数据流中一种适应性查询处理机制*)

宋宝燕[1] 张立杰[1] 陆 岩[1] 于 戈[2]
(辽宁大学信息科学与技术学院 沈阳 110036)[1] (东北大学信息科学与工程学院 沈阳 110004)[2]

摘 要 针对数据流中连续查询特征,本文提出一种适应性的查询处理机制,它不但能在有限时间内最大可能地输出结果元组,也可对有限的元组以最快时限输出。而此查询处理机制主要依托于基于输出速率的代价模型,此模型将不断变化的流速、谓词选择率、操作符处理时间作为代价函数变量,将输出速率作为代价模型的函数值。因此此代价模型可适应环境以及数据流本身不断变化的因素,并可作为查询计划动态选择的标准。实验证明此适应性查询处理机制最终能有效地提高输出速率、增加查询吞吐量、减少时间延迟,降低查询间内存占有量。

关键词 数据流,适应性,代价模型

An Adaptive Query Processing Mechanism in Data Stream System

SONG Bao-Yan[1] ZHANG Li-Jie[1] LU Yan[1] YU Ge[2]
(School of Information Science and Technology, Liaoning University, Shenyang 110036)[1]
(School of Information Science and Engineering, Northeastern University, Shenyang 110004)[2]

Abstract Aimed at properties of continuous queries, we present an adaptive query processing mechanism. It can not only optimize the time at which the last result tuple appears, but also optimize for the number of answers computed at any specified time after the query evaluation. This mechanism depends mainly on an output-rate-based cost model. Firstly, this model treats changed input rate, predicative selectivity and execution time of operators as its function variables. Secondly, it will compute output rate as function output. So this cost model can continuously adapt to variable factors of environment and data stream itself. Meanwhile, it can be a standard to select query plan. Experiment has proved that this mechanism validly increases throughput and output rate, meanwhile reduces output time latency and memory requirements.

Keywords Data stream, Adaptability, Cost model

1 引言

近年来,信息处理技术在应用领域上(如网络、金融应用、电子商务、传感器网络等)得到了很大的拓展。在这些应用中,数据不再拘泥于静态的关系数据,而是一种连续、无界、实时传输、不定速度的流式数据(即数据流)。传统数据处理技术突显出自身的局限性,目前数据流处理技术则成为数据库研究领域的又一热点。

查询处理技术是数据流技术中的关键技术之一。数据流中查询多为连续查询,查询一经注册,该查询一直执行,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、可用 RAM 等)将不断地发生变化;数据流本身的一些特征(如谓词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,适应性查询处理技术应运而生,并成为数据流处理技术中的关键技术。

本文针对数据流中连续查询的特征,提出了一种适应性查询处理机制,它不但能在有限时间内最大可能地输出结果元组,提高吞吐量,也可对有限的元组以最快时限输出,减少输出延迟。此查询处理机制的适应性技术主要依托于基于输出速率的代价模型(Outrate Model),此模型将不断变化的流速、谓词选择率、操作符处理时间等作为代价函数变量,将输出速率作为代价模型的函数值。此代价模型的计算可适应变化的因素,并可作为动态选择查询计划的标准。实验证明,此适应性查询处理机制最终能有效地提高查询性能。

2 相关工作

目前,在数据流技术的研究中,适应性查询处理技术已经得到越来越多的关注,国内外很多研究组织和知名大学都从不同的角度对其进行不同层次的研究。

· 斯坦福大学的 STREAM 系统[1]是一个用于在线处理的通用数据流系统,其适应性主要考虑了输入流的流量和操作符的选择率与执行时间。从限制流量的角度考虑:增加谓词,限制输入流量,减少内存需求;从选择度和执行时间两方面考虑:采用 chain scheduling 策略,对操作符进行合理批调度,减少中间队列从而减小内存需求。

