软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(9):2165−2181 [doi: 10.3724/SP.J.1001.2013.04365]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向骨干网流量分析与管理的计数器结构*

张 进, 黄清杉, 赵文栋, 彭来献

(解放军理工大学 通信工程学院,江苏 南京 210007)

通讯作者: 张进, E-mail: boost_zj@163.com

摘 要: 针对高速骨干网流量分析与流量管理对大规模高速统计计数器的需求,提出了一种简单、高效的主动式计数器结构——DALCA,其新颖之处在于对计数器向量进行分层,除第 1 层外,其他各层子计数器向量采用多级哈希表的形式进行组织,获得了良好的空间效率和时间效率.与现有的主动式计数器结构 BRICK 相比, DALCA 的空间效率与其接近,但是时间效率明显优于 BRICK.采用真实的骨干网流量数据,通过仿真实验对 DALCA 的性能进行了评估,结果表明:执行查询操作时,DALCA 的访存带宽需求约为 BRICK 的 1/10;执行更新操作时,DALCA 的访存带宽需求约为 BRICK 的 1/6.

关键词: 骨干网;流量分析;流测量;统计计数器;多级哈希表

中图法分类号: TP393 **文献标识码**: A

中文引用格式: 张进,黄清杉,赵文栋,彭来献.面向骨干网流量分析与管理的计数器结构.软件学报,2013,24(9):2165−2181. http://www.jos.org.cn/1000-9825/4365.htm

英文引用格式: Zhang J, Huang QS, Zhao WD, Peng LX. Statistics counter architecture for backbone network traffic analysis and management. Ruan Jian Xue Bao/Journal of Software, 2013,24(9):2165−2181 (in Chinese). http://www.jos.org.cn/1000-9825/4365.htm

Statistics Counter Architecture for Backbone Network Traffic Analysis and Management

ZHANG Jin, HUANG Qing-Shan, ZHAO Wen-Dong, PENG Lai-Xian

(Institute of Communication Engineering, PLA University of Science and Technology, Nanjing 210007, China)

Corresponding author: ZHANG Jin, E-mail: boost_zj@163.com

Abstract: A simple and efficient active statistics counter architecture named DALCA (direct addressing layered counter array) is presented for high speed network traffic analysis and management. The novelty of DALCA is that the counter vector is split up into multiple layers, and all layers except the first one are organized as multi-level hash tables. This makes DALCA efficient in both from the space and time perspective. DALCA has similar space efficiency and significantly higher time efficiency compared with BRICK; the state-of- the-art active statistics counter architecture. The performance of DALCA is evaluated using real world backbone network traffic traces. Simulation results show that the memory bandwidth demand of DALCA is about 1/10 and 1/6 that of BRICK during query and update operations respectively.

Key words: backbone network; traffic analysis; flow measurement; statistics counter; multi-level Hash table

网络流量分析和流量管理需要统计计数器的支持,随着链路数据率的飞速提升,在高速骨干网中,统计计数器的设计面临着极大的挑战:一方面,所需要的计数器数目巨大;另一方面,所允许的更新和查询时间极短.以骨干网数据流的流量测量为例,已有的研究结果表明,在高速骨干网链路中,并发数据流的数目通常达到百万条数量级[1],为了进行逐流的流量统计,所需的计数器的数目至少应与数据流的数目一致;如果每条数据流需要多个统计项(例如,同时统计各条数据流的总报文数和总字节数,或者同时统计各对主机之间的 TCP 报文数和 UDP

* 基金项目: 江苏省自然科学基金(BK2010103)
收稿时间: 2012-06-05; 修改时间: 2012-10-19; 定稿时间: 2012-12-03

报文数),则所需的计数器数目更为庞大.另一方面,在骨干网链路中,报文的到达时间间隔极短,例如在 OC-768 链路中,假设最短报文长度为 64 字节,则链路满负载时,报文到达间隔最小为 12.8ns.这就需要统计计数器能够在极短的时间内完成更新和查询.通过上述分析可见,面向高速骨干网链路的统计计数器结构必须采用容量大且速度快的存储器来实现.但是,存储器的容量和速度是一对相互矛盾的性能指标,容量大的存储器,如 DRAM,其速度相对较慢;而速度快的存储器,如 SRAM,其容量相对较小.如何实现大规模的高速统计计数器结构,是骨干网流量分析及管理中所面临的一个难点问题.

已有的研究工作主要从两个不同的角度来处理大规模高速计数器结构设计:第 1 类方法是进行近似计数,而非准确计数.这一类工作的典型代表有 SAC[2],SLAC[3],DISCO[4]和 CEDAR[5].近似计数可以减小计数器的位宽,降低总的空间需求,使得整个计数器结构可以用高速存储器实现;第 2 类方法假设计数器仅仅需要支持在线更新,而不需要支持在线查询.这样,通过对计数值进行在线压缩[6,7],或者采用 SRAM+DRAM 的混合存储结构[8],以减小对高速存储器空间的需求.不支持在线查询的计数器称作被动式计数器(passive counter);与其相对,支持在线查询的计数器称作主动式计数器(active counter)[2].考虑到某些应用既需要准确计数,也需要支持在线查询,文献[9]提出了一种准确计数的主动式计数器结构——BRICK.在典型的网络流量参数条件下,与原始的计数器向量相比,BRICK 可以将空间效率提升一倍,且几乎不会产生计数误差.这里,“几乎”的含义是产生计数误差的概率足够小,在实际应用中可以忽略不计.与近似计数的或者被动式的计数器结构相比,准确计数的主动式计数器结构显然更具价值.但是,BRICK 的缺点在于其时间效率较低;此外,BRICK 需要基于 64 位硬件平台进行实现,且需要特定的处理器指令的支持,实现方式不够灵活.

本文提出了一种新的准确计数的主动式计数器结构 DALCA(direct addressing layered counter array). DALCA 对原始的计数器向量进行分层,将其拆分成多层子计数器向量.计数值的高位存放在高层子计数器向量中,低位存放在低层子计数器向量中.除第 1 层子计数器向量外,其他各层子计数器向量采用多级哈希表的形式进行组织.低层的子计数器溢出时,将其后继子计数器在多级哈希表中的块地址存储下来;在需要访问该后继子计数器时,通过读取事先存储的块地址,再经过一次哈希运算,即可直接寻址到该子计数器.DALCA 的名称中,“Direct Addressing”的含义正在于此.与 BRICK 相比,DALCA 访问高层子计数器的方式更为简单直接,在时间效率方面具有明显的优势.同时,与 BRICK 相比,DALCA 采用多级哈希表对高层子计数器向量进行组织,该机制所节省下来的空间抵消了其存储块地址所需花费的空间,因而 DALCA 的空间效率和 BRICK 基本接近.此外,DALCA 无需基于 64 位硬件平台实现,实现方式比 BRICK 更为灵活.采用真实骨干网流量数据,通过仿真实验,将 DALCA 和 BRICK 的性能进行了比较,结果表明,DALCA 的空间效率和 BRICK 接近,但是时间效率明显优于 BRICK.就查询操作而言,DALCA 的访存带宽需求约为 BRICK 的 1/10;就更新操作而言,DALCA 的访存带宽需求约为 BRICK 的 1/6.

本文的主要贡献如下:

(1) 提出了一种新的主动式计数器结构 DALCA,其空间效率和现有的主动式计数器结构 BRICK 接近,但时间效率明显优于 BRICK;此外,DALCA 的实现方式较 BRICK 更为灵活;

(2) 给出了计算多级哈希表中,插入失败的元素个数的概率分布方法.为了对 DALCA 的更新失败概率进行定量分析,其核心步骤是需要获知多级哈希表中插入失败的元素个数的概率分布.但是,现有的研究工作仅仅给出了多级哈希表中插入失败的元素个数的期望值的计算方法[10,11].本文第 3.2 节给出了一种计算插入失败的元素个数的概率分布方法,仿真实验的结果验证了方法的正确性和有效性.该方法可以在其他有关多级哈希表的研究工作中得到应用.

本文第 1 节对相关研究进行总结.第 2 节说明 DALCA 的组成结构,并描述其更新与查询算法.第 3 节给出 DALCA 的参数优化设计方法.第 4 节通过仿真实验对 DALCA 的性能进行分析,并且与现有的主动式计数器 BRICK 进行性能比较.第 5 节讨论 DALCA 实现中的一些问题.第 6 节为本文的结论以及下一步的研究方向.

1 相关研究现状

根据是否存在计数误差,已有的面向骨干网流量分析与管理的大规模高速计数器结构,可分为近似计数和准确计数这两类;根据是否支持在线查询,又可分为被动式(passive)和主动式(active)这两类.已有的研究成果的分类见表 1.

Table 1 Summary of related works

表 1 相关研究工作总结

	近似计数	准确计数
被动式	CB[6,7]	Hierarchical memory[8]
主动式	SAC[2], SLAC[3], DISCO[4], CEDAR[5]	BRICK[9]

Lu 等人提出了一种近似计数的被动式计数器 CB(counter braid)[6,7],其基本方法是,借鉴低密度校验码(LDPC code)的编、解码过程,首先对计数值进行在线压缩,然后通过离线的解压过程,将原始的计数值恢复出来,结果可能存在一定的误差.CB 的优点是空间效率高,每计数器所需空间约为 7bits;其缺点在于无法支持在线的查询,并且存在一定的解码错误概率,导致计数结果并非完全准确.Zhao 等人提出采用 SRAM 和 DRAM 组成的两级存储器结构实现大规模高速计数器[8],每个计数器的低比特位存放在 SRAM 中,而完整的计数值存放在 DRAM 中.SRAM 中的计数值逐包更新,当 SRAM 中的计数值溢出时,将其更新到 DRAM 中.该结构的优点是对 SRAM 空间需求低,每计数器仅仅需要 4 比特~6 比特的 SRAM;其缺点同 CB 一样,也属于被动式计数器结构,无法支持计数值的在线查询.

