# 数据流历史数据的存储与聚集查询处理算法*

张冬冬[1+], 李建中[1,2], 王伟平[1], 郭龙江[1,2]

[1](哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

[2](黑龙江大学 计算机科学与技术学院,黑龙江 哈尔滨 150080)

## Algorithms for Storing and Aggregating Historical Streaming Data

ZHANG Dong-Dong[1+], LI Jian-Zhong[1,2], WANG Wei-Ping[1], GUO Long-Jiang[1,2]

[1](School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

[2](School of Computer Science and Technology, Heilongjiang University, Harbin 150080, China)

+ Corresponding author: Phn: +86-451-86415872, E-mail: zddhit@hit.edu.cn, http://db.cs.hit.edu.cn

Received 2004-04-24; Accepted 2005-04-01

**Zhang DD, Li JZ, Wang WP, Guo LJ. Algorithms for storing and aggregating historical streaming data. *Journal of Software*, 2005,16(12):2089−2098.** DOI: 10.1360/jos162089

**Abstract**: The current research work over data streams is mainly focused on dealing with the arrival of recent data in memory, neglecting the analysis and management of historical streaming data. An approach is proposed to store and query historical streaming data by using multi-layer recursive sampling method and HDS-Tree structure, which indexes the aggregation of historical streaming data and supports all kinds of aggregation queries over historical streaming data. The time-complexity and the error of aggregation algorithms are also analyzed based on HDS-Tree. The analytical and experimental results show that the approach can be effectively used to store and analyze the historical streaming data.

**Key words**: data streams; historical data; aggregation algorithm; HDS-Tree

摘 要: 目前数据流的研究成果主要集中在分析处理存储于内存中的最近一段时间内的数据流数据,忽略了对数据流历史数据的分析处理与存储管理.提出了一种数据流历史数据的存储管理及聚集查询处理方法,通过对历史数据实施多层递阶抽样存储,并在内存中建立存储数据流历史数据聚集值的 HDS-Tree 索引,实现对无限数据流历史数据的存储管理,有效地支持各种聚集查询.同时,还给出了基于 HDS-Tree 的聚集查询算法的时间复杂性分析和查询误差分析.理论分析与实验结果表明,该方法可以有效地用于数据流历史数据的存储与分析.

关键词: 数据流;历史数据;聚集算法;HDS-Tree

* Supported by the National Natural Science Foundation of China under Grant No.60273082 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA444110 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032704 (国家重点基础研究发展规划(973)); the Natural Science Foundation of Heilongjiang Province of China under Grant No.zjg03-05 (黑龙江省自然科学基金)

作者简介: 张冬冬(1976−),男,内蒙古呼和浩特人,博士生,主要研究领域为数据流处理,海量数据管理;李建中(1950−),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,并行计算;王伟平(1975−),男,博士生,主要研究领域为数据流处理;郭龙江(1973−),男,讲师,主要研究领域为数据流挖掘.

中图法分类号: TP311 文献标识码: A

近年来,由于数据流的应用领域越来越广泛,数据流的研究引起了人们的极大兴趣.数据流不同于存储在磁盘上的传统关系数据,而是以快速、无限、连续的流的形式存在.典型的数据流包括无线传感器网络应用环境中由传感器传回的各种监测数据、股票交易所的股票价格信息数据、网络监测系统与道路交通监测系统的监测数据、电信部门的通话记录数据以及 Web 上的数据等[1,2].管理和处理数据流的系统称为数据流系统.数据流可以被视为无限多重集合,集合中每个元素具有形式$\langle s,t\rangle$,其中 $s$ 是一个元组,$t$ 为标识 $s$ 的时间戳,$t$ 的取值可以是 $s$ 进入数据流系统的时间或者数据源产生 $s$ 的时间[3].由于内存资源有限,数据流系统只能通过在存储器中开辟滑动窗口来保存近期到达的数据流数据,以实时地支持查询请求.随着数据流源源不断地流入滑动窗口,必然有部分旧数据从滑动窗口中流出.我们称流出滑动窗口的数据为历史数据.

数据流管理技术已经成为一个热点研究问题.目前的研究工作主要集中在分析处理近期数据流数据上,而忽略了对数据流历史数据的分析处理与存储管理[4−7].在实际应用中,存储器的容量是有限的,而源源不断到达的数据流是无限的,因此存储器中只能存储一段时间内的数据流数据,以实时地支持查询要求.然而,当查询请求所涉及到的数据查询范围超出了存储器中存储的数据,或者查询请求需要直接访问历史数据时,已有的数据流查询技术就无能为力了.

对数据流历史数据的查询请求是普遍存在的,例如,网站内容维护人员需要查询历史日志信息以跟踪了解某些用户群体的个性化点击信息,确保网页上的内容能够更大程度地满足这些用户的需求.因此,数据流的历史数据的存储与查询是一个值得研究、关注的问题.那么,是否可以用传统 DBMS 的关系表来存储管理这些数据流中的历史数据呢?答案是否定的,原因是:(1) 数据流具有无限的连续性.随着时间的推移,数据流历史数据的容量可以无限大,传统 DBMS 难以存储管理如此之大容量的数据.(2) 多数情况下,为满足实时性,只需得到一定误差范围内的近似结果即可满足数据流系统中的查询请求,而传统的 DBMS 很难支持任何近似查询操作.(3) 对于一些聚集查询请求,传统的 DBMS 需要扫描外存中的关系表数据来得到查询结果,这个过程会产生许多 I/O 操作,不满足数据流系统中的实时查询要求.

显然,传统的数据库技术不能用于存储与分析持续快速增长的数据流历史数据,需要研究与数据流特点相适应的数据流历史数据的管理技术.Chen 等人[8]提出在内存或者外存中存储部分实体化的 CUBE 结构来管理数据流,但是该方法不能很好地支持数据流上的查询.Zhang 等人[9]从传统的时空数据库技术角度出发研究了基于 SB-tree 结构的数据流多粒度聚集问题,提出对历史久远的数据采取粗粒度(如以年为单位)的聚集值存储方式,对于相对较近的数据采取细粒度(如以分钟为单位)的聚集值存储方式,但这种方法存在的问题是:(1) 丢弃了原始历史数据,不能支持分组聚集查询和基于任意时间段的聚集查询.(2) SB-tree 结构中的部分结点存在空闲空间,导致存储空间的利用率不高.(3) 每个新到达的数据都需要被及时地插入 SB-tree 结构,不适合于处理流速快的数据流.

