一种基于并行 Bloom Filter 的高速 URL 查找算法

周舟[1]　付文亮[2]　嵩天[2]　刘庆云[1]

（1. 中国科学院 信息工程研究所 信息内容安全技术国家工程实验室，北京 100093；
2. 北京理工大学 计算机学院， 北京 100081）

摘　要：URL 查找是众多网络系统中重要的组成部分，如 URL 过滤系统、Web 缓存等.随着互联网的迅速发展，URL 查找面临的主要挑战是实现大规模 URL 集合下的高速查找，同时保证低存储和低功耗.本文提出了一种基于并行 Bloom Filter 的 URL 查找算法，CaBF.该算法高度并行化，提供大规模 URL 集合下的高速最长前缀匹配，并很好地适应集合中不同数量的 URL 组件.理论分析和真实网络数据集上的实验表明，该算法相比现有算法可以降低假阳性概率达一个数量级（或者在满足相同假阳性概率的前提下降低存储和硬件逻辑资源消耗）.此外，该方法的体系结构很容易映射到 FPGA 等硬件器件上，提供每秒超过 150M 次的 URL 查找速度.

关键词：URL 查找；布鲁姆过滤器；最长前缀匹配；现场可编程门阵列

中图分类号 ：TP393　　**文献标识码** ：A

Fast URL Lookup Using Parallel Bloom Filters

Abstract：URL lookup is fundamental to numerous networking systems, including URL filters, web caches, etc. With the explosive development of the Internet, the main challenge in implementing URL lookup operation is to achieve fast lookup speed and accommodate large URL sets while keeping memory and power consumption low. This paper presents a new URL lookup scheme based on parallel Bloom Filters. It can adapt to set cardinality and perform fast longest prefix matching (LPM) with large URL sets in a highly parallel fashion. The theoretical analysis and experiments on real-life data traces show that the proposed approach leads to reduced false positive probability for up to an order of magnitude (or reduced memory and hardware logic resources with the same false positive probability guaranteed) compared with existing methods. Moreover, the architecture can be easily mapped to the state-of-the-art FPGAs with moderate resource consumption to provide over 150M lookups per second.

Key words：URL lookup; Bloom Filter; longest prefix matching; FPGA

1 引言

近年来，URL查找技术受到广泛关注.它在很多网络应用中扮演着必不可少的角色，如搜索引擎[1]、Web缓存[2]、七层交换[3]、URL过滤[4].其核心操作是在规模庞大的URL集合中查找一个URL（或者该URL的最长前缀）.

通常，URL的长度远大于IP地址，并且长度不固定，这使得URL查找与传统IP地址的最长前缀匹配不同.同时，互联网的迅速发展带来的两个趋势进一步增加了URL查找的复杂性.首先，网站的数量急剧增加，从2001年9月的3400万到2013年9月的7.39亿（其中活跃的数量超过1.89亿）[5]，这会潜在增加URL集合的规模.其次，网络线速在不断增加，OC-192（10 Gbps）甚至更大带宽已经普遍采用，这要求URL查找必须在严格的时间约束下完成.因此，URL查找面临的主要挑战是实现大规模URL集合下的高速查找，同时兼顾存储和功耗.

实现高速URL查找的一种方案是使用TCAM（ternary content addressable memory）[3]作为基本硬件单元.虽然TCAM能够提供恒定的查找时间，但它有明显的缺点，如过高的功耗和成本.其他方案借助trie结构或者哈希表[6,7]，它们的主要问题是存储开销过大，并且查找性能受制于URL的长度和哈希碰撞.

除了上述方法，一种更有效的数据结构Bloom Filter可以用来解决URL查找问题.Bloom Filter具有高空间效率和高速成员查询的特点，同时它的假阳性概率足够小.Dharmapurikar等人最早引入Bloom Filter进

收稿日期：2014-03-16

基金项目：国家高技术研究发展计划（“863”计划）基金资助项目（2011AA010703）；中国科学院战略性先导科技专项基金资助项目（XDA06030200）；国家自然科学基金资助项目（61303260）

行IP地址的最长前缀匹配（longest prefix matching, LPM）[8].该方法的通用性使得它很容易应用到URL查找问题.它结构简单，能够提供良好的平均性能.然而，由于它将URL按照长度分组并将每个分组关联一个Bloom Filter，它没有充分考虑URL的语法特性.事实上，URL具有明显的层次化结构，由若干分割组件（component）组成，利用这些特性能够显著提高查找性能.

不同于传统基于URL长度分组的Bloom Filter方法，我们先将URL分成组件，然后将各个组件按照它们的原始顺序依次存储在不同的Bloom Filter中.这个改进有如下一些优点，对查找性能的提升有至关重要的作用.

