 n_2} = \left(1-\frac{1}{m}\right)^{k_1 n_1 + k_2 n_2} \approx e^{-\frac{k_1 n_1 + k_2 n_2}{m}} = p \quad (9)$$

第一档元素出现假阳性率为

$$f_1 = e^{(k_1 \ln(1-p))} \quad (10)$$

第二档元素出现假阳性率为

$$f_2 = e^{(k_2 \ln(1-p))} \quad (11)$$

令 $g_1 = k_1 \ln(1-p), g_2 = k_2 \ln(1-p)$，则

$$g_1 : g_2 = r_1 : r_2 = k_1 : k_2 \quad (12)$$

对 g_2 求 k_2 导数，当 g_2 取最小值时

$$k_{2\min} = \ln 2\left(\frac{r_2 m}{r_1 n_1 + r_2 n_2}\right) \quad (13)$$

将 $k_{2\min}$ 代入式(11)得 f_2 的最小值为

$$f_{2\min} = (1-p)^{k_{2\min}} = \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_2 m}{r_1 n_1 + r_2 n_2}\right)} \quad (14)$$

由式(12)中 k_1, k_2 的比率关系得 $k_1 = k_2 r_2 / r_1$，所以，

$$k_1 = \ln 2\left(\frac{r_1 m}{r_1 n_1 + r_2 n_2}\right) \quad (15)$$

同理，我们取 g_1 为最小值时，和前面的过程相似，可以得到

$$k_{1\min} = \ln 2\left(\frac{r_1 m}{r_1 n_1 + r_2 n_2}\right) \quad (16)$$

那么 f_1 的最小值为

$$f_{1\min} = (1-p)^{k_{1\min}} = \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_1 m}{r_1 n_1 + r_2 n_2}\right)} \quad (17)$$

由式(15)和式(16)我们发现 $k_1 = k_{1\min}$. 得出结论，两档子集可以同时获得每档最小假阳性率，将此最小假阳性率代入式(6)，得

$$F_{\text{baskets}}(L) = n_1 c_1 f_{1\min} + n_2 c_2 f_{2\min} = \sum_{i=1}^{2} n_i c_i \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_i m}{\sum_{j=1}^{L} n_j r_j}\right)} \quad (18)$$

集合查询失效总代价转换为依赖于 $r-pair(r_1, r_2)$ 最低查询失效总代价函数，式(7)得证，下面就更一般的情况进行证明.

3.2 多档布鲁姆过滤器($L>2$)

$BBF = \{\{n_1, n_2, \cdots, n_L\}, \{c_1, c_2, \cdots, c_L\}, m, \{k_1, k_2, \cdots, k_L\}\}$.

集合中 L 档元素都映射到 $\boldsymbol{V}$ 向量后，$\boldsymbol{V}$ 向量中任一位为 0 的概率为

$$\left(1-\frac{1}{m}\right)^{k_1 n_1} \cdots \left(1-\frac{1}{m}\right)^{k_L n_L} = \left(1-\frac{1}{m}\right)^{\sum_{i=1}^{L} k_i n_i} \approx e^{-\frac{\sum_{i=1}^{L} k_i n_i}{m}} = p \quad (19)$$

每档子集的元素出现假阳性率为

$$\begin{aligned} f_1 &= (1-p)^{k_1} = e^{(k_1 \ln(1-p))} \\ f_2 &= (1-p)^{k_2} = e^{(k_2 \ln(1-p))} \\ &\cdots \\ f_L &= (1-p)^{k_L} = e^{(k_L \ln(1-p))} \end{aligned} \quad (20)$$

令 $g_1 = k_1 \ln(1-p), \cdots, g_L = k_L \ln(1-p)$，那么

$$g_i : g_j = k_i : k_j = r_i : r_j \quad (1 \le i, j \le L) \quad (21)$$

g_1 是 k_1 的函数，p 也是 k_1 的函数，对 g_1 取 k_1 的导数，当 g_1 取最小值时

$$k_{1\min} = \ln 2\left(\frac{m}{\sum_{i=1}^{L} n_i r_i / r_1}\right) \quad (22)$$

将 $k_{1\min}$ 代入式(20)得 f_1 最小值：

$$f_{1\min} = (1-p)^{k_{1\min}} = \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_1 m}{\sum_{i=1}^{L} n_i r_i}\right)} \quad (23)$$

从式(21)$k_i = k_1 r_1 / r_i$，得到 k_i 如式(8).

类似 3.1 节推导，式(8)所示 k_i 可以使其他每档子集同时达到最小假阳性率：

$$f_{i\min} = (1-p)^{k_{i\min}} = \left(\frac{1}{2}\right)^{\ln 2\left(\frac{r_i m}{\sum_{j=1}^{L} n_j r_j}\right)} (1 \le i \le L) \quad (24)$$

用每档最小假阳性率 $f_{i\min}$ 代入查询失效总代价

式(6)，得到集合最低查询失效总代价式(7)，定理 1 得证. 此时最小假阳性率由参数 $r-pair(r_i,r_j)$ $(1\leq i,j\leq L)$决定，集合最低查询失效总代价目标函数为 $r-pair(r_i,r_j)$ $(1\leq i,j\leq L)$的函数.