诸多网络流量分析和网络流量管理任务[12−15]需要在线查询计数结果,也即需要采用主动式计数器.现有的主动式计数器大部分仅仅支持近似计数,例如 SAC[2],SLAC[3],DISCO[4]和 CEDAR[5].上述计数器结构的共同特点是:根据当前计数值的大小,按照一定的概率对计数器进行更新,更新概率的大小和计数值的大小成反比.查询时,根据实际计数结果,对原始计数值进行估计.因此,SAC[2],SLAC[3],DISCO[4]和 CEDAR[5]虽然能够支持计数值的在线查询,但是计数结果存在一定的误差.文献[5]对 SAC,DISCO 和 CEDAR 的性能进行了比较,指出:在消耗同等存储空间的前提下,CEDAR 的计数精度最高;典型参数条件下,当每个计数器所占用的存储空间分别为 9 比特,10 比特,11 比特和 12 比特时,CEDAR 的相对计数误差分别为 10%,7%,5%和 3%[5].近似计数的计数器结构无法应用于网络测量与测试设备等要求准确计数的场合.此外,对于网络流量分析的数据流算法[12−14]而言,若采用近似计数的计数器结构,由于计数误差的引入,难以从理论上保证数据流算法的精度.Hua 等人提出了 BRICK[9],这是已有研究成果中唯一准确计数的主动式计数器结构.BRICK 的基本思想是:对计数器进行分组,对同一组内的计数器进行分层压缩,并采用等级排位(ranking index)的方法对高层子计数器进行寻址.在典型的参数条件下,BRICK 中每计数器所需的存储空间为 10 比特~11 比特[9];而同等空间消耗下,CEDAR 的计数结果存在 5%~7%的相对误差[5].可见,与近似计数的主动式计数器结构相比,准确计数的主动式计数器结构在的存储开销上并无明显提高. BRICK 的缺点在于时间效率较低;此外,在具体实现时,BRICK 要求采用 64 位的硬件平台,并且需要特定的处理器指令的支持,实现方式不够灵活.本文所提出的计数器结构 DALCA 同 BRICK 一样,也是一种准确计数的主动式计数器.DALCA 的存储开销比 BRICK 高出约 5%,但是时间效率高出 BRICK 数倍.DALCA 可以基于 32 位平台进行实现,实现方式比 BRICK 更为灵活.

哈希表广泛应用于路由查找、报文分类、流量测量和安全监测算法的设计之中[11,16].传统的哈希表采用开放定址或者冲突链表的方法来处理哈希冲突.为了提高哈希表的空间效率,并减小最坏条件下的查找时间,面向骨干网报文处理的哈希表结构通常设计成多级哈希表的形式[10,11,16].多级哈希表将传统的哈希表空间划分为 d 个块,各个块采用独立的哈希函数;在插入元素时,顺次访问各个块,并将元素插入到第 1 个非空的块中.若 d 个块均不为空,则插入失败.在 DALCA 设计中,需要获知插入失败的元素数目的概率分布,但现有文献中仅仅给出了计算插入失败的元素个数的期望值的方法[10,11].本文对此问题进行了研究,给出了计算多级哈希表中插入失败的元素数目的概率分布的方法,并通过仿真实验验证了所提出方法的有效性.据作者所知,本文是首次将多级哈

希表应用于大规模高速计数器结构的设计之中.

2 DALCA 的结构与算法

2.1 组成结构

在介绍 DALCA 的组成结构前,首先解释一下几个基本术语.我们将一组计数器组成的集合称作计数器向量.对于一个位宽为 b 比特的计数器 c,可以对其进行拆分,例如将其拆分成两部分,分别记作 c_1 和 c_2,c_1 为 c 的高 b_1 比特,c_2 为 c 的低 b_2 比特,其中,$b_1+b_2=b$,称 c_1 和 c_2 为 c 的子计数器.一组子计数器组成的集合称作子计数器向量.DALCA 的基本设计思想是对计数器向量进行分层,更新过程中,若第 i 层中的子计数器溢出,则向第 i+1 层中的子计数器进位;查询时,将分布在多个层中的子计数器串联起来,以获得完整的计数值.随着层次的提高,一层中所包含的子计数器的数目也逐步减少.正是基于这一原理,与原始的不分层的计数器向量相比,DALCA 可以有效降低整个计数器向量所需的存储空间.

假设计数器的总数目为 N,为了便于和 BRICK 进行性能比较,同 BRICK 一样,假设计数周期内总的更新次数,或者说所有计数器的计数值之和为 M,在实际应用中,M 可以预先确定.例如,以统计一段时间内各条数据流所包含的报文数目为例,M 即为这段时间内所到达的报文总数[9],可以根据链路数据率以及测量时间计算出来.注意到,M/N 即为平均计数值.将计数器向量划分为 L 层,第 i 层子计数向量记做 C_i,C_i 中的子计数器个数为 N_i ($N_1=N$).第 1 层到第 L–1 层中的各个子计数器均拥有 1 比特的溢出标志.第 i 层子计数器向量对应的溢出标志向量记做 O_i.在计数周期开始时,将溢出标志初始化为 0;如果某个子计数器在计数过程中发生了溢出,则将其溢出标志置为 1.为简化设计,我们假设各层子计数器的位宽相等,均为 W 比特;第 L 层子计数器向量因为无需溢出标志,故其位宽设置为 W+1 比特.

从第 2 层开始到第 L 层,子计数器向量组织成多级哈希表(multilevel Hash table[11])的形式.将子计数器向量 C_i($2\leqslant i\leqslant L$)划分为长度相等的 d 个块(不妨设 d 为 2 的正整数次幂),每块长度为$\lceil N_i/d\rceil$;第 i 层子计数器的第 j 个块记作 C_i^j,C_i^j 的块地址为 j–1.若 C_i 中的某个子计数器 $C_i[x]$计数溢出时,需向 C_{i+1} 中的子计数器 $C_{i+1}[y]$进位,则称 $C_{i+1}[y]$为 $C_i[x]$的后继子计数器.当 $C_i[x]$首次溢出时,依次查询 $C_{i+1}^1[H_i^1(x)],C_{i+1}^2[H_i^2(x)],\ldots,C_{i+1}^d[H_i^d(x)]$,并将第 1 个非空的子计数器,假设是 $C_{i+1}^p[H_i^p(x)]$,作为 $C_i[x]$的后继子计数器. $H_i^1,H_i^2,\ldots,H_i^d$ 是一组哈希函数,用于计算 C_i 中子计数器的后继子计数器在 C_{i+1} 中所有可能的地址. $\forall j\in[1,d],H_i^j:[0,N_i-1]\to[0,\lceil N_{i+1}/d\rceil-1]$.若 $C_i[x]$首次溢出时发现 $C_{i+1}^1[H_i^1(x)],C_{i+1}^2[H_i^2(x)],\ldots,C_{i+1}^d[H_i^d(x)]$ 均不为空,则在第 i 层的溢出列表中为 $C_i[x]$分配后继子计数器,溢出列表的结构下文将详细说明.在确定后继子计数器的具体位置后,需要将后继子计数器的块地址(即 p–1)存入 $C_i[x]$的高 $\log_2(d)$比特中.这样,下次访问 $C_i[x]$的后继子计数器时,根据块地址 p–1 和相应的哈希函数 $H_i^p(x)$,就可直接寻址到 $C_i[x]$的后继子计数器 $C_{i+1}^p[H_i^p(x)]$ .DALCA 名字中的 DA(direct addressing)的含义正源于此.通过上文的描述可见,$C_i[x]$一旦发生溢出,其有效的计数位宽为 W–$\log_2(d)$比特;此时,$C_i[x]$的高 $\log_2(d)$比特用于存放其后继子计数器的块地址.

第 1 层~第 L–1 层的子计数器向量设置有溢出列表,分别记做 $T_1,T_2,\ldots,T_{L-1}$.溢出列表的查表关键字为发生溢出的子计数器的地址,表项值为发生溢出的子计数器及其所有后继子计数器的计数值.溢出列表的结构如图 1 所示.表项关键字的位宽为$\lceil\log_2 N\rceil$比特,表项值的位宽为$\lceil\log_2(M+1)\rceil$比特.如果需要将某个子计数器放入溢出列表时发现溢出列表已满,则称此计数器更新失败.DALCA 的设计目标是设置 $T_1,T_2,\ldots,T_{L-1}$ 的长度之和小于门限值 T_{OV}=32,并且在此前提下,保证各级子计数器发生更新失败的概率均小于门限值 $P_f=10^{-10}$.在实际应用中,如此低的更新失败概率基本可以忽略不计.因为溢出列表的总容量很小,因此在软件实现时,可以将溢出列表放置在处理器内部的寄存器中,从而实现非常快速的查找.有关 DALCA 实现方法的讨论详见第 5 节.

图 2 给出了 DALCA 的结构示意图,为突出主题,略去了溢出列表.图 2 中,计数器向量中共有 8 个计数器,划分为 3 层,即 N=8,L=3.第 1 层~第 3 层子计数器向量的长度分别为 8,4,2.为方便表述,对第 i 层中的子计数器按照图 2 中自上而下的顺序开始编址,记作 $C_i[0],C_i[1],\ldots$.第 2 层和第 3 层子计数器向量划分为两个长度相

等的块,即 d=2.第 1 层子计数器向量采用哈希函数 $H_1^1(\cdot)$ 和 $H_1^2(\cdot)$ 对后继子计数器进行进位寻址,第 2 层子计数器向量采用哈希函数 $H_2^1(\cdot)$ 和 $H_2^2(\cdot)$ 对后继子计数器进行进位寻址.以图 2 中子计数器 $C_1[0]$为例,当其首次溢出时,首先访问 $C_2^1[H_1^1(0)]=C_2^1[0]$,发现其不为空;然后访问 $C_2^2[H_1^2(0)]=C_2^2[0]$,发现其为空.于是,确定 $C_1[0]$的后继子计数器为 $C_2^2[0]$,并且将 $C_2^2[0]$ 的块地址 1 存入 $C_1[0]$的最高比特位.图 2 中,带箭头的直线表示哈希函数,其中,实线表示某一子计数器和其后继子计数器之间的最终映射关系.

