东南大学

硕士学位论文

数据流连续查询处理系统设计与实现

姓名：王浩

申请学位级别：硕士

专业：计算机应用技术

指导教师：徐宏炳

20060301

摘要

近年来，在金融服务、网络监控、电信数据管理及传感器监控等领域中，出现了一类新的数据密集型应用，这类应用的特征是：数据不宜用持久稳定关系建模，而适合用瞬态数据流建模。在这种数据流模型中，单独的数据单元可能是相关数据的元组，例如网络测量、呼叫记录及传感器读数等产生的数据。由于这些数据以大量、快速、时变的数据流形式持续到达，若把持续到达的流数据简单的放到传统的数据库管理系统中进行管理是不可行的，由此产生了一些新的基础性的研究课题。

论文在研究目前国际上最新的数据流管理技术的基础上，介绍了自行研究开发的基于硬件预处理器的数据流管理系统原型SEUSTREAM的体系结构，设计了一种可以支持对数据流上进行连续查询的查询语言X-SQL，并以数据流预处理器为目标机器设计了数据流连续查询语言X-SQL的编译器，该编译器对用户提交的查询命令进行词法、语法及语义分析，并进行查询优化，得到查询抽象语法树，并针对数据流预处理器对抽象语法树进行处理，生成其所需的控制信息。

关键字：数据流管理系统，连续查询，连续查询语言，编译器

Abstract

Recently there has witnessed a focus on processing new data-intensive applications, such as financial service, network monitoring, telecommunications data management, sensor detection, and so forth. The common feature of these applications is that, the data generated in these applications are more suitable for being modeled by transient data streams than by permanent relations. In the model of data streams, a data unit could be a tuple of related data, such as network measurement, call records, sensor reading, and so on. For the sake of the characteristic of huge size, high speed of data arriving, time-varing, the data in data stream model could not be managed by traditional database.

On the basis of studying the new theory and technology of data stream management, a new data stream management system prototype based on preprocessing the data streams by hardware, which named SEUSTREAM, is developed. Firstly the architecture of SEUSTREAM and the preprocessor of data streams are introduced. Then the paper gives the design of a continuous query language X-SQL, which support the continuous query on data streams. And then a compiler of X-SQL is introduced. Its target machine is the data stream preprocessor in SEUSTREAM. In SEUSTREAM, users submmit continuous queries of data streams by X-SQL language. After lexical analysis, sytax parsing, semantic analysis and query optimization, the queries commited by users would be transferred into an abstract syntax tree (AST) by the compiler. And then the compiler would generate machine code of the preprocessor.

Key words DSMS, Continuous Query, Continuous Query Language, Compiler

东南大学学位论文
独创性声明及使用授权的说明

一、学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：刘洁 日期：2006.3.30

二、关于学位论文使用授权的说明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布(包括刊登)论文的全部或部分内容。论文的公布(包括刊登)授权东南大学研究生院办理。

签 名：刘洁 导师签名：[illegible] 日 期：2006.3.30

绪论

在形形色色的计算机应用领域中，有一类重要的计算机应用，其所涉及数据量非常之大，其中数据也不随程序的结束而消失并被多个应用程序所共享，我们称之为数据密集型应用。这类应用所要处理的问题即为数据管理问题。数据管理一直是计算机科学技术领域中的一门重要技术和研究课题，由此开始了数据库系统的研究。数据库系统的研究和开发在其30年的历史中取得了巨大的成功，数据库系统的应用已经遍及各个领域，奠定了数据库系统作为当今社会信息基础设施核心的技术地位。

近年来，出现了一类新的数据密集型应用，这类应用中的数据特征是：连续、无限、快速、随时间变，数据不宜用持久稳定关系建模，而适合用数据流建模。这类应用出现在许多应用领域中，包括金融服务，网络监控，安全，电信数据管理，Web应用，生产制造，传感检测等等。在这种数据流模型中，单独的数据单元可能是相关的元组，例如网络测量，呼叫记录，网页访问，传感读数等产生的数据。但是，由于这些数据以大量、快速、随时间改变的数据流形式持续到达，由此产生了一些基础性的新的研究问题。

在数据流出现的应用中，如果把流数据存放到传统的数据库管理系统(DBMS)中，用DBMS对数据流进行管理会带来很多问题。关键的问题在于数据流无限性的本质和传统的DBMS存储容量的有限性的矛盾，数据流的持续性、变化性和传统DBMS的静态性、持久性的矛盾。连续查询(continuous queries)是数据流应用的典型特征，但传统DBMS并不支持连续查询。近似性(approximation)和自适应性(adaptivity)是对数据流进行快速查询、数据分析和数据采集等操作的关键要素，而传统DBMS的主要目标是通过稳定的查询设计，得到精确的答案。

对数据流处理的研究是非常活跃的领域，目前国外已经在这方面开展了大量的研究工作，取得了一定的成果，并且开发出了一些数据流管理系统原型。这几年来，著名的国际会议 VLDB 和 ACM SIGMOD 等都将数据流作为一个重点问题来讨论，数据流管理技术正成为国际数据处理领域的一个热点问题。研究数据流管理技术不仅有重要的理论意义，而且有着巨大的应用前景，已经引起了越来越多的学者的重视。我们现在正在研究基于硬件预处理器的数据流管理原型系统——SEUSTREAM，通过基于硬件预处理器的体系架构，该系统能够处理多条高速数据流上的多用户并发查询。

第一章 数据流管理技术概述

1.1 数据流的出现

近年来，出现了一类新的数据密集型应用，这类应用中的数据特征是：大量、连续、无限、快速、随时间改变。这类数据持续到达，数据达到速度及数据量还有可能是不可预知的，我们把这种数据称为数据流[1]。

这类应用出现在许多应用领域中，包括传感器网络[10]、交通管理[3]、网络监控[4]、电信数据管理[6]、股票分析[5]、生产制造过程控制等。在这种数据流模型中，单独的数据单元可能是相关的元组，例如网络测量，呼叫记录，网页访问，传感读数等产生的数据。由于这些数据以大量、快速、随时间改变的数据流形式持续到达，以一种流的形式源源不断地进入系统，无法以永久的关系形式全部保存下来之后再行处理。因此，许多已有的数据库技术难以应用于数据流，由此产生了一些基础性的新的研究问题，有关数据流数据处理技术的研究越来越引起人们的关注。

1.1.1 数据流特征

电话记录，股票报价以及从传感器读数都是数据流的例子。数据流是一个实时的、连续的、潜在无界的、有序的(隐含的通过到达时间或者明确的时间戳)项的序列。

与传统关系数据库中的数据相比，数据流数据具有许多自己的特点：

- 数据实时到达，数据流速度有可能不可预知，要求快速即时的响应；
- 系统无法控制数据元素的到达次序；
- 数据随时间变化，规模宏大且无界（即不能预知其最大值）；
- 数据基本上采用的是单遍扫描算法，数据一经处理很难再次被取出；
- 数据结构化程度较低，需要多层次化和多维化处理。

1.1.2 数据流模型

在数据流模型中，数据实时到达，而不是像传统数据库中数据存储在可随机访问的磁盘或内存中，它们以一个或多个“连续数据流”（continuous data steams）的形式到达。令 t 表示任一时间戳，a_t 表示在该时间戳到达的数据，流数据可以表示成$\{\cdots,a_{t-1},a_t,a_{t+1},\cdots\}$。

实时数据流是按照某一顺序的数据项序列，可能只能被扫描一次。由于数据项可能猝发式到达，数据流可以模拟为元素列表序列，每个流项目以关系元组或对象实例的形式存在，(例如 STREAM [15,16])，项是存储在虚关系中的瞬时元组，可能横越远程节点并能被水平分割；在基于对象(object-based)的模式(Tribeca[12])中数据源和项目类型按照分层相关的方法模拟。

在多数情况下，流的操作只有在给定的范围内是有意义的，这种窗口模式按照下列三个标准分类：

1. 端点移动的方向：两个固定的端点定义一个固定的窗口；两个滑动端点（当新数据项到达时替换掉旧的数据项）定义一个滑动窗口；一个固定的端点和一个移动端点（向前或向后）定义一个界标窗口。

2. 物理或是逻辑：按照抵达时间定义物理时间窗，按照元组的数量定义逻辑（也称基于计数的）窗口。

3. 更新时间间隔：在每一个新元组到达时重新评价窗口的更新，批处理导致了一个跳跃的窗口。如果更新的时间间隔大于窗口尺寸，结果是一系列非重叠的滚动窗口。

1.1.3 数据流查询

数据流上的查询[1]与传统数据库中的查询类型基本上相似，但是有一个主要区别，数据库中的查询主要是一次查询，数据流上的查询则更多的是连续查询。一次查询是指查询提交后，系统根据当前的数据集合的快照给出查询的结果。连续查询是指查询随着新数据的到来而不断地返回查询结果。与一次查询相比，连续查询的特点是长期运行。在数据流管理系统中，连续查询被注册后，有两种方式终结：一是注册的查询中明确的指出了查询持续的时间，二是用户明确地提出撤销该查询的请求，否则随着数据流新的数据元素的到达，该查询将源源不断地返回查询结果。

数据流上的查询可以分为预定义查询和即席（Ad Hoc）查询两种查询形式。预定义查询是一类在任何相关数据到来之前在数据流管理系统已经定义的查询。预定义查询通常是连续查询。预定义查询注册到系统后，主要针对数据流后续到来的数据计算查询结果；而即席查询是针对数据流中流过的数据进行查询，即对历史数据进行查询。由于系统无法保存数据流流过的所有数据，系统只能通过提取和组织流过数据的概要信息来支持即席查询，因此即席查询一般只能得到近似的查询结果。

1.2 传统 DBMS 处理数据流遇到的问题

传统的数据库系统旨在处理永久、稳定的数据,强调维护数据的完整性、一致性，其性能目标是高的系统吞吐量和低的代价，其设计目标是维护数据的绝对正确性、保证系统的低代价、提供友好的用户接口。这种数据库系统对传统的商务和事务型应用是有效的、成功的，然而它不适合无限、快速、实时的数据流应用。

在数据流应用中，如果把连续到达的流数据存储到传统的数据库管理系统(DBMS)中，用 DBMS 对传统数据操作的方式对数据流数据进行操作是非常困难的。传统的 DBMS 并不是为快速连续的存放单独的数据单元而设计的，而且也并不支持“连续查询”（continuous queries），而“连续查询”是数据流应用的典型特征。另外，“近似性”（approximation）和“自适应性”（adaptivity）是对数据流进行快速处理的关键要素。

数据流数据的特征要求新的数据处理技术，下面我们开始介绍当前正在研究的一些数据流处理的技术。

1.3 数据流管理技术研究现状

目前，国外已经在数据流管理领域开展了大量的研究工作，取得了一定的成果，并且开发出了一些数据流管理系统原型。最具典型的原型系统主要有 stanford 大学的 Stanford Stream Data Manager(简称 STREAM)、Berkeley 大学的 Telegraph 系统和 Brandeis University、Brown University 与 M.I.T.联合开发的 Aurora 系统。其它数据流系统主要有 Tapestry、Tribeca、Niagara、COUGAR、ATLaS、Fjords、Hancock 等[6-14]。

1.3.1 STREAM 系统

STREAM 系统是由 Stanford 大学开发的一个通用的数据流管理系统[15,16]。它的设计目标是处理高速数据流，支持大量并发的连续查询。在数据流速和查询负载超出可获得资源的情况下，系统优化内部配置，可以为连续查询提供相对准确的近似结果。系统内部的多查询优化策略、有效的资源分配算法和灵活的调度策略保证了系统的高性能。有限资源和结果近似性的自动平衡是系统最重要的关键技术。

STREAM 系统支持查询语言 CQL (Continuous Query Language)[15]，用户可以采用 CQL 注册数据流查询，也可以直接输入查询计划。STREAM 可按照用户的查询需求针对数据流进行实时的连续

查询。用户或应用程序向系统注册查询，被注册的查询将一直被保留到查询被显式地注销为止。

STREAM 系统中，查询首先被注册，随着新数据的到来，查询不停地被执行。CQL 是 STREAM 系统标准查询语言，它既支持传统的关系操作，又支持流操作。CQL 扩展了 SQL 语言，主要表现在 From 子句。From 子句包含一个可选的滑动窗口和一个取样子句。滑动窗口由分割子句（Partition by）、窗口大小和可选的过虑谓词组成。分割子句把数据分成几组，在窗口内计算每组的值，然后合并结果，类似标准 SQL 中的 Group by。窗口大小由 row 或 range 确定，如：range 10 minutes preceding。取样子句定义了取样的百分比，如 sample（3）。

对于无限的数据流，STREAM 系统通过滑动窗口将其转化为有限的关系元组集合。系统又定义了一些新的运算符，如 Istream（插入流）和 Dstream（删除流），通过它们将关系元组转化为流，为用户提供连续的查询结果。

STREAM 系统中，每一个查询都有一个独立的查询计划。查询计划由三种不同的结构组成：查询运算符（query operators）、操作队列（inter-operator queues）、大纲（synopses）。一般来说，队列是内存中用来存储数据的一块缓冲区，由最大容量限制，因此当数据量超出缓冲区的最大容量时，旧的数据就被丢弃，以便容纳新数据。队列可以为多个运算符提供数据源。在传统的关系数据库中，两个关系的连接操作需要多次的扫描关系表，而数据流具有瞬时性，数据不停地被更新，为了实现上述功能，STREAM 系统定义了大纲数据结构，用于暂时存储数据。当然，大纲不仅仅用于连接操作。大纲与队列的重要区别是：大纲是为某种操作而服务的，容量大小通常根据操作的需要而决定。

有效的资源管理是数据流管理系统的关键技术之一。有限资源和结果近似性的自动平衡是 STREAM 系统最重要的关键技术。STREAM 系统中，重点关注的是内存的管理。针对内存管理主要采用了两种技术：根据统计信息和查询注册信息，对大纲进行压缩；采用优化的调度策略，降低队列所占用的存储空间。在数据流的流速和查询负载超出可以获得资源的情况下，可以为连续的查询提供相对准确的近似结果。那么如何在有限的资源下获得最佳的查询结果呢？STREAM 系统采用的主要技术是静态近似策略和动态近似策略。其中相较而言，动态近似策略是更有挑战性的课题，它比静态近似具有更多的优越性。STREAM 系统最主要的资源消耗在于大纲和队列。因此主要的动态近似方法有：采用柱状图、小波压缩等技术实现大纲的压缩；采用动态取样、数据流丢弃等技术减少队列所占用存储空间。

STREAM 系统还有不少不完善的地方。它是一个基于关系模型的集中式数据库，而对于许多数据流的应用，分布式处理更为重要。因此 Stanford 大学正在计划 STREAM 系统升级为分布式系统。查询计划的形成策略还有待于进一步研究，资源分配算法还不能很好的支持近似回答。

1.3.2 Aurora 系统

Aurora 是由 Brandeis University、Brown University 和 M.I.T 联合开发的数据流管理系统，它支持大量的触发器，因此又被称为触发器网络[17,18,19]。

Aurora 系统是一个面向数据流监测应用的数据流管理系统。采用了工作流系统中常用的 box (功能盒操作符) & arrow(数据流向符) 模型。用户根据查询需要，将查询表示为由 arrow 流向符连接起来的若干 box 操作符组合而成的网络模式，这个模式就是初步查询计划。用户注册一个查询，就是要生成一个相应的查询计划。查询计划被简单变换成调度器中的不同查询处理队列。调度器的调度分两个层次：

(1) 确定选择哪个查询进行调度；

(2) 确定每个查询如何执行，即操作符序列的执行顺序。

Aurora 系统采用了批处理技术来降低调度代价和操作符执行代价。系统支持两种方式的批处理：对操作符调度的批处理和对元组的批处理。Aurora 系统对于大量的实时数据，根据定义的服务质量(QoS) 模型，采用 Loadshedding 技术，必要时，修改查询计划，卸掉一定的过量负载，以保证系统的响应速度。