4　分档布鲁姆过滤器查询算法中 k_i 的获得

上节定理考查了每档子集合最小假阳性率和算法设计的关系，发现每档子集合可以同时达到最小假阳性率，这是一个十分有趣的现象. 将每档子集的最小假阳性率代入集合查询失效总代价表达式，得出集合最低查询失效总代价目标函数，该函数由参数 $r-pair(r_i,r_j)(1\leq i,j\leq L)$决定. 本节主要讨论在 BBF 中如何优化设计，获得每档子集分配的 Hash 函数个数 k_i.

定义 2　分档布鲁姆过滤器查询算法查询总代价为式(7)的最小值.

集合最低查询失效总代价目标函数如式(7)所示，发现 n_i 和 c_i 是独立于 k_i 的常数. 要想使式(7)表示的集合最低查询失效总代价最小，就必须求得 $r_1,r_2,\cdots,r_L$ 使得式(7)最小，然后通过式(8)计算出 k_i. 问题转化为求连续函数的无约束极小值，采用遗传算法求解[15,16]. 算法的步骤如下.

算法.

1. 随机产生规模为 n 个二进制编码染色体，每个染色体代表一个 $r_1,r_2,\cdots,r_L$ 序列；

2. 用类目标函数梯度适应度函数计算每个染色体的适应值；

3. 检查染色体是否已经达到结束条件，如是，转到步 6，返回最优的染色体，否则继续；

4. 使用选择、交叉、变异，产生新一代 n 个染色体；

5. 利用新一带染色体代替老的染色体，转到步 2；

6. 将染色体解码，获得 $r_1,r_2,\cdots,r_L$；

7. 通过式(8)计算每档的映射函数个数 $k_i(1\leq i\leq L)$.

文献[16]提出一类新的改进的适应度函数的遗传算法——目标函数梯度的遗传算法. 将函数在搜索点的函数值及其变化率加入适应度函数，使得按概率选择的染色体不但具有较小的函数值，而且具有较大的函数值变化率，从而来提高遗传算法的收敛速度. 受到该算法的启发，我们设计了类目标函数梯度的遗传算法来求解 k_i，也是通过更改适应度函数完成.

考查式(7)的每一子项 $n_ic_if_i$，n_i 和 c_i 是相对于 f_i 独立的常数，所以乘积 n_ic_i 也是相对于 f_i 独立的常数. 为了使得式(7)最小，对于 n_ic_i 乘积越大的档，可以降低其假阳性率 f_i. 从式(20)看出，f_i 越小 k_i 就越大，也就是说，降低假阳性率就需要为该档分配更多的 Hash 函数进行映射，从式(21)可以，增大 k_i，就需要增大 r_i. 鉴于上面分析，当集合查询失效总代价到最小时，n_ic_i 越大，对应的 r_i 也应该越大. 所以本文设计类目标函数梯度适应度函数，首先将元组 (n_ic_i,r_i)根据 n_ic_i 从小到大排列，取此时 r_i 的逆序数代入适应度函数.

定义 3.　如果在一个排列中，某两个数的先后顺序与标准顺序相反，则称有 1 个逆序. 一个排列的逆序的总数称为排列的逆序数，计为 $inverse_number=\tau(r_1,r_2,\cdots,r_L)$①.

这里的适应度函数用类目标函数梯度适应度函数：

$$F(L)=\lambda f(L)+(1-\lambda)f(L)/inverse\_number \tag{25}$$

其中：

$$f(L)=\begin{cases}C_{\max}-F_{\text{basket}}(L), & f(L)<C_{\max}\\ 0, & f(L)\geq C_{\max}\end{cases} \tag{26}$$

λ 为一可调整权值. 新的适应度函数充分考虑了式(7)达到最小值的理想状态，如此设计是为了避免利用遗传算法求解函数优化过程随机漫游的现象，使得染色体的选择按照“好”染色体方向发展，进而加快遗传算法的收敛速度，这在下一节算法性能分析实验中得以证实.

5　性能评估

我们将从下面三种性能指标来比较分档布鲁姆过滤器算法和标准布鲁姆过滤器算法的性能：

(1) 布鲁姆过滤器结构长度. 它在算法中用于表示集合的存储空间大小，即向量 $\mathbf{V}$ 的长度. 一旦算法已经选定，这就是一个常数；

(2) 计算时间. 它指集合元素映射时，需要进行 Hash 运算的时间，取决于映射 Hash 函数的个数 k 或者 k_i；

(3) 集合查询失效总代价，如定义 1.

无论是分档算法还是标准算法，存储结构大小 m($\mathbf{V}$ 向量长度)在算法初始化时就已确定，如两者

① $\tau(1,2,3,4,5)=0+0+0+0+0=0$，$\tau(1,4,2,3,5)=0+2+0+0+0=2$

取值相同,则所需存储大小相等.下面我们就计算时间与失效代价来比较分档算法和标准算法的性能.(容易看出,其他布鲁姆过滤器各种扩展算法由于没有区分对待查询元素,因此在后两项指标上的分析应该和标准算法基本一致,所以我们直接用标准算法来比较).

