分类号 TP311 学号 04060049

UDC 密级 公开

工学硕士学位论文

面向海量数据的多数据库 UNION 查询研究与实现

硕士生姓名 王 佳

学科领域 计算机科学与技术

研究方向 计算机软件与理论

指导教师 贾 焰 教授

国防科学技术大学研究生院

二〇〇六年十一月

摘 要

随着计算机应用的发展和普及，特别是网络应用的迅速发展，数据库的规模得到了空前的增长，多数据库成为海量数据存储管理的主要技术手段之一，如何通过多数据库有效的存储和管理海量数据，提供高性能的查询处理，成为人们关注的热点。UNION 查询是多数据库中代价最大的典型操作之一，研究高性能的多数据库 UNION 查询具有重要意义。

本文在分析了现有多数据库中 UNION 查询的处理算法的基础上，针对目前 UNION 查询没有特定解决方案的问题，提出一套面向海量多数据库的 UNION 查询体系结构：通过全局 UNION 查询优化以及分解后的子查询优化，提高了查询并行执行效率；同时，针对 UNION 子查询结果合并问题，提出了增量多路连接、双缓冲并行处理和多路并行归并等算法，形成了一套面向海量数据的多数据库 UNION 查询后处理策略，可以有效地降低 UNION 查询响应时间，减少查询处理的网络传输开销。

本文在大规模事务处理系统（StarTPMonitor）上，基于多数据库 UNION 优化算法，设计并实现了一个面向海量数据的多数据库 UNION 查询子系统 UQS (Union Query System)，测试表明系统在性能指标上达到了预期目标。

主题词：CORBA, 海量数据库, 多数据库, UNION 查询

ABSTRACT

With the rapid development of internet and the increase of network information, a huge amount of data was produced and stored in database. The volume of database storage has scaled magnitude of orders. At the same time, accompany with the new requirement of application, though the techniques using in traditional database has been limited, the existing database can not been discarded. How to improve query performance on the basis of effectively managing these massive data in multi-database becomes the focus on current research. In massive multi-database, union query is one of the operation which cost most, therefore, it is a research work of general and practical significance to solve the problem of improving the efficiency of executing union query to satisfy the demand of performance.

This paper analyzes the problem in face of union query in massive multi-database, based of the architecture of union query system, it used parsing tree to represent union query, then process global optimization and local optimization to union query on the foundation of passing tree, which improved the performance of union query effectively. Based on the query optimizing algorithms in massive multi database, the paper also chose certain strategies as incremental multi-way join, double buffer pool and parallel multi-way merge to optimize UNION query, then designed and implemented the algorithm and strategies for union query in CORBA-Based massive parallel database.

At last, this paper studies the issue of union query for massive multi-database based on StarTPMonitor. Then, on the basis of the study, implemented a union query system (UQS) for massive multi-database, resolved the problem of union query in StarTPMonitor effectively, the practical tests indicate that the function characteristics and the performance characteristics of this union query system is better than the target we desired. In consequence, the algorithm is more effective in practice for union query in massive parallel data query.

Key Words: CORBA, massive database, multidatabase, union query

独 创 性 声 明

本人声明所呈交的学位论文是我本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和撰写过的研究成果，也不包含为获得国防科学技术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谢意。

学位论文题目：面向海量数据的多数据库 UNION 查询研究与实现

学位论文作者签名：王佳　　日期：2006 年 11 月 16 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：面向海量数据的多数据库 UNION 查询研究与实现

学位论文作者签名：王佳　　日期：2006 年 11 月 16 日

作者指导教师签名：[illegible]　　日期：2006 年 11 月 17 日

第一章 绪论

在全球信息化大潮的驱动下，信息无节制地膨胀，各行各业对信息化的需求不断增加，随着实际应用的数据规模和系统复杂度的不断增加，人们把眼光投向了已经有多年研究经验的多数据库系统，希望能借此构造海量数据库系统来满足现实存储数据量庞大的应用需求。本文从这种应用出发讨论面向海量数据的多数据库系统中 UNION 查询的研究与实现。

1.1 应用背景

随着现代计算机科学技术的发展以及金融、电信、交通、信息安全等众多行业信息化需求的不断增加，在系统业务功能逐步增强的同时，系统的数据量也不断增大，数据量高达几百 GB 甚至于 TB 级的海量数据库应用已经出现，与此同时，人们对高性能联机事务处理和复杂查询操作能力的需求也在不断提高。面对如此庞大的数据库应用系统，如何对这些数据进行有效管理，如何针对这些数据实现高效的访问等成为急需解决的关键问题。

目前，多数据库系统[1][2](MultiDataBase Systems，MDBS) 成为海量数据存储管理的关键技术之一。MDBS 是在已经存在的数据库 (称为局部数据库：Local DataBase, LDB)之上为用户提供一个统一的存取数据的环境。一个 MDBS 是由一组独立发展起来的 LDB 组成，并在这些 LDB 之上为用户建立一个统一的存取数据的层次，使得用户像使用一个统一的数据库系统一样使用 MDBS。简单地说，多库系统对用户查询的处理过程如下：多库系统对异构数据库的数据模式进行集成，形成一个统一的全局数据模式，在此基础上把用户针对全局数据模式的全局查询转换为针对各 LDB 的子查询，并分发到各结点的 LDB 上执行，最后将各 LDB 结点的返回结果进行合并，产生最终的查询结果并提供给用户。

在网络环境下，面向海量信息查询的应用越来越多，诸如数据挖掘、搜索引擎、电信业务、网络安全、网络监管、网络信息收集等，其主要特点可以概括为以下几点：

1) 数据规模庞大，且在持续不断地增长；
2) 在持续海量数据加载的同时，对复杂查询和联机事务处理的能力要求高，对响应时间的要求比较苛刻；
3) 对系统可靠性及可扩展性要求高；

由国防科技大学计算机学院 613 研究室自主研发的大规模事务处理系统就是这样一个面向海量数据的多数据库管理系统。它是由一组事务处理中间件和多个

数据库组成的服务平台，用于对多个异构自治数据库进行管理，为用户提供统一的使用和管理视图。它是面向海量信息系统处理应用，通过多数据库技术、并行数据库技术和分布式对象计算技术，基于 CORBA[3][4][5]分布对象计算模型，构建的大规模事务处理领域中间件，主要用于简化海量多数据库系统的开发、部署和运行维护。

作为大规模事务处理系统最主要的业务之一，查询处理扮演着十分重要的角色，而其中的 UNION 查询（两个或多个关系进行并运算从而消去其中任何重复元素而派生出一个结果表）更是涉及到大量数据的处理与传输，从而对 UNION 查询的执行效率以及查询性能提出了更高的要求。但即使是最好的查询方案也不得不遍历分布在各个数据库节点上的全部数据源才能找到满足查询条件的完整实体集。随着数据源的数量和查询条件的数量的增加，UNION 查询所要做的数据传输量也将迅速的增加，系统需要大量的网络和磁盘 I/O 开销，以至于在海量多数据库环境下，即使是很简单的查询处理也变得很难实现。因此，如何对 UNION 查询进行优化处理以满足用户对高性能的需求，已成为系统当前急需解决的问题。

1.2 技术背景

大规模事务处理系统是本文的研究基础。大规模事务处理系统平台是用于开发海量多数据库系统的分布式大规模事务处理中间件，它提供了海量信息数据的并行加载和存储、海量信息数据的并行查询分析等功能。它由多个局部自治的数据库管理系统和一组中间件组成，中间件用于对这些数据库进行管理，并为用户的管理和操作提供统一视图。大规模事务处理系统作为一个海量信息系统开发和管理平台，其主要特点是：

- 提供海量信息的分布存储；
- 提供海量信息的并行加载；
- 提供对海量信息的并行查询与分析；
- 为用户提供统一的对多个异构自治数据库的管理和使用接口；
- 系统规模和处理能力可以在线调整，并动态负载均衡。

大规模事务处理系统主要是由一组查询服务对象和加载服务对象构成。数据查询服务采用基于多数据库的并行查询技术，负责提供对海量信息进行并行查询与分析处理，加载服务则负责海量信息的并行数据加载。大规模事务处理中间件的整体结构如图 1.1 所示。

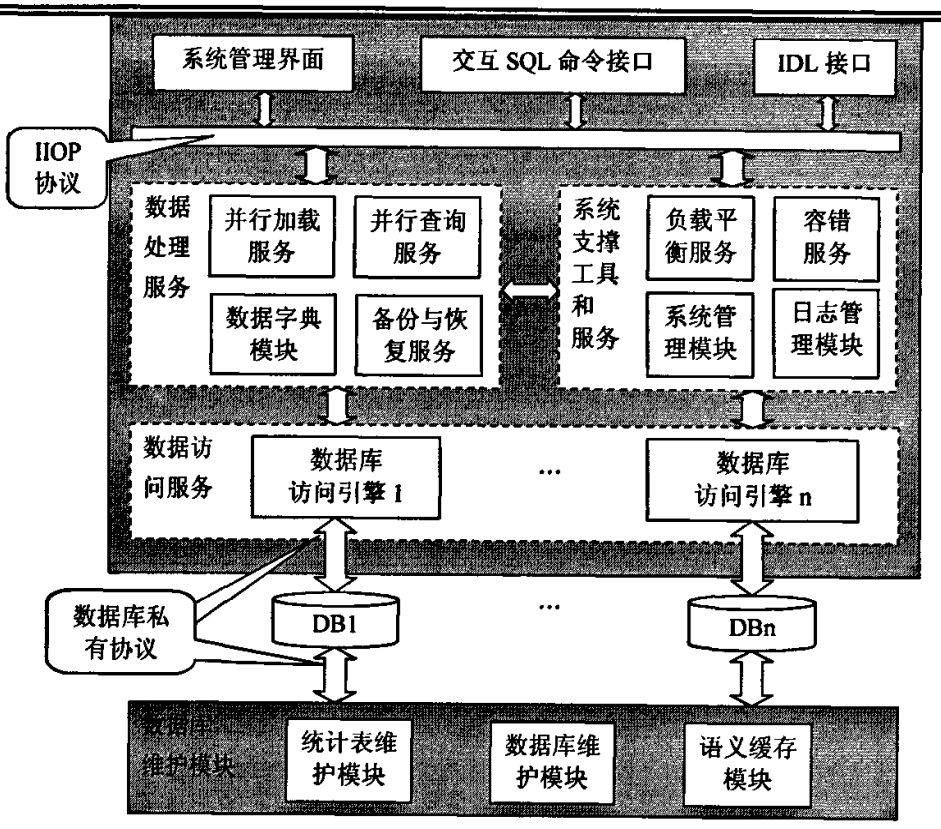


图 1.1 大规模事务处理中间件整体结构图

- 数据库访问层提供对于数据库的访问，屏蔽数据库之间的异构性，避免用户直接访问数据库，可以有效控制对数据库资源的使用和管理。
- 数据库维护层提供数据库日常维护功能，包括表空间的创建、回收和重用，历史分区（数据）的删除和新的分区的创建，统计表的维护，语义缓存的维护等。数据库维护层主要是简化数据库的管理和维护。
- 数据处理服务是大规模事务处理中间件的重要组成部分，包括一组数据处理模块和一组系统支撑工具和服务。

原有的大规模事务处理系统不支持 UNION 查询，为了完善系统的功能，本文将在大规模事务处理系统基础上，研究 UNION 查询的实现和优化技术。

1.3 研究现状

1.3.1 多数据库查询处理技术

多数据库领域的研究目前集中在集成异构数据库系统方面，这些研究大多解决异构数据库的数据模型、模式集成和事务处理方面的问题，它们一般针对各自

的模式框架有相应的查询处理方法。

1.3.1.1 国外研究现状

美国密歇根大学迪尔伯恩分校(Univ. of Michigan-Dearborn)的 Qiang Zhu 和加拿大沃特卢大学(Univ. of Waterloo)的 Per-AkeLarson 等在 CORDS[11]项目中对多数据库的查询处理特别是查询优化做了较多的研究工作，他们提出了查询采样、查询探测、模糊查询等技术对全局查询进行优化，还提出了衰减代价评估模型以实现对全局查询的优化。此外，他们还对动态多数据库环境中的代价模型进行了一些定性分析。但 CORDS 采用的是关系模型，任何加入 CORDS 的数据源在与其他数据源进行数据交互时，必须转换成表格式数据进行处理。

土耳其中东技术大学(METU)的 A.Dogac, F.Ozcan 等在 MIND 项目中提出了一个基于 CORBA 的多数据库系统，使用 CORBA 来处理系统级的异构性和分布性 MIND 有集成的全局模式，它使用传统的面向对象模型作为多数据库的全局模型，使用对象查询语言书写全局查询。他们对查询处理也做了一些有益的研究，并提出了几种查询优化技术，比如基于代价的优化、站点之间连接的优化以及动态查询优化技术等。MIND 系统给出了多数据库查询处理的一些实现技术，但并没有给出一套形式化的定义和理论对这些算法进行描述。

加拿大阿尔伯特大学(Univ. of Albert)的 Ling Liu, Calton Pu 等在 DIOM 项目中提出了一个基于协调器的互操作管理体系结构的具体实现。其查询处理部分由查询接口管理器、动态查询路由、动态查询执行计划器、动态查询结果聚合器构成。其用户接口是基于 WWW 的，用户提交一个基于 WWW 的表单查询，查询接口管理器将表单查询转换成用 IDL 表示的查询。查询路由主要负责定位和选取对查询结果有帮助的相关信息源，查询执行计划器主要负责将用户查询分解为针对各信息源的子查询，并产生一个经过优化的查询执行调度计划。子查询经过翻译后在相应的信息源上执行，查询结果聚合器将各子查询的结果打包给 DIOM 对象，并根据客户的源查询语句组装结果。其语义关联操作和客户查询描述是解决查询结果的语义异构的主要技术。

美国微电子计算机技术公司(MCC)的 Camot[12]项目主要是研究企业分布应用异构信息源的集成问题。它给用户提供了一个透明的信息导航方法，为大型异构分布式信息系统设计了应用程序接口。Camot 是基于对象模型的，所开发出的原型系统包括通信服务、支撑服务、分布服务、语义服务和访问服务五个部分。通信服务和支撑服务主要是在异构的通信平台上提供统一的网络访问方法。分布服务提供了松懈事务管理器和分布式代理工具与客户端应用、目录服务和数据库管理员进行交互，其基于 OMNIBASE 的分布式语义查询和事务管理器(DSQTM)，可以动态地将访问服务提出的查询扩展到所有语义等价和相关的信息源[13][14]。语义服

务提供了 Camot 系统所有集成资源的一个全局视图。访问服务提供了一个面向对象的查询处理方法，允许使用不同的用户接口访问 Camot 所提供的服务。Camot 项目的研究重点在如何集成异构信息源上，对查询处理特别是查询优化的研究相对较少。

1.3.1.2 国内研究现状

东南大学研制的 Galaxy[8][9]是一个基于 CORBA 的分布式异构数据源集成系统，使用对象集成模型(OIM)作为数据集成的公共模型，对象集成查询语言(OIQL)作为其查询语言。OIQL 在 SQL 语言的基础上增加了一些构造符，用于完成对象之间的导航式查询以及对一些集合类型对象的查询。Galaxy 可以查询数据库、WWW 数据等信息源，但对查询优化考虑得并不多。东北大学在基于 CORBA 的多数据库系统 SCOPE/CIMS 中，使用对象查询语言作为全局查询语言，并提出了基于模式集成语义的查询处理规则和路径表达式的查询处理方法。SCOPEICIMS 系统主要是为满足 CIMS 环境下信息集成需求而设计的，对查询计划的生成及查询优化研究的不多。另外，中山大学对联邦数据库也有一定程度的研究。