· 加州大学伯克利分校研发的 TelegraphCQ 系统[2,3]是一个通用的数据流管理系统,目前主要应用于传感器网络上。它从路由选择的角度考虑,着重提出了自适应路由选择器 eddy,即通过考虑操作符选择率与执行代价,选择适当的路

*)辽宁省 2005 年博士启动基金(20041029)、国家"863"高技术计划 CIMS 主题(编号:2002AA1Z2308, 2002AA118030)资助。**宋宝燕** 博士,教授,研究方向为数据流技术、数据库技术;**张立杰** 硕士研究生,研究方向为数据流技术;**于 戈** 教授,博士生导师,CCF 会员,研究方向为数据库技术。

由，减少内存需求和执行时间。

· 布朗大学的 Aurora[4] 系统是专门针对流监控应用而开发的一个数据处理系统。它从减少载量的角度考虑，通过监测资源载量，采用载量脱落(load shedding)技术，有效地减小元组个数，最终减小内存使用量；从 QoS 和资源消耗角度考虑，Aurora 系统采用 train scheduling 策略。

尽管这些系统提出了一些有效的适应性查询处理方法，但它们或多或少存在一些不足。例如 STREAM 系统的 chain 调度策略针对数据流突发情况，最大限度地减少执行期间内存使用量，但却造成了较大的输出延迟；TelegraphCQ 系统的 Eddy 查询处理机制合理地考虑执行代价和选择率，同时忽略了变化的流速。而 Aurora 系统载量脱落在及时缓解内存压力的同时也降低了查询的准确度。所以，如何在变化的情况下，适应性地进行查询处理，平衡吞吐量、输出延迟、内存使用量等几方面因素，仍是我们需要解决的关键问题。

3 适应性查询处理机制

针对数据流查询的特征，我们对数据流中适应性查询处理进行了深入研究。为了更好地适应变化的因素，如流速、选择率、执行时间等，以及平衡各系统查询性能指标，如内存使用量、输出延迟、吞吐量等，我们提出了一个适应性查询处理机制。

此查询处理机制的适应性技术主要依托于本文提出的基于输出速率代价模型 OutrateModel。OutrateModel 将不断变化的流速、选择率以及执行时间等因素作为函数自变量，通过计算 OutrateModel 的值，适应不断变化的查询环境以及数据流本身的易变因素，从而弥补 EDDY 等策略的不足；OutrateModel 将输出速率作为代价模型的结果值，所以能保证优化后的查询计划拥有最大的输出速率、有限时间最大的元组吞吐量以及有限元组的最小输出延迟，弥补 chain 等策略的不足。

· 基于速率的代价模型—— OutrateModel

对于不同类型的操作符，OutrateModel 的计算方法不同。用表 1 定义的各参数来模拟各操作符，并根据这些参数进行各操作符 OutrateModel 计算。

表 1 代价模型中的参数

r_A	流 A 的输入流速
C_π	对单位元组进行投影操作的执行时间
C_σ	对单位元组进行投影操作的执行时间
T_A	流 A 的时间窗口大小
$\|A\|$	系统自动分配给流 A 哈希桶的个数
C_N	在 NLJ 中处理一个元组的时间代价
C_H	在 HJ 中处理一个元组的时间代价
f	选择率
σ	连接操作符选择率

对于单目操作符(选择、投影)，输入元组直接进行操作即可得到结果，所以代价模型的计算较为简单。投影操作 OutrateModel 的计算与输入流速 r_i、投影操作符对单个元组处理时间 C_π 有关；而选择操作与输入流速 r_i、选择操作符对单个元组的处理时间 C_σ、选择操作符选择率 f 有关。但是对于复杂的连接操作符，输入元组需要借助滑动窗口技术保存历史信息，从而得到最终结果。所以代价模型的计算较为复杂，不同的连接方式(嵌套循环连接—NLJ、哈希连接—HJ)，不同时间点 t_i(时间窗口未满与时间窗口已满)，代价模型 OutrateModel 中分子、分母的计算方法也不同。OutrateModel 的模型计算公式如下：

