流数据近似统计算法研究*)

聂国梁[1] 卢正鼎[1] 聂国栋[2]

(华中科技大学计算机科学与技术学院 武汉430074)[1] (山东省建行 济南250012)[2]

摘 要 流数据的统计是许多决策支持系统的关键所在。研究了流数据的分布特点,定义了评价函数 F,设计了一种系统框架,扩展了指数级直方图,提出了松散性指数级直方图及其动态维护算法,基于滑动窗口技术解决了流数据的统计问题。该方案利用 $o(\frac{1}{\varepsilon}\log^2 N)$ 比特的空间,解决了流数据最近 N 个数据中值为1的个数统计问题,并保证相对误差不大于 ε。理论和实践表明,F 值越大,其优势越明显。

关键词 流数据,滑动窗口,近似算法,流数据算法

The Study of Approximate Statistics Algorithm of Data Stream

NIE Guo-Liang[1] LU Zheng-Ding[1] NIE Guo-Dong[2]

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

(China Construction Bank Shandong Branch, Jinan 250012)[2]

Abstract The statistics of data stream plays an important role in many data stream's decision support systems. After studying the distribution character of data stream, this paper defines the evaluation function F, and designs a system framework based on a novel data structure called loose exponential histogram (LEH) to estimate the statistics of data stream over sliding window. Using $o(\frac{1}{\varepsilon}\log^2 N)$ bits of memory, the framework can estimate the number of 1's in data stream over sliding window of size N with relative error between 0 and ε. Theory and experiments prove that the greater the value of F is, the better the performance of the framework is.

Keywords Data stream, Sliding window, Approximation algorithms, Data stream algorithms

1 引言

随着互连网的发展,越来越多的数据以流的形式出现,如网络中的数据包,网页的点击记录等等,这些数据的共同特点是:流量巨大,不可复现。这种类型的数据,称为流数据。流数据的统计,在许多决策系统中起着重要作用,如网络监控、网页的点击统计等等。而传统的统计算法,却不能应用与流数据的统计。因此,研究流数据的统计算法具有重要的理论意义和实用价值。

国内外学者对流数据的统计进行了大量的研究:Alon 在理论上证明了流数据频率统计算法的最小空间复杂度[1]。文[2,3]分别提出了两种适应流数据频率统计的采样算法。Ananthakrishna 利用分位数解决了流数据的统计问题[4]。Gilbert 基于小波技术对流数据的统计问题提出了解决方案[5]。Guha 研究了直方图并将其扩展到可以解决流数据的统计问题[6]。Datar 提出了指数级直方图,解决了流数据的统计问题[7]。Gibbons 提出了波浪技术,并将其应用到流数据的统计问题[8]。

但是,以上算法都没有考虑流数据的分布特点。本文在充分考虑了流数据的分布特点的基础上,设计了一种系统框架,提出了一类新的指数级直方图,结合滑动窗口[7],解决了流数据的统计问题。

2 理论基础

2.1 流数据

流数据 S 定义如下:

$S::=\{A_i \mid A_{i-1}<_t A_i, i\in(1..\infty)\}$。

$A::=(data, timestamp)$。

$data::=$ 数据值,用1个 bit 表示。

$timestamp::=$时间戳,代表数据到来顺序,用 bits 表示。

$<_t::=$时间戳序(小于)。

流数据全部保存在存贮介质上是行不通的,因此通常都采用滑动窗口来处理流数据,滑动窗口以外的数据被认为是无效的,而数据是否有效决定与其时间戳是否过期。时间戳 T 是否过期检测方法如下:如果 $\max(timestamp)-T+1-N<0$(N:滑动窗口尺寸),则 T 过期;否则,T 不过期。

为了描述流数据的分布特点,定义流数据的相似度 F 如下:

$$F=\sum_{i=1}^{\infty}(A_i.data\otimes A_{i+1}.data)。$$

F 越大,流数据中的邻接个体数据之间的相似性越好。

2.2 问题描述

要解决的流数据统计问题描述如下:

流数据类型为 bit,滑动窗口尺寸为 N(N 是正整数),要求能随时求得该窗口中 bit 值为1的数据个数,并且保证相对误差不大于 ε(任意小的正实数)。

2.3 系统框架

基于流数据的相似度,本文设计了一种解决流数据统计问题的系统框架(图1)。

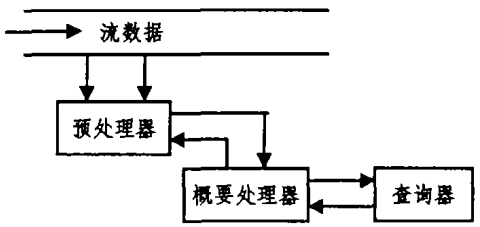

图1 系统框架

聂国梁 博士研究生,主要研究数据库技术和流数据算法。卢正鼎 教授,博士生导师,从事分布异构系统和数据库技术研究。

该系统框架包括预处理器、概要处理器和查询器三个主要模块。

·预处理器，在流数据与概要处理器之间起着桥梁作用。它顺序读取到来的流数据并对其进行累计缓存，一旦流数据值发生变化或者累计数目达到窗口尺寸 N，则把累计缓存的结果传送给概要处理器。预处理器包含一个结构为(value, count, timestamp)的存贮单元 Buf，用来缓存累计的流数据。