1.3.2 UNION 查询技术及其优化

UNION 查询是多数据库中代价最大的典型操作之一。在多数据库中，参与 UNION 操作的多个关系可能处在不同的场地上，UNION 的执行策略不同将导致 UNION 操作的代价有很大的差别。如何选取合适的操作执行次序以取得较小的代价是 UNION 查询优化需要研究的主要问题。

在多数据库系统中，查询代价的估算方法一般表示为：

查询代价[10] = I/O 代价 + CPU 代价 + 通信代价

在远程通信网或数据传输率较低的系统中，站点间的数据通信往往会比查询执行中的 I/O 及 CPU 开销大得多，因而作为首要的优化目标来考虑。多数据库查询中，UNION 操作需要的网络通信代价则相对较高。为了使多数据库能有效地处理 UNION 操作，国内外学者一直在进行这方面的研究，形成了各种不同的算法。一般可分为两类：基于排序算法的 UNION 查询技术和基于 Hash 算法的 UNION 查询技术。

由于排序操作可以避免一次传输元组的数量，因而可以减少站点间数据的传输量。可是，使用排序操作也能导致本地磁盘 I/O 次数的增加及本地处理时间的延长。因为排序操作经过了大量的本地运算，延长了本地处理的时间。

另一类查询优化算法则没有使用排序，比如说分布式 INGRES 及 R*算法就采用了 Hash 的算法[15][16]。这是因为对 UNION 操作的查询优化，究竟用排序方案还是用 Hash 方案，取决于数据传输和局部处理的相对代价。如果认为传输代价是

主要的，则采用排序方案处理策略比较有利。相反，如果认为局部处理代价是主要的，则采用 Hash 方案处理策略比较合适。

基于 Hash 的 UNION 算法是又一种被广泛采用的分布式查询优化算法。Hash 划分是指在关系元组的存储位置与它某一属性之间建立对应的函数关系 Hash()，使该属性的每一确定值与唯一的存储位置相对应：

Site_address = Hash (Relation.attribute)

基于 Hash 的查询优化算法也各不相同，比如可以利用站点依赖信息的算法。Hash 划分后的每一个关系就根据 Hash 函数值被水平分片存储到了多个不同的站点中。这样，不同关系通过相同的 Hash 划分后，在合并时就保持站点依赖。

利用该算法做 UNION 操作的各表先在几个不同站点中进行划分，且满足站点依赖，然后分别在每一站点上做 Hash 的操作，最后合并结果。由于该方法涉及 Hash 操作不需要站点的额外开销，因此基于站点依赖信息的算法是一种非常理想的 UNION 查询优化算法。

但实际上，并非所有 UNION 查询条件中的相关属性与 Hash 划分时的属性一致相同，那么这时的查询就可能得不到希望的站点依赖。因此，我们需要先检查这些属性是否使关系都满足站点依赖的条件，若不满足，为了使合并的每个表在各自的查询属性上仍然保持站点依赖，就需要关系元组在该属性上再做一次 Hash 划分。显然，这又引入了额外的系统开销。

通过对比两种 UNION 查询技术，本文中 UNION 查询策略采用的是基于排序的 UNION 查询技术。寻找速度快、附加存储空间开销小的高效排序算法是首要的工作，现有的排序技术主要分为内排序和外排序两种方式，其中内排序分为插入排序、交换排序（冒泡排序、快速排序等）、分配排序（基数排序等）、归并排序等多种方式，外排序分为 2 路合并排序、多路替代选择合并排序等方法，各种排序算法的稳定性和时间复杂度不同，选择排序算法时应注意待排数据本身的特征、辅助空间的大小、关键字的结构以及分布情况等等，在海量多数据库中，需要解决异构分布式环境下海量数据的排序问题，传统的排序算法不能解决网络环境下大量数据的并行排序问题，因此在本文中，UNION 查询分为两个阶段：第一个阶段为内排序阶段，将数据按内存大小分段读入内存进行内存排序，形成多个有序的独立排序段；第二个阶段为合并阶段，选取基于败者树的堕落替代选择合并算法将各个独立排序段合并成最终的有序数据文件。这种基于排序的两趟算法减少了由于多数据源间数据进行排序所需要的大量网络传输，适合于海量数据的排序。

1.4 本文贡献

基于上述分析，当前 UNION 查询优化都集中在基于成本的静态执行计划和查

询后处理优化上。由于局部数据库的自治性，制定高效的执行计划相当困难，而子查询执行的并行性也增加了查询后处理优化的复杂性。

鉴于此，我们在吸取了相关技术优点的基础上，提出构建一套基于 CORBA 的面向海量多数据库 UNION 查询的解决方案，本文的主要贡献如下：

1) 提出一套具有良好局部数据库之间互操作，高可用、高性能、面向海量数据的多数据库系统的 UNION 查询子系统体系结构；
2) 对多数据库系统中异构数据源的 UNION 查询提供良好的模式支持；
3) 在 UNION 查询处理过程中对查询进行代数优化，以提高查询执行的并行性；
4) 在海量信息的应用背景下，查询后处理能通过采取多种排序、合并算法提高系统的响应时间，满足用户的需求。

1.5 论文结构

本文共分为六部分。

第一章：绪论，主要介绍课题的应用背景、技术背景、研究现状、本课题的研究内容以及论文的组织情况。

第二章：介绍了 UNION 查询系统所使用的分布对象中间件技术，同时介绍了与 UNION 查询相关的数据分布算法和并行排序算法，最后，介绍了现有数据库在 UNION 查询处理与优化时的主要方法。

第三章：介绍 UNION 查询系统的体系结构及设计要点，着重介绍了查询分解、查询优化和查询后处理模块的设计。

第四章：详细阐述了 UNION 查询系统的具体实现，包括查询内部表示、查询分解、全局查询重写、子查询重写、查询结果提取以及查询结果集合并等功能的具体实现。

第五章：对 UNION 查询系统进行了功能和性能测试，测试结果显示 UNION 查询系统完全按设计目标正确地完成 UNION 查询功能，并以可以接受的开销为系统提供了高效的 UNION 查询。

结束语：对全文进行了总结，并指出下一步的研究方向。

第二章　相关技术

UNION 查询服务主要用于为海量多数据库提供高性能的 UNION 查询，本文中的 UNION 查询服务系统是基于大规模事务系统实现的子系统，因此我们需要关注的技术主要有：分布对象中间件技术、数据分布方法、并行排序方法、查询处理和查询优化算法。

2.1 分布对象中间件技术

当前，面向对象的分布及算技术已经成为当前网络中间件的主流技术，出现了以 EJB/J2EE 、COM+/DNA、CORBA 为代表的三个技术分支[6]。

1) J2EE 技术：易用，跨平台(采用纯 Java 语言，平台无关性)，低性能(只能使用 Java 语言实现，相对于 C++等语言性能稍低)；
2) COM/DCOM 技术：易用，不跨平台（只能用于或仅在 Windows 平台上使用效果较好），高性能；
3) CORBA 技术：跨平台，跨语言（使用 IDL，具备语言无关性），高性能；

大规模事务处理系统是在 CORBA 平台 StarBus 之上构建的，本文也将采用 CORBA 技术实现多数据库的 UNION 查询服务。

CORBA 公共对象请求代理体系结构（Common Object Request Broker Architecture）是由对象管理组织（OMG Object Management Group）提出的应用软件体系结构和对象技术规范，它是开放的、独立于供应商的支持网络应用程序互操作的规范。CORBA 规范是基于抽象的对象模型的分布式对象标准，在分布式软件总线支持下，实现分布式异构环境下的面向对象软件构件之间的通信与系统集成，是表达应用系统、应用构件之间有效通讯的连接技术。它是目前较好的跨平台技术，它独立于网络协议、编程语言和软硬件平台，支持异构的分布式计算和不同编程语言的对象重用，现在已成为软件开发的主流，并被业界广泛接受。CORBA 的结构如图 2.1 所示：

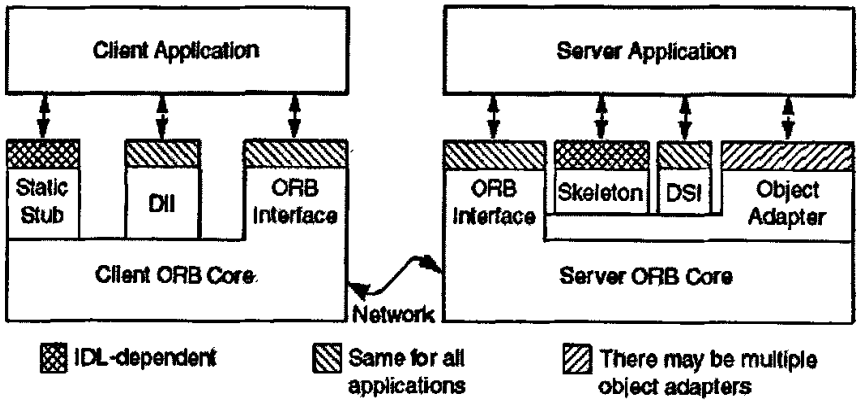


图 2.1 公共对象请求代理体系（CORBA）

一般情况下，由客户应用程序提出请求，服务器应用程序接受这些请求并作出响应。请求流由客户应用程序向上提交，通过 ORB(Object Request Broker, ORB)，上传到服务器应用程序。可由下列几种形式实现这个过程：

1、客户机有两种选择方式提出请求。第一种，使用由对象接口定义，用 IDL 编译器生成相应编程语言的静态存根（static stubs）；第二种，使用动态调用接口 DII（Dynamic Invocation Interface, DII）。不论哪种方式，客户机都直接将请求传送给与这个进程链接的 ORB 核心。

2、客户机 ORB 核心通过网络传送给与服务器应用程序相链接的服务器 ORB 核心。

3、服务器 ORB 核心将这些请求分配给对象适配器（Object Adapter），由它产生目标对象。

4、对象适配器进一步将请求分配给实现目标对象的伺服程序。与客户机一样，服务器可以选择静态或动态调度机制用于它的伺服程序。这取决于是由对象接口定义，用 IDL 编译器编译的静态框架，还是其伺服程序可使用动态框架接口（Dynamic Skeleton Interface，DSI）。

5、伺服程序执行请求后，将结果返回给客户应用程序。

2.2 并行算法

从数据库的查询算法来看，数据库的 UNION 查询处理流程主要分为四个步骤：

1) 数据重新分布，即根据分布算法将数据分布到执行节点上；

2) 全局查询的分解，即将 UNION 查询分解为各个子查询分别执行；

3) 子查询并行执行，即在所有执行节点并行执行子查询；

4) 结果汇总，把执行节点的查询结果汇总，返回给用户。

在这上述步骤中，为使 UNION 查询得到最大的并行性，在查询时将数据分布到各个执行节点上，这需要大量的网络和磁盘 I/O 开销，如果可以在数据加载时预先进行分布，则可以减少第一阶段的开销，极大提高并行处理的性能。

通过将 UNION 查询在执行前进行充分的优化，可以减少大量的本地开销，并提高查询的并行度。选择良好的并行排序算法不仅能充分的对查询进行优化，而且有利于降低结果合并时的开销，提高查询后处理的效率。

以下将分别对数据分布方法和并行排序算法作详细介绍。

2.2.1 数据分布方法

数据分布方法即并行数据库物理存储方法，又叫数据划分，是指如何把一个数据库对象（关系、索引等）均匀地分布到各个节点机上，使得在查询处理的过程中系统的并行性能够得到充分的发挥，以此来最小化查询处理的响应时间。研究表明：数据划分对并行数据库系统的性能具有极大的影响，数据划分策略对于能否充分利用系统的 CPU 和带宽资源，减少通信开销，平衡系统负载和减少计算量，从而能否最佳地发挥并行性和系统性能至关重要。

按划分属性数，数据划分方法可以分成一维数据划分和多维数据划分，它们都属水平划分。一维数据分布方法是最简单的数据分布方法，一维数据分布的特点是：通过划分关系的一个属性的域值来划分整个关系，得到一组子关系，然后在多处理机之间分布这些子关系。常用的一维数据划分方法有 round-robin 划分、hash 划分、range 划分。一维数据划分不能很好地支持在关系的非划分属性上具有条件谓词的数据查询。这样的查询必须被送到所有包含操作关系的元组的处理节点上进行。为此，人们提出几种多维数据划分方法，多维分布方法是研究如何利用关系的多个属性对整个数据空间进行划分和放置，即划分属性的维数大于 1，如 CMD 方法、BERD 方法和 MAGIC 方法。

由于本系统的表的划分属性都是一个，即一般只对一个属性做选择谓词的查询，所以在本系统中我们没有选择复杂的多维数据分布方法，而是选择了简单而有效的一维数据分布方法。

根据表在并行数据库各个节点的存储方法的不同，我们把表分为两大类：划分表（division table）和复制表（replica table）。复制表相对来说一般比较小，数据量一般在 100，000 条以下，一般可以 Cache 在内存里。划分表一般是数据量比较大的表，经过划分也不太可能 Cache 在内存里。

对划分表，我们同时提供了 round-robin 划分、hash 划分、range 划分三种不同一维数据分布方法的支持，用户可以根据需求灵活选用。

下面我们来看看这几种一维数据划分方法：

1. Round-Robin 分布方法

设处理单元数 N，Round-Robin 方法将把第 i 条记录放置于 i MOD N 处理单元上。

元组的排序：数据生成的时间序、或属性值的降/升序等。

如图 2.2 所示：

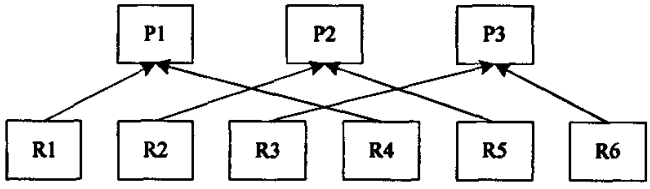


图 2.2 Round-Robin 分布

优劣：

1) 数据规模比较大时，能达到较高的并发度；
2) 数据规模较小，则 Round-Robin 算法也必须启动大多数的处理单元，浪费系统资源；
3) 破坏数据访问的局部性。

2. Hash 分布方法

设处理单元数 N，构造一个散列函数 Hash (Attr)，其中 Attr 是关系 R 的划分属性。{1，2，…，N}是值域。

划分属性：由一个或者多个属性组成。是元组分布的依据，属性值相同的元组分布在同一个处理单元的磁盘上。

如图 2.3 所示：

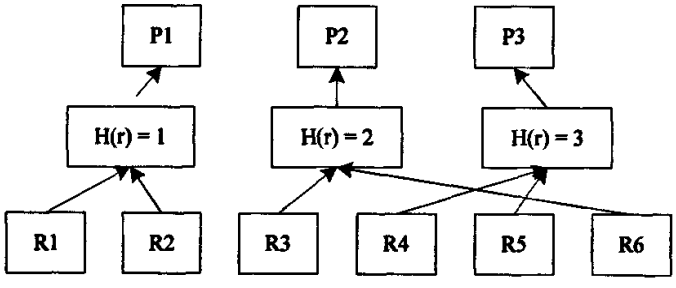


图 2.3 Hash 分布方法

优劣：

1) 有效支持大规模和小规模的精确匹配查询的数据访问；
2) 良好的散列函数的选取并不太容易；
3) 划分属性不是数据库关系的关键属性时，划分属性存在大量的相同取

值的数据元组，需要很好地解决碰撞问题；

4) 不能保证整个数据空间在各处理单元上的均匀分布，所以有较大的概率会引起大规模数据访问的不均匀性，即若干处理单元将成为瓶颈；

5) 不能很好的支持具有良好访问局部性的数据访问模式。

3. Range 分布方法

将关系 R 的划分属性 Attr 按照一定规则排序，一般可采用属性值域的升/降序。将所排值域序列划分为 N 个子区间，为 $I_1=[V_0, V_1]$，$I_2=[V_1, V_2]$，…，$I_N=[V_{N-1}, V_N]$，其中，N 为系统处理单元总数。每个处理单元一个区间。

