一种资源共享情况下的连续查询算子调度策略

胡子敬　李红燕

（北京大学智能科学系　视觉与听觉信息处理国家重点实验室　北京100871）

摘　要　算子调度是数据流应用中的重要组成部分。数据流中的数据具有无限、大量、连续、快速及时变等特点。如何适应有限系统资源情况下的所有这些特性是一个重要的问题。我们首先讨论了一些调度策略，然后提出了连续查询中的资源共享问题。提出了一种算子调度策略来解决此问题，并证明了其正确性。实验证明，该策略是非常有效的。

关键词　数据流，算子调度，资源共享

An Operator Scheduling Strategy in a Data Stream System

HU Zi-Jing　LI Hong-Yan

(Department of Intelligence Science, Peking University, Beijing 100871)

Abstract　Operator scheduling is an important component of a data stream application. The data in a data stream is multiple, continuous, rapid and time-varying. How to cope with all these characteristics under limited system resource is an important issue. We first talk about some scheduling strategies, and then advance resource sharing problem among continuous queries. An operator scheduling strategy is proposed to cope with this problem. Its correctness has also been proved. Our experiments show that our strategy is very efficient.

Keywords　Data stream, Operator scheduling, Resource sharing

1　引言

在网络监控、电话通讯、传感器网络以及普适计算等许多信息处理应用领域中，数据是以一种连续的流的形式存在的。在这些情况下需要对流数据进行处理，同时还要对连续查询提供支持。这些都是传统的数据库技术所不能妥善解决的问题，因此便应运而生了一种新的数据模型——数据流模型[1]。针对这一新的模型已经出现了许多原型系统，例如 Stanford 的 Stream 系统，Brown/MIT 的 Aurora 系统，Berkeley 的 Telegraph 系统，以及 OGI/Wisconsin 的 Niagara 系统等等。

连续查询[2]是数据流模型中的一个重要概念，它的特点是用户只需提交一次查询，之后这些查询便在系统中长时间地运行下去，不断处理新到来的数据。每一个数据流应用系统中都包含着大量这样的连续查询。与传统的 DBMS 相对应，每一个连续查询在系统中都由一个查询计划来表示。虽然目前所出现的原型系统中对查询计划的结构表示不尽相同，但是它们全部都由三个主要的部分所组成[1,3,4]：算子(operator)，缓冲队列以及主题。一个算子完成一些简单的操作，比如选择、投影以及连接等等，构成了基本的查询语义。缓冲队列用于对算子的输入输出进行缓冲。数据流中的数据一旦流过就不能再次访问，因此针对某些需要处理以往数据的查询就必须为其维护一定的历史数据，这就是所谓的主题。任何一个连续查询的语义都可以由一系列的基本算子所组成。

另外，数据流中的数据具有无限，大量，连续，快速以及时变等特点。与此同时，大多数数据流应用系统都有较高的实时性的要求，例如医院的 ICU 监护系统，生产制造行业等等。鉴于此，如何针对数据流数据的特点来分配有限的系统资源以满足系统实时性的要求就成为了数据流应用系统的一个重点问题。在所有的系统资源中(内存，计算，磁盘 I/O，网络带宽等)，内存无疑是最为重要的一种资源，它直接影响着系统执行的性能。因此这里我们只针对内存资源进行讨论。目前，主要有两种方法来减少内存的占用[5]：1. 利用数据的特征来减小主题的大小；2. 利用某种连续查询的算子调度策略来减小缓冲队列的大小。方法1需要对数据本身的特征有一定的认识，要针对具体问题进行具体分析，通用性较差，这里我们讨论的是调度策略的问题。

本文首先介绍了一些现有的调度策略，着重介绍了 Stanford 的 Chain 策略，之后引入了资源共享的问题，并在此基础上进行了讨论，最后，我们在 Chain 策略的基础上给出了一种解决资源共享问题的方法，克服了 Chain 的不足，同时也降低了系统运行时内存的占用。

