ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.2066−2079
DOI: 10.3724/SP.J.1001.2008.02066

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

数据流处理中确定性 QoS 的保证方法*

武珊珊, 于 戈+, 吕雁飞, 谷 峪, 李晓静

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

A Deterministic QoS Guaranteeing Approach for Data Stream Processing

WU Shan-Shan, YU Ge+, LÜ Yan-Fei, GU Yu, LI Xiao-Jing

(School of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

+ Corresponding author: E-mail: yuge@mail.neu.edu.cn, http://www.neu.edu.cn

Wu SS, Yu G, Lü YF, Gu Y, Li XJ. A deterministic QoS guaranteeing approach for data stream processing. *Journal of Software*, 2008,19(8):2066−2079. http://www.jos.org.cn/1000-9825/19/2066.htm

Abstract: Different from the traditional best-effort query service providing, the issue of deterministic QoS guarantee for data stream processing is discussed. Based on the theory of network calculus, an approach of QoS modeling and QoS guaranteeing for data stream processing is proposed. Before system running, the schedulability of all the queries with their QoS requirements satisfied simultaneously is verified. During run-time, service curves representing respective QoS requirements are allocated to each query admitted by QoS schedulability verification, in order to guarantee the expected QoS requirements. Moreover, QoS-guaranteeing batch scheduling and query sharing are extended to improve the query processing efficiency. Finally, experimental results show that the proposed approach offers deterministic QoS guarantee to continuous queries over data streams efficiently.

Key words: data stream; deterministic QoS guaranteeing; task scheduling; network calculus

摘 要: 与以往尽最大努力的查询服务提供方式不同,讨论了数据流处理中确定性 QoS 的保证问题.以网络演算为理论基础,提出了一种数据流处理中的 QoS 建模和 QoS 保证方法.系统运行前验证所有查询在满足各自的 QoS 前提下的可调度性.在运行时为通过 QoS 可调度性验证的每个查询分配代表其 QoS 需求的服务曲线,从而保证各查询期望的 QoS.为了提高查询处理效率,还讨论了保证 QoS 的批调度和查询共享.实验结果表明,该 QoS 保证方法能够有效地为数据流上的连续查询提供确定性的 QoS 保证.

关键词: 数据流;确定性 QoS 保证;任务调度;网络演算

中图法分类号: TP311 **文献标识码**: A

近年来,随着信息处理在工业生产、经济信息处理等领域的广泛应用,数据已不仅仅拘泥于文件、数据库等传统的静态形式,而是以流的形式快速、无限、连续、实时地到达.典型的数据流应用包括无线传感器网络应用环境中对传感器传回的各种数据的监测、股票价格信息的在线分析、网络监测系统与道路交通监测系统

* Supported by the National Natural Science Foundation of China under Grant Nos.60473073, 60503036 (国家自然科学基金); the Program for New Century Excellent Talents in University of China under Grant No.NCET-06-0290 (新世纪优秀人才支持计划); the Fok Ying Tung Education Foundation of China under Grant No.104027 (霍英东青年基金优选课题资助)

Received 2007-01-29; Accepted 2007-08-02

的数据监测等.能够处理数据流的系统环境统称为数据流系统.这些应用往往要求数据流系统在资源有限的情况下处理大量、快速、连续到达的数据流,并且通常要求保证一定的查询服务质量(QoS).

目前,大多数数据流系统所能提供的尽最大努力的查询服务[1-3]只能保证一些统计性的查询性能,如平均响应延时、长期的吞吐率等.尽最大努力的查询服务所带来的不确定性很可能导致任务执行失败,甚至带来灾难性的后果.以传感器网络中的火灾预警应用为例,散布在目的区域内的传感器不断采集温度、气体密度等信息并实时传送给数据流系统进行火灾事件检测.系统一旦发现某个地区的温度和气体密度达到火灾发生的警戒条件,就需要在规定时间内报警.尽最大努力的查询服务即使能够提供满足预警时限的平均响应延时,但那些超过规定时间的警报对可能发生的火灾则错过了最佳的预警时机,也很有可能导致火灾的发生.可见,在关键型任务的数据流应用中,需要数据流系统提供确定性的查询服务质量保证.

有关数据流课题的研究已受到数据库领域越来越多的关注,并出现了一些初具规模的数据流原型系统[4-7].Aurora 系统首先提出了数据流上的 QoS[5,8].Aurora 中的 QoS 是由应用领域专家定义的一组标准化的二维 QoS 图,每个 QoS 图表示某个性能指标在不同取值时对应的 QoS 收益,例如平均响应延时、元组丢弃率等.Aurora 根据运行时的 QoS 收益变化实时调整资源分配.可见,Aurora 定义的是一种后验 QoS,即通过运行时的 QoS 情况动态指导查询处理策略,尽可能获得更好的运行时 QoS.此外,Timothy 等人提出了一种数据流上 QoS 驱动的适应性调度框架[9].在该框架中,系统包含一个调度算法池,系统运行过程中适应性地从调度算法池中选择合适的算法进行查询调度.但这里讨论的 QoS 仍然是一种尽最大努力的查询性能要求,例如,最大化输出速率或者最小化队列大小.Qstream[10,11]是目前唯一一个提出先验 QoS 的数据流系统.由于 Qstream 是在一个实时操作系统上开发的,它的 QoS 确定性保证大都依赖于底层的实时操作系统环境,因此它的 QoS 保证具有很大的底层依赖性.

本文提出一种数据流查询处理中确定性 QoS 的保证方法.第 1 节提出确定性 QoS 保证需要解决的问题.第 2 节给出 QoS 建模方法.第 3 节介绍保证 QoS 的调度算法和可调度性验证方法.第 4 节讨论保证 QoS 的优化策略和实现机制.第 5 节通过实验对方法的有效性进行验证和分析.第 6 节总结全文.

1 问题描述

在给定的数据流系统中,系统内各查询的 QoS 需求、输入特性以及系统的处理能力决定了查询的 QoS 满足性.如图 1 所示,我们将这 3 方面因素构成的三维空间称为该系统的 Q-Space.

定义 1(QoS 可调度性). 数据流系统中有 N 个连续查询 $Q_i,i=1,\ldots,N$.对于 Q-Space 中的点 $P(A,B,1/C)$(其中,A,B 分别表示所有查询的输入特性和 QoS 需求,C 为系统的处理能力),若所有查询 Q_i 都能满足用户要求的 QoS,则称点 P 可调度.

于是,我们把数据流查询处理的确定性 QoS 保证定义为以下几个问题:

- 如何形式化描述查询 QoS;
- 已知 Q-Space 的中点 P,如何验证点 P 的可调度性;
- 在可调度性的前提下,如何找到一种可行的调度算法,能够确保各查询的 QoS.

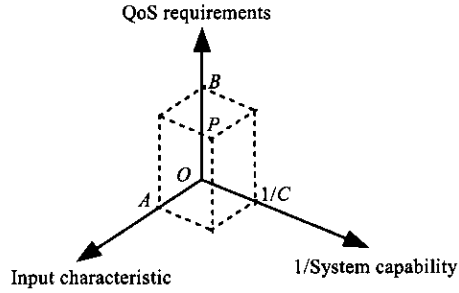


Fig.1 Illustration of Q-Space

图 1 Q-Space 示意图

网络演算[12,13]是一种网络要素计算方法,从理论的深度为 Internet 上数据包的调度、流量控制及网络 QoS 分析提供了一套合理而可靠的理论基础,而且分析网络所得到的结果是确定的、严格的.在数据流系统中,数据流中的元组可以看作是网络链路中的数据包,而数据流的查询调度则类似于网络中的数据包调度.这样,将网络演算的 QoS 保证理论应用在数据流的查询处理中,为数据流上确定性的 QoS 保证提供了一种可行而有效的实现方法.因此,本文提出了数据流处理中一种基于网络演算的确定性 QoS 的保证方法.