输出速率 r_o = 输出元组个数 C_{tuples} / 完成输出所需时间 C_{time}。

具体解释如式子(1)：

$$r_o=\begin{cases} f(r_i,C_\pi) & \text{当操作符为投影时} \\ f(r_i,C_\sigma,f) & \text{当操作符为选择时} \\ f(r_i,t_i,\sigma)=\dfrac{C_{tuples}(r_i,t_i,\sigma)}{C_{time}(r_i,t_i)} & \text{当操作符为连接时} \end{cases} \tag{1}$$

一个查询通常包含多个操作符，不同的操作符序列构成不同的查询计划。对不同查询计划进行 OutrateModel 计算时，首先应对单个操作符进行 OutrateModel 分析与计算；然后对不同查询计划中的操作符序列自底向上地进行 OutrateModel 计算；最后通过比较，拥有最大 OutrateModel 值的查询计划为最优化的查询计划。因为它不但能在有限时间内最大可能地输出结果元组，提高吞吐量，也可对有限的元组以最快时限输出，减少输出延迟。

4 操作符代价的计算

本小节将具体讨论不同类型的操作符的 OutrateModel 的计算方式。

4.1 投影操作符

对于投影操作，当流速不断地变化时，我们需要考虑两种情况，如图 1 所示。

1) 当 $C_\pi \leqslant 1/r_i$，即投影操作符对单个流数据的执行时间小于等于输入流数据相继到达的时间间隔时，输出流速等于输入流速，即 $r_o=r_i$。

2) 当 $C_\pi > 1/r_i$，输出速率等于投影操作符对元组的处理速度，即 $r_o=1/C_\pi$。

4.2 选择操作符

对于选择操作，基于输出速率代价模型的计算需要结合谓词选择率。已知输入流速即单位时间内输入项数为 r_i，则输出的项数为 fr_i，f 为谓词选择率。我们依然比较选择操作符对单个流数据的处理时间以及流数据的输入时间间隔，分两种情况计算选择操作的输出速率。

1) 当 $C_\sigma \leqslant 1/r_i$，输出速率 $r_o=fr_i$

2) 当 $C_\sigma > 1/r_i$，输出速率 $r_o=f/C_\sigma$

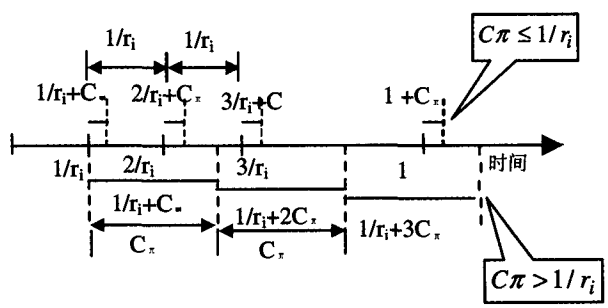

图 1 投影操作输出速率示意图

4.3 连接操作符

当查询为连续查询的时候，连接两个无限输入流需要无限的内存。根据实际应用可得对两流上的所有元组进行连接通常是无意义的。因此，我们对两流中最近 T 秒的元组进行

连接，T 即为滑动时间窗口的大小。在 OutrateModel 的计算中，时间窗口 T 为一个临界值，所以下面分子 C_{tuples} 与分母 C_{time} 的计算均讨论以下两种情况：1）当 $t_i<T$ 时，即在第 i 个单位时间，时间点为 t_i，时间窗口未满；2）当 $t_i>T$ 时，即在第 i 个单位时间，时间点为 t_i，时间窗口已满。