Aurora 系统中最基本的结构是 box 和 arrow，每个 box 是一个处理单元，arrow 反映了数据的流

动方向和处理单元处理数据的先后次序。查询是通过图形化的用户界面，以 box 和 arrow 形式。这是 Aurora 与其它数据流管理系统的显著不同。

Aurora 系统定义了专用的运算符，以完成各种查询需要，这主要包括 Filter、Drop，Union、Wsort、Tumble、Windowed、Slide、Latch、Resample、Map、Groupby 和 Join 等。其中，Filter 对输入流过虑；Union 将多个输入流合并为一个输出流；Wsort 缓冲所有的输入流，并按属性排序输出；GroupBy 按照属性进行分组；Join 是连接操作；Windowed 截取一段数据流，即固定时间窗口；Slide，滑动窗口；Resample，是再取样；Map，映射；Tumble 和 Latch 运算符比较复杂，具体参加文献[18]。

Aurora 系统支持三种查询模式：连续查询（实时处理）、视图和 ad 查询。第一层结构为连续查询，该层的数据被实时处理后，不保存。服务质量规范（QoS specification）决定资源如何分配给各个处理单元。通常，一个用户查询被分解为若干个 box，box 由连接点出增加或删除，连接点可以将数据暂存一段时间。第二层结构是视图，是在调度器的控制下物化或者部分物化的视图。通过视图可以加快查询的速度，服务质量规范说明各个视图的重要程度。第三层结构为 ad 查询，ad 查询可以在任意时刻附加到连接点，实现对历史数据的查询。

归纳起来，Aurora 系统具有如下特点：1）DAHP 模式（DBMS-Active， Human-Passive）；2）适宜处理时间序列信息，便于获取数据的新值和旧值；3）触发器居于主导地位，能够有效实现大量的触发器；4）能够根据负荷和资源的情况，实现精确查询或近似查询；5）具有实时性，可以在线处理数据。因此 Aurora 系统的主要应用环境是：处理的对象是数据流；需要大量的触发器；有较高的实时性要求；以非精确的数据处理为主或近似查询为主。

对许多的数据流应用，其本质上是分布式的。因此开发分布式的数据流管理系统是必要的。Aruora*是以 Aurora 为基础的分布式数据流管理系统。具体见文献[18]。

1.3.3 TelegraphCQ 系统

TelegraphCQ[20,21]是一个通用的数据流管理系统，在开放式关系数据库管理系统 PostgreSQL 基础上开发。它继承了 UC Berkeley 的 Telegraph 数据流项目开发成果，以 Psoup 系统为查询处理系统，以 Flux 系统作为负载平衡和容错处理系统。在系统中，注册的数据流查询经过预处理后被变换成一个操作符执行序列，而后交给元组路由选择器 Eddy。

Psoup 是 Berkeley 大学的 Telegraph 系统的查询处理器。在 Psoup 中查询首先要被注册，注册后返回用户一个句柄。接下来不停地执行查询请求。查询语句的一般模式是：SELECT select-list FROM from-list WHERE conjoined Boolean factors BEGIN begin time END end time。

查询被分解为两部分：SELECT–FROM-WHERE 子句和 BEGIN-END 子句。SELECT–FROM-WHERE 子句为标准查询子句（SQC），存储在数据结构 query steM 中。满足 SQC 的元组记录在 Result Structure 中。BEGIN-END 子句表达了一个滑动的窗口，该子句存储在一个独立的结构 WindowsTable 中。新的数据流入系统后，被分配一个全局的元组 ID（称为 tupleID）和物理时间戳（physicalID），然后插入到数据结构 Data Structure 中，Result Structure 中的元组中包含元组的 tupleID 和 physicalID。新的数据元组也可以检索其它的 Data steM，实现连接操作。注册一个查询后，被分配一个唯一的 ID 号（queryID），标准查询子句插入去 query steM 中。新查询主动检索 Data steM，满足条件的元组记录到 Result Structure 中。由此可见，Psoup 将数据和查询都作为流来处理。最终查询结果的获得是根据 BEGIN-END 子句检索 Result Structure，获取 tupleID 后，根据 tupleID 从 Data steM 中得到元组。查询的执行过程如图 1-1 所示。

由上述分析可以看出，Data SteM，Query SteM 和 Result Structure 是 Psoup 中最重要的三种数据结构。每个数据流有一个 Data SteM 数据结构，用于存储和索引数据流，Data SteM 采用基于树的索引结构，每个属性建立一颗树。整个系统有一个 Query SteM 结构，用于存储和索引查询，树结构是用来表示满足 SQC 的元组，常用的结构由两种：a、二维表结构，每个 From-list 有一个独立二维表结构，行根据元组的物理时间戳进行排序，列根据查询 ID 来排序；b、每个查询带有一个链表，包含所以满足查询条件的元组。

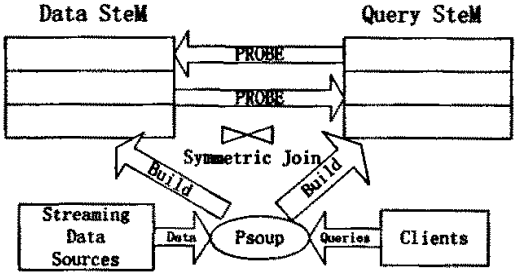

图 1-1 Psoup 查询的执行

从 Psoup 的基本实现技术可以看出，Psoup 将数据和查询都作为流来处理，因此新数据可以主动检索已注册的旧查询，新查询也可以查询已经存在的旧数据，而其他数据管理系统要么将数据作为流，要么将查询作为流，不能将二者同时作为流来处理。Psoup 将查询和结果的传送分开处理，查询在服务器端不停地被执行，结果记录在 Result Structure 中，结果记录在 Result Structure 中，客户端通过 Result Structure 获得结果，因此客户端短时间的离线也不会影响查询结果。

目前 Psoup 仅支持内存操作，它今后的研究方向是支持磁盘上的流存储和研究物化视图与滑动窗口之间的关系。

1.3.4 国内研究情况

国内方面，这方面的研究正在起步，文献资料也比较少，有些学校和研究所正在对数据流进行研究。复旦大学集中在流数据管理和挖掘的研究，中科院计算机技术研究所试图构建一个面向网络信息安全的数据流计算模型，重点研究的一些问题有网络数据流的获取、网络数据流建模、数据流查询模型、数据流计算算法等，他们的目标是在这个模型的基础上开发出具体的有显示度的宏观网络监控应用系统。我们现在正在研究基于硬件预处理器的数据流管理原型系统 SEUSTREAM，该系统基于硬件预处理器的体系架构，能够处理多条高速数据流上的多用户并发查询。

1.4 主要研究工作和内容安排

本文主要围绕数据流管理系统设计和查询处理方面进行研究，本文的主要研究工作有以下几个方面：

1.设计了连续流查询语言 X-SQL。由于数据流应用的特殊需求以及数据流本身的特点，使得数据流上的查询与传统关系上的查询有很大的不同。作为数据流管理系统的一个重要组成部分，我们设计了流查询语言 X-SQL，通过类似 SQL 的语法和滑动窗口结合，X-SQL 能支持数据流上的连续查询语义。同时，X-SQL 去除了 SQL 中的一些不需要的元素，使得语言本身简洁、易写，语义更加明确。

2.设计实现了 X-SQL 的编译器，该编译器能够按照用户输入的并发查询进行分析处理，并生成系统前端进行数据流处理所需要的初始化状态参数和指令序列。

全文的组织如下：

第一章介绍了课题的研究背景，以及数据流管理系统研究的主要内容和国内外的研究现状。第二章对基于硬件预处理的的数据流管理系统进行了介绍。第三章介绍了可以支持连续查询语义的流查询语言 X-SQL。第四章给出了 X-SQL 编译器的设计过程。第五章是对本文的总结。

第二章 基于硬件预处理的数据流管理系统

数据流的流动性和无限性和计算资源的有限性之间的矛盾，使得如何提高数据处理速度成为数据流管理系统的关键。现在的大多数据流管理系统都是基于软件实现的，会存在一定的处理延迟，在数据流速达到一定程度的时候，处理速度就会成为系统性能瓶颈。为了能够快速及时地处理高速数据流，我们提出了一种新的数据流处理系统体系结构——采用硬件预处理的并行数据流管理结构。本章主要介绍基于硬件的数据流管理系统SEUSTREAM的体系结构和基于FPGA的数据流预处理器的设计。

2.1 数据流管理系统（DSMS）

2.1.1 DSMS的基本需求

数据流和连续查询的特性要求数据流管理系统必须具备下述功能：

- 数据模式和查询语义必须允许基于顺序和基于时间的操作，如在5分钟尺寸大小的移动窗口上的查询。
- 长时间运行的查询在执行生命期中可能会遇到系统条件的变化，例如流速率的变化,需要设计具有自适应性的查询计划和调度策略。为确保可伸缩性，需要负载均衡。流查询计划可能不会使用必须在结果产生之前消耗完全部输入的模块化操作。多个近似连续查询的共享执行是必要的。
- 在线流算法是受限的，只能一遍扫描数据。不能存储全部的流暗示着需要使用近似概要结构，称为大纲(synopses)摘要(digests)，作用在概要上的查询可能会返回不精确的结果。
- 监控实时数据流的应用必须快速响应不同寻常的数据值。为即时准确地响应用户查询，通过服务质量保证(QoS)的检测指导系统调度和负载均衡的策略。

2.2 SEUSTREAM体系结构

为了能够快速及时地处理高速数据流，对数据流采用专门硬件进行预处理是一个很好的选择。基于硬件预处理数据流管理系统SEUSTREAM并不仅仅依靠查询优化、系统调用等传统手段来提高数据流的处理速度，而是考虑一种全新的体系结构来达到目的。图2-1是基于硬件预处理数据流管理系统的体系结构示意图[34]。

基于硬件预处理的数据流管理系统分为两部份：前端预处理器和后端数据引擎。前端预处理器负责对进入系统的数据流进行预处理，包括通过系统后端传送来的控制命令进行状态初始化，对数据流数据进行滤波、数据压缩和数据加标记处理等。前端预处理器采用软硬件协同的方式以提高数据处理速度。经过前端处理之后的数据通过网络传往后端，经过数据流划分控制器，在并行调度器的控制下，根据用户查询请求得出的调度策略动态划分数据流元组，在并行查询执行器的控制下由各自的处理器并行地完成查询处理。

在系统的后端数据引擎，用户提交的查询经过查询语法分析器处理后保存在注册查询缓冲中，用户提交的查询经过词法、语法及语义分析，并进行查询优化，得到查询抽象语法树，并针对数据流预处理器对抽象语法树进行处理，生成其所需的控制信息，经过查询输出控制器下载到前端硬件查询处理模块。

系统后端的数据引擎主要负责连续查询的语法分析、并行查询计划生成、将预处理过的元组划分到不同的数据缓存，对并行处理得到的查询结果的进行最后组装分发及负载均衡和服务质量监控

等功能。系统前端的查询预处理器接收后端数据引擎的查询请求，对多路数据流采集器传送来的流数据进行处理，并将处理结果送交至后端查询引擎。

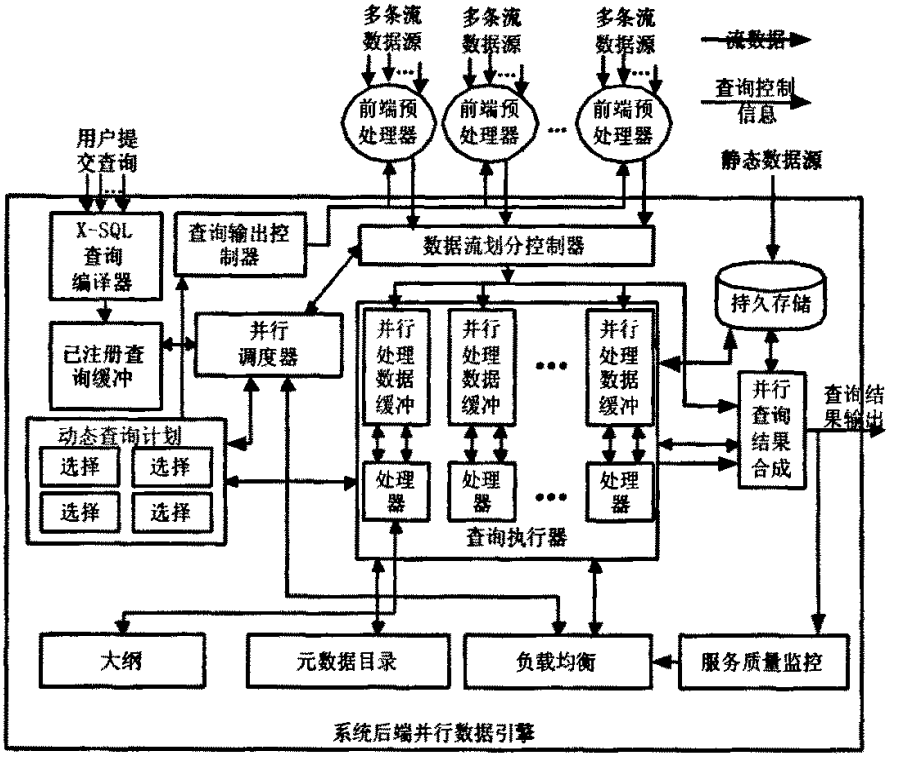

图 2-1 基于硬件预处理的数据流管理系统体系结构图

2.3 基于 FPGA 的数据流预处理器

2.3.1 硬件预处理器的研究背景

数据流具有大量、快、不可预期、无限的特征，这决定了对数据流的查询只能是实时的或者近似实时的。而目前大多 DSMS 的实现都是基于软件实现的，软件实现具有灵活程度高的特点，但是处理速度还是有一定限制，当数据流速度在一定范围内的时候虽然可以工作的很好，但是当数据流速达到一定的程度以后，处理速度就会成为瓶颈。比较可行的一个解决方法是设计专门的硬件，将数据流处理的一些操作放到硬件中去实现。

SEUSTREAM 由一个后端数据引擎和一些前端数据流预处理器组成。前端接收从后端根据注册查询生成的控制信息，然后根据控制信息对数据流进行处理。

2.3.2 基于 FPGA 的数据流预处理器的体系结构

数据流预处理器的作用是对进入系统的数据流进行预处理，主要是通过系统后端传送来的控制命令进行状态初始化，对数据流数据进行滤波、数据压缩和数据加标记处理。前端预处理器采用软硬件协同的方式以提高数据处理速度。经过前端处理之后的数据通过网络传往后端。数据流预处理器的体系结构见图 2-2。

网络控制模块负责数据流预处理器与外界的通讯功能，包括：

- 从数据流采集器传送过来的多个源数据流
- 从系统后端传送过来的查询控制命令

➢ 向系统后端提交查询结果流

目前该预处理器已经可以完成多条数据流查询的连接操作。对于投影、选择和聚集等操作将在下一步的工作中实现。

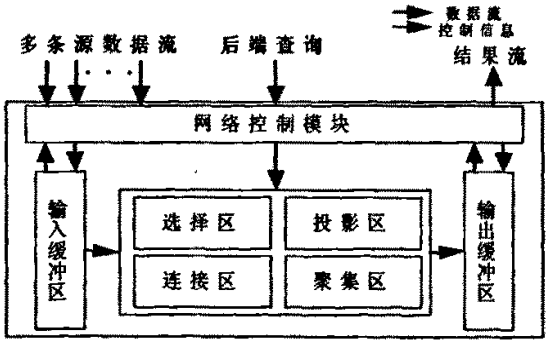

图 2-2 数据流预处理器的体系结构

2.3.3 数据流预处理器连接算法及其实现

目前大多数学者研究的数据流连接算法都是基于软件实现的，直接用硬件实现会有很大的困难。目前数据流预处理器已经可以实现多数据流、多查询的并发连接处理，其所采用的算法是适合用硬件实现的流连接方法 M3Join [31,32]。算法以多线程并行和类似路由器的处理方式保证数据的高速处理，同时通过先分解连接请求后组合部分连接结果的方法保证连接的高效率和高共享度。

M3Join 的基本步骤：(1)将原始数据流元组(图 2-3)插入到缓冲区(insert)；(2)该元组依照路由表进入连接区连接(probe)；(3)删除各连接区中过期的元组(invalidate)。

图 2-3 是数据流元组的结构，包括路由标记（Route Tag）、连接因子标记（Join Factor Tag）、旧时间戳（Old TimeStamp）、新时间戳（New TimeStamp）和数据域（Data Fields）组成。

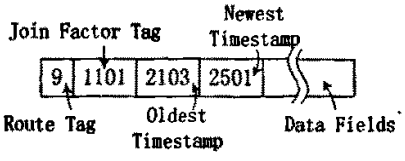

图2-3 元组结构

算法 M3Join 的实现架构 Roujoin 由四部分组成：输入缓冲区(Input Buffer)；连接路由表(Join Routing Table)；连接区(Join Area)；输出缓冲区(Output Buffer)。