本文提出一种支持数据流历史数据的存储以及查询的方法.我们通过对历史数据实施两种类型的抽样存储(分别称为初始抽样过程和多层递阶抽样过程)实现历史数据的存储.初始抽样过程对近期流出滑动窗口的历史数据实施抽样,将抽样后得到的样本存储在外存中,以减少历史数据对外存空间的占有量.多层递阶抽样过程对外存中的样本实施持续的再抽样存储,进一步减小样本数据对外存空间的存储压力.此外,为支持对数据流历史数据的查询,我们还在内存中建立一个历史数据聚集值的索引结构,称为 HDS-Tree.使用这种历史数据存储方法和 HDS-Tree,可以有效地支持历史数据的查询与分析.

## 1 历史数据的抽样存储方法

我们对数据流历史数据采取抽样存储与聚集值存储相结合的管理方式,以支持对数据流历史数据的各种聚集查询.数据流按照时间戳的先后顺序进入数据流系统[1],因此数据流的历史数据也按照时间先后顺序产生.本文假设时间戳根据数据流进入系统的时间确定.由于我们只考虑离散时间,因此每一个时间戳可以映射成一

个整数,从而可以把一组具有先后顺序的时间戳看作是一个从小到大排列的整数序列[8],称为时间戳序列.

定义 **1**(时间段). 给定两个时间戳 $t_s$ 和 $t_e$,且 $t_s \le t_e$,则$[t_s,t_e]$构成一个时间段(记为 $T$),$t_s$ 和 $t_e$ 分别称为 $T$ 的下界和上界,$t_e-t_s$ 的值称为 $T$ 的跨度,记为$|T|$.

设有两个时间段 $T_1=[t_{s_1},t_{e_1}]$,$T_2=[t_{s_2},t_{e_2}]$,如果 $t_{s_1}=t_{s_2}$ 且 $t_{e_1}=t_{e_2}$,则认为 $T_1$ 与 $T_2$ 相等;如果 $t_{e_1}=t_{s_2}$,则称 $T_3=[t_{s_1},t_{e_2}]$ 为 $T_1$ 与 $T_2$ 之和,记为 $T_3=T_1+T_2$,并且称 $T_1$ 与 $T_2$ 是可相加的;如果 $t_{s_1}\ge t_{e_2}$ 或者 $t_{e_1}\le t_{s_2}$,则称时间段 $T_1$ 与 $T_2$ 不相交,记为 $T_1 \cap T_2 = \varnothing$;如果 $t_{s_1}\le t_{s_2}$,且 $t_{e_1}\ge t_{e_2}$,则认为 $T_1$ 覆盖了 $T_2$,记为 $T_1 \supseteq T_2$ .

定义 **2** (时间段的段划分点集). 设有时间段 $T=[t_s,t_e]$,时间戳序列 $t_1,\ldots t_i,\ldots,t_m$.如果$\forall i,1\le i\le m,t_s<t_i<t_e$ 成立,则此时间戳序列可把 $T$ 划分成 $m+1$ 个子时间段$[t_s,t_1],[t_1,t_2],[t_2,t_3],\ldots,[t_{m-1},t_m],[t_m,t_e]$,即 $T=[t_s,t_1]+[t_m,t_e]+\sum_{i=1}^{m-1}[t_i,t_{i+1}]$.我们称此时间戳序列为 $T$ 的段划分点集,记为$\langle t_1,\ldots t_i,\ldots,t_m\rangle$,用 $TD$ 表示.

对于任意时间段 $T=[t_s,t_e]$,在 $t_s$ 时刻与 $t_e$ 时刻之间从滑动窗口流出的历史数据称为 $T$ 的原始数据.对 $T$ 的原始数据抽样后形成的数据集合称为 $T$ 的样本.$t_s$ 和 $t_e$ 分别称为该样本的下界时间戳和上界时间戳.$T$ 的样本容量与 $T$ 的原始数据容量的比值称为 $T$ 的抽样比例(或者历史数据的抽样比例),记为 $f$,$0\le f\le 1$.类似地,将数据集抽样前和抽样后大小的比值称为抽样系数.

历史数据的抽样存储分为初始抽样过程和多层递阶抽样过程.初始抽样发生在产生历史数据的初期进行,对流出滑动窗口的最近一段时间内的历史数据进行随机抽样,并定期将形成的样本写入外存.多层递阶抽样对外存中的样本数据实施进一步的抽样存储.初始抽样过程的目的是减小近期产生的历史数据对外存空间的占有量,而多层递阶抽样过程的目的是减小外存中的样本对外存空间的占有量.

### 1.1 初始抽样方法

对历史数据的初始抽样采用二项式随机抽样法[10].首先,选定一个未来时刻 $t_e$,与当前时刻 $t_s$ 构成一个时间段 $T_c=[t_s,t_e]$,$T_c$ 的跨度根据外存空间使用情况以及实际应用环境确定,比如 10 分钟、1 个小时或者 1 天.然后,选定一个正数 $p$,且 $0<p\le 100\cdot f_{\max}$,$f_{\max}$ 为数据流系统设定的最大抽样比例.随着时间的流逝,当每次从滑动窗口中流出一个数据时,在数值区间[0,100]之间生成一个随机数 $x$,如果 $x\le p$,则把该数据加入 $T_c$ 的样本,否则抛弃该历史数据.当系统时钟到达 $t_e$ 时,将 $T_c$ 的样本写入外存.不断重复上述过程,则从滑动窗口流出的历史数据会以不同时间段的样本形式被写入外存.在抽样过程中,同时对 $T_c$ 的原始数据执行聚集操作,如 SUM,COUNT,MAX 等.在把 $T_c=[t_s,t_e]$的样本写入外存的同时,需要记录该样本的相关信息,包括聚集值、抽样比例(记为 $f$)、外存中的存储地址(记为 $pd$)以及样本的下界时间戳 $t_s$ 和上界时间戳 $t_e$ 等.这些信息称为样本元数据.

在初始抽样过程中,对每个流出滑动窗口的历史数据的抽样都是一个独立事件,每个历史数据依概率 $p/100$ 加入样本.假设 $T_c$ 的原始数据容量为 $n$,则 $T_c$ 的样本容量的期望值为 $\sum_{i=1}^{n} p/100 = n\cdot p/100$ ,从而可知 $T_c$ 的抽样比例为$(n\cdot p/100)/n=p/100=p\%$.