2 基于网络演算的 QoS 建模方法

在实际的数据流系统中,多个查询可能对应于多个输入数据流,既存在多个查询共享同一个数据流的情况,又存在一个查询使用多个数据流的情况.但从逻辑上看,可以对共享的数据流进行复制,使得每个数据流只对应 1 个查询,再将参与一个查询的多个数据流合并为一个逻辑流,只是查询对来自不同数据流的数据有不同的处理而已.这样,每个查询对应于 1 个输入流,且对每个查询的处理都独立于其他查询.在具体的实现过程中,针对多个查询共享计算分支的情况,第 4.2 节将进行详细的讨论.而多个查询共享数据流的情况,可以看作是共享分支为空的特殊情况.

2.1 查询QoS的表示

定义 2(输入累积函数 $R(t)$). 对于任意连续查询,用输入累积函数 $R(t)$表示从系统运行开始到 t 时刻进入查询队列的流数据的个数总和.

定义 3(处理累积函数 $R^*(t)$). 对于任意连续查询,用处理累积函数 $R^*(t)$表示从系统运行开始到 t 时刻已经被查询处理过的流数据的个数总和.

将数据流系统看作一个服务系统,输入流是系统中的服务消费者,查询引擎作为服务提供者连续不断地从输入队列获得流数据并提供相应的查询服务.由于选择度的存在,查询输出的结果个数不等于输入数据的个数,故用输入累积函数和处理累积函数来分别描述服务系统的输入和服务情况.这样,数据流系统就是一个无损的服务系统.

根据无损系统的性质,连续查询的积压数据和延时表示如下:

定义 4(积压数据和延时). 对于任意的连续查询,t 时刻系统中的积压数据可用 $R(t)$与 $R^*(t)$的垂直距离表示为 $B(t)=R(t)-R^*(t)$;而第 n 个输入数据的延时可用 $R^*(t)$与 $R(t)$在函数值 n 的水平距离表示为 $d(n)=(R^*)^{-1}(n)-(R)^{-1}(n)$.

对于确定性的 QoS 保证,有意义的 QoS 描述应该包含两个方面:① 输入特性;② 期望获得的查询性能,如响应时间上限、吞吐率下限等.我们用到达曲线来表征输入特性,而用户期望的查询性能要求可以借助于服务曲线进行刻画.由网络演算可知,已知输入的到达曲线和查询获得的服务曲线与延时和积压数据相关的性能参数都能进行确定性分析,这就为确定性 QoS 保证提供了可靠的理论依据.因此,如下定义数据流上连续查询的 QoS.

定义 5(查询 QoS 的表示). 对于任意连续查询 Q,若输入流的到达曲线为α,期望的服务曲线为β,则将查询 Q 的 QoS 表示为 $QoS_Q=\langle\alpha,\beta\rangle$.

我们将在第 2.2 节和第 2.3 节分别讨论如何用到达曲线和服务曲线表示输入流特性和服务质量要求.

2.2 输入流

定义 6(到达曲线). 连续查询 Q 的输入累积函数为 $R(t)$,广义增函数$\alpha(t)$称为查询 Q 的输入到达曲线,当且仅当对于任意 $0\le s\le t$ 有 $R(t)-R(s)\le\alpha(t-s)$成立.

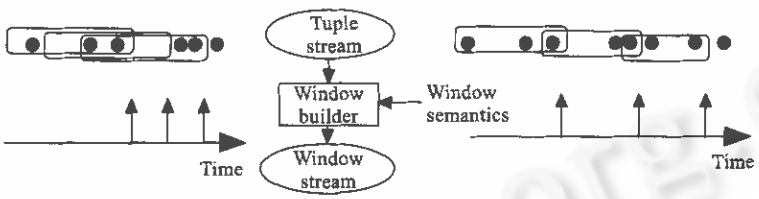
QoS 是数据流系统与用户之间的一种服务契约,当且仅当输入满足一定的到达特性时,系统才能按照协商的契约提供用户期望的 QoS.特别是在确定性 QoS 保证中,必须预先抽取数据流特性.而对于那些到达特性完全不可预知的输入数据流来说,则很难进行确定性 QoS 保证.到达曲线将输入特性描述为任意一个时间段内数据量的上限约束条件,并不需要精确地描述数据流上任意时刻的到达特性.对大多数应用中的数据流来说,可以通过历史特性或领域特征参数抽取出数据流在任意时间段内的爆发量上限.因此,本方法适用于大多数数据流应用环境.另外,到达曲线能够刻画很多常见的输入流特性.我们以数据流应用中典型的 JCP(jitter constrained periodic)[14]流为例,用到达曲线对输入特性进行建模.在实际的数据流应用中,数据采集设备通常周期性地取样,但由于网络传输延迟或前级处理,往往会带来流速的波动.从时间点 t_0(不失一般性,令 $t_0=0$)起,流中数据以平均时间间隔 T 连续到达,允许流数据的到达在一定时间范围内波动:提前τ到达或者推迟τ',但相邻的流数据到达的时间间隔不许小于 $D(D<T)$.可见,可以用四元组(D,T,τ,τ')唯一确定一个 JCP 流.当连续 l 个事件到达的时间间隔

都等于最小允许时间间隔 D 时,将其称作长度为 l 的流爆发.

定理 1. 由(D,T,τ,τ')四元组唯一确定的 JCP 流,其流爆发的长度 l 上限约束为 L,其中,$L=\left\lfloor\frac{\tau+\tau'}{T-D}\right\rfloor+1$.当 $l=L$ 时的流爆发称为最大流爆发;JCP 流的输入到达曲线为 $\alpha_{(D,T,\tau,\tau')}(t)=\begin{cases}\left\lceil\frac{t}{D}\right\rceil, & t\le KD\\ \left\lceil\frac{t+\tau+\tau'}{T}\right\rceil, & t>KD\end{cases}$,其中,$K=L-1$.

证明思路:显然,只有在爆发的开始时间为事件到达的最晚时间τ'的情况下才能获得最大流爆发.取 K 为连续最小允许时间间隔的个数,即 $K=L-1$,故有 $K=\max_{\tau'+kD\ge kT-\tau}(k\in N)=\left\lfloor\frac{\tau+\tau'}{T-D}\right\rfloor$,可得 $L=\left\lfloor\frac{\tau+\tau'}{T-D}\right\rfloor+1$.为了获得 JCP 流的输入曲线,考虑流爆发的最坏情况,即整个流都是由最大流爆发构成的事件序列.到达曲线的构造及证明从略.

从查询处理的基本单位来看,可以将查询的输入分为以下两种情况:① 选择、投影与静态表的连接等查询操作,它们对于每个输入元组可以进行独立的查询操作并获得相应的结果,因此,每个输入元组可以看作一个基本的查询任务;② 而对于滑动窗口上的 COUNT,SUM,MIN,MAX 等聚集查询来说,只有在窗口内所有输入数据都到达后才能获得查询结果.在这种查询中,每个窗口具有独立的语义,且查询 QoS 往往也是定义在窗口上的.如图 2 所示,随着窗口在原始元组流上不断滑动,一个个独立的"窗口元组"构成"窗口流"作为输入进入数据流系统.我们将每个窗口元组看作是一个基本的查询任务被调度处理,并且输入累积函数和处理累积函数也是以"窗口元组"的个数来度量的.



