数据流中一种适应性查询优化方法

董楠楠 陆 岩 宋宝燕
(辽宁大学信息学院 沈阳 110036)

摘 要 对数据流中的查询处理机制进行了深入的研究,从内存使用量和查询的实时性两方面进行综合考虑,提出了一种基于多因素的动态查询优化及调度策略MultiFactor,它根据各操作符消耗系统中元组数量的快慢来动态调整操作符调度次序,按查询的截止时间来确定各操作符调度时间,同时提出了多流连接查询的调度方法。给出了MultiFactor适应性优化的时机以及调整策略。

关键词 数据流,查询优化,调度策略,适应性查询
中图法分类号 TP311.13 **文献标识码** DB

Adaptive Query Optimum Scheduling Strategy on Data Streams

DONG Nan-nan LU Yan SONG Bao-yan
(School of Information, Liaoning University, Shenyang 110036, China)

Abstract Query processing mechanisms of data stream were investigated in this paper and we proposed a dynamic optimum strategy, which is based on multiple factors, namely MultiFactor. This strategy decides a scheduling sequence by the number of tuples which consumed by a operator per unit time. Scheduling time-slices are decided by deadline of a query. Also the paper proposed a Scheduling method of multi-stream join queries. This paper gave the adaptive optimization of the timing and the adjust strategy of MultiFactor.

Keywords Data stream, Query optimum, Scheduling strategy, Adaptive query

1 引言

近年来,信息处理技术的应用领域得到了广泛拓展,流式应用得到人们极大关注。这种应用的处理强调实时性,同时需要特别考虑内存的限制,因而数据流处理技术成为数据库研究领域的又一热点[1-3]。

数据流中查询多为连续查询,查询一经注册,该查询就一直执行,除非人为加以制止。在一个查询执行期间,随着时间的推移,系统的运行环境,如CPU的占用率、内存使用情况等将不断地发生变化;数据流本身的一些特征,如操作符的选择率和数据流的速率等也在不断地发生改变。为提高查询性能,针对不断变化的系统环境和数据流特征,适时调整查询处理策略是非常必要的。因此,适应性查询处理技术应运而生,并成为数据流查询处理技术中的难点[4-6]。

本文针对数据流中的查询处理机制进行了深入的研究,从内存使用量和查询的实时性两方面进行综合考虑,提出了一种基于多因素的动态优化及调度策略MultiFactor。该策略根据各操作符消耗系统中元组数量的快慢来动态调整操作符调度次序,按查询的截止时间来确定各操作符调度时间。给出了适应性优化的时机、Multi Factor策略的相应算法,并进行了性能测试。

2 MultiFactor

MultiFactor一方面可以适时调整查询计划,另一方面可以按查询计划进行非等值时间片轮转调度。

2.1 MultiFactor查询优化策略

传统的操作符调度方法的操作符调度次序是固定的(即查询计划是固定的),主要考虑因素是操作符调度的时机。在数据流系统中,由于查询是连续查询,因此随着系统执行环境、流自身特性的改变,适时改变查询计划,并同时考虑操作符调度的时机是非常必要的。MultiFactor根据各操作符消耗元组数量的快慢来确定操作符调度序列。单位时间内消耗元组数量越多,则赋予该操作符的优先级越高,其越优先得到调度,从而使内存的需求最低。

设一个查询仅有3个选择操作符OP_1,OP_2,OP_3。在各个操作符前都设置一个队列来存储到达的元组。各操作符在单位时间内处理元组的数量分别为n_1,n_2,n_3,选择率分别为S_1,S_2,S_3,为各个操作符分配的时间分别为t_1,t_2,t_3,数据流的平均流速为V。这些参数值均可通过一定时间的统计得出平均值。

我们假设数据流的平均流速要大于OP_1对元组的处理速度,即$V>n_1$。

下面是分配优先级的具体过程:

1) 初始OP_1处理t_1时间单元,处理的元组数量为$n_1 * t_1$,同时到达OP_1前队列中的元组数量为$M_1=V * t_1$,在OP_2前队列中增加的元组数量为$n_1 * S_1 * t_1$。此时,系统中总的元组数量$M_2=M_1-n_1 * t_1+n_1 * S_1 * t_1=M_1-(1-S_1) * n_1$