4.3.1 代价模型中分子的计算

针对不同的时间点 t_i，分子 C_{tuples} 的计算是不同的。

· 当 $t_i<T$ 时，基于时间的滑动窗口未满，输出元组个数是时间点 t_i 与流速 r 的函数。

已知 A、B 两流的输入流速为 r_A 与 r_B，时间点为 t_i 时，从 A 流读入的元素个数为 r_At_i，从 B 流中读入的元素个数为 r_Bt_i。假设当时间点 $t_0=0$ 时查询开始执行，则在第一个单位时间，即 $i=1$，$t_i=1$，输出的结果元组数为 $1\sigma r_Ar_B$，其中 σ 为连接操作的选择率。第二个单位时间内输出元组个数为总的输出元组个数 $\sigma 2r_A2r_B-\sigma r_Ar_B$ 减去第一个单位时间内输出元组个数，即 $3\sigma r_Ar_B$。依次类推，第三个单位时间内输出元组个数为 $5\sigma r_Ar_B$。所以在第 i 个单位时间内输出元组个数如式(2)所示：

$$C_{\text{tuples}}=(2t_i-1)\sigma r_Ar_B \quad \text{当 } t_i<T \text{ 时} \tag{2}$$

· 当 $t_i>T$ 时，基于时间的滑动窗口已满，输出元组个数仅为流速的函数，与时间点无关。

时间窗口已满后，新来的元组就会插入到滑动窗口中，而过期的元组则需置无效，所以在任意单位时间内，输出的结果元组个数为：

$$C_{\text{tuples}}=(2T-1)\sigma r_Ar_B \quad \text{当 } t_i>T \text{ 时} \tag{3}$$

4.3.2 代价模型中分母的计算

假设流 A 与流 B 进行连接操作。在流 A 中，每流入一个新元组，都会触发下面 3 个操作：检查 B 流窗口 b，寻求可连接的元组，即 *probe*(b)；向 A 流窗口 a 中插入新元组，即 *insert*(a)；在 A 流窗口 a 中将过期元组置无效，即 *invalidate*(a)。同理，在流 B 中，每流入一个新元组也会触发上述 3 个操作。用各个操作所存取的元组个数来计算 *insert*、*probe* 与 *invalidate*；用 C 表示处理一个元组所需的时间代价。则单位时间内的元组进行连接，连接的处理时间 C_{time} 为：

$$C_{\text{time}}=C[r_A(probe(b)+insert(a)+invalidate(a))]+C[r_BC(probe(a)+insert(b)+invalidate(b))]$$

连接是双向连接，由流 A 到流 B 的单向连接与流 B 到流 A 的单向连接组成。为了讨论方便，我们把流 A 到流 B 单向连接的时间代价用 $C_{A\triangleright B}$ 表示，流 B 到流 A 的单向连接的处理时间用 $C_{A\triangleleft B}$ 表示。下面以分析流 A 向流 B 的单向连接的时间代价 $C_{A\triangleright B}$ 为例进行讨论，如式(4)所示：

$$C_{A\triangleright B}=C[r_A(probe(b))+r_B(insert(b)+invalidate(b))] \tag{4}$$

假设流 A 到流 B 的单向连接方式为循环嵌套连接，则在 $C_{A\triangleright B}$ 的计算中，C 由 C_N 代替，C_N 代表循环嵌套下处理一个元组所需的时间代价。*probe*(b)为探测到的窗口 b 中所有元组；*insert*(b)等于 1，因为除了它本身被插入外，无需存取其他元组；*invalidate*(b)也等于 1，因为插入一个元组就要置无效一个元组。$C_{A\triangleright B}$ 如下式所示：

$$C_{A\triangleright B}=C_N(r_A(r_Bt_i)+2r_B)$$

假设流 A 到流 B 的单向连接方式为哈希连接，则在 $C_{A\triangleright B}$ 的计算中，C 由 C_H 代替。*insert*(b)仍等于 1，而 *probe*(b)与 *invalidate*(b)的值则为流 B 中哈希桶数 $|B|$ 的函数。*probe*(b)与 *invalidate*(b)都等于流 B 一个哈希桶中所有元组数。$C_{A\triangleright B}$ 如下式所示：

$$C_{A\triangleright B}=C_H\left(r_A\left(\frac{r_Bt_i}{|B|}\right)+r_B\left(\frac{r_Bt_i}{|B|}+1\right)\right)。$$

