一种存储和索引历史数据流数据的方法*

葛君伟，公丕强，刘兆宏
(重庆邮电大学 中韩 GIS 研究所，重庆 400065)

摘 要：通过对流数据的抽样存储，并在内存中建立 B^+ 树结构，对抽样数据和常用聚集操作进行索引，实现了对无限数据流历史数据的抽样存储管理，有效地支持了数据流历史数据查询。

关键词：数据流；历史查询；抽样；存储；索引；B^+ 树
中图分类号：TP311.13 文献标志码：A 文章编号：1001-3695(2007)06-0104-03

Method of Storing and Indexing Historical Streaming Data

GE Jun-wei, GONG Pi-qiang, LIU Zhao-hong
(Sino-Korea Chongqing GIS Research Institute, Chongqing University of Posts & Telecommunications, Chongqing 400065, China)

Abstract: An approach was proposed to store the historical streaming data by using sampling method and B^+ tree structure which indexed the aggregation of historical streaming data and supported all kinds of queries over historical streaming data. This approach is effective in querying historical streaming data to get approximate answer.

Key words: data stream; historical query; sample; store; index; B^+ tree

0 引言

数据流(Data Stream)应用的出现引起了国内外专家和学者的关注。数据流管理技术作为一种新兴技术已经得到广泛研究。目前通用的 DSMS(Data Stream Management System)包括 TelegraphCQ[1]、Aurora[2] 和 STREAM[3]。

TelegraphCQ 致力于多数据流上的并发连续查询的自适应和共享处理；Aurora 是一个面向监测应用的 DSMS，用于处理在线时序安排和负载平衡；STREAM 集中于资源管理和近似连续查询。美国加州大学伯克莱分校正在构建一个 TelegraphCQ 系统，该系统用于连续的数据流处理。TelegraphCQ 的目的在于处理大量高速变化的数据流而进行的大量连续查询流。布朗大学的 Aurora 工程建造了一个专门用于流监控的数据处理系统。斯坦福大学已经开始了一种全面 DSMS 的设计和原型实现。该系统为 STREAM (Stanford Stream Data Manager)。STREAM 是一个以关系为基础的数据流管理系统，它重点在于内存管理和近似查询。它可以用于处理快速的、易变的、大量涌入的数据流信息，其连续查询能力非常好。STREAM 的主要处理技术包括连续的自我监控和再优化、适应于不同需要的近似查询、合理的资源分配和使用。

目前，对数据流的研究大多集中在对当前流数据的分析和处理上。相对于数据流的连续性和无限性，存储器的存储能力是有限的。随着数据的流入，旧的数据将被抛弃；当查询涉及到历史数据时，因为没有可用的数据而不能得到查询的结果。

例如，在监控某一地点实时温度的传感器网络中，查询过去一段时间的最高温度。因为没有历史数据的存储，无法得到答案。那为什么不用传统的数据库管理系统呢？众所周知，传统的数据库管理系统(DBMS)是静态的、有限的，查询可以处理所有存储的数据，但是用传统数据库存储流数据是不可能的。DBMS 与 DSMS 的比较如表 1 所示[4]。

表 1 DBMS 与 DSMS 比较

比较项	传统的数据库管理系统(DBMS)	数据流管理系统(DSMS)
设计原则	旨在处理永久性数据。其设计与开发主要强调维护数据的完整性、一致性，不考虑与数据及其处理相关联的时限	针对具有简单结构与联系，稳定和可预报数据(或资源)要求的任务，支持数据及其处理的实时限制
对象	静态存储的信息，可在任何需要的时候进行存取、检索	以连续的、有序的“流”形式输入。经过在线检索后，输入的数据被淘汰或归档。其中被淘汰的数据无法进行二次检索
典型操作	在持久的关系上进行一次查询，可对欲检索的记录集随机存取	对暂态的流数据进行持续检索，对输入的流数据进行顺序扫描
数据存储	利用外存储器保持历史数据，容量近乎无限，数据的存储由应用程序通过数据管理语言显式执行(被动方式)	存储和处理均在有限的主存中进行，容量有限，数据的摘要信息由系统提取并更新(主动方式)，过期的历史数据不予保存
数据有效性	查询所得结果表示当前状态，即有效数据	无当前状态，数据序列中数据分量的到达时间及其到达顺序决定检索结果

收稿日期：2006-04-04；修返日期：2007-05-16 基金项目：韩国委托研究项目；重庆邮电大学科研基金资助项目(C2003-1)

作者简介：葛君伟(1962-)，男，江西人，所长，教授，博士，主要研究方向为计算机信息管理系统、空间信息系统、软件工程；公丕强(1980-)，男，山东人，硕士研究生，主要研究方向为空间数据库(gongpiqiang@hotmail.com)；刘兆宏(1975-)，男，四川人，讲师，主要研究方向为时空数据库和空间定位技术.