5.1 计算时间比较

计算时间是指用布鲁姆过滤器算法进行集合元素映射,所需要的 Hash 运算的总时间.我们假设完成一次 Hash 运算所需时间为 T(每次 Hash 运算的运行环境不尽相同,所以从技术上来说统一定义时间为 T 并不完全正确,但是,近似将每次 Hash 运算时间假设相等,并不会影响算法的比较).

对于标准布鲁姆过滤器算法 $SBF=\{n,m,k\}$,进行集合到向量的映射时,每个元素需要进行 k 次 Hash 运算,那么其总的计算时间为

$$C_{\text{standard}} = n \cdot k \cdot T \tag{27}$$

对于分档算法 $BBF=\{\{n_1,n_2,\cdots,n_L\},\{c_1,c_2,\cdots,c_L\},m,\{k_1,k_2,\cdots,k_L\}\}$,子集 S_i 的元素需要进行 Hash 运算 k_i 次,计算时间为

$$C_{\text{basket}} = (n_1k_1 + n_2k_2 + \cdots + n_Lk_L) \cdot T \tag{28}$$

因为两个算法是在同一个集合上进行,所以 $n_1+n_2+\cdots+n_L=n$,那么式(27)可以改写为

$$C_{\text{standard}} = (n_1 + n_2 + \cdots + n_L) \cdot k \cdot T \tag{29}$$

将 $k_i=\ln 2\left(\dfrac{r_i m}{\sum_{j=1}^{L} n_j r_j}\right)$(见式(8)),代入(28),得

$$C_{\text{basket}}=(n_1k_1 + n_2k_2 + \cdots + n_Lk_L)\cdot T = \ln 2\left(\frac{n_1r_1m}{\sum_{j=1}^{L} n_jr_j} + \frac{n_2r_2m}{\sum_{j=1}^{L} n_jr_j} + \cdots + \frac{n_Lr_Lm}{\sum_{j=1}^{L} n_jr_j}\right)\cdot T = \ln 2\left(\frac{n_1r_1 + n_2r_2 + \cdots + n_Lr_L}{\sum_{j=1}^{L} n_jr_j}\right)m \cdot T = \ln 2 \cdot m \cdot T \tag{30}$$

将 $k=\ln 2\left(\dfrac{m}{\sum_{i=1}^{L} n_i}\right)$代入式(29),得

$$C_{\text{standard}} = \ln 2 \cdot m \cdot T \tag{31}$$

很明显,分档布鲁姆过滤器算法和标准布鲁姆过滤器算法两者计算时间相同.虽然我们在 BBF 中为不同的子集合分配了不同数目的 Hash 函数,并没有增加集合映射时 Hash 运算的总时间.

5.2 集合查询总代价比较

查询代价是评估查询算法性能的一个重要指标.查询代价可以分为两类:正常代价和失效代价.拿布鲁姆过滤器算法来说,正常代价是计算时间的线性函数,由 5.1 节证明得知两算法的正常查询代价相同.而由于布鲁姆过滤器算法存在假阳性误判,必然会产生查询失效,在使用布鲁姆过滤器进行集合元素查询时,就必须承担因查询失效额外付出的 I/O 操作代价,在下文就两算法的集合查询失效总代价进行比较.对于分档布鲁姆过滤器查询算法,分档布鲁姆过滤器查询失效总代价为式(7)的最小值.对于标准布鲁姆过滤器算法,如果我们同样考虑不同的子集合有不同的查询失效代价,那么其查询失效总代价为

$$F_{\text{standard}}(L) = (n_1c_1 + n_2c_2 + \cdots + n_Lc_L)\cdot f = \sum_{i=1}^{L} n_ic_i\left(\frac{1}{2}\right)^{\ln 2\left(\frac{m}{\sum_{j=1}^{L} n_j}\right)} \tag{32}$$

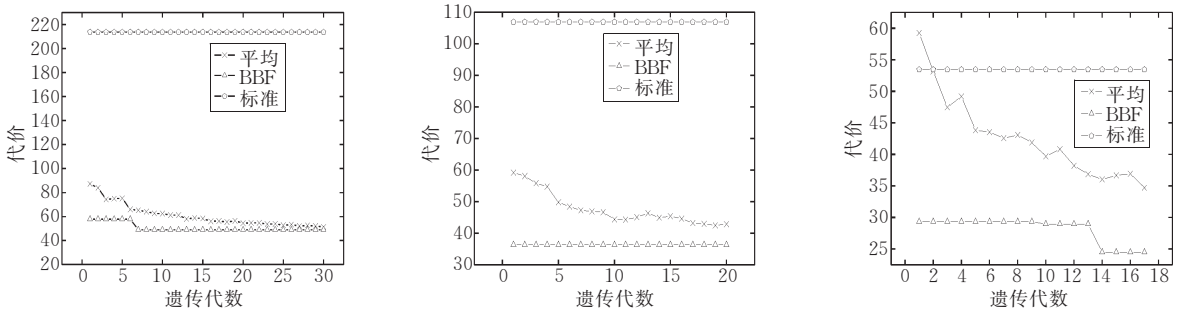
对于标准布鲁姆过滤器算法,给定 n_i, c_i 和 m,集合查询失效总代价 $F_{\text{standard}}(L)$ 就可以直接计算获得.而对于分档布鲁姆过滤器算法,我们通过第 4 节的优化求解过程可以获得每档的 k_i 和总代价.下文给出一些数字实例来进行两算法比较.