如图 2.4：

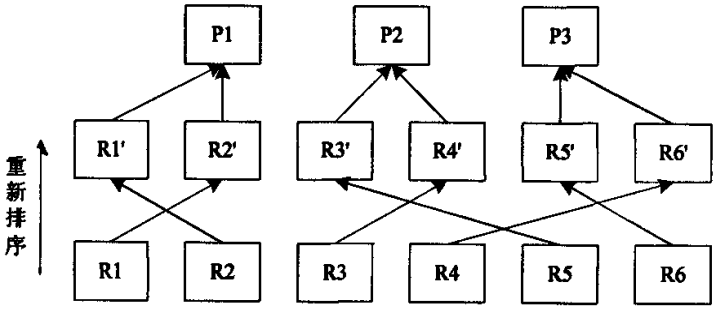


图 2.4 Range 分布方法

优劣：

1) 有效地支持数据访问规模较大的数据操作；

2) 簇聚性；

3) 不能保证子空间划分的均匀性，导致负载不平衡。

在大规模事务处理系统中，为了提高查询的速度，同时增强对后台数据库系统的管理，采用了并行域技术。并行域是对多个数据库系统的一种逻辑划分，一个并行域将相关的几个数据库组织在一起。在该系统中，表的存储位置为并行域，即每个表都要指定所存储的并行域，如果该表为划分表，则依据数据划分方法将该表分布存储在其并行域包含的每个数据库系统中；如果该表为复制表，则将该表复制存储在其并行域包含的每个数据库系统中，每个数据库系统都有一份相同的副本。

同时，并行域也是查询的单位，所有的查询都必须在同一个域内完成，不允许跨域直接的表的关联查询。

根据应用的特点，要减少查询处理流程的第一步时耗，就要让并行查询生成的子规划能够局部执行，这也是系统数据划分的首要原则。

复制表由于比较小，一般采用数据复制的方法存储，数据复制也是为了使得与划分表的关联操作能够局部执行的需要。

划分表与其业务相关（即可能做关联操作）的划分表按同样的属性数据划分，这也是为了使得与其他划分表进行的关联操作能够局部执行的需要。划分表进行数据划分是通过将大的数据集分割成比较小的数据集，从而减少查询过程中的 I/O 次数，提高对数据操作的并行性，以便提高查询性能。此处支持对 Range 划分、hash 划分、Round-Robin 划分。由于系统的数据量很大，而 round-robin 划分方法能比较均匀地划分数据，所以一般都选择这种分布方式。

数据划分的目标应该和查询优化的目标一致，即把相关划分表都按照同样的属性划分到同一个并行域中去，相关的复制表也复制到这个并行域中，这样在进行关联操作的时候就可以在局部执行而没有网络复制的开销。

但如果数据划分没有设计好的话，不仅不能带来性能上的提高，性能反而会有所下降。而且由于数据划分的原因有些查询语句在系统不支持，比如 UNION 查询，这就需要我们通过设计 UNION 查询系统来实现大规模事务系统在并行域的情况下支持 UNION 语句的查询。

2.2.2 并行排序算法

并行数据库操作算法是并行数据库区别于与传统数据库系统的核心。如果没有设计良好的并行数据库操作算法，那么数据库系统根本无法利用多处理机和多 I/O 设备带来的任何查询性能的提高。

目前已经形成的经典并行排序算法，主要分为以下三类：

1、基于合并操作的并行排序算法：

基于合并操作的并行算法假设 P 个处理机逻辑上构成一个树。每个处理机具有 N/P 页被排序关系 R 的数据。树的叶节点先把它们自己的那段关系 R(i)排序，这样就构成了 P 个关于排序属性的 R 的有序段，然后每个叶节点传送有序段到它的父节点，由父节点合并这些有序段。产生更长的有序段，这个过程反复进行，直到根节点产生最终的排序关系。比较经常用到的是基于二叉树的合并排序算法，如图 2.5：

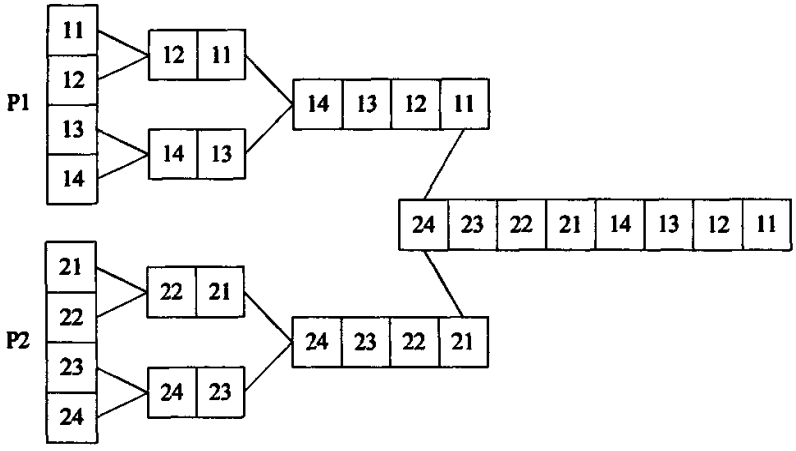


图 2.5　基于二叉树的并行排序合并算法

2、基于比较—交换操作的并行排序算法

基于比较—交换操作的并行排序算法是由并行排序网络演变而来的。这类并行排序算法的基本思想是反复使用比较—交换操作实现并行排序。比较经典的基于比较—交换操作的并行排序算法是并行块双调排序算法，它的基本操作是块比较操作，我们把它简单一记为 CP 操作。在这种算法中，每个处理机其实就是一个 CP 操作容器，在这个容器中，输入的是两个有序的数据段，输出的是两个等长的有序数据段 L 和 H，并且满足 L 中的任何元素不大于 H 中的任意一个元素。两个处理机的并行块双调排序算法如图 2.6：

其具体过程如下：

1. 每个处理机分别执行一次 CP 操作;
2. 处理机 1 将高值段传给处理机 2，处理机 2 将高值段传给处理机 l;
3. 两个处理机再执行一次 CP 操作，处理机 1 将高值段传给处理机 2,处理机 2 将低值段传给处理机 l;
4. 每个处理机再执行一次 CP 操作，并分别输出各自的有序段，完成对整个输入文件的全排序。

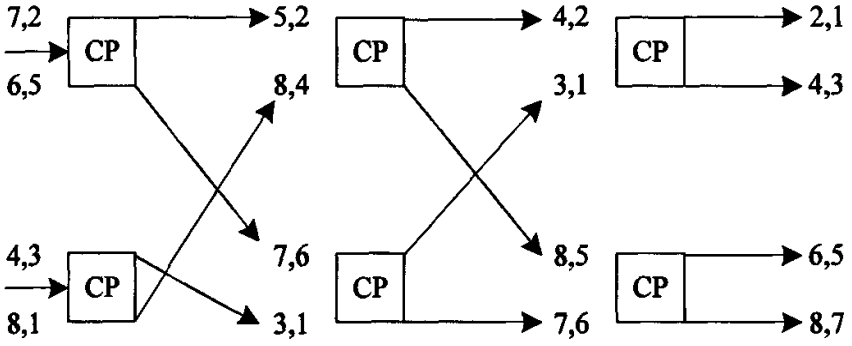


图 2.6 两个处理机的并行块双调排序算法

3、基于划分的并行排序算法。

基于划分的并行排序算法分为两个阶段，第一个阶段把关系 R 划分为 P 个子集合，每个子集合传送到唯一一个处理机。第二阶段各个处理机并行地排序分配给它的子集合，完成 R 的排序。本文使用的就是这种排序方式。

基于划分的并行排序算法。针对我的数据划分是均匀划分的，该排序算法的整个过程如下：

1. 获得各个处理机中的数据库的关于排序属性的最大值 max(i)和最小值 min(i)。
2. 各个处理机进行通讯获得整体的最大最小值 max final 和 min final。
3. 将这个最大最小值广播到各个处理机中。
4. 将最大最小值之间的区域平均划分为 P 段，D(1)， D(2),...D(P)
5. 各个处理器并行进行下面的操作：
   a) 如果当前字段的排序属性值在自己的划分区域中，则把这个字段加入本机的结果链表中。
   b) 如果当前字段的排序属性值不在自己的划分区域中，则把这个字段值打包到相应的数据包中，直到整个处理机的所有元组都遍历结束。
   c) 把各个打包的数据结果发送给相应的处理机。
   d) 接收别的处理机发送的打包结果。
   e) 解包并将结果依次写入结果链表中。
   f) 得到各个处理机的结果指针。

整个排序的过程如图 2.7 所示：

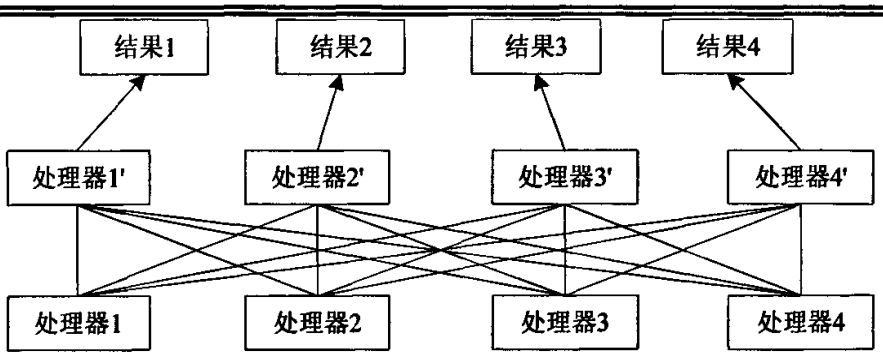


图 2.7 并行排序全过程

考虑采用基于划分的排序方法出于以下几点考虑：

1. 算法比较清晰，尽量地减少了消息传递，有利于提高加速比。

采用基于划分属性的并行排序方法，从并行实现上是相对简单的，并且相对于其它两种并行排序方法，基于划分属性的并行排序方法可以最大限度地减少消息传递带来的性能开销，能够很好地解决并行计算中的网络瓶颈问题。这样的实现具有相对较高的可扩展性。

2. 算法会在各个节点机上产生各自的结果。

基于划分属性的并行排序算法不同于合并排序的算法，它会在所有节点上产生各自的结果并可以很容易地将结果进行汇总，它不会大量占用某一个节点的内存或者硬盘，而是把这个数据库的负载尽量均匀地分布到各个处理器和处理器内存中，更大限度地发挥了无共享结构的优势。而合并的算法中总是将中间的结果储存到某个单一的处理器中，这样如果结果很大，则很容易造成内存或者硬盘资源不够，不利于并行数据库系统的扩展。

3. 结果很容易进行分别输出。

基于划分属性的并行排序算法可以很容易地进行打印组织和获取结果，因为在各个处理器上得到了值域完全不交叉的结果集合，可以很容易地组织按照顺序打印。

4. ORACLE[32]本身的排序算法采用了链表结构来存储结果，这十分有利于消息传递中的插入排序操作。

处理流程如图 2.8 所示：

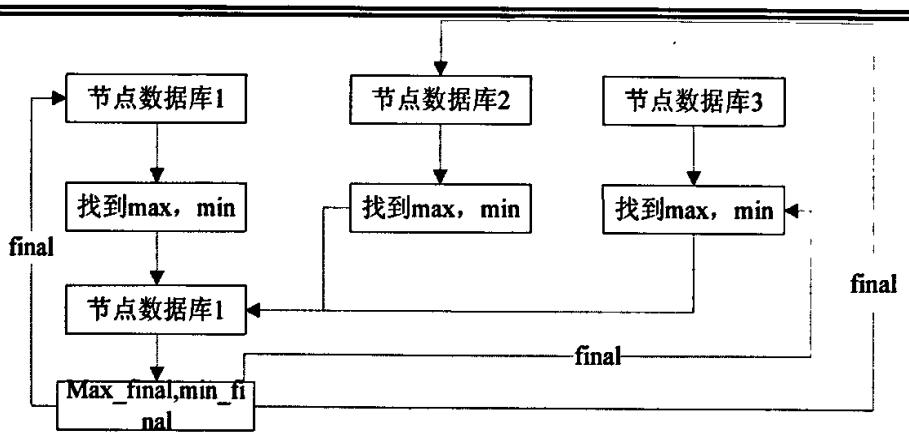


图 2.8 并行排序处理流程

这样的程序流程具有以下好处;

1) 便于程序的同步。阶段性的流程虽然在并行性上减少了许多，但是过分细力度的并行其实并不是并行数据库所要做的，关系数据库查询语言的非过程性就注定了基于关系数据库查询的并行单元不会太小。
2) 集中压缩结果。一方面能够尽量地减少消息传递的流量，另一方面，更加减少了消息传递的次数，从而减少了因为消息传递而形成的众多额外开销。
3) 各个节点并行发送和接收。可以尽量避免死锁的产生，更大程度地发挥并行效率。
4) 内存的消耗被分配到了各个节点，而且各个节点的内存消耗几乎一样。

2.3 查询处理与优化

伴随着数据库技术的不断发展，相应的查询优化技术的研究与发展也在不断进行，如何进行查询的处理要与具体的数据库系统相接合，本节将介绍传统数据库、并行数据库以及多数据库下查询处理和查询优化的方法。

2.3.1 查询处理

多数据库系统具有全局模式，用户需要使用多数据库全局查询语言提交对多数据库的查询，多数据库查询处理要求能自动地将全局查询语言转换成与局部数据库对应的子查询语言，并生成查询执行计划，交付给有关的局部数据库去执行。在异构多数据库系统环境下，各个局部数据库系统的局部优化能力不同，在不同场地的查询代价函数不同，这些不同使得对全局查询的处理难度增加。

多数据库查询处理流程如图 2.9 所示。

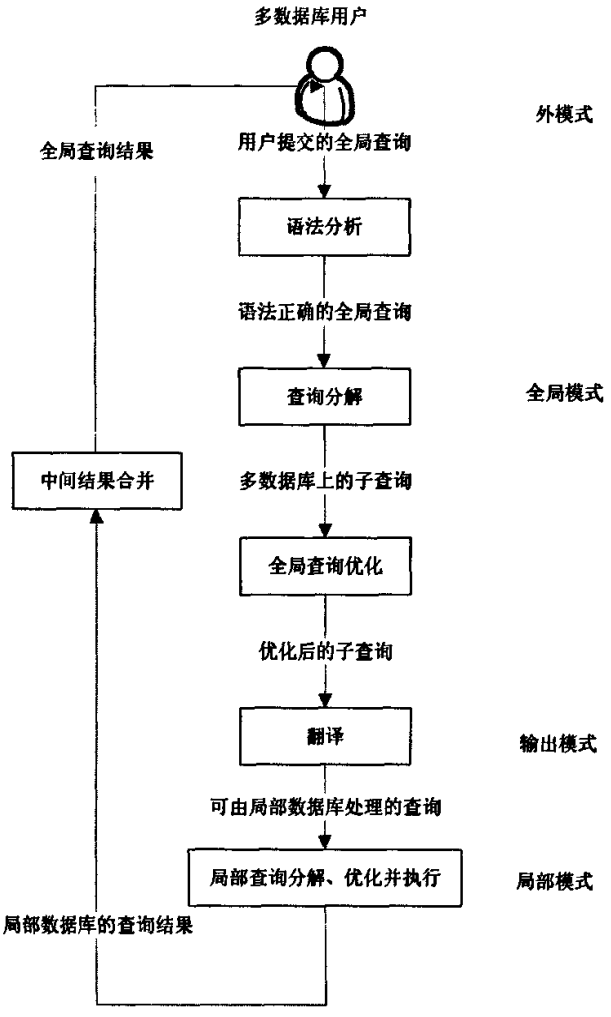


图 2.9　多数据库查询处理流程图

其步骤一般如下：

1. 用户使用多数据库查询语言提交一个全局查询，多数据库系统接受查询并进行语法检查。这里的语法检查除了需要检查多数据库语言的正确性外，还要根据全局模式中的信息对其进行语义上的检查。
2. 检验正确后，查询分解器根据全局模式信息将该查询分解成若干子查询。每个子查询对应于一个局部数据库系统。
3. 查询优化器制定出一个执行计划，说明需要访问那些局部数据库系统、如何组装中间结果、在哪里执行全局处理等。
4. 启动并执行查询计划。即将子查询发送到各个局部数据库执行引擎去执行。
5. 结果合成器合并个子查询的执行结果，并组成最终的全局查询结果返回给