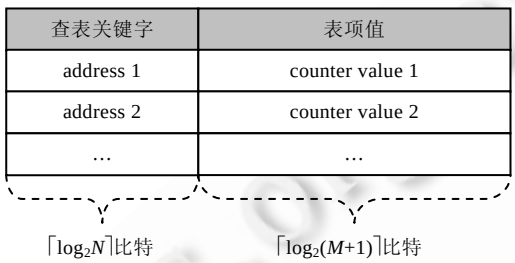


Fig.1 Structure of overflow table

图 1 溢出列表的结构示意图

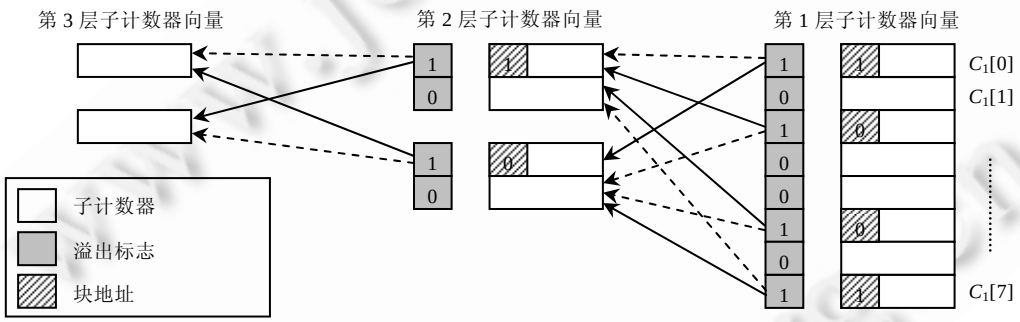


Fig.2 Architecture of DALCA

图 2 DALCA 的结构示意图

与 BRICK 相比,DALCA 需要付出额外的空间用于存储块地址,因此直观上看,DALCA 的空间效率可能会低于 BRICK.然而通过第 4.1 节的分析可见,DALCA 的空间效率和 BRICK 基本接近.这是因为 BRICK 中为了维持较低的更新失败概率,高层子计数器向量均给予了较大的空间冗余;反之,DALCA 采用多级哈希表来存储高层子计数器向量,只需较小的空间冗余就能够保证极低的更新失败概率.即 DALCA 采用多级哈希表所节省的空间抵消了其存储块地址所花费的空间,因而在空间效率方面,DALCA 和 BRICK 差异不大.但是, DALCA 中所存储的块地址为访问高层子计数器带来了极大的方便,就时间效率而言,DALCA 比 BRICK 有着明显的优势.DALCA 和 BRICK 的性能比较结果详见第 4 节.

2.2 更新与查询算法

DALCA 的更新算法描述如图 3 所示.算法输入为待更新的计数器的地址 $i(i\in[0,N-1])$,输出为空.算法的基本流程是:从第 1 层子计数器 $C_1[i]$开始更新,若 $C_1[i]$在更新完之后发生了溢出,则继续更新 $C_1[i]$的后继子计数器,如此迭代,直至更新完某个子计数器之后没有发生溢出为止.图 3 所示的更新算法中,循环控制变量 j 表示当前所更新到的子计数器向量的层数,变量 a 为当前层中待更新的子计数器的地址.当更新 $C_j[a]$时,首先判断 $C_j[a]$及其后继子计数器是否在第 j 层的溢出列表 T_j 中:若是,则更新 $T_j(a)$并退出循环(第 4 行、第 5 行);若 $C_j[a]$不在 T_j 中,则更新 $C_j[a]$.若更新之后没有发生溢出,则退出循环;否则,进入溢出处理流程(第 11 行~第 26 行).需要指出的是,判断 $C_j[a]$是否溢出应当结合 $O_j[a]$进行,若 $O_j[a]$=0,则 $C_j[a]$的有效计数位宽为 W 比特;若 $O_j[a]$=1,则 $C_j[a]$

的有效计数位宽为 W–$\log_2 d$ 比特.在溢出处理流程中,首先判断 $C_j[a]$是否是首次溢出:若 $C_j[a]$不是首次溢出,则计算 $C_j[a]$的后继子计数器的地址,更新地址指针 a,继续迭代过程(第 12 行、第 13 行);若 $C_j[a]$是首次溢出,则进行首次溢出处理流程(第 15 行~第 25 行).第 12 行中,$C_j[a][W:(W-\log_2 d+1)]$表示取得 $C_j[a]$的高 $\log_2 d$ 比特.当 $C_j[a]$首次溢出时,首先尝试为 $C_j[a]$在第 j+1 层中分配后继子计数器:若能够成功的在 C_{j+1} 中为 $C_j[a]$分配后继子计数器,则存储后继子计数器的块地址,计算后继子计数器的地址,并更新后继子计数器(第 18 行~第 20 行);若无法为 $C_j[a]$在第 j+1 层分配后继子计数器,则将 $C_j[a]$连同其后继子计数器一起放入第 j 层的溢出列表中,并更新 $T_j(a)$(第 22 行、第 23 行).

DALCA 的查询算法如图 4 所示.算法输入为待查询的计数器地址 i($i\in[0,N-1]$),输出为计数值 $C[i]$.由于计数值 $C[i]$可能由位于多个层中的子计数器组合而成,因此查询算法的基本流程就是按照从低层到高层的顺序将这些子计数器依次读取并串联起来,从而获得完整的计数值 $C[i]$.图 4 所示的算法描述中,循环计数器 j 表示当前查询到第 j 层,a 为该层中待查询的子计数器的地址,b 为前 j–1 层中 $C[i]$的各个子计数器的有效计数位宽之和.在查询到第 j 层中的子计数器 $C_j[a]$时,首先判断 $C_j[a]$是否在第 j 层的溢出列表 T_j 中:若是,则将 $T_j(a)$的值移位累加到 $C[i]$中并退出循环(第 4 行、第 5 行);否则,判断子计数器 $C_j[a]$是否发生了溢出:若 $C_j[a]$没有发生溢出,则将 $C_j[a]$的值移位累加到 $C[i]$中并退出循环(第 8 行、第 9 行);若 $C_j[a]$已经溢出,则取得 $C_j[a]$的有效计数值,移位累加到 $C[i]$中,并更新地址指针 a 和有效计数位宽之和 b,继续迭代过程(第 11 行~第 14 行).第 11 行中,$C_j[a][(W-\log_2 d):1]$表示取 $C_j[a]$的低 W–$\log_2 d$ 比特.

算法. Update.
Input: i;
Output: *NULL*.
1 $a=i$;
2 **for** j=1 to L
3 **if** a is in T_j **then**
4 $T_j(a)=T_j(a)+1$;
5 **break**;
6 **end if**;
7 $C_j[a]=C_j[a]+1$;
8 **if** $C_j[a]$ is not overflow **then**
9 **break**;
10 **else** /*overflow*/
11 **if** $O_j[a]==1$ **then**
12 $p=C_j[a][W:(W-\log_2 d+1)]$;
13 $a=p*\lceil N_{j+1}/d\rceil+H_j^{p+1}(a)$;
14 **else** /*first time overflow*/
15 $O_j[a]=1$;
16 $p=\min\{k\mid C_{j+1}^{k+1}[H_j^{k+1}(a)]$ is empty$\}$;
17 **if** $0\leqslant p<d$ **then**
18 $C_j[a][W:(W-\log_2 d+1)]=p$;
19 $a=p\times\lceil N_{j+1}/d\rceil+H_j^{p+1}(a)$;
20 $C_{j+1}[a]=2^d$;
21 **else** /*no empty entry available*/
22 put a into T_j;
23 $T_j(a)=2^W$;
24 **end if**;
25 **break**;
26 **end if**;
27 **end if**;
28 **end for**;
29 **return** *NULL*;

Fig.3 Update algorithm of DALCA
图 3 DALCA 的更新算法

算法. Query.
Input: i;
Output: $C[i]$.
1 $C[i]=0$; $a=i$; $b=0$;
2 **for** j=1 to L
3 **if** a is in T_j **then**
4 $C[i]=C[i]+T_j(a)\times 2^b$;
5 **break**;
6 **end if**;
7 **if** $j=L$ **or** $O_j[a]=0$ **then**
8 $C[i]=C[i]+C_j[a]\times 2^b$;
9 **break**;
10 **else** /*has overflowed into upper layer*/
11 $C[i]=C[i]+C_j[a][(W-\log_2 d):1]\times 2^b$;
12 $b=b+W-\log_2 d$;
13 $p=C_j[a][W:(W-\log_2 d+1)]$;
14 $a=p\times\lceil|C_{j+1}|/d\rceil+H_j^{p+1}(a)$;
15 **end if**;
16 **end for**;
17 **return** $C[i]$;

Fig.4 Query algorithm of DALCA
图 4 DALCA 的查询算法

3 DALCA 的参数设计

在实际应用中,所需的计数器数目 N 可根据被统计对象的规模进行设定;总的更新次数 M 也可根据平均计数值的大小进行较为保守的设定[9].本节研究 DALCA 的参数优化设计问题,即给定 N 和 M,通过设置恰当的哈希表级数 d、哈希表负载率 r、子计数器位宽 W 和子计数器层数 L,在保证溢出列表长度和更新失败概率均小于设定的门限值的前提下,使得 DALCA 所需的总空间最小.直观地分析,哈希表负载率 r 越大,则 DALCA 的空间效率越高.哈希表负载率 r 的大小取决于哈希表级数 d.d 越大,则所能允许的 r 越大.但是 d 越大,块地址越长,因而为了存储块地址所需付出的空间开销也越大.可见,d 可能存在着最佳取值.在哈希表级数 d、哈希表负载率 r 一定时,子计数器位宽 W 越小,则一级子计数器的空间开销越小;但是 W 越小,子计数器层数越多,且高层子计数器的数目也越多,因此,高层子计数器的空间开销也越大.可见,在哈希表级数 d、哈希表负载率 r 一定时,W 也存在着使得总的空间开销最小的最佳取值.我们采用枚举-检验法来完成 DALCA 的参数优化设计,方法的基本流程如下:

(1) d 依次取 4,8,16 和 32;固定 d,负载率 r 从 0.25 开始,以 0.05 为步长,增长到 0.95;固定 d 和 r,求 W 的最佳取值;W 的取值确定后,L 的取值也随之确定;

(2) 对步骤(1)得到的 d,r,W 和 L 进行合法性检验,若能够保证溢出列表总长度小于门限值 T_{OV}=32,且各层子计数器的更新失败的概率均小于门限值 $P_f=10^{-10}$,则检验通过;

(3) 在所有通过检验的参数设定中,选择使 DALCA 的总空间最小的一组(d,r,W,L)作为最佳的参数设定.