·概要处理器，利用一个直方图结构来存放预处理器传送过来的流数据。它接收查询器送来的统计查询命令，对直方图进行计算，把得到的结果送给查询器，以供用户查看。

·查询器，为用户提供一个接口来输入查询命令。查询器把查询命令发送给概要处理器，然后接收概要处理器的回传结果，显示给用户。

2.4 松散的指数级直方图

Datar 为解决流数据的统计，提出了指数级直方图(Exponential Histogram，简称 EH)[7]。指数级直方图分若干层，每层最多包含 $K+1$个桶，$K=\lceil \frac{1}{\varepsilon} \rceil$。第 i 层的桶的容量为2^i，i 为非负整数。当第 $i+1$层不空时，第 i 层最少有 K 个桶。它利用 $o(\frac{1}{\varepsilon}\log^2 N)$比特的空间，估计出尺寸为 N 的滑动窗口中 *bit* 值为1的流数据个数，并保证其相对误差不大于 ε。

指数级直方图解决了流数据的统计问题，但是它有如下局限性：每层上的桶容量大小固定，且当上一层非空时，本层桶的个数不能少于 K。这些局限性增大了指数级直方图的空间耗费，限制了其可接受数据的范围。

为避免指数级直方图的局限性，减少空间耗费和扩大可接受数据的范围，本文提出了松散性指数级直方图(Loose Exponential Histogram，简称 LEH)。定义如下：

$Bucket::=(timestamp, size)$。

$timestamp::=$时间戳，代表数据到来顺序，用 bits 表示。

$size::=$容量，代表桶中包含的值为1的数据个数，大小为 $2^i, i\in(0..\log N)$。

$Band::=\{Band_i \mid Bandi$ 非 $\Phi \wedge Band_{i-1} <_t Band_i, i\in(0..\log N-1)\}$。

$Band_{i-1} <_t Band_i ::=$当且仅当 $\sum_{Bucket\in Band_{i-1}} Bucket.size \geqslant K*2^{i-1}$，$Band_i$ 才可能非空，$i\in(0..\log N)$。

$Band_i::=\{Bucket_j \mid Bucket_{j-1} <_t Bucket_j \wedge Bucket_j.size \geqslant 2^i, j\in(1..K+2)\}$。

$<_t::=$ 时间戳序(小于)。

2.4.1 名词术语

$Count_i=\sum_{bucket\in Band_i} size$。

LastBucket 为 LEH 中 *timestamp* 最小的桶。

$B_{i,j}$代表 $Band_i$ 中第 j 个 *Bucket*。

$C_{i,j}$代表 $B_{i,j}.size$。

$T_{i,j}$代表 $B_{i,j}.timestamp$。

2.5 LEH 的有效性证明

下面证明松散性指数级直方图的有效性。

证明：假定 *LastBucket* 为 $B_{i,j}$，用 E_a 和 E_r 分别表示绝对误差和相对误差。i 分两种情况：

(1)$i=0$，即桶没有经过合并。如果 $C_{i,j}=1$，则 $E_a=0, E_r=0$，满足问题2.2；如果 $C_{i,j}>1$，则表示 $B_{i,j}$中包含的数据是连续的，利用 $C_{i,j}$和 $T_{i,j}$，能确定 $B_{i,j}$中包含的有效数据的数目，所以 $E_a=0, E_r=0$，满足问题2.2。

(2)$i>0$，如果 $C_{i,j}=2^i$，当计算统计结果时，取 $C_{i,j}=2^{i-1}$，则 $E_a\leqslant 2^{i-1}$；如果 $C_{i,j}>2^i$，维护算法保证最多有2^i-1个数据是合并来的，$C_{i,j}-2^i+1$个数据是连续的，依据 $T_{i,j}$，可以求得 $W=\max\{w*2^i \mid w*2^i+2^i\leqslant C_{i,j}\wedge T_{i,j}+w*2^i$ 不过期，w 为非负整数$\}$，当计算结果时，取 $C_{i,j}=W+2^{i-1}$，则 $E_a\leqslant 2^{i-1}$。$E_r\leqslant\frac{2^{i-1}}{\sum_{j=0}^{i-1} Count_j}$，即 $E_r\leqslant\frac{2^{i-1}}{k2^i-k}$，得 $E_r\leqslant\varepsilon$。所以松散性指数级直方图能满足问题2.2(证毕)。