图 3 是 3 档布鲁姆过滤器查询算法的集合查询失效总代价比较.横坐标表示利用遗传求解的遗传迭代次数,纵坐标表示查询失效代价.虽然标准算法和迭代次数没有任何关系,但是为了比较,我们用直线表示,画在同一个图中."平均"和 BBF 曲线分别表示分档布鲁姆过滤器算法优化过程中每一代的代价平均值和最小值.在进行实验时,集合元素的个数和元素查询失效代价随机产生.图 3(a)、3(b)和 3(c)进行集合映射的向量长度分别取 900bit、1050bit 和 1200bit.从图中可以看出,分档算法查询失效总代价大大小于标准算法.如图 3(a),平均每个元素仅仅需要 5 位表示,即 $m/n=900/(55+30+95)=5$,8 次遗传迭代后,分档算法的代价为 48.98438,比标准算法减少查询代价 77.1%,同时算法应用的每档最优 Hash 函数个数为 $\{k_1,k_2,k_3\}=\{6,6,2\}$.图 3(b)发现,算法的收敛速度十分迅速,只需要经过一代迭代,分档算法的的查询代价就达到最小值为:36.36719 和标准算法相比降低 66%.图 3(c)中,经过 14 次迭代,分档算法达到最小值为 24.49219,此时通过染色体解码得到的每档 Hash 函数个数为 $\{k_1,k_2,k_3\}=\{7,7,3\}$.图 4 给出了 6 档查询代价比较.这些代价比较图表明,新的分档布鲁姆过滤器查询算法能够大大降低布鲁姆过滤器查询

算法的查询失效代价，利用分档来区分对待不同的元素，是一种有效改进.

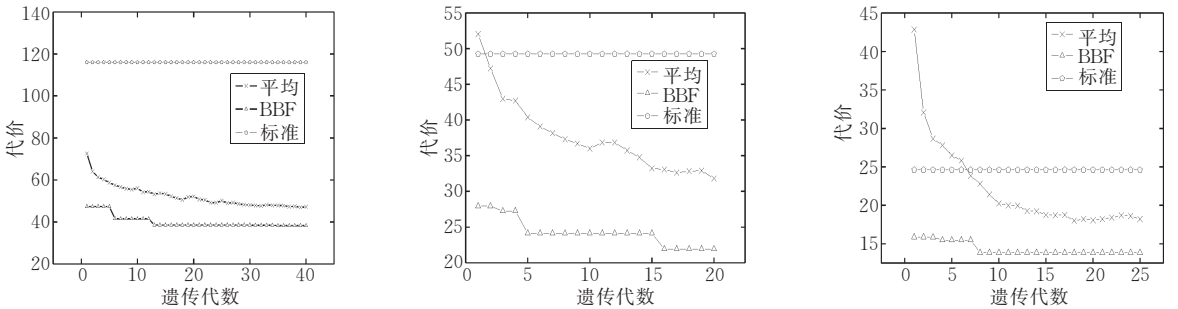
图 3(a)、3(b)和 3(c)集合映射的向量长度从 900bit 到 1050bit 到 1200bit，由小到大变化，相应的标准算法和分档算法的查询失效总代价为{213.75,106.875,53.4375}和{48.9844,36.36719,24.49219}，代价变化趋势是由大到小. 因此，对于布鲁姆过滤器算法来说，向量长度 m 越大，查询代价就越低，这是一个通用的结论. 同时发现采用类目标函数梯度遗传算法求解，染色体向“好”的方向发展，算法的收敛速度快，只需要少数几步就可以求得算法的最优值.

为了更一般地评估分档和标准算法的查询代价，我们又进行了多次实验，实验的分档子集个数从 2～10 档，实验的部分结果如表 1 所示.



(a) 向量长度 =900bit　(b) 向量长度 =1050bit　(c) 向量长度 =1200bit

图 3　3 档布鲁姆过滤器查询算法的集合查询失效总代价比较(BBF={{55,30,95},{19,19,1},m,{k_1,k_2,k_3}})



(a) 向量长度 =1800bit　(b) 向量长度 =2100bit　(c) 向量长度 =2400bit

图 4　6 档布鲁姆过滤器查询算法的集合查询失效总代价比较(BBF={{80,28,25,84,27,38},{9,10,16,1,31},m,{k_1,k_2,k_3,k_4,k_5,k_6}})

表 1　2～10 档数据标准算法和分档算法总代价比较

档数	子集元素个数	子集元素查询代价	SBF 总代价	BBF 总代价	比率/%
2	{31,46}	{20,2}	22.2500	7.7188	35
3	{95,56,28}	{17,2,1}	219.3750	67.2344	31
4	{91,47,23,32}	{14,6,9,1}	112.1875	51.1875	46
5	{48,66,37,30,82}	{19,11,7,5,1}	133.0625	72.2500	54
6	{40,34,26,48,94,42}	{20,15,15, 4, 2, 2}	135.2500	53.2813	39
7	{61,68,63,45,44,81,77}	{16,14,12,15, 6, 2, 2}	492.3750	256.3438	52
8	{64,27,36,79,33,24,34,26}	{10,17, 9, 3, 6, 7, 3, 1}	67.3125	37.6992	56
9	{71,84,39,91,48,63,45,29,77}	{16,13,19, 6, 3, 2, 2, 3, 1}	504.8750	260.3750	52
10	{34,32,42,38,23,42,27,92,36,22}	{20,20,13,10,15, 8,11, 3, 6, 3}	118.1875	62.7734	53