(a) Time-Based window
(a) 基于物理时间的窗口

(b) Count-Based window
(b) 基于个数的窗口

Fig.2 From tuple stream to window stream

图 2 元组流转换成窗口流

假设 $0<H\le Z$,对于数据流 S 上大小为 Z、跳数为 H 的滑动窗口 W 来说,窗口中总是包含最近 Z 个时间单位内到达数据流 S 的元组,且窗口每 H 个时间单位向前滑动一次.原始元组流的输入累积函数为 $R(t)$,输入到达曲线为α,用窗口输入累积函数 $R_W(t)$和窗口处理累积函数 $R_W^*(t)$ 分别表示已到达的窗口个数和系统已经被处理的窗口个数.根据窗口大小和跳数采用的时间单位不同,将窗口任务分为基于物理时间和基于个数[4]两种情况来考察:

- 基于个数的窗口:由 $R_W(t)=\left\lfloor\frac{[R(t)-Z,0]^+}{H}\right\rfloor$,定义 $\alpha_W=\frac{\alpha}{H}$ 为窗口流的到达曲线,有

$$R_W(t)-R_W(s)\le\alpha_W(t-s).$$

- 基于时间的窗口:由于 $R_W(t)=\left\lfloor\frac{[t-Z,0]^+}{H}\right\rfloor$,显然,窗口流是以 $1/H$ 匀速到达的,故窗口流的到达曲线为

$$\alpha_W=t/H.$$

这样,数据流上连续查询的输入模型就完全可以用符合其输入特性的到达曲线来描述.

2.3 查询服务质量约束

定义 7(服务曲线). 连续查询 Q 的输入累积函数和处理累积函数分别为 $R(t)$和 $R^*(t)$.β称为查询 Q 的服务曲线当且仅当β是广义增函数,并且$\beta(0)=0$,$R^*\geq R\otimes\beta$.其中,最小加卷积$\otimes$定义为

$$(f\otimes g)(t)=\inf_{0\leq s\leq t}\{f(t-s)+g(s)\},\ f,g\in F.$$

服务曲线的定义给出了查询处理累积量的一个下限,即最小的服务量.与到达曲线一样,服务曲线比单纯的服务速度、服务延时等约束使用起来更加灵活,可以表达出很多复杂的 QoS 需求.

数据流上查询处理的 QoS 要求形式多、内容广,例如处理性能、可靠性、安全性等.所有这些方面对数据流查询处理的服务提供来说都是很重要的.但是,针对数据流本身特有的实时性和无界性,与时间和空间相关的查询性能要求就愈显关键.因此,下面从时间和空间两个方面考虑数据流查询的 QoS 约束.

- 时间相关的 QoS 约束

规则 1. 要求响应延迟上限为 D,即延时 $d(t)$需要满足 $d(t)\leq D$,则定义服务曲线为 $\beta=\delta_D=\begin{cases}0, & \text{if } t\leq D\\ \infty, & \text{if } t>D\end{cases}$.

规则 2. 要求输出等待时间上限为 L,称 L 为查询的反应时间,即需满足$\pi(t)\leq\tau(t)+L$,则服务曲线受限于 $\max\{t:\beta(t)=0\}\leq L$,其中,$\tau(t)=\max\{s:s\leq t\wedge B(s)=0\}$且$\pi(t)=\min\{s:s>\tau(t)\wedge R^*(s)>R^*(\tau(t))\}$.

规则 3. 已知输出等待时间上限为 L,要求吞吐率下限为 P,即需满足 $R^*(t)-R^*(\tau(t))\geq P(t-\tau(t)-L)$,则定义服务曲线为 $\beta=\gamma_{P,L}=\begin{cases}0, & \text{if } 0\leq t>L\\ P(t-L), & \text{if } t>L\end{cases}$,其中,$\tau(t)=\max\{s:s\leq t \text{ and } B(s)=0\}$.

通常,称这种服务曲线为 rate-latency 约束.其中,$rate=P$,$latency=L$.

- 内存相关的 QoS 约束

规则 4. 要求查询输入等待队列的大小不得超过 M,即需满足$(R(t)-R^*(t))\times sizeof(r:r\in R)\leq M$,则定义服务曲线为 $\beta=\left[0,\alpha(t)-\dfrac{M}{sizeof(r:r\in R)}\right]^+$,其中,$\alpha$为该查询输入数据流的到达曲线,符号$[x]^+$表示 $\max\{x,0\}$.

- 复合的 QoS 约束:应用要求查询处理同时满足多个 QoS 约束

规则 5. 若查询要求满足的 N 种 QoS 各自相应的服务曲线为β_i,$i=1,\ldots,N$,则该查询的服务曲线可定义为

$$\beta=\sup_{i=1,\ldots,N}\{\beta_i\}.$$

3 确定性 QoS 的保证算法

确定性 QoS 是通过查询执行前的 QoS 可调度性验证和运行中的调度算法来保证的,下面分别详细加以介绍.

3.1 保证QoS的调度算法

在基于服务曲线的 QoS 保证中,由于各种查询 QoS 都被抽象为服务曲线,因此,调度的目标就是找到一种最优的调度策略,使得查询能够获得各自需要满足的服务曲线.可见,基于服务曲线的调度为保证各种查询 QoS 提供了一种通用的解决方法.根据服务曲线定义,定理 2 给出了为连续查询分配服务曲线的策略.

定理 2. 若查询 Q_i 的第 n 个查询任务 T_i^n 在 a_i^n 时刻到达,则定义 T_i^n 的 QoS 截止期 D_i^n 为

$$D_i^n=(R_i^*)^{-1}(n)=\min\{t\in N:R_i^n(t)\geq n\} \tag{1}$$

其中,

$$R_i^n(t)=\inf_{s\in[0,a_i^n]}[R_i(s)+\beta_i(t-s)] \tag{2}$$

如果 Q_i 的每个任务都可以在各自的 QoS 截止期前完成处理, Q_i 就获得了期望的服务曲线β_i.

证明:根据服务曲线的定义,公式(2)给出了 t 时刻之前应该提供给 Q_i 的最小服务量.假设在满足服务曲线β_i的前提下,T_i^n 可在其 QoS 截止期 D_i^n 之后的某个时间 t'输出,即 $R_i^*(D_i^n)\leq R_i^*(t')\leq n$.由 D_i^n 定义可知,$R_i^n(D_i^n)\geq n$,于

是,$R_i^*(t') \le R_i^n(D_i^n)$,这与服务曲线定义相悖.因此,为保证 Q_i 获得服务曲线β_i, D_i^n 是 T_i^n 最晚的处理结束时间.证毕. □

由于 EDF 算法在单处理机环境下对独立进程调度是最优的[15],因此理论上讲,在系统资源足够的前提下,按照 QoS 截止期对查询任务进行 EDF 调度是在保证各个查询服务曲线前提下的最优调度.换句话说,如果按照 QoS 截止期采用 EDF 算法不可调度,那么,其他任何调度算法都不能保证 QoS 的可调度性.但在实际数据流系统中,任务的处理是不能被中断的,如果在某任务的处理过程中有 QoS 截止期更紧急的任务到达,则新到达的优先级较高的任务只有等当前任务处理结束才能获得查询引擎.可见,这种优先级反转造成的任务调度延迟上限为系统内单个任务的最大处理时间.于是,为了确保每个连续查询获得 QoS 要求的服务曲线,定理 3 对分配给每个流的服务曲线进行修正.

