ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.7, July 2007, pp.1831−1843
DOI: 10.1360/jos181831

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

支持截止期敏感应用的数据流任务调度方法*

于 戈+, 李晓静, 杨晓春, 欧征宇, 邓庆绪

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

Stream Task Scheduling Method for Deadline-Sensitive Applications

YU Ge+, LI Xiao-Jing, YANG Xiao-Chun, OU Zheng-Yu, DENG Qing-Xu

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

+ Corresponding author: Phn: +86-24-83683113, Fax: +86-24-23895654, E-mail: yuge@mail.neu.edu.cn, http://www.ise.neu.edu.cn

Yu G, Li XJ, Yang XC, Ou ZY, Deng QX. Stream task scheduling method for deadline-sensitive applications. *Journal of Software*, 2007,18(7):1831−1843. http://www.jos.org.cn/1000-9825/18/1831.htm

Abstract: Most of the existing real-time processing systems over data streams focus on minimizing average tuple latency while less attention has been paid to deadline of each individual tuple. This paper presents a real-time adaptive batch task scheduling (ATS) mechanism to support the strict deadline requirements of mission-critical applications over time-varying and bursting data streams. The ATS strategy aims at maximizing task throughput and minimizing deadline miss ratio by minimizing both scheduling overheads and deadline miss overheads. The paper proposes a concept of the optimal scheduling unit—batch granularity, and designs a closed-loop feedback control mechanism to adaptively select the dynamic optimal batch size in a non-predictable data stream environment. The theoretical analyses and experimental results show the efficiency and effectiveness of the ATS batching technique.
Key words: data stream management; real-time task scheduling; query processing; deadline; feedback control

摘 要: 当前,数据流上的实时处理系统大多关心平均元组延时最小化要求,而很少考虑每个元组的截止期要求.提出一种实时的自适应批任务调度策略——ATS(adaptive batch task scheduling),以支持时变突发的数据流上关键任务的严格截止期需求.ATS 调度策略可以降低调度开销和过期处理开销,从而实现截止期错失率最小化和有效任务完成率最大化.提出了最优调度单位概念——批粒度,设计了闭环反馈控制机制,以在不可预测的数据流环境中自适应地动态选择最优批大小.理论分析和实验表明了 ATS 批调度策略的有效性和高效性.
关键词: 数据流管理;实时任务调度;查询处理;截止期;反馈控制
中图法分类号: TP311 文献标识码: A

随着传感器网络以及通信技术的发展,在许多应用中,需要处理大量、连续的实时数据.提供该类数据管理功能的系统称为数据流管理系统(data stream management system,简称 DSMS)[1−6].对于设定的多个连续查询,DSMS 必须在某一给定时间内处理每个到达的流数据.流数据到达的速率决定了 DSMS 的负载.如果流数据到达速率过大,由于 DSMS 没有足够的资源在特定时间内处理完到达的每个流数据,将会发生过载.通常情况下,该类流数据源产生于监控类应用中,该类应用中的数据源分为下述 3 种工作模式:(1) 时间驱动,即数据在固定

* Supported by the National Natural Science Foundation of China under Grant Nos.60473073, 660503036 (国家自然科学基金)
Received 2006-08-29; Accepted 2006-11-21

时间内产生.这种情况下,流速是可预测的,具有稳定性、周期性等特点;(2) 事件驱动,即数据只有在事件驱动时产生.这种情况下,流速是不可预测的,具有不稳定性、非周期性等特点;(3) 时间驱动和事件驱动相结合.例如,发电厂内的监控和故障系统,机器状态数据如温度、速度、加速度会在固定时间间隔内被收集.而当探测到异常时,会收集更多的状态数据.因此,在第 1 种工作模式下,可以系统负载建立数学模型并设计静态调度策略;而在另外两种工作模式下,必须采用动态适应性调度策略.

许多截止期敏感的流式关键任务应用中,如高速公路上的交通监控、工厂生产线管理、战场指挥以及传感器网络等,要求查询必须实时处理到达的每一个流数据.例如,在一个智能交通指挥系统中,当某一给定路段发生的一起交通事故被发现时,系统必须通知附近的车辆,进行疏导以避免该路段造成拥挤等.如果某车辆在进入该路段之前未能收到系统的通告,则认为通告过期,且在此之后收到的通告也被视为无效.

现今的 DSMS 大多支持软实时处理,适用于输入流数据速率具有稳定性的实时应用.一般具有如下特征:(1) 最小化平均元组延时以满足 QoS 要求;(2) 优化操作符调度策略以降低系统开销,并提高系统内存等资源利用率.然而在这些系统中,绝大多数调度策略不能满足单个元组的截止期需求.

我们认为,对于一个查询而言,一个流数据的延时可定义为该流数据到达系统后到该流数据被查询处理完的时间间隔.若该间隔小于该查询的截止期,则认为查询结果有效,否则认为无效.很显然,一批流数据的平均元组延时小于查询截止期,并不意味着每个元组均在截止期内被处理.极端情况下,可能大多数被处理的元组都不满足各自的截止期.因此,在截止期敏感的应用中,必须考虑单个元组的延时而不是平均元组延时.

给定一个查询,不同的流数据可能有不同的到达时间.因此,它们对应的紧急程度也不同.因此,需根据数据的紧急程度逐个处理到达的流数据.然而,当前的操作符调度策略[7−12]只关心每个操作符队列能处理尽可能多的流数据,而很少考虑这些流数据紧急程度的差别;另一方面,由于数据流的突发性,在数据驱动的监控应用中,若将一个查询处理一个流数据看作一个流任务,将产生大量的流任务.调度处理这些流任务,传统调度策略[13−16]将产生大量的调度开销,因此也不能适用于数据流上的截止期调度.

由于上述原因,当前 DSMS 中的操作符调度策略以及实时数据库中的任务调度方法已经不能完全满足截止期敏感的流式应用的需求.因此,本文重点研究在时变、突发性数据流环境下如何设计截止期调度策略,使得在截止期内完成尽可能多的流任务.

实时调度策略的性能表现在如下两个方面:(1) 系统调度开销,包括队列管理、内存管理以及上下文切换;(2) 过期处理开销.若一个流任务已经过期,却有可能仍然被系统调度处理,则该部分的开销被认为是过期处理开销.批处理策略通过分批式调度可显著降低调度开销.当前数据流上的批处理策略[17−20]主要是基于计数的方法,即通过静态的统计计算,估算出分批的元组个数,对一批中的所有元组进行统一调度.但由于没有考虑数据流的时变性、突发性以及不同元组之间的截止期区别,而不能进行精确的截止期调度.例如,对一批新到达的流数据(100 个元组),估算有 80 个可在截止期内完成,则采用降载策略(load shedding),丢掉其中 20 个,将其余 80 个作为一批调度处理.然而,实际上在截止期内完成的流数据很可能多于或少于 80 个,相应地,可能会多丢弃了一些可在截止期内处理完的流数据,或者会造成系统资源浪费,多处理了一些过期数据,这样都影响了系统的性能.

