ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.4, April 2006, pp.740−749
DOI: 10.1360/jos170740

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

基于滑动窗口的数据流连续 J-A 查询的处理方法*

王伟平[1+], 李建中[1,2], 张冬冬[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)

Sliding Window Based Method for Processing Continuous J-A Queries on Data Streams

WANG Wei-Ping[1+], LI Jian-Zhong[1,2], ZHANG Dong-Dong[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: wpwang@hit.edu.cn, http://www.hit.edu.cn

Wang WP, Li JZ, Zhang DD, Guo LJ. Sliding window based method for processing continuous J-A queries on data streams. *Journal of Software*, 2006,17(4):740−749. http://www.jos.org.cn/1000-9825/17/740.htm

Abstract: Sliding window join aggregation continuous queries (J-A queries for short) are often used in data stream applications. The intuitive method for processing these queries is to construct steaming operator tree and execute the tree in pipeline. The space cost of this method is $O(\alpha\times\beta)$, where α and β are the sizes of two sliding windows respectively. To reduce the requirement of memory, which is the most expensive resource in data stream query processing system, two novel sliding window J-A continuous query processing algorithms IC and TC are presented in this paper, whose space cost are both $O(\alpha+\beta)$. Theoretical analysis and experimental results show that the algorithms are more effective.

Key words: data stream; sliding window; join aggregation; continuous query

摘 要: 数据流滑动窗口连接聚集连续查询(简记 J-A 查询)是经常使用的一类查询.这类查询的直观处理方法是创建查询操作树,以流水线的方式计算查询结果.这种方法需要在主存中保存滑动窗口连接的结果,查询处理的主存空间开销为 $O(\alpha\times\beta)$,其中α,β为参加连接两个滑动窗口的大小.在数据流的查询处理中,内存是最重要的计算资源.提出了两种滑动窗口 J-A 连续查询处理算法——IC 算法和 TC 算法,使得查询处理的空间开销降为 $O(\alpha+\beta)$.理论分析和实验结果表明,所提出的算法具有更高的效率.

关键词: 数据流;滑动窗口;连接聚集;连续查询

中图法分类号: TP311 文献标识码: A

近年来,在很多应用领域中均出现了一种称为数据流的新型数据模式,例如传感器网络中的监测数据、互联网中流动的 IP 数据包、Web 服务器上的用户登录记录、电信公司的通话记录等.与传统的数据模式不同,数据流数据的特点是实时无限的,无法全部保存下来进行处理.数据流查询处理成为学术界的一个挑战性问题,研

* Supported by the National Natural Science Foundation of China under Grant Nos.60273082, 60473075 (国家自然科学基金); the Natural Science Foundation of Heilongjiang Province of China under Grant No.zjg03-05 (黑龙江省自然科学基金)

Received 2004-04-27; Accepted 2005-07-11

究人员对其开展大量研究工作,提出了许多有效的数据流查询处理技术[1−3].

数据流系统中的查询类型主要为连续查询.连续查询注册到系统后,随着数据流新数据的到来而不断返回查询结果,除非用户发出指令撤销该查询,否则连续查询将不断地执行.滑动窗口连续查询是数据流上常用的一类连续查询.滑动窗口是指在数据流上设定的一个区间,该区间只包括数据流最近的部分数据.随着新数据的到来,窗口向前移动,用新数据替换旧数据.滑动窗口可以分为顺序滑动窗口和时间滑动窗口两类[4].顺序滑动窗口内保存最近到来的 K 个元组,其大小固定.时间滑动窗口存储的是最近 T 时间内到达的元组,大小可变.滑动窗口连续查询只在数据流的滑动窗口内处理查询.

数据流滑动窗口连续查询处理问题得到了研究人员的广泛关注.Kang 等人讨论了滑动窗口连接查询的处理问题,提出了几种平衡的流水线连接算法来处理滑动窗口连接查询,并给出了一种适合于数据流的 unit-time-basis 连接代价模型,根据数据流流速计算连接算法的代价,选择代价小的连接算法用于处理查询[5].Lukasz 等人讨论了多路滑动窗口连接查询的处理问题,给出了启发式规则来选择合适的连接执行顺序[6].Viglas 等人提出了另一种多路滑动窗口连接的处理方法,他们对 SHJ 算法进行扩展,实现了一个能够处理多个输入流的连接操作算子 Mjoin[7].Arasu 等人讨论了滑动窗口聚集查询的处理问题[8].已有的这些研究工作仅涉及滑动窗口单一查询的处理问题,而没有考虑数据流滑动窗口连接聚集查询(即同时包含连接和聚集操作的查询,以下简记为 J-A 查询)的处理问题.滑动窗口 J-A 连续查询有着很多应用,下边是两个滑动窗口 J-A 连续查询的实例.