可见,在参数优化设计过程中,需解决下列两个核心问题,即:

问题 1. 假设 d 和 r 已知,问 W 应如何取值,使得 DALCA 所需的总空间**最小.

问题 2. 有 N 个元素依次插入到级数为 d 的多级哈希表中,表空间为$\lceil N/r \rceil$$(0<r<1)$.若某个元素插入时,其对应的 d 个位置均不为空,则称此元素插入失败.求插入 N 个元素后,有 T 个元素插入失败的概率 $P_f(T)$.

3.1 子计数器位宽的优化设置

如上文所述,假设共有 N 个计数器 $C_1,C_2,\ldots,C_N$,计数周期内,所有计数器的计数值之和为 M,即 $\sum_{i=1}^{N} C_i = M$.因此,计数值的上限为 M,计数器的最大位宽为$\lceil \log_2(M+1) \rceil$.除最高层子计数器外,各层子计数器向量由 W 比特宽的子计数器和 1 比特宽的溢出标志构成;最高层子计数器无需溢出标志,为了保持各层子计数器的位宽一致,将最高层的子计数器位宽设为 W+1.假设哈希表级数 d 恰好为 2 的整数次幂.设 DALCA 中子计数器的总层数为 L,则 L 需满足:

$$W+1+(L-1)\cdot(W-\log_2 d)=\lceil \log_2(M+1) \rceil \tag{1}$$

解上述方程,得到 L 为

$$L=\left\lceil \frac{\log_2(M+1)-W-1}{W-\log_2 d} \right\rceil +1 \tag{2}$$

设第 i 层所需的子计数器的数目最多为 $N_i(N_1=N)$.当计数值等于 2^W 时,一级子计数器就需要向二级子计数器进位;由于计数值等于 2^W 的计数器最多有$\lceil M/2^W \rceil$个,因此 $N_2=\lceil M/2^W \rceil$.第 3 层中的子计数器数目为经过第 1 层、第 2 层共同表示,但还是需要进位的子计数器的数目.注意到,第 1 层、第 2 层计数器的有效计数位之后为 $2W-\log_2 d$,而非 $2W$,因此 $N_3=\left\lceil \frac{M}{2^{2W-\log_2 d}} \right\rceil$.一般地,第 $i(2\leqslant i\leqslant L)$层所需的子计数器数目为

$$N_i=\left\lceil \frac{M}{2^{(i-1)\cdot W-(i-2)\cdot \log_2 d}} \right\rceil (i\in[2,L]) \tag{3}$$

于是,DALCA 所需的总空间:

**总空间指的是各层子计数器的空间之和,不包括溢出列表.下文如果不特别说明,同样遵照此约定.

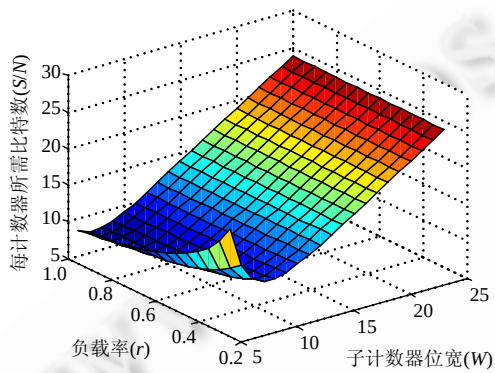
$$S=(W+1)\cdot\left(N+r^{-1}\cdot\sum_{i=2}^{L}N_i\right) \tag{4}$$

其中,r 为哈希表负载率.将公式(3)带入公式(4),两边除以 N,得:

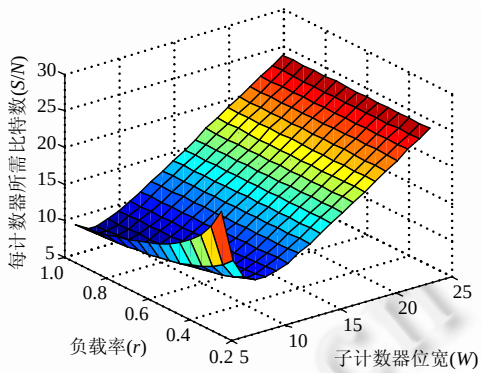
$$S/N\approx(W+1)\cdot\left(1+\frac{M}{r\cdot N}\cdot\sum_{i=2}^{L}\frac{1}{2^{(i-1)\cdot W-(i-2)\cdot\log_2 d}}\right) \tag{5}$$

其中,S/N 即为每个计数器所需的比特数,L 由公式(2)给出,约等号表示由于舍弃了公式(3)中的向上取整操作而引起的误差.当 M,N,d 和 r 给定,满足 $\frac{dS}{dW}=0$ 的 W 即为最佳值.考虑到对公式(5)直接求导计算的过程较为复杂,我们给出了不同 W 取值下,S/N 的数值计算的结果.

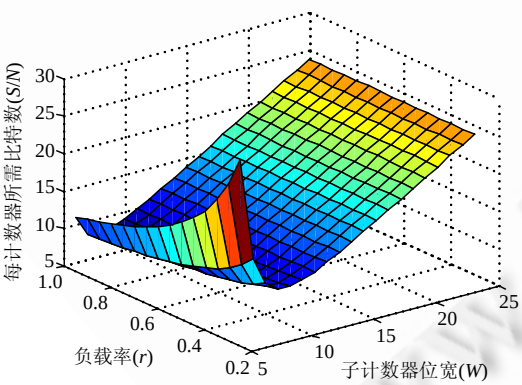
图 5 给出了 $N=2^{20}$,$M=2^{24}-1$,每计数器所需的比特数 S/N 随着负载率 r 和子计数器位宽 W 的变化关系.图 5 中,d 分别取 4,8,16,32,哈希表负载率 r 从 0.25 开始,以 0.05 为步长,增长到 0.95.由图 5 可见,不同负载率下,总存在着最佳的子计数器位宽 W,使得每计数器所需的比特数最小.



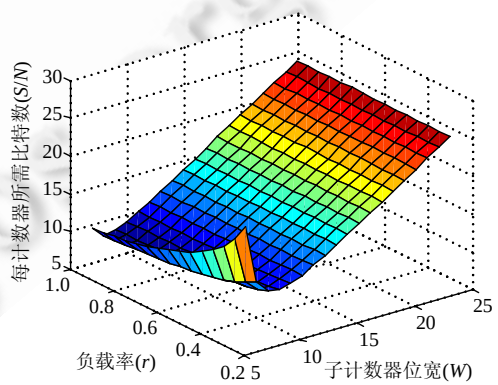
(a) d=4



(b) d=8



(c) d=16



(d) d=32

Fig.5 Bits per counter under various load rate and sub-counter width, when $N=2^{20}$, $M=2^{24}-1$

图 5 $N=2^{20}$,$M=2^{24}-1$ 时,每计数器所需的比特数随负载率和子计数器位宽的变化关系

3.2 合法性检验

在给出合法性检验的方法之前,首先有必要介绍 d-left 哈希表.d-left 哈希表的表空间划分为等长的 d 个块,每块包含若干个桶,每个桶的深度为 b.在插入元素 e 时,通过 d 个哈希函数,分别在 d 个块中寻址到相应的桶,并将 e 插入到这 d 个桶中负载最轻的那个桶中去;若同时有多个负载最轻的桶,则选择最左边的那个.图 6 给出了 d-left 哈希表的示例,其中,表空间划分为 4 块,即 d=4,每块包含 5 个桶,每个桶的深度为 b=3.元素 e 插入到最左

边的块的第 4 个桶中,如图 6 所示.可见,上文提到的多级哈希表是 *d*-left 哈希表的一种特例,即对应于桶深为 1 的 *d*-left 哈希表.

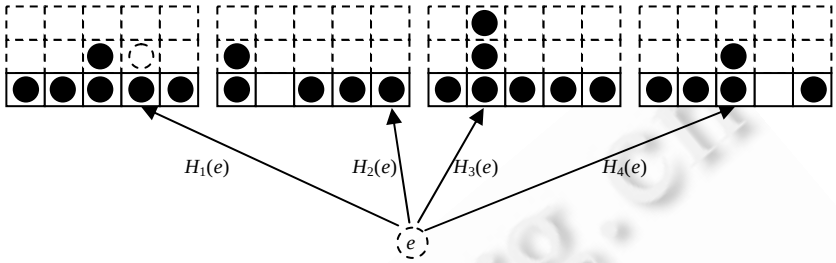


Fig.6 An example of *d*-left hashing table

图 6 *d*-left 哈希表示例

Mitzenmacher 等人给出了一种计算各个桶中的元素数目分布的方法.定义平均桶负载为插入到 *d*-left 哈希表中的元素总数目比上 *d*-left 哈希表中的桶的总数目.令 $X_j(s,i)$为平均桶负载为 s 时,第 j 个块中至少有 i 个元素的桶在所有桶中所占的比例,其中,$1\leqslant j\leqslant d$,$1\leqslant i\leqslant b$.为便于表述,令 $k=(i-1)\times d+j$ 且 $Y_k(s)=X_j(s,i)$.

Mitzenmacher 等人指出,当桶的数量和桶深都趋于无穷大时,$\forall k\in[1,d\times b]$,$Y_k(s)$可以用下列微分方程组来描述:

$$\frac{\mathrm{d}y_k}{\mathrm{d}s}=d^d(y_{k-d}-y_k)\prod_{t=k-d+1}^{k-1}y_t \tag{6}$$

其中,$y_0(r)=y_{-1}(r)=\ldots=y_{1-d}(r)=1/d$,$y_k(0)=0$.上述微分方程组可以通过常用的数值方法进行求解.一般地,当桶的数量和桶深都足够大时,上述方程组也能够极佳地逼近 $Y_k(s)$.对公式(6)直观的解释是,y_k 的增长速率等于下列事件发生的概率,即:插入某个元素时,其恰好落在第 j 个块中的某个负载为 $i-1$ 的桶中(这样,第 j 个块中负载为 i 的桶的数目就增长了).这就意味着从第 1 块~第 $j-1$ 块中随机选择的 $j-1$ 个桶,其负载至少为 $i\left(d^{j-1}\prod_{t=k-j+1}^{k-1}y_t\right)$;从第 j 块中随机选择的桶,其负载恰好为 $i-1(d(y_{k-d}-y_k))$;从第 $j+1$ 块到第 d 块中随机选择的 $d-j$ 个桶,其负载至少为 $i-1\left(d^{d-j}\prod_{t=k-d+1}^{k-j}y_t\right)$.