定理 3. 在数据流系统中,为了确定性保证查询 Q_i 满足服务曲线β_i 所表示的 QoS 需求,定理 2 分配给查询 Q_i 的服务曲线应当修正为 $\beta_i^* = \begin{cases} 0, & t=0 \\ \beta_i(t+\Delta_{\max}), & t>0 \end{cases}$.其中,$\Delta_{\max}=\max\{\Delta_i\}$,$\Delta_i$ 为查询 Q_i 对单个任务的处理代价的上限.

证明:假设 Q_i 的第 n 个任务 T_i^n 在 a_i^n 时刻到达,并且在 d_i^n 时刻被处理完毕.由于 $d_i^n - \Delta_{\max} \le D_i^n$.由公式(2)可知,任意时刻 $t \le D_i^n$ 有 $\inf_{s\in[0,a_i^n]}[R_i(s)+\beta_i(t-s)] \le R_i^*(d_i^n)$,即 $R_i^*(d_i^n) \ge \inf_{s\in[0,a_i^n]}[R_i(s)+\beta_i((d_i^n-\Delta_{\max})-s)]$.

由服务曲线定义可知,按照定理 2 分配的 QoS 截止期进行 EDF 调度,每个查询实际获得的 QoS 是服务曲线 $\beta_i(t-\Delta_{\max})$.因此,为了使查询 Q_i 获得代表查询 QoS 需求的服务曲线β_i,在定理 2 中定义 QoS 截止期时所使用的服务曲线需要将β_i 向左平移$\Delta_{\max}$ 个单位,即修正后的服务曲线为 $\beta_i^* = \begin{cases} 0, & t=0 \\ \beta_i(t+\Delta_{\max}), & t>0 \end{cases}$.证毕. □

由定理 2 可以看出,任务的 QoS 截止期只与之前到达的任务有关,即每个任务到达输入队列时就可以计算其 QoS 截止期.可见,这种 QoS 截止期计算方法能够满足连续查询的实时性要求.另外,数据流中的查询通常是复杂的,对于同一个查询处理,不同任务的代价也不一定是相同的,比如,通过某个选择操作符的流数据由于要进行后续的操作,所以其占用的处理时间要长于没有通过选择操作符的流数据.为了保证可调度,上述定理中的 Δ_i 是任务的处理代价上限.这有可能使人认为这种算法会浪费一定的资源.实际上,这种调度方法是 working-conserving 的,即只要有任务在队列中时就会进行调度.在输入到达速度较快的情况下,如果多个查询都有任务在队列中,那么算法实际上还会按照分配的服务曲线分配优先权,也能保证一定程度上的公平性.

综上,算法 1 给出了保证查询 QoS 的查询调度算法 QED(QoS earliest deadline).

算法 1. QED 调度算法.

(1) 计算各查询修正后服务曲线 β_i^*;
(2) **while** (所有查询的输入队列不空){
(3) **for** (i=1 to N){
(4) $T_i^n = Q_i$ 输入队列最前端的任务;
(5) **if** (T_i^n 的 QoS 截止期尚未计算)
(6) 使用 β_i^* 计算 T_i^n 的 QoS 截止期 D_i^n;
(7) **endfor**
(8) 按照 EDF 策略调度各查询的就绪任务;
(9) **endwhile**

3.2 QoS可调度性验证

在数据流系统中,查询引擎的处理能力是有限的.当系统内有多个查询时,它们为了满足各自的 QoS 需求会相互竞争查询引擎.这就需要验证系统的 QoS 可调度性,使得查询引擎在保证各个查询 QoS 的前提下为各查询所共享.

定理 4. 数据流系统中有 N 个连续查询 Q_i,各自的 QoS 要求为 $QoS_{Q_i}=\langle\alpha_i,\beta_i\rangle$,单个任务的处理代价上限为 Δ_i,其中,$i=1,\ldots,N$.如果所有查询满足公式(3),则按照 QED 算法调度查询任务,可以保证各查询 Q_i 获得要求的服务曲线 β_i.

$$\sum_{i=1}^{N}(\alpha_i\otimes\beta_i^*)\times\Delta_i\le t,\ \forall t\ge 0 \tag{3}$$

其中,$\beta_i^*=\begin{cases}0, & t=0\\ \beta_i(t+\Delta_{\max}), & t>0\end{cases}$,并且 $\Delta_{\max}=\max\{\Delta_i\}$.

证明:根据定理 3,计算任务 QoS 截止期的服务曲线使用修正后的 β_i^* 可以确保查询 Q_i 获得服务曲线 β_i 所表示的 QoS.由于不同查询的任务处理的代价不同,所以,查询获得的处理能力不能以处理任务的个数为依据,而需要有一个统一的衡量标准,因此要对处理能力进行标准化.假设查询引擎单位时间能做 C 个基本操作,查询 Q_i 处理一个任务最多需要花费 C_i 个基本操作,则 $\Delta_i=C_i/C$.于是,$\sum_{i=1}^{N}(\alpha_i\otimes\beta_i^*)\times C_i$ 利用到达曲线和服务曲线的卷积,给出了最坏情况下,处理任意时刻 t 之前到达的所有任务所要求的基本操作的个数.而在 t 时刻之前,系统所能提供的总的处理能力为 $C\times t$.于是,若如下公式(4)总成立,则各查询可以获得期望的服务曲线,即系统总能满足各查询服务量的要求.

$$\sum_{i=1}^{N}(\alpha_i\otimes\beta_i^*)\times C_i\le C\times t,\ \forall t\ge 0 \tag{4}$$

化简公式(4),定理 4 中的公式(3)得证. □

到达曲线和服务曲线在查询定义时就可以获得,而查询对单个任务的处理代价在系统运行前可以通过测试统计得到.故在系统运行前就可以检测系统内查询的可调度性,从而先验地确定各查询是否能够保证用户的 QoS 要求.

4 确定性 QoS 保证的优化和实现

为了提高系统效率,可以从两个层面考虑资源的最小化:① 查询调度级优化;② 查询处理级优化.本文提出的确定性 QoS 保证是在查询调度的层面上通过查询调度和可调度性验证来完成的,也就是说,QoS 保证是进行的,不受查询执行策略的影响.因此,第 4.1 节和第 4.2 节分别从批调度和查询共享两方面进行优化.诚然,查询处理级优化能够有效地降低查询处理代价,节约的系统资源增益可以增大 QoS 可调度空间,但具体的优化策略不属于本文讨论的层面.

4.1 批调度优化策略

第 3.1 节中提出的 QED 算法是以单个任务为调度单位的.由于数据流具有快速、无限和大量的特点,这种一次 1 个任务的调度策略会带来较大的调度开销和上下文切换代价,而且对每个任务都需要计算 QoS 截止期.因此,将同一个查询的若干个任务组成一批,每调度一次查询就可以处理多个任务,不仅可降低查询的调度代价,同时也可相应降低 QoS 截止期的计算代价.但是,批的引入可能使一批中早到达的任务等待时间比较长;还有可能由于某个查询的批过大,而使其他查询不能得到及时的处理,从而影响其他查询的 QoS.因此,批大小的设置必须遵循以下原则:① 不破坏 QoS 可调度性;② 不增加运行时的计算和调度开销;③ 为了得到较高的处理效率,应尽量增大批大小.