表 1 是 2～10 档数据使用标准布鲁姆过滤器算法和分档算法产生的查询失效总代价比较. 每档元素个数和代价都是随机产生. 第一列为档数；每档子集的元素个数和元素查询失效代价为第 2、3 列；第 4、5 列分别表示标准算法和分档算法的集合查询失效总代价；最后一列表示分档算法失效总代价占标准算法代价的比率. 从表中我们发现这些比率都小于 60%，分档算法和标准算法相比，失效总代价至少降低 40%，使用分档算法可以使集合失效总代价显著减少，大大提高布鲁姆过滤器算法性能.

6　分档布鲁姆过滤器算法在协作式缓存文件系统中的应用探讨和仿真实验

Harvest[17] 项目是由多个 Cache 代理服务器组成的层次式缓存系统，采用 ICP(Internet Cache

Protocol)协议进行 Cache 系统分级互连. 在缓存系统内部,各缓存代理服务器间通过交换 ICP 报文,确定请求目标在哪台缓存代理服务器中,然后利用 HTTP 获得请求内容. ICP 将缓存系统中代理服务器的关系定义为两类[19]:一类是父子关系,子服务器没有命中的请求可以转交给父服务器继续请求;另一类是兄弟关系,一个兄弟服务器不会为另一个兄弟服务器代转请求,只做命中或未命中应答. ICP 协议是 NLANR 实验室做 Harvest 项目时设计的,ICP 协议已广泛应用于 Squid 和 NetCache 系统中.

ICP 通信是建立在查询/应答基础上的,通过兄弟之间的协作,可以减少到外部网络(或上层缓存服务器)的通信量,然而兄弟缓存服务器之间通信开销却很大,当一个缓存服务器没有用户要求的文件时,必须向兄弟同胞广播 ICP 请求,当同胞数增多时,通信量的开销和处理能力的开销也随着增加,假设有 n 个兄弟,如果时间 t 每个代理平均需要查找 m 个文件,则需要发送的报文请求量为 $m\times n\times(n-1)$,这将占用大量的网络带宽,导致网络拥塞. 由于必须等待所有的同胞都回答没有响应的请求,才向父服务器发送请求,网络延时也相应增加.

针对 ICP 广播请求大大占用网络带宽的缺点,Summary Cache[11] 和 Cache Digest[18] 通过保存其它缓存代理文件目录摘要来减少请求数,当有新的请求时,首先检查本身的缓存和其他兄弟缓存的摘要,如果其他兄弟缓存摘要里有文档信息,则向相应的兄弟缓存代理服务器请求以获取文件,如果没有,就直接向上级服务器发送请求. Summary Cache 的文件目录用布鲁姆过滤器表示,将文件目录存储到代理的高速缓存中,可以加快文件查找过程,而高速缓存价格昂贵且容量有限,不适合存放大数据结构,布鲁姆过滤器表示的文件目录摘要简洁,查找方便,而且缓存代理之间的交互也是布鲁姆过滤器表示的文件目录,可以减少交互时的带宽消耗. 因为 Summary Cache 的布鲁姆过滤器对待查询文件都是一致的考虑,并没有区分对待,所有的文件平均查找命中时间和平均查找失效概率相同,这对于没有区分文件的缓存系统来说,是完全可以胜任的,但是,当缓存系统的文件有了区分时,对于一些十分关键的文件来说,必须特别重视,降低其查询失效率,降低其查询失效后对系统的影响,那么 Summary Cache 对文件的一致对待已经不能满足要求.

我们以图 5 基于 Harvest 文件系统结构来说明分档布鲁姆过滤器的应用. 缓存系统采用层次式结构,每层由兄弟缓存代理服务器组成,兄弟缓存代理服务器通过协作完成对文件请求的响应,这种缓存代理结构可以将纯层次代理模型和分布式代理模型的优点结合[19-21].