下面讨论在临界值 T 的两侧，针对不同的时间点 t_i，分母 C_{time} 的计算公式。

· 当 $t_i<T$ 时，基于时间的滑动窗口未满，新元组的到达不会触发置无效操作。

在此情形下，不同连接方式下单向连接的时间代价计算公式如下：

$$\begin{aligned}C_{A\triangleright B}(\text{NLJ})&=C_N[r_A(probe(b))+r_B(insert(b))]\\&=C_N(r_Ar_Bt_i+r_B)\\C_{A\triangleright B}(\text{HJ})&=C_H[r_A(probe(b))+r_B(insert(b))\\&=C_H\left(r_A\frac{r_Bt_i}{|B|}+r_B\right)\end{aligned} \tag{5}$$

比较式(5)中两种单向连接的时间代价计算公式，可知在 $t_i<T$ 时，选择哈希方式进行两流连接的执行时间代价较小。所以此情形下，以后均采用哈希连接方式进行连接操作。

· 当 $t_i>T$ 时，基于时间的滑动窗口已满，此时新流入的元组要触发 3 个操作。

时间窗口已满，此时新来的元组都要触发置无效操作。在第 i 个时间单位，只要 $t_i>T$，则窗口内的元组数只与速率有关，而与时间点无关，即窗口中元组个数始终等于 r_BT。在此情形下，不同连接方式下单向连接的时间代价计算公式如下：

$$\begin{aligned}C_{A\triangleright B}(\text{NLJ})&=C_N[r_A(probe(b))+r_B(insert(b)+invalidate(b))]=C_N(r_Ar_BT+2r_B)\\C_{A\triangleright B}(\text{HJ})&=C_H[r_A(probe(b))+r_B(insert(b)+invalidate(b))]=C_H\left(r_A\frac{r_BT}{|B|}+r_B\left(\frac{r_BT}{|B|}+1\right)\right)\end{aligned} \tag{6}$$

从上式不能直观地比较出拥有最小时间代价的连接方式，因为不同的连接方式，*probe* 与 *invalidate* 操作存取的元组个数均不同。通常情况下，当 B 流流速较小时，哈希连接所耗费的时间代价较小，因为哈希连接的 *probe* 操作所存取的元组个数较小。但是，当 B 流流速逐渐增大时，哈希连接就逐渐失去优势，因为哈希连接的 invalidate 操作所存取的元组个数较大。我们通过比较式(6)中两种连接方式下单向连接的时间代价计算公式，推导得公式如下：

$$C_{A\triangleright B}(\text{NLJ})>C_{A\triangleright B}(\text{HJ})\Leftrightarrow(r_Ar_BT+2r_B)\times C_n>\left(r_A\left(\frac{r_BT}{|B|}\right)+r_B\left(\frac{r_BT}{|B|}+1\right)\right)\times C_h\Leftrightarrow r_B>r_A\frac{|B|-C_h/C_n}{C_h/C_n}$$

将已知参数代入上式中，如上式成立，则说明流 A 到流 B 的单向循环嵌套连接优于哈希连接；否则，说明流 A 到流 B 的单向哈希连接优于循环嵌套连接。

代价模型计算公式中的分子 C_{tuples}、分母 C_{time} 计算均已讨论完毕。根据当前时间窗口临界值与输入流速，选择适当的分子、分母公式代入式(1)中即可得到相应 OutrateModel 的值。

5 查询计划的代价计算

上面讲述了如何将输入速率作为变量，输出速率作为函数值来进行单个操作符的计算。而一个查询通常包含多个操作符，不同的执行顺序构成多种不同的查询计划。而为了达到要求的性能并获得良好的查询效率，必须对查询计划进行优化。查询优化可分为两步进行：静态查询优化和动态查询优化。