因此,在实时数据流处理中,要设计一种优化的批任务调度机制必须解决下面两个关键问题:

- 对给定的工作负载,如何确定最佳的批大小.
- 在不可预测的数据流环境中,如何动态调整批的大小以适应数据流特性的改变.

前者属于静态调度问题,而后者属于动态调度问题.调度的目标是各查询在截止期内处理尽可能多的已到达的流数据.

本文第 1 节给出截止期敏感的数据流实时任务以及任务调度的相关概念定义.第 2 节介绍调度系统模型.第 3 节提出 ATS 调度策略和相关调度算法.第 4 节对本文提出的调度算法及相关算法进行实验与评价.第 5 节讨论相关工作.第 6 节总结全文,并提出进一步研究的目标.

1 问题描述

这一节中,本文首先讨论实时数据流上的连续查询任务,然后给出系统性能指标的定义,最后给出本文调度策略中的关键因素——批粒度和调度批的定义.

1.1 连续流任务

在流式应用中,数据以流的形式连续不断地到达,每个新的流数据到达时,驱动了每个查询的执行.在一个DSMS 中可能会注册大量的连续查询,而每个查询需要的处理时间可能不同.截止期是定义在查询上的,而不是定义在查询计划内部的一个或多个操作符上,使得基于操作符的调度策略不再适用于截止期调度策略.因此,本文将整个查询看作一个任务单位.

不失一般性,本文假设每个输入的数据流仅与一个查询关联,而且所有查询均有而且仅有一个输入的数据流.如果物理的输入数据流之间存在相关性,那么本文通过下述方法使它们在逻辑上相互独立:1) 若某个数据流被多个查询作为输入,则将该数据流复制成多个逻辑副本,分别被各个查询使用;2) 若某个查询有多个输入数据流(如查询中存在连接操作),则将这多个输入数据流合并成一个单逻辑流.如何实现不同查询之间的物理数据流共享问题则由查询引擎解决,不是本文的研究重点.在下面的讨论中所提到的数据流均指这样的逻辑流.因此,系统中各个查询任务之间互相独立且每个任务可被独立调度.

首先,本文给出数据流管理系统中数据的定义.

定义 **1**(基本数据和查询数据单元). 数据流上最小的数据单位称为基本数据,记为 x,形式化描述为$\langle r,\tau\rangle$.其中,r 是数据元组,τ是该数据元组对应的时间戳,表示该数据元组产生时对应的系统时间.一个查询所处理的一个基本数据称为一个查询数据单元,记为 D_Q,形式化描述为$\langle rs_Q,\tau_Q\rangle$,$rs_Q$ 为数据元组集合,τ_Q 是对应的时间戳.

对于非阻塞性查询而言,数据元组集合只包含一个基本数据,即 $D_Q=\langle\{r\},\tau\rangle$;而对于阻塞性查询,如窗口查询,有 $D_Q=\langle r\text{-}set,\tau_l\rangle$,这里,数据元组集合包括窗口内的所有基本数据,$\tau_l$ 为窗口内最旧的基数据对应的时间戳.滑动窗口查询以及 Landmark 查询均属于窗口查询.

下面再给出数据流上查询任务的定义.

定义 **2**(基本任务和连续任务). 一个查询 Q 处理一个查询数据单元 D_Q 称为一个基本任务 T,记为$\langle Q,D_Q\rangle$.一个连续查询 Q 处理一个该查询的输入数据流 S 称为一个连续任务 CT,形式化描述为$\langle Q,S\rangle$,$\langle Q,S\rangle=\langle\langle Q,D_{Q1}\rangle,\ldots,\langle Q,D_{Qi}\rangle,\ldots\rangle$,其中,$\langle Q,D_{Qi}\rangle$为第 i 个基本任务.

实际上,一个连续任务是由基本任务构成的集合.由此,本文给出数据流上任务截止期的定义.

定义 **3**(截止期和紧急时间). 对于一个基本任务$\langle Q,D_Q\rangle$,查询数据单元 D_Q 中的流数据在系统中的处理延时上限称为查询 Q 对于 D_Q 的截止期,记为 dl.而每个基本任务在其截止期到达之前的剩余处理时间称为查询 Q 对于 D_Q 的紧急时间,记为 u,从而有 $u=dl-CurrentTime$,$CurrentTime$ 为当前系统时间.

1.2 性能指标

在传统实时系统中,衡量实时调度策略的性能指标主要是截止期错失率(deadline miss ratio,简称 DMR),即未能在截止期内完成的任务数目与任务总数的比值.然而在数据流管理系统中,一个连续任务对应一个无界的任务流.因此,任务总数趋于无限.而且,数据流管理系统中可能存在大量的连续查询,对应了大量的连续任务.因此,DMR 已经不能用来衡量数据流上截止期调度策略的性能.下面给出数据流上任务调度的性能指标定义.

定义 **4**(**stream deadline miss ratio**,简称 **SDMR**). 给定一个时间段Γ,$SDMR(\Gamma)$定义如下:

$$SDMR(\Gamma)=M(\Gamma)/N(\Gamma) \tag{1}$$

这里,$N(\Gamma)$表示在时间段Γ内产生的基本任务总数,而 $M(\Gamma)$表示其中未能在截止期内完成的基本任务数目.

不难看出,$SDMR(\Gamma)$越小,系统有效输出越大,则系统性能越好.同时,在不可预测的数据流环境中,通过计算$SDMR(\Gamma)$,可以实时监控系统性能,并根据流特性的变化适应性地调整调度策略.

1.3 批单位和调度单位

为了保证每个元组的截止期约束,实现精确的截止期调度,与传统的基于计数的分批方法不同,本文设计了一种基于时间的任务分批策略.

定义 5(批任务、批长度和批大小). 对于连续任务$\langle Q,S\rangle$,在时间段$[l,u]$内所有的基本任务 T 组成一个批任务,时间间隔 $u-l$ 定义为批长度.一个批任务中的基本任务个数称为批大小,等于批长度乘以输入流的流速.

直观地,如果每次调度具有那些相同截止期的任务,则可以减小系统开销和过期处理开销.所以,具有相同截止期的任务可以组成一个基本批.下面我们将说明怎样确定两个任务是否具有相同的截止期.

假设用 Φ_Q 表示查询 Q 的截止期的误差上限,那么对于两个基本任务$\langle Q,\langle rs_1,\tau_1\rangle\rangle$和$\langle Q,\langle rs_2,\tau_2\rangle\rangle$,如果$|\tau_1-\tau_2|\le\Phi_Q$,则它们具有相同的截止期.