3 算法

3.1 预处理算法

预处理器的主要功能是累计流数据，减少数据的插入次数。预处理算法描述如下：

预处理器接收到数据 V，分3种情况处理：

(1)如果 $Buf.Count$ 为0，则更新 Buf，算法结束。

(2)如果 $V.data$ 等于 $Buf.value$，则增加 $Buf.count$，更新 $Buf.timestamp$。如果 $Buf.count$ 为 N，则把 Buf 传给概要处理器，清空 Buf。算法结束。

(3)如果 $V.data$ 不等于 $Buf.value$，则把 Buf 传给概要处理器，然后更新 Buf。算法结束。

本文只考虑当有新数据到来时处理要花费的时间。当一个新的数据到达预处理器，平均要花费的处理时间为 $o(1)$。

3.2 LEH 动态维护算法

流数据连续到来，LEH 需要动态维护。维护算法描述如下：

step 1：接收到预处理器传来的数据 D，其结构为(*value*, *count*, *timestamp*)。不妨设 LastBucket 为 $B_{i,j}$，检查其有效性：如果 LastBucket 无效，则对其进行删除并使其指向下一个桶，更改 $Count_i$，继续检查 LastBucket，直到其有效或者所有的桶都无效为止。

step 2：如果 $D.value$ 为0，则算法结束。否则，生成新桶并插入到 $Band_0$中，然后修改 $Count_0$，i 初始化为0。

step 3：对 $Band_i$ 进行检测，j 初始化为0。如果 $count_i<(K+2)*2^i$，则算法结束。

step 4：求得 $Q=\max\{q \mid C_{i,j}=q*2^{i+1}+p*2^i \wedge Count_i-q*2^{i+1}\geqslant K*2^i, p$ 和 q 为非负整数$\}$，$P=(C_{i,j}-Q*2^{i+1})/2^i$。如果 $Q>0$，生成桶$(T_{i,j}+P*2^i, Q*2^{i+1})$并插入 $Band_{i+1}$。如果 $Count_i-Q*2^{i+1}<(K+2)*2^i$ 且 $P>0$，修改 $B_{i,j}$和 $Count_i$，i 加1，转步(3)；如果 $Count_i-Q*2^{i+1}<(K+2)*2^i$ 且 $P=0$，删除 $B_{i,j}$，修改 $Count_i$，i 加1，转 step 3；否则，删除 $B_{i,j}$且 $C_{i,j+1}$增加 $P*2^i$，修改 $Count_i$，j 加1，继续执行 step 4。

在概要处理器中，当新的数据要插入的时候，时间复杂度最坏情况下为 $o(\frac{1}{\varepsilon}\log N)$，平均为 $o(1)$。算法的空间复杂度为 $o(\frac{1}{\varepsilon}\log^2 N)$。

3.3 查询算法

概要处理器接收来自查询器的命令，执行查询算法。查询算法描述如下：

step 1：查询预处理器的 Buf。不失一般性，设 $Buf.value$ 为 V，$Buf.count$ 为 C。

step 2：设 LastBucket 为 $B_{i,j}$，检查其有效性：如果 LastBucket 无效，则对其进行删除并使其指向下一个桶，更改 $Count_i$，继续检查 LastBucket，直到其有效或者所有的桶都无

效为止。

Step 3：如果所有的桶都无效，则 $P=0$；否则设 Last-Bucket 为 $T_{i,j}$时有效，求得 $W=\max\{w*2^i|w*2^i+2^i\leqslant C_{i,j}\wedge T_{i,j}+w*2^i$ 不过期，w 为非负整数$\}$，使得 $P=C_{i,j}-W$。查询结果为 $\sum_{j=0}^{i}Count_j-P+\lfloor 2^{i-1}\rfloor+C*V$。算法结束。

查询算法的时间复杂度最坏情况下为 $o(\frac{1}{\varepsilon}\log N)$，平均为 $o(1)$。

4 实验

本文对 LEH 算法和 EH 算法[7]在空间耗费和时间耗费上进行了比较。实验环境为 CPU：Pentium1.6GHZ；内存：256MDDR；OS：Windows2000。对两种算法的空间耗费进行比较，如图2。