到稿日期:2010-06-12 返修日期:2010-10-02 本文受国家自然科学基金项目(60873068,60703068)资助。
董楠楠(1981-),女,硕士生,助理实验师,主要研究方向为数据流技术,E-mail:dong.n.n@163.com;陆 岩(1981-),女,硕士生,主要研究方向为数据流技术;宋宝燕(1965-),女,博士,教授,主要研究方向为数据库、数据流技术。

$*t_1$。

2) 经过 OP_2 处理后减少的元组的数量为 n_2*t_2，在 OP_3 前产生的元组的数量为 $n_2*t_2*S_2$。而同时在 OP_1 前增加的元组数量为 $V*t_2$。此时，系统中元组的总数量为 $M_3=M_2-n_2*t_2+n_2*t_2*S_2+V*t_2=M_2-(1-S_2)*n_2*t_2+V*t_2$。

3) 经过 OP_3 处理后减少的元组数量为 n_3*t_3，由于 OP_3 为最后一个操作，因此由 OP_3 产生的元组不占用内存空间，直接以流的形式输出。此时系统中的元组数量 $M_4=M_3-n_3*t_3+V*t_3=M_3-(n_3-V)*t_3$。

MultiFactor 根据操作符单位时间消耗元组数量的快慢来确定操作符调度次序。操作符优先级计算为：

$$PRI_{(OP_1)}=\frac{M_1-M_2}{t_1}=(1-S_1)*n_1 \tag{1}$$

$$PRI_{(OP_2)}=\frac{M_2-M_3}{t_2}=(1-S_2)*n_2-V \tag{2}$$

$$PRI_{(OP_3)}=\frac{M_3-M_4}{t_3}=n_3-V \tag{3}$$

从式(1)可以看出，影响 OP_1 优先级的因素为选择度 S_1 和单位时间处理元组个数 n_1。对于一个操作符而言，其选择率越小，经其处理后产生的元组数量越少，内存的需求量也就越低；其单位时间处理元组的速度越快，则其处理相同数量的元组所用的时间越少，从而降低了查询的响应时间。

从式(2)可以看出，OP_2 的优先级除受到选择度和单位时间处理元组个数的影响之外，还受到输入流速 V 的影响。OP_2 优先级与流速成反比，当数据流的流速突然增大时，OP_2 的优先级将降低。这样 OP_1 将有更多的可能性被系统调度，从而可以尽快地处理掉突发到达的大量元组。

从式(3)可以看出，OP_3 优先级的变化不受选择度的影响，这是因为 OP_3 是最后一个操作符，它产生的元组将作为结果从系统中输出，不再占用内存空间。而影响系统内存使用的是操作符的选择度。

2.2 连接

连接操作是数据流查询中不可缺少而又相对复杂的操作，它的执行效率将直接影响到系统的性能。我们将上面对单流查询的优化策略也扩展到含有连接操作符的查询中。

2.2.1 MultiFactor 的两流连接

设流 R 和流 S 的平均流速为 V_R，V_S，R 流上有一选择操作符 OP_1，S 流上有两个选择操作符 OP_2 和 OP_3。具体的计算过程如下：

1) 初始从 R 流到达 OP_1 前队列中的元组数量为 $M_1=VR*t_1$。

2) 经过 OP_1 处理 t_1 时间单元后，处理的元组数量为 n_1*t_1，产生元组数量为 n_1*S_1*t。同时，从 S 流到达 OP_2 前队列中的元组数量为 V_S*t_1。此时系统中元组的总数量为 $M_2=M_1+V_S*t_1-n_1*t_1+n_1*S_1*t_1$。

3) 经过 OP_2 处理 t_2 时间单元后减少元组的数量为 n_2*t_2，产生元组的数量为 $n_2*t_2*S_2$。同时，R 流到达元组数量为 V_R*t_2。此时系统中元组的数量为 $M_3=M_2+V_R*t_2-n_2*t_2+n_2*t_2*S_2$。