2　相关工作

一个合适的调度策略在正常情况下应该能够及时处理所有到达的数据，并且当到达系统的数据产生爆发时，应该能最大程度地降低系统内存的占用，以保证系统资源不被耗尽。这里所说的爆发是指数据在短时期内的迅速增多，超过了系统的处理速度。但从长时间范围内来看，数据的平均到达速度应该小于系统的处理速度，否则就需要一些其他的机制来进行处理，比如 load shedding[6]。文[7]中已经证明了使系统内存占用最小化的离线算子调度问题是 NP 完全问题，这也说明了不可能得到一个最优的调度策略。

目前的一些调度策略主要有：

循环调度：这种策略循环扫描系统中的所有算子，并依次调度准备好的算子进行执行。执行期间，算子运行一段特定的时间或者直到处理掉了输入队列中的所有数据，然后进入下一个算子的执行阶段。它能够有效地避免饿死，但是对流数据爆发的应变能力较差。

先入先出：先入先出策略按照元组的到达顺序依次处理每一个元组，在一个元组处理完之前不处理下一个元组。它可以最小化数据处理的响应时间，适用于对系统的服务质量(Quality of Service[6])要求较高的情况。但它与循环调度一样，对数据爆发的应对能力较差。

贪心法：在这种策略中，令单位时间内有较高过滤元组能力的算子具有较高的优先级。系统在调度时按照优先级顺序，从高到低依次调度准备好的算子进行处理。这种策略可以在一定程度上降低数据爆发时系统的内存占用，但是它并没有考虑到算子排列顺序的因素。

Chain 策略：Chain[8]是斯坦福的 STREAM 小组提出的一种调度策略。它是在贪心法的基础上，综合考虑算子排列顺序之间的影响，把算子序列分段作为基本单元进行调度，并根据单位时间过滤元组的能力为各个段分配优先级。其基本思想就是尽量使过滤元组能力强的算子先得到执行。数据爆发时，Chain 在单流查询情况下的内存占用方面达到了近似的最优，同时它对包含有连接操作的多流查询时同样也有很好的效果。

虽然文[8]中证明了 Chain 策略与其他策略相比具有明显的优越性，但是它也存在以下不足：1数据流中的数据爆发时不能保证重要数据得到及时处理；2并没有考虑到查询计划中资源共享的问题。文[7]中已经针对问题1给出了一些 Chain 策略的改进方法。我们这里针对系统资源共享问题进行相应的讨论。

3 资源共享问题定义及讨论

3.1 符号及定义

本节所用符号及含义如表1所示。

表1 符号表

符号	含义
Q	一个连续查询
O_i	查询 Q 的表达式中的第 i 个算子
q_i	算子 O_i 所对应的输入队列
S	一个数据流
$selO_i$	算子 O_i 的选择率
T_i	算子 O_i 处理一个元组所需时间

同时我们还引用了文[8]中“过程图”以及“段”的概念，并对其定义进行了一些适当的修改与扩充。

定义1(算子选择率) 算子 O_i 的选择率 $selO_i$ 表示的是单位时间内，算子 O_i 的流出元组数与总的流入元组数之比，它表示算子过滤元组的能力。

定义2(算子执行时间) 算子 O_i 的执行时间 T_i 表示的是算子 O_i 处理一个元组所用的时间。

算子的选择率及其执行时间可在系统运行过一段时间以后，根据所搜集到的统计信息计算得到。

定义3(过程图) 查询 Q 的过程图表示的是一个元组在一个连续查询的处理过程中大小随时间变化的情况(如图3所示)。图中横坐标表示时间，用 t_i 表示，纵坐标表示元组大小，用 s_i 表示。它包含了 $n+1$ 个点 $(t_0,s_0),(t_1,s_1),\cdots,(t_n,s_n)$ 其中 $s_0=1;0=t_0<t_1<\cdots<t_n$，且都为正整数。图中的第 i 个点表示的是查询的第 i 个算子 O_i。