针对数据流中一些历史查询, 不仅为了节省存储空间, 还满足数据流中的近似查询, 本文介绍了一种抽样方法。尤其是基于时间的滑动窗口的元组数不确定的情况, 采用经典的 B^+ 树来索引所存储的抽样。

目前对于数据流的研究主要集中于怎样处理当前到达的数据, 而忽略了历史流数据的分析和管理。本文采用一种抽样的方法存储流数据, 支持历史查询的近似查询, 同时存储一些常用聚集操作的结果以支持这些常用聚集操作的历史查询。

1　基于滑动窗口的数据流抽样

数据流的查询过程是持续查询(Continuous Query) [5] 。持续查询所关心的并不是全部的数据, 而是近期到达的部分数据, 所以数据流中的持续查询采用滑动窗口(Moving Window) 机制[5]、基于滑动窗口的查询。

滑动窗口可以看做是数据流有限部分的历史性快照。基于这种定义可以将滑动窗口划分为三种类型, 即基于时间的滑动窗口、基于元组的滑动窗口和分区滑动窗口。本文的抽样方法便是基于滑动窗口的抽样。不同机制的滑动窗口, 其特性不同, 抽样方法也不同。

1.1　基于元组的滑动窗口抽样

数据流 S 基于元组的滑动窗口实质是大小固定的滑动窗口, 窗口模型以正整数 N 作为参数。直观地看, 带有时间的基于元组的滑动窗口的输出关系是有序数据流到目前为止最近到达的 N 个元组。形式上来说, 是由数据流 S 到目前为止最大时间戳的 N 个元组组成。

基于元组的滑动窗口的元组数 N 是已知的, 在 N 已知的情况下从中抽样 n 个(已经有很多算法支持类似的抽样) 。例如文献[6] 介绍了一种 Chain-sample 抽样算法。

1.2　基于时间的滑动窗口抽样

数据流 S 基于时间的滑动窗口实质上是大小变化的窗口; 窗口模型以时间间隔 T 作为参数, T 为数据流运行时间的计算周期。从直观上来说, 基于时间的滑动窗口定义了时间间隔 T 上的输出关系, 从而捕获到有序数据流最新到达的元组。输出关系 $R(\tau) = \{ S \mid S, \tau \in S \wedge (\tau' \leq \tau) \wedge \tau' \geq \max\{\tau - T, 0\}$ [7] 可定义为(有两种特殊情况) :

①当 $T = 0$, $R(\tau)$ 由数据流 S 中带有时间戳的元素组成;

②当 $T = \infty$, $R(\tau)$ 由数据流 S 中所有时间戳的元素组成。

在基于时间的滑动窗口中, 窗口中的元组数 N 是不确定的, 可能在某个时间 T 内, 数据流速快、元组数量大; 也可能在某个时间 T 内, 接收到的数据流数据少, 则 N 的值就小。所以本文抽样的样品容量 n 也应该是动态变化的。

1.2.1　抽样比例

定义 1　抽样比例(Sampling Ratio) $f = n/N$。

其中, n 为样本容量; N 为 T 周期内滑动窗口元组数。

为了查询方便, 为抽样所得的样品建立 B^+ 树索引。n 的值取决于数据流中元组的大小和存储样品的数据页(Data Page) 的容量以及数据页中存储的其他信息, 如聚集结果等。抽样比例 f 与抽样数据一起存储, 可以近似逆推窗口的数据, 以满足一些近似查询。

1.2.2　抽样算法

基于时间的滑动窗口中 N 是变化的, 不能提前预知, 所以从中抽取 n 个元组的方法势必与基于元组的滑动窗口的方案有所不同。

定义 2　(时间段) 给定两个时间戳 t_{i-1} 和 t_i, 且 $t_{i-1} \leq t_i$, 则[t_{i-1}, t_i] 构成一个时间段(记为 T) 。t_{i-1} 和 t_i 分别称为 T 的下界和上界, $t_i - t_{i-1}$ 的值称为 T 的跨度, 记为 ΔT。