初始抽样过程的抽样比例随着系统的负载情况动态变化.当系统负载重的时候(如外存资源紧张,数据流流速快等),减小 $p$ 的取值,降低抽样比例,以减轻系统的负载;当系统负载轻的时候,可以适当增大 $p$ 的取值,提高样本的抽样比例,以提高查询的精度.

初始抽样过程中同一个样本的抽样比例与抽样系数相等.

### 1.2 多层递阶抽样方法

随着初始样本数据量的增加,外存空间将被耗尽.因此,我们需要对外存中存储的样本实施多层递阶抽样存储,减轻外存的存储负担,实现在有限的外存空间中存储无限的数据流历史数据样本,并实现在外存中按照不同稀疏粒度存储历史数据的样本,即近期产生的历史数据的抽样比例大于久远历史数据的抽样比例,保证对近期历史数据的查询精度高于对久远历史数据的查询精度.多层递阶抽样是一个对外存中的样本进行迭代抽样的过程.下面是多层递阶抽样的基本思想(我们将在第 2.3 节详细给出多层递阶抽样算法描述):

(1) 在第 $i$ 次多层递阶抽样后,设外存中所有样本的最小下界时间戳为 $t_{old_i}$(稍后讨论 $t_{old_i}$ 的确定方法),最大

上界时间戳为 $t_{new_i}$(由多层递阶抽样时的系统时钟确定),则外存存储的样本集合为 $S_i=\{X_1^{(i)},X_2^{(i)},\ldots,X_m^{(i)}\}$,其中 $X_j^{(i)}$为时间段 $T_j^{(i)}$的样本,$1\le j\le m$.$T_k^{(i)}$与 $T_{k+1}^{(i)}$是可相加的,且 $\sum_{k=1}^{m}T_k^{(i)}=[t_{old_i},t_{new_i}]$.$T_j^{(i)}$的抽样比例由以下公式确定

$$f_j^{(i)}=\min\left\{e^{(t_{m_j}-t_{new_i})r/D},f_{\max}\right\} \tag{1}$$

其中 $r$ 为数据流的平均速率,$D$ 为外存的最大存储容量(元组个数),$t_{m_j}=(t_{s_j}+t_{e_j})/2$,$t_{s_j}$ 和 $t_{e_j}$ 分别为 $T_j^{(i)}$的下界和上界,$f_{\max}$ 为数据流系统设定的最大抽样比例.

(2) 设 $S_i=\{X_1^{(i)},X_2^{(i)},\ldots,X_p^{(i)}\}$为第 $i$ 次多层递阶抽样后外存中存储的样本集合,$\Delta S_i=\{X_{p+1}^{(i)},X_{p+2}^{(i)},\ldots,X_{p+r}^{(i)}\}$为初始抽样过程在第 $i$ 次和第 $i$+1 次多层递阶抽样期间写入外存的样本集合,$S_{i+1}=\{X_1^{(i+1)},X_2^{(i+1)},\ldots,X_q^{(i+1)}\}$为第 $i$+1 次多层递阶抽样后外存中的样本集合,$q\le p+r$,则 $S_{i+1}$ 的生成过程可由一个正整数 $C^{(i+1)}$决定,称为抽样因子.对于任意的 $k$,$1\le k\le q$,样本 $X_k^{(i+1)}$是由集合 $\{X_l^{(i)},X_{l+1}^{(i)},\ldots,X_h^{(i)}\}$中的样本分别经过随机抽样合并后形成,其中 $l=(k-1)\cdot C^{(i+1)}+1$,$h=\min\{p+r,k\cdot C^{(i+1)}\}$.易知,$\sum_{k=1}^{p}T_k^{(i)}+\sum_{k=p+1}^{p+r}T_k^{(i)}=\sum_{k=1}^{q}T_k^{(i+1)}$.

(3) 在不同的多层递阶抽样过程中,抽样因子的取值可以不同,且由用户根据实际情况指定,详见第 2.3 节.

在多层递阶抽样过程中,同一个样本的抽样比例要小于抽样系数.

定理 **1**. 多层递阶抽样过程能够实现在外存中存储全部样本数据.

证明:设在最近一次多层递阶抽样结束后外存中存储的样本集合为$\{X_1,X_2,\ldots,X_n\}$,其中 $X_i$ 为 $T_i=[t_{s_i},t_{e_i}]$的样本,令$\Delta t_i=t_{e_i}-t_{s_i}$,$t_{m_i}=(t_{s_i}+t_{e_i})/2$,则多层递阶抽样过程确定了 $T_i$ 的样本容量 $V_i$ 为 $V_i=r\cdot\Delta t_i\cdot f_i$.由式(1)可知,外存中存储的所有样本的容量总和为

$$\sum_{i=1}^{n}V_i=\sum_{i=1}^{n}r\cdot\Delta t_i\cdot e^{(t_{m_i}-t_{new})r/D}\approx\int_{t_{old}}^{t_{new}}r\cdot e^{(t-t_{new})\cdot r/D}\cdot dt=D\cdot\left(1-e^{(t_{old}-t_{new})r/D}\right)=D\cdot\left(1-\frac{1}{e^{(t_{new}-t_{old})\cdot r/D}}\right),$$

其中 $t_{old}$ 和 $t_{new}$ 分别为外存中所有样本的最小下界时间戳和最大上界时间戳.易见 $D\cdot\left(1-\frac{1}{e^{(t_{new}-t_{old})\cdot r/D}}\right)<D$ 且 $\lim_{t_{new}\to+\infty}D\cdot\left(1-\frac{1}{e^{(t_{new}-t_{old})\cdot r/D}}\right)=D$,因此可知外存中可以存储全部样本数据. □

定理 1 说明了多层递阶抽样过程能够保证在未来的任何时刻,外存都可以存储全部样本数据.在实际应用中,存储年代久远的历史数据的意义不大.因此,在每次进行多层递阶抽样时,我们动态设定一个有效时间戳 $t_{old_i}$,当外存中久远历史数据的时间戳小于 $t_{old_i}$ 时,把它们从外存中废弃掉,以节省外存的可利用存储空间.$t_{old_i}$ 的取值可以采取两种方式得到:

(1) 人工干预设定 $t_{old_i}$;

(2) 设定最小有效抽样比例 $f_{\min}$,外存中的抽样比例小于 $f_{\min}$ 的久远历史数据可认为是无效数据.令 $e^{(t_{old_i}-t_{new_i})r/D}=f_{\min}$,可得 $t_{old_i}=t_{new_i}+\frac{D}{r}\ln f_{\min}$,其中 $t_{new_i}$ 为第 $i$ 次多层递阶抽样后外存中所有样本的最大上界时间戳.