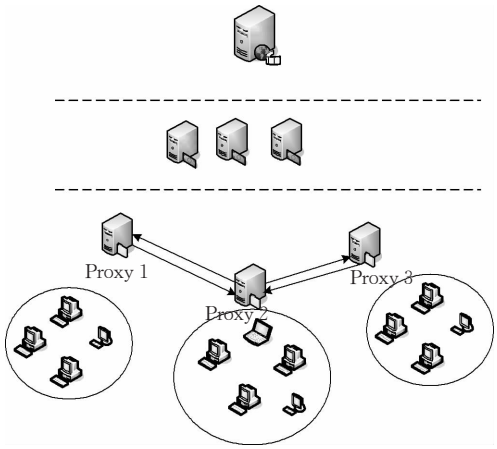


图 5 基于 Harvest 的协作式缓存文件系统

和以往的文件系统区别的是,该协作式缓存文件系统是安全敏感文件系统,出于安全原因,系统中工作站主机会不定期地更新操作系统补丁或更新防病毒软件的病毒库,同时,工作站之间可能有一些普通文件的交换. 因此,文件从对系统安全影响的角度出发,可分为四类:关键系统补丁文件、病毒库更新文件、系统运行文件、普通用户文件. 当网络中系统出现严重安全漏洞时如不及时打上系统补丁,就可能导致系统出现崩溃,甚至央及整个网络无法正常运行(如 worm 病毒针对系统漏洞的攻击会导致网络阻塞). 在这个安全敏感的文件系统中,对不同类别的文件,我们需要按照其对系统安全的不同影响,区别对待. 对于关键系统补丁文件,我们需要加快其查找过程,降低其查找失效率,进而提高系统整体安全. 因此,我们使用分档布鲁姆过滤器来表示文件目录摘要,用以调整不同类别文件的查询失效概率,从而使得整个系统的失效总安全代价降低,提高系统的安全性. 和 Summary Cache 类似,此缓存系统中兄弟缓存代理通过定期交换分档布鲁姆过滤器表示的缓存摘要文件来获得其他兄弟缓存文件目录,其文件下载过程也和 Summary Cache 类似:

(1) 客户将文件下载请求传输到其负责的代理服务器,文件目录用 BBF 表示.

(2) 代理首先在本地缓存目录 BBF 查找文件,如找到,就到自身的硬盘获取数据,如果没有,就查找其兄弟缓存目录 BBF.

(3) 如果在某个兄弟缓存目录 BBF 上找到文件,将文件请求发到该兄弟代理,从其硬盘中获取数据.

(4)如果同级的代理服务器没有对应文件，将文件查找请求传送给上级父结点.

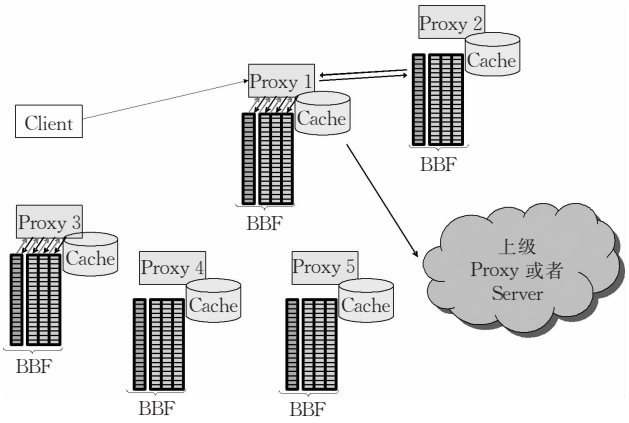


图 6　请求消息传输

在这个安全敏感的文件系统中，为了有效管理文件的查询，系统设计时首先应确定不同文件类型的查询失效安全影响，按照不同文件对系统安全的影响，分为以下 4 个等级，简单估计对系统安全后果的影响为

(1)关键系统补丁文件：如果没有及时打上此补丁，可能会照成系统崩溃或者严重影响网络中其他的主机运行(后果损失：500)；

(2)病毒库更新文件：感染病毒，机器运行速度变慢，或者删除文件，造成数据丢失(后果损失：100)；

(3)系统运行文件：直接影响操作系统运行的文件(后果损失：20)；

(4)普通用户文件：普通的用户文件(后果损失：10).

假设在某段时间内，文件系统中缓存的关键系统补丁文件有 1000 个，病毒库更新文件有 2000 个，系统运行文件 2000 个，普通用户文件 5000 个，共 10000 个文件. 代理之间表示文件目录的 BBF 大小为 $m=131072\text{bit}=16384\text{Byte}$. 下面给出就此例子的算法设计.

1. 根据文件的属性，将文件分为系统补丁文件、病毒库更新文件、系统运行文件和普通用户文件四类，完成 BBF 数据结构参数，就上面的例子，其表示为 $BBF=\{\{1000,2000,2000,5000\},\{500,100,20,10\},131072,\{k_1,k_2,k_3,k_4\}\}$.

2. 根据 3 小节，建立查询失效引起的安全代价最小目标函数.

3. 利用类目标函数遗传算法求解 $\{r_1,r_2,r_3,r_4\}$，并获得最小安全失效总代价.

4. 利用 $\{r_1,r_2,r_3,r_4\}$ 计算出 $\{k_1,k_2,k_3,k_4\}=\{12,12,8,8\}$，因此分档布鲁姆过滤器为 $BBF=\{\{1000,2000,2000,5000\},\{500,100,20,10\},131072,\{12,12,8,8\}\}$.

那么对于第一类文件(关键系统补丁文件)，在映射和查找时利用 12 个 Hash 函数映射，第二类的文件(病毒库更新文件)利用 12 个 Hash 函数，第三类的文件(系统运行文件)用 8 个 Hash 函数映射，第四类的文件(用户文件)用 8 个 Hash 函数映射，这样分配 Hash 函数个数可以区分对待不同安全等级的文件查找.

为了进一步比较验证算法性能，进行系统查询仿真实验. 为了简化实验过程，直接采用 32bit 整数作为文件的名称，数据集合的元素是由计算机随机产生的 32bit 的无符号整数，元素范围为 $\{0,2^{32}-1\}$，系统中使用 H_3 Hash 函数作为实现时的 Hash 函数，H_3 函数是 Carter 和 Wegman 定义的一类通用 Hash 函数(universal Hash)[22]，具有很强的散列性，是一种常见的布鲁姆过滤器实现函数[23-24]. 实验中，随机产生 32×32 的 H_3 函数 0，1 转换矩阵，每个转换矩阵确定一个 Hash 函数.