4) 经过 OP_3 处理后减少的元组数量为 n_3*t_3，R 流到达元组的数量为 V_R*t_3，同时 S 流到达 OP_2 前队列中的元组数量为 V_S*t_3。由于 OP_3 为最后一个操作，因此由 OP_3 产生的元组不占用内存空间，直接以流的形式输出。此时系统中元组的数量为 $M_4=M_3+V_R*t_3+V_S*t_3-n_3*t$。

将不同时刻系统中的元组数量(M_i)以及所用的时间(t_i)带入优先级计算公式，便可以得到各个操作符的优先级。

从各个操作符优先级的计算公式(式(4)－式(6))可以看出，各个操作符的优先级除受到选择度和单位时间处理元组个数的影响外，还受到输入流速 V_R 和 V_S 的影响。

$$PRI_{(OP_1)}=\frac{M_1-M_2}{t_1}=(1-S_1)*n_1-V_S \tag{4}$$

$$PRI_{(OP_2)}=\frac{M_2-M_3}{t_2}=(1-S_2)*n_2-V_R \tag{5}$$

$$PRI_{(OP_3)}=\frac{M_3-M_4}{t_3}=n_3-V_R-V_S \tag{6}$$

2.2.2 MultiFactor 的多流连接

数据流上的一个连接操作可以看作是一个双向连接的过程，本文将一个流 R 和流 S 的连接操作分成两个组成部分，一个是流 R 到流 S 的单向连接，另一个是流 S 到流 R 的单向连接。为了讨论方便，把流 R 到流 S 的单向连接用 $R\succ S$ 表示，流 S 到流 R 的单向连接用 $R\prec S$ 表示。假设每个时间单位出现在流 S 和流 R 上(滑动窗口)的平均元组数量为 V_S 和 V_R，连接操作在流 R 和流 S 的滑动窗口间进行，它们所处理的元组的时间戳的时间间隔为 t，则连接操作符将处理的元组数量为 $t*(V_R+V_S)$，产生的元组数量为 $t*V_R*S_{(S)}+t*V_S*S_{(R)}$，其中 $S_{(S)}$ 表示用流 R 中的一个元组和流 S 中的元组(在一定的滑动窗口之内)做连接，平均能消耗掉流 S 中元组的选择度，即为 $R\succ S$ 的选择度，$S_{(R)}$ 同理可得。这样可以得到连接操作的选择度为 $S_{join}=[V_R*S_{(S)}+V_S*S_{(R)}]/(V_R+V_S)$。用 n_R 和 n_S 分别表示连接操作符在流 R 和流 S 上单位时间处理元组的个数，则系统在时间间隔 t 内处理元组的数量为 $t*(V_R+V_S)$，处理这些元组总共所用时间为 $t*V_R/n_R+t*V_S/n_S$，所以连接操作符单位时间处理元组个数为 $n_{join}=n_R*n_S*(V_R+V_S)/(V_R*n_S+V_S*n_R)$。

$$t_1=\frac{T}{1+\frac{n_1*S_1}{n_2}+\cdots+\frac{n_1*S_1*S_2*\cdots*S_{i-1}}{n_i}} \tag{7}$$

得到连接操作符选择度和处理元组速度的计算公式之后，便可以将前面介绍的方法扩展到多流连接的查询中。其优先级的计算仍可以采用上面的方法进行。对于这个问题的处理可以采用嵌套的方式。如图 1 所示，嵌套中的每个部分使用的算法与两个流的连接算法相同。

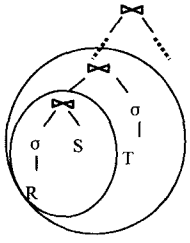

图 1 多连接示意图

2.3 MultiFactor 调度方法

通过 MultiFactor 查询优化策略得到查询计划之后，将开始查询执行。MultiFactor 调度是一种基于时间片轮转的调度方法。

在 MultiFactor 中，操作符消耗元组的速度越快，其优先

级越高。我们给这样的操作符较长的时间片，以减少对系统内存资源的使用。所以，对于一个查询计划的第一个操作符，应为其分配尽量多的时间。但是考虑到用户对查询响应时间的要求(查询截止时间)，在数据流不断到达时不能只调度第一个操作符，其它的操作也应得到相应的调度。在数据流系统中，操作符调度的频繁切换必定会消耗大量的系统时间。所以我们为每个操作符尽量收集更多的元组，以使这些操作符在所分配的时间内将其队列中的元组一次性处理完成。