其中，连接区由窗口缓冲区(Window Buffer)、连接因子区(Join Factor Area)、因子组合区(Factor Combination Area)和探测元组缓冲区(Probe Tuple Buffer)组成。 该架构是数据驱动的，各区域仅对本地的数据执行操作，然后交给下一区域继续处理。 输入缓冲区将所有原始流元组按顺序存放，一旦处理速度滞后时可以缓冲。流元组由连接路由表引导至合适的连接区执行连接操作。窗口缓冲区存放相应流时间窗内的原始流元组，这些数据可以按时间顺序存放或再按关键字建索引，以提高查找速度。另外为避免更新时元组移动，窗口缓冲区设置为循环队列。如果探测元组缓冲区存在数据，则按顺序取出一元组与窗口缓冲区的元组执行连接因子区所有因子操作，将结果标记在元组的连接因子标记位(Join Factor Tag)，然后根据元组连接前路由标记(Pre-join Route Tag)、连接因子标记位和因子组合区内容设置连接后元组的路由标记。输出缓冲区存放连接后的待输出元组，并根据各连接的最大窗口确定是否将结果输出。

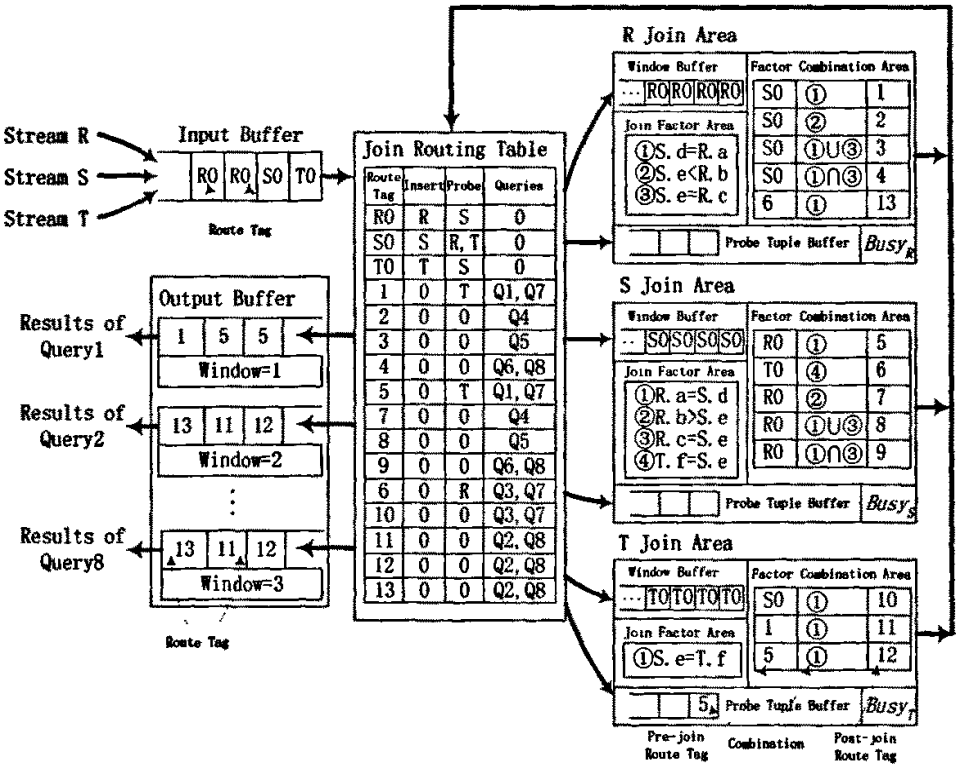

图2-4 实现架构Roujoin

2.3.3.1 架构 Roujoin 初始化

算法 2.1 Roujoin 初始化算法

输入:并发连接请求;

输出:Roujoin,包括连接路由表、连接因子区、因子组合区和输出缓冲区中的数据。

(1)按先与后或规范连接条件表达式。一般情况下与操作比或操作减少结果元组,通过等价转换将与操作下推。

(2)将或操作的连接请求分解为等价的多条请求。

(a)对需或操作连接因子按参与连接流归组。

(b)对每一组重写一连接请求,系统分别处理新的请求,只在输出结果时将其合并。

(3)提取所有不同连接因子,建立连接请求表。

(4)根据连接请求表设置连接因子区,填入相关因子。

(5)根据连接因子和连接请求表设置因子组合区的连接前标记、组合及连接后路由标记。

(6)设置窗口缓冲区大小。采用 Large Window Only (LWO)方法。将缓冲区大小设成中所有因子窗口的最大值。

(7)根据连接请求表构造连接路由表,采用规则:

(a)原始元组要插入到相应的窗口缓冲区,其它置 0;

(b)连接后元组改变路由标记,该标记同时存在于连接路由表;

(c)需要探测的元组设置相应连接区标记,否则置 0;

(d)元组符合某查询请求则设置其编号,否则置 0。

采用简单的启发式算法确定三流以上连接的顺序:假设三流 R、S、T,如果中间结果 R⋈S 也是另一并发请求的结果,则先执行 R⋈S。按(7)生成的连接路由对于三流以上的连接会产生重复元组(如 S 元组先探测 R 后探测 T,与同一 S 元组先探测 T 后探测 R,生成的三元组会重复),检查并删除能

产生重复连接元组的路由。

(8)按连接请求表设置输出缓冲区的窗口值，输出时判断是否将结果输出。

2.3.3.2 流元组执行连接

连接路由表和每个连接区及各输出缓冲区均设置一线程,流元组在各线程中完成处理和传递,最终完成连接。连接路由表线程的任务有三项:(1)将原始元组插入到窗口缓冲区;(2)将元组插入相应的探测元组缓冲区;(3)将结果元组输出到相应的输出缓冲区。连接区线程的任务是依次读取探测元组缓冲区的元组,执行连接,改变结果元组路由标记,并将该元组传递给连接路由表线程。当连接区 j(设所有连接区编号为 $1..J$)在处理数据时,则标记 $Busy_j$ =1,否则 $Busy_j$ =0。输出缓冲区线程的任务是判断元组是否符合窗口值。

为方便表述, 引入以下符号和函数:

α :输入缓冲区头(最先进入)元组;

ω :连接区返回连接路由表的元组;

π :连接区 j 的探测元组缓冲区头元组;

$\upsilon_{1..m}$: π 的连接因子标记位,初始值均为 0;

$\sigma_{1..m}$:连接区 j 的连接因子;

$\mu_{1..p}$:连接区 j 的因子组合;

$\theta_{1..n}$:连接区 j 的窗口缓冲区元组;

γ :输出缓冲区头元组;

λ :元组路由标记;

$Factor(\pi, \theta_i, \sigma_q)$: π 与 θ_i 作 σ_q 操作的逻辑结果;

$Join(\pi, \theta_i, \omega, \lambda)$: π 与 θ_i 连接后为 ω, 并记路由标记 λ;

$InsertValue(\lambda)$: λ 的路由的插入值;

$ProbeValue(\lambda)$: λ 的路由的探测值;

$QueryValue(\lambda)$: λ 的路由的查询请求值;

$PreJoinTag(\mu_i)$: μ_i 的连接前标记;

$Combination(\mu_i, \upsilon_{1..m})$: $\upsilon_{1..m}$ 执行 μ_i 对应组合操作逻辑结果;

$PostJoinTag(\mu_i)$: μ_i 的连接后标记;

算法 2.2. 连接路由表线程

(1) while (TRUE)

(2) 清除所有窗口缓冲区过期元组;

(3) if (($\sum_{j=1}^{J} Busy_j$ =0) and (ω =NULL) and (α ≠NULL)) then

(4) λ := $RouteTag(\alpha)$;

(5) α 插入到 $InsertValue(\lambda)$窗口缓冲区;

(6) α 插入到 $ProbeValue(\lambda)$探测元组缓冲区;

(7) end if;

(8) if (ω ≠NULL) then

(9) λ := $RouteTag(\omega)$;

(10) if ($ProbeValue(\lambda)$ ≠0) then

(11) ω 插入到 $ProbeValue(\lambda)$探测元组缓冲区;

(12) end if;

(13) if ($QueryValue(\lambda)$ ≠0) then

(14) ω 插入到 $QueryValue(\lambda)$输出缓冲区;

(15) end if;

(16) end if;
(17)end while;
算法 2.3 连接区 j 线程
(1) while (TRUE)
(2) if ($\pi \neq$NULL) then
(3) $Busy_j$:=1;
(4) for (i:=1;$i \leq n$;i++)
(5) for (q:=1;$q \leq m$;q++) //执行连接因子区操作
(6) if ($Factor(\pi, \theta_i, \sigma_q)$ =1) then
(7) υ_q:=1;
(8) end if
(9) end for;
(10) for (q:=1;$q \leq p$;q++) //执行因子组合区操作
(11) if (($RouteTag(\pi)$=$PreJoinTag(\mu_q)$ and ($Combination(\mu_q, \upsilon_{1..m})$=1)) then
(12) $Join(\pi, \theta_i, \omega, PostJoinTag(\mu_q))$;
(13) ω 记录 π, θ_i 的最旧和最新时间;
(14) ω 返回连接路由表线程;
(15) end if
(16) end for
(17) end for;
(18) else
(19) $Busy_j$:=0;
(20) end if;
(21) end while;
算法 2.4 输出缓冲区线程
(1) while (TRUE)
(2) if (($\gamma \neq$NULL) and (γ 最新时间戳-γ 最旧时间戳<Window)) then
(3) 输出;
(4) end if;
(5) end while;

当系统被注册新查询或取消旧查询时,在 $\sum_{j=1}^{J} Busy_j = 0$ 并且输出缓冲区为空时,暂停处理新元组,根据请求调整架构 Roujoin,然后继续执行连接。

2.3.3.3 M3Join 的硬件实现

在 SEUSTREAM 系统中数据流预处理器对 Roujoin 架构设计了一个名为 WJASIP（Window Join ASIP）的 ASIP 实现[31]。假定每一个原始的数据流由 4 个属性，并且每个属性含有 8 位。我们使用 2 个比特位来区分原始流。路由标记位 Route-tags 占用 8 位。因此一个 post-join 元组的位数是 136 位。WJASIP 的设计见图 2-5。

在 WJASIP 中设计了 7 条专用指令，通过将算法翻译成专用指令序列来执行这个算法。这些指令序列就运行在硬件的 IEU 中。因此，当系统中被注册的查询改变时，硬件电路本身不需要进行改动，而只是改变 IEU 中的指令序列即可。指令的长度都是 20 位的。我们使用这些指令来完成算法 2.3 的第 5 到 16 步。也就是说，我们使用这些指令来完成一个探测元组和一个窗口元组的比较和连接操作。在一个 IEU 中有一个 64 位的寄存器来保存比较结果和逻辑操作结果。如果比较结果位（算

法 2，步骤 5～9）全是 0，那么这些位的结合(combinations)也是 0。那么，就没有必要继续处理了，指令 4 就是设计来处理这种事情的。它的工作仅仅是来检测该寄存器的 0～15 位。在 IEU 中我们将窗口缓冲区的最大值设置为 2048 个元组。探测 FIFO 中的一个元组和窗口缓冲中的每一个元组构成了一个元组对。对于每一个元组对，都要执行一遍 RAM 中的指令序列。然后电路就会取出下一个探测缓冲区中的元组来执行指令序列，直到探测缓冲区空为止。

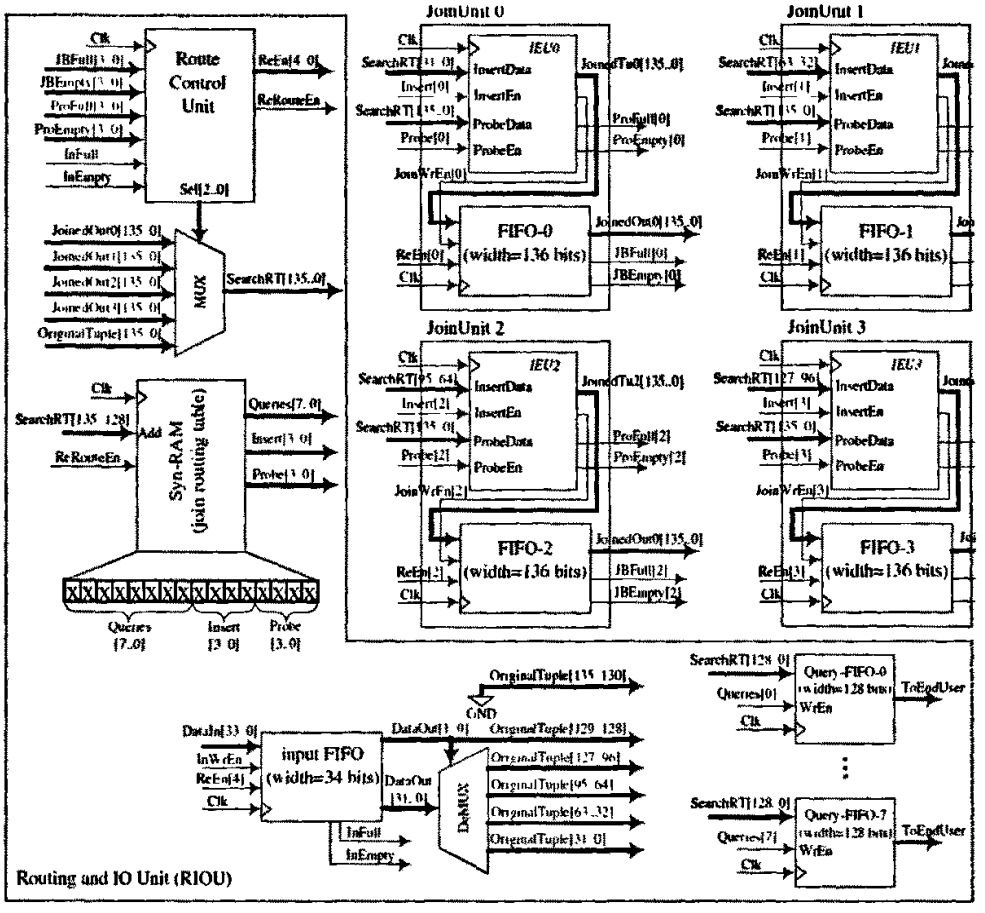

图 2-5 WJASIP 设计

WJASIP 中的指令如下：

1）逻辑操作：

a） AND op1Addr, opAddr, resultAddr

b） OR op1Addr, opAddr, resultAddr

其中 AND 指令中：AND 操作代码占两位，值为 10；op1Addr 是操作数 1 的寄存器地址，占 6 位；op2Addr 是操作数 2 的寄存器地址，占 6 位，resultAddr 是保存结果的寄存器地址，占 6 位。

2）比较跳转操作：

JMPCT routeTag, instructionAddr

其中 JMPCT 的操作代码占两位，值为 01；routeTag 是待比较的路由标记，占 8 位；instructionAddr 是如果 routeTag 不同则跳转的指令地址，占 10 位。

3）比较操作：

CMP attr1, attr2, cc, addr

其中 CMP 的操作代码占四位，值为 0011；attr1 是窗口元组（window tuple）属性，占 2 位；attr2 是探测元组（probe tuple）属性，占两位；cc 是比较代码，占三位（000 代表<，001 代表=，010 代表>，011 代表>=，100 代表<>，101 代表<=，111 代表输出衡为 0）；addr 是

比较结果位寄存器地址，占六位。

如果 attr1 和 attr2 满足 cc 代表的比较操作，则结果输出位置为 1，否则置 0。

4）跳转到最后一条指令：

JMPLI

其中 JMPLI 操作代码占四位，值为 0001，其余位未定义。该条指令将使得程序跳转到最后一条指令。

5）无条件跳转指令

JMP address

其中 JMP 操作代码占五位，值为 00001；address 为欲跳转到的指令地址，占 10 位；其余五位未定义。

6）连接操作：

JOIN regAdd, routeTag

其中 JOIN 操作代码占四位，值为 0010；regAdd 占六位；routeTag 占八位。

7）无操作指令：NOP

其中 NOP 操作代码占五位，值为 00000；其余十五位未定义。

2.4 本章小结

本章介绍了我们实验室正在研究设计的基于硬件预处理的数据流管理系统 SEUSTREAM。该系统并不仅仅依靠查询优化、系统调用等传统手段来提高数据流的处理速度，而是考虑一种全新的体系结构来达到目的。通过面向数据流的专门的硬件来对数据流进行预处理来提高数据流管理系统对数据流的查询处理的速度。本章还介绍了数据流预处理器的体系结构和该预处理器所实现的多流多查询并发连接算法 M3Join。目前该处理器已经可以实现多条数据流的并发连接算法。该处理器也是我们接下来第四章 X-SQL 编译器的目标机器。