由算子的选择率及算子的执行时间就可以构造出查询的过程图。

定义4(段) 段表示的是一个顺序执行的算子序列。在查询的过程图中，对任意点 (t,s) 及 (t_i,s_i)，$t_n\geq t_i>t$，令 $f=(t,s)=(t_{max},s_{max})$，其中 $(s-s_{max})/(t_{max}-t)=\max((s-s_i)/(t_i-t))$，令 $x_0=(t_0,s_0)$，$x_1=f(x_0)$，$x_2=f(x_1)$，$\cdots$，$x_m=f(x_{m-1})$，这样就得到该查询中从点 (t_0,s_0) 开始到点 (t_n,s_n) 的段划分：$x_0,x_1,x_2,\cdots,x_m$。

类似算子的选择率，我们还可以定义段的选择率。段就是系统执行的一个基本单元，这样执行可以确保从某一点开始执行时，单位时间内消耗元组的程度达到最大。将段的起点与终点用虚线相连就得到了元组大小在此查询处理过程中的下界[8]，如图1中虚线所示。

图1给出了两个连续查询 Q_1 和 Q_2 的过程图，其中 Q_1 的语义由顺序排列的算子 O_1，O_2，O_3 组成，Q_2 由顺序排列的算子 O_1，O_4，O_5 组成。

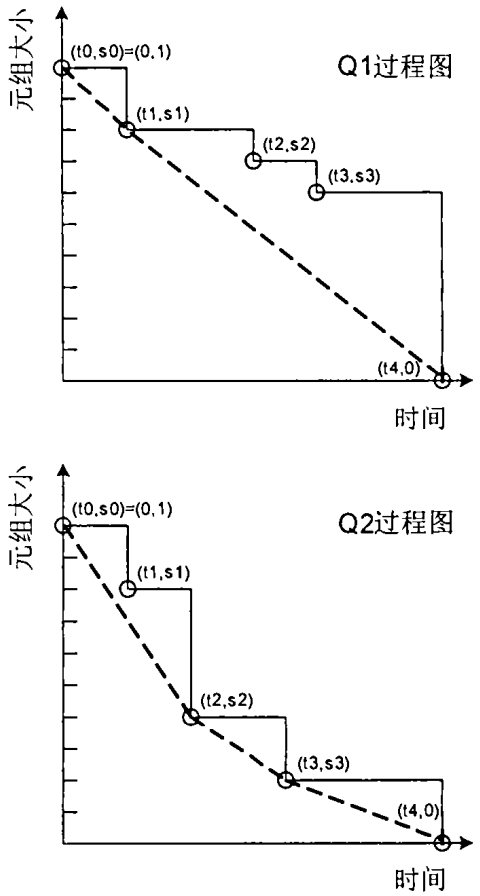

图1 Q_1 和 Q_2 的过程图

3.2 资源共享的意义

资源共享是说当两个连续查询表达式包含相同的子表达式时，就可以在这两个连续查询间共享相同子表达式的存储资源以及计算资源，从而实现资源的节约[5]。在实际应用中，这种资源共享的情况是经常发生的，针对某些重要的数据流往往存在着多个查询。需要注意的是，这里所说的相同的子表达式是指以数据流作为输入端的表达式，对于两个连续查询表达式中间的部分并不考虑在内。这是由于此时查询表达式所对应的数据流各不相同，因此无法共享资源。

连续查询可分为单流查询与多流查询。单流查询是指查询只对一个数据流进行处理，比如选择、投影等操作所组成的查询。多流查询是指对多个数据流进行综合查询，比如含有连接操作的查询。由于多流查询问题较为复杂，因此我们目前所进行的讨论只针对单流查询。

由于 Chain 策略在划分段时没有考虑到资源共享的问

题，因此段的边界与共享子表达式的边界就不一定相同，如图1中Q_2的第一个段与共享子表达式O_1。这种不协调性就导致了必须为每一个查询都维持独有的资源才能够应用Chain。资源之间不能进行共享，使得系统为存在相同子表达式的查询维持了多份数据的拷贝，同时需要对相同的数据进行多次相同的操作，这势必会浪费系统的内存和计算资源。上例中Q_1和Q_2进行资源共享的实例如图2所示。