设一个查询中有 3 个操作符 OP_1，OP_2，OP_3。系统首先调度 OP_1，给 OP_1 尽量多的调度时间，但是必须保证在查询截止时间到达之前将 OP_1 产生的元组经由其它操作符处理结束。

1) 设初始时分配给 OP_1 的时间片为 t_1 时间单元。

2) 经过 OP_1 处理后在 OP_2 前队列中产生的元组数量为 $t_1 * S_1 * n_1$，将这些元组全部处理完应分给 OP_2 的时间片为 $t_2 = t_1 * S_1 * n_1 / n_2$。

3) OP_2 处理了 t_2 时间后，在 OP_3 的队列中产生的元组数量为 $n_1 * S_1 * S_2 * t_1$，将这些元组全部处理完所用时间为 $t_3 = n_1 * S_1 * S_2 * t_1 / n_3$。

从以上各式中可以看出，各个操作符时间片的分配均依赖于分配给 OP_1 的时间片 t_1。而 t_1 的确定主要考虑的是查询的截止时间和当前数据流的情况，设查询的截止时间为 T，则 $T = t_1 + t_2 + t_3$，而 $t_2 = t_1 * S_1 * n_1 / n_2$，$t_3 = n_1 * S_1 * S_2 * t_1 / n_3$，将 t_1 代入其它式子即可算出分配给各个操作符的时间片。

从上面的计算中可以看出，t_1 对于 OP_1 的调度时间而言是一个最大值。也就是说，当数据流不断有元组到达时，OP_1 将不断被系统调度，但是调度时间不应超过 t_1。由于数据流的突发性，若在 t_1 时间到达之前，OP_1 就不再有元组到达了，应该提前开始调度 OP_2，以减少查询的响应时间。

2.4 算法实现

本文第 2.1 和 2.2 节详细介绍了 MultiFactor 查询优化策略，并给出了确定一个查询中各个操作符优先级的具体公式。MultiFactor 优化策略的具体实现如算法 1 所示。

算法 1 MultiFactor 查询优化算法

```
输入:随机数据流 S(或流 S 与流 R)
输出:各个操作符的优先级
步骤:
1)init();/*对各个参数进行初始化*/
2)if(Join operator exist in the query)
/*如果查询中存在连接操作符*/
3)for(int i=1;i<Onumber;i++)
4){ OP[i].ComputeJoinM();}
5)else
6)for(int i=1;i<Onumber;i++)
7){ OP[i].ComputeM();}
8)public void computePriority()
                    /*计算各个操作符的优先级*/
9){ for(int i=1;i<Onumber;i++)
10)Priority[i]=(m[i]-m[i+1])/(t[i]-t[i-1]);}
11)setPriority(); /*设置优先级*/
12)End
```

第 2.3 节分析了计算各个操作符时间片的方法并给出了具体公式。其具体实现如算法 2 所示。

算法 2 MultiFactor 调度时间分配算法

```
输入:随机数据流 S(或流 S 与流 R)
输出:各个操作符的时间片
步骤:
1)init(); /*对各个参数进行初始化*/
2)for(int i=1;i<=Onumber;i++)
3){ if(OP[i].state==JOIN) /*该操作是连接操作*/
4)  {
5)      sjoin=computeJoin_s( );
6)      njoin=computeJoin_n();
7)   }}
8)if(v>n[1])
  /*数据流的到达速率大于 OP[1]的处理速率*/
9)ComputeTime1();
10)else
  /*数据流的到达速率小于 OP[1]的处理速率*/
11)  ComputeTime2();
12)End
```

3 适应性优化的时机

由于数据流本身的特性，计算时所需的参数(操作符的选择率、数据流的流速等)都不是一成不变的，因此各个操作符的执行次序也需要随时间动态地改变。但是考虑到系统性能，对执行次序的计算不能过于频繁，不能因为参数的些许变化就重新进行计算，因此需要一个对各个参数进行统计并进行计算的时间限度。为此我们设定了一个时间限度变量 τ。