定理 **2**. 对于任意的 $i$,如果第 $i$ 次和第 $i$+1 次多层递阶抽样的最大时间间隔为 $\frac{D\cdot e^{(t_{old_i}-t_{new_i})r/D}}{r\cdot f_{\max}}$,则外存空间不会产生样本数据溢出(证略).

需要补充说明的是,多层递阶抽样过程维护的历史时间段的抽样比例可以有多种不同方式,式(1)只是其中的一种.任何形式的历史时间段的抽样比例,都应由外存空间大小、数据流流速和数据历史的久远程度等参数来决定,前提条件是要保证当前时刻外存能够存储全部样本数据,并且保证最近产生的历史数据的抽样比例大于比较久远的历史数据的抽样比例,符合实际应用系统的查询需求.

## 2 支持历史数据聚集操作的 HDS-Tree 结构

为了实时地支持对历史数据的聚集操作,我们将样本元数据(包括样本的聚集值、抽样比例 $f$、外存地址 $pd$ 以及样本的下界时间戳 $t_s$ 和上界时间戳 $t_e$ 等)组织起来形成内存存储结构 HDS-Tree.本节主要介绍 HDS-Tree

的结构及相关操作.初始抽样过程不断地产生新的样本元数据,而多层递阶抽样过程不断地更新旧的样本元数据,因此,上述两个抽样过程都会引起 HDS-Tree 结构的变化.

### 2.1 HDS-Tree的定义

HDS-Tree(historical data of stream-tree)是一棵最大扇出度为 $b$ 的有向树,其存储结构如下所述:

(1) 根结点与内结点类似,每个结点对应一个时间段 $T$,存储的信息为($aggV$,$P_1$,$Key_1$,$P_2$,...,$P_{q-1}$,$Key_{q-1}$,$P_q$),其中 $aggV$ 为样本聚集值集合,如 SUM,MAX,MIN 等,$q\le b$,$Key_i$ 是关键字,$Key_1<Key_2<\ldots<Key_{q-1}$,<$Key_1$,$Key_2$,…,$Key_{q-1}$>是 $T$ 的一个段划分点集.$P_i$ 是指针,指向的子结点对应的时间段为[$Key_{i-1}$,$Key_i$].

(2) 每个叶结点存储时间段[$t_s$,$t_e$]内的样本数据 $S$ 的元数据($aggV$,$f$,$pd$,$t_s$,$t_e$)以及指向右边相邻叶结点的指针 $pr$,其中 $aggV$ 是聚集值集合,$f$ 是抽样比例,$pd$ 是 $S$ 所在的外存地址.

HDS-Tree 具有以下性质:

(1) 树中的每个结点对应的时间段的下界和上界分别由父结点中的关键字决定(根结点对应的时间段由外存中所有样本的最小下界时间戳和最大上界时间戳决定).

(2) 所有叶结点处于同一层,且叶结点中任何两个通过 $pr$ 指针相邻的叶结点对应的时间段是可相加的.

(3) 任何一层的所有结点对应的时间段之和相同;根结点或任何一个内结点对应的时间段等于其所有后代结点对应的时间段之和.

(4) 除了根结点最右儿子的所有后代结点(包括最右儿子结点)之外,所有内结点是满的.

### 2.2 HDS-Tree与初始抽样

在初始抽样过程中,每当将样本写入外存时,需要把它的样本元数据存储在 HDS-Tree 中.由于样本元数据是按时间顺序产生的,因此样本元数据以增量追加方式插入 HDS-Tree 中.设 HDS-Tree 的高度为 $H$,根结点指针为 $R$,则 HDS-Tree 的插入过程具体描述为:首先,从 HDS-Tree 的根结点开始自顶向下搜索 $H$–1 个结点之后到达结点 $N_f$,形成搜索路径 $N_1N_2...N_{H-1}$,并更新此路径上每个结点的聚集值,其中 $N_1=R$,$N_{H-1}=N_f$,$N_{i+1}$ 为 $N_i$ 的最右儿子(即 $N_{i+1}=N_i\rightarrow P_q$,且 $Key_q$ 为 $N_i$ 中的最大关键字,如果 $N_i\rightarrow P_q$ 为空,则为其新创建一个结点).然后,创建一个叶结点 $I$,将样本元数据存入 $I$ 中,并将 $I$ 作为 $N_f$ 的一个子结点.最后,自底向上顺序访问结点 $N_{H-1}$,$N_{H-2}$,...,$N_1$,找到第 1 个未满结点(设为 $N_d$),将样本的上界时间戳 $t_e$ 作为关键字存入 $N_d$ 中,如果未找到 $N_d$,则新增加一个根结点 $R'$,将 $R$ 作为 $R'$的最左儿子,并将 $t_e$ 存入 $R'$中.

上述插入过程需要维护 HDS-Tree 的相关参数(称为 HDS-Tree 元数据),包括 HDS-Tree 的高度 $H$、根结点指针 $R$ 以及指向最右叶结点的指针 $P_n$.下面给出插入过程的定义:

输入:样本元数据($aggV$,$f$,$pd$,$t_s$,$t_e$)和 HDS-Tree 元数据($H$,$R$,$P_n$).

输出:插入样本元数据后的 HDS-Tree 结构.