在介绍完 *d*-left 哈希表之后,我们转入正题,给出计算多级哈希表中插入失败的元素数目的概率分布的方法.如上文所述,多级哈希表可看作是桶深 $b=1$ 的 *d*-left 哈希表,其插入某个元素时发生插入失败的概率,等价于在 *d*-left 哈希表中该元素插入到非空的桶中的概率.设有 N 个元素依次插入到级数为 d、共包含$\lceil N/r\rceil(0<r<1)$个桶的 *d*-left 哈希表中,用随机变量 Z_i 表示第 i 个插入的元素是否落在了非空的桶中:若是,则 $Z_i=1$;否则,$Z_i=0$.N 个元素插入完成后,落在非空的桶中的元素数目用随机变量 Z 表示,有 $Z=\sum_{i=1}^{N}Z_i$.设 $Z_i=1$ 的概率为 $P(i)$,于是,Z_i 的母函数为

$$g_{Z_i}(z)=\sum_{j=0}^{\infty}P(Z_i=j)z^j=1-P(i)+P(i)z \tag{7}$$

考虑到 $N>>E[Z]$,因此认为对于任意 $i,j\in[1,N]$且 $i\neq j$,Z_i 和 Z_j 相互独立.于是,Z 的母函数为

$$g_Z(z)=\prod_{i=1}^{N}g_{Z_i}(z)=\prod_{i=1}^{N}(1-P(i)+P(i)z) \tag{8}$$

其中,z^k 的系数即为 N 个元素插入完成后,恰好有 k 个元素落在非空的桶中的概率,即 $\Pr\{Z=k\}$.对应于多级哈希表,$\Pr\{Z=k\}$即为恰好有 k 个元素发生插入失败的概率.

设 d-left 哈希表中,平均桶负载为 $\frac{i}{\lceil N/r \rceil}$ 时,元素数目至少为 2 的桶在所有桶中所占比例为 $\Pr\{l \geqslant 2\}$.由公式(6)可知:

$$\Pr\{l \geqslant 2\} = \sum_{k=d+1}^{2d} y_k \left(\frac{i}{\lceil N/r \rceil} \right) \tag{9}$$

如上文所述,$P(i)$为第 i 个插入的元素落在了非空的桶中的概率,这一概率等价于 $\Pr\{l \geqslant 2\}$的增长速率,即:

$$P(i) = \Pr'\{l \geqslant 2\} = \sum_{k=d+1}^{2d} \frac{\mathrm{d}y_k}{\mathrm{d}s}\bigg|_{s=\frac{i}{\lceil N/r \rceil}} \tag{10}$$

于是,由公式(8)和公式(10),即可计算出 $\Pr\{Z=k\}$.

当 N 较大时,为了降低计算开销,可以采用分段逼近的方法计算 $P(i)$.选定单位步长Δ后,$\forall i \in [(k-1)\times\Delta, k\times\Delta]$($k$ 为正整数),$P(i)=P(k\times\Delta)$.图 7 给出了分段逼近法的示意图.

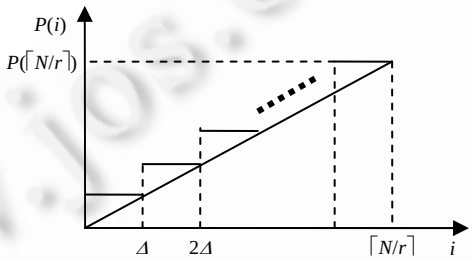
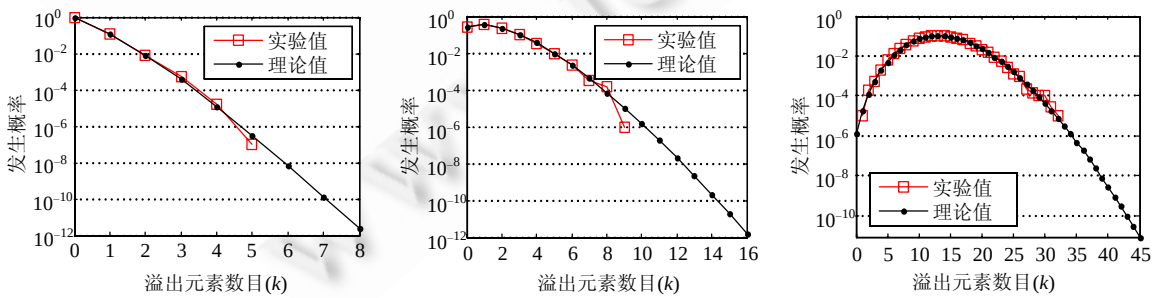


Fig.7 Calculating $P(i)$ using a segmented approximation method

图 7 利用分段逼近法计算 $P(i)$

为了验证上述计算方法的正确性,我们将理论计算与仿真实验的结果进行了对比.仿真实验中,哈希表级数 d=8,哈希表负载率 r=0.65,元素总数分别取 $N=10^4$,$N=10^5$ 和 $N=10^6$,相应的,仿真实验重复次数分别为 10^7 次、10^6 次和 10^5 次,对每次实验中发生插入失败的元素的数目进行记录.采用分段逼近法计算 $P(i)$,设置单位步长为 $\Delta=\lfloor 10^{-4}N/r \rfloor$,即保证平均负载率以 10^{-4} 为步长增长.图 8 给出了仿真结果和理论计算结果的对比.由图 8 可见,理论计算的结果和仿真结果基本一致,但是当 k 增大到一定程度,由于对应的 $\Pr\{Z=k\}$取值较小,仿真实验的结果已经不足以得到对 $\Pr\{Z=k\}$进行精确的估计,因而仿真实验的结果与理论计算的结果差别较大.仿真实验用的计算机采用 Pentium Dual-Core 3.2GHz 处理器,2GB 内存;仿真程序采用 Visual C++编写,完成 3 次实验所需的时间均约为 80m.



(a) $N=10^4$ (b) $N=10^5$ (c) $N=10^6$

Fig.8 Analytic and experimental results of the distribution of the number of failed elements during insertion

图 8 插入失败的元素数目的概率分布的理论计算结果与仿真结果

3.3 设计实例

表 2 给出了 $N=2^{20}$,$M=2^{24}$ 时,DALCA 的参数优化设置的结果.由表 2 可见,随着 d 由 4 增长到 16,所能够允许的负载率 r 也逐步提高,因而 DALCA 的空间效率也逐步提高.但是当 d=32 时,DALCA 的空间效率反而小于 d=16 时的空间效率.这是因为此时通过提升负载率所节省的空间,不足以抵消存储更长的块地址所需花费的空间.注意到,d 越大,L 越大,这意味着读取完整的计数值所需的访存次数也越多.可见,并非多级哈希表级数 d 越大,DALCA 的性能越佳.考虑到 d=8 或 d=16 时 DALCA 在空间效率和时间效率之间能够达到较好的平衡,因此在实际应用中,取 d=8 或者 d=16 较为适宜.

Table 2 Optimized parameter settings when $N=2^{20}$,$M=2^{24}$

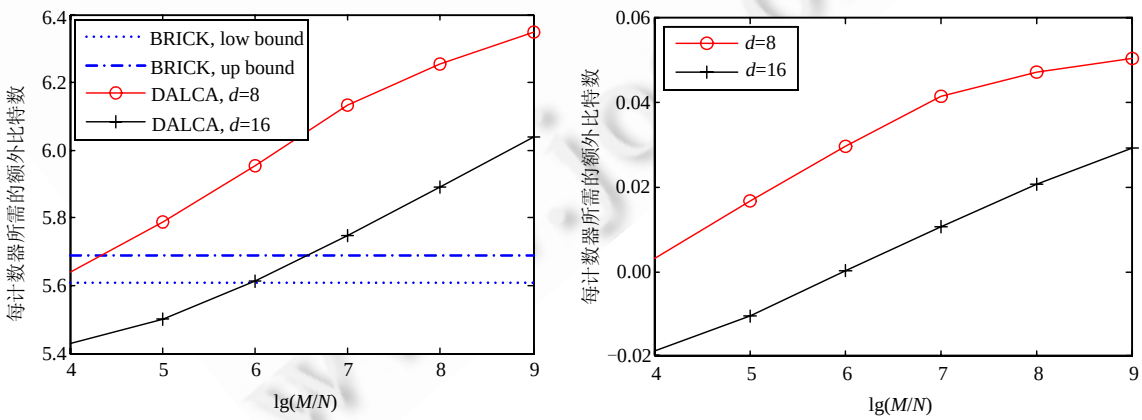
表 2 $N=2^{20}$,$M=2^{24}$ 时,参数优化设置的结果

d	r	S/N	W	L	N_2~N_L	T_1~T_{L-1}
4	0.35	10.63	8	4	2^{16}, 2^{10}, 2^4	23, 6, 2
8	0.65	9.64	7	5	2^{17}, 2^{13}, 2^9, 2^5	15, 6, 3, 2
16	0.80	9.43	7	7	2^{17}, 2^{14}, 2^{11}, 2^8, 2^5, 2^2	5, 3, 2, 2, 1, 1
32	0.90	9.48	7	9	2^{17}, 2^{15}, 2^{13}, 2^{11}, 2^9, 2^7, 2^5, 2^3	4, 3, 2, 2, 2, 1, 1, 1

4 性能分析与比较

4.1 空间效率

图 9 给出了 $N=2^{20}$ 时,DALCA 和 BRICK 的空间效率比较.根据已有的研究结论[9]以及我们对真实网络流量的分析结果,骨干网链路中的平均流长在 2^4~2^7 之间,即 $\log_2(M/N)$的取值范围为[4,7].图 9(a)中给出了 $\log_2(M/N)$取 4~9、d 分别取 8 和 16 时,DALCA 中每计数器所需的额外比特数(即,S/N–$\log_2(M/N)$).另外,根据文献[9], BRICK 中每计数器所需的额外比特数的变化范围为[5.62,5.69],因此作为比较,在图 9(a)中同时也画出了 BRICK 中每计数器所需的额外比特数的上界和下界.由图 9(a)可见,随着 $\log_2(M/N)$值的增加,DALCA 的空间需求的增长速度高于 BRICK,但是相对增长幅度不大.图 9(b)给出了 DALCA 每计数器所需的比特数(S/N)相对于 BRICK 的增加幅度,由图 9(b)可见,相对增幅最大约为 5%.图 9(b)中,相对增幅为负的点表示在对应的 d 和 $\log_2(M/N)$取值下,DALCA 的空间效率优于 BRICK.此外,由图 9 可见,d=16 时,DALCA 的空间效率要略优于 d=8 时的空间效率.其原因在于:d=16 时,多级哈希表负载率为 80%,明显高于 d=8 时的负载率 65%.