首先将系统中 {1000，2000，2000，5000} 4 类文件共 10000 个文件的集合用布鲁姆过滤器表示，这是元素的插入过程，分别使用 12，12，8，8 个 Hash 函数分别完成元素的插入，得到文件信息的摘要目录.

为了获得系统一段时间内总共产生的安全失效总代价，我们需要随机模拟一段时间内的用户查询过程，获得系统这段时间造成的总体安全失效代价，并统计用于查询的时间. 用户进行查询时需先指定查询类别，如：关键系统补丁文件、病毒库更新文件、系统运行文件和用户文件. 实验中采用 10000 个不在集合中的元素完成布鲁姆过滤器的查询，进行误判数目统计. 具体的过程是如果元素对应的 k 个比特位都为 1，则统计为一次误判，这是因为这 10000 个元素都是不在集合中的元素. 统计误判率为累计误判元素个数和不在集合中元素总数的比值. 我们采取在 SBF 和 BBF 代码中直接加入计时器的方法获得 10000 个元素的查询总时间. 仿真实验机器配置为 Pentium 1.7GHz，512MB SDR，Windows XP.

对于上述实验过程的每个实验参数的组合，随机产生 1000 次数据集合，完成 1000 次实验，实验结果取 1000 次的平均值.

表 2 中第 1 列是用户查询的各类文件个数分布，2，3 列是查询的安全失效代价，4，5 列是查询的计算时间，单位 ms，从表 2 中发现：

(1) 当用户查询的文件分布是[1000，2000，2000，5000]时，SBF 的安全代价为 2064.4，BBF 算法的安全代价仅为 SBF 的 27%，BBF 算法的安全失效代价远小于 SBF 算法.

表 2 文件系统的安全失效总代价和查询时间仿真结果($m=131072$bit)

查询时各类文件数目				SBF 代价	BBF 代价	SBF 查询时间/ms	BBF 查询时间/ms
关键系统补丁	病毒库更新	系统运行	用户				
1000	2000	2000	5000	2064.4	561.0	815.1	813.4
2000	2000	2000	4000	3447.8	660.2	815.8	816.2
2500	2500	2500	2500	4075.7	736.6	816.5	809.9
5000	2000	2000	1000	7256.4	949.1	812.2	810.5
10000	0	0	0	13228.5	1362.5	815.1	813.7
0	10000	0	0	2613.4	264.6	813.3	810.8

(2) 当某段时间内查询关键系统补丁文件很多,个数为 10000 时,如第 5 行所示,BBF 算法的代价仅为 SBF 的 10.2%,显著减少了系统的损失,提高了系统的整体安全性.这是因为 BBF 算法为关键系统补丁文件分配了更多的 Hash 函数,降低了此类文件的查询误判率,而此类文件对系统安全最为重要,因此必然提高安全性能.

(3) 查询时间比较中,发现 BBF 算法的查询时间和 SBF 算法基本相当,验证 5.1 小节的理论结论.

(4) 在 Pentium 1.7GHz 计算机上,对于每个元素查询,需要时间不超过 0.1ms,这在实际中是可以接受的.

7 结 论

本文提出了一种新的代价敏感分档布鲁姆过滤器查询算法.该算法保持了布鲁姆过滤器算法节约空间的特点,并扩展到考虑查询代价.新算法区分对待集合中的元素,将元素根据查询失效代价分为不同的档次,为每档分配不同个数的 Hash 映射函数,完成集合表示和查询.算法首先考查每档集合的查询失效率和算法设计的关系,建立由 $r-pair(r_i, r_j)$ 表示的每档最低假阳性率组成的集合最低查询失效总代价目标函数,然后通过类目标函数梯度遗传算法优化分档布鲁姆过滤器的设计,以确定每档应分配的 Hash 函数个数,使算法的查询代价最少.

本文的创新点在于区别对待集合中待查询的元素,并在布鲁姆过滤器算法中考虑查询代价,优化其设计.理论分析和实验表明,新算法使集合查询失效总代价降低至少 40%,而算法的计算时间并不增加.查询算法对分布式系统、数据库有用,分档布鲁姆过滤器算法在分布式计算、计算机网络资源定位、数据库的交互查询、P2P 网络资源交互、传感器网络信息交换、计算机网络监测、计算机缓存系统设计、流数据处理以及生物计算等产生大量数据、需要区分对待进行交互查询的应用领域,有重要的应用前景.