在系统初始状态，调度器按照原始的操作符顺序进行调度。在运行一段时间后，利用搜集的各个系统参数执行对各个操作符排序的计算。我们将连续查询的执行时间 t 看成由多个 τ 组成，并且在各个 τ 内对系统参数的收集是相互独立的，然后利用第 i 个 τ 所收集到的参数数据为第 $i+1$ 个 τ 进行计算。

由于计算需要时间开销，因此如果在查询执行时间到达第 i 个 τ 值时再为第 $i+1$ 个 τ 进行计算，则需要一定的等待时间。所以 MultiFactor 策略采用的是在查询执行时间还未达到第 i 个 τ 时(如达到 $90\%\tau$ 时)就对所收集的数据进行计算，提前得到计算结果，这样可以减少等待时间，从而减少了系统的时间延迟。

对于 τ 值的取值，我们采用的也是自适应的调节方法。把第 $(i+1)$ 个 τ 将使用的操作符路径与第 i 个 τ 使用的操作符路径进行比较，如果连续多次都没有改变，则说明数据流相对稳定，可以增加时间间隔；如果操作符路径发生了改变，则说明数据流特性在发生变化，应该提高系统的灵敏度，即减小路径计算的时间间隔。

具体的策略如下：

1) 如前所述的适应性方法中，我们在每个时间间隔 τ 内为下一个时间间隔 τ 计算下一次的操作符路径，形成新的查询计划，并将新的操作符路径与原来的路径进行比较，以确定是否发生了改变。

2) 如果经过连续 n 次计算之后，即在 $n\tau$ 时间内，操作符路径一直没有发生改变，则说明数据流相对稳定，可以增加时间间隔，因此将 τ 增加一倍，$\tau = 2\tau$。

(下转第 193 页)

空间中，使用这3种分类器的分类效果较好。而算法的时间复杂度由于与特征提取方法无关，因此，实验4与实验3的结果完全相同。

结束语 本文提出的基于正交投影的分类器算法，其特点是由各类的所有训练样本生成子空间，克服了最小距离和最近邻分类器由单个特征点代表各类的局限。同时不论训练样本的数量多少，该算法都可以方便地生成子空间，避免了贝叶斯分类器中各类协方差矩阵因小样本问题而引起的奇异性问题。本文算法由于计算非常简单且易于理解，因此非常适合于小样本数据的识别问题。

参 考 文 献

[1] Li S Z. Face Recognition Based on Nearest Linear Combinations [C]// Proceedings of CVPR. 1998:839-844

[2] Li S Z, Lu J W. Face Recognition Using the Nearest Feature Line Method[J]. IEEE Trans. on Neural Networks, 1999, 10(2):439-443

[3] Chien J T, Wu C C. Discriminant Waveletfaces and Nearest Feature Classifiers for Face Recognition[J]. IEEE Trans. on PAMI, 2002, 24(12):1644-1649

[4] Zheng Wen-ming, Zou Cai-rong, Zhao Li. Face Recognition Using Two Novel Nearest Neighbor Classifiers[C]// Proceedings of ICASSP. 2004:725-728

[5] Friedman J H. Reguralized discriminant analysis[J]. J. Amer. Statistic. Assoc., 1989, 84(405):165-175

[6] 王卫东，郑宇杰，杨静宇. 采用虚拟训练样本优化正则化判别分析[J]. 计算机辅助设计与图形学学报，2006，18(9)：1327-1331

[7] 王卫东，杨静宇. 采用虚拟训练样本的二次判别分析方法[J]. 自动化学报，2008，34(4)：400-407

[8] Lay D C. Linear Algebra and Its Applications (Third Edition) [M]. Pearson International Edition, 2003

(上接第144页)

3) 相反，如果在n次计算中某一次计算得到的操作符路径与前一次相比发生了改变，则说明数据流的特性在发生变化，应该提高系统的灵敏度，即减小路径计算的时间间隔，因此将τ减小一半，$\tau=\tau/2$。

4 性能测试与评价

本文将MultiFactor算法与FIFO，Greedy算法进行了比较。实验环境为Pentium4 1.7GHz的CPU，内存256M，操作系统是Windows 2000 Professional。与其它的调度算法相比，MultiFactor查询优化策略不仅考虑了对系统内存的需求，而且考虑了系统时间延迟，是对二者进行的一种综合考虑。