(a) 每计数器所需的额外比特数

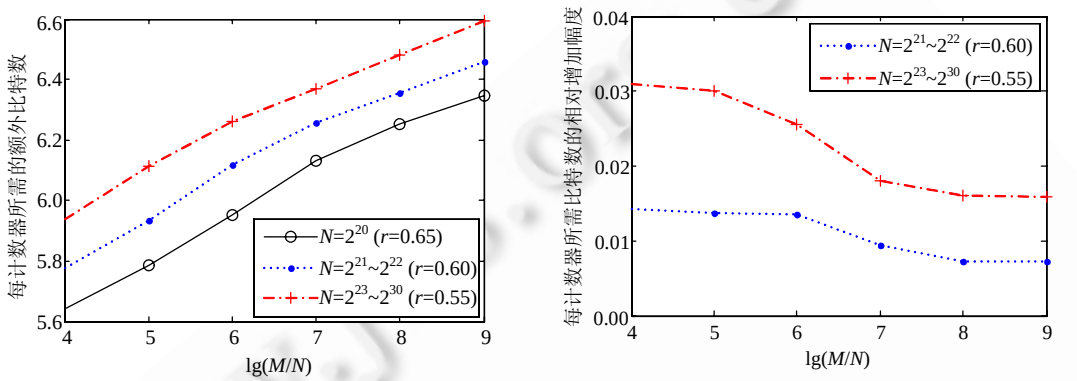
(b) DALCA 每计数器所需的比特数相对于 BRICK 的下限值(5.62)的增加幅度

Fig.9 Comparison of space efficiency of DALCA and BRICK

图 9 DALCA 和 BRICK 的空间效率比较

当计数器的总数目 N 增大时,各层子计数器的数目可能也随之增大.因此,为了满足溢出列表总长度约束和

子计数器向量的更新失败概率约束,需要适当地降低哈希表负载率.以 d=8 为例,N=2^{20} 时,由上文可知,哈希表负载率 r=0.65;而当 N=2^{21} 或者 N=2^{22} 时,只需要将哈希表负载率降为 0.60;当 N=2^{23},2^{24},…,2^{30} 时,只需要将哈希表负载率降为 0.55,就能够保证溢出列表总长度和各层子计数器的更新失败概率均小于设定的门限值.图 10(a)给出了不同 N 和 M 取值下每计数器所需的额外比特数.由于 r 一定时每计数器所需的额外比特数非常接近,因此图 10 中仅仅针对每个 r 分别画出一条曲线,而没有为每个 N 画出一条曲线.图 10(b)给出了每计数器所需的比特数相对于 N=2^{20}(此时 r=0.65)时的增加幅度.从图 10(b)可见,即使当 N=2^{30} 时,相对增长幅度最大约为 3%.可见,DALCA 的空间效率受 N 的影响不大,因而具备良好的可扩展性.



(a) 不同 N 和 M 取值下,每计数器所需的额外比特数 (b) 每计数器所需的比特数相对于 N=2^{20},r=0.65 时的增加幅度

Fig.10 The variation of space efficiency of DALCA when N increments from 2^{20} to 2^{30}

图 10 N 从 2^{20} 增大到 2^{30} 时,DALCA 的空间效率的变化情况

前文我们均假定各层子计数器的更新失败概率低于门限值 P_f=10^{-10}.对于某些应用而言,可能需要进一步降低此门限值,比如要求 P_f=10^{-20}.通过实验我们发现,通过略微降低哈希表负载率 r,即可保证溢出列表总长度小于门限值 T_{ov}=32 的前提下,显著降低子计数器更新失败的概率.具体而言,当 N=2^{20} 时,设置负载率 r=0.62;当 N=2^{21},2^{22} 时,设置负载率 r=0.6;当 N=2^{23},2^{24},…,2^{30} 时,设置负载率 r=0.52 即可满足 P_f=10^{-20} 且 T_{ov}=32 的约束要求.此时,相对于 P_f=10^{-10} 时 DALCA 的每计数器所需比特数的相对增幅如图 11 所示.

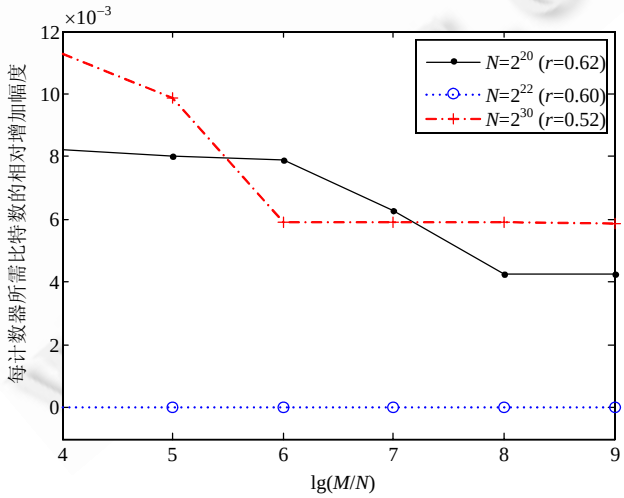


Fig.11 Relative increment of bits per counter when P_f=10^{-20} to that when P_f=10^{-10}

图 11 P_f=10^{-20} 时,每计数器所需比特数相对于 P_f=10^{-10} 时的增长幅度

由图 11 可见,降低溢出概率门限值所带来的空间开销增幅最大约为 1.2%,基本可以忽略不计.假设子计数器层数 L=10,计数周期为 1m,P_f=10^{-10} 时,需要约 1 900 年才会发生一次计数错误.若 P_f=10^{-20} 时,则此误差已经可以忽略不计.

4.2 时间效率

本节采用真实的骨干网流量数据,对 DALCA 和 BRICK 的时间效率进行了比较.实验所采用的流量数据来源于 CAIDA 所提供的 Abilene-I 和 Abilene-III 骨干网流量数据,分别采集自 OC-48 和 OC-192 链路.我们对 Abilene-I 中的 IPLS-KSCY-20020814-090000-0.gz 流量数据文件和 Abilene-III 中的 20040601-193121-0.gz 流量数据文件的前 2GB 进行了处理,将其由 ERF 格式转化为本文实验所需要的数据流级的流量数据,下文将这两条流量数据分别称作 trace 1 和 trace 2.在转换过程中,采用五元组〈源 IP,目的 IP,源端口号,目的端口号,协议类型〉作为流标识,没有考虑流的超时结束问题.trace 1 和 trace 2 的详细信息见表 3.已有的研究工作指出,网络数据流的平均流量约为 16~32 左右[9,17],由于我们选取的流量数据的持续时间较长,且没有考虑数据流的超时结束问题,因此,所统计出的平均流量大于文献[9,17]中的统计结果.

Table 3 The detailed statistics information of backbone traces used in experiments

表 3 实验所采用的骨干网流量数据的详细信息

名称	链路速率	持续时间(s)	总包数	总流数	平均流长
Trace 1	OC-48	583.778	33 367 912	255 607	130.5
Trace 2	OC-192	264.745	28 468 064	599 898	47.5

衡量 BRICK 和 DALCA 时间效率的性能指标是执行逐包查询和逐包更新操作所需的平均访存次数(average memory access times)以及平均所需访问的比特数(average bits accessed).所需访问的比特数等于访存次数乘以存储器的数据总线位宽(或称为访存位宽).对于 DALCA 而言,访存模式则非常简单——若计数值分布在前 k 层子计数器中,则查询操作只需要 k 次读操作.对于 BRICK 而言,查询时不仅要读取子计数器的值,还需要读取溢出标志,若发现当前的子计数器溢出,还需要继续读取后继的子计数器.BRICK 的访存位宽为 64 比特,根据文献[9]给出的 BRICK 的参数设置方案,某些高层子计数器向量和溢出标志向量较短,可以打包读取,从而减小访存次数.表 4 给出了 BRICK 查询操作的访存模型.