静态查询优化：参照数据库中的优化技术，应用等价变换法则来优化查询计划。如：把选择和投影操作尽早进行等。

动态查询优化：根据不断变化的因素，动态地调整查询计划。如：先执行选择率大的选择操作符等。

本文在静态查询优化的基础上进行动态查询优化。动态查询优化依赖于各个操作符的 OutrateModel 值。基于 OutrateModel 的查询计划动态优化算法如算法 1 所示。

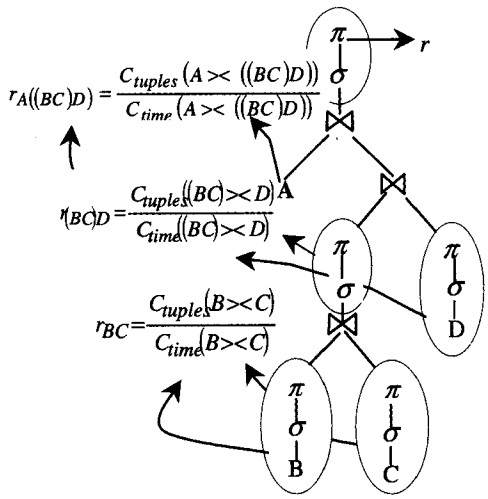

图 2　查询计划代价计算示意图

算法 1　基于 OutrateModel 的查询计划动态优化算法

输入：一个查询的多个查询计划语法树、各操作符的参数、各输入流的输入流速

输出：最终优 OutrateModel 与最优查询计划

步骤：

(1)选择一个查询的一个查询计划语法树，从各个支流开始自底向上遍历。把经过的所有单目操作符(σ 和 π)分为一组，直到遇到一个双目操作符($\bowtie$)。每个双目操作符的直接祖先为一组(这些直接祖先是 σ 和 π 运算)，直到遇到双目运算符或语法树的根节点。每个单独的双目操作符为一组。

(2)如果无连接操作符，则直接运用算法 2 计算出单目操作符组的最优 OutrateModel 及最优查询计划，并将其输出，程序结束。

(3)否则遍历所有连接操作符并获取其深度。从最大深度连接操作符开始循环调用方法 joinoutputrate(joinnote)，以此计算连接操作符节点 joinnote 的 OutrateModel，并将其作为下个操作符的输入速率。当连接操作符节点 joinnote 左孩子无操作符节点时，初始输入流速直接为连接操作符的左端输入流速；当左孩子为单目操作符组时，调用算法 2 计算出左端输入速率；当左孩子为连接操作符时，递归调 joinoutputrate(joinnote)方法计算出左端的输入速率。同理计算出右端输入流速。最终利用式(2～6)，选择合适的计算公式计算出连接操作符的 OutrateModel。

(4)判断最小深度连接操作符是否为根节点。如为根节点，此连接操作符的 OutrateModel 为最终查询计划的 OutrateModel，否则继续采用算法 2，计算出查询计划最终的 OutrateModel。

(5)以此方法循环计算各查询计划的 OutrateModel，OutrateModel 值最大的查询计划为最优的查询计划。输出最终 OutrateModel 以及最优查询计划。

算法 2　基于 OutrateModel 的单目操作符组动态优化算法

输入：输入流速、各单目操作符参数

输出：最终 OutrateModel 以及最优查询序列

步骤：

(1)初始化。每个操作符定义为 $operator(t, f)$，其中 t 为执行时间，f 为选择率；保存组内所有操作符的数组为 $operators[]$；所有操作符可能组成的序列为 $BufferString[]$。

(2)如有投影操作符，将投影操作符进行合并，置于操作符序列的最后端；此时执行时间代价 $t=t_1+f_1t_2+f_1f_2t_3+\cdots+f_1f_2\cdots+f_{n-1}t_n+f_1f_2\cdots f_nt_\pi=\sum_1^n t_i\prod_1^{n-1}f_i+t_\pi\prod_1^n f_i$。