第三章 连续流查询语言 X-SQL

DSMS 对数据流进行查询需要一种合适的查询语言，查询语言的设计也是数据流管理系统的一个重要部分。但是由于传统的关系数据库查询语言 SQL 不能够支持连续查询语义，对数据流查询力不从心。为了满足数据流应用的要求和规范，需要设计用于处理数据流查询的查询语言。本章主要讨论我们 SEUSTREAM 中使用的连续流查询语言 X-SQL。

3.1 流查询语言的需求背景

SQL(Structured Query Language)，即结构化查询语言，源于美国 IBM 公司 1975-1979 年研制的关系数据库实验系统 SystemR，后经众多计算机公司和软件公司采用，并不断修改、完善，最后指定为美国国家标准。不久，国际标准化组织 ISO 采纳它为国际标准。从此以后，市场上较为流行的数据库管理软件，大都在不同程度上以各种方式装配 SQL 语言接口，从而使关系数据语言 SQL 成为广大用户最为欢迎的语言，也是他们与数据库打交道的一个重要工具。

SQL 语言对于关系数据库中的数据查询得心应手，但是对于无界数据流的查询却力不从心。因为传统关系型 DBMS 的数据是静态、持久的，从一个关系（或称为表）中查询数据非常简单，不要考虑时间因素，而查询结果是一个关系元素的集合。在对数据流查询的情况下，关系的对应物是数据流，动态性、易失性等实时特点使得对于流的查询需要使用滑动窗口的概念。但是由于流和关系的相似性和 SQL 语言简洁易用的特性，现在许多数据流管理系统的连续流查询语言都采用类似 SQL 的语法表达方式。X-SQL 流查询语言也是采用了类似 SQL 的语法，对一般的查询表达式中添加了滑动窗口的文法因子，使之可以对数据流进行连续查询。

3.2 数据流的实例介绍

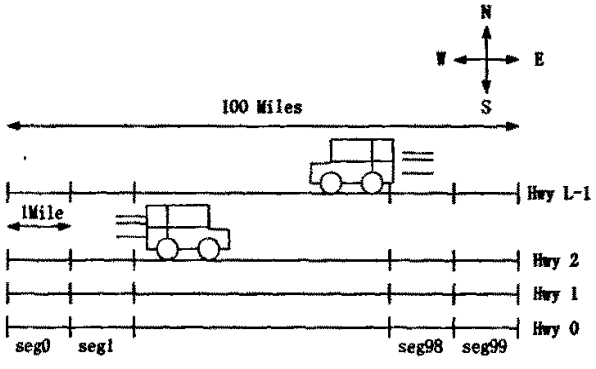

图 3-1 道路交通管理应用

本节我们介绍连续查询的数据流实例——道路交通管理应用。这儿我们只是对该应用进行简单介绍，具体的信息可以参见文献[3]。

道路交通管理应用中使用可变的收费计量方式（基于交通状况对车辆的通行费进行自适应和实时的计算）来管理高速公路系统上的交通流量。车辆收费的依据是车辆所在的路段的车流量情况和车辆自身的速度，为了能够达到对通行费进行可变计算的目标，每一个车辆上装备了一个传感器来持续地将它的位置和速度信息发送到中心服务器。服务器将从所有的高速公路系统上所有的车辆传送来的信息聚集起来，实时地计算交通费用，并将计算出来的费用回送到每一个车辆上去。

图 3-1 显示了道路交通管理应用系统。有 L 个高速公路，分别编号为 0,...L-1。每一个高速公路

是 100 英里，东西方向。交通流向是双向的。在每一个方向，一个高速公路被分为 100 个 1 英里的路段，并且在每一个路段的交界处有离开坡道和进入坡道。当一个车辆是在一个高速公路上时，它的位置和速度信息每隔 30 秒向中心服务器报告一次。速度单位是 miles/hour，位置由三个属性来指定：高速公路的编号(0~L-1)，方向(east, west)，和高速公路上的位置（用从高速路西端开始的英尺数来表示）。

当车辆行驶在拥挤的路段时要付通行费，当运行在非拥挤路段时不需要付费。如果一个段上最近 5 分钟的的所有车辆平均速度小于 40miles/hour，那么这个路段时拥挤的。拥挤段的计费公式如下：baseToll×（numVehicles-150）^2，这里的 baseToll 是一个预先指定的常数，numVehicles 时当前时间该段上的车辆数。注意一点：通行费是随着车辆的进入或退出而该段而变化的。当服务器检测到一个车辆进入拥挤段时，服务器输出该拥挤段当前通行费。

在流的角度来讲，本文中简化后的应用有：一个输入流——车辆的位置和速度流；一个计算通行费的连续查询；一个通行费输出流。

3.3 流上的连续查询语言

DSMS 对数据流进行查询需要一种合适的查询语言，查询语言的设计也是数据流管理系统的一个重要部分。但是传统的关系数据库查询语言 SQL 不能够支持连续查询语义，对数据流查询力不从心。为了满足数据流应用的要求和规范，需要设计用于处理数据流查询的查询语言。

3.2.1 数据流定义

在数据流管理系统(DSMS)中，查询的对象是流，为了给出流连续查询的精确语义，本节介绍数据流的定义[22]。就像在标准关系模型中，每个关系有一个固定的模式，这个模式由一组有名字的属性组成。为每一个流元素的到达我们假定一个离散的有序的时间域 T。一个时间常量是 T 中的一个值。更具体一点，我们把 T 表达为非负的整数集合{0,1,...}，其中 0 代表最早的时间常量。时间域 T 对一个应用领域里的时间进行建模，而不是指系统时钟。因此，虽然 T 可能经常是 Datetime 类型，我们的语义仅仅需要一些离散有序的域。

定义 3.1 流： 一个流 S 是一个（可能是无限的）元素<s,t>的集合，这里的 s 是一个属于模式 S 的元组，t 是 T 中的一个值，用来表示该元素的时间戳。

注意这里的时间戳不是流模式的一部分，并且同一个流中可能会有 0 个或多个元素含有相同的时间戳。我们仅仅需要给定的时的数据流元素的数目有限即可。

有两类流：基本流，到达 DSMS 的源数据流；导出流，是从查询中的操作生成的中间流。我们使用数据流元组的概念来表示数据流中的元素。在道路交通管理应用实例中，仅有一个基本流，包含车辆速度—位置度量值，模式如下：

PosSpeedStr(vehicleId,speed,xPos,dir,hwy)

其中属性 vehicleId 代表车辆，speed 代表速度（单位 MPH），hwy 是高速公路的标号，dir 是方向（west or east），xPos 代表车辆在高速公路中位置（从 west 端开始的英尺数）。时间域是 Datetime 类型，在这个应用中时间戳代表了位置及速度值被采样时刻的物理时间。

在实例中如果用户需要考虑车辆 vehicleId 为 1 的车发送过来的数据，那么可以查询流中该车辆的数据形成新的数据流，即导出数据流：

PosSpeedStr("1",speed,xPos,dir,hwy)

3.2.2 连续查询语义

为了简便起见，假设时间是一系列整数。设 A(Q,*t*)表示连续查询 Q 在时刻 t 的查询结果，τ 表

示当前时间，0 表示开始时间。如果一个连续查询是单调的，那么它对新到来的数据项进行查询，并对结果进行二次评价，将满足条件的元组追加到结果中去。由此，一个单调的连续查询 Q 在 τ 时刻的查询结果为[2]。

$$A(Q,\tau)=\bigcup_{t=1}^{\tau}(A(Q,t)-A(Q,t-1))\cup A(Q,0) \tag{1}$$

相应的，非单调查询则需要对每一个二次评价之后的结果进行重新计算，由此产生如下语义[16]：

$$A(Q,\tau)=\bigcup_{t=0}^{\tau}A(Q,t) \tag{2}$$

3.2.3 流查询语言分类

现在正在研究的各种数据流管理系统采用了适合各自体系结构流查询语言。目前提出的关于数据流的查询语言主要有三类：基于关系的语言、基于对象的语言和基于过程的语言[1]。

（1）基于关系的流语言

目前基于关系的流查询语言主要有三种：CQL， StreaQuel 和 Aquery。它们都具有类似 SQL 的句法，同时还提供了对窗口和排序的支持。CQL(Continuous Query Language)用于 STREAM 系统，该系统将数据流和窗口看作通过时间戳排列成的关系。关系到流的操作提供将查询结果转化为流的功能，利用这些操作用户可以显式的指定查询语义，而且也可以显式地定义采样率。

StreaQuel 是 TelegraphCQ 系统实现的查询语言，支持增强窗口技术，并且其所有的输入和输出都采用数据流形式而不再需要关系到流的操作。每一个 StreaQuel 查询后面都跟随着一个 for-loop 结构，其中变量 t 指明了循环的时间。Loop 循环包含了一个 WindowIS 语句，它指明了窗口的类型和大小。其中 S 表示一个流，NOW 表示当前时间。为了在 S 上制定一个大小为 5 的滑动窗口并使查询执行 50 个时间单位，则应将以下 for-loop 语句加到查询语句的后面(可以看见，如果更改 for-loop 语句中的增量条件为 t=t+5，那么查询将每 5 个时间单位执行一次)：

```
For (t=NOW;t<NOW+50;t++)
WindowIS(S, t-4, t)
```

Aquery 查询语言是以查询代数和 SQL 为基础的用于排序的语言。表的属性列被看作数组，可以使用与顺序相关的操作：next， previous， first 和 last 等。例如：一个对股票报价流进行的连续查询，要求连续报道 VIA 公司股票的不同的价格变化，可以将其定义为如下形式：

```
SELECT price-prev(price) FROM Trades
WHERE company='VIA'
```

（2）基于对象的语言

面向对象的数据流建模方法是根据类型层次对数据流的元素进行分类。这种方法在 Tribeca 网络监控系统中得到了应用。其他可能的方法是将数据源模拟成 ADTs(抽象数据型)。在 COUGAR 传感器网络中传感器的每一个类型被模拟为 ADT，ADT 的接口由传感器信号处理方法组成。目前提出的这种查询语言具有类 SQL 的语法，并且包含了$every()语句，用于指明重复执行的频率。

（3）基于过程的语言

过程式查询语言则让用户自己指定数据的流动，其代表为 Aurora。在优化阶段，虽然 Aurora 系统可以延迟执行重组织、增加、删除操作，但用户可通过一个图形化结构自己构造查询计划，自己组织 box(相当于查询操作)并将它们直接连接起来指定数据流向。Aurora 系统包含了许多其它语言中未明确定义过的操作，如 map 将一个函数应用于每一个数据项， resample 在一个窗口中重新插入丢失的数据项、drop 则当输入速率很快时将任意地摒弃一些数据。

所有的语言都包括对窗口的扩展支持。比较前面提到的几个基本操作符，除了 top-k 和模式匹

配，所有必需的操作符都在语言中显式定义。而用户定义的聚集可以实现模式匹配的定义功能，可以满足未来流应用的需要。总的说来，带有窗口和序列的附加功能的基于关系的语言目前是最为流行的范例。

3.3 流查询语言 X-SQL

3.3.1 X-SQL 概述

X-SQL 是我们基于硬件预处理的数据流管理系统中所使用的连续流查询语言。X-SQL 的是一种基于关系的类 SQL 语言，这有助于我们利用关系代数的理论基础和 SQL 语言的技术基础。X-SQL 通过滑动窗口操作的支持来实例化连续查询的抽象语义。用户可以通过在系统的后端输入 X-SQL 的连续查询，然后通过交叉编译后得到的查询指令及相关信息通过网络传送到数据流预处理器中去，完成对数据流的各种查询操作。

X-SQL 语言由两部分组成，数据流定义语言（DSDL）和数据流查询语言（DSQL）。其中 DSDL 用来为数据流管理系统输入有关的元数据信息，DSQL 可以被用户使用对数据流管理系统中的数据流进行查询。

3.3.2 滑动窗口模型

滑动窗口（sliding window）是数据流上应用比较多的一种特殊的数据抽样方法[23,24]。滑动窗口是指在数据流上设定一个区间，该区间只包括数据流中最近到来的那部分数据。随着新数据的到来，窗口向前移动，用最新的数据替换最旧的数据。与其它抽样方法相比，滑动窗口的优点在于其语义明确。更重要的是，在许多应用中，例如传感器网络，用户关心的只是数据流中新近到来的那部分数据。此时，滑动窗口是一种十分理想的抽样方法。通过对连续查询的范围限定方式来划分，滑动窗口包括两种类型：一种是窗口中包含最近到来的 t 个元组，称为基于顺序(sequence)或基于计数(count)限定的滑动窗口。另一种是窗口中包含最近 t 个时间单元内到来的元组，称为基于时间(timestamp)限定的滑动窗口。数据流上连续查询处理的执行方式有两类：一类是立即执行方式；一类是周期执行方式。立即执行是指数据流上每个新数据到来，均触发执行一次查询；而周期执行是指每隔固定的时间间隔触发执行一次查询。由于连续查询的执行方式不同，所以滑动窗口的滑动方式也随之不同。当连续查询是立即执行方式时，窗口的滑动是以元组为单位的，即每到一个新的元组，窗口就向前滑动，计算查询结果；当连续查询是周期执行方式时，窗口的滑动是以固定个数的元组或固定的时间间隔为单位向前滑动，计算查询结果。

X-SQL 支持的连续查询是基于滑动窗口概念的。在数据流的连续查询系统中，滑动窗口可以看做是数据流的有限部分的集合。任意时刻的滑动窗口包含数据流的有限片段的历史快照。X-SQL 支持两种类型的滑动窗口操作符：基于时间的滑动窗口和基于元组的的滑动窗口。

3.3.2 X-SQL 数据流定义语言（DSDL）

在关系数据库管理系统（RDBMS）中，每个关系对应一张表，每张表保持一个固定的模式，这个模式由一组有名字的属性组成。在数据流管理系统(DSMS)中，查询的对象是数据流，类似 RDBMS 中的表，在 DSMS 中，每一个流也有自己的模式定义，这些元数据信息存放在数据字典中。有关数据流的元数据信息通过 X-SQL 中的 DSQL 语言进行输入。

1）创建一个流

用户可以通过声明流的名字和所有属性的名字及其类型来创建流，每个流有均一个隐式的时间戳属性 TimeStamp，不必显式的定义。

CREATE STREAM <*stream name*>(

<attribute name> *<dataType>*

[,*<attribute name>* *<dataType>*]*

);

在 X-SQL 命令中可以自由使用空白（也就是空格，tab，和换行符），因为 X-SQL 的语法特性参照 SQL-92 标准而来。这就意味着你可以用和上面不同的对齐方式键入命令。两个划线("--") 引入注释。任何跟在它后面的东西直到该行的结尾都被忽略。与 SQL 类似，X-SQL 也对关键字和标识符大小写不敏感的语言，只有在标识符用双引号包围时才能保留它们的大小写属性。

date 类型应该是用来存储日期的。X-SQL 支持标准的 SQL 类型 int，smallint，real，double precision，char(N)，varchar(N)，date，time，timestamp 等。

例 3.1 在数据流实例道路管理应用中，有一个基数据流，包含车辆速度一位置度量值，其模式如下：

PosSpeedStr(vehicleId,speed,xPos,dir,hwy)

属性 vehicleId 代表车辆，speed 代表速度（单位 MPH），hwy 是高速公路的标号，dir 是方向（west or east），xPos 代表车辆在高速公路中位置（从 west 端开始的英尺数）。时间域是 Datetime 类型，在这个应用中时间戳代表了位置和速度的值被采样的时候的物理时间。

为了创建该流的元数据信息，用户可以使用如下的 X-SQL 语句：

```
CREATE STREAM PosSpeedStr (
    vehicleId     varchar(80),
    speed         real,
    xPos          int,
    dir           varchar(4),
    hwy           int
);
```