定义 6(基本批和基本批长度). 对于一个连续任务$\langle Q,S\rangle$,给定一个时刻,若批任务内各基本任务具有相同的截止期,且不在同一批内的基本任务截止期不同,则该批称为一个基本批,它的长度称为基本批长度(也称为批粒度),记为Φ.换言之,给定时刻τ,在时间段$[\tau,\tau+\Phi]$内产生的所有基本任务属于同一个基本批.

由此可见,批粒度的大小需要根据事件的截止期敏感程度来决定.事件的截止期越敏感,即截止期时间越精确,则批粒度越小;反之,则越大.

对于一个紧急时间为 u,截止期误差上限为Φ_Q的查询,$\Phi=\Phi_Q$.如果 Q 为窗口类型查询,设其窗口跳步长度(按时间长度定义)的误差上限为Φ_w,如果$\Phi_w<\Phi_Q$,则$\Phi=\Phi_Q$;否则,$\Phi=\Phi_w$,即窗口长度为基本批长度.不失一般性,我们假设所有查询的截止期具有相同的误差上限且小于所有窗口型查询的跳步长度的误差上限,否则从中选择最小的作为Φ.

对一个连续任务$\langle Q,S\rangle$而言,最佳的批长度在$[\Phi,u]$范围内.因为如果一批的长度超过 u,一些任务可能在被调度前就过期.这样,可以通过一个正整数 k 来调整批的长度,即批的长度为 $k\times\Phi$.最佳批任务定义如下:

定义 7(调度批和调整因子). 设连续任务$\langle Q,S\rangle$的基本批长度为Φ,则长度为 $k\times\Phi$的批任务称为一个调度批,或一个调度单位,其中,k 是一个正整数,称为调整因子.

随着数据流速的改变,可以通过 k 来适应性地调整调度批的长度.如果流速很快,系统将过载,调度批长度应该减小,即减小 k 值;如果流速很慢,到达的数据相对会少,那么调度批长度应该增大,即增大 k 值.这样,使得每个调度批的大小基本均匀,保证了负载平衡.因此,如何获得调度效率最高的 k 值是解决问题的关键.

另外,不同的查询任务具有不同的截止期,调度器应该根据具体的应用需求分配给任务不同的优先级.优先级可用 EDF(early deadline first)[13],MUF(maximum-urgency-first)[14]或其他规则计算得出.

2 系统模型

本文提出的批任务调度策略是在数据流管理系统 RealStream 内实现的.下面介绍 RealStream 系统模型.

RealStream 系统模型如图 1 所示.当时间戳为τ_r的元组 r 到达系统时,Basic Task Collector 负责将该元组生成对应的查询数据单元 $D_Q=\langle rs_Q,\tau_Q\rangle$,并分配截止期 dl_Q,然后加入到对应查询 Q 的基本任务队列中.同时,它也负责按照上面提到的方法将物理流转换为逻辑流.

对于截止期为 dl_Q 的查询 Q_i,其对应的基本任务队列中数据采用$\langle D_Q,dl_Q\rangle$结构.其中,$1\le i\le n$,n 为 RealStream 系统中查询总数.假设所有任务采用相同的基本批长度Φ,否则将其中最小的Φ_i作为Φ.Basic Batch Manager 通过Φ对基本任务队列中的基本任务进行分批,将时间段Φ内产生的基本任务划分为一个基本批.例如,对一个查询 Q,设置一个开始时刻,时间戳在Φ内的基本任务组成一个基本批,然后排入对应的基本批队列.Task Scheduler 负责从基本批生成调度批并送往查询引擎执行.$O_{i1},O_{i2},\ldots$是查询 Q_i 的操作符队列,表示依次地对每一个元组进行操作.

图 2 给出了调度器的体系结构,它由 5 部分组成,即 Priorty Manager,Scheduler,Overdue Controller,SDMR Monitor 以及 Adaptivity Controller.前两部分构成 BTS(basic task scheduling)调度器.

Priority Manager 负责各批任务优先级的计算,然后从中选出优先级最高的批任务.本文优先级的计算方式

采用 EDF 规则,所以,批任务〈D_{Q_i}, dl_{Q_i}〉的优先级计算方式为

$$Priority(i)= dl_{Q_i} - CurrentTime \tag{2}$$

这里,*CurrentTime* 是当时的系统时间.*Priority*(*i*)值越小,则优先级越高.如果 *Priority*(*i*)<0,则对应的批任务已经过期.另外,很容易得出,在每个批任务队列中,第 1 个批任务是它所在批任务队列中优先级最高的.所以,Priority Manager 只需计算每个批任务队列中第 1 个批任务的优先级即可.

Overdue Controller 负责丢弃过期数据;SDMR Manager 负责计算每次执行后 *SDMR* 值,衡量系统的性能;Adaptivity Controller 负责通过从 SDMR Manager 得到的反馈信息计算调整因子 *k*.

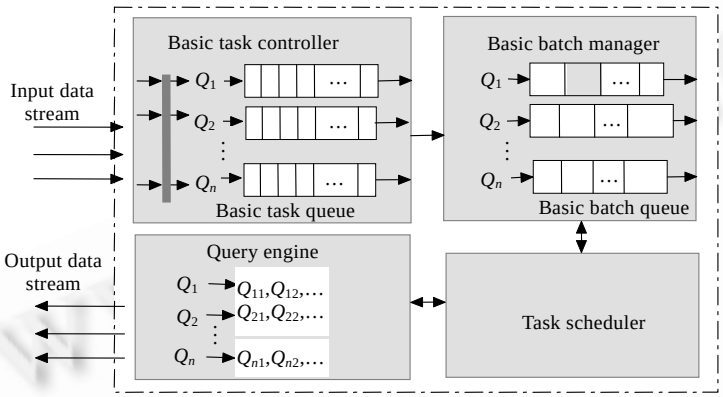

Fig.1 Architecture of RealStream system

图 1 RealStream 系统结构

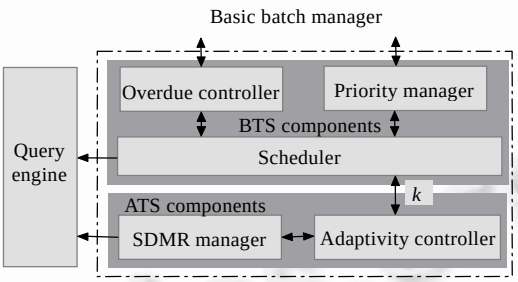

Fig.2 Architecture of the ATS scheduler

图 2 ATS 调度器系统结构

3 ATS 调度

本节首先介绍一种调度批长度固定不变的调度算法——BTS,然后进一步提出一种调度批长度可以自适应动态改变的调度算法——ATS.