根据上述原则,在 QED 算法的基础上提出了保证 QoS 的批调度算法 PQED(packaged QoS earliest deadline),其基本思想是,在满足可调度条件的前提下,尽量增大被调度查询的处理能力,即给被调度的查询分配尽量大的服务曲线,以使其获得较大的批大小.

为了在不破坏 QoS 可调度性的前提下尽可能地提高查询 Q_i 的服务曲线 β_i,考虑满足可调度性的极限情况,即提高后的服务曲线 $\beta_i^{\max}$ 正好能够使可调度性条件满足.根据定理 4 中的可调度性条件,$\beta_i^{\max}$ 可由公式(5)获得.

$$\sum_{j\neq i,i=1}^{N}((\alpha_j\otimes\beta_j^*)(t)\times\Delta_j)+(\alpha_i\otimes\beta_i^{\max})(t)\times\Delta_i=t,\ \forall t\geq 0 \tag{5}$$

当查询引擎切换到某个查询时,调度器需要知道该查询此次被调度所能处理的任务个数,即批大小.假设在 t 时刻查询 Q_i 获得查询引擎,且任务 T_i^n 在输入队列的最前端,在查询 Q_i 之外的所有查询的输入队列中,查询 Q_j 的第 m 个任务 T_j^m 具有最小 QoS 截止期 D_j^m,其中 $i\neq j$.可知,在查询 Q_i 处理结束后,下一个将要处理的任务就是 T_j^m.公式(6)给出了在调度查询 Q_j 之前,为了获得服务曲线 $\beta_i^{\max}$,查询 Q_i 此次调度应该处理的任务个数.

$$Z_i^n(t)=(\alpha_i\otimes\beta_i^{\max})(D_j^m-t) \tag{6}$$

考虑当前查询 Q_i 输入队列的任务个数 $B_i(now)$,于是,此次调度查询 Q_i 的批大小可由公式(7)确定.

$$BSize(t)=\min\{Z_i^n(t),B_i(now)\} \tag{7}$$

从上面的分析来看,批大小的计算只需要提高后服务曲线和相应到达曲线的卷积,而并不需要获得提高后服务曲线的表达式.由公式(5)可知,若已知各查询的到达曲线和服务曲线的卷积,则所有查询到达曲线和提高后服务曲线的卷积都可由其他查询的到达曲线与服务曲线的卷积线性表示.虽然卷积运算的复杂性相对较高,但是这些卷积运算都可以在系统运行前进行.因此,这部分计算并不会影响到数据上连续查询的运行时处理代价,故作为离线处理.而在实际查询调度的在线处理中,根据离线处理中计算出的到达曲线与提高后服务曲线的卷积函数,只需代入自变量 D_j^m-t 就可以获得为保证提高后服务曲线需要处理的最少任务量.再对当前输入队列大小取两者的最小值,就完成了批大小的计算.最坏情况是,当处理当前查询的过程中有某个查询新到达的任务具有最紧急的 QoS 截止期时,则需要重新计算在处理该任务之前,当前查询需要处理的最少任务量.可见,这种离线、在线分离的批大小计算方法在实际运行中的代价很小,不会增加运行时的计算和调度开销,而且能够有效地节省 QoS 截止期的计算代价和查询调度代价,十分适合数据流上的在线处理.另外,从公式(5)可以看出,批调度利用系统的空闲资源来增大被调度查询获得的服务能力,即批的大小受到系统资源空闲程度的限制.在系统的查询负载较轻的情况下,被调度查询获得的服务能力较大,批较大;而在载重的情况下,由于系统空闲资源较少,则不利于批的形成.

4.2 确定性QoS保证中的查询共享

假设 M 个连续查询 $Q_1,Q_2,\ldots,Q_M$ 有共享的查询分支 Q_0.经过共享分支 Q_0 后,查询的剩余查询分支分别记作 $Q_1',Q_2',\ldots,Q_M'$.共享查询分支接受相同的输入数据,进行相同的查询操作,并产生相同的分支结果.但对于每个查询来说,系统分配给每个查询的服务曲线也因 QoS 而异.因此,在用 QED 算法调度查询的过程中,同一个元组针对不同查询会有不同的 QoS 截止期.元组首先被 QoS 截止期较早的查询调度处理,为了避免重复计算共享分支,可以将共享分支的计算结果缓存在共享缓冲区内,QoS 截止期较晚的查询可以直接从共享结果缓冲区中读取共享分支的计算结果.这样,查询共享就能有效地降低查询处理代价.在确定性 QoS 保证方法中,有两处涉及了查询处理代价:

① 服务曲线修正.为了避免优先级反转,定理 3 在对服务曲线的修正过程中,将表示查询 QoS 需求的服务曲线向左平移了系统内单个任务的最大处理时间.在查询共享的情况下,仍存在优先级反转的问题,而且不可避免具有最大处理时间的查询首先处理共享的查询分支.因此,服务曲线修正仍需考虑这个最大处理时间.

② QoS 可调度性验证.QoS 可调度性验证的实质是,任意时刻之前,各个查询为获得各自服务曲线所需要的查询处理代价总和不超过系统所能提供的最大处理能力,才能保证 QoS 可调度.在查询共享的情况下,避免了系统对相同查询分支的重复计算,在一定程度上节约了查询处理代价.因此,QoS 可调度性条件应该更容易满足.

下面给出定理 5 讨论查询共享对 QoS 可调度性问题.

定理 5. 数据流系统中有 N 个连续查询,且前 M 个查询具有相同的共享分支.指定共享分支由前 M 个查询中的任何一个查询处理,而其他有共享分支的查询只处理剩余查询分支.如果所有共享分支的指定情况都满足 QoS 可调度性,则在实际的共享查询处理中,按照 QED 算法调度系统内的查询任务,可以保证各个查询获得所要求的服务曲线.

证明:假设共享分支对每个任务的处理代价为 C_0,各个查询剩余分支的处理代价分别为 C_i'.为保证各个查询获得期望的 QoS,在任意时刻 t 之前,任意查询 Q_i 必须处理的任务数为 $(\alpha_i \otimes \beta_i^*)(t), 1 \le i \le M$.然而在实际的查询分支共享的 QoS 保证中,查询共享分支对某个任务的处理不能确定到底由哪个查询执行.为了进行确定性 QoS 保证,必须考虑任意时刻要求任务处理量最大的查询首先计算共享分支.于是,为了满足 QoS 可调度性,必须满足下式:

$$\sum_{i=1}^{M}(\alpha_i \otimes \beta_i^*)(t) \times C_i' + \sup_{1 \le j \le N}\{\alpha_j \otimes \beta_j^*\}(t) \times C_0 + \sum_{i=M+1}^{N}(\alpha_i \otimes \beta_i^*)(t) \times C_i \le C \times t,\ \forall t \ge 0 \tag{8}$$

假设第 j 个查询处理共享分支,则可调度性验证条件为

$$\sum_{i=1, i \ne j}^{M}(\alpha_i \otimes \beta_i^*)(t) \times C_i' + (\alpha_j \otimes \beta_j^*)(t) \times (C_0 + C_j') + \sum_{i=M+1}^{N}(\alpha_i \otimes \beta_i^*)(t) \times C_i \le C \times t,\ \forall t \ge 0 \tag{9}$$