Table 4 Memory access model of BRICK during querying

表 4 BRICK 查询操作的访存模型

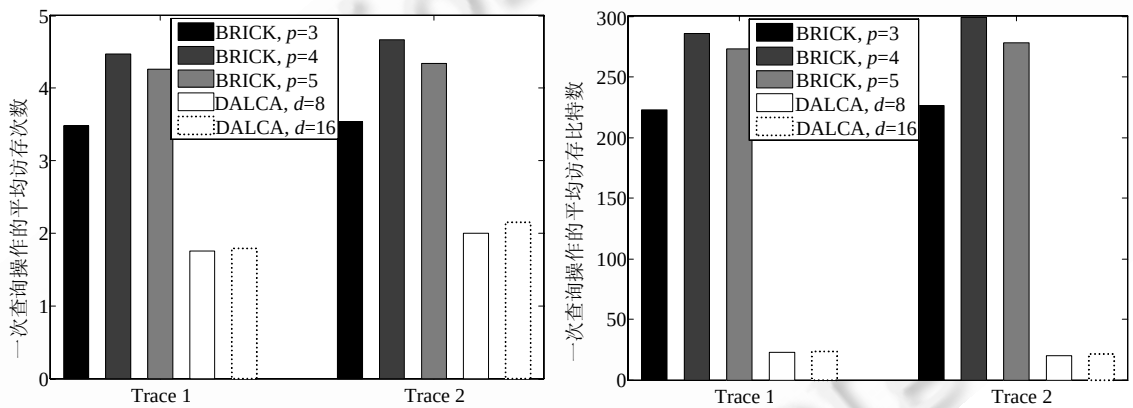
p	访存序号	访问内容	等级序号	等级区间范围	区间内的访存次数
3	1	$A_1[i]$	1	$[1,M/N\times2^3-1]$	2
	2	I_1			
	3	A_2	2	$[M/N\times2^3,M/N\times2^{20}-1]$	4
	4	I_2, A_3			
4	1	$A_1[i]$	1	$[1,M/N\times2^2-1]$	2
	2	I_1			
	3	A_2	2	$[M/N\times2^2,M/N\times2^4-1]$	4
	4	I_2			
	5	A_3, I_3	3	$[M/N\times2^4,M/N\times2^8-1]$	5
	6	A_4	4	$[M/N\times2^8,M/N\times2^{20}-1]$	6
5	1	$A_1[i]$	1	$[1,M/N\times2^2-1]$	2
	2	I_1			
	3	A_2	2	$[M/N\times2^2,M/N\times2^4-1]$	4
	4	I_2, A_3			
	5	I_3, A_4, I_4, A_5	3	$[M/N\times2^4,M/N\times2^{20}-1]$	5

表 4 中,p 表示 BRICK 的子计数器级数[9].以 p=3 为例,此时,BRICK 由三级子计数器构成,分别是 A_1,A_2,A_3,其大小分别是 $64\times(\log_2(M/N)+3)$,15×4 和 3×13;前两级子计数器所对应的溢出标志向量分别为 I_1,I_2,其大小分别是 64×1 和 15×1.假设需要读取第 i 个子计数器,则第 1 次读操作读取 $A_1[i]$,第 2 次读操作读取 I_1,第 3 次读操作读取 A_2,第 4 次读操作读取 I_2 和 A_3.对应上述的读取模式,可以将计数值分为两个等级:

第 1 个等级的区间范围是[1,$M/N\cdot 2^3-1$],若计数值在此区间内,则查询操作需要两次访存操作;

第 2 个等级的区间范围是[$M/N\cdot 2^3$,$M/N\cdot 2^{20}-1$],若计数值在此区间内,则查询操作需要 4 次访存操作.

图 12 给出了 DALCA 和 BRICK 查询操作的时间效率比较.对于 trace 1,设置 log(M/N)=8,d=8 时,DALCA 的最佳子计数器位宽为 12 比特,子计数器级数为 3;d=16 时,DALCA 的最佳子计数器位宽为 12 比特,子计数器级数为 4.对于 trace 2,设置 log(M/N)=6,d=8 时,DALCA 的最佳子计数器位宽为 9 比特,子计数器级数为 4;d=16 时,DALCA 的最佳子计数器位宽为 9 比特,子计数器级数为 5.由图 12(a)可见,进行查询操作时,DALCA 的平均访存次数约为 BRICK 的一半.BRICK 为了一次读取一组(文献[9]中称为一个 bucket)中共 64 个的子计数器的溢出标志,需要将子计数器的值和子计数器的溢出标志分开存储,从而导致在查询时访存次数倍增.以 Trace 1 为例,由表 4 可见,当 BRICK 中子计数器层数取 p=3 时,对于所有计数值小于 $2^{11}-1$ 的计数器而言,均需要 2 次访存操作才能得到其计数值.而在同等情况下,由于 DALCA 的最佳子计数器位宽为 W=12,因此对于所有计数值小于 $2^{12}-1$ 的计数器而言,只需 1 次访存操作即可得到其计数值.可见,设计机理上的差别导致了 DALCA 执行查询操作时的吞吐量要高于 BRICK.DALCA 的访存位宽小于 16 比特(trace 1 为 13 比特,trace 2 为 10 比特),而 BRICK 的访存位宽为 64 比特.因此,DALCA 的平均所需访问的比特数明显小于 BRICK,仅仅约为 BRICK 的 1/10,如图 12(b)所示.



(a) 平均访存次数 (b) 平均所需访问的比特数

Fig.12 Comparison of time efficiency of DALCA and BRICK during querying

图 12 DALCA 和 BRICK 查询操作的时间效率比较

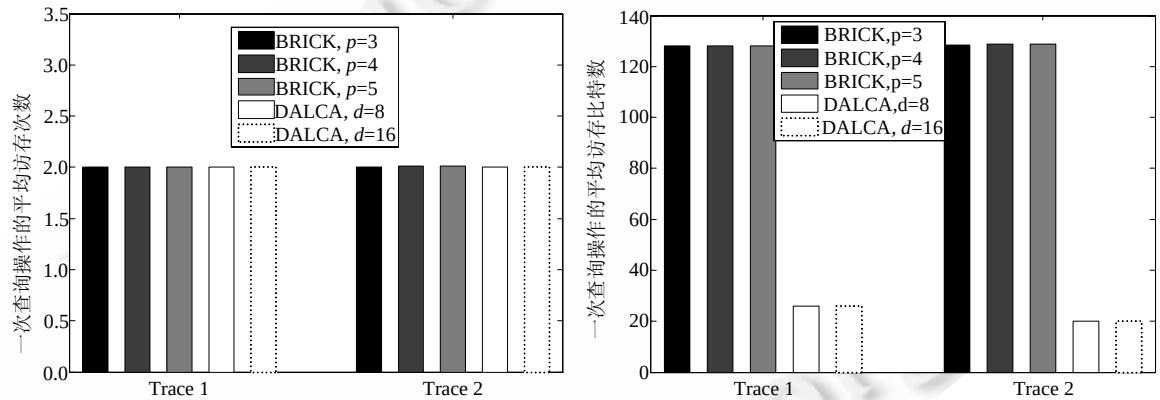
更新操作从第 1 层子计数器开始;若发现第 1 层子计数器发生了溢出,则还需更新第 2 层子计数器;依此类推.若一个计数周期内某个计数器的计数值最终为 f,则对于 DALCA 而言,第 1 层子计数器需要读写各 f 次;若 $f>2^W-1$(W 为 DALCA 的子计数器宽度),则第 2 层子计数器需要读写各$\lfloor f/2^W \rfloor$次;若 $f>2^{2W-\log_2 d}-1$,则第 3 层子计数器需要读写各$\left\lfloor f/2^{2W-\log_2 d} \right\rfloor$次;依此类推.DALCA 更新操作额外的代价来源于当某个子计数器首次溢出时,需要在高层子计数器向量为其分配后继子计数器,此时最多需要访问 d 个位置,以确定后继子计数器的地址.对于 BRICK 而言,更新高层子计数器时,多出了读取溢出标志向量的代价,本文称此为 BRICK 的溢出代价.此外,当某个子计数器首次溢出时,不仅需要读取,还需要写溢出标志向量,本文将这两次访存操作称为首次溢出代价.表 5 给出了 BRICK 的更新操作的访存模型,以 p=3 为例,当计数值达到 $M/N\times 2^3$ 时,就需要更新二级子计数器.为了对二级子计数器进行寻址,需要读取一级子计数器的溢出标志向量 I_1,因此,溢出代价为 1;若是首次溢出,还需要将 I_1 写回,因此首次溢出代价为 2.当计数值达到 $M/N\times 2^7$ 时,就需要更新三级子计数器.由表 4 可见,由于 A_3 和 I_2 可以打包读取,因此此时无需付出溢出代价和首次溢出代价.需要指出的是,表 5 中的访存模型已经应用了优化策略,对于某些可以合并读取的子计数器,我们将之视为同一级子计数器.例如 p=5 时,A_4 和 A_5 可以合并为一级子计数器,因而表 5 中对应于 p=5 的情形,只需 3 个溢出门限,而非 4 个.

Table 5 Memory access model of BRICK during updating

表 5 BRICK 更新操作的访存模型

p	溢出门限序号	溢出门限	溢出代价	首次溢出代价
3	1	$M/N\times2^3$	1	2
	2	$M/N\times2^7$	0	0
4	1	$M/N\times2^2$	1	2
	2	$M/N\times2^4$	1	2
	3	$M/N\times2^8$	0	0
5	1	$M/N\times2^2$	1	2
	2	$M/N\times2^4$	0	0
	3	$M/N\times2^7$	0	0

图 13 给出了 DALCA 和 BRICK 查询操作的时间效率比较结果.由图 13 可见,DALCA 和 BRICK 平均每次更新操作所需的访存次数均约等于 2,如图 13(a)所示.这是因为 DALCA 和 BRICK 的一级子计数器位宽都较宽,对于一级子计数器的读写占据了更新过程中所有访存操作的绝大部分比例,只有当一级子计数器溢出时,才需要访问二级子计数器.此外需要指出的是,虽然两者的访存次数接近,但是由于两者访存位宽差别较大,导致 DALCA 的平均所需访问的比特数仍然明显小于 BRICK,仅仅约为 BRICK 的 1/6,如图 12(b)所示.



(a) 平均访存次数 (b) 平均所需访问的比特数

Fig.13 Comparison of time efficiency of DALCA and BRICK during updating

图 13 DALCA 和 BRICK 更新操作的时间效率比较

由上述仿真实验的结果可见,DALCA 在所需访问的比特数方面有较为明显的优势,其原因主要在于其访存位宽远小于 BRICK.较小的访存位宽将减小硬件布线的代价,降低功耗,并有利于接口速率的提高.若实现时采用最常见的 QDR SRAM,其位宽为 32 比特,对于 DALCA 的访存次数不会产生影响;但是对于 BRICK 而言,访存次数就会提高一倍,因而吞吐量也会下降为采用 64 比特宽的存储器时的一半.此外,由于 DALCA 所需的访存位宽较窄,因此,若存在多个可同时更新的统计项,则在一次访存中可以对其进行打包查询或者更新.例如,假设同时需要统计各条数据流的总报文数和总字节数,或者同时需要统计各条数据流的总报文数以及其中 TCP 报文的数目,利用位宽为 32 比特的存储器,DALCA 即可实现对两个统计计数器的打包访问.