首先，由于不同的Bloom Filter拥有不同的元素个数（cardinality），当总存储一定的情况下，根据集合元素个数优化调整各个Bloom Filter的存储和哈希函数个数可以降低假阳性错误.通常来说，在实际网络环境下，存储在每个Bloom Filter中的元素逐个迅速递减，呈现高度倾斜的分布规律.基于组件个数的优化调整比基于整个URL的方法更加灵活，粒度更细.

其次，需要解决的问题可以用多属性元素的成员查询建模.元素的每个属性是一个URL组件.通过引入同一URL多个组件相关性的检验机制可以降低假阳性概率.

再次，在多个Bloom Filter中并行哈希较短的组件比在单个Bloom Filter中哈希整个URL速度更快.尤其当每个Bloom Filter使用的哈希函数个数很多时，哈希计算的性能可以得到明显提升.不仅如此，借助于更少的哈希函数个数能够进一步提升哈希计算，节省硬件逻辑资源，同时达到传统基于URL长度分组的Bloom Filter方法相同的假阳性概率.

针对上述方面，本文提出了一种基于并行Bloom Filter的URL查找算法CaBF（cardinality adaptive parallel Bloom Filter）.该算法高度并行化，提供大规模URL集合下的高速最长前缀匹配，并很好地适应集合中不同数量的URL组件.相比传统的Bloom Filter结构，本文算法可以降低假阳性概率达一个数量级（或者在满足相同假阳性概率的前提下降低存储和硬件逻辑资源消耗）.其体系结构很容易映射到FPGA等硬件器件上.整个设计目标是为URL过滤、Web缓存等网络系统提供每秒超过150M次的最长前缀匹配速度.

2 相关工作

2.1 Bloom Filter 理论

Bloom Filter是一种简单，空间效率很高的随机数据结构.它用一个长度为m的位向量简洁地表示一个大小为n的集合S，并能判断一个元素是否在集合S中.位向量的所有位首先初始化为0.Bloom Filter使用k个相互独立的哈希函数$h_1,h_2,\ldots,h_k$将集合中的每个元素散列到$\{1,2,\ldots,m\}$的范围.对于给定的元素$x\in S$，位置为$h_i(x)$的比特位都置为1，其中$1\le i\le k$.在判断元素y是否属于集合S时，计算$h_i(y)$，其中$1\le i\le k$.如果所有k个比特位都是1，则认为$y\in S$，否则，$y\notin S$.该判定不存在假阴性错误，但是存在假阳性错误，即不可能把属于集合S的元素误判成不属于集合S，但可能把不属于集合S的元素误判为S的元素.

Bloom Filter的假阳性概率f是m，n和k的函数，表示为

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k\approx\left(1-e^{-\frac{kn}{m}}\right)^k \tag{1}$$

给定m，n，当$k=(m/n)\ln 2$时，假阳性概率达到最小值

$$f=\left(\frac{1}{2}\right)^k\approx(0.6185)^{\frac{m}{n}} \tag{2}$$

哈希函数的个数k与处理开销密切相关.每次成员查询需要k次哈希计算和k次访存.注意到，为了最小化假阳性概率，需要设置较大的k值，这样会带来很多额外开销.

经过若干代数推导可以发现Bloom Filter的两个重要性质：（1）线性性质：对于给定的f，m和n之间存在线性关系；（2）指数反比性质：对于给定的n，m和f之间存在指数反比关系[9].

Bloom Filter广泛应用于网络系统中，如Web系统[10]、P2P系统[11]、路由转发[12]和流量测量[13,14].为了解决标准Bloom Filter在删除、计数和多集合等方面的局限性，研究者们提出了许多Bloom Filter的变种[15].

本文最相关的工作是多维动态Bloom Filter[16]和文献[17]中的Bloom Filter结构.前者缺乏检测同一元素多个属性之间依赖关系的机制，这样会增加假阳性概率.后者解决了这一问题.然而，这两种方法不对同一元素的不同属性加以区分，而是为表示每个属性的Bloom Filter进行相同配置，因而，它们很难处理多个元素拥有不同属性个数，并且同组属性的元素个数高度变化的情况.

2.2 传统基于 Bloom Filter 的 URL 查找算法

当前，实现高速URL最长前缀查找的一个行之有效的方法是使用Bloom Filter.虽然其基本结构最早是应用于IP地址查找中[8]，该结构的通用性使得它很容易应用到URL查找问题.在本文中，我们将这个结构称之为BBF，其基本结构如图1所示.