多数据库用户。

多数据库查询处理时与传统数据库查询和分布式数据库查询在以下几个方面可以采用相同或类似的技术。

1. 查询语句的语法分析。无论是多数据库还是其他的数据库系统。尽管它们所采用的查询语言有可能并不相同，有的使用关系查询语言 SQL，有的使用对象查询语言 OQL，但对用户所提交的查询语句都必须进行语法分析，并根据字典信息（多数据库中是全局模式信息）对关系（或对象）、属性等的合法性进行检查，以保证提交给查询系统的是一个正确的查询。这里可以采用 YACC 等编译技术对查询语句进行分析，在本文的多数据库查询处理中将不进行专门论述。
2. 查询语句的内部表示。在传统数据库和分布式数据库中，对查询语言进行语法分析后，需要将查询语句转换成某种内部表示，如关系代数表达式、查询树表示等，以便进行基于代数的优化处理。在多数据库系统中，需要解决同样的问题，本文将采用查询树作为查询的内部表示。
3. 集合代数优化技术。一般传统数据库和分布式数据库多采用关系模型，关系代数的优化是它们进行代数优化的主要技术。多数据库系统无论是采用关系模型还是采用对象模型，都需要用到集合代数操作。从数学意义上来说，关系代数是集合代数的一种，因此集合代数的一些操作规则在这些数据库中是相通的，也就是说关系代数的大部分操优化规则在多数据库查询中也是适用的。

在多数据库系统中，全局查询的执行通常需要经过多次翻译或转换。当查询经过不同系统层次时，翻译器将其转换成相应层次上的语言和数据表示形式，并解决各层次上表示之间的差异问题。

在全局查询分解处理的过程中，多数据库还必须处理库间数据依赖（Interdependency）、管理全局资源、支持附加语言特性（如全局语言和中间语言）。库间数据依赖是指各成员数据库中的数据之间存在着的依赖和约束关系。由于库间依赖的存在，会形成功能级联到查询范围以外的多个数据库中。全局资源包括全局模式信息、负责全局处理的软件模块、局部工作空间，这些资源通常分布在各个场地上。

与传统数据库和分布式数据库相比，多数据库系统的查询处理主要在以下两个方面更为复杂。

1. 全局查询分解。在多数据库查询处理过程中，需要对用户所提出的全局查询进行分解，以得到局部数据库上的子查询。全局查询分解需要用到多数据库的全局模式信息，全局模式信息记录了多数据库的全局信息到局部数

据库信息的一种映射，因此在查询分解之前需要建立多数据库的模式映射。这些内容在其他数据库查询处理中是没有的。

2. 查询优化。由于多数据库系统具有异构、分布和自治的特点，其查询优化也变得非常复杂。前面所讨论的集合代数优化技术在多数据库中会遇到新的情况，例如，原有的集合代数对带有限定条件的输出类进行操作时还必须进行相应的扩充。此外，查询处理器和局部代理之间的通信代价也是多数据库查询优化必须考虑的内容。除此之外，多数据库查询处理中的优化还包括其他许多方面，例如，不同的局部系统优化能力不同，如何拟定局部优化策略以实现局部查询优化等。

2.3.2 查询优化

2.3.2.1 集中式数据库查询优化

关系模型的查询优化基本上划分为两个层次，一个是抽象级别较高的层次，即代数优化，一个是抽象级别较低的层次，即非代数优化。

代数优化仅涉及逻辑数据结构及在逻辑数据结构上定义的运算。一般选择规范关系作为逻辑数据结构，并以关系代数和集合代数的运算作为运算符。关系模型的各种查询语言都能转换成代数表达式。代数优化首先研究关系代数表达式的等价变换规则，所谓关系代数表达式的等价是指用相同的关系代替两个表达式中相应的关系所得到的结果关系是一样的。利用等价变换规律就可以得到关系代数表达式的优化算法。

代数优化方法是试探性的，以求得近似最优解，因为有时不存在求得最优解算法，有时虽存在最优解算法但复杂性太高不能实际应用。一般有以下规则：

1. 施加于同一数据对象上的若干简单谓词可合并为一复合谓词；
2. 选择运算应尽早执行，有时它能使执行时间成数量级地减少；
3. 对于某些使用频率较高的属性，应在它上面建立索引或分类文件；
4. 投影应尽早进行，但不先于选择；
5. 确定输出顺序的排序运算最后执行；
6. 一串连接运算或一串集合运算的执行顺序由参与运算的关系的势决定；
7. 应找出公共子表达式，并估计应采取的优化步骤。

非代数优化在较深入的存取路径级进行。利用存取路径，合并运算或引入排序运算进行优化。优化器可以较容易地为查询产生多个等价的查询计划，估计每个查询执行计划的代价，从中选择代价最小的查询计划来执行。

2.3.2.2 分布式数据库查询优化

集中式数据库中查询优化技术一般在分布式数据库中还有有用的，但是由于

分布式的特点，使得分布式的查询优化又有了新的内容：

(1) 网络传输延迟问题；

(2) 网络中多处理器的存在提供了并行数据处理和传输的机会。

分布式数据库的查询优化器的主要任务是控制和加快查询的执行和数据传输过程。在分布式查询处理中，一个查询可以根据它所使用的数据的存储位置分解为一系列局部查询，而这些局部查询像在集中式数据库系统中一样可以进行再分解。因此，在分布式数据库中查询的执行可能包括一系列可以并行展开的局部操作，以及控制场地间数据传输的全局操作。

在分布式数据库系统中，全局查询是针对全局模式定义的全局关系所进行的查询，而全局模式是定义在各个局部数据库的局部模式之上的。关系代数表达式与 SQL 查询之间可以相互转换。假设查询是以关系代数表达式来表示的，如果将全局模式关系全部用局部模式关系来替换，将得到一个定义在局部模式关系上的代数表达式，称这个查询的中间表达式为查询的正则形式。在把一个全局查询转换为查询的正则形式的过程中，必须服从等价变换规则，以保证得到的查询的正则形式与原查询是等价的。在分布式数据库系统中，一个特别有用的等价转换是作用在合并操作和连接操作之上的、选择和投影操作的分配率。

在所有基本的关系运算中，UNION 查询无论是在时间上还是在系统资源开销上都是最大的。在分布式数据库中主要采用增量多路连接算法，与普通的二路连接操作算法相比，局部数据处理将有所增加，但数据在不同场地间的传输将会大量减少，而 UNION 查询优化的主要目标正是减少数据传输量，降低网络传输量。

2.3.2.3 多数据库查询优化

多数据库查询优化的准则是使通信代价最低和使响应时间最短，即以最小的总代价、在最短的响应时间内获得需要的数据。所谓响应时间，是从接收查询到完成查询所需的的间。它既与通信时间有关，也与局部处理时间有关;而通信代价与所传输的数据量和通信次数成正比。具体说来大致有如下几种方法：

(1) 选择适当的中间结果连接算法；

(2) 尽量减少磁盘数据存取的次数；

(3) 计算对象复用，例如数据库连接对象、内存对象等；

(4) 优化对象调度方式，减少无谓的等待时间，提高并发度；

(5) 减少查询求解过程中的中间关系；

(6) 避免重复计算子查询；

(7) 尽可能让局部数据库多承担工作，减轻多数据库服务器的负担；

结合前文对集中数据库和分布式数据库查询优化方法的分析，这里我们给出多数据库中代价估算的方法。

- ✓ 磁盘代价：中间关系的大小和结果合并时候的连接的大小是主要影响，其中，中间关系大小=元组数*每个元组的字节数； 连接大小取决于连接的策略。
- ✓ 服务模块并发度：由于多数据库的分布特性，查询的并发执行对性能影响较大。
- ✓ CPU 代价：CPU 的代价主要来自连接、选择和投影等运算。
- ✓ 通信代价：在远程通信网络和高速局域网中，通信耗时是不同的。

2.3.2.4 查询优化技术比较

在集中式数据库系统中，查询优化的目的在于寻求总代价最小的执行策略。通常，总代价是以查询处理期间的 CPU 代价和 I/O 代价来衡量的。由于集中式数据库系统大部运行在单个处理器的计算机上，使总代价最小就意味着使查询的响应时间最短。

在分布式数据库与多数据库系统中，常以两种不同的目标来考虑查询优化。一种目标是以总代价最小为标准。除了像集中式数据库系统那样考虑 CPU 代价和 I/O 代价之外，总代价还包括数据通过网络传输的代价。另一种目标是以每个查询的响应时间最短为标准，这一点在分布式数据库系统中具有重要的意义。因为，分布式数据库系统与多数据库是由多台计算机组成的系统，数据的分布和冗余也增加了查询的并行处理的可能性，从而可以缩减查询处理的响应时间，加快查询处理速度。

由此可见，与集中式查询相比，分布式数据库与多数据库查询处理增加了不少新的内容和复杂性。不同的查询处理方法，其查询的通信代价和并行处理程度是大不一样的。虽然，在也使用某些集中式查询处理中的技术和方法，但就其问题的规模和优化的因素，都与集中式查询处理有质的不同。

第三章 UNION 查询系统设计

本章讨论了在大规模事务处理系统中 UNION 查询服务模块的体系结构，以高性能为主要目标，设计了一套面向海量信息多数据库中的 UNION 查询系统—UQS(Union Query System)。

本章组织结构如下：第一节介绍了 UQS 的设计思想；第二节主要介绍 UQS 的体系结构；第三节详细介绍了系统在 UNION 查询处理的主要思想；第四节基于分解后的 UNION 查询结果主要介绍查询后处理优化的方法；最后一节针对频繁的 UNION 查询连接对系统性能的影响及其解决方法展开了详尽的讨论。

3.1 UQS 的设计思想

大规模事务处理系统具有规模大、速度高、持续运行等特点，高性能是 UNION 查询服务首先需要考虑的问题，如何在面向海量信息的多数据库中提高查询执行的并行性和减少网络传输开销是设计的关键所在。

UNION 查询系统的基本设计思想可以主要概括为以下几点：

1) 针对查询执行代价和查询响应时间进行优化，通过控制和加快查询处理的执行流程和数据的传输过程，充分利用多数据库的并行处理能力，加快查询响应的速度；
2) 通过在 UQS 查询模块之间实现松耦合，进一步提高 UNION 查询处理的并行性；
3) 优化查询算法，对 UNION 查询进行代数优化，提高查询执行的效率，降低本地资源的开销；
4) 针对海量信息查询应用的特点，优化查询后处理策略，提高结果合并的执行效率，减少网络传输，提高查询响应时间。

3.2 UQS 的体系结构

如图 3.1 所示，在数据查询层，UNION 并行查询服务（UQS）包括三个模块：

- 查询分解器，它将用户提交的查询语句进行语法分析，将语句分解为多个独立的子查询语句；
- 查询处理器，针对海量并行数据库的特点，将查询语句进行查询分解以及查询优化，并将处理后的语句提交给并行数据访问服务；
- 结果合成器，对查询结果进行查询后处理，将并行数据访问服务返回的结果集进行合并。

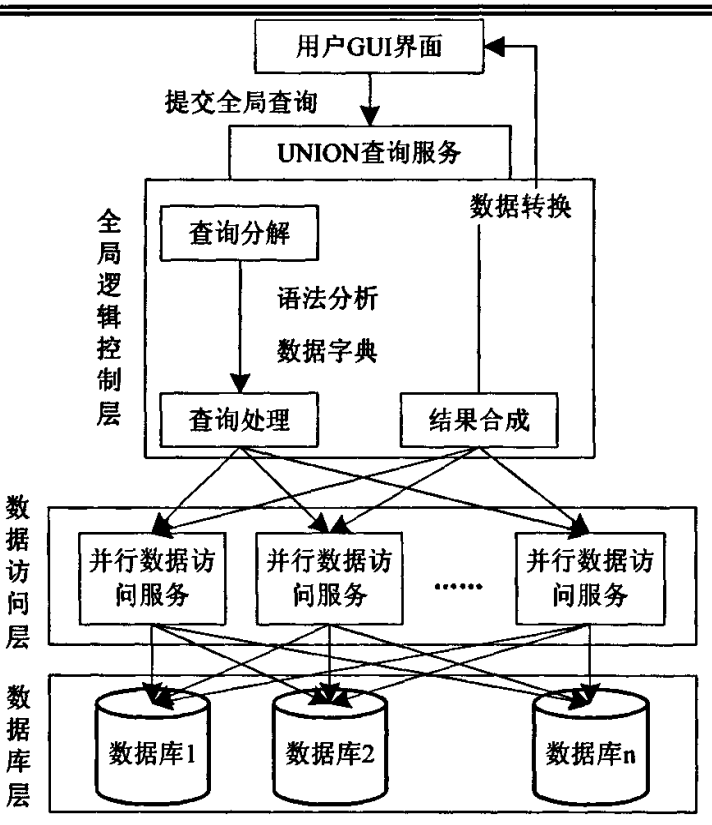


图 3.1 UQS 体系结构

3.3 UQS 查询处理

通常，在 DBMS 组成中对于用户所观察到的数据库性能影响最大的是查询处理器。查询处理器通常表示为两个部分：

(1) 查询编译器，它将查询翻译成一种内部形式，称作查询计划。查询计划是要在数据上执行的一系列操作。通常，查询计划中的操作是关系代数的实现。查询编译器包括三个主要部分：

a) 查询分析器，它由文本形式的查询出发，建立一个树结构。

b) 查询预处理器，它对查询进行语义检查(例如，检查查询中所提到的关系是否都确实存在)，并进行某些树结构转换，将分析树转换为表示最初的查询计划的代数操作符树。

c) 查询优化器，它将最初的查询计划转换为对于实际数据的最有效的操作序列。查询优化器利用元数据和关于数据的统计数据来确定哪一个操作序列可能是最快的。例如，一个索引的存在可能会使得某个查询计划比另一个计划快许多。

(2)执行引擎，它负责执行选中的查询计划中的每一步。执行引擎与 DBMS 中大多数的其他成分都有交互，或直接交互，或通过缓冲区。为了对数据进行操作，它必须从数据库取得数据并放到缓冲区中。它需要和调度器进行交互，以避免访问被加了锁的数据。它需要和日志管理器进行交互，以确保对于数据库的所有修改都正确地记了日志。

在本文中，UNION 查询服务对 UNION 查询得处理也采取相同的结构：由语法分析器将查询表示为语法分析树的形式，对其进行语法语义的检查，再由查询处理器对查询语句进行全局 UNION 查询优化和局部子查询优化，最后与并行数据访问服务进行交互执行 UNION 查询。

3.3.1 UNION 查询分解

UQS 接受到 UNION 查询请求后，将其翻译为语法分析树的内部形式。这时，查询分解器需要对 UNION 查询语句进行正确性检查，并将 UNION 查询分解为多棵子查询语法分析树的形式。