如果将共享分支指定给任意查询都满足公式(9),即

$$\forall t \ge 0, \forall 1 \le j \le M, (\alpha_j \otimes \beta_j^*)(t) \times C_0 \le C \times t - \sum_{i=1}^{M}(\alpha_i \otimes \beta_i^*)(t) \times C_i' - \sum_{i=M+1}^{N}(\alpha_i \otimes \beta_i^*)(t) \times C_i,$$

则

$$\forall t \ge 0, \sup_{1 \le j \le N}\{\alpha_j \otimes \beta_j^*\}(t) \times C_0 \le C \times t - \sum_{i=1}^{M}(\alpha_i \otimes \beta_i^*)(t) \times C_i' - \sum_{i=M+1}^{N}(\alpha_i \otimes \beta_i^*)(t) \times C_i.$$

故公式(8)成立.因此,我们可以将查询共享分支指定给任意一个查询执行进行可调度性验证,如果所有查询分担共享分支的情况都满足可调度性条件,则在实际共享查询处理中一定能够满足各个查询的 QoS.证毕. □

从公式(9)可以看出,若第 j 个查询分担共享分支,则与没有查询共享的情况相比,系统需要的处理资源降低了 $\sum_{i=1, i \ne j}^{M}(\alpha_i \otimes \beta_i^*)(t) \times C_0$ 个基本操作.如果共享分支处理代价比较大,那些在不共享情况下不可调度的查询,在共享查询分支的情况下都可能满足 QoS 可调度性.

4.3 算法部署

图 3 给出了保证 QoS 的数据流系统结构.系统接收注册的查询后,QoS 建模器将查询的 QoS 需求用到达曲线和服务曲线表示,调度验证器确定各查询能否在满足各自 QoS 的前提下可调度.若可调度性,调度器遵循 QED 算法从输入任务队列取出任务交与查询引擎执行.调度器中维护一个有序的查询优先级队列,各查询的优先级定义为该查询输入队列中最早的 QoS 截止期,且 QoS 截止期越早,优先级越高.调度器每次选取优先级最高的查询调度,并由批大小确定器计算该查询此次执行所能处理的任务量.

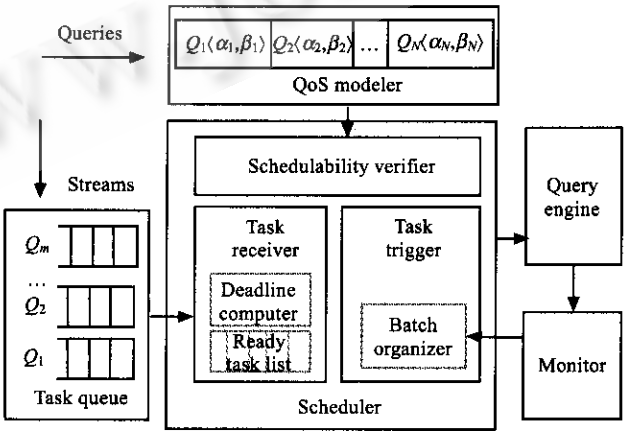


Fig.3 Architecture of QoS-guaranteeing DSMS

图 3 保证 QoS 的数据流系统结构

表 1 给出了保证 QoS 的查询过程中,任务接收器、任务触发器和批大小确定器的工作机制.

Table 1 Runtime working mechanism

表 1 运行时工作机制

Task trigger	
(1) **while** (TRUE)	(6) **if** (task queue *i* is empty)
(2) Get query with highest priority from *PriorityOrderList*, denoted by Q_i;	(7) Set priority of Q_i to ∞ and reorder *PriorityOrderList*;
(3) *curBatchSize=Batch organizer*();	(8) **endif**
(4) **while** (queue size≤*curBatchSize*)	(9) **endwhile**
(5) Remove and process a task;	(10) **endwhile**
Task receiver	
(1) **while** (T_i^n becomes the most front task of queue *i*)	(5) **if** (Q_i is query with the highest priority except current running query)
(2) Compute deadline D_i^n;	(6) *bs*=Call *Batch Organizer*();
(3) Assign D_i^n as the priority of query Q_i;	(7) *curBatchSize=bs*;
(4) Reorder *QueryPriorityList*;	(8) **endif**
	(9) **endwhile**
Batch organizer	
Offline:	Online:
(1) $U_i(t)=(\alpha_i \otimes \beta_i)\times \Delta_i,\ i=1,\ldots,N$;	(1) *P*=priority of query with highest priority except running query;
(2) $(\alpha_i \otimes \beta_i^{\max})(t)=\frac{1}{\Delta}\left(t-\sum_{j=1,j\neq i}^{N} U_i\right),\ i=1,\ldots,N$	(2) $\tau=P-t$;
	(3) $BatchSize(i,t)=(\alpha_i \otimes \beta_i^{\max})(\tau)$
	(4) Return *BatchSize*;

4.4 关于QoS适应性调整的讨论

考察定理 4 给出的 QoS 可调度性验证条件,系统中查询计划一旦确定,则查询处理一个元组最多花费的基本操作处理个数 C_i 就是一定的.这样,可调度性就完全取决于输入到达曲线和查询需要保证的服务曲线以及系统的处理能力.对于 Q-Space 中的任意一点 $P(A,B,1/C)$,如果 P 不可调度,则可以通过调整 A,B,C 使得调整后的系统能够满足可调度性,这个过程称为 QoS 的适应性调整.不失一般性,可以分别通过以下 3 种途径实施 QoS 的适应性调整,具体的调整策略超出了确定性 QoS 保证的讨论范围,将作为未来的工作加以研究.

- 系统资源推荐.如果系统处理能力提高到原来的 k 倍,即 $C'=kC$,则公式(4)左侧变为原来的 $1/k$.可见,通过选取合适的 k 值,即向用户推荐系统计算能力的提高程度,可以把系统调整到一个新的可调度状态.
- QoS 降级.降低 QoS 要求则会使相应的服务曲线变小,根据卷积运算的保序性,到达曲线和服务曲线的卷积也就会随之减小.因此,可以在用户允许的范围内降低查询的 QoS 需求而使系统达到可调度状态.
- 输入降载.若以采样率 $\rho(0<\rho<1)$ 对输入流采样,则到达曲线变为 $R'(t)=\rho R(t)$.对任意 $s\le t$,有

$$R'(t)-R'(s)\le \rho\alpha(t-s).$$

令 $\alpha'(t)=\rho\alpha(t)$,则输入到达曲线降低为 α',选取合适的采样率可使系统满足 QoS 可调度性条件.

5 实 验

对于保证查询 QoS 的处理机制来说,最重要的衡量标准就是系统是否对各个查询的 QoS 提供确定性的保证.因此,这里采用查询的 QoS 错失率(QoS missing ratio,简称 QMR)作为确定性保证的指标.

用 QMR_Q 表示查询 Q 处理过程中,任务错失 QoS 的比率,即

$$QMR_Q=\frac{\text{查询}Q\text{中满足QoS的任务总数}}{\text{查询}Q\text{处理的任务总数}}\times 100\%.$$

用 $QMR_{overall}$ 表示系统内所有任务错失 QoS 的比率,即

$$QMR_{overall}=\frac{\text{所有查询满足QoS的任务总数}}{\text{所有查询处理的任务总数}}\times 100\%.$$

当 $QMR_{overall}$ 为 0 时,表明所有查询对任务的处理都没有 QoS 错失.

在相同的实验环境下,将 QED(PQED)算法与先进先出(first in first out,简称 FIFO)调度算法、最短处理时间

优先调度(shortest processing time,简称 SPT)算法、时间片轮转(round robin,简称 RR)算法进行对比.实验使用 Java 编程语言开发,在 JDK1.5 虚拟机上运行,系统环境为 WindowsXP SP2,PentiumM 1.73GHz CPU,512MB 主存的微机.