通过该语句的执行，数据流管理系统中创建流 PosSpeedStr 的相关元数据信息。

2）删除一个流

最后，如果不再需要某个流，那么可以用下面的命令删除它：

DROP STREAM streamName;

通过该语句的执行，DSMS 系统中流 PosSpeedStr 的元数据信息将一并消失。

3.3.3 X-SQL 数据流查询语言（DSQL）

要从一个流中检索相关信息就是对这个流进行连续查询。X-SQL 语言用 SELECT 语句来实现连续查询的功能。不同于传统 SQL 中的 SELECT 句型，该语句除了列出要返回的属性选择列表部分，列出从中检索数据的流的流列表部分，以及可选的条件以外，还有一个声明作用于流上的滑动窗口大小的窗口声明部分。X-SQL 语法如下：

SELECT *<targetListExpression>* [,*<targetListExpression>*]*

FROM *<streamName>* [AS *nickName*] [, *<streamName>* [AS *nickName*]]*

[WHERE *<whereCondition>*]

WINDOW *<windowExpression>*[, *<streamName>* [AS *nickName*]]*

在上面给出的 X-SQL 的 DSQL 的语法中，除去 WINDOW *<windowExpression>*部分的剩余部分和标准 SQL 的查询语句类似。X-SQL 正是通过 WINDOW 子句来支持滑动窗口查询，进而可以对数据流进行连续查询。下面介绍 WINDOW<windowExpression 的具体语法。

3.3.4 X-SQL 对滑动窗口的支持

在 X-SQL 查询语句中，WINDOW 部分的作用是对 FROM 部分的流列表中出现的数据流给予相应的窗口声明。通过 WINDOW 子句的作用，X-SQL 支持基于时间的滑动窗口和基于关系的滑动窗

口。

（1）基于时间的滑动窗口

流 S 上基于时间的滑动窗口用一个时间段 T 作为输入参数，作为滑动窗口的大小。在 X-SQL 中的查询语句中的 WINDOW 部分中在 S 在 FROM 部分出现的相应位置加上 T。在这儿我们不具体的说明时间段 T 的语法限制，但我们假定它是可以计算的时间。直觉上来讲，基于时间的窗口通过在一个有序的输入流上滑动大小为 T 的窗口来捕捉最新的部分来定义它的输出关系。正式的讲“SELECT * FROM PosSpeedStr WINDOW Range T”的输出关系 R 的定义如下：

$$R(\tau)=\{s|<s,\tau'>\in S\wedge(\tau'<\tau)\wedge(\tau'\geq\max\{\tau-T,0)\}$$

两个重要的情况是 T=0 和 T=∞。当 T=0 时，R(τ)中的元组是从流 S 中时间戳为τ的元组中获得的，使用“Now”表示。当 T=∞，R(τ)中的元组包含 S 中到τ为止的所有元组，使用“Unbounded”来表示。

例 3.2 在查询

SELECT * FROM PosSpeedStr WINDOW Range 30 seconds

中，”WINDOW 30 Seconds”表示查询输入流 PosSpeedStr 上一个 30 秒钟的基于时间的滑动窗口。在任意的时刻，相似地，在任意时刻“WINDOW Now”包含了该时刻的位置一速度度量值的包（有可能为空）。“WINDOW Unbounded”包含目前位置所有的位置一速度度量值的包。

（2） 基于元组的窗口

一个基于元组的滑动窗口用一个正整数 N 作为参数，具体在 X-SQL 查询中使用时在 WINDOW 表达子句中与该流相应位置加上“ROWS N“。直观地看，一个基于元组的窗口的输出关系是在一个有序的数据流中的最新的 N 个元组。正式的讲，对于查询“SELECT * FROM S WINDOW Rows N”的输出关系 R，R(T)包含 S 中的时间戳<=T 并且最大的 N 个元组（如果流 S 中所有的元组个数小于 N 的话，那么包含所有的元组）。假定我们指定了一个 N 个元组的滑动窗口，并且在一些时刻在第 N 个最近的时间戳有若干元组（为了清楚我们假设其它的 N-1 个更新的元组是各不相同的）。然后我们必须以某种方式“破坏联系”来产生窗口中的 N 个元组。我们假定这些联系被任意破坏。因此，基于元组的滑动窗口可能非决定的——因此可能不合适——当时间戳不唯一。N=∞的特殊情况时用“WINDOW Unbounded“来表示，因为它和基于时间滑动窗口时 T=∞等价。

例子 3.3 由于流元素时间戳是非唯一的，对于流 PosSpeedStr 来讲一个基于元组的滑动窗口不怎么能讲的通（N=∞的情况除外）。比如，在任意时刻滑动窗口“SELECT * FROM PosSpeedStr WINDOW Rows 1“代表最新的位置一速度度量值，这儿当多个度量值具有相同的时间戳时就会有歧义。

3.3.5 X-SQL 中滑动窗口的简单表达方式

X-SQL 中的语法比较灵活，允许几种简单的表达方式：

1） 基于元组的滑动窗口在查询表达时可以不加上 Rows 关键字，例如如下两条查询等价：

查询 1：SELECT * FROM S WINDOW Rows N;

查询 2：SELECT * FROM S WINDOW N;

2） 当一条查询语句中涉及的多条流的滑动窗口相同时可以在 WINDOW 子句中只写一次，例如下面两条查询是等价的：

查询 1：SELECT * FROM R,S WINDOW Rows N, Rows N;

查询 2：SELECT * FROM R,S WINDOW N;

相似地，下面两条查询也是等价的：

查询 1：SELECT * FROM R,S WINDOW Range T seconds, Range T seconds;

查询 2：SELECT * FROM R,S WINDOW Range T seconds;

3.4 本章小结

本章中，我们首先介绍了流查询语言的需求背景和一个数据流应用的实例——道路交通管理应用，然后给出了流上连续查询的抽象语义，并对现有的数据流查询语言进行了综述。然后着重介绍了我们基于硬件预处理器的数据流管理系统中使用的流查询语言 X-SQL，并结合前面提出的数据流应用实例中的数据流，介绍了 X-SQL 的具体设计和用法。

第四章 X-SQL 编译器的设计

查询处理在数据流管理系统中是一个非常重要的部分。在 SEUSTREAM 中用户提交的查询语句首先由提交到数据流管理系统查询缓冲区，然后经过查询编译器处理，生成查询计划，将得到的目标机器指令序列及资源分配初始化参数下载到目标处理器中去执行。本章主要讨论基于硬件预处理的数据流管理系统 SEUSTREAM 中的 X-SQL 查询编译器设计。

4.1 编译器概述

4.1.1 编译技术的发展

编译器是现代计算技术的基本组成部分。它担任翻译器的角色，将面向人的高级语言转换为面向计算机的机器语言。早在 1954 年至 1957 年期间，IBM 的 John Backus 带领的一个小组对 FORTRAN 语言及其编译器的开发，揭开了编译程序的新纪元。这个项目的开发的工作量和辛苦程度是很大的。几乎与此同时，Noam Chomsky 对自然语言结构的研究使得编译器的结构异常简单，甚至还有一些自动化。Chomsky 的研究导致了根据语言文法的难易程度以及识别它们所需算法来对语言进行分类，包括 4 个文法层次：0 型，1 型，2 型和 3 型文法，并且其中的每一个都是前者的专门化。2 型文法即上下文无关文法(Context-free grammar)，它被证明使程序设计语言中最有用的，而且今天它以及代表着程序设计语言结构的标准方式。有穷自动机（Finite Automaton, FA）和上下文无关文法紧密相关，它们与 3 型文法对应，用来识别由正则表达式定义的词法记号。分析问题的研究是在 60 年代和 70 年代，它相当完善地解决了这一问题，现在它已经是编译理论的一个标准部分。人们接着又深化了生成有效的目标代码的方法，这就是最初的编译器，它们一直使用至今。

4.1.2 编译器的自动构造工具

分析器的自动生成一直是编译理论研究的一个方向。早期的程序员手工编写分析器，不但费时费力，而且编写的分析器不稳定、不易修改和移植。在自动化大潮冲击之下，越来越多的程序员抛弃了这种手工做法。

1）LEX 与 YACC

当分析问题变的好懂起来时，人们就在开发程序上花费了很大的功夫来研究这一部分的编译器的自动构造，这些程序中最著名的就是 YACC（yet another complier-compiler），由 Steve Johnson 在 1975 年为 Unix 系统编写。YACC 是 UNIX 系统的标准实用程序（Utility）。Berkeley 大学开发了和 YACC 完全兼容、但代码完全不一样的优化分析器生成工具 BYACC（Berkeley YACC），GNU 也同样推出和 YACC 兼容的优化分析器生成工具 BISON。BYACC 和 BISON 都是以源代码的形式在 Internet 上免费发行。YACC 以形式文法和语法制导翻译作为输入，经过处理产生输入文法的分析表及使用分析表驱动的语法分析和翻译源程序。语法分析是对输入文件第二次重组。输入文件是有序的字符串，词法分析对该字符串序列进行第一次重组，将字符序列转换成为单词序列；第二次重组是在第一次重组的基础上将单词序列装还成为语句。通俗的说就是，根据前面定义的所有的词组和自身的设定的一系列语法规则，看使用这些词组组成的句子是否符合规定的语法规则。由于 BYACC 在规约的时候可以完成一定的语义子程序的功能，即在规约过程中可以执行不同的动作，利用这些操作可以生成我们需要的语法树。

类似的，有穷自动机的研究也发展了另外一种称为扫描程序（scanner）生成器的工具，LEX 是其中的佼佼者。1972 年，贝尔实验室的 M.E.Lesk 和 E.Schmidt 在 Unix 操作系统上首次实现了词法分析器生成工具，称之为 LEX（LEXical Analyzer Generator）。从此，LEX 作为 Unix 的标准程序随

Unix 系统一起发行。与此同时，GNU 推出和 LEX 完全兼容的词法分析器生成工具 FLEX （Fast LEXical Analyzer Generator）。由于 GNU 的软件是以源程序的方式发行的，因此，FLEX 很容易在不同的操作系统平台下编译成可执行文件。

2）ANTLR

LEX 和 YACC 的功能非常强大，但也有不足之处，它们自动生成的程序结构比较晦涩，理解起来比较费力。

由旧金山大学的 Terence Parr 领导开发的 ANTLR（以前叫做 PCCTS，Purdue Compiler Construction Tool Set，普渡大学编译器构建工具集）是一种分析器自动生成工具，它可以接受语言的文法描述，并能产生识别这些语言的程序。而且我们可以在文法描述中插入特定的语义动作，告诉 ANTLR 怎样去创建抽象语法树(AST)和怎样产生输出。

现在 ANTLR 越来越流行，不仅因为它功能更强、容易扩展、开源，而且 ANTLR 生成的代码和使用递归下降方法（手工生成分析器的主要方法）生成的代码很相似，易于阅读理解。与之相比，另外一种著名的分析器生成工具 YACC（Yet Another Compiler-Compiler，基于 LR 分析方法）生成的程序就比较晦涩。

ANTLR 本身是使用 Java 开发的，可以生成使用 java 描述的分析器，也可以生成 C++描述的源程序（从 2.7.3 版本开始，ANTLR 开始支持 C#，将来还会支持 Python）。

4.1.3 编译器的结构

编译器是将一种语言翻译成另外一种语言的计算机程序。编译器的工作是从输入源程序开始到输出目标程序为止的整个过程。这个过程如图 4-1 所示。

图 4-1 编译过程

就其过程而言是非常复杂的，一般可以划分为六个阶段：词法分析、语法分析、语义分析、源代码优化、目标代码生成和优化等，如图 4-2 所示。

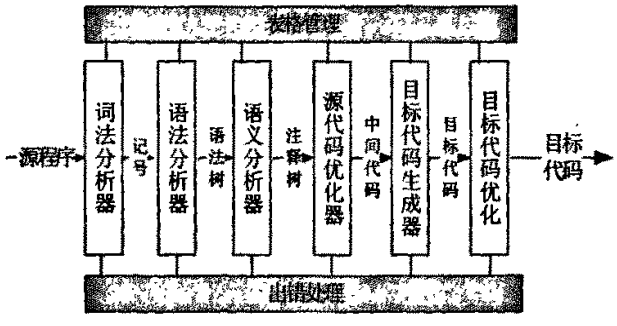

图 4-2 编译工作过程的各阶段

在一些小型、微型机中，在对编译质量要求不高的场合，可以不要源代码优化和目标代码优化两个阶段，源程序经过语法分析后直接生成目标代码。这时候语义分析可以归入目标代码生成阶段来完成。

编译程序的工作过程包含了许多阶段，对应地，编译器也包含多个模块，不同的模块执行不同的逻辑操作。一个完整的编译器一般包括：词法分析器、语法分析器、语义分析器及代码生成器等模块。下面大致介绍各个模块及其主要工作。

1） 词法分析器（Scanner）

词法分析器的功能是对源程序字符串进行扫描，按照词法规则识别出正确的单词，将其收集到编译器内部可处理的最小单位一记号（token）中，并交由语法分析器进行进一步处理。

高级语言中表达意义的最小语法单位是单词，其由字符组成。每一种高级语言都对应着各自的

一组字符集。单词有的由单个字符组成，如+、-、*和/等；有的由两个字符组成，如:=，>=等；有的有一个或多个字符组成，如常数、标志符、基本字和标准标志符等。词法分析器的主要任务是从输入的源程序字符串中逐个地把这些单词识别出来，并转化成机器比较容易使用的内码形式。

2) 语法分析器（Parser）

语法分析器的主要工作是组词成句，这与自然语言中句子的语法分析类似，它从词法分析器中获取记号形式的源代码，并完成定义程序结构的语法分析（syntax analysis）。每一种语言都有自己的语法规则，称为文法。按照这些文法，可以由单词组成语法单位，如短语、语句、过程和程序等。语法分析就是通过语法分解，确定整个输入串能否构成语法上正确的句子和程序。

语法分析定义了程序的结构元素及其关系。通常语法分析的结果表示为分析树（parse tree）或语法树（syntax tree）。

3) 语义分析程序（semantic analyzer）

程序的语义就是它的意思，有静态语义和动态语义两种。大多数程序设计语言都具有在执行之前被确定而不易由语法表示和由分析程序分析的特征，称为静态语义（static semantic），比如涉及的变量是否是已经定义的，类型的声明和使用是否一致等等。语义分析程序的主要工作就是检查程序每个结构的静态语义，将分析得到的程序结构的额外信息（称为属性）添加到语法树中。

4) 源代码优化程序（soure code optimizer）

编译器通常包括许多代码改进或优化步骤。绝大多数最早的优化步骤是在语义分析之后完成的，而此时代码改进可能只依赖于源代码。这种可能性是通过将这一操作提供为编译过程中的单独阶段指出的。每个编译器不论在已完成的优化种类方面还是在优化阶段的定位中都有很大的差异。

5) 代码生成器（code generator）

代码生成器得到中间代码（IR），并生成目标机器的代码。尽管大多数编译器直接生成目标代码，但是为了便于理解，本书用汇编语言来编写目标代码。正是在编译的这个阶段中，目标机器的特性成为了主要因素。当它存在于目标机器时，使用指令不仅是必须的而且数据的形式表示也起着重要的作用。例如，整型数据类型的变量和浮点数据类型的变量在存储器中所占的字节数或字数也很重要。

6) 目标代码优化程序（target code optimizer）

在这个阶段中，编译器尝试着改进由代码生成器生成的目标代码。这种改进包括选择编址模式以提高性能、将速度慢的指令更换成速度快的，以及删除多余的操作。

4.1.4 编译器中主要数据结构

编译器的各阶段使用的算法与相关数据结构中间相互交互，来完成编译器的工作过程。本节主要讲一些编译器中主要的数据结构。

1) 记号（token）

当扫描程序（scanner）将字符收集到一个记号中时，它通常是以符号来表示这个记号；也就是说，作为一个枚举数据类型的值来表示源程序的记号集。有时还需要保留字符串本身或由此派生出来的其它信息（例如：与标志符记号相关的名字或数字记号值）。在大多数语言中，扫描程序一次只需要生成一个记号，即单符号先行（single symbol lookahead）。在这种情况下，可以用全局变量放置记号信息；而在别的情况下，则可能会需要一个记号数组。

2) 语法树（syntax tree）

如果分析程序确实生成了语法树，它的构造通常为基于指针的标准结构，整棵树可以作为一个指向根节点的单个变量来保存。语法树中的每一个节点都是一个记录，其中的数据域存放的是由分析程序和语义分析程序所搜集的信息。