Insert ($N$,$aggV$,$f$,$t_s$,$t_e$,$h$) /*$N$ 为当前访问结点,$h$ 为结点 $N$ 距离 HDS-Tree 的最底层叶结点的层数*/

(1) IF ($H$=0,即第 1 次创建 HDS-Tree 结点) THEN 初始化 HDS-Tree;

(2) ELSE IF ($h$!=2,即当前访问结点 $N$ 不是叶结点的上一层结点) THEN

(3) 利用二分法查找到 $N$ 中最大关键字对应的儿子结点指针 $P_c$;

(4) IF ($P_c$ 是一个空结点) THEN 申请一个内结点 $I$,初始化 $I$,令 $P_c$ 指向 $I$;

(5) $node$=Insert ($P_c$,$aggV$,$f$,$t_s$,$t_e$,$h$–1); /*递归调用该算法*/

(6) $N\rightarrow aggV$=accumulate($N\rightarrow aggV$,$aggV$); /*重新计算当前访问结点保存的聚集值*/

(7) IF ($node$ 不是一个空结点) THEN 返回 $N$; /*说明上面的第(5)步已成功插入关键字 $t_e$*/

(8) ELSE /*此时 $h$=2,即当前访问结点 $N$ 恰好是叶结点的上一层结点*/

(9) 建立一个叶结点 $I$,将样本元数据存入 $I$,令 $P_n$ 指向的叶结点中的 $pr$ 指向叶结点 $I$,并更新 $P_n$;

(10) 按顺序把叶结点 $I$ 作为 $N$ 的一个儿子,$N\rightarrow aggV$=accumulate($N\rightarrow aggV$,$aggV$);

(11) IF ($N$ 未满) THEN 把 $t_e$ 作为一个关键字存入 $N$,返回 $N$; /*在结点 $N$ 中存储样本的上界时间戳 $t_e$*/

(12) ELSE IF ($N$ 是根结点) /*此时结点 $N$ 已满*/

(13) 新建立一个根结点 $N'$,把 $N$ 作为 $N'$的最左儿子,更新 HDS-Tree 元数据中 $R$ 与 $H$ 的值;

(14) 把 $t_e$ 作为一个关键字存入 $N'$,$N'\rightarrow aggV$=accumulate($N'\rightarrow aggV$,$aggV$);

(15) 返回 $N'$;

(16) ELSE 返回一个空结点指针; /*说明结点 $N$ 已满,没有在当前结点成功插入关键字 $t_e$,返回*/ □

算法中的 accumulate($x$,$y$)函数定义如下:

(a) 如果聚集操作是 SUM 或者 COUNT,则 accumulate($x$,$y$)的实现方式为 $x=x+y$;

(b) 如果聚集操作为 MAX,则 accumulate($x$,$y$)的实现方式为 $x$=max($x$,$y$);

(c) 如果聚集操作为 MIN,则 accumulate($x$,$y$)的实现方式为 $x$=min($x$,$y$);

(d) 如果聚集操作为 AVG,则根据 HDS-Tree 中存储的 SUM 与 COUNT 两种聚集操作求得 AVG 聚集值,accumulate($x$=($x_{sum}$,$x_{count}$),$y$=($y_{sum}$,$y_{count}$))的实现方式为 $x_{sum}=x_{sum}+y_{sum}$,$x_{count}=x_{count}+y_{count}$.

最坏情况下,该算法需要从根结点开始向下搜索 $H-1$ 层结点后建立叶结点,然后再从叶结点向上递归寻找未满的祖先结点,并把样本的上界时间戳 $t_e$ 作为一个关键字存入此祖先结点中,因此算法的最坏时间复杂性为

$$O\left(2\cdot\log_2 b\cdot\left(\lceil\log_b n\rceil+1\right)\right),$$

其中 $b$ 为结点的最大扇出度,是常量,$n$ 为 HDS-Tree 中的叶结点个数(即外存中存储的样本个数),$\lceil\log_b n\rceil+1$为 HDS-Tree 的高度,$O(\log_2 b)$为算法中语句(4)的时间复杂度.

定义 **3**(最优树). 设 $Tree$ 是任意一棵高度为 $h$ 的树,$h\geq 2$,$Tree$ 的结点个数为 $n$,每个结点的最大扇出度为 $b$ 且 $b>1$.若$(b^{h-1}-1)/(b-1)<n\leq(b^h-1)/(b-1)$成立,即 $h\leq\lceil\log_b[n(b-1)+1]\rceil$,称 $Tree$ 为最优树.

最优树在保持结点个数不变的情况下,高度最低.因此,最优树具有最高的查询效率.

定理 **3**. 初始抽样过程保持了 HDS-Tree 的最优性,即一棵最优 HDS-Tree 经过插入操作以后仍然是最优 HDS-Tree.

证明:设 $Tree$ 是一棵最优 HDS-Tree,$Tree'$是初始抽样过程中使用 HDS-Tree 的插入算法把一个样本元数据 $Meta$ 插入 $Tree$ 的结果 HDS-tree.由算法描述可知,当在 $Tree$ 中插入 $Meta$ 时,会在 $Tree$ 中的最底层结点位置上增加一个叶结点,而且插入算法是以追加方式把 $Meta$ 插入 $Tree$ 中后形成 $Tree'$.插入算法引起 $Tree$ 的结点变化始终发生在 $Tree$ 的最右路径上,而且可能会引起 $Tree$ 的高度增加一层.设 $Tree$ 的结点个数为 $n$,高度为 $h$,$Tree'$的结点个数 $n'$,高度为 $h'$.下面分两种情况证明 $Tree'$是最优树:

(1) 当 $h'=h$ 时.因为 $Tree$ 是最优树,所以$(b^{h-1}-1)/(b-1)<n\leq(b^h-1)/(b-1)$成立.由于插入算法仅引起 HDS-Tree 的结点个数增加,因此 $n'>n>(b^{h-1}-1)/(b-1)=(b^{h'-1}-1)/(b-1)$成立.

(2) 当 $h'=h+1$ 时.设 $Tree$ 的根结点为 $N$,$Tree'$的根结点为 $N'$,则由插入算法中语句(12)~(15)的描述可知,$Tree'$中的结点 $N$ 是根结点 $N'$的一个最左儿子,且 $N$ 是满的,因此子树 $N$ 为一棵满 $b$ 元树,即子树 $N$ 中的结点个数为 $(b^h-1)/(b-1)$,于是 $n'>(b^h-1)/(b-1)=(b^{h'-1}-1)/(b-1)$.

综合(1)和(2)可知,$n'>(b^{h'-1}-1)/(b-1)$成立.由于任何一棵高为 $h'$的树的结点个数必然小于高为 $h'$的满 $b$ 元树的结点个数,因此 $n'\leq(b^{h'}-1)/(b-1)$成立.于是 $Tree'$是优化树. □

### 2.3 HDS-Tree与多层递阶抽样

多层递阶抽样过程中会引起外存中样本容量和数量的变化.由于 HDS-Tree 中的一个叶结点对应外存中的一个样本,因此多层递阶抽样过程也会引起内存中 HDS-Tree 的内容和结构的变化.抽样因子是多层递阶抽样过程的一个重要参数,它的取值受 HDS-Tree 结构的影响.在第 $i$ 次抽样过程中,抽样因子 $C^{(i)}$的取值为 $b^{\lambda}$,其中 $b$ 为 HDS-Tree 的最大扇出度,$\lambda=1,2,\ldots,H$,$H$ 为 HDS-Tree 的高度,$\lambda$的取值由用户根据实际情况指定.