例 1:在互联网性能监测应用中,主干网中流动的 IP 数据包构成了数据流 A,与主干网相连的一个分支网中的 IP 数据包构成数据流 B.如果网络管理人员需要监测在过去一个小时中,主干网上的 IP 数据包中来自于该分支网的数据包个数,则需要使用如下查询:

SELECT *COUNT*(*)
FROM *A*[60 MINUTE], *B*[60 MINUTE]
WHERE *A.src=B.src* AND *A.dest=B.dest*

其中,*A*[60 MINUTE]和 *B*[60 MINUTE]分别为数据流 A 和 B 上时间长度为 60 分钟的时间滑动窗口.

例 2:在一个智能大厦中,监测温度的传感器产生数据流 A,监测烟浓度的传感器产生数据流 B.A 数据流包括属性(location,time,temperature).B 数据流包括属性(location,time,strength).数据流 A 和 B 不断地向监控中心发送监测数据.如果在过去的 10 分钟内,某个房间的温度达到了 40°C 以上并且烟的浓度大于 0.6 的情况连续出现 5 次,则需要启动自动救火装置.管理人员可以使用下面的查询找到这样的房间:

SELECT *location*, *COUNT*(*) FROM *A*[10 MINUTE], *B*[10 MINUTE]
WHERE *A.location=B.location* and *A.temperature*>=40 and *B.strength*>0.6
GROUP BY *location*
HAVING *COUNT*(*)>5.

目前,还没有文献讨论滑动窗口 J-A 连续查询的处理问题.本文提出了两种滑动窗口 J-A 连续查询的处理算法,理论分析和实验结果表明,两种算法具有很好的时间和空间复杂性.

本文第 1 节给出简单滑动窗口 J-A 连续查询的处理算法.第 2 节讨论复杂滑动窗口 J-A 连续查询的处理方法.第 3 节给出实验结果及其分析.第 4 节对本文的工作进行总结.

1 简单滑动窗口 J-A 连续查询处理算法

在这一节中,首先给出不包含分组操作(group by)的滑动窗口 J-A 连续查询的处理算法.在第 2 节中,将对本节提出的算法进行扩展,给出包含分组操作的滑动窗口 J-A 连续查询和多路滑动窗口 J-A 连续查询的处理方法.下面,首先给出本文中使用的一些概念.

定义 **1**(数据流). 一个数据流是一个按照时间递增顺序排列的无穷时间序列 $S=\{\langle s_1,t_1\rangle,\langle s_2,t_2\rangle,\ldots,\langle s_i,t_i\rangle,\ldots\}$,$s_i$ 是在时刻 t_i 出现的序列元素.

定义 **2**(滑动窗口). 设 T 是一个时间长度,$t>T$ 是一个变化的时刻,则称 $SW[t-T:t]$为 S 的一个时间间隔为 T

的滑动窗口,其中 t 和 T 的单位相同,并且 t 为相对于 S 的起始观测时刻的时间距离.

显然,$SW[t-T:t]$是随着 t 的变化而滑动,故称其为滑动窗口.

定义 **3**(滑动窗口快照). 滑动窗口 SW 在 T_K 时刻的快照定义为 $SW_{T_K}=SW[T_K-T:T_K]$.

一种直观的处理滑动窗口 J-A 连续查询的方法是建立查询操作树,以流水线的方式执行连接操作和聚集操作.这种处理方法需要保存滑动窗口连接的结果,查询处理的空间开销为 $O(\alpha\times\beta)$,占用了大量的主存空间.在数据流的查询处理中,内存是最重要的计算资源.下面给出两种有效的滑动窗口 J-A 连续查询处理算法:IC 算法和 TC 算法,查询处理只需要保存滑动窗口数据,不需要额外的主存空间.聚集操作主要有 5 种:*COUNT*,*SUM*,*AVG*,MAX 和 MIN.其中,IC 算法可以处理 *COUNT*,*SUM* 和 *AVG* 聚集操作,而 TC 算法可以处理上面全部 5 种聚集操作.为了叙述方便,首先引入一些符号,见表 1.

Table 1 Symbol table

表 **1** 符号表

Symbol	Meaning
SWA, *SWB*	Sliding window corresponding to stream *A*,*B* respectively
A, β	Number of items in *SWA*, *SWB* respectively
T	Time size of sliding window *SWA*, *SWB*
λ_a	Arrival rate of stream *A*
M	Number of hash buckets in the hash table of sliding window
C	Cost of accessing one item in sliding window
Cw	Cost of writing one item in sliding window
σ	Selectivity of join operator
sizeA, *sizeB*	Item size of stream *A*, *B* respectively
$a\sigma b$	Concatenation of item *a* and item *b*
$\times$	Join operator

1.1 IC算法

由于聚集操作 *COUNT* 和 *SUM* 的实现算法一致,而 *AVG* 可以通过 *COUNT* 和 *SUM* 来求得,因此下面以处理 *COUNT* 聚集操作为例,来说明 IC 算法.IC 算法的基本思想是:采用增量式的方式处理滑动窗口 J-A 连续查询,利用第 K 次的 *COUNT* 值来计算第 K+1 次的 *COUNT* 值.数据流 A 或 B 每到来一个元组,均触发执行一次 IC 算法,计算一次 *COUNT* 值.IC 算法是平衡的,即对于数据流 A 和数据流 B 中的元组,IC 算法的执行过程相同.IC 算法基于下面定理 1.