5.1 QoS保证的性能分析

如表 2 所示,实验设计了两种输入模型和两种 QoS 需求,随机组合输入模型和 QoS 需求,共生成 10~450 个查询.其中每种输入模型和 QoS 需求被选中的概率都为 0.5.不同的 JCP 流的平均到达速率按正态分布随机产生;速率上限约束流的任务到达间隔按负指数分布产生,且不同流的速率上限服从正态分布.不同查询的单个任务处理代价上限按正态分布随机产生,且每个查询的不同任务处理代价也服从正态分布.按照此实验方案处理 300 秒数据.

Table 2 Parameters for data set

表 2 数据集参数

Input model		QoS requirements		Processing cost per task (ms)
JCP stream (ms)	Rated upper bounded stream (each/ms)	Delay (ms)	Output (each/ms)	
$T\sim N(20,2)$, $D=T/2$, $\tau=2T$, $\tau'=1.5T$	Rate upper bound: $r\sim N(0.05,0.005)$	$D\sim N(10,1)$	Throughput$\sim N(0.05,1)$	$\Delta\sim N(0.018,0.002)$

改变系统中的查询数目,根据可调度性验证可知,当查询个数超过 403 个以后,就无法满足 QoS 可调度性条件.如图 4 所示,在可调度范围内,QED 和 PQED 算法的 $QMR_{overall}$ 始终为 0,表明所有查询对所有任务的处理都能满足各自的 QoS 需求.而对于 FIFO,SPT 和 RR 算法来说,即使在 QoS 可调度范围内也有任务错失 QoS,特别是 FIFO 算法和 RR 算法有大量的任务错失 QoS.SPT 算法的 $QMR_{overall}$ 虽然比较小,但 $QMR_{overall}$ 是以所有查询的任务为基数计算的,而 SPT 算法偏向于任务处理代价小的任务,这样,SPT 算法的吞吐量较大,并且处理的任务大都能满足 QoS,因此表现出的 $QMR_{overall}$ 较优.如果考察某些任务处理代价较大的查询,QMR 将会很高.可见,在 QoS 可调度范围内,QED(PQED)算法表现出了明显的 QoS 确定性保证特性,是其他算法所不能比拟的.

其次,在 QoS 可调度性验证的计算中,当查询数目超过 403 个以后将不能满足 QoS 可调度性验证条件.但是从实际的实验结果来看,查询个数超过 403 个以后,QED(PQED)算法的 $QMR_{overall}$ 仍然保持在 0,直到查询个数超过 420 个以后才有 QoS 错失情况出现.其原因是,为了确保 QoS,QoS 可调度性验证是以处理代价上限和输入到达上限情况来考虑的,因此有资源的预留.从本次实验的数据来看,在查询个数为 400 左右的情况下,有不到 20 个查询能够在理论上不满足 QoS 可调度性验证的情况下仍然保证 QoS 无错失,即资源预留大约在 5%还是可以接受的.在 400 个查询的执行中,保持实验参数不变,改变系统内所有查询的输入流速、查询吞吐率约束以及系统处理能力,实验结果如图 5~图 7 所示,箭头标出了可调度性验证条件的临界情况.由实验结果可知,在 QoS 可调度范围内,QED(PQED)算法始终能够确保各个查询的 QoS 无一错失.

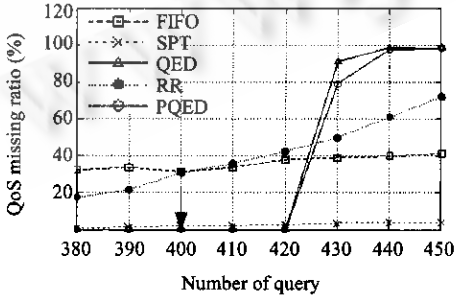


Fig.4 Variation of query number

图 4 查询个数的变化

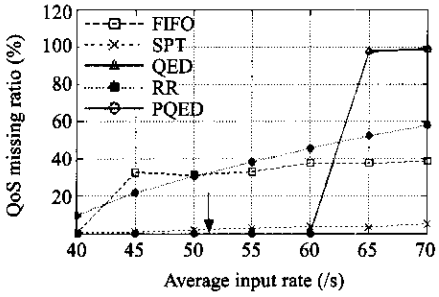


Fig.5 Variation of input rate

图 5 输入流速的变化

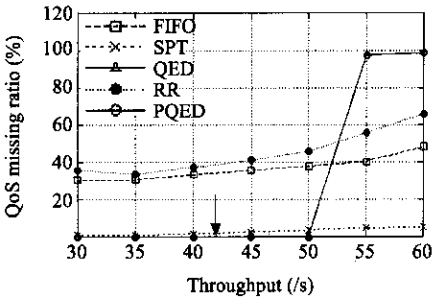


Fig.6 Variation of throughput lower bound

图 6 吞吐率下限约束的变化

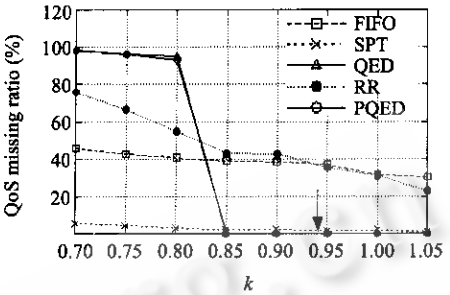
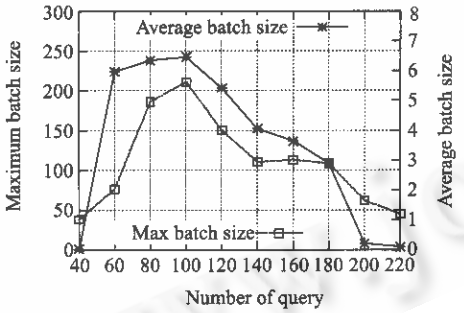


Fig.7 Variation of system capability

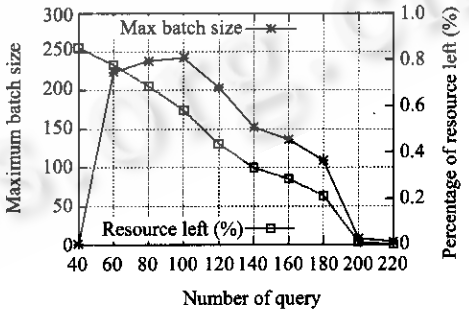
图 7 系统处理能力的变化

5.2 批调度优化的性能分析

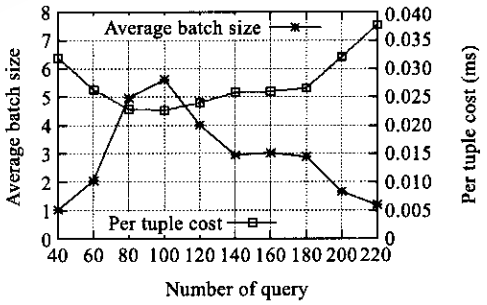
由于 PQED 批调度算法是在满足 QoS 可调度性验证的前提下进行的,因此,PQED 算法与 QED 算法具有相同的可调度范围和 QoS 保证程度.因此,在图 4~图 7 所显示的实验结果中,PQED 调度下的 $QMR_{overall}$ 与 QED 调度下的 $QMR_{overall}$ 基本一致.如图 8(a)所示,在查询数目较少时,由于系统负载较低,到达的任务都能很快被查询处理掉,因此查询输入队列长度较小,批大小受队列长度的限制.随着系统内查询数目的增大,系统负载加大,查询输入队列长度变大,因此,批大小逐渐增大.当查询数目增大到一定程度以后,批大小受限于查询提高后的服务曲线所提供的服务能力.由于批调度是利用系统的空闲资源来增大被调度查询获得的服务能力,如图 8(b)所示,随着查询数量的增大,系统内空闲资源的百分比逐渐下降.因此,在查询数目继续增大的情况下,系统内查询的最大批大小和平均批大小都逐渐降低.另外,由图 8(c)可见,随着批大小的增大,任务平均处理时间也有所减小.可见,批调度的引入在保证查询 QoS 的前提下有效地提高了查询处理的效率.