3.3.1.1 查询内部表示

对于 SQL 这样一种在语法上呈现层次结构的查询语言，没有良好的内部表示形式，处理起来将是十分困难的。查询树模型是表示 SQL 结构的一种有效方式，它结合了语法和运算两种性质，实际上是语法树和算符树的合并。查询树不仅适合算术和逻辑运算，也适合所有面向集合的数据模型运算，诸如分组运算、排序运算、集函数运算、交并差运算等，并且查询树便于优化和查询交换,语法分析器的输出就是一棵查询树(如图 3.2)，它是一个 SQL 语句的内部表现形式，这时组成该语句的每个部分都是分别存储的，封装在各个相应的处理类中。在 SQL92 中一个标准的查询语句形式如下所示：

```
SELECT {ALL | DISTINCT} expression[,...]
   [FROM table [alias][, ...]]
   [WHERE condition]
   [GROUP BY column [,...]]
   [HAVING condition [,...]]
   [{UNION [ALL] | INTERSECT | EXCEPT} select ]
   [ORDER BY column [ASC | DESC | USING operator] [,...]]
```

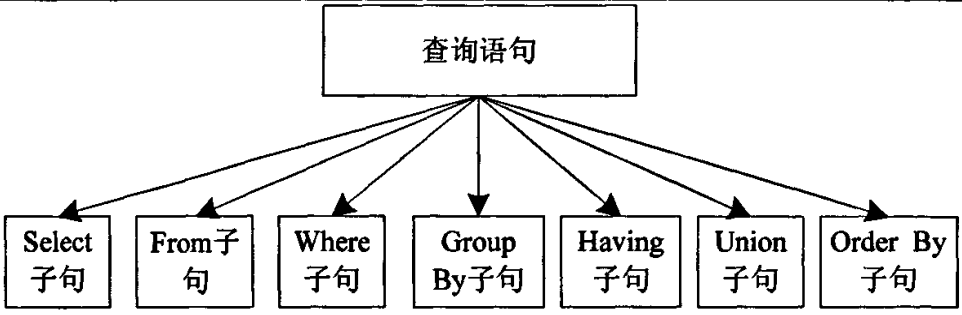


图 3.2 查询语法树

3.3.1.2 查询分解预处理

语法分析器进行语法分析之后，查询树在内存中建立起查询语法树。在进行全局查询分解和转换之前，有必要对查询进行一些预处理操作，主要包括语义检查和查询语句规范化处理。

在查询树建立之后，我们可以肯定：用户输入的全局查询语句在词法和语法上都是正确的。但是我们还是不能肯定的说用户输入的就是一条正确的语句，这是因为语句在语义上可能是不正确的，例如：用户要查询的全局表是不存在的或要查的字段是表中没有的。语义检查的主要目的就是确保用户输入的语句在语义上也是正确的。

语义检查主要分为两种类型：

1) 存在性检查：存在性检查主要是检查用户在全局查询语句中所涉及的对象是否在全局模式文件中已经定义。例如：用户输入查询 select * from a where a.age> 23；在执行存在性检查的时候就需要在模式信息中查询表 a 是否存在以及表中是否有 age 字段。
2) 类型检查：涉及二元比较运算的两个操作数是否类型一致或不一致但可以相互转换。在比较的时候，同种类型的操作数显然能够比较，但是不同类型的操作数是否可以比较就要看情况了，比如：整型数和字符串是没有办法比较的，但是整型和浮点数就是可以通过转换之后来比较的。

在 UQS 中，由语法分析器对用户提交的 UNION 查询语句进行语法正确性检查，然后由查询分解器检查语义的正确性，主要需要检查的内容有：

1) 表的存在性检查；
2) 表属性存在性检查；
3) UNION 查询语句各子句表的分布域一致性检查；
4) UNION 查询语句各子句属性类型一致性检查。

3.3.1.3 查询分解原则

多数据库系统的 UNION 查询分解包括全局查询分解和全局子查询分解。

(1) 全局查询分解原则

全局查询分解的目的是把涉及多个全局表的查询分解为只对单个全局表的全句子查询和进行中间结果合并处理的全局子查询。从查询效率和正确性考虑，全局查询分解应遵循如下原则：

- 分解之后的每个全局子查询只能针对一个全局表/视图进行查询。
- 尽可能把查询条件下发给全局数据库，减少子查询所返回的无效数据量，提高查询效率。
- 多个全局表之间的连接条件是不能下发的，要作为无法下发的条件最后由全局结果合并器在结果合并的时候执行。
- 全局查询中的嵌套子查询不能下发，需要在本次查询执行之前先计算出结果。
- 没有不能下发的查询条件的时候，就可以跳过全局子查询的分解，直接分解为局部子查询。

(2) 全局子查询的分解原则

全局子查询分解的目的是将多个 LDBMS 的全局子查询分解为一组能在单个 LDBMS 上执行的局部子查询。从查询效率和正确性考虑，全局子查询的分解应遵循如下原则：

- 每个局部子查询只能涉及同一局部数据源上的表/视图。
- 查询条件和选择项都要完全下发给局部数据源执行。
- 查询分割以数据源为单位，而不是以全局子查询涉及的数据库表为单位，对同一个局部数据源的查询分解在同一条语句中。

3.3.1.4 基于查询任务树的并发调度体系

UNION 查询调度实际上是按照某个查询计划分派查询任务以便完成中间结果合并的过程，需要充分利用多数据库系统的分布性来实现各个子查询任务的并发执行以及子查询结果的并发组装。在 UNION 查询系统中，将一个全局 UNION 查询操作划分为多个同步执行的任务，通过执行任务树，得到最终结果。

在系统中提交全局查询 UNION 语句 Q：

Q：select id from table1 union select num from table2;

如果全局表 table1 对应三个局部数据源 D1、D2、D3，全局表 table2 对应两个局部数据源 E1、E2，查询语句 Q 的任务分派如图 3.3 所示：

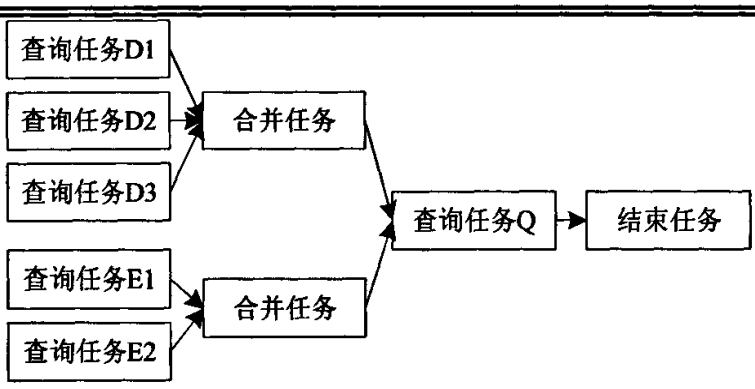


图 3.3 UNION 查询任务分派图

3.3.2 UNION 查询优化

在查询优化的研究中，我们经常把查询优化算法所产生的表示查询执行步骤的过程称为查询执行计划。查询优化涉及到三个方面的问题：第一是查询执行计划的表示模型和查询执行计划空间的估计问题；第二是查询执行计划的复杂性模型问题；第三是查询执行计划空间的搜索问题。

查询优化通常基于下列事实：一个查询通常有若干语义上等价的关系代数表达式，这些表达式只在执行代价上有差别。我们的目的就是为了找出那个执行代价最小或者说是接近最小的表达式。

并行查询优化问题可以形式地定义如下：给定一个查询 Q、一个基于确定的查询执行计划表示模型的查询执行计划空间 S 和一个查询执行计划复杂性模型 Cost(P)，如何在 S 中搜索出一个查询执行计划 P_0 使得 $Cost(P_0) = MIN_{P \in S}\{Cost(P)\}$。由于查询优化问题是一个十分复杂的问题，很难给出一个具有多项式时间复杂性的求解最优查询执行计划的算法。

查询优化通常都是和数据分布放在一起考虑的，在大规模事务处理系统中，数据的分布对划分表来说，支持三种一维数据划分方法：round-robin、hash 和 range 划分。对复制表来说，复制表在并行域内的每个节点有一个完整而相同的备份。而且数据划分存储下去以后，在进行查询的时候，数据再也不进行重新分布了，即在查询的时候数据不会在各个节点之间通过网络传输。

大规模事务处理系统的 UNION 并行查询处理与优化流程图如图 3.4，首先由语法分析器生成语法分析树，然后按照代数优化的等价原则，优化生成的全局查询树。然后把查询树分解为子查询语法树并将生成的子查询规划发送给各个节点分别执行，由于各个节点是分别独立的，因此，各个节点对发过来的子查询规划分别进行优化，这一步优化我们要充分挖掘商业数据库自己的查询优化潜力，所以这一步的查询优化有一些商业数据库自己作的，另外我们还利用缓存等手段提

高响应速度，这一点在后面有详细的介绍。最后，结果合成器再把各个子节点执行完的结果进行归总，返回给用户。

下面我们会详细地讨论 UNION 查询优化中的几项关键步骤。

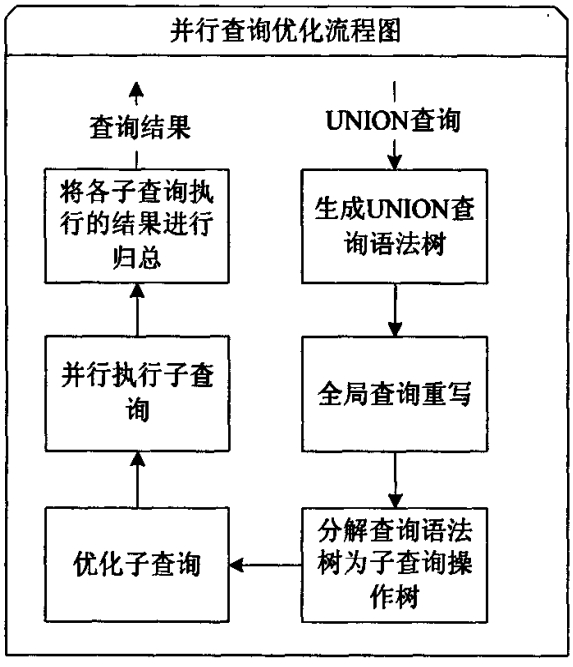


图 3.4　大规模事务系统中 UNION 并行查询优化流程图

在 UNION 查询系统中，利用语法分析器分析查询语句是处理查询语句的首要步骤。而获得正确的并行查询则需要查询处理器提取 UNION 查询语句的并行敏感信息，在此基础上变换语句从而保证最终的子查询语句能够在单个数据库节点机器上成功执行。

3.3.2.1 全局 UNION 查询重写

通过语法分析以后，每一个 UNION 查询对应一棵语法分析树，这时的查询语言还是全局查询语言。我们需要对全局 UNION 查询语句进行重写以提高查询以及结果合并的效率。不同的全局查询重写对应不同的系统性能，为达到优化系统性能的目的，需要确定出一个执行计划，说明需要访问哪些局部数据库，如何拼装中间结果，在哪个站点执行全局处理等。

3.3.2.2 子查询重写

经过全局查询重写和分解后，已经得到了可以并行执行的子查询规划，但是这时得到的 UNION 子查询并没有经过任何的优化，在 UNION 查询系统中通过对分解后的 UNION 子查询进行代数优化来减少查询节点本地的开销和网络的传输量，下面将讨论如何进行对子查询进行代数优化。

所谓代数优化指的是利用关系代数的一些基本定律，如交换律、结合律、幂等律等把用户的 SQL 语句变换成等价的关系代数表达式。从理论上来说，代数优化与数据分布方法无关，完全可以在对 SQL 语句进行分析的时候完成。但是，由于代数优化的结果很可能与数据分布的现状矛盾，使得代数优化的结果为后来的分布优化所推翻，因为，我们可以把这两者放在一起考虑。

SQL 语言是关系操作的组合，对任何一条语句，我们都可以画出一副关系操作树来。代数优化就是利用等价变换来重构这颗树，改变运算执行顺序以达到优化查询的目的。常用的等价变换有：交换律、结合律、幂等律、分配律、因子提取等等。借助于这些定律，我们就可以对用户查询请求所构成的查询树进行等价变换，以达到优化的目的。在这里，等价变换的基本思路就是把中间结果较小的运算先作，即把这个操作往树的叶节点推。变换的通用准则为：

a) 利用一元运算（投影和选择）的幂等律把一元运算归并为一个运算；
b) 尽可能把操作树的各种选择和投影往树的叶节点推；
c) 把各种选择操作向下推到树的叶子上，然后按照数据分布方法（可将之描述成限定代数式），分析用户操作树的叶子，将和数据分布方法矛盾的叶子用空集合表示；
d) 利用限定关系表达式判别连接操作，如果判断为矛盾，则用空集合表示；

3.3.2.3 子查询计划生成与执行

通过语法分析和变换，UNION 查询语句变成了可以并行执行的子查询语句。为了实现高效的并行查询，系统通过多线方式分别发送查询语句给不同并行数据访问服务，由每个并行数据访问服务连接指定的数据库节点执行子查询。

3.3.3 UNION 查询结果合成

在 UNION 子查询执行完成之后，所有子查询结果都保存在各个数据查询服务器中。但是，各个子查询结果集还不满足用户原始查询要求。因此在用户提取查询结果时必须合成各个子查询集。

一般对查询结果集的操作有移动游标、获取当前记录、批量提取记录等。以上操作对于单个数据库的查询来说非常简单。查询服务器只需要对查询的游标进行处理就可以了。而在并行环境下需要依次处理多个子查询结果。如果查询语句中没有跟并行相关内容，那么各个子查询结果就是用户所需结果。当查询语句中出现 DISTINCT，GROUP 函数、GROUP BY 子句或 ORDER BY 子句时都不能简单地合并查询结果，需要根据语句分析阶段的结果执行。UNION 查询正是必须进行这类操作的典型应用。

对 UNION 查询结果集进行处理的重点是定位当前记录和取得当前记录的每

个域值。定位当前记录就是确定合成当前记录涉及的各个子查询记录。利用定位当前记录的结果就能确定当前记录的各域值。其他操作都可以在此基础上实现。UNION 查询结果处理按照查询语句中内容分类处理：

1. GROUP BY 子句：通过比较各个子查询结果的记录的分组属性值定位当前的记录。
2. ORDER BY 子句：通过比较各个子查询结果确定当前记录。
3. GROUP 函数：根据各个子查询结果计算当前记录。
   a) MAX、MIN：直接比较各子查询结果；
   b) SUM 、COUNT：把各子查询结果相加；
   c) AVG：添加 COUNT 函数，把子查询结果总和除以结果个数得到平均值；
4. 带 GROUP 函数的 HAVING 子句：首先计算得到 GROUP 函数值，然后利用 HAVING 条件筛选记录。