定理 **1**. 设 A,B,C,D 为 4 种关系,其中关系 A 和 B 模式相同,C 与 D 模式相同,则下面的等式成立:

$|(B\times D)|=|(A\times C)|+|(B\backslash A)\times D|+|(D\backslash C)\times B|-|(B\backslash A)\times(D\backslash C)|-|(A\backslash B)\times(C\cap D)|-|(C\backslash D)\times(A\cap B)|-|(A\backslash B)\times(C\backslash D)|$.

(证明略).

设当前数据流 A,B 的滑动窗口快照为 SWA_{T_K},SWB_{T_K} 为当前滑动窗口 *SWA*,*SWB* 中最近到来的一个元组加入到滑动窗口的时刻,滑动窗口的时间长度为 T.在 T_J 时刻数据流 A 到来一个新的元组 f,触发执行 IC 算法.IC 算法的详细定义如下:

IC 算法.

输入:*SWA*,*SWB*,T_J,f,*CountValue*,T.

输出:*CountValue*.

(1) Insert f into *SWA*

(2) *SetA*←*null*; *SetB*←*null*;

(3) 根据 f 的时间戳,删除滑动窗口 *SWA*,*SWB* 中所有过期的元组,将 *SWA* 和 *SWB* 中过期的元组放到集合 *SetA* 和 *SetB* 中;

(4) *val*1←*COUNT*($\{f\}\times$*SWB*);

(5) *val*2←*COUNT*(*SetA*×*SWB*);

(6) $val3 \leftarrow COUNT(SetB \times (SWA - \{f\}))$;

(7) $val4 \leftarrow COUNT(SetA \times SetB)$;

(8) $CountValue \leftarrow CountValue + val1 - val2 - val3 - val4$;

(9) Return $CountValue$;

在 IC 算法中,函数 $COUNT(A)$的功能是计算集合 A 中元组的个数.在 f 到来之前,滑动窗口的快照为 SWA_{T_K}, SWB_{T_K}, IC 算法的(1)~(3)步对滑动窗口进行了更新.此后,滑动窗口的快照为 SWA_{T_J}, SWB_{T_J}, 已知当前的 $CountValue = |(SWA_{T_K} \times SWB_{T_K}|$,算法欲计算$|(SWA_{T_J} \times SWB_{T_J})|$.则由定理 1 可知:

$$|(SWA_{T_J} \times SWB_{T_J})| = |(SWA_{T_K} \times SWB_{T_K})| + |(SWA_{T_J} \setminus SWA_{T_K}) \times SWB_{T_J}| + |(SWB_{T_J} \setminus SWB_{T_K}) \times SWA_{T_J}| - |(SWA_{T_J} \setminus SWA_{T_K}) \times |(SWB_{T_J} \setminus SWB_{T_K})| - |(SWA_{T_K} \setminus SWA_{T_J}) \times (SWB_{T_K} \cap SWB_{T_J})| - |(SWB_{T_K} \setminus SWB_{T_J}) \times (SWA_{T_K} \cap SWA_{T_J})| - |(SWA_{T_K} \setminus SWA_{T_J}) \times (SWB_{T_K} \setminus SWB_{T_J})| \quad (1)$$

由 IC 算法定义可知: $SWA_{T_J} \setminus SWA_{T_K} = \{f\}$.由于数据流 B 没有新到的元组,因此, $SWB_{T_J} \subseteq SWB_{T_K}$, 故 $SWB_{T_J} \setminus SWB_{T_K} = \phi$. $SWA_{T_K} \setminus SWA_{T_J} = SetA$, $SWB_{T_K} \setminus SWB_{T_J} = SetB$.在算法第(3)步开始, $SWA_{T_K} \cap SWA_{T_J} = SWA \setminus \{f\}$, $SWB_{T_K} \cap SWB_{T_J} = SWB$.将这些等式代入到式(1)中,则有:

$$|SWA_{T_J} \times SWB_{T_J}| = |SWA_{T_K} \times SWB_{T_K}| + |\{f\} \times SWB| - |SetA \times SWB| - |SetB \times (SWA \setminus \{f\})| - |SetA \times SetB| \quad (2)$$

由上述分析可知,IC 算法能够计算出正确的 $COUNT$ 值.在 IC 算法中滑动窗口仍采用 HASH 表的方式组织.下面分析 IC 算法的时间和空间复杂度.IC 算法第(1)步、第(3)步的时间开销分别为 C_w,$(\alpha+\beta)C$.由于滑动窗口 SWB 有 M 个桶,平均每个桶中共有 β/M 个元组;算法第 4 步的时间开销为$(\beta/M) \times C$;算法第(5)步、第(6)步的时间开销分别为$|SetA| \times \beta/M \times C$,$|SetB| \times \alpha/M \times C$;算法第(7)步中的连接操作,这里采用的是循环嵌套连接算法,时间开销为$|SetA| \times |SetB| \times C$,则算法总的时间开销为

$$C_w + (\alpha+\beta)C + (\beta/M) \times C + |SetA| \times \beta/M \times C + |SetB| \times \alpha/M \times C + |SetA| \times |SetB| \times C \approx (\alpha + \beta + \beta/M + |SetA| \times \beta/M + |SetB| \times \alpha/M + |SetA| \times |SetB|) \times C \quad (3)$$