(1) 算法描述

输入: [t_{i-1}, t_i] 周期内滑动窗口的 N 个元组

输出: [t_{i-1}, t_i] 周期内滑动窗口抽得的 n 个元组

①t_i 时刻起, 读入最先到达的 n 个元组, 分别存储在数组 array[0] , array[2] , …, array[n - 1] 中作为候选样品。

②顺序读入剩余的元组, 处理机制如下:

(a) 判断是否处理完所有窗口中的元组, 完成则转到③; 否则继续(b) 。

(b) $t = n$。

(c) 读入第 $t+1$ 个元组, 计算 R = TRUNC(($t+1$) * RANDOM( )) 。其中 $0 \leq R \leq t$。若 $R < n$, 则第 $t+1$ 个元组以 $n/(t+1)$ 的概率替换候选样品 array[R] ; 若 $R \geq n$, 重复(c) 。

(d) 直到读完窗口中的 N 个元组。

③输出 array[0] , array[2] , …, array[n - 1] , 即为此滑动窗口的抽样。

(2) 算法

```
{
  for ( int j = 0 ; j < = n - 1; j + + )
    //最先到达的 n 个元组作为候选样品
  READ-NEXT-TUPLE( array[ j] ) ;
  //分别存入 array[ 0] , array[ 2] , … , array[ n - 1]
  t = n;
  //当前处理的元组
  while not eof //处理剩余的元组
  {
  t = t + 1;
  R = TRUNC( t* RANDOM( ) ) ; //0 ≤ R ≤ t - 1
  If R < n
  //替换元组 R
  READ-NEXT-TUPLE( array[ R] ) ;
  else
  SKIP-TUPLE( l) ; //跳过这个元组
  }
}
```

2　数据流历史信息的存储与索引

2.1　存储

本文采用 B^+ 树结构来索引存储的数据。抽样结果存储在数据页中; 每个滑动窗口都有一个抽样, 每个抽样对应一个

数据页。抽样比例 f 也存储在对应的数据页中,根据抽样数据和抽样比例 f,可以近似逆推滑动窗口的原始数据,以满足一些近似查询。

在对滑动窗口抽样时,整个窗口必须被扫描一遍。可以在抽样的同时做一些该窗口的常用聚集操作,如 SUM、MAX、MIN 等;把这些常用聚集操作的结果也存储在对应的数据页中。当历史查询是所存储的常用聚集操作时,可以得到精确的而不是近似的结果。

2.2 索引

如此海量的历史流数据,为支持数据流查询的快速响应,本文为众多的数据页建立索引,采用经典的 B^+ 树结构。之所以采用 B^+ 树结构,主要是基于 B^+ 树的如下特点:所有的叶子节点中包含了全部关键字的信息,及指向含这些关键字记录的指针;且叶子节点本身依关键字的大小自小而大地顺序链接。

图 1 是一个 3-B^+ 树结构。

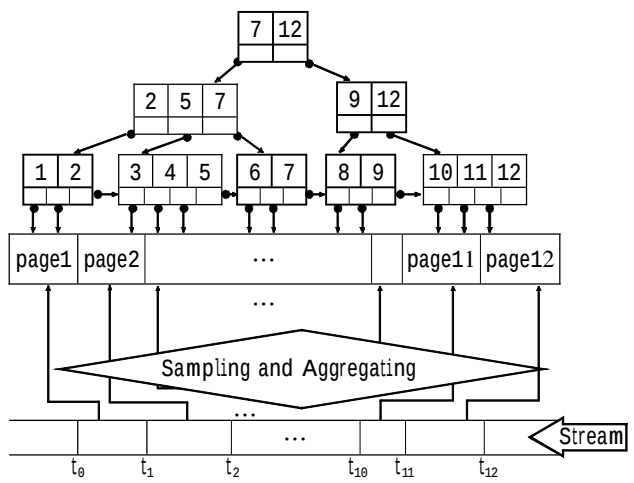

图 1 基于时间的数据流抽样存储与索引结构图

(1) 根节点和内节点存储的信息

根节点和内节点存储的信息为

$(P_1, \text{key}_1, P_2, \cdots, P_j, \text{key}_j)$

其中,Key_j 是关键字,$\text{key}_1 < \text{key}_2 < \cdots < \text{key}_j$ ($j \leq m$, m 是 m-way B^+ 树)。