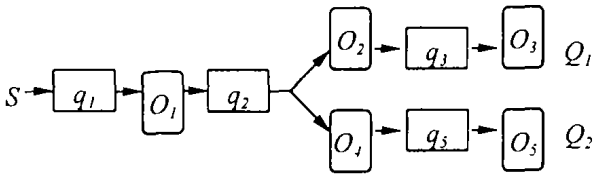

图2　Q_1和Q_2进行资源共享的实例

需要注意的是对O_1的输出队列q_2中的数据项来说，只有当它被两个算子O_2和O_4分别处理过以后才能被丢弃，这就产生了一定的调度复杂性。

3.3　针对共享子表达式情况的处理

在上面的例子中我们可以看到，共享表达式输出端的元组只有在被所有查询中对其进行操作的算子处理过之后才能被丢弃。因此，在这种情况下一味地按照Chain策略进行调度不但不能降低内存的占用，反而还会使占用有所增加，也就是说此时算子的选择率已经不能反映系统消耗元组的实际情况，因此贪心法也不能适用于资源共享的情况。例如上例中，当q_2中一个元组刚被算子O_2操作过后，此时由于元组还需要被算子O_4处理，因而不能被丢弃，此时系统内存的占用不但没有降低，反而增加了q_3中所产生的元组，增加的大小为$sel_{O_2}\times T_2$。

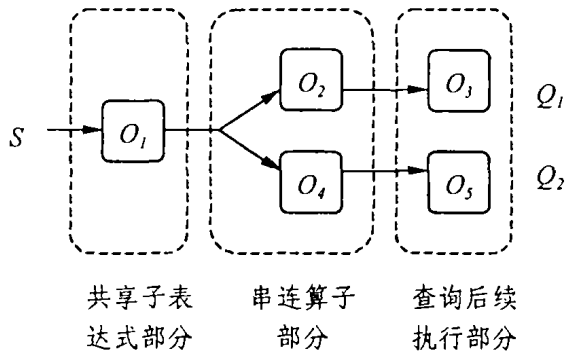

图3　Q_1和Q_2所组成的合并查询的各个组成部分

在所有这些共享某一子表达式的查询中，我们把输入端为公共子表达式输出端的所有算子称为共享算子。在我们的策略中，我们将这些共享算子串连，形成一个逻辑意义上的一个新算子，称其为串连算子。将所有这些查询作为一个逻辑意义上的合并查询。这个合并查询可以看作是由3个顺序的部分所组成：共享子表达式部分，串连算子和各个查询的后续操作部分，上例中，Q_1和Q_2所组成的合并查询的结构如图3所示。

我们可以根据这三个部分大致给出合并查询的过程图，Q_1和Q_2的合并查询的过程图如图4所示。

图4中$t_1=T_1$，$t_2=t_1+T_4+T_2$，$t_3=t_2+T_3+T_5$。$s_1=sel_{O_1}*T_1$，$s_2=sel_{O_2}*T_2+sel_{O_4}*T_4$。$t_0$到$t_1$时刻是共享子表达式$O_1$的执行阶段，$t_1$到$t_2$时刻是串连算子$O_2$和$O_4$的执行阶段，$t_2$到$t_3$是$Q_1$和$Q_2$的后续操作部分。串连算子部分与后续操作部分中，算子的执行顺序并不固定，因此我们用虚线来表示元组的大小变化。

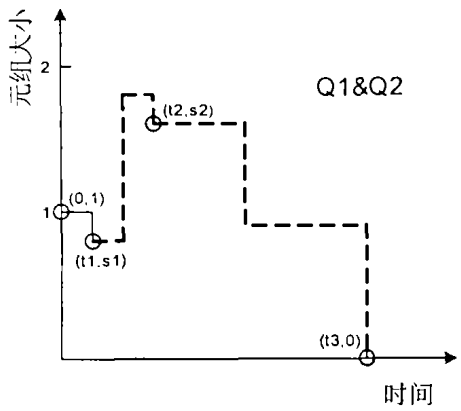

图4　Q_1和Q_2的合并查询的过程图

由上面的分析可知，在合并查询的串连算子执行过程中，元组的大小可能会大于初始的大小1。由于一个算子的选择率最大为1，也就是说它的执行至多增加1个元组。同时，执行最后一个共享算子时，由于共享元组已经可以被释放，因此对它的执行不增加额外的元组数。