首先，URL按照它们的长度被分为不同组.然后，每组关联一个Bloom Filter用来存储组内的所有URL.在查找过程中，对于一个包含p个组件的URL，BBF

在前p个Bloom Filter中并行查询.第一个Bloom Filter使用第一个组件作为键值，第二个Bloom Filter组合前两个组件作为键值，依次类推.部分Bloom Filter可能会返回匹配.由于Bloom Filter存在假阳性错误，需要通过额外的前缀表进一步验证这个URL前缀是否真正存在.由于目标是寻找最长前缀，Bloom Filter匹配的最长前缀首先被验证.如果这个前缀真实存在于前缀表中，我们可以获取相应的附加信息（控制、转发策略等）.否则，验证Bloom Filter匹配的次长前缀.显然，BBF的假阳性概率需要足够低，以减少前缀表查找的代价

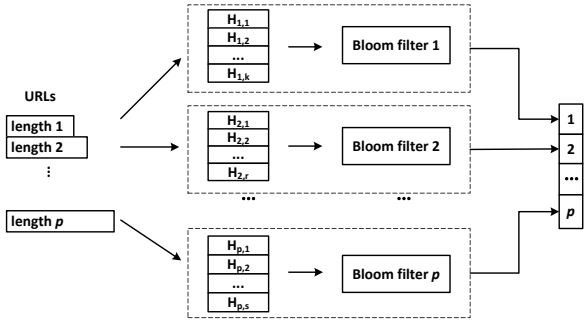


图 1 传统基于 Bloom Filter 的 URL 查找

下面，我们分析BBF的假阳性概率.

假设BBF有p个并行Bloom Filter，每个Bloom Filter包含m_i比特、n_i个URL和k_i个哈希函数.用f_i表示第i个Bloom Filter的假阳性概率.

根据Bloom Filter理论，k_i的最优值为$(m_i/n_i)\ln 2$，因此，f_i表示为

$$f_i = \left(\frac{1}{2}\right)^{k_i} = \left(\frac{1}{2}\right)^{\frac{m_i}{n_i}\ln 2} \qquad \forall i \in [1,2,\cdots,p] \quad (3)$$

对于最长前缀匹配，BBF的假阳性概率为

$$f_{BBF} = 1 - \prod_{i=1}^{p}(1 - f_i) \quad (4)$$

进一步，

$$f_{BBF} \le \sum_{i-1}^{p} f_i = \sum_{i=1}^{p}\left(\frac{1}{2}\right)^{k_i} \quad (5)$$

虽然BBF的基本结构简单合理，但是由于在相应的Bloom Filter中存储的是URL的集合而不是组件的集合，它没有充分考虑URL的语法特性.最近提出的BBF的改进结构：分布式负载平衡Bloom Filter[12]同样有这个问题.经过证明，这两种Bloom Filter结构拥有同样的假阳性概率和存储消耗.

3 基于并行 Bloom Filter 的 URL 查找算法 CaBF

3.1 CaBF 结构

CaBF的基本思想是，使用能够适应集合元素个数的多个不同大小的并行Bloom Filter表示URL的各个组件，同时引入一个附加机制检验同一URL多个组件之间的相关性.

CaBF的基本结构如图2所示.不同于将URL根据长度分组存储在相应Bloom Filter中的方法，CaBF首先将URL分成组件并按照各个组件的原始顺序进行分组.然后，每组关联一个Bloom Filter.这意味着，所有原始URL的第i个组件存储在第i个Bloom Filter中.CaBF允许一个包含p个组件的URL前缀在前p个Bloom Filter中并行查找.

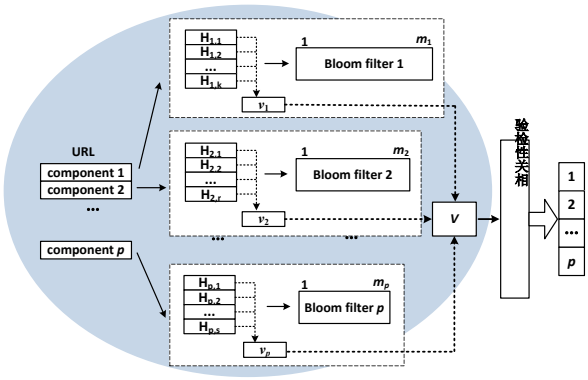


图 2 CaBF 结构

用上述的并行Bloom Filter查询不同URL组件成员是一种十分有效的方法，但是，单独利用这个方法还无法判断同一URL多个组件之间的相关性.忽略相关性会对查询准确性造成影响.因此，CaBF引入一个额外的相关性检验机制解决这一问题.该机制考查URL的验证值是否真实存在.该检验值由每个组件关联的Bloom Filter中所有哈希值来产生.

CaBF 的检验机制采用基于加权值的方式设计.假设一个待查询的 URL 前缀包含 p 个组件，第 i 个 Bloom Filter 使用 k_i 个哈希函数.第 i 个组件的分值为 $v_i = \sum_{j=1}^{k_i} \frac{H_{i,j}}{2^j}$.进一步，这个URL前缀的检验值$V$为所有组件分值的总和，即$V = \sum_{i=1}^{p} v_i$.

选择上述方法主要有两方面原因：（1）软硬件容易实现，只需要简单的移位和加法逻辑；（2）由于给顺序产生的哈希值指派不同的权重，最终产生的检验值可以较好的表示先后关系，区分度较高.