给定$\lambda$的大小,多层递阶抽样算法可以描述如下:

(1) 访问 HDS-Tree,找到所有高度为$\lambda$的子树 $N_1,N_2,\ldots,N_K$.

(2) 对于任意 $j$,$1\leq j\leq K$,将子树 $N_j$ 中所有叶结点对应的样本依次调入内存进行随机抽样形成新样本 $X_j$,然后删除外存中所有已被调入内存的原始样本,并将新样本 $X_j$ 写入外存.

(3) 对于任意 $j$,$1\leq j\leq K$,删除 HDS-Tree 中的子树 $N_j$,并生成一个新叶结点 $L_j$ 代替子树 $N_j$ 的原始位置,其中叶

结点 $L_j$ 中存储了第(2)步中新样本 $X_j$ 的样本元数据.

在多层递阶抽样过程中,当$\lambda$=1 时,外存中的样本数量没有发生变化,而样本容量减小了,从而 HDS-Tree 的结构不发生变化,仅是叶结点中样本元数据的内容发生了变化.当$\lambda$>1 时,HDS-Tree 的高度会降低.例如图 1(a)显示了实施多层递阶抽样前 HDS-Tree 的结构,图 1(b)显示了实施多层递阶抽样后 HDS-Tree 的结构,其中$\lambda$=2.

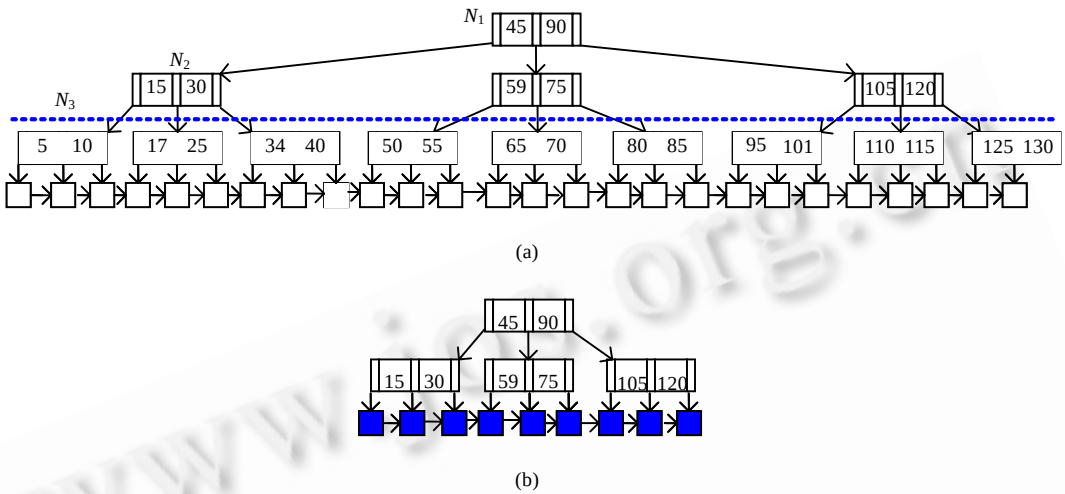


Fig.1 HDS-Tree structure before and after recursive-sampling procedure with $\lambda$=2

图 1 $\lambda$=2 时多层递阶抽样前、后的 HDS-Tree 结构

在第 $i$ 次多层递阶抽样过程中,需要保证新样本 $X_j^{(i)}$中数据的随机性.设新样本 $X_j^{(i)}$由样本集合 $S_j^{(i-1)}$经随机抽样得到,则对于 $S_j^{(i-1)}$中的不同样本实施迭代抽样时应该设置不同的抽样比例.设 $S_j^{(i-1)}$中所有样本的最小下界时间戳为 $t_1$,最大上界时间戳为 $t_2$,由式(1)知,新样本 $X_j^{(i)}$的抽样比例为 $f_j^{(i)}=\min\{e^{((t_1+t_2)/2-t_n)\cdot r/D}, f_{\max}\}$. $\forall X_k^{(i-1)}\in S_j^{(i-1)}$,设在第 $i$ 次多层递阶抽样前样本 $X_k^{(i-1)}$对应的叶结点存储的抽样比例为 $f_k^{(i-1)}$,则样本 $X_k^{(i-1)}$被调入内存进行迭代抽样时的抽样系数应为 $f_j^{(i)}/f_k^{(i-1)}$.

定理 **4**. 多层递阶抽样过程保持了 HDS-Tree 的最优性(证明略). □

由于$\lambda$>1 时的多层递阶抽样过程可以降低 HDS-Tree 的高度,从而可以减小 HDS-Tree 对内存空间的占有量.因此,为确保内存可以容纳下 HDS-Tree 结构,当 HDS-Tree 达到最大高度之前,应该实施一次$\lambda$>1 的多层递阶抽样过程以降低 HDS-Tree 的高度.下面的定理 5 给出了内存中 HDS-Tree 的最大高度.

定理 **5**. 给定内存空间大小 $M$,HDS-Tree 的最大高度为 $1+\log_b \frac{(b-1)M+S_{Node}}{S_{Node}+S_{Leaf}(b-1)}$,其中 $S_{Node}$ 和 $S_{Leaf}$ 分别为一个内结点和一个叶结点的大小(证明略). □

## 3 历史数据的聚集查询算法

对历史数据实施抽样存储以及维护 HDS-Tree 结构的最终目的是支持历史数据的聚集查询请求.数据流历史数据的聚集查询可以定义为 $Agg\{s|\langle s,t\rangle\in Stream\wedge t\in Scope\wedge Constraint(s)\}$,其中 *Stream* 表示数据流,*Agg* 是聚集函数,如 SUM,COUNT,AVG,MAX,MIN 等,*Scope* 为查询时间段,*Constraint* 为查询约束条件(包括选择谓词条件或者分组聚集条件).为方便描述起见,我们将 *Constraint* 为空时的聚集查询语句称为简单聚集查询,*Constraint* 非空时的聚集查询语句称为复杂聚集查询.使用 HDS-Tree,我们可以高效地执行这两类聚集查询.