3.1 BTS调度策略

BTS 调度算法是指每次调度时调度批长度固定不变,是一个常量.这个值可以根据系统的负载情况在开始工作时确定下来,即将调整因子 *k* 设定为一个固定值.在低载情况下,调度批长度可以大些;而在超载情况下,调度批长度可以小些.算法描述如下:

Algorithm 1. BTS.

1. Get adjusting factor *k*;
2. Repeat Step 3 to Step 5;

3. Call Priority Manager to compute the position i of the highest priority batch;

4. Select all basic tasks from k basic batches in the ith batch queue BQ_i to build a scheduling batch T_i;

5. Send T_i to Query Engine to execute.

BTS 调度策略是一种传统的批调度策略,相比 task-at-a-time(即调度单位为一个基本任务,简称 TAAT)策略,BTS 调度策略可极大地降低数据流上查询处理的系统开销,从而最终降低 *SDMR*.

3.2 ATS调度算法

在时变、突发性数据流环境中,数据到达速率很不稳定,时快时慢.为了进一步改善系统性能,调度批长度应该能够随着流速的改变而适应性地发生改变,即流速慢,则增大调度批长度以减少调度开销;流速快,则减小调度批长度以减少过期处理开销.调度批长度过大或者过小都会增大 *SDMR* 值,降低系统性能.由于数据流的变化,每次都会有一个最佳的调度批长度.ATS 算法的目标就是通过调整 k 来使每次的调度批长度接近最佳值.

为了适应流速变化而不断改变 k 值,本文设计了闭环反馈控制机制,它主要依据 PID 自动控制理论.考虑到简单实用,该机制仅采用了正比例计算和积分计算,即 PI 控制.系统运行时,SDMR Manager 会周期性地反馈 $\Delta SDMR$ 值给 Adaptive Controller,然后,Adaptive Controller 根据$\Delta SDMR$ 值动态地改变 k.

PI 控制函数定义如下:

$$\Delta k=\lfloor K_P\times\Delta SDMR+K_I\times SDMR\rfloor \tag{3}$$

其中,$\Delta SDMR$ 是前后两次执行完查询操作后 SDMR 的差值,$SDMR$ 是系统目前总的 $SDMR$ 值,K_P 是正比例系数,K_I 是积分系数.

PID 参数的整定有很多方法,如 PID 归一参数整定法、根轨迹法等.本文采用试凑法,先确定 K_I,再确定 K_p,观察 K_I 和 K_p 对系统响应的作用.通过逐步试凑,在达到满意的输出响应时,即为合适的 K_I 和 K_p 值.

Adaptivity Controller 机制的执行步骤如下:

Algorithm 2. Adaptivity Controller.

1. $k=k-\lfloor K_P\times\Delta SDMR+K_I\times SDMR_i\rfloor$;

2. if $k\leq 0$ then $k=1$.

将 BTS 算法的第 1 步替换为用 Adaptivity Controller 改变 k 值,即为 ATS 算法.

对于 k 的初始值,当输入负载较大时,可取值小些;当输入负载较小时,可取值大些.但由于本文算法具有自适应功能,能随负载变化自动调节 k,初值的选择对系统性能影响不大.考虑到输入负载的大小是随机的,因此,本文选择 k 的初始值为 1.

下面讨论算法中 k 的收敛性.根据后面第 4.2 节的实验可知,K_P 和 K_I 均为正数.k 的变化取决于数据流速变化情况,可分为两种情况:

(1) 当数据流速稳定时,即负载大小稳定.如果系统能够有效处理完所有数据,则$\Delta SDMR$ 为 0,$SDMR$ 趋于 0.因此,k 将收敛于一个常数;如果系统不能处理完所有数据,则$\Delta SDMR$ 趋于 0,$SDMR$ 趋于常数.因此,k 将收敛于一个常数.如果 $SDMR$ 较大,k 将收敛于 1.

(2) 数据流速有变化,即负载发生变化.如果数据流速变快,导致负载增加,则$\Delta SDMR\geq 0$,$SDMR\geq 0$,因此,k 不变或者减小到一个常数.如果发生过载,则 k 将减小到 1.如果数据流速变慢,导致负载减小,则$\Delta SDMR\leq 0$,$SDMR\geq 0$,因此,k 可能保持不变、减小和增大.当负载仍较大时,k 会减小.只有当$\Delta SDMR$ 变化幅度足够大时,才能导致 k 增大.但当 k 增大到超过某一个值时,必将导致$\Delta SDMR\geq 0$,这时,k 不会继续增加.因此,k 有上界,不会发散.

由此可见,在所有情况下,k 都将收敛于一个使 $SDMR$ 最小化的值.所以,ATS 算法能够更好地适应时变、突发的数据流,也能够充分利用系统资源,降低 $SDMR$ 值,提高系统性能.

3.3 优化策略

为了减小过期处理开销,在截止期内不能处理的任务不应该被执行,应该在执行时或执行前给予删除.本文采用动态统计计算的方法对调度算法进行优化.

统计计算方法将提前删除可能过期的任务.对于一个查询 Q_i,统计函数 $S(Q_i)$用来估算在一个调度批内能在截止期内完成的任务数.统计函数 $S(Q_i)$是根据操作符个数、操作符平均选择度和平均调度开销等条件计算的.令$(O_1,O_2,\ldots,O_n)$表示组成一个查询序列的操作符节点,设 O_i 的选择度为θ_i,开销为 C_i,则该查询处理 n 个基本任务的执行开销为

$$C_{exec}=n\times(C_1+\theta_1\times C_2+\ldots+\theta_1\times\ldots\times\theta_{n-1}\times C_n) \tag{4}$$

设 C_{sche} 表示平均调度开销,则总开销为

$$C_{total}=C_{exec}+C_{sche} \tag{5}$$

如果用 dl 表示基本批的截止期,则下面的不等式成立:

$$C_{total}\leq(dl-CurrentTime) \tag{6}$$

由公式(4)~公式(6)可以得出在截止期内能够处理的任务数的近似值:

$$S(Q_i)=\lfloor(d-CurrentTime-C_{sche})/C'_{exec}\rfloor \tag{7}$$

其中, $C'_{exec}=C_1+\theta_1\times C_2+\ldots+\theta_1\times\ldots\times\theta_{n-1}\times C_n$

加入统计计算的 BTS 算法称为 BTS1 算法,主要步骤如下:

Algorithm 3. BTS1.

1. Get adjusting factor k;
2. Repeat Step 3 to Step 7;
3. Call Priority Manager to compute the position i of the highest priority batch;
4. Use $S(Q_i)$ to calculate how many tasks can be finished in k basic batch of the ith batch queue;
5. Drop the oldest tasks that are potentially overdue from k basic batch of the ith batch queue;
6. Select the rest tasks to build a scheduling batch T_i.
7. Send T_i to Query Engine to execute.