3.4 UQS 查询后处理

查询后处理是将全局子查询得到的子查询结果以及中间结果转换成全局结果的过程。UQS 查询后处理优化方案如图 3.5 所示。

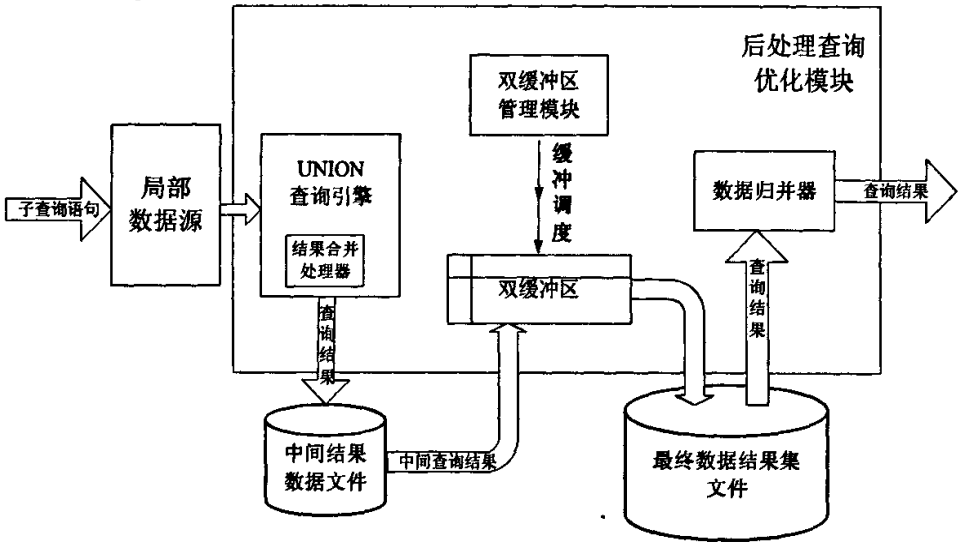


图 3.5 改进的查询后处理优化方案

在查询后处理方案中，UQS 在结合了传统二路连接结果合并算法的基础上，提出了一套新的查询优化方案，其主要包括 UNION 查询引擎、双缓冲区管理模块以及数据归并器三大部分。UNION 查询引擎主要是通过增量多路连接策略来获得子查询语句的中间结果，并写入中间结果数据文件当中，该策略在二路连接算法

的基础上进行了优化，进一步降低了查询引擎的运算次数，很大程度上提高了查询性能；双缓冲管理模块通过双缓冲并行处理策略来减少数据文件之间的网络传输时间，降低系统耦合度；而数据归并器则负责通过多路并行归并算法来进行数据文件的结果合并，利用外排序算法来缩短查询的响应时间。

3.4.1 UNION 子查询结果提取

当 UNION 子查询返回查询结果时，需要通过各个局部数据源进行子查询结果的提取，选取良好的子查询提取策略，可以避免无谓的空等，有利于调高结果合并的效率。

3.4.1.1 传统的子查询结果提取

传统的子查询结果提取方法是这样的：系统中只有一个结果合并处理器，局部数据源代理的结果返回策略是要取得所有结果之后才会返回给结果合并处理器，结果合并处理器在有两个局部数据源结果到达的时候就开始合并，并把合并的中间结果保存起来供与下一个局部数据源结果合并之用，其操作见图 3.6。 由于以前只是把全局查询语句简单的用替换方法分解成对应的局部查询语句，每一条局部查询语句中可能包含对局部数据源中多个表的查询，因此，局部数据源代理返回的结果是没有办法有序的，在结果合并处理中做连接所使用的也就只能是外连接，这样对一个 M 行的表和一个 N 行的表要做 M*N 次的笛卡尔积的运算。当 M 和 N 都万条级的情况下，运算量大的惊人。

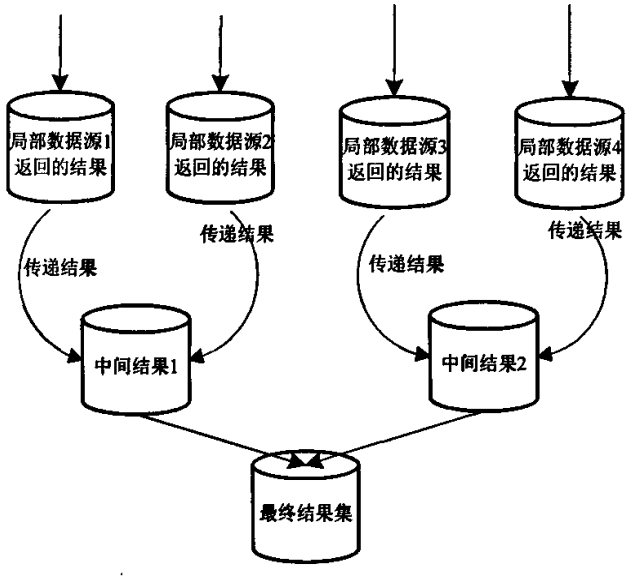


图 3.6 传统数据库结果合并操作示意图

3.4.1.2 高效的子查询结果提取

传统的子查询结果提取方法是把中间结果都提取到查询服务所在的中央节点，这种方法会占用大量的网络、CPU 等资源。为了减少网络传输和大文件排序所引起的磁盘 I/O,我们采用增量多路连接的方法来提取子查询结果，其操作见图 3.7：

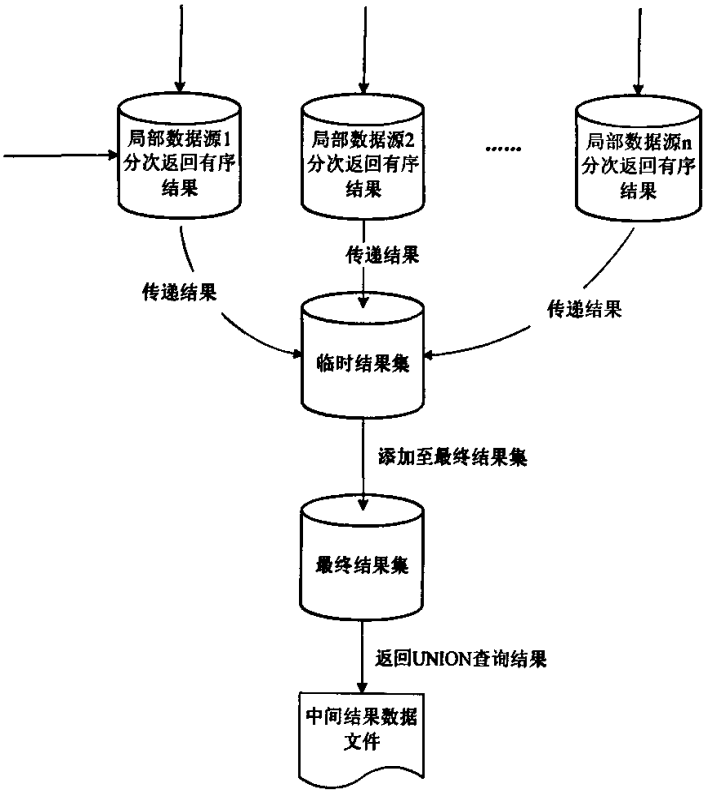


图 3.7 增量多路连接策略示意图

局部数据源代理的结果返回策略是取得部分结果之后就返回给结果合并处理器，结果合并处理器在所有局部数据源代理都有结果到达的时候就开始合并，并把合并的临时结果保存起来，当某个局部数据源有新的数据到达时，就与其它的局部数据源以前返回的结果一起合并，并加入到最后结果之中。

在分散的并行数据库中进行 UNION 查询，如果按照传统数据库的方式，由于数据是未排序好的，必须将数据集中到一个数据库中来进行结果的合并。这样对一个 M 行的表和 N 行的表就需要做 M*N 次的笛卡儿积的运算。

由于采取增量多路连接策略的优化算法来获得查询后处理的中间结果，并将查询结果写入数据文件中。在查询重写过程中将各个子结果集进行排序操作，而系统中则还是只有一个结果合并处理器，局部数据源代理的结果返回策略是取得部分结果(比如 1000 条)之后就返回给结果合并处理器，结果合并处理器在所有局

部数据源代理都有结果到达的时候就开始合并，并把合并的临时结果保存起来，当某个局部数据源有新的数据到达时，就与其它的局部数据源以前返回的结果一起合并，并加入到最后结果之中。一起合并的好处是不需要做笛卡尔积的运算，把对一个 M 行的表和一个 N 行的表的合并运算的计算次数从 M*N 次降低到 M+N 次。

3.4.2 UNION 查询结果集归并

通过子查询提取 UNION 结果时需要进行结果集的归并，通过选取简单有效的归并策略可以减少大量的网络开销和计算开销。

3.4.2.1 结果合并处理器规模

1. 一个结果合并处理器的情况

只有一个结果合并处理器时，海量并行数据库中结果合并策略是与局部数据源代理的结果返回策略息息相关的，这是因为如果局部数据源代理是要取得所有结果之后才会返回给结果合并处理器的话，最优的策略是把所有已经返回给结果合并处理器的结果先行合并，再把这个临时结果和后来的结果合并。

2. 多个结果合并处理器的情况

这样就可以实现临时结果合并的并行处理，例如：有四个局部数据源代理 A、B、C、D。假设局部数据源代理都自带了结果合并处理器。首先当 A 和 B 取得所有结果后，如果 B 的数据量小，那么可以让 B 把数据传给 A 来合并，临时结果保存在 A 处；其后，C 和 D 又都取得了结果，假设 C 传给 D 的代价小，就让 D 来进行结果合并处理的工作。最后把 A 和 D 中的临时结果合并作为级后结果。

但是，在实际上我们很难估计出在哪个局部数据源进行合并工作效率更高，因为这里面不只是要考虑传送数据量的多少，还有机器的计算能力、与系统中其它机器的通信能力等很多因素要考虑。另外，调度实现起来也很复杂。

由此可见，在多个结果合并处理器的情况下，局部数据源的分次传送就对提高系统性能没有多大帮助了，因为多个结果合并处理器的并行计算是需要局部数据源数据完全到达后才能工作的。

3.4.2.2 双缓冲并行处理

针对海量数据库系统所特有的海量数据的特点，在 UNION 查询服务中采取了双缓冲的方式来进行中间结果集的合并。这样不但可以增加查询后处理的并行度，通过并行的方式来达到性能最大化的目的，还可以通过该技术降低数据库服务器与 UNION 查询服务之间的耦合度，实现模块之间的松耦合。双缓冲区模型的结构如图 3.8 所示：

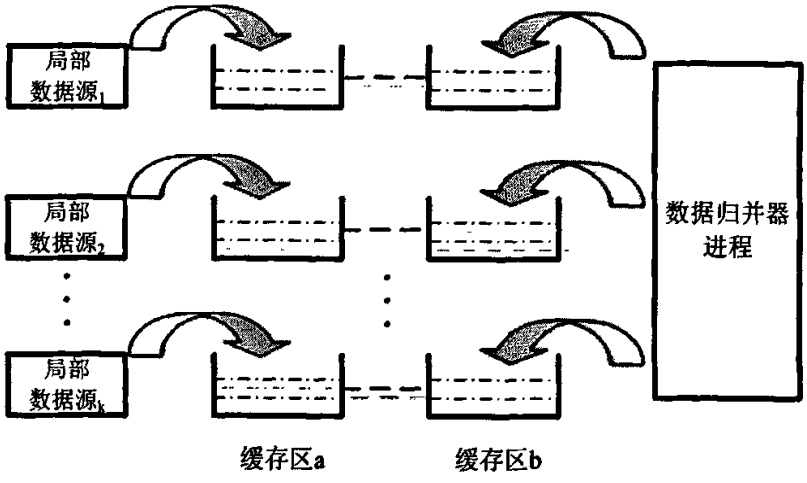


图 3.8 双缓冲区模型示意图

图中，局部数据源 1，局部数据源 2，...，局部数据源 k 每个进程各自拥有两个缓冲区。进程之间没有任何关联，各自将查询结果放入缓冲区 a 中。若缓冲区 b 不为空时，则缓冲区 a 继续响应返回查询结果的请求。当发现缓冲区 b 为空时，停止往缓冲区 a 中写入查询结果，将缓冲区 a 推到缓冲区 b 的位置，然后再取一个空缓冲区作为缓冲区 a 继续响应返回查询结果的请求。数据归并器进程只需处理缓冲区 b 中的查询结果即可，不需要与多个局部数据源之间进行交互，加快了处理文件的速度。

3.4.2.3 多路并行归并算法

针对应用系统具有规模大、速度高、持续运行等特点，当数据规模超过了内存限制，就必须采用一定的外排序算法，将待排数据分片读入内存进行处理。在诸多外排序算法当中，多路并行归并算法具有归并操作简单,占用资源少,有利于算法效率的提高等特点。我们可以利用该算法进一步优化查询后处理。

在前一小节提到的双缓冲并行处理策略里，为了对缓冲区 b 中的数据进行排序操作，我们将结合多路并行归并算法对 P 个节点的数据进行归并输出。其示意图如图 3.9 所示。

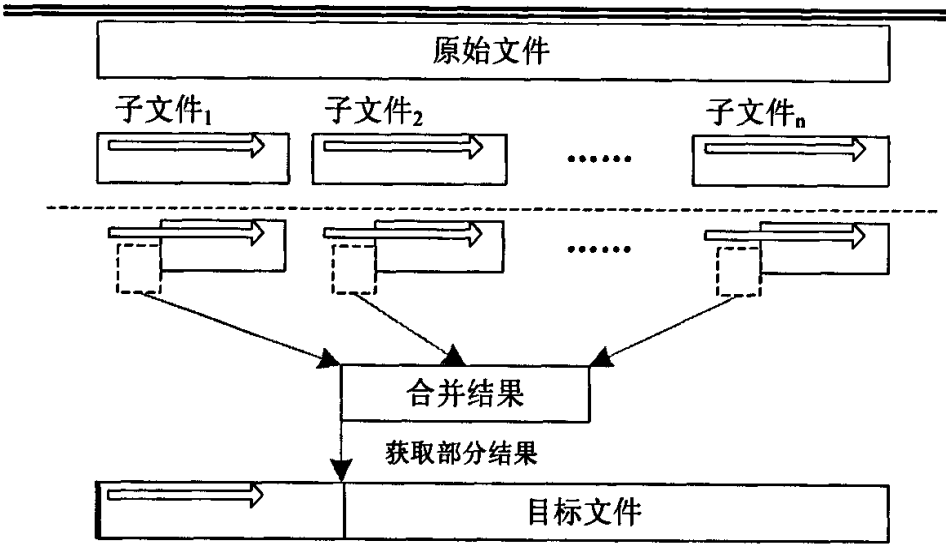


图 3.9 多路并行归并算法示意图

由于排序的第一个结果出来的比较早，所以数据归并器可以先把已排序好的结果首先呈现给用户，这样可以获得快速的查询响应。

3.5 UNION 查询服务连接池

如果 UNION 查询对数据库的访问不是很频繁，那么查询对数据库连接的使用最简单的方法就是需要时就建立，用完就关闭。但是如果 UNION 查询对于数据库的访问很频繁，每次使用都去建立和关闭数据库连接将很大的影响系统的性能，这是由于连接数据库不仅要开销一定的通讯和内存资源，还必须完成用户验证、安全上下文配置这类任务，因而往往成为最为耗时的操作。

一个数据库连接大致是以下几项组成的：数据库驱动程序、数据连接的 URL、数据库用户名、数据库密码、初始化连接数、最大连接数、连接的最大空闲时间、取连接时若无可用连接的最大等待时间等。

由于大规模事务处理系统的特点是大量用户的频繁访问，为了减少用户进行 UNION 查询的连接开销，每个数据库用户维护一个连接池，连接池管理器管理所有用户的连接池。当客户请求数据库连接时，首先看该用户的连接池中是否有空闲连接，这里的空闲是指，目前没有分配出去的连接。如果存在空闲连接则把连接分配给客户，并作相应处理，主要的处理策略就是标记该连接为已分配。若连接池中没有空闲连接，就创建一个数据库连接给客户，如果数据库连接耗尽就等待一段时间再看是否有空闲的连接或是否可用创建新连接。