虽然 IC 算法需要处理多次连接操作,但是,由于每次参加连接的数据量均很小,算法具有很高的效率.IC 算法除了滑动窗口之外,需要两个集合 $SetA$,$SetB$ 存放过期的元组,算法的空间开销为 $O(\alpha \times sizeA + \beta \times sizeB + |SetA| \times sizeA + |SetB| \times sizeB)$.一般情况下,$SetA$ 和 $SetB$ 占用的空间很小,因而 IC 算法具有很好的空间复杂性.

1.2 TC算法

下面首先以 $COUNT$ 为例,说明 TC 算法如何处理 $COUNT$,SUM 和 AVG 聚集操作;然后以 MAX 为例,说明 TC 算法如何处理 MAX 和 MIN 聚集操作.

称处理 $COUNT$ 聚集操作的 TC 算法为 TC-COUNT 算法.在 TC-COUNT 算法中,滑动窗口 SWA,SWB 中每个元组均包含一个 CV 属性.设 SWA,SWB 的连接属性为 a,对于 SWA 中的一个元组 f,定义 $f.CV = |\{g | g \in SWB, g.timestamp > f.timestamp, g.a = f.a\}|$.需要强调的是,$f.CV$ 的值不是 SWB 中所有与 f 匹配的元组的个数,而是 SWB 中所有在 f 之后到来的并且与 f 匹配的那些元组的个数.利用 CV 属性值计算 $COUNT$ 值时只需要执行一次连接.下面给出证明.

定理 **2**. 设滑动窗口 SWA,SWB 长度为 T;$\forall f \in SWA$(或 SWB);$f.CV = |\{g | g \in SWB$(或 SWA);$g.timestamp > f.timestamp, g.a = f.a\}|$,其中 a 为连接属性,$timestamp$ 属性值为这个元组加入到滑动窗口的时刻.若在某个更新时刻 T_K,滑动窗口 SWB 删除了元组 g,则$\forall f \in SWA_{T_K}$, g 的删除不会改变 $f.CV$ 的值.

证明:假设在 T_K 时刻,滑动窗口 SWB 中元组 g 的删除修改了 SWA_{T_K} 中元组 f 的 CV 属性的值,则由 CV 的定义可知,有 $g.a = f.a$ 且 $g.timestamp > f.timestamp$ 成立.由于元组 g 被删除,说明 $T_K - g.timestamp > T$,则可知 $T_K - f.timestamp > T$,从而元组 $f \notin SWA_{T_K}$.这与 $f \in SWA_{T_K}$ 相矛盾.因此假设不成立,g 的删除不会改变 $f.CV$ 的值. □

由定理 2 可知,在滑动窗口更新时,不需要执行过期元组与滑动窗口的连接来修改 CV 属性的值.下面给出如何利用 CV 属性值计算滑动窗口连接的 $COUNT$ 值.

定理 **3**. 设滑动窗口 SWA,SWB 的大小为 T,$\forall f \in SWA$(或 SWB),$f.CV = |\{g | g \in SWB$(或 SWA),$g.timestamp$

>*f.timestamp*,*g.a*==*f.a*}|,则有:

$$|SWA \times SWB| = \sum_{\forall f \in SWA,SWB} f.CV .$$

证明:$\forall r \in SWA \times SWB$,则 $r=f \circ g$,其中,$f \in SWA, g \in SWB$.当产生连接结果 r 时,由 CV 的定义可知,如果 *f.timestamp*>*g.timestamp*,则 *g.CV*=*g.CV*+1,反之 *f.CV*=*f.CV*+1.即 $\forall r \in SWA \times SWB$,它对 $\sum_{\forall f \in SWA,SWB} f.CV$ 值的贡献为 1.

由此可得: $|SWA \times SWB| = \sum_{\forall f \in SWA,SWB} f.CV$. □

TC-COUNT 算法应用定理 2、定理 3 来计算 *COUNT* 值.TC-COUNT 算法也是平衡的,设在 T_J 时刻数据流 *A* 到来一个新的元组 *f*,TC-COUNT 算法定义如下,其中滑动窗口使用 HASH 表组织.

TC-COUNT 算法.

输入:*SWA*,*SWB*,T_j,*f*,*T*.

输出:*CountValue*.