不失一般性,下面我们以 AVG 聚集查询为例介绍历史数据的聚集查询算法,并进行误差分析.

### 3.1 简单聚集查询算法

简单聚集查询只需自顶向下递归访问 HDS-Tree 就可以得到查询聚集值.下面是简单聚集查询算法的描述:

输入:HDS-Tree 的根结点 $N$,简单聚集查询的查询时间段 $T_q$,聚集查询结果的初始值 *value*.

输出:历史数据的简单聚集值查询结果 *value*.

SimpleAggregate ($N$,$T_q$,*value*) /**value* 为聚集查询结果*/

(1) IF ($T_q$=$N$→$T$) THEN *value*=accumulate(*value*,$N$→*value*);/**N*→*T* 和 *N*→*value* 分别为时间段和聚集值*/

(2) ELSE IF (*N* 是叶结点) THEN *value*=accumulate $\left(value,\frac{T_q \cap N \to T}{N \to T} \times N \to value\right)$;

(3) ELSE FOR (结点 *N* 的每一个非空儿子 $N$→$N_i$);

(4) IF ($T_q$∩$N$→$N_i$→$T$≠∅) THEN SimpleAggregate ($N$→$N_i$,$T_q$∩$N$→$N_i$→$T$,*value*).

最坏情况下,SimpleAggregate 算法的时间复杂性为 $O(b(\lceil \log_b n \rceil+1))$,$b$ 为结点的最大扇出度,是一个常量,$n$ 为 HDS-Tree 的叶结点个数(即外存中存储的样本个数),$\lceil \log_b n \rceil+1$ 为 HDS-Tree 的高度.

定理 **6**. 设简单 AVG 聚集查询的估计值为 $\overline{Y_n}$,真实值为$\mu$,则

$$P\left\{\left|\overline{Y_n}-\mu\right| \le \frac{z_p S}{\sqrt{n}}\right\}=p,$$

其中 $z_p$ 为标准正态分布的$(p+1)/2$ 分位点,$n$ 为查询时间段覆盖的数据元组个数,$S$ 为样本标准差(证明略).

### 3.2 复杂聚集查询算法

复杂聚集查询的过程是根据查询时间段访问 HDS-Tree 中叶结点存储的地址信息,找到外存中存储的对应样本,然后通过在内存中扫描样本得到查询聚集值.具体算法描述如下:

输入:HDS-Tree 的根结点 *N*,复杂聚集查询的查询时间段 $T_q$.

输出:历史数据的复杂聚集值查询结果 *result*.

ComplexAggregate($N$,$T_q$)

(1) 初始化 *result*,访问 HDS-Tree,分别得到包含 $T_q$ 的下界时间戳和上界时间戳的叶结点指针 $pt_s$ 和 $pt_e$;

(2) WHILE ($pt_s$≠$pt_e$)

(3) 利用地址信息 $pt_s$→*pd* 找到外存中的样本 *X*,并将其调入内存;

(4) 在内存中对样本 *X* 执行复杂聚集操作,并根据 *X* 的抽样比例估算出原始数据的聚集值 *value* ;

(5) *result*=accumulate (*result*, *value*) ; /* accumulate 函数定义参见第 2.2 节*/

(6) $pt_s$=$pt_s$→*pr*; /*令 $pt_s$ 指向它的右邻叶结点*/

(7) 输出 *result*.

ComplexAggregate 算法需要访问的 I/O 次数与查询时间段的跨度大小有关,查询时间段的跨度越大,I/O 访问次数也就越多.

定理 **7**. 设复杂 AVG 聚集查询的估计值为 $\overline{Y_n}$,真实值为$\mu$,则

$$P\left\{\left|\overline{Y_n}-\mu\right| \le \left(\frac{z_p^2 G_n}{nT_n^2(u)}\right)^{1/2}\right\}=p \text{ (证明略).}$$

## 4 实验结果

我们在 Windows 2000 环境下利用 VC 程序模拟搭建了数据流处理系统,并且实现了本文中的所有算法.实验机器的配置为 P4 2.4GHz,256MB 主存,40GB 硬盘.实验数据来源于 TPC-H 测试平台 CUSTOMER 表中的数据[11].

### 4.1 实验系统和实验内容

在实验系统中,由一个进程负责连续从 CUSTOMER 表中读取数据元组并发送到滑动窗口中.我们给每一个进入滑动窗口的数据元组按照先后顺序分配一个时间戳,它是一个递增的整型值.同时,还有一个进程对流出滑动窗口的历史数据实施初始抽样,并在内存中维护 HDS-Tree.另一个进程负责定期地对外存中的样本实施多

层递阶抽样,以保证外存中能够存储全部不同时间段的样本.

在实验中,我们设定了 HDS-Tree 的最大扇出度 $d$ 为 6,外存中所有样本个体的最小下界时间戳为 0,最大上界时间戳为 1 000 000.外存的最大存储容量 $D$ 为 100 000 个元组,数据流的平均流速为 1000 个/s,最大抽样比例 $f_{max}$ 为 90%.历史数据聚集查询语句中的聚集操作作用在 CUSTOMER 表中的 ACCTBAL 属性列上,谓词约束条件作用在 CUSTOMER 表中的 NATIONKEY 属性列上.实验中每种算法运行了 10 次,我们取 10 次运行结果的平均值作为实验结果.

实验内容是考察简单聚集查询和复杂聚集查询的查询结果误差度随查询时间段的变化情况.由于篇幅所限,我们只给出了有代表性的 AVG 聚集查询实验结果.

聚集查询算法的性能受查询时间段的跨度和查询时间段的历史久远程度两个因素影响,因此我们根据查询时间段将数据流历史数据的聚集查询进行了分类.设查询时间段为 $T[t_s,t_e]$,固定 $T$ 的上界(即 $t_e$)不变,把 $T$ 的下界($t_s$)作为自变量,我们将得到的一组查询语句称为 *Landmark*[12]聚集查询,实验中设定了 $t_e$ 的取值为最大上界时间戳;固定查询时间段为 $T[t_s,t_e]$的跨度不变,把 $T$ 的下界(即 $t_s$)作为自变量,我们将得到的一组查询语句称为 *Snapshot*[12]聚集查询,实验中设定了 $T[t_s,t_e]$的跨度取值为 5 000.