现假设有n个连续查询共享某一子表达式，易得在执行串连算子的过程中元组的大小由初始的1至多变为n。当各个查询的后续操作部分执行时，由于不存在共享元组的问题，因此算子的执行只能减少元组大小。由此可知，合并查询中元组大小最大变为n，n表示共享子表达式的连续查询数。

不考虑资源共享时，Chain与其他策略相比的优点就是它将段作为基本执行单元，以此来确保具有低选择率的算子能够先得到执行，从而使系统单位时间内消耗元组的大小达到最大。为了能够继承Chain的这个优点，我们对合并查询的共享子表达式部分和各个查询的后续操作部分独立地用Chain策略进行段划分。我们将串连算子作为一个单独的段，用所有共享算子的平均选择率作为该段的选择率。对于不进行资源共享的查询，我们仍按照Chain策略进行段的划分。

4　调度策略描述

下面给出我们调度策略的具体描述：

1. 为每个连续查询的算子确定其选择率，操作时间等信息；

2. 为不共享子表达式的查询按照Chain策略建立相应的过程图，进行段的划分，并计算其选择率；

3. 对每个共享子表达式的连续查寻进行合并，得到各个段的划分，并计算其选择率；

4. 根据各个段的选择率为其分配优先级；

5. 综合考虑此时系统中所有的段，按照优先级顺序执行准备好的段；

6. 在每个系统时间单元执行步骤5。

5　误差分析

由于对于不共享子表达式的查询，我们采用的是与Chain一致的策略，因此我们只分析共享子表达式时的情况。

在与可以共享资源的循环调度和先入先出策略相比时，由于我们在合并查询的共享子表达式部分和各个查询的后续操作部分采用了Chain的段划分策略，同时文[8]中已经证明了这种划分相对于其他策略的优越性。即使在段的划分没有

（下转第137页）

5 王兰成,等. PLS:一种基于信息自动标引的最小推进分词算法及其实现.计算机科学,2002(8.增刊):24～26

6 田梅. 档案机读目录XML描述及其主题信息自动标引的研究:[硕士学位论文].上海:南京政治学院上海分院信息管理系,2004

(上接第133页)

发挥其优越性时(每个段只包含一个算子),我们的策略仍按照贪心法进行调度。因此可以说我们的策略比单纯的采用循环或者先入先出策略时,占用的系统内存要少。

文[8]中已经证明了Chain策略与最优调度策略的误差为n,其中n表示连续查询的总数。在我们提出的合并查询中(假设它也由n个查询组成,且数据流缓冲区中没有元组),某一时刻共有m个元组正被这个合并查询进行操作,也就是说这m个元组已经被第一个共享表达式的算子处理过。我们已经证明了在合并查询中,1个元组的大小在过程图中的最大值变为n,由此可得任意时刻合并查询中元组大小的最大值为$m\times n$。由于没有策略能够不占用内存,因此这个值也可以被认为是与用最优策略调度时内存占用量的最大误差。而与Chain策略的误差n相比,我们的策略在最坏情况下多出了$m\times n-n$个元组大小的空间。然而,由于这m个元组都来自数据流缓冲区,而这个缓冲区为n个查询所共享,因此,与不进行资源共享的策略比起来,共享策略在数据流缓冲区部分就已经节省了$m\times(n-1)=m\times n-m$个元组的空间。将上面两个值相减得$m-n$,它表示的是假设只有数据流被共享时,最坏情况下我们的共享资源策略比不共享资源的Chain策略多占用的元组数。

很明显,当系统发生数据爆发时(数据流缓冲区及共享子表达式部分有大量的元组),我们的策略相比Chain可以非常明显地降低系统内存的占用。同时,我们还可以规定当合并查询正在处理的元组数达到n时,暂停处理数据流中的数据。这样就可以保证我们的策略在任意时刻都比不能共享资源的Chain策略占用更少的内存。