在实际操作中，包括哈希表和 Bloom Filter 在内的任何数据结构都可以用来存储集合的检验值，并执行相关性检验.注意到，对于某个待查找的 URL 前缀，只有当所有组件被并行 Bloom Filter 报告匹配时，才跳转到相关性检验步骤.如果这个检验值真实存在，则报告检验结果为真，否则为假.

经过并行 Bloom Filter 和相关性检验后，仍不可避免假阳性错误的存在.某些网络应用，如 Web 缓存、URL 过滤和搜索引擎等，允许存在一定的假阳性，到此可以返回匹配结果.对于需要精确匹配最长前缀的情况，要通过额外的前缀表进一步验证这个 URL 前缀是否真正存在.由于目标是寻找最长前缀，相关性检验结果为真的最长前缀首先被验证.如果这个前缀真实存在于前缀表中，我们可以获取相应的附加信息（控制、转发策略等）.否则，验证相关性检验结果为真的次长前缀.

在实际网络环境中，每个 Bloom Filter 中存储的组件个数变化很大，第一个 Bloom Filter 表示的组件个数远大于其他 Bloom Filter，然后逐个迅速递减，呈现高度倾斜的分布规律.如果将全部存储空间平均分配给每个 Bloom Filter，则会造成前几个 Bloom Filter 不能满足需求，而其他的 Bloom Filter 超过所需的情况，这样就大大降低了存储使用的有效性.在这种情况下，假阳性概率没有达到最优.

CaBF 的一个先进特性是能够优化调整每个 Bloom Filter 的存储空间.考虑到 Bloom Filter 的线性性质和指数反比性质，分配给每个 Bloom Filter 的存储空间 m_i 与它的集合元素个数 n_i 成正比，同时调整哈希函数个数.具体来说，$m_i/n_i=m/n$，其中 $\sum m_i = m$，$\sum n_i = n$.

通过分析真实网络数据集上的统计数据[18,19]，我们发现 URL 的组件个数明显集中于较少的范围.平均 99.8%的 URL 的组件个数不超过 8 个.这说明减少并行 Bloom Filter 的个数用于表示较短的但数量庞大的 URL 是一个行之有效的方法.对于 URL 长度过长的情况，可以利用剩余的一个 Bloom Filter 处理.假设并行 Bloom Filter 的数量为 p 个，需要找到一个包含 $p+k$ 个组件的 URL 的最长匹配前缀.我们将这个 URL 的前 p-1 个组件并行地在前 p-1 个 Bloom Filter 中查询.对于这个 URL 的剩余组件，分别将它的第 p 个组件、第 p 和第 p+1 个组件的组合……第 p 到第 $p+k$ 个组件的组合在最后一个 Bloom Filter 中查询.

值得注意的是，在总存储空间一定的情况下，CaBF 极大降低了假阳性概率，同时只消耗了少量的存储空间用于实现相关性检验部分.

3.2 假阳性概率分析

当集合 S 的 CaBF 构建完成后，对于元素 y，需要判断 y 的最长前缀是否属于集合 S.假设 CaBF 有 p 个并行 Bloom Filter，每个 Bloom Filter 的假阳性概率为 f_{BFi}，包含 m_i 比特、n_i 个 URL 和 k_i 个哈希函数.显然，当 $k_i=(m_i/n_i)\ln 2$ 时，f_{BFi} 取最小值 $(1/2)^{ki}$.由于引入了相关性检验过程，每次前缀匹配的假阳性概率 f_i 为

$$f_i = \prod_{j=1}^{i} f_{BF_j} . f_{V_i} \qquad \forall i \in [1,2,\cdots,p] \tag{6}$$

其中，f_{Vi} 是包含 i 个组件前缀的检验值的冲突概率.

每个 Bloom Filter 产生的分值 v_i 的数学期望为

$$E(v_i) = \frac{1+m_i}{2}\left(1-\frac{1}{2^{k_i}}\right) \tag{7}$$

方差为

$$Var(v_i) = \frac{(m_i-1)^2}{12}\left(1-\frac{1}{2^{k_i}}\right)^2 \tag{8}$$

其中，k_i 取最优值 $(m_i/n_i)\ln 2$.因此，前 i 个分值的总和 V_i 服从正态分布

$$V_i \sim N\left\{\sum_{j=1}^{i}\frac{1+m_j}{2}\left(1-\frac{1}{2^{k_j}}\right), \sum_{j=1}^{i}\frac{(m_j-1)^2}{12}\left(1-\frac{1}{2^{k_j}}\right)^2\right\} \tag{9}$$

给定 n 个 URL，以及存储它们 V_i 的独立数据结构，f_{Vi} 的上界为

$$f_{V_i} \le 2\Phi\left(\frac{n}{2\sigma}\right)-1 = 2\Phi\left(\frac{n}{2\sqrt{\sum_{j=1}^{i}\frac{(m_j-1)^2}{12}\left(1-\frac{1}{2^{k_j}}\right)^2}}\right)-1 \tag{10}$$

其中，Φ(*)表示标准正态分布.对于最长前缀匹配，CaBF 的假阳性概率为

$$f_{CaBF} \le 1-\prod_{i=1}^{p}(1-f_i) \tag{11}$$

进一步，

$$f_{CaBF} \le \sum_{i=1}^{p} f_i = \sum_{i=1}^{p}\left(\frac{1}{2}\right)^{\sum_{j=1}^{i}k_j} f_{V_i} \tag{12}$$

由于 CaBF 分配给每个 Bloom Filter 的存储空间与集合的元素个数成正比，因此，所有 Bloom Filter 的 k_j 相等，为 $(m/n)\ln 2$，其中 $\sum_{i=1}^{p} m_i = m$，

$\sum_{i=1}^{p} n_i = n$.