(1) *f.CV*←0;*CountValue*←0;
(2) Insert *f* into *SWA*;
(3) 删除滑动窗口 *SWA* 和 *SWB* 中过期的元组
(4) 找到 *SWB* 中与 *f* 对应的 HASH 桶 *B*
(5) FOR ∀ tuple *g*∈*B* DO
(6) IF *g.a*==*f.a*
(7) *g.CV*++;
(8) FOR ∀*k*∈*SWA* DO
(9) *CountValue*←*CountValue*+*k.CV*;
(10) FOR ∀*g*∈*SWB* DO
(11) *CountValue*←*CountValue*+*g.CV*;
(12) Return *CountValue*;

称处理 MAX 聚集操作的 TC 算法为 TC-MAX 算法.TC-MAX 算法的思想与 TC-COUNT 算法类似,滑动窗口 *SWA*,*SWB* 中每个元组均包含一个 *MV* 属性.设 *SWA*,*SWB* 的连接属性为 *a*,聚集属性为数据流 *B* 的 *c* 属性,为了叙述方便,设属性 *B.c* 的值域为[0,*U*],则 $\forall f \in SWA$,定义 *f.MV*=MAX{*g.c*|*g*∈*SWB*,*g.timestamp*>*f.timestamp*∧*g.a*=*f.a*};$\forall g \in SWB$,定义:

$$g.MV = \begin{cases} g.c, & \exists f \in SWA, f.timestamp > g.timestamp \wedge g.a = f.a \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

滑动窗口内每个元组的 *MV* 属性值,可以在数据流新数据到来时进行计算更新.与定理 2、定理 3 的证明类似,可以证明:

$$\text{MAX}\{f.c|f \in SWA \times SWB\} = \text{MAX}\{f.MV|f \in SWA,SWB\} \quad (5)$$

设在 T_J 时刻数据流到来一个新的元组 *f*,TC-MAX 算法的定义如下,其中滑动窗口使用 HASH 表组织.

TC-MAX 算法.

输入:*WA*,*SWB*,T_j,*f*,*T*.

输出:Max*Value*.

(1) *f.MV*←0;Max*Value*←0;
(2) 删除滑动窗口 *SWA* 和 *SWB* 中过期的元组
(3) IF *f*∈*SWA*
(4) Insert *f* into *SWA*;
(5) 找到 *SWB* 中与 *f* 对应的 HASH 桶 *B*

(6) FOR ∀$g\in B$ DO
(7) IF $g.a==f.a$
(8) $g.MV\leftarrow g.c$;
(9) ELSE
(10) Insert f into SWB;
(11) 找到 SWA 中与 f 对应的 HASH 桶 B
(12) FOR ∀$g\in B$ DO
(13) IF $g.a==f.a$
(14) $g.MV\leftarrow$MAX($g.MV,f.c$);
(15) FOR ∀$k\in SWA$ DO
(16) Max$Value\leftarrow$MAX(Max$Value,k.MV$);
(17) FOR ∀$k\in SWB$ DO
(18) Max$Value\leftarrow$MAX(Max$Value,k.MV$);
(19) Return Max$Value$;

由算法定义可知,TC-COUNT 算法和 TC-MAX 算法的执行过程相似,只是对 CV 和 MV 属性值的更新方法不同,因此可以通过维护每个元组的 CV 和 MV 属性值,来处理同时包含 $COUNT$ 和 MAX 聚集操作的滑动窗口连续 J-A 查询,亦即 TC 算法可以处理同时包含 5 种聚集操作的滑动窗口连续 J-A 查询.

由于 TC-COUNT 算法与 TC-MAX 算法的时间复杂度和空间复杂度相同,下面以 TC-COUNT 算法为例来进行分析.在 TC-COUNT 算法的第(1)步为初始化的过程,时间开销仅为两个变量赋值的时间,我们忽略了这一步的时间开销.算法第(2)步的时间开销分别为 C_w.算法第(3)步要遍历 SWA,SWB,时间开销为$(\alpha+\beta)C$.由于滑动窗口 SWB 有 M 个桶,平均每个桶中共有β/M 个元组,算法第(5)~(7)步的时间开销为$(\beta/M)\times C$.算法第(8)~(11)步的时间开销为$(\alpha+\beta)C$,则算法总的时间开销为

$$C_w+2(\alpha+\beta)C+\beta/M\times C\approx(2\alpha+2\beta+\beta/M)\times C \quad (6)$$

TC-COUNT 算法为每个元组分配一个 CV 属性,其类型定义为整型.因此 TC-COUNT 算法的空间开销为

$$O(\alpha\times sizeA+\beta\times sizeB+(\alpha+\beta)\times SizeOf(Int)) \quad (7)$$

2 复杂滑动窗口 J-A 连续查询的处理算法

本节将扩展第 1 节中的算法来处理复杂的滑动窗口 J-A 连续查询,包括分组(含 group by)的滑动窗口 J-A 连续查询和多滑动窗口 J-A 连续查询.