对于很长时间(系统或用户配置的一段时间)没有用到的连接就自动关闭。在系统关闭时释放所有资源，关闭所有连接。图 3.10 是本系统中实现的连接池示意图。

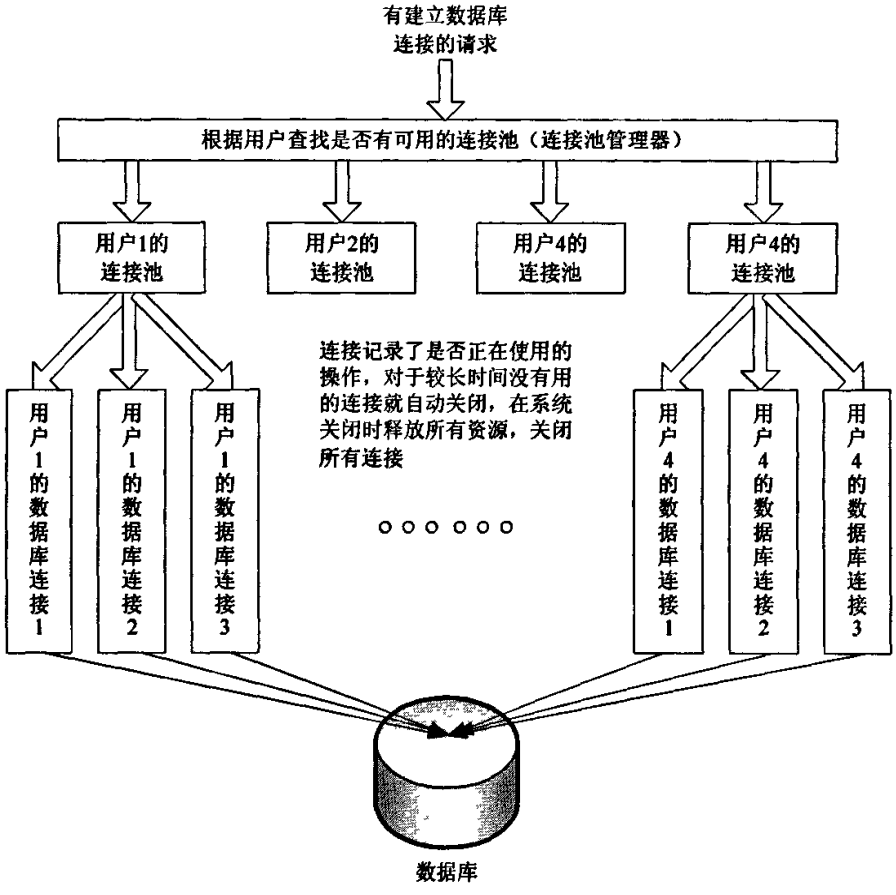


图 3.10　数据库连接池示意图

第四章　UQS 系统实现

本章根据第三章中 UNION 查询系统设计的基本思想，实现了面向大规模事务处理系统的 UNION 查询服务 UQS，本章将对实现过程中的问题展开详细讨论并给出具体实现。

本章结构组织如下：第一节详细介绍了 UNION 查询服务的系统结构以及工作流程，第二节对接收到的用户 UNION 查询的处理过程进行了具体的描述，第三节描述了对查询优化的实现过程，最后一节介绍了查询后处理的具体实现。

4.1　UQS 的组成

4.1.1 UQS 的系统结构

本文中 UQS 是基于分布式中间件 CORBA 实现的，各组成部分之间通过 ORB 进行通讯并采用多线程方式工作，而多线程则是通过 CORBA 的线程库实现的。图 4.1 显示了 UQS 的主要功能组件，UQS 主要分为四个模块：语法分析部分、查询分解部分、查询处理部分和结果合成部分：

本章以下内容将详细介绍上述四大模块的设计和实现细节。

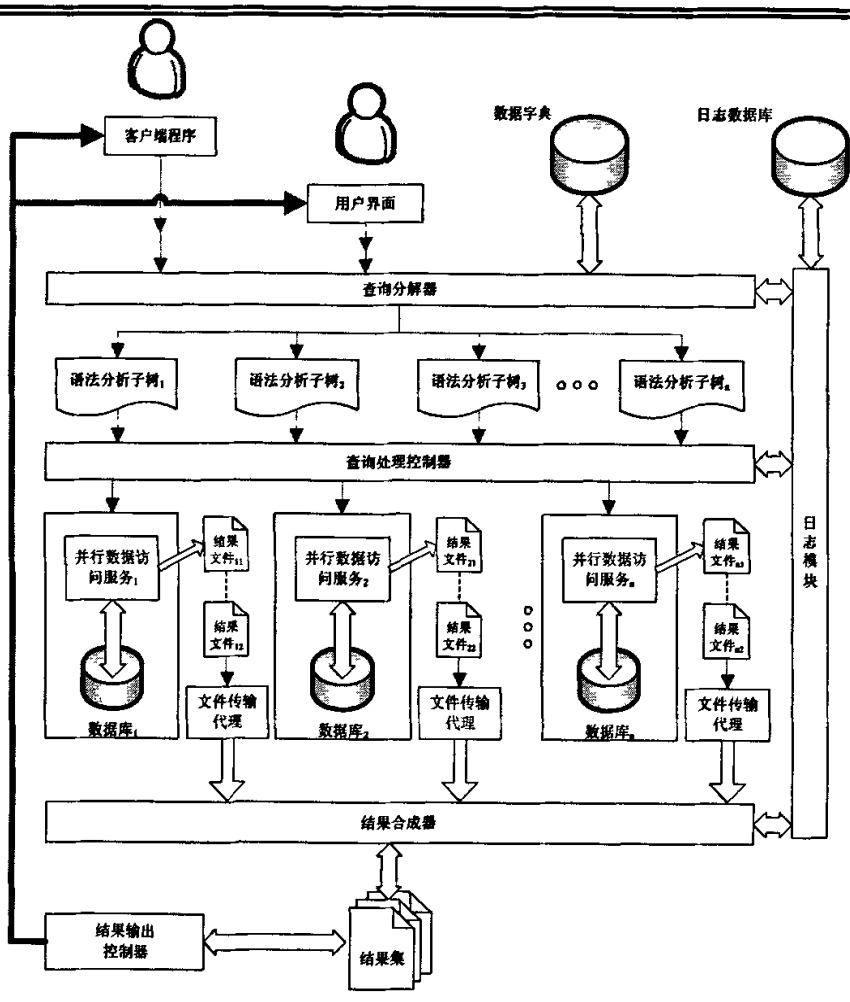


图 4.1 UQS 模型

4.1.2 UQS 的组件交互

UQS 各组件之间的交互情况如图 4.2 所示，IResultSet 类使用 CDictionary 类、CConnectionPool 类和 SyntaxParser 类进行 UNION 查询的分析，通过 IResult 类取结果并使用 CSetTimeOutThread 类判断查询语句是否超时。

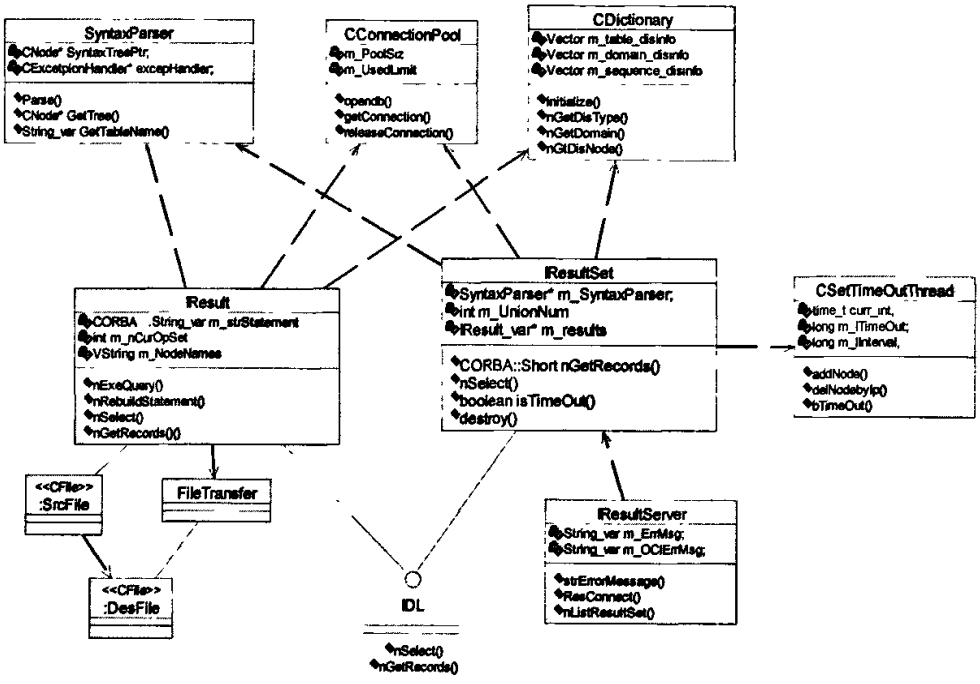


图 4.2 UQS 各组件交互图

4.1.3 UQS 的整体流程

当用户需要进行 UNION 查询时，调用 UNION 查询服务对象 IResultSet 对象的 nSelect()方法进行查询，每一个用户提交查询语句时需要建立连接，并通过数据字典获取查询的基本信息，最后 UNION 查询服务将处理后的查询语句提交给并行数据访问服务对数据库进行查询。

UNION 查询服务的整体流程图如图 4.3 所示：

1) 用户初始化 UNION 查询服务；
2) UNION 查询服务连接数据字典，获取数据字典信息；
3) UNION 查询服务初始化连接池；
4) 用户将查询语句提交给 UNION 查询服务；
5) 查询服务对查询语句进行语法分析；
6) 查询服务将分析得到的语法树进行分解，得到多棵语法子树；
7) 获取并行数据访问服务的连接；
8) 增加连接引用；
9) 在每个并行数据访问服务上执行子查询语句。

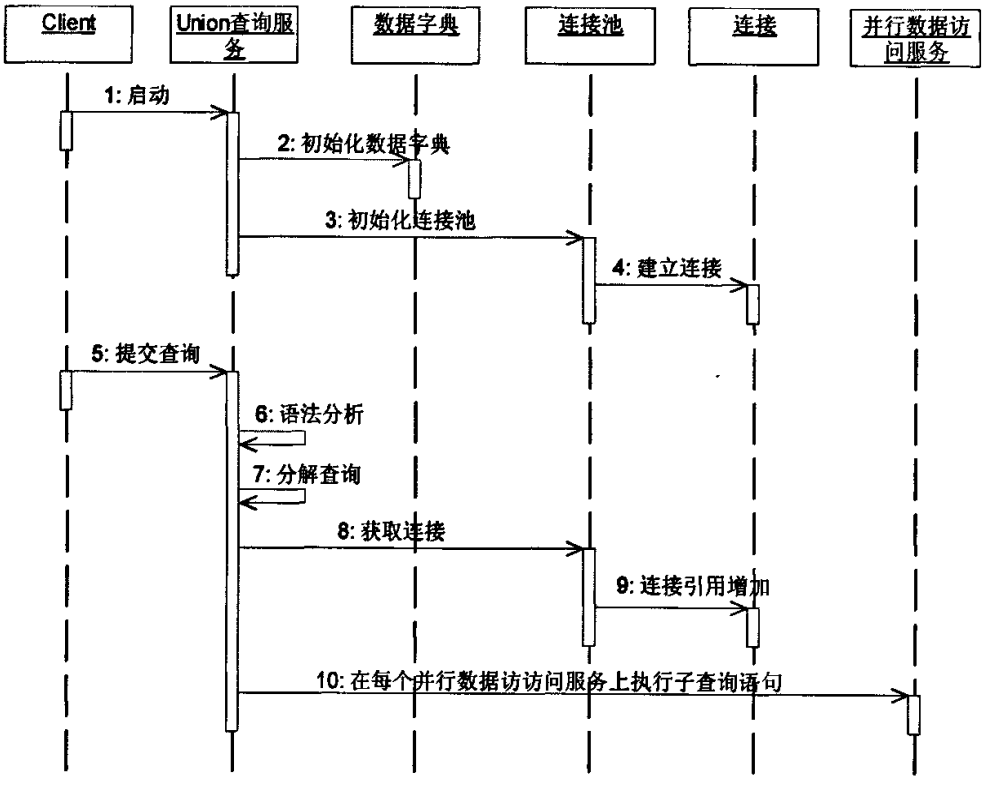


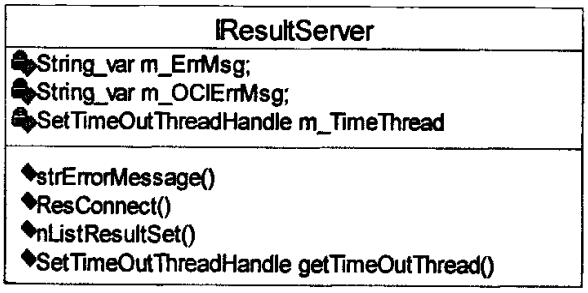
图 4.3 UNION 查询整体流程图

4.1.4 相关接口说明

本节主要描述UNION查询服务对用户应用程序提供的查询接口，包括UNION查询借口的功能和参数的详细说明。

1) IResultSetServer 类

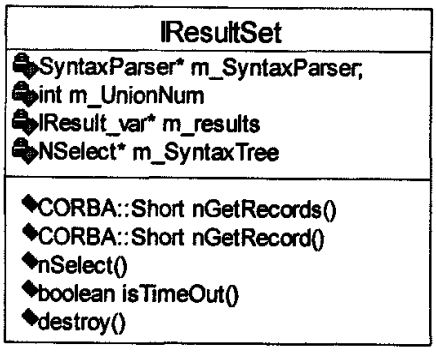
IResultServer 对象的主要功能包括获取 IResultSet 对象，判断连接时的错误信息以及判断 UNION 查询是否超时。其主要的成员变量和内部方法有：



- string strErrorMessage(in short nErrorId)： 根据错误 id 返回错误信息
- IResultSet ResConnect(in string strClientID, in short timeOut)：连接并行查询访问服务
- long nListResultSet(in string strClientID, out ResultSetSeq ResultSets)：返回服务方所有没有释放的 nUserID 的结果对象，如果 nUserID 为 0，则返回服务方所有结果对象

2) IResultSet 类

IResultSet 类用来进行 UNION 查询的分析与处理，通过并行数据访问服务来获取各个子查询结果集合并进行多路并行归并，将得到的最终结果返回给用户，其主要成员变量和内部方法有：



- short nGetRecord(out Record value)：取当前结果记录，并向后移动一条记录
- short nGetRecords(in long nCount, out Records value)：取当前记录起的 nCount 条结果记录，并向后移动 nCount 条记录
- short nGetFieldCount()：取结果纪录的属性个数
- short nGetFieldTypeByIndex(in short nIndex)：取结果纪录的属性类型，属性类型在 Predefine.h 中定义
- short nGetFieldNameByIndex(in short nIndex, out string strName)：取属性名
- short nGetFieldValueByFieldName(in string FieldName, out string

strValue)：根据属性值取当前结果记录值

- void destroy()：释放结果集对象
- short nSelect(in string strStatement, in short nBufSize, in short timeOut)：执行查询（select）语句 strStatement

4.2 UNION 查询分解模块的实现

4.2.1 UNION 查询分解流程

UNION 查询服务对用户提交的查询通过语法分析器分析后得到 UNION 查询语法分析树，经过验证语法与语义的正确性，将语法树分解为多个独立的语法子树并提交各查询优化器，其主要流程如图 4.4 所示：

用户将查询语句提交给语法分析器进行语法分析；

1) 语法分析器分析得到 UNION 查询语句子查询的数据；
2) 建立与子查询数目相等的数量的并行数据访问服务的连接；
3) 通过语法分析器得到查询语法树；
4) 向数据字典获取查询信息；
5) 通过语法分析与数据字典得到表的域分布信息；
6) 分解语法树为多棵子语法分析树；
7) 将分解后的语法分析树还原为查询语句；
8) 将子查询语句提交给查询优化器进行优化。