对比公式（12）和公式（5），当 CaBF 和 BBF 的存储空间和集合的元素个数相同时，CaBF 的假阳性概率的上界远小于 BBF.这归功于 CaBF 适应集合元素个数的特性和附加的相关性检验机制.本文第 4 节在真实网络数据集上进一步证明 CaBF 拥有较低的假阳性概率.

4 实验结果与分析

对 CaBF 的性能评价在两个真实网络数据集上进行.我们将 CaBF 算法与其他三种基于 Bloom Filter 的算法进行比较.这三个算法分别为：（1）BBF－，该算法不考虑 URL 前缀的分布规律，平均分配每个过滤器的存储空间；（2）BBF＋，该算法优化配置 BBF 结构，根据 URL 前缀的分布规律调整每个过滤器的存储空间；（3）CaBF－，该算法使用 CaBF 结构，但不能适应集合的元素个数.

上述每个算法都配备了 8 个并行 Bloom Filter.实验结果取 10 次重复实验的平均值.所使用的计算机配置为 Intel Core 2 Quad CPU 2.83 GHz 和 4 GB 内存.

4.1 数据集

第一个数据集是一个 URL 黑名单.它由 URLBlacklist.com[18]提供.实验使用 2013 年 9 月的数据集，其中包含 3,659,916 个 URL.每次实验从集合中随机选择一定数量的 URL 插入到每个基于 Bloom Filter 的结构中，同时插入一个哈希表用于识别匹配的 URL 前缀.经过初始化后，我们对集合中的每个 URL 执行成员查询，同时比较 Bloom Filter 结构中匹配的数量和哈希表中匹配的数量，得出假阳性概率.

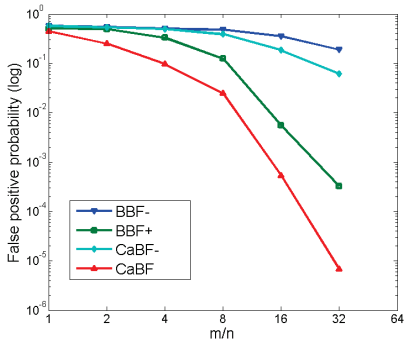
另一个数据集是由搜狗实验室提供的大规模用户查询数据[19]，提取的是 2006 年 8 月一个月的数据.其中，每个子数据集是一天的日志，共有 31 个子数据集，平均包含 714,231 个 URL.对于两个子数据集的任意组合，我们将第一个子数据集插入每个基于 Bloom Filter 的结构中用于初始化.接下来，用第二个子数据集比较 CaBF 和其他 Bloom Filter 结构的性能.

4.2 假阳性概率比较分析

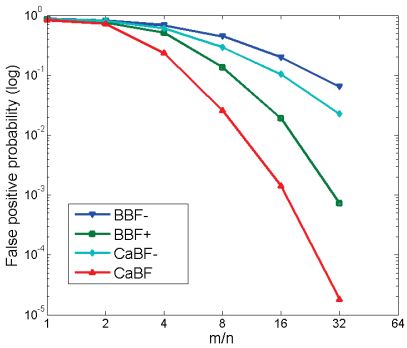
这组实验比较 BBF－、BBF＋、CaBF－和 CaBF 算法的假阳性概率，分别在不同的存储大小 m、集合大小 n 和哈希函数个数 k 的情况下考察.

（1）存储大小影响

图 3 给出了不同存储大小 m 下 BBF－、BBF＋、CaBF－和 CaBF 算法的假阳性概率.URL 黑名单数据集和搜狗数据集中插入 Bloom Filter 的 URL 数量分别为 1M 和 700K.