参 考 文 献

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

[2] Mullin J K. Optimal semijoins for distributed database systems. IEEE Transactions on Software Engineering, 1990, 16(5): 558-560

[3] McIlroy M. Development of a spelling list. IEEE Transactions on Communications, 1982, 30(1): 91-99

[4] Druschel P, Rowstron A. PAST, a large-scale, persistent peer-to-peer storage utility//Proceedings of the 8th Workshop on Hot Topics in Operations Systems. Elmau/Oberbayern, Germany, 2001. Washington, DC, USA, 2001: 65-70

[5] Stoica I, Morris R, Karger D et al. Chord: A scalable peer-to-peer lookup service for Internet applications//Proceedings of the ACM SIGCOMM. San Francisco, USA, 2001: 149-160

[6] Ratnasamy S, Francis P, Handley M et al. A scalable content-addressable network//Proceedings of the ACM SIGCOMM. San Francisco, 2001: 161-172

[7] Rhea S C, Kubiatowicz J. Probabilistic location and routing//Proceedings of the INFOCOM2002. New York, 2002. Washington, DC, USA, 2002: 1248-1257

[8] Whitaker A, Wetherall D. Forwarding without loops in Icarus//Proceedings of the Open Architectures and Network Programming. New York, USA, 2002: 63-75

[9] Wu-Chan F, Shin K G, Kandlur D D et al. The BLUE active queue management algorithms. IEEE/ACM Transactions on Networking, 2002, 10(4): 513-528

[10] Estan C, Varghese G. New directions in trace measurement and accounting//Proceedings of the ACM SIGCOMM. Pittsburgh, USA, 2002: 323-336.

[11] Fan L, Cao P, Almeida J et al. Summary cache: A scalable wide-area Web cache sharing protocol. IEEE/ACM Trans on Networking, 2000, 8(3): 281-293

[12] Mitzenmacher M. Compressed bloom filters. IEEE/ACM Transactions on Networking, 2002, 10(5): 604-612

[13] Saar C, Yossi M. Spectral bloom filters//Proceedings of the ACM SIGMOD international Conference on Management of Data. San Diego, California, 2003: 241-252

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

[15] Goldberg D. Genetic Algorithms in Search, Optimization, and Machine Learning (1st Edition). Addison-Wesley Longman, 1989

[16] He Xin-Gui, Liang Jiu-Zhen. Genetic algorithms using gradients of object functions. Journal of Software, 2001, 12(7): 981-986(in Chinese)
(何新贵，梁久祯. 利用目标函数梯度的遗传算法. 软件学报, 2001, 12(7): 981-986)

[17] Chankhunthod A, Danzig P, Neerdaels C. A hierarchical Internet object cache//Proceedings of the USENIX 1996 Annual Technical Conference. San Diego, California, 1996: 153-163

[18] Rousskov A, Wessels D. Cache digests. Computer Networks and ISDN Systems Archive, 1998, 30(22-23): 2155-2168

[19] Jiang Cai-Ping, Li Zi-Mu, Yang Feng-Jie. Web caching system performance analysis based on concentrated management. Mini-Micro Systems, 2004, 25(8): 1428-1431(in Chinese)
(姜彩萍，李子木，杨风杰. 集中管理 web 缓存系统及性能. 小型微型计算机系统, 2004, 25(8): 1428-1431)

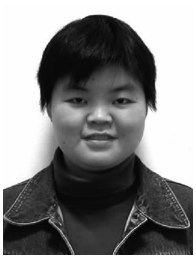
[20] Li X, Xiaodong Zh, Artur A et al. Building a large and efficient hybrid peer-to-peer Internet caching system. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(6): 754-769

[21] Rodriguez P, Spanner C, Biersack E. W. Analysis of Web caching architectures: Hierarchical and distributed caching. IEEE/ACM Transactions on Networking, 2001, 9(4): 404-418

[22] Carter L, Wegman M. Universal classes of hash functions. Computer and System Sciences, 1979, 18(2): 143-154

[23] Ramakrishna MV. Practical performance of Bloom filter and parallel free-text searching. Communications of ACM, 1989, 32(10): 1237-1239

[24] Kaya I, Kocak T. A low power lookup technique for multi-hashing network applications//Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures. Karlsruhe, Germany, 2006. Washington, DC, USA, 2006: 179-184



XIE Kun, born in 1978, Ph. D. candidate. Her research interests include dependability system and network, network measurement and distributed computing.

ZHANG Da-Fang, born in 1959, Ph. D, professor, Ph. D. supervisor. His main research interests include dependability system and network, network measurement, software fault-tolerance and software testing.

XIE Gao-Gang, born in 1974, Ph. D, associate professor. His research interests are next generation network, network measurement and mobile computing.

WEN Ji-Gang, born in 1978, Ph. D. candidate. His main research interests include peer-to-peer computing, streaming and pervasive computing.

MIN Ying-Hua, born in 1935, IEEE Fellow, professor, Ph. D. supervisor. His main research interests include dependable computing, computer control, software reliability, IC design and testing.

Background

This work is supported by the National Natural Science Foundation of China under grant Nos. 60473031, 60273070, 60403031 and the National High-Tech Research and Development Plan of China under grant No. 2005AA121560.

A Bloom filter is an excellent data structure that can succinctly represent a data set in order to support membership queries, and filter out effectively any element that does not belong to the set. It is widely used in databases, networks and distributed systems and it has great potential for distributed applications where systems need to share information about available data. Although Bloom filters are now starting to receive significant attentions from the algorithmic community and there have been a number of recent results, there may well be further improvements to be found. This paper presents a novel basket Bloom filter (BBF). The BBF differentiates elements in a data set depending on their query invalidation cost, by clustering elements into different baskets. Its trait is to treat element differently in the Bloom filter algorithm and the discusses the query invalidation cost of BF, which is the main contribution of this paper.