3) 符号表（symbol table）

符号表中的信息与标志符有关：函数、变量、常量和数据类型。符号表几乎与编译器所有阶段相交互：扫描程序、分析程序或是将标志符输入到表格中的语义分析程序；语义分析程序将增加数

据类型信息和其他信息：优化阶段和代码生成阶段也将利用由符号表提供的信息选出恰当的代码。因为对符号表的访问如此频繁，所以插入、删除和访问操作都必须比常规操作更有效，一般使用 hash 表来表示。

4) 常数表（literal table）

常数表的功能是存放在程序中的用到的常量和字符串，因此快速插入和查找在常数表中也十分重要。但是在其中却无需删除，这是因为它的数据一般是全局数据，而且常量或字符串在该表中只出现一次。通过允许重复使用常量和字符串，常数表对与缩小程序在存储器中的大小显得非常重要。在代码生成器中也需要常数表来构造用于常数和在目标代码文件中输入数据定义的符合地址。

5) 中间代码（intermediate code）

根据中间代码的类型和优化的类型，该代码可以是文本串的数组、临时文本文件或是结构的连接列表。对与进行复杂优化的编译器，应该特别注意选择允许简单重组表示。

4.1.5 编译器结构中的其它概念

1）分析（analysis）和综合（synthesis）

这是编译器结构划分的一个观点。在编译器中，把分析源程序及其特性的编译器操作归为编译器的分析部分，将生成目标代码所涉及的操作集合称为编译器的综合部分。分析部分包括词法分析、语法分析和语义分析，综合部分包括代码生成等。优化阶段是分析和综合兼而有之。

2）前端（front end）和后端（back end）

这是编译器结构组成划分的另一观点，将编译器分成了只依赖于源语言的操作和只依赖于目标语言的操作两部分。前一部分称为前端，另一部分称为后端。这与将其分成分析和综合两部分是类似的：扫描程序、分析程序和语义分析程序是前端，代码生成器是后端。但是一些优化分析可以依赖于目标语言，这样就是属于后端了，然而中间代码的综合却经常与目标语言无关，因此也就属于前端了。在理想情况下，编译器被严格地分成这两部分，而中间表示则作为其间的交流媒介。

这一结构对于编译器的可移植性（portability）十分重要，此时设计的编译器既能改变源代码（它涉及到重写前端），又能改变目标代码（它还涉及到重写后端）。在实际中，这是很难做到的，而且称作可移植的编译器仍旧依赖于源语言和目标语言。其部分原因是程序设计语言和机器构造的快速发展以及根本性的变化，但是有效地保持移植一个新的目标语言所需的信息或使数据结构普遍地适合改变为一个新的源语言所需的信息却十分困难。然而人们不断分离前端和后端的努力会带来更方便的可移植性。

3）遍（pass）

编译器发现，在生成代码之前多次处理整个源程序很方便，这些重复就是遍。首遍是从源程序中构造一个语法树或中间代码，在它之后的遍是由处理表示、向它增加信息、更换或生成不同的表示组成。遍可以和阶段相应，也可无关——遍中通常包含由若干个阶段。实际上，根据语言的不同，编译器可以是一遍（one pass）——所有的阶段由一遍完成，其结果是编译得很好，但代码通常效率不高。大多数带有优化的编译器都需要超过一遍：典型的安排是一遍用于扫描和分析，将另一遍用于语义分析和源代码层的优化，第三遍用于代码生成和目标层的优化。更深层的优化问题则可能需要更多的遍。

4.2 X-SQL 编译器的设计

在数据流管理系统 SEUSTREAM 的后端，用户通过提交用 X-SQL 语言编写的查询语句来对数据流进行连续查询。用户注册的一条或者多条数据流连续查询语句保存在注册查询缓冲中，然后被 X-SQL 语言编译器进行处理，生成前端预处理器的初始化状态和运行所需要的指令序列。然后硬件预处理器的初始化状态参数和指令序列经过查询输出控制器被下载到前端硬件查询处理模块中，然后开始前端预处理器开始对数据流进行处理。

4.2.1 X-SQL 编译器的工作流程

X-SQL 查询程序首先由查询语言语法分析器转化成一棵表示用户查询的语法树，然后经过查询转换转化成可以表示代数操作的查询树，接着进行查询优化生成一棵经过优化的查询树，也可以称之为查询计划树，因为它现在已经可以或者在执行时解释或者编译成可以直接执行的机器指令序列。

X-SQL 编译器的各组件之间的接口如下：

- X-SQL 词法分析器，负责从文本件中读取源程序（即用户提交的 X-SQL 查询），并产生记号表示。事实上，词法分析器是一个类，里面封装了对 X-SQL 源程序进行词法分析所需的程序逻辑。
- X-SQL 语法分析器，负责对词法分析后的 Token 流进一步分析处理，得出一棵语法树
- 语义分析程序，负责语义及类型检查。根据模式定义的语义规则对语法树进行合法性检查，形成一颗语义正确的语法树。同时将所有出现在 X-SQL 语句中的标志符的信息填入到标志符转换表中。
- 查询转换程序，根据优化规则，对查询树进行优化，产生一棵优化查询树。
- 查询分解程序，把优化查询树分解成最简单的子查询树的集合，然后将子查询树放入子查询缓冲区中，以利于子查询的共享。
- 代码生成器，负责根据子查询生成前端数据流预处理器状态初始化所需要的参数以及所需的指令序列。

4.2.2 X-SQL 编译器体系结构

我们所开发的编译器分为前端和后端两个部分。其中，前端具有很高的源程序依赖性，包括词法分析器、语法分析器、语义分析器、查询优化程序和查询分解程序；后端和目标机器相关，包括代码生成器模块。这种编译器结构的划分方法具有很大的优点，提高了编译器的可移植性，方便以后目标机器的升级或改变。X-SQL 编译器的体系结构见图 4-3。

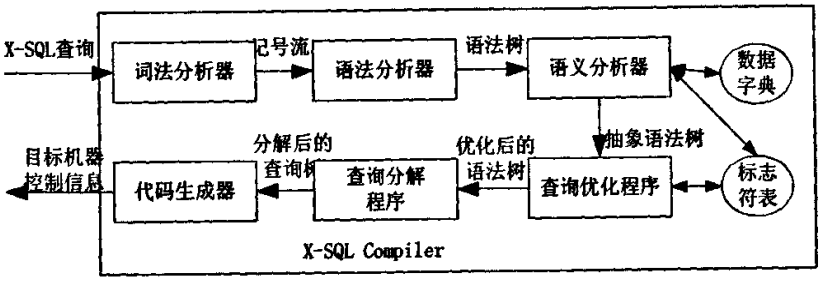

图 4-3 X-SQL 编译器的体系结构

4.3 词法及语法分析

4.3.1 词法及语法分析的设计

词法分析和语法分析是编译器对源程序进行处理的最初的两个阶段。词法分析阶段识别低级的对象，如数字、操作符和专门的符合等；语法分析阶段识别高级的对象，如程序设计语言中的语句等。对于词法及语法分析器的设计，可以手工设计，也可以采用编译器自动构造工具来进行自动生成。手工编写分析器，不但费时费力，而且编写的分析器不稳定、不易修改和移植，所以我们决定采用第二种方法。现在流行的编译器自动构造工具比较著名的有 LEX、YACC 和 ANTLR 等。在项目中我们采用 ANTLR 作为我们的 X-SQL 编译器的词法和语法自动生成工具。这大大简化了语法分

析程序和词法分析程序的开发。

ANTLR 可以接受语言的文法描述，并能产生识别这些语言的程序。而且我们可以在文法描述中插入特定的语义动作，告诉 ANTLR 怎样去创建抽象语法树(AST)和怎样产生输出。在使用 ANTLR 进行工作时，文法文件是 ANTLR 的核心，是程序员和 ANTLR 进行交流的接口。文法就是语言识别的规则。它是 ANTLR 生成程序的依据。文法文件的编写基本是面向被解决的问题的。程序员只需要集中精力思考解决问题的逻辑，而不是羁绊于某种程序设计语言的实现细节，因此降低了出现错误的可能性。

4.3.1.1 文法文件的语法简介

此处只是简单地介绍一下文法文件的语法，具体内容可以参阅 ANTLR 的相关文档。

文法文件一般包括 header 块、options 块、文法分析器类（parser）及规则定义、词法分扫描器类（Scanner）及 token 定义。其中最为重要的是规则和 token 的定义。

规则的定义形式为扩展巴科斯范式（EBNF），包括规则名、规则体、一个用作结束标志的分号和异常处理部分（可省略）。例如如下的 token 定义就表示一个十进制的整数：

```
NUMBER:
(1..9)(0..9)*
;
```

其意义是：数字（NUM）的第一字符是‘1’到‘9’中的一个字符，后面是 0 个或多个‘0’到‘9’之间的字符。

规则的名字必须是小写字母开始，而 token 的名字则必须是大写字母开始。

4.3.1.2 设定 ANTLR 生成的语言

ANTLR 有很多选项，可以通过在文法文件中的 options 块中进行设置，其中包括 ANTLR 最终生成的语言。如果要生成 C++描述的分析器程序，就要如下设定：

```
options
{
language=Cpp;
// Other options
}
```

language 选项的默认值是“Java”。language 设为“Csharp”将生成 C#描述的分析器程序。

4.3.1.3 编写文法文件

根据 X-SQL 语法，写出 X-SQL 的文法文件 X-SQL.g。以此作为 ANTLR 的输入文件。运行 ANTLR 程序，即可以得到

全部 ANTLR 语法都是 Lexer 语法解析程序和语法树解析程序的子类。首先创建一个文法解析程序的子类。在类的声明之后，可以使用扩展巴柯斯范式符号指定规则。

4.3.2 X-SQL 词法分析

用户提交的 X-SQL 语句在进行词法分析之前是一个由字符串构成的文本。例如查询语句：

SELECT * FROM R,S,T WHERE (R.c=S.e or S.e=T.f) and R.a=S.d WINDOW 300

如果不按照 X-SQL 规定的语义去解释，那么它仅仅是一串字符流，没有任何意义。通过词法分析和语法分析，这个字符串流就会变成一棵 X-SQL 语法树（Sytax Tree），也称为分析树（Parse Tree）。而词法分析的功能就是根据一定的规则对这串无结构的字符串流进行识别，把它分解为一个个的记号（token）。

生成的记号流就是语法分析器的输入。每个记号都包含两个属性：记号名和记号值。在 X-SQL 里出现的记号有以下几种：

- X-SQL 关键字（KeyWords）
- X-SQL 运算符（Operator）
- 标识符（Identifier）
- 数字
- 字符串等

如果记号是一个关键字，记号名就是关键字的种类。如果记号是一个标识符，记号名代表标识符的种类。

4.3.3 X-SQL 语法分析器

语法分析（syntax analysis）的任务是确定程序结构。程序设计语言的语法通常是由上下文无关（context-free grammar）的文法规则（grammar rule）给出，其方式同扫描程序识别的由正则表达式提供的记号的词法结构相类似。上下文无关文法的确利用了与正则表达式中极为类似的命名惯例和运算。二者的主要区别在于上下文无关文法的规则是递归的。例如一般来说，表达式结构中应允许嵌套其它的表达式，而在正则表达式中却不能这样做。这个区别造成的影响很大。由上下文无关文法识别的结构类比由正则表达式识别的结构大大增多了。用作识别这些结构的算法也与扫描算法差别很大，这是因为它们必须使用递归调用或是显式管理的分析栈。用作表示语言语义结构的数据结构现在也必须是递归的，而不再是线性的（如同用于词法和记号中的一样）了。经常使用的基本结构是一类树，称作分析树（parse tree）或语法树（syntax tree）。

X-SQL 语法分析器的输入是词法分析程序的输出———一个 Token 流。输出是一棵抽象语法树，它是内存中的数据结构。X-SQL 语法分析器可以分为两个子模块：语法规则规约（Syntax Rule Drivation）模块和语法树（Parse Tree Generation）生成模块，如图 4-4。

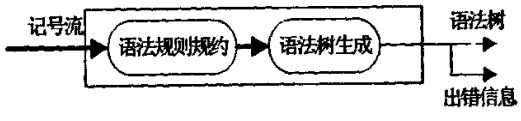

图 4-4 X-SQL 语法分析器

4.3.3.1 语法规约（Syntax Tree）

语法规约模块的输入是词法分析器的输出的记号流。它从 token 序列中抽取相应的语法规则号、token 值和 token 名字，用以生成相应的语法树。

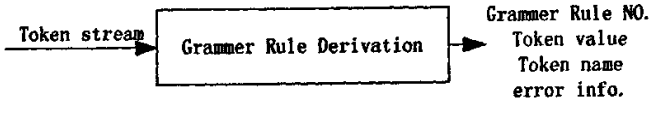

图 4-5 语法规约示意图

4.3.3.2 语法树（Syntax Tree）

语法树是一个作了标记的树，其中内部的节点由非终结符标出，树叶节点由终结符标出，每个内部节点的子节点都表示推导的一个步骤中的相关非终结符的替换。

语法树是语法分析器的输出，也是编译器对源程序进一步进行处理的输入。它是它是 X-SQL 语句的第一个有结构的表示。语法树是由节点(node)和边(edge)组成的树结构，一个节点就是一个 Java 对象，而树枝是 Java 中的一个引用（对象句柄）。语法树的节点可以分为两种，叶子节点和非叶子节点。叶子节点与记号相对应，而非叶子节点对应于一个语法规则的记号序列。

语法树的节点可以表示下面的类型：原子字面量（Atomic Literal）；X-SQL 关键字；变量名字（Variable name）；约束变量名字（Bound variable name）；表达式（Expression），包括数学表达式（arithmetic expression）、字符串表达式（String expression）、比较表达式（Comparison expression）、布尔表达式(Boolean expression)和集合表达式(Set expression)；选择查询语句(SELECT statement)。

语法树的节点数据结构如下：

```
class AST{
int ttype;
string text;
AST down;
AST right;
}
```

其中 ttype 是该 X-Sql 记号的类型（Token Type），text 是该 Token 的值，down 是该 AST 节点的左面第一个孩子节点，right 是其右边第一个兄弟节点。

4.3.3.3 语法树的例子

对于 X-SQL 查询：

SELECT * FROM R,S WHERE R.a=S.d WINDOW 100

进行词法分析和语法分析之后会得到如图所示的语法树。

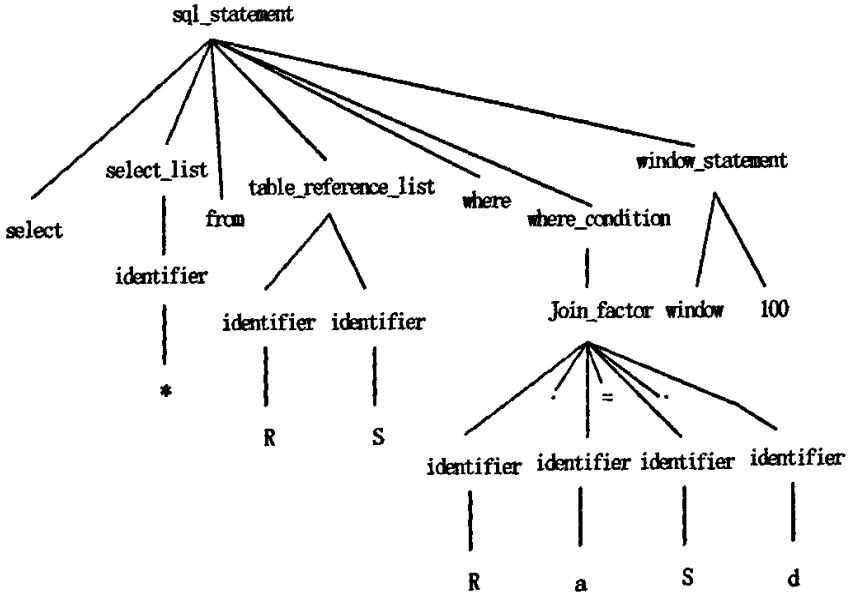

图 4-6 生成的语法树示意图

4.4 语义分析程序

4.4.1 语义分析程序的功能

如果想得到预期的查询结果，首先要保证输入的 X-SQL 语句符合数据流的模式所定义的语义。X-SQL 语义分析程序是检查 X-SQL 查询语句的语义是否正确，包括变量定义的检查，类型匹配的检查。它主要包括两方面的功能：