将 BTS1 算法的第 1 步替换为用 Adaptivity Controller 来调整 k,则得到 ATS1 算法——优化的 ATS 算法.

3.4 算法分析

对于所有的调度算法,相同的查询任务都需要使用相同数量的资源.不同调度算法之间的区别是它们的调度开销,包括队列管理、内存管理以及上下文切换不同.最主要的影响因素是在一段固定时间内任务切换的次数.所以,我们把它作为系统开销的一个衡量指标.采用 ATS 算法调度时,一个任务或者被查询处理,或是由于在截止期内不能被处理而被提前删除,从而可以得到下面的定理.

定理 **1**. 在时间段 T_1 内 ATS 算法系统开销的复杂度介于$(O(T_1/U),O(T_1/\Phi))$之间,这里,U 是系统中所有查询紧急时间的最大值,Φ是基本批长度.

引理 **1**. 数据流上连续任务的所有批调度算法,在时间段 T_1 内,它们的系统开销复杂度都不小于 $O(T_1/U)$.

根据引理 1,设 $K=U/\Phi$,则 $O(T_1/U)=O(T_1/\Phi/K)$.在采用 ATS 算法时,总会得到一个接近于 K 的调整因子 k,且 $k>1$,所以 ATS 调度是最佳的.

4 性能实验评价

在本节,我们设计了几个实验来比较在时变、突发数据流上不同调度策略的性能,并对实验结果进行了讨论.调度算法主要包括 TAAT 算法(一个基本任务就是一个调度单位)、BTS 系列(BTS,BTS1)、ATS 系列(ATS,ATS1)以及理论上性能最优的 Ideal 作为基准.

4.1 实验设置

实验中,使用 SDMR 来比较上述 6 种调度策略的有效性和高效性.本文查询生成采用文献[6]中的模拟方法,设置两个查询参数:*depth* 和 *fan*-out.其中,*depth* 表示查询树的层数,*fan-out* 表示操作符的子操作符个数.每个叶子操作符均与一个数据流相连.

实验的测试数据采用满足泊松分布,基于如下定理:

计数过程$\{N(t),t\geq 0\}$是泊松过程的充分必要条件为$\{X_n,n\geq 1\}$是独立且参数同为λ的指数分布.

换而言之,当数据到达的时间间隔满足指数分布时,数据到达事件就满足泊松分布.实验中,数据到达的时间间隔是满足指数分布的.指数分布的密度函数如公式(8)所示.

$$f_e(x)=\begin{cases}\lambda e^{-\lambda x}, & x\geq 0\\ 0, & \text{eles}\end{cases} \tag{8}$$

其中,x 表示流数据的到达时间间隔,$f_e(x)$是 x 发生的概率,λ是指数分布参数,λ越大,则数据到达的平均时间间隔越小,即流速越快,所以可以通过改变λ来改变数据的流速.

查询个数是 100(每个查询对应一个数据流,长度为 10k,均服从泊松分布),固定操作符选择度是 0.5,其余参数见表 1.

Table 1 Experiment parameters

表 1 实验参数

Parameter	Value
Query depth	1~3
Query fan-out	1~3
Operator selectivity	0.01~1
Operator cost	1~20μs
System overhead cost	20~80μs
Exponential distribution parameter λ	0.1~1
Query deadline	1.0~5.0
Batch unit Φ	0.1s

实验的硬件环境是兼容机:2.4GHz 的 Pentium 4 CPU,主存为 512MB,硬盘为 80GB.软件环境是:操作系统为 Windows XP,算法采用 C 语言编写.

4.2 性能评价

图 3 是实验中系统负载(数据到达速率)情况.本实验中,流数据到达时间间隔服从指数分布,λ值为 0.5.图 3 的纵轴为流数据到达速率(个/0.1 秒),横轴为时间(秒).从图中可以看出,流数据到达速率变化非常快,在 50~75 之间不断发生变化.本实验正是采用具有这种特性的数据流来模拟不可预测、突发的数据流环境.

首先考察在批粒度固定的情况下,k 是何值时将获得最小的 *SDMR*,实验结果如图 4 所示.实验设置了 5 种不同的批粒度大小,即 0.01s,0.02s,0.04s,0.05s 和 0.1s.从图中可以看出,当调度批长度为Φ时,并不总能获得最优的 *SDMR* 值,特别是对于较小的批粒度.例如,当批粒度为 0.02s 的时候,批长度为 $2\times\Phi$时获得最优的 *SDMR* 值;而当批粒度为 0.01s 的时候,批长度为 $5\times\Phi$时获得最优的 *SDMR* 值.在时变、突发性数据流环境中,查询的紧急时间一般会很短,这时,批长度会很小,从而调度批长度可能要比批粒度大很多才能获得最小的 *SDMR* 值.在不可预测的数据流环境中,批长度需要随着流特性的改变而不断地作出相应调整才能最大程度地降低 *SDMR*.而这正是在 ATS 调度策略中设计反馈控制机制的原因,该反馈控制机制可根据流特性的变化不断调整 k 值,从而使调度批长度每次都能接近最佳批长度.本实验表明,在批粒度大小为 0.1 时,*SDMR* 达到最小.所以,下面的实验中批粒度大小为 0.1.

其次,用试凑法确定正比例系数 K_P 和积分系数 K_I.先确定 K_P,K_P 将取 0.5,1,2,5,10,20 几个值.从图 5 可以明显看出,当 K_P 值为 1 时,系统获得最小的 *SDMR*.这是因为 K_P 值太小,ATS 调度器会很少修改 k 值,尽管很多情况下该值确实需要修改;而 K_P 值太大,ATS 调度器会很频繁地修改 k 值,尽管很多情况下该值不需要修改.这两种情况都将导致 *SDMR* 的增大,最终影响系统性能.由此,这里选择 K_P=1.然后确定 K_I,K_I 将取 1,5,10,15,20 几个值.从图 6 可以明显看出,当 K_I 值为 10 时,系统获得最小的 *SDMR*.因此,这里选择 K_I=10.由此可得,$\Delta k=\lfloor \Delta SDMR+10\times SDMR\rfloor$.需要说明的是,$K_P$ 和 K_I 值的选择是与数据流特性相关的,对于不同的负载分布情况,可能会选择不同的 K_P 和 K_I 值.