图2为各个算法在运用于单流上的查询时系统内存使用量的比较。我们利用系统中元组的数量来代表内存使用量。从该图可以看出，在内存使用量方面，MultiFactor策略要好于其它两种策略。

图3为各算法在运用于单流上的查询时时间延迟的比较。可以看出MultiFactor策略与其它两种策略相比，时间延迟较小，并且随着时间的变化优势更加明显。并且MultiFactor调度策略根据查询的截止时间来得到调度时间片，而查询中的截止时间为600ms。所以从该图中可以看到MultiFactor的时间延迟是小于600ms的。

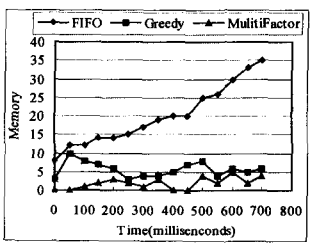

图2 单流上3种算法内存使用量比较

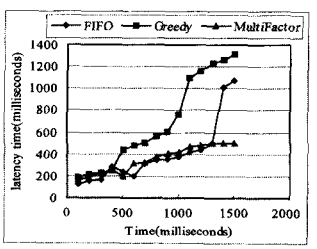

图3 单流上3种算法时间延迟的比较

图4是各个算法在包含连接操作符的查询上进行内存使用量的比较。从图中可以看出，具有连接操作符的查询比不含有连接的查询内存使用量有了明显的增加。MultiFactor策略在加入连接操作符之后，在内存使用量方面仍好于其它两种策略。

图5为各个算法在包含连接操作符的查询上时间延迟的比较。可以看出，具有连接操作符的查询比不含有连接的查询时间延迟上有了明显的增加，MultiFactor策略在加入连接操作符之后，在时间延迟方面仍好于其它两种策略。

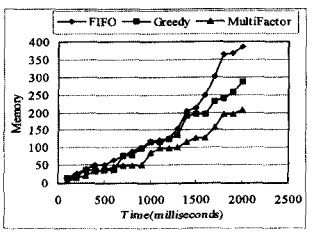

图4 多流上3种算法内存使用量比较

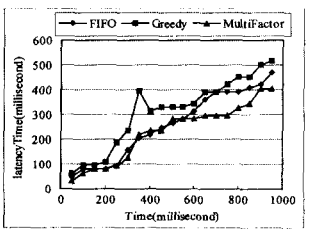

图5 多流上3种算法时间延迟比较

结束语 本文根据数据流的种种特性，考虑了系统的内存需求和查询的响应时间，提出了一种适应性查询优化及调度策略MultiFactor。该策略根据各操作符消耗系统中元组数量的快慢来动态调整操作符的调度次序，按查询的截止时间来确定各操作符的调度时间，并依据数据流各个特性因素的变化来适时调整系统的查询计划，是一种具有适应性的查询优化策略。

参 考 文 献

[1] Abadi J, Cherniack M, Christian C, et al. Aurora: a new model and architecture for data stream management[J]. the VLDB Journal, 2003, 12(2):120-139

[2] Carney D, Çetintemel U, et al. Operator Scheduling in a Data Stream Manager[C]// Proc. of the 29th International Conference on Very Large Data Bases (VLDB'03). Berlin, Germany, September 2003:213-216

[3] Das A, Dally W J. Stream Scheduling: A Framework to Manage Bulk Operations in a Memory Hierarchy[C]// Proc. of the 16th International Conference on Parallel Architecture and Compilation Techniques. 2007, 405:253-262

[4] Munagala K, Srivastava U, Widom J. Optimization of Continuous Queries with Shared Expensive Filters[C]// Proceedings of the twenty-sixeh ACM Sigmod-Sigct-Sigart Symposium on Principles of Database Systems. 2007:215-224

[5] Chandrasekaran S, Cooper O, Deshpande A, et al. TelegraphCQ: Continuous dataflow processing for an uncertain world[C]// Proc. of Conference on Innovative Data Systems Research. Asilomar, CA, 2003:269-280