(3)如果不存在投影操作，则经过此操作符序列的总执行时间 $t=t_1+f_1t_2+f_1f_2t_3+\cdots+f_1f_2\cdots f_{n-1}t_n=\sum_1^n t_i\prod_1^{n-1}f_i$。

(4)在第 i 个操作符序列 $BufferString[i]$ 中，调用方法 $outputrate()$ 来计算这个序列的最终输出速率。当 $t<1/r_i$ 时，输出速率 $r_o=r_i$；否则 $r_o=f_1f_2\cdots f_n/t=\prod_1^n f_n/t$。

(5)以此方法循环计算各查询计划的 OutrateModel，OutrateModel 值最大的查询计划为最优的查询计划。输出最终 OutrateModel 以及最优查询计划。

6　性能测试与评价

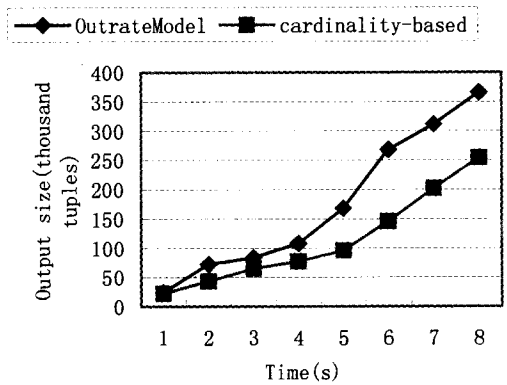

图 3　基于不同代价模型查询计划输出元组比较

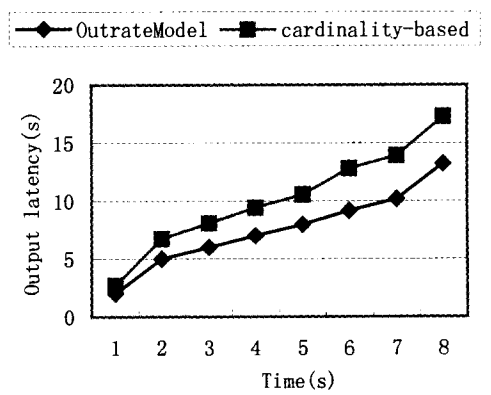

图 4　基于不同代价模型查询计划输出延迟比较

我们采用随机生成的数据作为测试用例，动态收集信息并对其进行统计计算，生成本模型需求的参数值，然后将 OutrateModel 与传统 cardinality-based cost model 进行比较。随着时间的推移，比较基于不同代价模型的查询计划吞吐量及输出延迟。测试环境为：CPU AMD Athlon 3000＋，内存 256M，硬盘 80G，操作系统为 Windows XP。实验结果如图 3、4 所示。

总结与展望　本文针对数据流上连续查询特征，提出了

一种按需式双信道的 Ad Hoc 网络路由方法*)

王庆辉[1] 郑贵平[2] 王光兴[2]

(沈阳化工学院信息工程学院 沈阳 110041)[1] (东北大学信息科学与技术学院 沈阳 110004)[2]

摘 要 为了提高 Ad Hoc 网络的信道带宽和降低通信时的相互影响,在单信道的 Ad Hoc 网络多路径路由方法基础上,提出了按需式的采用双信道的路由策略。在物理上,整个网络存在两个独立的信道。通过一次路由发现,获得两条节点不相交路径,按照每包分配策略,把数据交替在两个信道的不同路径上进行传输。在提高带宽的同时,降低了数据传输冲突,从而改善网络性能。

关键词 Ad Hoc 网络,双通道,按需式路由

An On-demand Dual-Channel Routing Scheme in Ad Hoc Networks

WANG Qing-Hui[1] ZHENG Gui-Ping[2] WANG Guang-Xing[2]