(a)



(b)



(c)

Fig.8 Performance of batch scheduling

图 8 批调度性能

5.3 查询共享优化的性能分析

为了考察保证 QoS 的查询共享优化的有效性,实验设计系统中有 4 个查询,参数见表 3.其中,前 3 个查询有共同的查询分支,实验逐渐增大共享分支的处理代价.在不考虑查询共享的情况下对 4 个查询进行 QoS 可调度性验证,不能满足 QoS 可调度性条件.而在前 3 个查询考虑共享查询分支的情况下进行 QoS 可调度性验证,当查询的共享分支处理代价超过 0.914ms 以后,能够满足可调度性验证条件.如图 9 所示,随着共享分支代价的增大,各个查询的 QoS 错失率逐渐变小,当共享分支处理代价增大到 0.86ms 时,各查询的 QoS 错失率就已为 0.由此可知,查询共享下的 QoS 可调度性验证能够有效地提高系统的可调度空间,但仍然存在资源预留的问题.值得注意的是,4 个查询的 QoS 错失率的变化几乎一致,这验证了 QED 算法在 QoS 满足程度上具有很强的公平性.

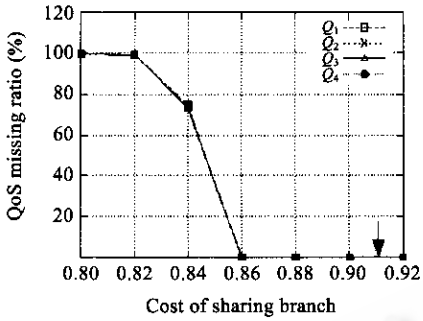


Fig.9 Effect of sharing degree on QMR

图 9 共享程度对 QMR 的影响

Table 3 QoS parameters in the example

表 3 例子中的 QoS 参数

Query	Input stream	QoS requirements	Δ upper bound (ms)
Q_1	JCP(1/0.8,1/0.25,6,4)	Delay constraint D=10ms	1.5
Q_2	Idem	Delay constraint D=13ms	2
Q_3	Idem	Delay constraint D=10ms	1.5
Q_4	Rate upper bound R=200each/s	Rate-Latency r=150each/s, latency=3ms	1.4

6 结论和未来工作

很多数据流应用对数据流系统的查询性能的要求都比较严格,因此,如何确定性地保证数据流上查询处理的服务质量对数据流系统的设计实现是十分关键的.本文以网络演算为理论依据,提出了一种数据流上通用的确定性 QoS 保证方法.该方法提供的查询 QoS 是一种先验的服务质量保证,即在查询执行之前就能判定数据流处理系统是否能够保证用户要求的查询服务质量.这种确定性的查询服务质量保证突破了以往数据流处理的尽最大努力的查询服务.实验结果表明,该方法能够有效地为连续查询提供确定性的查询服务保证.本文讨论的是确定性 QoS 的保证,即 100%的 QoS 保证.在今后的研究中,将把 QoS 的保证问题扩展到如何提供保证带有置信度的查询 QoS 上.

References:

[1] Carney D, Cetintemel U, Rasin A, Zdonik S, Cherniack M, Stonebraker M. Operator scheduling in a data stream manager. In: Freytag JC, Lockemann PC, *et al.*, eds. Proc. of the 29th VLDB Conf. San Fransisco: Morgan Kaufmann Publishers, 2003. 838–849.

[2] Avnur R, Hellerstein JM. Eddies: Continuously adaptive query processing. In: Chen W, Naughton JF, *et al.*, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2000. 261–272.

[3] Babcock B, Babu S, Datar M, Motwani R. Chain: Operator scheduling for memory minimization in data stream systems. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2003. 253–264.

[4] 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: Proc. of the 1st Biennial Conf. on Innovative Database Research. Asilomar: Morgan Kaufman Publishers, 2003. 245–256.

[5] Abadi D, Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S. Aurora: A new model and architecture for data stream management. The VLDB Journal, 2003,12(2):120–139.

[6] Chen J, DeWitt D, Tian F, Wang Y. NiagaraCQ: A scalable continuous query system for internet databases. In: Chen W, Naughton JF, *et al.*, eds. Proc. of the SIGMOD Conf. New York: ACM Press, 2000. 379–390.

[7] Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein M, Hong W, Krishnamurthy S, Madden S, Raman V, Reiss F, Shah M. TelegraphCQ: Continuous dataflow processing for an uncertain world. In: Proc. of the 1st Biennial Conf. on Innovative Database Systems Research. Asilomar: Morgan Kaufman Publishers, 2003. 269–280.

[8] Carney D, Cetintemel U, Cherniack M. Monitoring streams: A new class of data management applications. In: Freytag JC, *et al.*, eds. Proc. of the 28th VLDB Conf. San Fransisco: Morgan Kaufmann Publishers, 2002. 215–226.

[9] Sutherland TM, Pielech B, Zhu Y, Ding L, Rundensteiner EA. An adaptive multi-objective scheduling selection framework for continuous query processing. In: Proc. the 9th Int'l Database Engineering and Applications Symp. Montreal: IEEE Computer Society, 2005. 445–454.

[10] Schmidt S, Berthold H, Lehner W. Qstream: Deterministic querying of data streams. In: Nascimento MA, Özsu MT, Kossmann D, *et al.*, eds. Proc. of the 30th VLDB Conf. San Fransisco: Morgan Kaufmann Publishers, 2005. 1365–1368.

[11] Schmidt S, Legler T, Schar S, Lehner W. Robust real-time query processing with Qstream. In: Böhm K, Jensen CS, Haas LM, *et al.*, eds. Proc. of the 31st VLDB Conf. New York: ACM Press, 2005. 1365–1368.

[12] Le Boudec, JY, Thiran P. Network calculus. In: Lecture Notes in Computer Science, Vol.2050. Berlin: Springer-Verlag, 2001.

[13] Sariowan H. A service curve approach to performance guarantees in integrated service networks [Ph.D. Thesis]. San Diego: University of California, 1996.

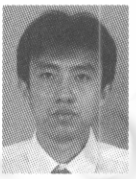
[14] Hamann CJ. On the quantitative specification of jitter constrained periodic streams. In: Proc. of the 5th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Washington: IEEE Computer Society, 1997. 171–176.

[15] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery, 1973,20(1):40–61.

武珊珊(1980－),女,辽宁沈阳人,博士生,主要研究领域为数据流.

于戈(1962－),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,数据流,传感器网络.

吕雁飞(1984－),男,硕士生,主要研究领域为数据流.



谷峪(1981－),男,博士生,主要研究领域为数据流.

李晓静(1983－),女,硕士生,主要研究领域为数据流.