实线代表 EH 算法的空间耗费，虚线代表 LEH 算法的空间耗费。显然，LEH 算法要节省桶的耗费。两者空间耗费的差别程度主要是由流数据的相似度决定的。

本文用流数据的插入次数来表示两种算法的时间耗费。EH 算法接收到新数据立即插入，所以其插入次数等于窗口尺寸 N[7]。而 LEH 算法在充分考虑了流数据相似度的基础上对数据进行了累计处理，因此大大减少了插入次数。表1给出 LEH 算法在不同的窗口尺寸下的插入次数。

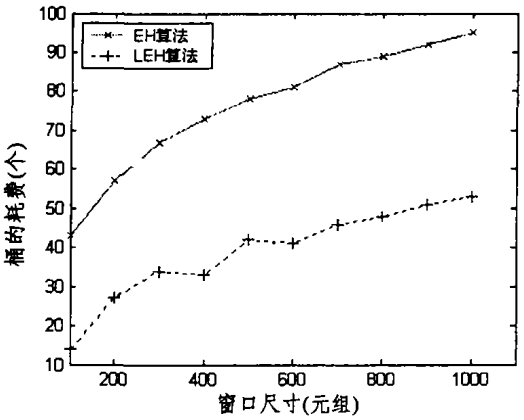

图2 EH 算法和 LEH 算法比较

表1 LEH 算法插入次数表

窗口尺寸(元组)	100	200	300	400	500	600	700	800	900	1000
插入次数(次)	20	41	65	87	113	130	145	167	189	211

显然，在相同尺寸的滑动窗口下，LEH 算法的数据插入次数比 EH 算法的插入次数少得多。差别程度依赖于流数据的相似度。相似度越大，LEH 的插入次数越少。

流数据的相似度越大，LEH 的优越性就越明显。图3描绘了针对不同相似度的流数据，LEH 具有不同的空间耗费。

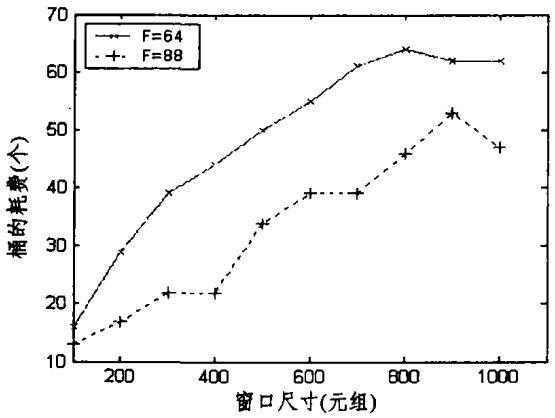

图3 相似度与 LEH 算法空间耗费的关系

$F=64$表示每100个数据 F 为64。图3显示：统计结果一样的流数据，其相似度越大，LEH 算法的空间耗费就越少。根据 F 的定义和预处理器的算法，可以得到：相似度越大，流数据插入概要处理器的次数就越少。

结论 本文充分考虑了流数据的相似度，设计了一种满足流数据统计查询的系统框架。对指数级直方图进行了优化扩展，提出了松散性指数级直方图，并针对这种直方图提出了一种流数据统计算法。相比文[7]中算法，该算法充分考虑了数据的分布特点，实验表明，时空优势都比较明显。当然，流数据尚有许多问题需进一步解决，如分布式连接查询及优化，适应性查询，数值变化最大的 K-top 数据的查询，group-by 在流数据中的应用等等，这也是下一步的研究课题。

参考文献

1 Alon N, Matias Y, Szegedy M. The space complexity of approximating the frequency moments. Journal of Computer and System Sciences, 1999, 58(1): 137～147

2 Gibbons P B, Matias Y. New sampling-based summary statistics for improving approximate query answers. In: proc. of the 1998 ACM SIGMOD, Seattle, 1998

3 Manku G S, Motwani R. Approximate frequency counts over data streams. In: Proc. of the 28th Intl. Conf. on Very Large Data Bases, Hong Kong, 2002

4 Ananthakrishna R, das A, et al. Efficient Approximation of Correlated Sums on Data Streams. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(3): 569～572

5 Gilbert A C, Kotidis Y. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. The VLDB Journal, 2001. 79～88

6 Guha S, Koudas N, Shim K. Data-Streams and Histograms. In: Proc. of the 33th annual ACM symposium on Theory of computing, Hersonissos, 2001

7 Datar M, Gionis A, Indyk P, Motwani R. Maintaining Stream Statistics over Sliding Windows. In: proc. of the thirteenth annual acm-siam symposium on discrete algorithms, San Francisco, 2002

8 Gibbons P, Tirthapura S. Distributed Streams Algorithms for Sliding Windows. In: Proc. of ACM Symposium on Parallelism in Algorithms and Architectures, Winnipeg, Manitoba, 2002