需要指出的是,我们采用平均访存次数,而非最大访存次数作为衡量 BRICK 和 DALCA 时间效率的性能指标.对于网络处理而言,有时也需要关心最坏条件下的性能(worst-case performance),以保证在链路满负载时仍能实现线速处理.因此,我们也对 DALCA 的最坏条件下的时间效率作一些分析.以 $N=2^{20}$,$\log_2(M/N)=4$ 为例,由表 2 可知,若 $d=8$,则 $L=5$,$W=7$,即 DALCA 共由 5 级子计数器组成,每层子计数器的位宽为 7 比特.于是,完成一次更新操作,DALCA 最大的访存次数为 17 次;作为对比,BRICK 的最大访存次数为 10 次[9].但是可以对 DALCA 进

行优化,以减小最大访存次数.首先,为了减小最大访存次数,可以采用较小的哈希表级数.例如,第 2 层子计数器依然采用 8 级哈希表进行组织,但是第 3 层~第 5 层子计数器可以用 4 级哈希表进行组织,并降低负载率.这样,对于 DALCA 的空间效率几乎没有影响,但完成更新操作所需的最大访存次数却可以减小为 13 次.随着 $\log_2(M/N)$值的增加,DALCA 的子计数器层数逐渐递减,例如当 $\log_2(M/N)=6$ 时,DALCA 的子计数器层数为 4,此时若采用前述优化策略,则完成更新操作所需的最大访存次数为 11 次;当 $\log_2(M/N)=8$ 时,DALCA 的子计数器层数为 3,完成更新操作所需的最大访存次数为 9 次.从上述分析可见,DALCA 的最坏条件下的性能和 BRICK 接近,在某些条件下甚至优于 BRICK.

基于软件实现时,由于存储器带宽通常由多个处理模块所共享,对于某个处理模块而言,存储器带宽和访问时延都在动态变化,因此对于软件实现而言,关注平均访存次数比关注最坏条件下的访存次数更有实际意义.而为了保证最坏条件下的性能,通常需要采用硬件实现.DALCA 结构简单,便于应用并行和流水线处理等硬件设计中常用的技术,从而达到极高的性能.有关实现问题的讨论详见第 5 节.

5 实现的讨论

采用软件的方式实现 DALCA 时,其难点在于如何高效地实现哈希函数计算和溢出列表查找.DALCA 特别适合采用网络处理器实现.网络处理器内部通常带有 CRC 校验模块和内容寻址存储器(content addressable memory,简称 CAM),前者可用于实现哈希函数;后者可直接用于实现溢出列表的快速查找.例如 Intel IXP 2800/2850 网络处理器[18],内部带有计算哈希函数的硬件单元模块,只需 3 个指令周期(约 2.1ns)即可完成一次 48 比特的哈希运算.此外,Intel IXP 2800/2850 网络处理器内部包含 16 个微引擎,每个微引擎中都带有容量为 16 的 CAM,只需 1 个指令周期(约 0.7ns)即可完成一次查找操作.可以将溢出列表放置在这些 CAM 中,从而实现快速查找.BRICK 要求基于 64 位处理器平台进行实现,而 DALCA 在当前仍然广泛应用的 32 位平台上依然可以实现.

DALCA 由于结构简单,也非常便于硬件实现.在硬件实现时,可以充分利用并行和流水线处理技术,将各级子计数器以及同级子计数器的各个块都采用独立的存储单元实现,从而获得极高的性能.在硬件实现时,哈希函数可以参考已有的一些方法[19−21]进行实现;而溢出列表由于长度较短,只需采用多个并行的比较器实现即可.采用硬件实现时,由于多级哈希表的各个块可以并行访问,因此可以将多级哈希表的级数 d 设置得较大,例如取 d=16,从而获得较高的空间效率.当前,中等工艺水平的 FPGA 如 Xilinx XC6VLX760,内部带有 1 440 个独立的存储器,总容量达到约 26M 比特,可工作在 300MHz 时钟频率以上[22].因此,基于当前的硬件工艺水平,DALCA 可以很容易地支持 OC-768 链路最小报文的线速更新与查询.与 BRICK 相比,在硬件实现时,DALCA 使用的存储器位宽更窄,存储器带宽需求更低,从而有助于节省布线资源,降低芯片能耗.

6 结论与展望

本文提出了一种新型的准确计数的主动式计数器结构 DALCA.与现有的同类计数器 BRICK 相比,DALCA 的空间效率与之接近,但是在时间效率方面,DALCA 具备明显的优势,且 DALCA 实现方式比 BRICK 更为灵活.DALCA 可以在高速网络流量分析与流量管理领域中得到应用.

在下一步的研究工作中,可以对哈希表结构进行优化,从而进一步提高 DALCA 的性能.例如,已有的研究工作表明,若多级哈希表中块的长度逐步递减,则可以获得比采用等长块时更好的空间效率[11,16].下一步的研究工作还包括将 DALCA 的设计思想应用到相关算法的设计中去.例如,可以将 DALCA 的设计思想应用到 CB[6,7]中去,以改善 CB 解码时间较长这一缺陷[9].

References:

[1] Estan C, Varghese G. New directions in traffic measurement and accounting. ACM SIGCOMM Computer Communication Review, 2002,32(4):323−336. [doi: 10.1145/964725.633056]

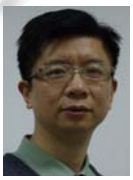
[2] Stanojevic R. Small active counters. In: Proc. of the IEEE Infocom. 2007. 2153−2161. [doi: 10.1109/INFCOM.2007.249]

[3] Cvetkovski A. An algorithm for approximate counting using limited memory resources. In: Proc. of the ACM Sigmetrics. 2007. 181−190. [doi: 10.1145/1254882.1254903]
[4] Hu C, Liu B, Chen K. Discount counting. In: Proc. of the IEEE ICNP. 2009.
[5] Tsidon E, Hanniel I, Keslassy I. Estimators also need shared values to grow together. In: Proc. of the IEEE Infocom. 2012. [doi: 10.1109/INFCOM.2012.6195564]
[6] Lu Y, Montanari A, Prabhakar B, Dharmapurikar S, Kabbani A. Counter braids: A novel counter architecture for per-flow measurement. In: Proc. of the ACM SIGMETRICS. 2008. 121−132. [doi: 10.1145/1375457.1375472]
[7] Lu Y, Prabhakar B. Robust counting via counter braids: An error-resilient network measurement architecture. In: Proc. of the IEEE Infocom. 2009. 522−530. [doi: 10.1109/INFCOM.2009.5061958]
[8] Zhao Q, Xu J, Liu Z. Design of a novel statistics counter architecture with optimal space and time efficiency. In: Proc. of the ACM SIGMETRICS. 2006. 323−334. [doi: 10.1145/1140103.1140314]
[9] Hua N, Xu J, Lin B, Zhao HQ. BRICK: A novel exact active statistics counter architecture. IEEE/ACM Trans. on Networking, 2011,19(3):670−682. [doi: 10.1109/TNET.2011.2111461]
[10] Mitzenmacher M. Studying balanced allocations with differential equations. Combinatorics, Probability and Computing, 1999,8(5): 473−482. [doi: 10.1017/S0963548399003946]
[11] Kirsch A, Mitzenmacher M. The power of one move: Hashing schemes for hardware. IEEE/ACM Trans. on Networking (TON), 2010,18(6):1752−1765. [doi: 10.1109/TNET.2010.2047868]
[12] Kumar A, Sung M, Xu J, Wang J. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In: Proc. of the ACM SIGMETRICS. 2004. 177−188. [doi: 10.1145/1005686.1005709]
[13] Zhao Q, Kumar A, Wang J, Xu J. Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. In: Proc. of the ACM SIGMETRICS. 2005. 350−361. [doi: 10.1145/1064212.1064258]
[14] Kumar A, Xu J. Sketch guided sampling—Using on-line estimates of flow size for adaptive data collection. In: Proc. of the IEEE Infocom. 2006. 467−482. [doi: 10.1109/INFOCOM.2006.326]
[15] Hu C, Tang Y, Chen K, Liu B. Dynamic queuing sharing mechanism for per-flow quality of service control. IET Communication, 2010,4(4):472−483. [doi: 10.1049/iet-com.2009.0404]
[16] Kanizo Y, Hay D, Keslassy I. Optimal fast Hashing. In: Proc. of the IEEE Infocom. 2009. 2500−2508. [doi: 10.1109/INFCOM.2009.5062178]
[17] Li T, Chen SG, Ling YB. Fast and compact per-flow traffic measurement through randomized counter sharing. In: Proc. of the IEEE Infocom. 2011. 1799−1807.
[18] Intel Corporation. IXP 2800 Hardware Reference Manual. 2004.
[19] Cheng G, Gong J, Ding W, Xu JL. A Hash algorithm for IP flow measurement. Ruan Jian Xue Bao/Journal of Software, 2005, 16(5):652−658 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/652.html [doi: 10.1360/jos160652]
[20] Ramakrishna M, Fu E, Bahcekapili E. Efficient hardware hashing functions for high performance computers. IEEE Trans. on Computers, 1997,46(12):1378−1381. [doi: 10.1109/12.641938]
[21] Kaya I, Kocak T. A low power lookup technique for multi-hashing network applications. In: Proc. of the IEEE Computer Society Annual Symp. on Emerging VLSI Technologies and Architectures. 2006. 179−184. [doi: 10.1109/ISVLSI.2006.3]
[22] Xilinx Corporation. Virtex 6 family overview. 2010.

附中文参考文献:

[19] 程光,龚俭,丁伟,徐加羚.面向 IP 流测量的哈希算法研究.软件学报,2005,16(5):652−658. http://www.jos.org.cn/1000-9825/16/652.html [doi: 10.1360/jos160652]

张进(1979－),男,江苏镇江人,博士,工程师,主要研究领域为宽带信息网络,无线网络.
E-mail: boost_zj@163.com



赵文栋(1972－),男,副教授,主要研究领域为 P2P 网络,分布式信息系统.
E-mail: nj_mouse@163.com

黄清杉(1986－),男,硕士,主要研究领域为宽带信息网络,网络测量.
E-mail: 13390793739@163.com

彭来献(1978－),男,博士,副教授,主要研究领域为宽带信息网络,无线网络.
E-mail: lx_peng@sina.com