6 模拟试验分析

我们模拟了两个简单的连续查询Q_1和Q_2,Q_1的各个算子在过程图中的点(t_i,s_i)分别为(0,1),(1,0.7),(2,0.4),(3,0.3)和(4,0),Q_2的算子所对应点的坐标分别为(0,1),(1,0.7),(2,0.3),(3,0.2)和(4,0)。其中Q_1和Q_2共享相同的数据流S以及第一个算子O_1。我们假设初始状态共享数据流中有1000个元组(对Chain来说则是2000个),并令每一个算子处理一个元组的执行时间都为1毫秒。

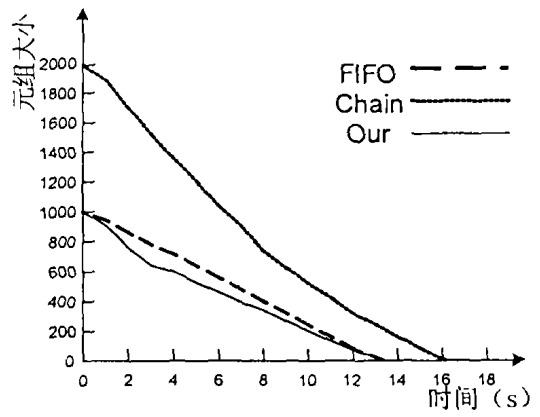

图5 试验结果

实验中,我们对先入先出,Chain以及我们的策略进行了编程实现,图5显示了应用不同策略时元组数随时间变化的情况,这里我们假设先入先出策略在执行各个查询的后续操作部分时采用顺序执行的办法,即顺序完成各个查询。

从试验结果中我们可以看出,我们的策略要比采用先入先出和Chain策略具有更好的效果。

结论与展望 本文针对现有策略中没有考虑资源共享问题的情况给出了一种数据流应用系统中查询算子的调度策略。这种策略继承了Chain策略的优点,并在一定程度上减小了引入资源共享问题所带来的误差。模拟的试验证明,与不能共享资源的Chain策略和可以共享资源的FIFO策略相比,我们给出的策略在数据爆发时能够更有效地减小内存占用。

在实际应用中,可以将我们的算法与Chain相结合,对能够进行资源共享的查询应用我们的策略,而对于不能进行资源共享的查询则应用Chain策略,这样可以更进一步提高系统的执行效率。

在将来的工作中我们会在真实环境中进一步检验我们的策略,并将这种策略扩展到多数据流查询的情况,比如含有连接操作的查询。

参 考 文 献

1 Babcock B, Babu S, Datar M, Motwani R, Widom J. Models and Issues in Data Stream Systems. In: Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp on Principles of Database Systems, June 2002. 1～16

2 Babu S, Widom J. Continuous Queries over Data Streams. SIGMOD, 2001, 30(3): 109～120

3 Abadi D J, Carney D, Cetintemel U, et al. Aurora: a new model and architecture for data stream management. In VLDB Aug. 2003. 120～139

4 Chandrasekaran S, Cooper O, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proc. of CIDR Conf. Monterey, California, Jan. 2003

5 Motwani R, Widom J, et al. Query processing, approximation, and resource management in a data stream management system. In: Proc. of First Biennial Conf. on Innovative Data Systems Research (CIDR). Monterey, California, Jan. 2003. 245～256

6 Carney D, Cetintemel U, Cherniack M, et al. Monitoring streams-a new class of data management applications. In: Proc. 28th Intl. Conf. on Very Large Data Bases. Hong Kong, China, Aug. 2002

7 Babcock B, Babuy S, Datarz M, Motwanix R, Thomas D. Operator Scheduling in Data Stream Systems: [Technical report]. Stanford University STREAM Group, 2004. Available at: http://www-db.stanford.edu/stream/

8 Babcock B, Babu S, Datar M, Motwani R. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data. San Diego, California, June 2003