1）对每一个 X-SQL 表达式，检查它的类型是否正确.这个过程需要读取存储在数据字典中的数据库模式信息以取得标识符的类型信息或根据数据字典中的信息推导出标识符的类型信息，然后根据 X-SQL 表达式的操作类型，判断操作数的类型是否正确。

2）对每一个 X-SQL 表达式，由它的子表达式类型来推导出它的类型。这个过程需要根据 X-SQL 表达式的操作类型和子表达式的类型来推导出父表达式的类型。

用户输入的 X-SQL 查询经过词法分析和语法分析之后得到的语法树是语义分析程序的输入。经过语义检查、类型检查、类型推导等工作之后，将会输出一棵语义和类型正确、类型完备的语法树。所谓类型完备就是语法树中所有的标识符的类型信息都已经被填充到标识符表中。图给出语义及类型检查的示意图。

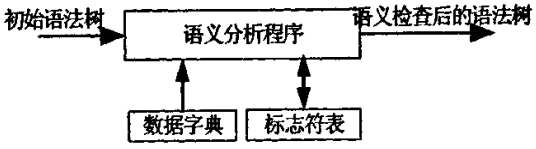

图 4-7 语义及类型检查示意图

4.4.2 语义检查

为了保证输入的 X-SQL 语句符合数据流的模式所定义的语义。首先需要对语法分析之后得到的语法树进行语义检查。语义检查主要是检查 X-SQL 语句中所用到的流或者关系及其属性的引用正确性。例如对于 X-SQL 语句：

SELECT * FROM R,S WHERE R.a=S.d WINDOW 100

其中有对数据流 R 的属性 a 和数据流 S 的属性 d 的引用，语义检查工作就是查询数据字典，看是否存在流 R 及其属性 a 和流 S 及其属性 d。如果存在则语义检查通过，否则语义检查程序抛出语义错误。

4.4.3 类型检查

经过语义检查之后，语义分析程序的工作还没有做完，下一步就是类型检查。类型检查就是根据操作的种类来检查操作数是否符合相关操作的类型要求。例如对于 X-SQL 语句：

SELECT * FROM R WHERE R.a>=0.8 WINDOW 100

如果 *R.a* 是整型、浮点型等可以和浮点数相比较的类型，则 WHERE 条件中的表达式 *R.a>=0.8* 是有意义的，否则语法分析程序会抛出一个类型检查错误。

4.4.4 类型推导

前面提到的语义及类型检查的一个重要内容就是根据操作的操作数类型来推导出该操作的结果类型。类型推导的工作就是根据操作的种类和操作数的类型来推导出该操作的结果类型。例如对于查询语句的算术表达式：

X+Y

如果 X 和 Y 的类型都是整型，那么表达式 $X+Y$ 的类型也是整型，如果 X 是整型，而 Y 是浮点型的话，那么表达式 $X+Y$ 的类型就是浮点型。

4.5 查询优化

4.5.1 查询优化概述

用户提交的查询经过编译器的词法分析、语法分析和语义分析之后，得到了一棵语义正确的查询语法树。此时的查询语法树已经可以直接作为依据生成目标代码了，但是目标机器的执行效率可能不高。为了进一步的提高查询效率，我们需要对查询进行优化。目前我们编译器的优化主要包括以下几个方面：

- 谓词化简

  谓词化简指的是应用逻辑运算的规则，将谓词表达式等价转换为最简单的形式。

- 谓词规范化

  谓词规范化是指将 X-SQL 查询语句中的谓词表达式转换为主析取范式形式。谓词规范化便于查询的分解，经过谓词规范化以后一棵初始的查询树可以转换为多棵子查询树（这些子查询树的谓词中不含有 OR 逻辑运算符），而这棵查询树对应的查询结果也就是这些子查询树查询结果的并集。

- 查询分解

  查询语法树经过谓词规范化后，其谓词表达式是一个主析取范式。这样，我们可以将一个 X-SQL 查询语句对应的一棵查询树分解为若干棵子查询树，这样有利于查询结果的共享。

4.5.2 谓词化简

谓词化简是指应用逻辑运算的规则，将谓词表达式等价转换为最简单的形式。谓词化简依据的逻辑运算的规则主要有（其中 A 和 B 是逻辑表达式）：

(1) A OR A <=> A
(2) A AND A <=> A
(3) A OR NOT(A) <=> TRUE
(4) A AND NOT(A) <=> FLASE
(5) A AND (A OR B) <=> A
(6) A OR (A AND B) <=> A
(7) A OR FALSE <=> A
(8) A AND TRUE <=> A
(9) A OR TRUE <=>TRUE
(10)A AND FALSE <=> FALSE
(11)NOT (A AND B) <=> NOT (A) OR NOT (B)
(12)NOT (A OR B) <=> NOT (A) AND NOT (B)
(13)NOT (NOT(A)) <=> A

谓词化简的算法对谓词表达式树结构进行字底向上的递归处理，算法描述如下：

算法 4.1 谓词化简算法

(1) 输入：一个语法树节点 AST 类型对象。
(2) 输出：void。
(3) 算法描述：
  (i) 如果输入参数为空，则返回。
  (ii)以输入参数的左子树为参数递归调用算法处理左子树。

(iii)以输入参数的右子数为参数递归调用算法处理右子树。

(iv) 若谓词表达式的形式为 A OR A，或者 A AND A，或者 A AND (A OR B) ，或者 A OR (A AND B)，或者 A OR FALSE，或者 A AND TRUE，或者 NOT (NOT(A))，则将其转化成为 A。

(v) 若谓词表达式的形式为 A OR NOT(A)，或者 A OR TRUE，则将其转化成为 TRUE。

(vi)若谓词表达式的形式为 A AND NOT(A)，或者 A AND FALSE，则将其转化成为 FALSE。

(vii) 若谓词表达式的形式为 NOT (A AND B)，则将其转化成为 NOT (A) OR NOT (B)。

(viii) 若谓词表达式的形式为)NOT (A OR B)，则将其转化成为 NOT (A) AND NOT (B)。

(4)算法结束。

4.5.3 谓词规范化

X-SQL 查询的语法树中谓词表达式是以树的形式进行存储的，三个逻辑运算操作符（AND、OR 和 NOT）具有不同的优先级。我们使用递归的算法来处理三种形式的逻辑谓词表达式：A AND (B OR C)；(A OR B) AND C；(A OR B) AND (C OR D)。

下面给出谓词规范化的算法。

算法 4.2 谓词规范化算法

(1) 输入：一个语法树节点 AST 类型对象。

(2) 输出：void。

(3) 算法描述：

(i) 如果输入参数为空，则返回。

(ii)以输入参数的左子树为参数递归调用算法处理左子树。

(iii)以输入参数的右子数为参数递归调用算法处理右子树。

(iv) 若谓词表达式的形式为(A ORB)AND C，则将其转化成为(A AND C)OR(B AND C)。

(v) 若谓词表达式的形式为 A AND (B OR C)，则将其转化成为(A ANDB)OR(A AND C)。

(vi)若谓词表达式的形式为(A OR B)AND(C OR D)则将其转化成为(A AND C) OR (A AND D)OR(B AND C)OR(B AND D)

(4)算法结束。

例如对于 X-SQL 查询：

SELECT * FROM R,S,T WHERE (R.c=S.e or S.e=T.f) and R.a=S.d WINDOW 300

经过谓词规范化之后变形为：

SELECT * FROM R,S,T WHERE (R.c=S.e and R.a=S.d) or (R.c=S.e and S.e=T.f) WINDOW 300

4.5.4 查询分解

查询语法树经过谓词规范化后，其谓词表达式是一个主析取范式。这样，我们可以将一个 X-SQL 查询语句对应的一棵查询树分解为若干棵子查询树。查询分解的目的是利于查询结果的共享，提高查询处理的效率。

对于上节的例子，进行查询分解之后分解为如下两条子查询的并集。

子查询 1：SELECT * FROM R,S,T WHERE R.c=S.e and R.a=S.d WINDOW 300

子查询 2 SELECT * FROM R,S,T WHERE R.c=S.e and S.e=T.f WINDOW 300

4.6 查询代码的生成

用户提交的查询语句经过编译器的词法分析、语法分析、语义分析和查询优化之后，得到了一棵语义正确的语法树，至此，编译器的工作也就告一段落了。以上的处理步骤都是与源程序有关，

称之为编译器的前端，接下来的工作主要是根据编译器前端生成的语法树，生成目标机器平台上的代码，这一部分称之为编译器的后端。

我们的目标机器是在第二章中介绍的基于 FPGA 的数据流预处理器。采用软硬混成的方式对数据流进行查询处理，目前该预处理器可以实现多流多任务并发连接查询。编译器后端的工作主要涉及数据流预处理器中的资源分配、各个模块状态的初始化和查询指令序列的生成。具体体系结构见图。

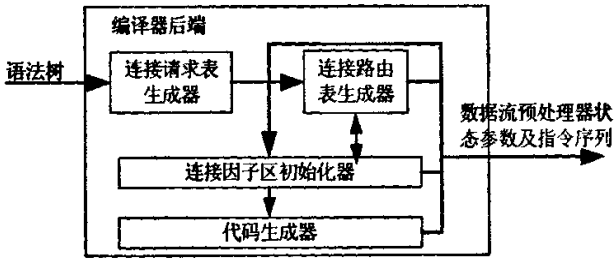

图 4-8 编译器后端结构图

下面我们用一组查询实例来详细说明代码生成各模块的工作流程。设三流 R(a,b,c)、S(d,e)、T(f,g)，同时注册有 8 个并发连接请求：

查询 0: SELECT * FROM R,S WHERE R.a=S.d WINDOW 100;

查询 1: SELECT * FROM R,S,T WHERE R.a=S.d and S.e=T.f WINDOW 200

查询 2: SELECT * FROM S,T WHERE S.e=T.f WINDOW 100;

查询 3: SELECT * FROM R,S WHERE R.b>S.e WINDOW 200;

查询 4: SELECT * FROM R,S WHERE R.a=S.d or R.c=S.e WINDOW 100;

查询 5: SELECT * FROM R,S WHERE R.a=S.d and R.c=S.e WINDOW 200;

查询 6: SELECT * FROM R,S,T WHERE R.a=S.d or S.e=T.f WINDOW 100;

查询 7: SELECT * FROM R,S,T WHERE (R.c=S.e or S.e=T.f) and R.a=S.d WINDOW 300;

经过词法分析、语法分析、语义分析和查询优化之后，这一组查询被划分为如下的一组子查询：

子查询 0：（父查询：0,4） SELECT * FROM R,S WHERE R.a=S.d WINDOW 100

子查询 1：（父查询：1） SELECT * FROM R, S,T WHERE R.a=S.d and S.e=T.f WINDOW 200

子查询 2：（父查询：2）SELECT * FROM S,T WHERE S.e=T.f WINDOW 100

子查询 3：（父查询：3） SELECT * FROM R,S WHERE R.b >S.e WINDOW 200

子查询 4：（父查询：4） SELECT * FROM R,S WHERE R.c=S.e WINDOW 100

子查询 5：（父查询：5） SELECT * FROM R,S WHERE R.a=S.d and R.c=S.e WINDOW 200

子查询 6：（父查询：6） SELECT * FROM R,S,T WHERE R.a=S.d WINDOW 100

子查询 7：（父查询：6） SELECT * FROM R,S,T WHERE S.e=T.f WINDOW 100

子查询 8：（父查询：7） SELECT * FROM R,S, T WHERE R.c=S.e and R.a= S.d WINDOW 300

子查询 9：（父查询：7） SELECT * FROM R,S, T WHERE S.e=T.f and R.a=S.d WINDOW 300

4.6.1 连接请求表的生成

连接请求表是对每条子查询的连接因子和窗口进行统计和存储。连接因子是不能再分解的最基本连接条件，简称因子，形如 $R_1.A\ op\ R_2.B$，其中 $R_1.A$、$R_2.B$ 分别为数据流 R_1、R_2 的属性，$op \in \{=、\neq、\leqslant、<、\geqslant、>=\}$。提取所有不同连接因子，建立连接请求表如下。

表 4.1 连接请求表实例

查询	流	连接因子	窗口
0	R,S	①	100
1	R,S,T	①②	200
2	S,T	②	100
3	R,S	③	200
4	R,S	④	100
5	R,S	①④	200
6	R,S,T	①	100
7	R,S,T	②	100
8	R,S,T	①④	300
9	R,S,T	①②	300

注：①R.a=S.d,②S.e=T.f,③R.b>S.e,④R.c=S.e 是连接因子。

4.6.2 连接区及连接路由表初始化

从分解后的查询子树中提取出连接因子建立连接请求表以后，下一步的工作是（1）根据连接请求表来设置连接因子区，填入相关因子；（2）根据连接因子和连接请求表来设置因子组合区的连接前标记、组合及连接后路由标记；(3)设置窗口缓冲区大小。将缓冲区大小设成中所有因子窗口的最大值；(4)根据连接请求表构造连接路由表。(5)按连接请求表设置输出缓冲区的窗口值。

流连接区的初始化过程如下：

1）根据连接请求表来设置连接因子区，填入相关因子；

2）根据连接因子和连接请求表来设置因子组合区的连接前标记、组合及连接后路由标记；

3）将缓冲区大小设成中所有因子窗口的最大值；

在例子中的查询输入下，各流对应的连接区状态参数如图 4-9：

Stream R's Join Area

window size = 300

Join Factor Area

0: R.a=S.d
2: R.b>S.e
3: R.c=S.e

Factor combination Area

R	0	4
S	0	3
S	0	5
7	0	8
S	2	13
S	3	15
S	0,3	17
S	0	19
21	0	22
S		25
27		28
S	0,1	31
33	0,1	34
S	0	37
39	0	40

(a) 流R对应的连接区

Stream S's Join Area

window size = 300

Join Factor Area

0: R.a=S.d
1: S.e=T.f
2: R.b>S.e
3: R.c=S.e

Factor combination Area

R	0	4
T	1	7
9	0,1	10
T	1	11
R	2	14
R	3	16
R	0,3	18
T		21
23	0	24
T	0	27
29	0	30
T		33
35	0,1	36
T	1	39
41	0,1	42

(b) 流S对应的连接区

Stream T's Join Area

window size = 300

Join Factor Area

1: S.e=T.f

Factor combination Area

5	1	6
R		9
S	1	12
19		20
R		23
25	0	26
R		29
31		32
R		35
37	1	38
R		41

(c) 流T对应的连接区

图 4-9 R、S 和 T 流对应的连接区

在连接区的初始化过程中，连接路由表的初始化也同步进行，二者相互合作，共同完成初始化过程。连接路由表的构造规则如下：

1）原始元组要插入到相应的窗口缓冲区，其它置 0；

2）连接后元组改变路由标记，该标记同时存在于连接路由表；

3）需要探测的元组设置相应连接区标记，否则置 0；

4）元组符合某查询请求则设置其编号，否则置-1。

在例子中，根据查询生成的连接路由表如下：

路由标记	插入	探测	查询
R	R	S	-1
S	S	R,T	-1
T	T	S	-1
3	0	0	0,4
4	0	0	0,4
5	0	T	-1
6	0	0	1
7	0	R	-1
8	0	0	1
9	0	S	-1
10	0	0	1
11	0	0	2
12	0	0	2
13	0	0	3
14	0	0	3
15	0	0	4
16	0	0	4
17	0	0	5
18	0	0	5
19	0	T	-1
20	0	0	6
21	0	R	-1
22	0	0	6
23	0	S	-1
24	0	0	6
25	0	T	-1
26	0	0	6
27	0	R	-1
28	0	0	6
29	0	S	-1
30	0	0	6
31	0	T	-1
32	0	0	7
33	0	R	-1
34	0	0	7
35	0	S	-1
36	0	0	7
37	0	T	-1
38	0	0	7
39	0	R	-1
40	0	0	7
41	0	S	-1
42	0	0	7