2.1 分组滑动窗口J-A连续查询的处理算法

只需要对前面提出的算法进行简单的修改,即可以支持含分组操作的滑动窗口 J-A 查询.这里,以计算聚集函数 $COUNT$ 的 IC 算法为例来进行说明,其思想也可用于 TC 算法.称支持分组操作的 IC 算法为 GIC 算法.GIC 算法将每个分组的 $COUNT$ 值保存在链表 $Glist$ 的各个节点中,每当数据流上到来一个新的元组时,算法更新滑动窗口产生集合 $SetA$,$SetB$.计算连接$\{f\}\times SWB$,$SetA\times SWB$,$SetB\times(SWA\backslash\{f\})$,$SetA\times SetB$.对于每个连接结果,利用其分组属性值扫描 $Glist$ 链表,然后根据 IC 算法的规则更新其对应分组的 $COUNT$ 值.最后,将 $Glist$ 链表的内容输出.若分组操作含有 HAVING 子句,则在输出 $Glist$ 的内容之前,利用 HAVING 子句中的谓词条件对结果进行过滤.

2.2 多滑动窗口J-A连续查询的处理算法

Viglas 等人提出了一种多滑动窗口连接查询的处理方法,他们对 SHJ 算法进行了扩展,实现了一个能够处理多个输入的连接操作算子 MJoin[7].如图 1 所示,当数据流 S_1 到来一条数据 f 后,MJoin 首先更新所有的滑动窗口,然后用 f 扫描数据流 S_2 的滑动窗口,再用得到的连接结果扫描 S_3 的滑动窗口,重复这样的操作,直到扫描完所

有的滑动窗口,将连接结果输出.MJoin 算法根据连接的选择性决定扫描滑动窗口的顺序,先扫描选择性低的连接操作所对应的滑动窗口.

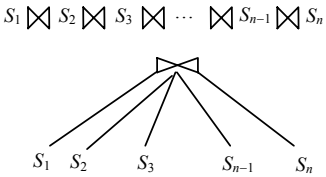

Fig.1 Mjoin operator

图1 MJoin操作符

基于 MJoin 的思想,可以实现多滑动窗口 J-A 连续查询处理算法.下面介绍如何扩展 TC-COUNT 算法来处理多滑动窗口 J-A 连续查询,称扩展后的算法为 MTC-COUNT 算法.当数据流 S_1 到来一个元组 f,利用 MJoin 算法得到一组滑动窗口连接结果.对于其中每个连接结果 r,$r=f\circ g_2\circ\ldots\circ g_n$,其中 $g_i\in S_i$,$2\le i\le n$.连接结果 r 的有效期取决于元组 $g_2\ldots g_n$ 中最早到来的元组 g_k,若在某个时刻元组 g_k 过期,则连接结果 r 也应同时过期.与 TC-COUNT 算法相同,在 MTC-COUNT 算法中,每个元组也含有一个 *CV* 属性,连接结果 r 对 *COUNT* 值的贡献是 1,则将元组 g_k 的 *CV* 属性值加 1,则由定理 3 可知:

$$|S_1\times S_2\times\ldots\times S_n|=\sum_{i=1}^{n}\sum_{g\in S_i} g.cv \tag{8}$$

将所有滑动窗口中元组的 *CV* 属性值相加,即为多路滑动窗口连接结果的 *COUNT* 值.

3 实验结果及分析

实验中,算法使用 Visual C++实现,实验的硬件环境为:CPU 主频 2.4G HZ、主存 256MB.同时,实现了利用查询操作树处理滑动窗口 J-A 连续查询的 PC 算法,与本文提出的 IC 和 TC 算法进行了对比.第 3.1 节测试了简单滑动窗口 J-A 连续查询处理算法性能;第 3.2 节测试了复杂滑动窗口 J-A 连续查询处理算法的性能.

3.1 简单连续J-A查询处理算法的性能实验

本节的实验测试了 PC,IC 和 TC 算法的执行时间以及算法的内存开销.实验数据为随机生成的 2 个数据流的数据,其中每个元组大小为 18 字节.实验中数据流 *A*,*B* 的参数选择是一致的.实验分别考察了两种情况下算法的性能:

(1) 固定流速、变化滑动窗口大小的情况;

(2) 变化流速、固定滑动窗口大小的情况.

3.1.1 固定流速变化滑动窗口大小

在数据流流速为 100 字节/秒,滑动窗口大小变化的情况下,3 种算法平均执行时间如图 2 所示.其中 IC 算法的性能最好,而 PC 算法的性能最差.PC 算法在主存保存连接结果,需要申请新的内存空间并进行大量的内存写操作,算法的时间开销很大.IC 算法的性能优于 TC 算法,这是因为在数据流流速均匀的情况下,每当数据流到来一个新的元组,滑动窗口 *SWA*,*SWB* 中过期的元组也均为一个,即 *SetA*,*SetB* 的大小为 1,因而 IC 算法的性能要优于 TC 算法.如图 2 所示,PC 算法需要读写中间连接结果队列,因而算法的平均执行时间受滑动窗口大小的影响最大;TC 算法需要遍历两个滑动窗口计算 *COUNT* 值,其平均执行时间也随着滑动窗口变大而缓慢增加;IC 算法的平均执行时间受滑动窗口变化的影响最小.图 3 给出了 3 种算法内存的使用情况.PC 算法需要保存连接的结果.由于只关心连接结果的个数而不关心连接结果的具体值,因此我们在实现 PC 算法时,每个连接结果只保留了元组 ID 和时间戳两个属性,大小为 4 字节.由图 3 可以看出,PC 算法的空间开销远远大于其他算法;TC 算法需要每个元组有一个 *CV* 属性,因而,其空间开销稍大于 IC 算法;而 IC 算法的空间开销最小.