每一个基于时间的滑动窗口都有一对开始时间和结束时间,定义为$[t_{i-1}, t_i]$,称 t_i 为物理时间。$[t_{i-1}, t_i]$ 滑动窗口对应的数据页映射为 key_i,称 key_i 为逻辑时间。采用逻辑时间代替物理时间的存储方式,在一定程度上节省了存储空间,方便基于时间的历史查询。

(2) 页节点包含所有的关键字

每个关键字有一个指针指向该关键字对应时间的数据页。

每个页节点有一个指针指向右边的叶子节点。

数据页中存储的信息如图 2 所示。key_i 是关键字;$f_i = n_i / N_i$ 是第 i 个滑动窗口的抽样比例;SUM、MAX…是对应的数据流窗口常用的聚集操作结果;n_i 是抽样的容量,其值由数据页的大小、元组的大小样以及数据页中所要存储的信息决定。

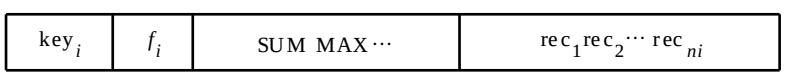

图 2 页结构

(3) 叶子节点

所有的叶子节点通过向右的指针链接在一起,所以在处理一些历史查询时方便快捷。例如,查询 t_0 到 t_{11} 数据流中的 MAX。此查询仅涉及到 t_0 到 t_{11} 的滑动窗口,即仅涉及 page1 到 page11。通过内存中建立的 B^+ 树索引,很快找到 page1 中的 MAX 值,然后可以直接通过叶节点的右指针依次找到剩余页的 MAX 值,从而很容易得到此查询结果。

3 结束语

采用抽样方法存储海量的流数据,减轻了存储器的负载。抽样比例和常用聚集操作结果也被一同存储在数据页中,用 B^+ 树结构索引存储的数据页。这个方法能够有效地存储和分析历史流数据。当查询涉及历史流数据时,如果查询是存储在数据页中的常用聚集操作,则得到准确的结果;如果查询不是存储的聚集操作,则利用存储的抽样数据和抽样比例得到近似查询结果。因为在很多情况下,数据流查询并不要求绝对精确,近似的结果更能满足数据流的快速、实时响应。

笔者将数据流以离散的时间段划分,在处理那些起止时间都是划分时间点的查询有较好的响应;而查询所涉及的起止时间不是划分点时,即使查询是存储的常用聚集操作也不能得到准确的答案,仍是近似的。下一步会考虑尽量将划分的粒度缩小,即 ΔT 小一点。但是粒度太小了会使窗口数增加,进而使数据页增加,花费更大的代价以维护 B^+ 树。

本文讨论的前提是在内存中建立 B^+ 树。随着时间的推移、数据的流入,B^+ 树会很大,以至于无法存储在内存中,可以再为 B^+ 树做二级索引。

参考文献:

[1] IRISH C, OWEN C, AMOL D, *et al*. TelegraphCQ: continuous dataflow processing[C]. //ALON Y H. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2003: 668-668.

[2] DANIEL J A, DON C, UGUR C, *et al*. Aurora: a new model and architecture for data stream management[J]. Int'l Journal on Very Large Data Bases, 2003, 12(2): 120-139.

[3] ARASU A, BABCOCK B, BABU S, *et al*. STREAM: the stanford stream data manager[J]. IEEE Data Engineering Bulletin, 2003, 26(1): 19-26.

[4] 桂浩,冯玉才,李又奎. 面向流数据的数据管理系统的研究[J]. 计算机应用研究, 2005, 22(1): 88-90, 133.

[5] SHIVANATH B, JENNIFER W. Continuous queries over data streams[J]. SIGMOD Record, 2001, 30(3): 109-120.

[6] BABCOCK B, DATA M, MOTWANI R. Sampling from a moving window over streaming data[EB/OL]. (2001-09-26). http://dbpubs.stanford.edu/pub/2001-33.

[7] ARASU A, BABU S, WIDOM J. The CQL continuous query language: semantic foundations and query execution[EB/OL]. (2003-10-22). http://dbpubs.stanford.edu/pub/2003-67.