(a) URL blacklist (n=1M)



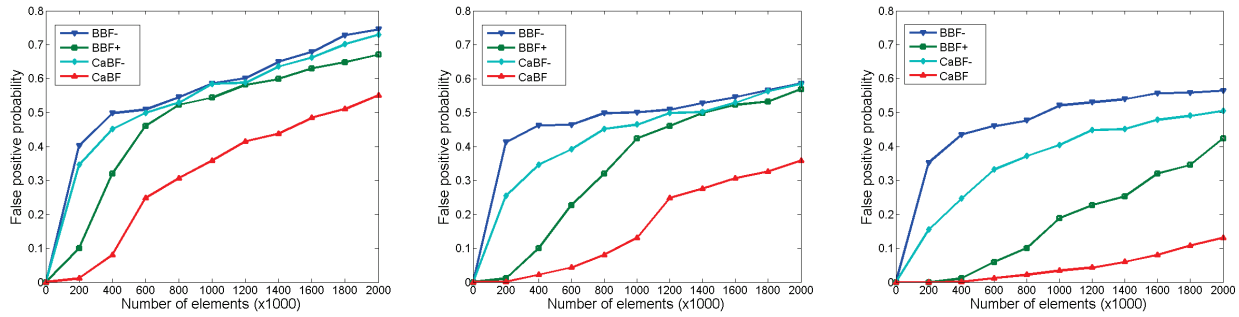
(b) Sogou (n=700K)

图 3 不同存储大小下各个 Bloom Filter 算法的结果对比

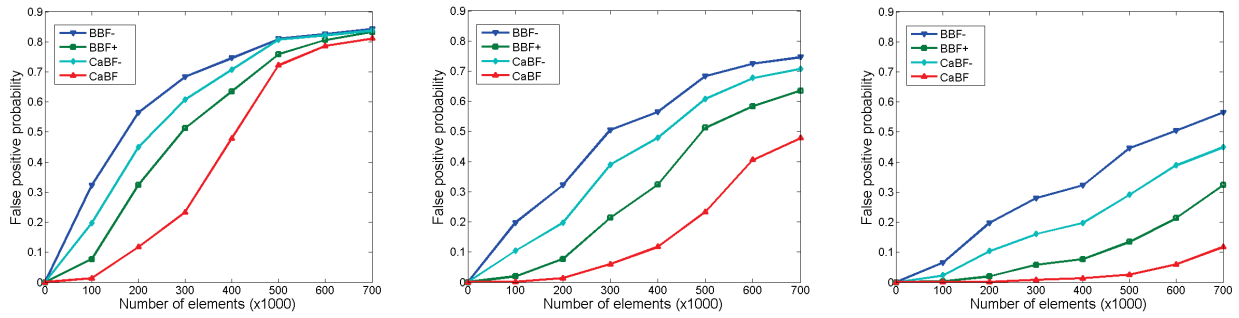
从图 3 可以看出，由于 BBF－和 CaBF－不考虑 URL 前缀的分布，它们比 BBF＋和 CaBF 产生更多的假阳性错误，同时 CaBF－的性能优于 BBF－.对比 BBF＋算法，CaBF 算法在两个数据集的测试中能够降低假阳性概率从 2 倍以上到一个数量级.这是由于 CaBF 引入了同一 URL 多个组件相关性的检验机制，能够有效提高查询准确性.当给定的存储空间较小，即 m/n 的比率较小时，CaBF 算法的性能增益不是太明显，因为并行过滤器部分产生了太多假阳性错误，严重影响了相关性检验部分的效能.然而，随着存储空间的增加，当存储大小开始超过集合元素个数 4 倍以上时，CaBF 算法的性能增益变得越来越明显.

（2）集合大小影响

图 4 给出了不同集合大小 n 下 BBF－、BBF＋、CaBF－和 CaBF 算法的假阳性概率.每次实验在给定固定大小的存储空间的情况下，变化集合大小 n.对于 URL 黑名单数据集，n 从 200K 变化到 2M；对于搜狗数据集，n 从 100K 变化到 700K.



(a) URL blacklist (*m*=2Mb)　(b) URL blacklist (*m*=4Mb)　(c) URL blacklist (*m*=8Mb)



(d) Sogou (*m*=1Mb)　(e) Sogou (*m*=2Mb)　(f) Sogou (*m*=4Mb)

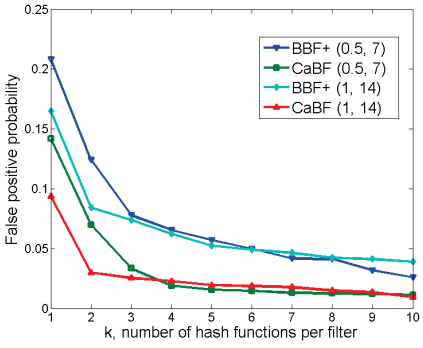
图 4 不同集合大小下各个 Bloom Filter 算法的结果对比

所有实验结果都支持同一个结论：将一定数量的 URL 插入相应的 Bloom Filter 结构中，CaBF 算法在四种算法中都能够获得最低的假阳性概率.具体来说，针对 URL 黑名单数据集和搜狗数据集，当 n 分别小于 400K 和 200K 时，相比 BBF＋算法，CaBF 算法的性能增益能够达到一个数量级；相比 BBF－和 CaBF－算法，CaBF 算法能够提升性能达到两个数量级.当 n 进一步增加时，CaBF 算法的性能依然超越其他算法，提升最大能够达到一个数量级.总体来说，CaBF 算法的假阳性概率增长趋势低于其他算法，特别是在给定更多存储空间的情况下.