(School of Information & Engineering, Shenyang Institute of Chemical Technology, Shenyang 110041)[1]

(School of Information Science & Engineering, Northeastern University, Shenyang 110004)[2]

Abstract Based on a single channel multipath routing scheme, an on-demand routing scheme called Dual- Channel Routing(DCR) scheme is proposed in this paper. The scheme establishes and utilizes two routes of maximally disjoint paths and distributes them into two different channels respectively. Here, "channel" is used upon a logical level. Physically, a channel can be a frequency band(under FDMA), or an orthogonal code(under CDMA). Then, there is no interference between two routes. A per-packet allocation scheme is used to distribute data packets into two channels of active sessions in our scheme. Even if only one of the two routes of the session is invalidated, the source uses the remaining valid route to deliver data packets in two channels alternately. This traffic distribution efficiently utilizes available network resources and prevents nodes of the route from being congested in heavily loaded traffic situations. The results of simulation show that our scheme has better network performance.

Keywords Ad Hoc, Dual-channel, On-demand routing

1 引言

无线 Ad Hoc 网络的路由协议按工作过程可分为按需式路由协议[1,2]和表驱动路由协议[3]两大类。在表驱动路由协议中,节点需周期性地广播路由信息分组来维护全网所有节点的路由。它的优点是当节点需要发送数据分组时,只要去往目的节点的路由存在,所需的延时很小。缺点是路由维护和管理需要花费较大的开销。为了降低控制开销,按需式路由协议仅在需要时才构建和维护路由。这样,相对表驱动路由协议,按需式路由协议路由开销较低。然而,在动态网络中频繁的路由发现过程带来的较高的路由发现时延严重影响了网络性能。一个比较自然的解决方法是采用多路径路由[4~7],即通过一次路由发现过程寻找到多条路径,并把业务同时分配到这些路径中来实现负载均衡。但是这类协议的缺陷在于由于无线传输的广播特点,不同路径可能相互干扰,并且不容易寻找到两条完全不相交路径,网络性能的提升还是有限的[6]。

信道接入协议(MAC)控制节点如何接入信道,避免节点主机使用共享介质带来的冲突。

协议能否有效地使用有限的信道带宽,将对网络性能产生重要的影响。目前 Ad Hoc 网络中信道接入协议主要采用 802.11 DCF。802.11 DCF 是一种单信道的接入协议,即网络中的所有节点在一个冲突域内。这类协议的缺点是随着网络内节点数目的增多,网络性能快速下降,一种解决办法是采

*)受国家高技术研究发展计划项目(863-701-2-4)、辽宁省教育厅高等学校科技攻关计划项目(20040291)资助。**王庆辉** 讲师,博士,主要研究领域为 Ad Hoc 网络 QoS 及路由技术。**王光兴** 教授,博士生导师,主要研究方向为计算机网络。

一种适应性查询处理机制。此机制主要采用了我们提出的 OutrateModel 代价模型,从而能不断适应系统和数据流自身的特征,动态地选择查询计划,最终能有效地提高元组吞吐量、减少时间延迟以及降低查询间内存占有量,提高了数据流系统中连续查询的处理效率。

参 考 文 献

1 Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems. In: Proc. of the 2002 ACM Symp, In Principles of Database Systems, June 2002. 1~16

2 Chandrasekaran S, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR(2003)

3 Avnur R, Hellerstein J M. Eddies: Continuously adaptive query processing. In: Proc. of 2000 ACM SIGMOD International Conf.

4 Carney D, et al. Monitoring Streams - A New Class of Data Management Applications. VLDB(2002)

5 Viglas S, Naughton J F. Rate-based Optimization for Streaming Information Sources. In: SIGMOD Conf. 2002

6 Kang J, Naughton J F, Viglas S D. Evaluating Windows Joins over Unbounded Streams. In: Proc. International Conference on Data Engineering(ICDE), 2003