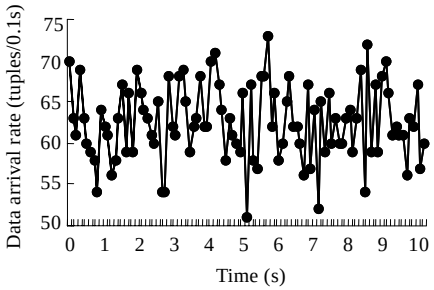

Fig.3 Arrival rate with Poisson distribution

图 3 泊松分布数据到达速率示意图

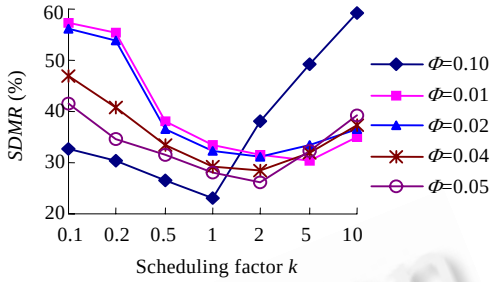

Fig.4 Effect of different batch sizes

图 4 批大小的影响

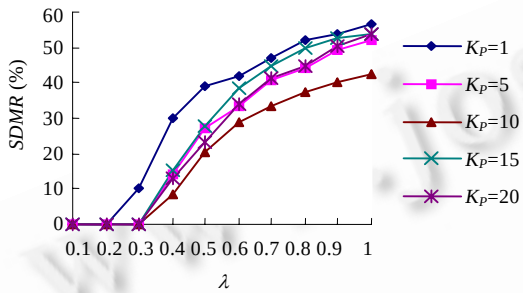

Fig.5 Effect of different proportional factor K_P

图 5 不同比例因子 K_P 的影响

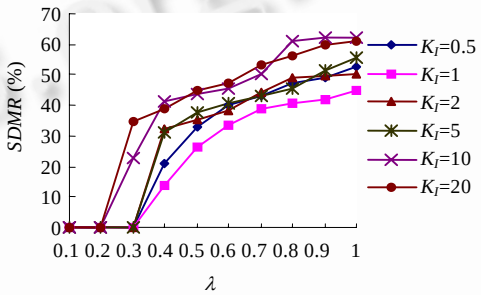

Fig.6 Effect of different integral factor K_I

图 6 不同积分因子 K_I 的影响

然后,考察在几种调度算法下系统各自的性能情况,主要考察系统开销、SDMR 和 CPU 使用率这 3 个性能指标.

图 7 给出系统开销情况.实验中操作符选择度被设为 1.从图中可以看出,TAAT 算法下系统开销明显大于其他算法;而 ATS 算法下的系统开销略大于 BTS 算法.这是因为其他算法都是基于批调度的,调度次数明显少于 TAAT;而 ATS 算法中由于反馈控制的引入,增加了运算开销,使系统开销略微变大.

另外,我们注意到,随着λ的增大,各算法下的系统开销有下降的趋势.这是因为不同λ下查询处理的流任务量相同,均为 10k 个.λ较小时,系统没有过载,有足够的资源处理所有的流任务;而随着λ的增大,流速变大,系统负载变大,当系统负载超过系统处理能力时,过期批任务数将会由 0 开始逐渐增多,这使得在截止期内处理的批任务数逐渐减少,总系统开销也随之降低.

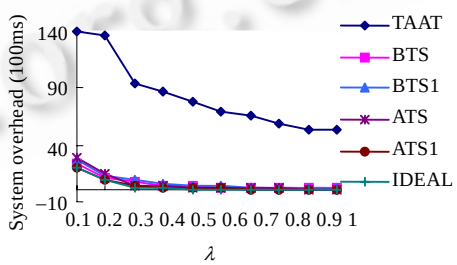

Fig.7 System overheads

图 7 系统开销的比较

接着分析 SDMR 情况,实验结果如图 8 所示.从图中可以看出,TAAT 算法下的 *SDMR* 值明显大于其他算法下的 *SDMR* 值;而 ATS 算法下的 SDMR 曲线又略低于 BTS 算法下的 SDMR 曲线.例如当λ=0.5 时,ATS 算法下的 *SDMR* 值比 TAAT 的小 30%,比 BTS 小 15%.这是由于 TAAT 算法下总调度开销明显大于其他两种算法,即很多系统资源白白浪费在调度开销上,使得 TAAT 算法在截止期内处理的流任务数明显少于其他两种算法.所以,TAAT 算法下的 *SDMR* 值明显增大.不同的是,尽管 BTS 算法的总调度开销略小于 ATS 算法,但 BTS 算法下的 *SDMR* 值却略大于 ATS 算法下的 *SDMR* 值.原因是,BTS 算法不能实现精确的批任务调度,调度批长度固定不变,而 ATS 算法能够适应流特性变化,动态地调整调度批长度.另外,由图中可以看出,统计计算对 ATS 的优化效果大于对 BTS 的,ATS1 下的 *SDMR* 值明显小于 ATS 下的 *SDMR* 值,而 BTS 和 BTS1 下的 *SDMR* 值却相差不多,原因可能是 ATS

算法会调整 k 的大小,有时调度批会很大,任务过期的概率也随之增加,统计计算能够有效地删除过期元组,极大地减少了过期处理开销;而 BTS 算法的调度批长度固定不变,一般取为 Φ,调度批长度很小.

然后,分析平均 CPU 使用率情况,图 9 给出了图 8 中实验下对应的平均 CPU 使用率.平均 CPU 使用率代表了不同调度策略下查询引擎处理相同的数据集所需要的系统资源量.从图中可以明显看出,相比理想情况下的平均 CPU 使用率,由于系统开销太大,TAAT 算法下的平均 CPU 使用率曲线总是高于其他两种算法下的平均 CPU 使用率;而 ATS 算法下的平均 CPU 使用率略高于 BTS 算法下的平均 CPU 使用率.结合上面讨论,这是因为 ATS 算法比 BTS 算法的复杂度要大.

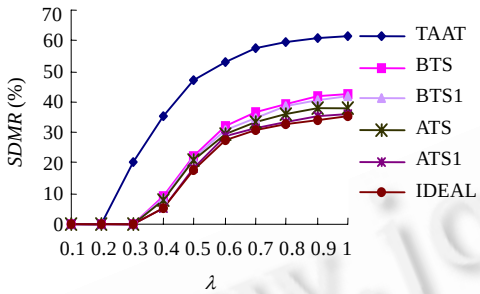

Fig.8 SDMR under fixed selectivity

图 8 SDMR 的比较(固定选择度)

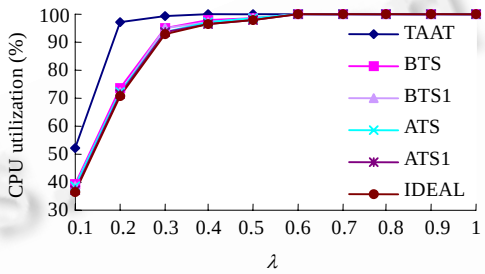

Fig.9 CPU utilization under fixed selectivity

图 9 CPU 使用率的比较(固定选择度)

接下来,将操作符的固定选择度 0.5 改为随机选择度,范围在 0.01 与 1 之间,然后重复上面的两个实验.