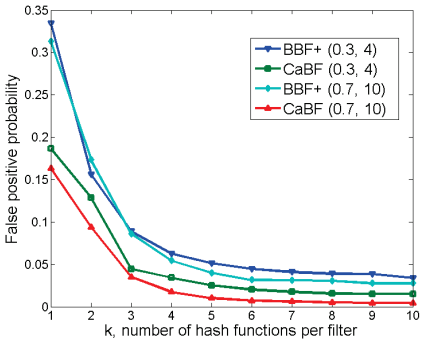
（3）哈希函数个数影响

图 5 给出了每个过滤器不同哈希函数个数 k 下 BBF＋和 CaBF 算法的假阳性概率.k 的最优值设置为 10，可以获得最低的假阳性概率.固定 m 和 n，进一步考察哈希函数个数低于最优值的情况下的性能.

虽然这两种算法在哈希函数个数低于最优值时性能会逐步下降，但是 CaBF 算法持续保持比 BBF＋好得多的性能.一个重要的发现是，利用少得多的哈希函数（实验中为 2-4 个），优化配置下 CaBF 能够获得与 BBF＋相当的性能.减少哈希函数个数的好处是能够降低哈希计算的代价.不仅如此，相应的硬件开销也同样降低.



(a) URL blacklist



(b) Sogou

图 5 不同哈希函数个数下各个 Bloom Filter 算法的结果对比

(*i*,*j*) 表示 *i*=*n* (M) ，*j*=*m* (Mb)

4.3 原型系统和结果

随着网络技术的不断发展，OC-192（10 Gbps）乃至更高速链路已普遍采用，使用软件方法无法满足高速网络下 URL 查找的需求，使用专用硬件加速成为必然.由于 FPGA 具备可编程、高并行等特性，因此它非常适合于解决高速网络下实时流量处理问题.

我们在 Xilinx Virtex-5 FX200T 的平台上实现了 CaBF 的硬件原型系统.分配给 CaBF 的片上存储空间为 12 Mb，有 8 个过滤器.存储 URL 集合的片下哈希表映射到一个 72 Mb 300 MHz 的 QDR-II SRAM.

实验结果如表 1 所示.Virtex-5 FX200T 有 16 Mb 的片上 BRAM，即 456 个存储块，每块 36 Kbits.设计的结构使用了 362 个存储块，占可用存储资源的 79%.CaBF 的逻辑资源使用率约为 37%.进一步考察发现，逻辑资源主要用来实现 H_3 哈希函数.整个体系结构需要上百个哈希函数.平均每个哈希逻辑需要耗费约 60 个 slice.本文第 4.2 节证明，优化配置下 CaBF 算法能够用少得多的哈希函数获得与 BBF＋算法相当的性能.根据这个趋势，在满足相同假阳性概率的前提下，CaBF 能够比传统算法降低近 60%的硬件逻辑资源消耗.

表 1　CaBF 结构的资源使用

资源	使用数	可用数	使用率
Slices 数量	11 440	30 720	37%
BRAM 数量	362	456	79%
片上存储 (Mb)	12	16	75%
片下存储 (Mb)	50	72	69%

系统的最大综合频率为 177 MHz.借助双端口 RAM 和多级流水线技术，CaBF 体系结构能够提供每秒超过 150M 次的查找速度.其性能已经与基于 TCAM 的算法相当，同时它的成本和功耗比 TCAM 低很多.利用处理能力更高的 FPGA 器件和更先进的 SRAM 存储器，CaBF 可以进一步提高查找性能.

针对 CaBF 的查询准确性，在真实网络数据集上进行了实验.300K 个 URL 插入到 CaBF 结构以及片下哈希表中.接下来，我们从搜狗数据集中提取 10M 个 URL 进行查找.观察到 17 次假阳性错误，相应的假阳性概率为 1.7×10^{-6}.

4.4 讨论

在实际网络环境中，随着 URL 规模的迅速增长，CaBF 算法需要具备可扩展性.根据 Bloom Filter 的线性性质，在集合大小增加的情况下，要保证查找性能，即保持原有的假阳性概率，需要线性地增加存储空间.对于软件实现方法，CaBF 算法会占用更多的内存，这需要通过增加内存容量来实现.而对于硬件实现方法，CaBF 算法会消耗更多的硬件资源，这需要采用更高级的 FPGA 器件和更大容量的存储器.

另外，当 CaBF 按照某种集合的元素个数分配存储空间时，能够获得很好的性能.然而，如果环境变化，即集合的元素个数分布变化时，之前优化的 CaBF 可能获得次优的性能，为了达到动态适应环境变化的目标，可以引入一种基于 mini-Bloom Filter 的方法[8].此方法下，整个系统的假阳性概率保持不变，灵活性得到极大提升.此外，为了使 CaBF 支持 URL 的添加和删除，需要利用附加计数 Bloom Filter[8].唯一的区别是，CaBF 在更新时还涉及相关性检验部分.