表4.2 连接路由表实例

4.6.3 目标指令序列的生成

连接区是硬件预处理器中用来实现不同数据流执行连接的部件单元。在 M3Join 算法中，每一个流均有一个连接区与之对应。连接区由窗口缓冲区、连接因子区、因子组合区和探测元组缓冲区组成。在 SEUSTREAM 中的预处理器的硬件实现中，每个连接区对应一个连接单元*(Join Unit)，在连接单元中有一个叫做 IEU（Instruction Execution Unit）的部件单元。硬件预处理器中有 7 条专用指令，通过将算法翻译成专用指令序列(也称为程序)来执行这个算法，而不是直接在硬件上运行。这些指令序列就运行在连接区的一组 IEU 硬件部件中。当注册的查询改变时，硬件不需要任何改动，而只需要改变预处理器中的状态参数和 IEU 中的指令序列即可以达到目的。指令格式参见第二章。我们使用这些指令来完成算法 2.3 中的第 5 到 16 步、即完成一个探测元组和一个窗口元组的比较和连接操作。

下面我们以流 S 对应的连接单元中 IEU 指令序列的生成为例来说明目标指令序列的生成。IEU 中的指令序列生成分为以下五个步骤：

1）首先查找因子连接区，将其中的连接因子转化为比较操作（CMP）；

2）加上跳转指令 JMPLI，这一条指令的作用是在上一步生成的的比较操作的结果位全为零的情况下，程序跳转到最后一条指令；

3）将有多条连接因子的因子比较结果用 AND 连接起来；

4）根据该流所涉及的查询和连接路由表的情况，使用 JMCPT、JOIN 和 JMP 三条指令组合，完成因子比较后连接元组的路由；

5）最后一条指令是一个空操作：NOP；

在我们的运行示例中，根据每条流的连接区状态，生成硬件预处理器中各自对应的 IEU 指令序列。表 4.3 中列出的是流 S 对应的连接区中 IEU 指令序列。

表4.3 流S对应的IEU中的指令序列

Instructions in S's IEU

Instrucitons Adress	Instrucitons	Machine Code
0	CMP 00, 0000, 001, 000000	0011, 0001, 0000, 1000, 0000
1	CMP 01, 1000, 001, 000001	0011, 0110, 0000, 1000, 0010
2	CMP 01, 0001, 010, 000010	0011, 0100, 0101, 0000, 0100
3	CMP 01, 0010, 001, 000011	0011, 0100, 1000, 1000, 0110
4	JMPLI	0001, 0000, 0000, 0000, 0000
5	OR 000000, 000001, 010000	1100, 0000, 0000, 0101, 0000
6	OR 000000, 000011, 010001	1100, 0000, 0000, 1101, 0000
7	JMPCT 00000000, 0000001100	0100, 0000, 0000, 0000, 1100
8	JOIN 000000, 00000100	0010, 0000, 0000, 0001, 0000
9	JOIN 000010, 00001110	0010, 0000, 1000, 0011, 1000
10	JOIN 000011, 00010000	0010, 0000, 1100, 0100, 0000
11	JOIN 010001, 00010010	0010, 0100, 0100, 0100, 1000
12	JMPCT 00000010, 0000010010	0100, 0000, 1000, 0001, 0010
13	JOIN 000001, 00000111	0010, 0000, 0100, 0001, 1100
14	JOIN 000001, 00001011	0010, 0000, 0100, 0010, 1100
15	JOIN 000100, 00010101	0010, 0001, 0000, 0101, 0100
16	JOIN 000000, 00011011	0010, 0000, 0000, 0110, 1100
17	JOIN 000100, 00100001	0010, 0001, 0000, 1000, 0100
18	JOIN 000001, 00100111	0010, 0000, 0100, 1001, 1100
19	JMPCT 00001001, 0000001101	0100, 0010, 0100, 0000, 1101
20	JOIN 010000, 00001010	0010, 0100, 0000, 0010, 1000
21	JMPCT 00010111, 0000010111	0100, 0101, 1100, 0001, 0111
22	JOIN 000000, 00011000	0010, 0000, 0000, 0110, 0000
23	JMPCT 00011101, 0000011001	0100, 0111, 0100, 0001, 1001
24	JOIN 000000, 00011110	0010, 0000, 0000, 0111, 1000
25	JMPCT 00001001, 0000001101	0100, 0010, 0100, 0000, 1101
26	JOIN 010000, 00010100	0010, 0100, 0000, 0101, 0000
27	JMPCT 00101001, 0000011101	0100, 1010, 0100, 0001, 1101
28	JOIN 010000, 00101010	0010, 0100, 0000, 1010, 1000
29	NOP	0000, 0000, 0000, 0000, 0000

至此，查询编译器的工作就完成了。在 SEUSTREAM 中，编译器将从用户提交的查询转化为数据流预处理器所需要的状态参数及指令序列，然后交与查询输出控制器下载到数据流预处理器中去。

4.7 本章小结

本章详细介绍了 SEUSTREAM 中的 X-SQL 查询编译器的设计过程。查询处理在数据流管理系统中是一个非常重要的部分。用户提交的查询语句首先由提交到数据流管理系统查询缓冲区，然后经过查询编译器处理，生成查询计划，将得到的目标机器指令序列及资源分配初始化参数下载到目标处理器中去执行。

第五章 总结

数据流模型的出现对传统数据库提出了有力的挑战。数据流的流动性和无限性与计算资源的有限性之间的矛盾，使得如何提高数据处理速度成为设计数据流管理系统的关键。现在的大多数据流管理系统都是基于软件实现的，数据流速达到一定程度的时候，处理速度就会成为系统性能瓶颈。为了能够快速及时地处理高速数据流，我们提出了采用硬件预处理的并行数据流管理体系结构。

本文首先介绍了基于硬件预处理器的数据流管理系统SEUSTREAM和基于FPGA的数据流预处理器的体系结构。然后给出了一种支持连续查询的数据流查询语言X-SQL，并结合查询实例给出了该语言的具体用法。然后以数据流预处理器为目标机器设计了数据流连续查询语言 X-SQL 的编译器，该编译器对用户提交的查询命令进行词法、语法及语义分析，并进行查询优化，得到查询抽象语法树，并针对数据流预处理器对抽象语法树进行处理，生成其所需的控制信息。

由于目前基于硬件预处理器的数据流管理系统 SEUSTREAM 还没有完全实现，基于 FPGA 的数据流预处理器目前只能支持多数据流的连接查询操作，对于投影、选择和聚集等操作将在下一步的工作中实现。所以未来我们的 X-SQL 编译器还有很多的工作要做。

致谢

行文至此本文也要结束了，随之而去的还有我两年多的研究生生活。回首两年多的求学生涯，首先要感谢的是我的导师徐宏炳教授。徐老师学识渊博、务实恶虚、平易近人，三年来对我的悉心教导使我受益无穷，他将是我一生学习和敬仰的长者。同时还要感谢董逸生教授、徐立臻教授两位老师，在研究生求学阶段给予的指导和帮助。

感谢实验室里的师兄们——王永利、钱江波、刘学军、梁作鹏、杨雪梅，谢谢他们的关心和照顾，他们不但是我学习上的师兄，也是我生活中的榜样。

还有我们的金智六人行——张超、雷宵、靳岩、尹呈和吴成涛。共同的实习生活让我的研究生生活充满色彩。

感谢东南大学计算机系数据库组给我提供了良好的学习研究环境。感谢实验室同学：孙超、於媛、李凌燕、秦文艳、孙江萍、刘杰、孟芝佳、沈洁、周长春、丁海龙和吴晓伟等共同营造的融洽氛围，它让我愉快的度过了难忘的研究生生活。

最后感谢的是我的父母家人，是他们给予我一贯的关心、照顾和理解，为我创造条件，投入到学习和研究当中。在这近三年的学习和研究期间，家人一直给我以精神上的鼓励、物质上的支持和学习上的帮助，离开了他们的支持，本文的一切都无从谈起。

谨以此文献给我最亲爱的爸爸妈妈！

参考文献

[1] B. Babcock, S. Babu, M. Datar,et al. Models and Issues in Data Stream Systems [J]. In Proc. of the 2002 ACM Symp, In Principles of Database Systems, June2002, pages1-16

[2] Lukasz G. and M. Tamer Ozsu. Issues in Data Stream Management. SIGMOD Record, Vol. 32, No. 2, June 2003 5-14

[3] A. Arasu, M.Chemiack, E Galvez, et al. Linear Road: A Stream Data Management Benchmark [J], Proceedings of the 30 VLDB Conference, Toronto, Canada,2004

[4] C. Cranor, Y. Gao, T. Johnson, and et al. GigaScope: High Performance Network Monitoring with an SQL Interface. In Proc. ACM Int. Conf. on Management of Data, 2002, page 623.

[5] Y. Zhu, D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In Proc. Int. Conf. on Very Large Data Bases, 2002, p.358-369.

[6] C. Cortes, K. Fisher, D. Pregibon, and et al. Hancock: a language for extracting signatures from data streams. In ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pages 9-17, 2000.

[7] Wang, H. and Zaniolo, C. ATLaS: A Native Extension of SQL for Data Mining and Stream Computations, UCLA CS Dept., 2002.

[8] Naughton, J., DeWitt, D., Maier, D.. The Niagara Internet Query System, University of Wisconsin, 2002.

[9] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. In Proc. Int. Conf. on Mobile Data Management, 2001, pages 3-14.

[10] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In Proceedings of 18th International Conference on Data Engineering, 2002

[11] D.Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data, pages 321–330, June 1992.

[12] M. Sullivan and A. Heybey. Tribeca: A System for Managing Large Databases of Network Traffic. In Proc. of the USENIX Annual Technical Conf., New Orleans, LA, 1998.

[13] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pages 49–60, Madison, Wisconsin, May 2002.

[14] M. Shah, J. Hellerstein, S. Chandrasekaran, and M. Franklin.Flux: An Adaptive Partitioning Operator for Continuous Query Systems[R]. Technical Report CS-02-1205, U.C. Berke-ley, 2002.

[15] A. Arasu, S. Babu, J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations[R]. Nov.2002. http://dbpubs.stanford.edu:8090/pub/2002-57.

[16] R. Motwani, J. Widom, A. Arasu, and et al. Query Processing, Approximation, and Resource Management in a Data Stream Management System. In Proc. Conf. on Innovative Data Syst. Res, 2003, pp. 245-256.

[17] D. Carney, U. Cetinternel, M. Cherniack, and et al. Monitoring streams-A New Class of Data Management Applications. In Proc. Int. Conf. on Very Large Data Bases, 2002, pp. 215-226.

[18] M. Cherniack, H. Balakrishnan, M. Balazinska, and et al. Scalable Distributed Stream Processing. In CIDR, Asilomar, CA. Jan. 2003.

[19] D. Carney, U. Cetintemel, A. Rasin, and et al. Operator Scheduling in a Data Stream Manager. In proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03), Berlin, Germany, September 2003.

[20] Chandrasekaran, S. and Franklin, M., Streaming Queries over Streaming Data. in International

Conference on Very Large Databases (VLDB), (Hong Kong, 2002).

[21] S. Chandrasekaran, O. Cooper, A. Deshpande, and et al. TelegraphCQ: Continuous Dataow Processing for an Uncertain World. In Proc. Conf. on Innovative Data Syst. Res, 2003, pp. 269-280.

[22] A. Arasu, S. Babu, J. Widom. The CQL Continuous Query Language- Semantic Foundations and Query Execution. http://www-db.stanford.edu/~arvind/papers/cql-vldbj.pdf

[23] L.Qiao, D. Agrawal, A. ElAbbadi. Supporting Sliding Window Queries for Continuous Data Streams. Technical Report, University of California, Santa Barbara, 2003.

[24] M. Hammad, W. Aref, A. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-network Databases. In Proc. of the 2003 Int LConf. on Scientific and Statistical Database Management. June 2003:75 - 84.

[25] Wang Yongli, Xu Hongbing, Dong Yisheng, Liu Xuejun, Qian Jiangbo. Data Partitioning over Data Streams Based on Change-aware Sampling, IEEE International Conference on e-Business Engineering (IEEE ICEBE 2005), 18-20 October, 2005, Beijing, China.

[26]Liu Xuejun, Xu Hongbing, Dong Yisheng, Qian Jiangbo, Wang Yongli. Online parallel aggregation for power data streams. Automatic Control and Computer Sciences (AC&CS), Accepted, to appear.

[27]Liu Xuejun, Xu Hongbing, Dong Yisheng, Qian Jiangbo, Wang Yongli. Dynamically Mining Frequent Patterns over Online Data Streams. Third International Symposium on Parallel and Distributed Processing and Applications (ISPA2005), Nanjing, China. LNCS 3758, pp. 645–654 .

[28]Wang Yongli, Xu Hongbing, Dong Yisheng, Liu Xuejun, Qian Jiangbo. A data partitioning method based on sampling for power load streams. Journal of Southeast University, 2005, 3.

[29]Wang Yongli, Xu Hongbing, Xu Lizhen, Liu Xuejun, Qian Jiangbo. A granularity-aware parallel aggregation method for power data streams, Accepted by the 2nd Conference on Web Information System and Application (WISA 2005), September 2005, Shenyang, China. Published in Journal of Wuhan University.

[30]Wang Yongli, Xu Hongbing, Dong Yisheng, Liu Xuejun, Qian Jiangbo. APForecast: An Adaptive Forecasting Method for Data Streams. Accepted by Special Session on Knowledge Discovery in Data Streams In Conjunction with the 9th International Conference on Knowledge-based Intelligent Information & Engineering Systems 2005, 14-16 September, 2005, Melbourne, Australia. LNCS 3682. 957-963.

[31]Qian Jiangbo, Xu Hongbing, Dong Yisheng, Liu Xuejun, Wang Yongli. FPGA Acceleration window Joins over Multiple Data Stream, Journal of Circuits, Systems, and Computers, Vol. 14, No. 4 (2005) 813-830.

[32] 钱江波,徐宏炳,王永利,董逸生,刘学军. 多数据流并发窗口连接方法.计算机研究与发展. 2005.10.

[33] 金澈清,钱卫宁,周傲英. 流数据分析与管理综述[J]. 软件学报. 2004 15(8):1172-1182.

[34] 刘学军, 徐宏炳, 董逸生等. 数据流管理技术[J]. 计算机科学. 已录用待发表.

[35] 刘学军,徐宏炳,董逸生,钱江波,王永利. 发现数据流上的频繁项集. 计算机研究与发展. 2005, 32 (4):6-10 .

[36] 王永利,徐宏炳,董逸生,钱江波,刘学军,杨雪梅. 基于低阶近似的多维数据流相关性分析.电子学报.已录用,2006.2 发表.

[37]王永利,徐宏炳,董逸生,刘学军,钱江波. 分布式数据流增量聚集. 计算机研究与发展，已录用，2006 年 3 月发表.

数据流连续查询处理系统设计与实现

作者： 王浩
学位授予单位： 东南大学
被引用次数： 2次

参考文献(3条)

1. 金澈清,钱卫宁,周傲英 流数据分析与管理综述[期刊论文]-软件学报 2004(08)

2. 王永利,徐宏炳,董逸生,钱江波,刘学军 基于低阶近似的多维数据流相关性分析[期刊论文]-电子学报 2006(02)

3. 王永利,徐宏炳,董逸生,钱江波,刘学军 分布式数据流增量聚集[期刊论文]-计算机研究与发展 2006(03)

本文读者也读过(10条)

1. 戴宣 TCP数据流测度的研究[学位论文]2007

2. 吴成涛 数据流管理系统中的概要数据结构算法的研究和实现[学位论文]2006

3. 田海生.TIAN Hai-sheng 数据流管理系统中Max、Min聚集算子的示例概要算法[期刊论文]-计算机应用 2008,28(8)

4. 刘云生.赵海谊.LIU Yun-sheng.ZHAO Hai-yi 基于结构化P2P的分布式数据流系统的查询处理模型[期刊论文]-计算机应用研究2007,24(12)

5. 周杰.毛宇光.ZHOU Jie.MAO Yu-guang 数据流查询语言的研究与实现[期刊论文]-计算机技术与发展2008,18(1)

6. 陶桦 网络运行状况监控研究[学位论文]2004

7. 王丽华.杜彦昌 浅谈数据流管理系统[会议论文]-2008

8. 程转流.王本年.CHENG Zhuan-liu.WANG Ben-nian 数据流中的频繁模式挖掘[期刊论文]-计算机技术与发展 2007,17(12)

9. 沈星星.程学旗.SHEN Xing-xing.CHENG Xue-qi 基于数据流管理平台的网络安全事件监控系统[期刊论文]-小型微型计算机系统2006,27(2)

10. 吕宗健.陈红.杜小勇 数据流管理系统访问控制策略[会议论文]-2008