图 10 是随机选择度下 SDMR 的情况,从中可以看出,ATS 和 BTS 算法下 *SDMR* 值的差距变大,但 ATS 算法仍然最接近理想情况.原因是 ATS 是精确的批调度.例如,在 λ=0.4 时,ATS 算法下的 *SDMR* 值是 5%,而 BTS 算法和 TAAT 算法下的 *SDMR* 值分别是 10%和 40%;在 λ=0.8 时,ATS 下的 *SDMR* 比 TAAT 小 30%,比 BTS 小 10%.另外,我们注意到,图 10 中的 SDMR 曲线要相应地高于图 8 中的曲线.这是因为如上面所述,先前做的实验是固定选择度,大小为 0.5,后做的实验是随机选择度,平均值为 0.55,相对而言,后面的实验要比前面的实验更为复杂.图 11 表示了随机选择度下采用不同调度算法的平均 CPU 使用率的情况.很容易看出,图 11 中平均 CPU 使用率特征与图 9 类似,这里不再详说.

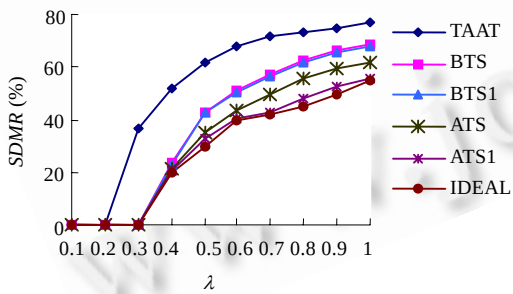

Fig.10 SDMR under random selectivity

图 10 SDMR 的比较(随机选择度)

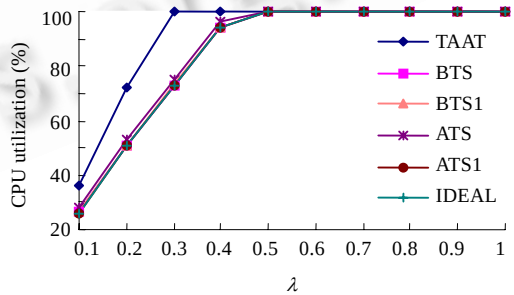

Fig.11 CPU utilization under random selectivity

图 11 CPU 使用率的比较(随机选择度)

在最后两个实验中,我们考虑系统负载增加时(即将查询个数和数据流的个数均增大到 300)系统各性能指标的情况.

图 12 是固定选择度下 SDMR 的情况.从中可以看出,每种算法下的 *SDMR* 值均从一个非 0 值开始增加,因为系统中的查询数增多,导致系统一开始就过载,不能在截止期内处理完所有的任务.但 ATS 算法下的曲线仍然最接近 Ideal 的,而 TAAT 的 *SDMR* 值仍然最大.ATS 下的 *SDMR* 值比 TAAT 下的小 15%~20%,比 BTS 下的小 5%~15%.图 13 是随机选择度下 SDMR 的情况.曲线特征与图 12 的类似,只是由于查询复杂度增加,*SDMR* 值

略大.

至于平均 CPU 使用率情况,由于一开始系统就过载,所以它始终保持在 100%,曲线略去.

另外,我们还对满足另一种典型分布(Zipf 分布)的流数据做了相同的实验,结果大致相同,这里予以省略.

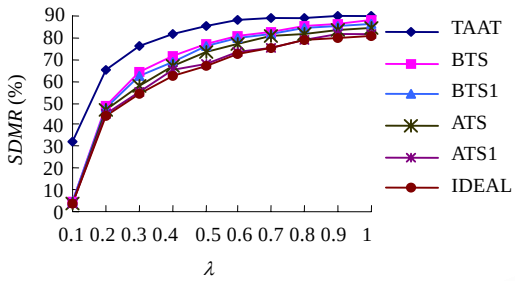

Fig.12 SDMR under fixed selectivity with 300 queries

图 12 固定选择度下 SDMR 的比较(300 个查询)

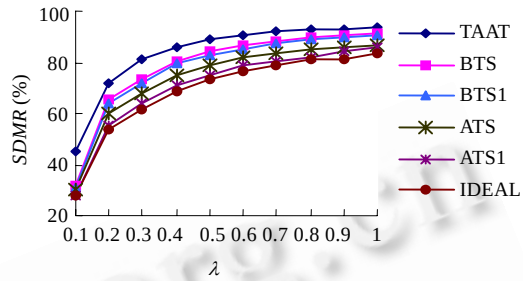

Fig.13 SDMR under random selectivity with 300 queries

图 13 随机选择度下 SDMR 的比较(300 个查询)

5 相关工作

当前的数据流管理系统[1−6]设计了各种高效的数据流处理技术.然而,它们大都关心最小化平均元组延时,而不能完全满足于实时应用的需求.RealStream 系统正是被设计用于处理固实时应用,而调度策略是构成 RealStream 系统中最重要的技术之一.

实时数据库中的调度策略[13−16,21,22]力争使每个到达任务都能在截止期内完成,它们的目标是最小化截止期错失率 DMR.这些策略均建立在 task-at-a-time 的基础之上.而 ATS 调度策略是一种精确的批调度策略,用于处理时变、突发数据流上大量的流任务.RealStream 系统中定义的 SDMR 类似于文献[21]中的 DMR,但 SDMR 具有时间参数 I 而且具有数据流上的时变适应性.

实时数据库中 RM(rate monotonic)[13],EDF[13],MUF[14]等调度策略可用于处理周期性任务,且假设每个周期中的任务具有相同的负载.但由于数据流的时变和突发性,ATS 算法所处理的批任务并不具有相同的周期,且每个周期中的负载也不尽相同.

另外,文献[7−11,18,19]中也考虑了数据流上的突发性和实时性需求.Carney 等人[7]提出二级调度策略,对操作符以及操作块分别进行批处理,进而降低系统开销,提高处理效率.Babcock 等人[8]讨论在系统负载高峰时,如何优化资源管理以实现运行时内存使用的最小化.Jiang 等人[9]在 Chain 式调度方法的基础上,针对高实时性的突发流给出 PCS(path capacity strategy)调度策略,同时实现了流数据平均处理延时的最小化.另外,Sharaf 等人[18]设计了一种基于速率的抢占式调度策略,用于处理到达流数据的异步性以及查询计划的异构性.Eddy 机制[11]中提出 tuple-at-a-time 调度策略,通过对每个元组动态选择路由,从而不断动态地优化查询计划中的操作符顺序,具有很强的适应性.所有上述调度策略均用于处理平均元组延时,而不能完全满足数据流上的固实时需求.而且它们均为操作符调度策略,用于解决复杂的数据处理.而 ATS 是一种基于任务的调度策略,用于最小化 SDMR.