5 结束语

本文提出了一种基于并行 Bloom Filter 的 URL 查找算法.不同于传统基于 URL 长度分组的算法，本文将 URL 分成组件，并将各个组件按照它们的原始顺序依次存储.提出的算法在两个方面提高查找性能：（1）使用能够适应集合元素个数的多个不同存储大小的并行 Bloom Filter 表示 URL 的各个组件；（2）引入一个机制检验同一 URL 多个组件之间的相关性.理论分析和真实网络数据集上的实验表明，该算法相比现有算法可以显著降低假阳性概率和硬件逻辑资源消耗.作为 TCAM 的良好替代方案，其硬件体系结构能够为 URL 过滤系统、Web 缓存等网络系统提供每秒超过 150M 次的最长前缀匹配速度.

参考文献

[1] Broder A Z, et al. Efficient URL caching for world wide web crawling[A]. Proc. of WWW[C]. New York: ACM, 2003. 679-689.

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

[3] Huang N F, et al. A fast URL lookup engine for content-aware multi-gigabit switches[A]. Proc. of AINA[C]. Washington: IEEE Computer Society, 2005, 641-646.

[4] Zhou Z, et al. A high-performance URL lookup engine for URL filtering systems[A]. Proc. of ICC[C]. Washington: IEEE Computer Society, 2010, 1-5

[5] Netcraft[EB/OL]. http://news.netcraft.com, 2013-09-30.

[6] Michel B S, et al. URL forwarding and compression in adaptive web caching[A]. Proc. of INFOCOM[C]. Washington:

IEEE Computer Society, 2000, 670-678.

[7] Prodanoff Z G, et al. Managing routing tables for URL routers in content distribution networks[J]. International Journal of Network Management, 2004, 14(3), 177-192.

[8] Dharmapurikar S, et al. Longest prefix matching using bloom filters[J]. IEEE/ACM Trans. on Networking, 2006, 14(2), 397-409.

[9] Yu H, et al. A memory-efficient hashing by multi-predicate bloom filters for packet classification[A]. Proc. of INFOCOM[C]. Washington: IEEE Computer Society, 2008, 1795-1803.

[10] Chang F, et al. Bigtable: a distributed storage system for structured data[J]. ACM Trans. on Computer Systems, 2008, 26(2), 1–26.

[11] Cai H, et al. Applications of bloom filters in peer-to-peer systems: issues and questions[A]. Proc. of NAS[C]. Washington: IEEE Computer Society, 2008, 97-103.

[12] Song H, et al. IPv6 lookups using distributed and load balanced bloom filters for 100Gbps core router line cards[A]. Proc. of INFOCOM[C]. Washington: IEEE Computer Society, 2009, 2518-2526.

[13] Lu Y, et al. Robust counting via counter braids: an error-resilient network measurement architecture[A]. Proc. of INFOCOM[C]. Washington: IEEE Computer Society, 2009, 522-530.

[14] 王洪波等. 高速网络超连接主机检测中的流抽样算法研究[J]. 电子学报, 2008, 36(4): 809-818.

Wang Hong-bo et al. On flow sampling for identifying super-connection hosts in high speed networks[J]. ACTA Electronica Sinica, 2008, 36(4): 809-818.(in Chinese)

[15] 谢鲲等. 布鲁姆过滤器查询算法[J]. 软件学报, 2009, 20(1): 96-108.

Xie K, et al. Bloom filter query algorithm[J]. Journal of Software, 2009, 20(1): 96-108.(in Chinese)

[16] Guo D, et al. Theory and network applications of dynamic bloom filters[A]. Proc. of INFOCOM[C]. Washington: IEEE Computer Society, 2006. 1-12.

[17] Xiao B, et al. Using parallel bloom filters for multiattribute representation on network services[J]. IEEE Trans. on Parallel and Distributed Systems, 2010, 21(1): 20-32.

[18] URLBlacklist.com[EB/OL]. http://urlblacklist.com, 2013-09-20.

[19] Sogou Labs User query log[EB/OL]. http://www.sogou.com/labs/dl/q.html, 2013-10-14.

作者简介

周舟 男，1983 年生，湖南长沙人，博士，助理研究员. 主要研究方向为网络安全、高性能网络.

E-mail：zhouzhou@iie.ac.cn

付文亮 男，1984 年生，河北邯郸人，博士研究生. 主要研究方向为网络安全、节能网络.

E-mail：fuwenl@bit.edu.cn

嵩天 男，1980 年生，辽宁沈阳人，博士，副教授. 主要研究方向为网络安全、计算机体系结构.

E-mail：songtian@bit.edu.cn

刘庆云（通信作者） 男，1980 年生，河北衡水人，硕士，高级工程师. 主要研究方向为网络安全.

E-mail：liuqingyun@iie.ac.cn