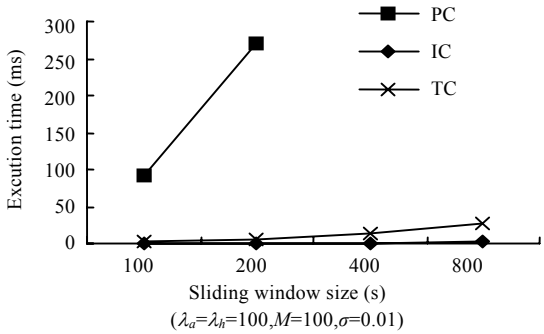

Fig.2 The effect of varying sliding window size on the algorithms execution time

图 2 变化的滑动窗口大小对算法执行时间的影响

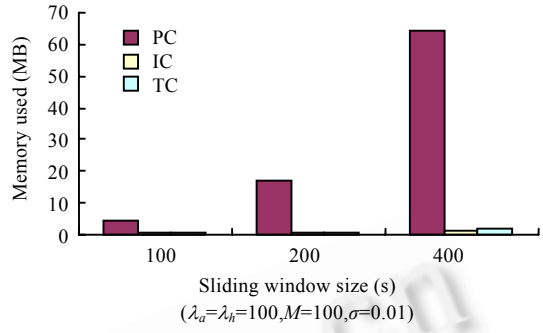

Fig.3 The effect of varying sliding window size on the algorithms memory requirement

图 3 变化的滑动窗口大小对算法内存需求的影响

3.1.2 固定滑动窗口大小变化流速

在本节的实验中,滑动窗口大小为 20 秒,HASH 桶数 M 为 20.在初始条件下,滑动窗口中保存了 20 秒的数据,在这 20 秒内,数据流的流速为 500 字节/秒.在随后的 5 秒中,数据流的流速服从 Zipf 分布,数据流在第 i 秒的流速为 $\lambda_i=\frac{1}{i^2}\times 500$ 字节/秒,其中 $1\le i\le 5$.算法的性能对比如图 4 所示:随着时间的推移,数据流流速变慢,滑动窗口中元组数减少,PC 算法、TC 算法平均执行时间是单调递减的;IC 算法的平均执行时间呈上升趋势,原因在于:当数据流流速变慢时,每次滑动窗口更新产生的过期元组增多,集合 $setA$,$setB$ 内的元组增多,从而导致 IC 算法的效率降低.由图 4 可以看出,IC 算法在前两秒的性能仍好与 TC 算法;在其后的几秒中,TC 算法的性能要优于 IC 算法;IC 算法和 TC 算法的性能仍然远好于 PC 算法.图 5 为 3 种算法的空间开销对比:IC 算法和 TC 算法的空间开销远小于 PC 算法.随着流速的降低,滑动窗口内的数据减少,因而 3 种算法的空间开销呈降低趋势.

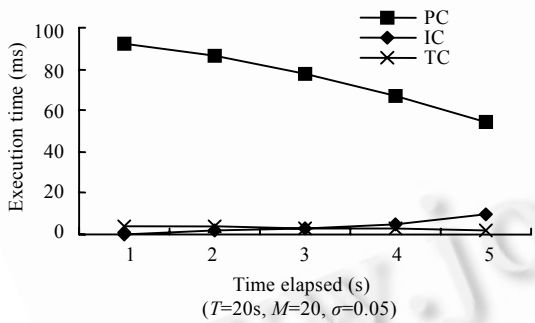

Fig.4 The effect of varying stream rate on the algorithms execution time

图 4 变化的数据流流速对算法执行时间的影响

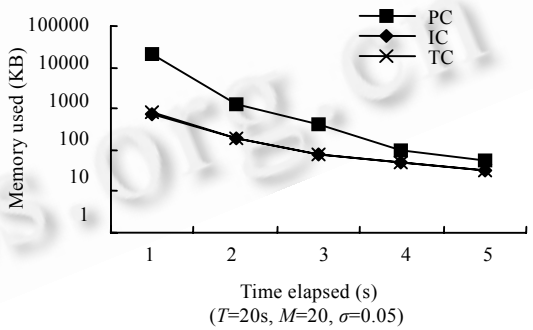

Fig.5 The effect of varying stream rate on the algorithms memory requirement

图 5 变化的数据流流速对算法内存需求的影响

3.2 复杂滑动窗口**J-A**连续查询处理算法性能实验

本节实验测试了复杂滑动窗口 J-A 连续查询处理算法的性能,包括含有分组操作的滑动窗口 J-A 连续查询处理算法和多滑动窗口 J-A 连续查询处理算法.