Aurora 系统对应用提出 QoS 规范要求[1],这类似于固实时系统中的性能指标.不同的是,Aurora 系统中的 QoS 规范只关心被查询处理的一批元组的平均元组延时,而 ATS 调度策略用于解决被查询处理的每个元组的截止期.

基于统计计算的 Load shedding 技术讨论了如何在突发数据流上进行降载以满足应用的实时需求[19,20].而 ATS 调度策略采用精确的批调度策略,且仅丢弃不能在截止期内完成的流任务.

FC-EDF(feedback control-earliest deadline first)[23]通过在 EDF 算法中加入 PID 控制器,设计了一种用于解决实时应用的反馈型实时控制调度算法.Tu 等人[24]提出一种反馈控制型任务调度策略,该策略通过在不可预

测、突发的数据流上动态调整负载以实现动态降载和自适应 QoS.类似于文献[23,24],ATS 调度策略也引入反馈型闭环控制策略以降低 *SDMR*.同时,ATS 也是一种批处理策略,极大地降低了系统开销.在实现机制、具体采样周期以及控制函数上,ATS 调度策略与上述调度策略不同.

6 结 论

本文以截止期为核心研究了数据流上的调度问题,目标在于最小化流截止期错失率——SDMR.本文的主要贡献是设计并实现了一种适应性实时任务调度策略.该策略解决了如何在稳定的数据流负载上给定最优批大小,以及如何在动态变化的数据流负载上适应性调整最优批大小.

为了满足流任务的严格截止期要求,并且降低系统开销和过期处理开销,本文提出了一种精确的批调度策略——基于时间的批调度.而且,为了适应关键任务型数据流上时变突发的数据流特性,本文提出了一种适应性批任务策略——动态批任务机制.通过合并这两种策略,本文最终设计并实现了一种适应性、精确的基于批的任务调度策略,称为 ATS 策略.该策略可以很好地适应数据流上的时变性、突发性,并通过大幅度降低系统开销,进一步提高了系统效率,极大地增加了系统中流任务的吞吐量.实验表明,ATS 策略性能优越于其他调度策略,而且接近于理想情况.

在下一步工作中,我们将进一步研究设计分布数据流管理系统中的截止期调度策略——ATS*,并为分布式实时数据流应用设计一种高性能调度机制.

References:

[1] 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. Journal of VLDB, 2003,12(2):120−139.

[2] 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 (CIDR). Asilomar: Morgan Kaufman Publishers, 2003. 245−256.

[3] 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.

[4] Chen J, DeWitt D, Tian F, Wang Y. NiagaraCQ: A scalable continuous query system for Internet databases. In: Franklin MJ, ed. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2000.

[5] Sullivan M, Heybey A. Tribeca: A system for managing large databases of network traffic. In: Proc. of the USENIX Annual Technical Conf. New Orleans: USENIX, 1998.

[6] 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.

[7] Carney D, Centintemel 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.

[8] 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. San Diego: ACM Press, 2003. 253−264.

[9] Jiang Q, Sharma C. Scheduling strategies for processing continuous queries over streams. In: Williams MH, MacKinnon LM, eds. Proc. of 21st British National Conf. on Databases. Springer-Verlag, 2004. 16−30.

[10] Babcock B, Babu S, Datar M, Motwani R, Thomas D. Operator scheduling in data stream systems. The VLDB Journal, 2004,12(4): 333−353.

[11] Avnur R, Hellerstein J. Eddies: Continuously adaptive query processing. In: Franklin MJ, ed. Proc. of the SIGMOD Conf. Dallas: ACM Press, 2000. 261−272.

[12] Madden S, Shah M, Hellerstein J, 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. ACM Press, 2002. 49−60.

[13] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in hard real-time environment. Journal of the ACM, 1973, 20(1):46−61.

[14] Stewart DB. Real-Time software design and analysis of reconfigurable multi-sensor based systems [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 1994. 113−126.

[15] Duda K, Cheriton D. Borrowed-Virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. In: Proc. of the 17th ACM Symp. on Operating Systems Principles. ACM Press, 1999.

[16] Jones MB, Rosu D, Rosu MC. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In: Proc. of the 16th ACM Symp. on Operating Systems Principles. ACM Press, 1997. 198−211.

[17] Babcock B, Babu S, Datar M, Motwani R, Widom J. Models and issues in data stream systems. In: Proc. of the 21st ACM Symp. on Principles of Database Systems. Madison: ACM Press, 2002. 1−16.

[18] Sharaf MA, Chrysanthis PK, Labrinidis A. Preemptive rate-based operator scheduling in a data stream management system. In: Proc. of the 3rd Int'l Conf. on Computer Systems and Applications. 2005. http://engr.smu.edu/cse/AICCSA-05

[19] Tatbul N, Cetintemel U, Zdonik S, Cherniack M, Stonebraker M. Load shedding in a data stream manager. In: Freytag JC, Lockemann PC, *et al*., eds. Proc. of the 2003 Int'l Conf. on Very Large Data Bases. San Fransisco: Morgan Kaufmann Publishers, 2003. 309−320.

[20] Babcock B, Datar M, Motwani R. Load shedding for aggregation queries over data streams. In: Proc. of the 2004 Int'l Conf. on Data Engineering. IEEE Computer Society, 2004. 350−361.

[21] Kang KD, Son SH, Stankovic JA. Managing deadline miss ratio and sensor data freshness in real-time databases. IEEE Trans. on Knowledge and Data Engineering, 2004,16(10):1200−1216.

[22] Lee J, Tiao A, Yen J. A fuzzy rule-based approach to real-time scheduling. In: Proc. of the 3rd IEEE Int'l Conf. on Fuzzy Systems. IEEE Computer Society, 1994. 1905−1910.

[23] Lu CY, Stankovic JA, Tao G, Son SH. Feedback control real-time scheduling: Framework, modeling, and algorithms. Journal of Real-Time Systems, 2001,23(1/2):85−126.

[24] Tu YC, Liu S, Prabhakar S, Yao B. Control-Based quality adaptation in data stream management systems. In: Andersen KV, Debenham JK, Wagner R, eds. Proc. of the 16th Int'l Conf., DEXA 2005. Copenhagen: Springer-Verlag, 2005. 746−755.

于戈(1962－),男,辽宁大连人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库系统理论与技术.

李晓静(1983－),女,硕士生,主要研究领域为数据库系统,数据流管理.

杨晓春(1973－),女,博士,副教授,CCF 高级会员,主要研究领域为数据库系统,传感器网络,数据流系统.

欧征宇(1979－),男,硕士,主要研究领域为数据库系统,数据流管理,实时系统.

邓庆绪(1970－),男,博士,副教授,CCF 高级会员,主要研究领域为实时系统,嵌入式软件.