### 4.2 历史数据的简单聚集查询

图 2 显示了 Landmark 简单聚集查询实验结果的相对误差变化情况.随着 $t_s$ 的增大,查询时间段的跨度逐渐减小,而查询结果的误差呈现逐渐增大趋势,说明在小规模数据量上的 Landmark 简单聚集查询容易产生大的误差.尽管如此,当 $t_s$ 接近最大上界时间戳时,相对误差也没有超出 0.05%.

Snapshot 简单聚集查询的误差变化情况如图 3 所示.随着 $t_s$ 的变化,查询结果的误差基本呈现平稳趋势,相对误差保持在 0.22%以内,这是由于简单聚集查询只需要访问内存中的 HDS-Tree,而不访问外存中的样本数据,因此具有相同的查询时间段跨度的 Snapshot 简单聚集查询的误差受历史时间久远程度的影响不大.

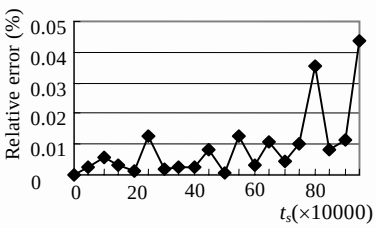


Fig.2 Error of landmark simple-aggregate

图 2 Landmark 简单聚集查询实验结果的误差

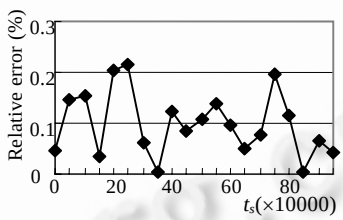


Fig.3 Error of snapshot simple-aggregate

图 3 Snapshot 简单聚集查询实验结果的误差

### 4.3 历史数据的复杂聚集查询

图 4 显示了 Landmark 复杂聚集查询实验结果的相对误差变化情况.随着 $t_s$ 的增大,访问的数据中较新历史数据的比重逐渐增大,因此误差呈现逐渐减小趋势.从图中可以看出,当 $t_s$ 超过一定大小的时间戳时,误差趋于平稳趋势,相对误差最终保持在 1.5%以内.

Snapshot 复杂聚集查询的误差变化情况如图 5 所示.随着 $t_s$ 的变化,访问的历史数据的抽样比例逐渐增大,因此查询结果的误差呈现逐渐降低趋势.与 Snapshot 简单聚集查询不同,Snapshot 复杂聚集查询需要访问外存中的样本数据,因此该查询受历史时间久远程度的影响很大.从图中可以看出,针对太久远历史数据的 Snapshot 复杂聚集查询误差比较大且不稳定,这是由于太久远历史数据的抽样比例非常低,样本数据太稀疏,以至于在样本中没有找到任何满足查询条件的数据的缘故.

### 4.4 实验小结

由于算法中抽样过程的随机性,使得聚集查询的误差变化情况在局部出现了不稳定性,但从总体上看,误差实验结果总是趋于一种上升或者下降或者平稳的趋势,与理论分析结果相吻合,从而验证了本文算法的有效性.

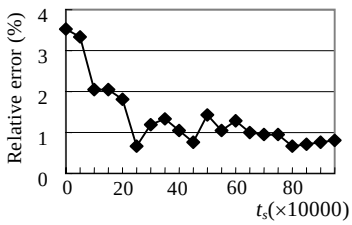


Fig.4 Error of landmark complex-aggregate

图 4 Landmark 复杂聚集查询实验结果的误差

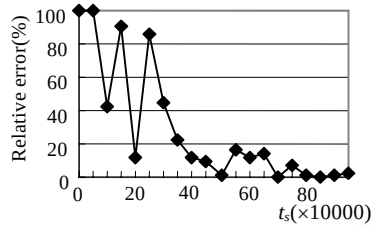


Fig.5 Error of snapshot complex-aggregate

图 5 Snapshot 复杂聚集查询实验结果的误差

## 5 总 结

本文提出了数据流历史数据的存储管理和聚集查询处理的方法.通过对最近产生的历史数据实施初始抽样存储,可以减轻外存空间的存储负担.另外,通过对外存中的样本实施多层递阶抽样存储,进一步减少了外存中历史数据的容量,实现了在有限的外存中存储无限的数据流历史数据的样本.通过在内存中建立 HDS-Tree,对历史数据的聚集值进行索引存储,可以支持各种针对数据流历史数据的聚集值查询请求.理论分析与实验结果表明,本文提出的方法可以有效地用于数据流历史数据的存储与分析.

**References**:

[1] Babcock AK, Babu S, Datar M. Model and issues in data stream systems. In: Popa L, ed. Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. Madison: ACM, 2002. 1−16.

[2] Golab L, Ozsu MT. Issues in data stream management. SIGMOD Record, 2003,32(2):5−14.

[3] Araru A, Babu S, Widom J. An abstract semantics and concrete language for continuous queries over streams and relations. Technical Report, Stanford University Database Group, 2002. Available at http://dbpubs.stanford.edu/pub/2002-57

[4] Guha S, Koudas N. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In: Stefano C, Christoph F, Pat S, eds. Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE Computer Society, 2002. 567−576.

[5] Madden S, Shah M, Hellerstein JM, Raman V. Continuously adaptive continuous queries over streams. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 49−60.

[6] Gehrke J, Korn F, Srivastava D. On computing correlated aggregates over continual data streams. In: Afef WG, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. Santa Barbara: ACM, 2001. 13−24.

[7] Dobra A, Gehrke J, Garofalakis M, Rastogi R. Processing complex aggregate queries over data streams. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 61−72.

[8] Chen Y, Dong G, Han J, Wah BW, Wang J. Multi-Dimensional regression analysis of time-series data streams. In: Bernstein PA, Loannidis YE, Ramakrishnan R, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 323−334.

[9] Zhang D, Gunopulos D, Tsotras VJ, Seeger B. Temporal aggregation over data streams using multiple granularities. In: Jensen CS, Jeffery KG, eds. Proc. of the 8th Int'l Conf. on Extending Database Technology. LNCS, 2002. 646−663.

[10] Olken F. Random Sampling from Databases [Ph.D. Thesis]. Berkeley, University of California, 1993.

[11] Transaction Processing Performance Council. TPC Benchmark H (Decision Support) Standard Specification. TPC, 2002. http://www.tpc.org/tpch/default.asp

[12] Chandraskearan S, Franklin MJ. Streaming queries over streaming data. In: Bernstein PA, Loannidis YE, Ramakrishnan R, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 203−214.