3.2.1 分组滑动窗口 J-A 连续查询处理算法的性能实验

实现了聚集函数为 *COUNT* 的分组滑动窗口 J-A 连续查询处理算法 GPC,GIC 和 GTC.图 6 是 3 种算法性能对比实验结果.其中,数据流的流速为每秒 100 个,连接选择性为 0.01.在数据流流速固定、滑动窗口大小变化的情况下,GIC 算法的性能最好,而 GPC 算法的性能最差.

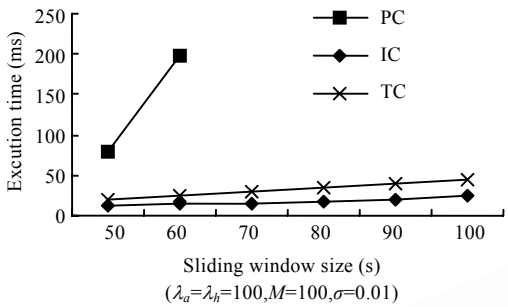

Fig.6 The effect of sliding windows size on the algorithms that process sliding window J-A query contained groupby

图 6 滑动窗口大小对处理包含分组操作的滑动窗口 J-A 查询算法的影响

3.2.2 多数据流连续 J-A 查询处理算法的性能实验

应用 PC 算法和 TC 算法的思想,实现了聚集函数为 *COUNT* 的多滑动窗口 J-A 连续查询处理算法——MPC 算法和 MTC 算法.实验中,每个滑动窗口大小均为 10 秒,数据流流速为 100 字节/秒,连接的选择性均为 0.01.性能对比实验结果如图 7 所示:MTC 算法的性能远优于 MPC 算法.随着参加连接的数据流个数的增加,MPC 算法产生的中间连接结果急剧增多,算法性能迅速下降,而 MTC 算法性能下降缓慢.

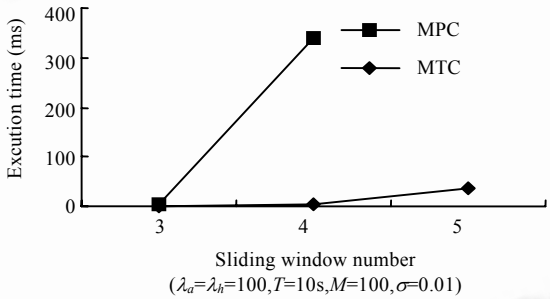

Fig.7 The effect of sliding window number on the algorithms that process sliding window J-A query contained multi-way join

图 7 滑动窗口个数对处理包含多路连接的滑动窗口 J-A 查询算法的影响

4 总 结

滑动窗口 J-A 连续查询是数据流上一类常用的查询类型.一种直观的处理方法是建立查询操作树,以流水线的方式执行连接操作和聚集操作.这种处理方法需要保存滑动窗口连接的结果,查询处理的空间开销为 $O(\alpha\times\beta)$,使用了大量的主存空间.在数据流的查询处理中,内存是最重要的计算资源.本文提出两种滑动窗口 J-A 连续查询的处理算法——IC 算法和 TC 算法,使得查询处理的空间开销降为 $O(\alpha+\beta)$.理论分析和实验结果同时表明,IC 算法和 TC 算法的执行效率也要远远好于查询操作树方法的执行效率.

References:

[1] Motwani R, Widom J, Arasu A, Babcock B, Babu S, Datar M, Manku G, Olston C, Rosenstein J, Varma R. Query processing, resource management, and approximation in a data stream management system. In: Stonebraker M, Gray J, Dewitt D, eds. Proc. of the 1st Biennial Conf. on Innovative Data Systems Research. Asilomar: Online Proceedings, 2003. 245−256.

[2] Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Seidman G, Stonebraker M, Tatbul N, Zdonik S. Monitoring streams—a new class of data management applications. In: Lochovsky FH, Wang S, Papadias D, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 215−226.

[3] 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.

[4] 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.

[5] Kang J, Naughton JF, Viglas SD. Evaluating window joins over unbounded streams. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Bangalore: IEEE Computer Society, 2003. 341−352.

[6] Golab L, Ozsu MT. Processing sliding window multi-joins in continuous queries over data streams. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 500−511.

[7] Viglas S, Naughton J, Burger J. Maximizing the output rate of multi-join queries over streaming information sources. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 285−296.

[8] Arasu A, Widom J. Resource sharing in continuous sliding-window aggregates. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 336−347.

王伟平(1975－),男,吉林舒兰人,博士生,主要研究领域为数据流查询处理.

张冬冬(1976－),男,博士生,主要研究领域为分布式数据流查询处理.

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.

郭龙江(1973－),男,博士生,主要研究领域为数据流挖掘.