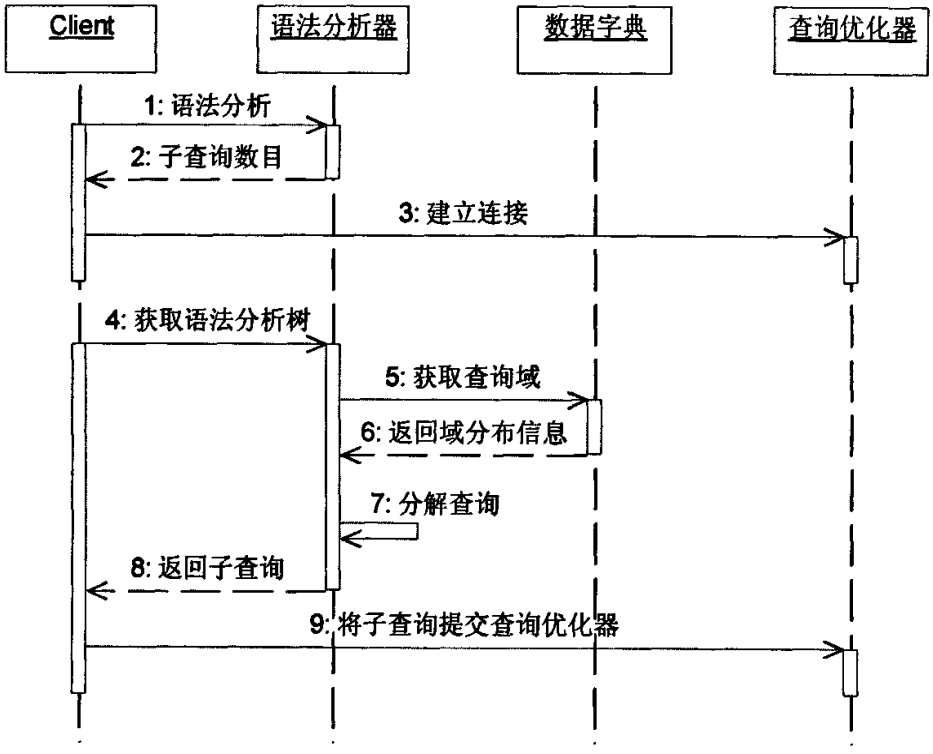


图 4.4 UNION 查询处理流程

4.2.2 相关接口说明

对于每个查询语句，UNION 查询服务用一颗语法树 SyntaxTree 来表示,语法树用语法分析器 CSyntaxParser 来定义，通过语法分析器对查询语句进行分析并获取查询的基本信息并构造语法树。

并行查询语言语法分析器的类图与接口如下所示：

- void Parse(String_var strStatement) throw (ParseError)：进行语法检查并生成语法树；
- CNode* GetTree(String_var strStatement) throw (ParseError) ：进行语法检查、分析并返回语法树的根结点
- String_var GetTableName()：获得表名
- EnNodeType GetNodeType()：获得操作类型，根据类型分发到下层服务器

- void CheckSyntax(String_var strStatement) throw (ParseError)：检查语法是否正确，不生成语法树

4.3 UNION 查询优化模块的实现

4.3.1 UNION 查询优化方法

UNION 查询的优化分为两部分：一为全局查询的优化，主要是在每棵子查询语法树上加入去重与排序操作，以减少子结果集合合并阶段的网络传输开销；二是对分解后的子查询进行优化，以更大程度的利用并行数据库来提高查询效率。

4.3.1.1 全局 UNION 查询优化的实现

在 UNION 查询服务中，其查询重写算法的描述如下：

1) 在每个子查询语句中打上 DISTINCT 标记；
2) 分析第一个子查询语句，根据其 FROM 子句中表名取得表的分布方式；
3) 取得 UNION 语句的 ORDER BY 子句，若不为空，则根据分析第一个子查询语句所得表的分布方式获取该属性在结果集的属性序号 i；
4) 取得各个子查询语句的 SELECT 子句；
5) 若 UNION 语句的 ORDER BY 子句为空，则将 SELECT 子句中属性依次加入该子查询的 ORDER BY 子句；
6) 若 UNION 语句的 ORDER BY 子句不为空，则将 SELECT 子句中第 i 列的属性加入 ORDER BY 子句，然后将 SELECT 子句中其余属性添加到 ORDER BY 子句中；
7) 查询重写结束。

4.3.1.2 子查询优化的实现

分解后的子查询语句的分析变换流程如下：

1) 通过语法分析器取得 FROM 子句、WHERE 子句、GROUP BY 子句、ORDER BY 子句和 HAVING 子句。
2) 根据 FROM 子句中表名查询全局数据字典，取得表的分布方式和分布的域。
3) 分析 WHERE 子句，提取条件涉及的属性信息，检查参与连接的表的分布方式是否与连接条件一致，包括连接条件中的属性是否为表的划分属性和两个连接的表是否采用相同的划分策略。当出现不一致情况时中止查询。
4) 分析选择列表，考虑下列条件：
   - 没有 GROUP 函数，不含 DISTINCT，不用变换语句。
   - 没有 GROUP 函数，含 DISTINCT，需要变换，添加按照选择列表中

属性排序语句。需要删除各子查询返回结果中重复值。

- MAX、MIN，当没有 GROUP BY 语句时不用变化语句。如果有 GROUP BY 语句时，增加按照 GROUP BY 子句属性排序的语句。
- COUNT、SUM 不含 DISTINCT：不需变换语句。
- AVG 不含 DISTINCT：需要变换，AVG(a)变为 COUNT(a)，SUM(a)。

5) 分析 GROUP BY 子句和 ORDER BY 子句，记录分组属性和排序属性。
6) 分析 HAVING 子句，如果包含 GROUP 函数，那么就必须将 HAVING 子句中 GROUP 函数添加到选择列表，删除 HAVING 子句。然后按照 4 处理。将 HAVING 子句保留到合成结果的最后阶段筛选记录。

4.3.2 UNION 查询优化流程

UNION 查询优化流程如图 4.5 所示：

1) 查询处理器将子查询提交给查询优化器；
2) 查询优化器通过语法分析得到子查询语法树；
3) 通过数据字典获取子查询语句表的分布信息；
4) 对子查询语句进行重写优化；
5) 将优化后的子查询提交给并行数据访问服务进行查询。

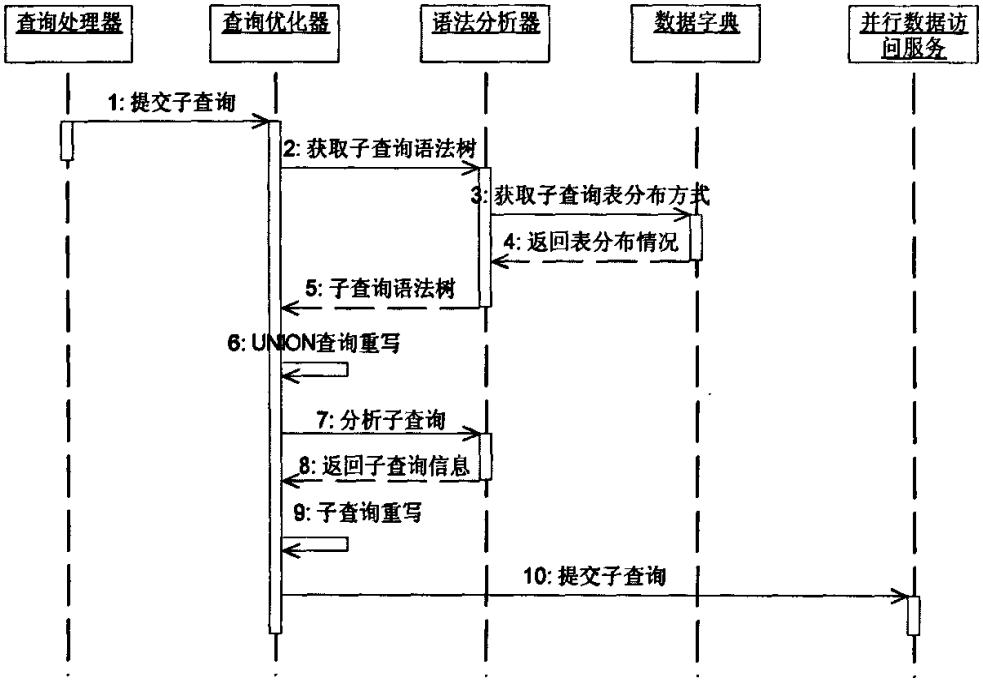


图 4.5 查询优化流程

4.4 查询后处理模块方案

4.4.1 查询后处理算法的实现

UNION 查询服务在查询后处理中通过增量多路连接策略来获得子查询语句的中间结果，并写入中间结果数据文件当中，该策略在二路连接算法的基础上进行了优化，进一步降低了查询引擎的运算次数，很大程度上提高了查询性能，并采用多路并行归并算法来进行数据文件的结果合并，利用外排序算法来缩短查询的响应时间。

4.4.1.1 增量多路连接算法以及复杂度分析

首先，我们来看一个三路连接的例子：假设要做自然连接 Q(A,B) X R(B,C) X S(C,D)，根据自然连接的交换律和结合律，我们可以按照任一顺序做两次连接，其中最差的方法是先做 Q X S，因为这实际上是一个笛卡尔积，较好的方法是先做 Q X R 或 R X S.

设关系 Q ,R ,S 的势分别为 N_Q，N_R，N_S，并且连接属性 B 和 C 上 J_B 和 J_C 个不同值分布均匀，那么：

如果先做 Q X S，则中间关系的势为 $N_Q * N_S$；

如果先做 Q X R，则中间关系的势为 $N_Q * N_R / J_B$；

如果先做 R X S，则中间关系的势为 $N_R * N_S / J_C$；

如果用三路连接来做这个连接运算，则花费更小，它的思想是浏览 R 中所有元组 BC，找出 Q 中所有元组 AB 和 S 中所有元组 CD，产生形如 ABCD 的元组作为结果，其过程形式描述如下；

FOR V bc ∈R DO

DO $\sigma_B = b(Q) X \{bc\} X \sigma_C = c(S)$

END FOR

在循环体内，对于 R 中每个元组 bc，生成中间关系的势为 $2 * N_Q / J_B + N_S / J_C$ 并且每做一次都可以释放它所生成的中间关系，因此整个连接的中间关系的势还是不变，显然它比做两次连接生成的中间关系的势小得多。

多路连接算法：

输入：连接 $R X S_1 X S_2 X \ldots X S_N$

其中，1.S_i 和 S_j 没有公共属性(i≠j)；2.R 与所有 S_i 都有公共属性

输出：连接运算结果

算法：

FOR V u∈R DO

FOR i = 1 To n DO

$T_i = S_i X \{u\}$； //T_i 是与 u 在 R 和 S_i 的共同属性上有相同值的 S_i 元组

END FOR I

DO $\{u\} X T_1 X T_2 X \ldots X T_n$

END FOR u;

4.4.1.2 多路并行归并算法

算法 1. 多路并行归并算法

输入：原始的查询结果数据

输出：排序后的完整查询结果数据

算法：

```
void kwaymerge ( Element *r )
{
    r = new Element[k];   //创建对象数组
    int *key = new int[k+1];  //创建外结点数组
    int *loser = new int[k];     //创建败者树数组
    for ( int i = 0; i < k; i++ )
        { //传送参选关键码
```

```
        InputRecord ( r[i] );
        key[i] = r[i].key;
    }
    for ( i = 0; i < k; i++)
        loser[i] = k;

    key[k] = -MaxNum;  //初始化
    for ( i = k-1; i; i-- )    //调整形成败者树
        adjust ( key, loser, k, i ); //调整

    while ( key[loser[0]] != MaxNum )
    { //选归并段
        q = loser[0]; //最小对象的段号
        OutputRecord ( r[q] );     //输出
        InputRecord ( r[q] );      //从该段补入对象
        key[q] = r[q].key;
        adjust ( key, loser, k, q ); //调整
    }
}
```

4.4.2 查询后处理流程

查询后处理的工作流程如图 4.6 所示：

1) 用户提取一定数目的 UNION 查询结果；
2) 按用户要求数目去各并行数据访问服务取结果；
3) 将子结果在缓冲中进行归并；
4) 将最终结果写入数据文件；
5) 将用户要求数目的查询结果返回。

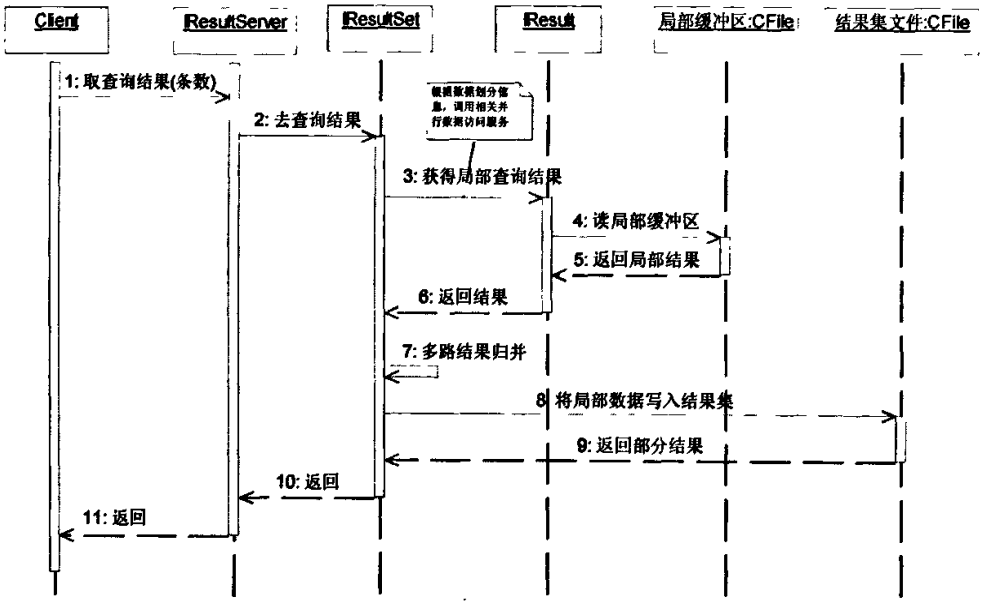


图 4.6　查询后处理工作流程

第五章　测试及分析

5.1　测试系统

当前，针对数据库进行综合测试最行之有效的方式就是采取 benchmark 的方式。面向应用的 benchmark 能够模拟真实的业务负载，以评价系统的当前性能；能够通过设计适当的测试用例并且辅以性能分析技术来预测系统的未来性能；能够通过模拟大规模并发用户重复执行和运行测试，确认性能瓶颈并且优化和调整应用。

在本章，将通过面向大规模事务处理系统的 benchmark 来对 UNION 查询服务进行功能和性能测试。

5.1.1 业务与应用环境

大规模事务处理系统 benchmark 面向的应用场景可以描述为：在一个广泛的地理环境分布的数据传输网络上，分布着一些数据中心，每个数据中心拥有大量（10^8 级别）的注册用户。数据中心的主要业务是为用户提供数据传输服务的同时将用户的短文本进行备份，其中数据为短文本。当发送源用户和接收用户隶属于不同的数据中心时，数据传输过程需要进行数据中心的转发。数据中心管理用户要求对备份的数据进行一系列的查询（注：数据库表的设计、测试语句的设计、数据的分布、业务模式与应用系统一致的，但由于保密的原因，表名称、属性名称进行了修改）。

本文中的 benchmark 的目的是在保留应用本质的性能特征的基础上，在保持系统的资源使用率和操作的复杂度的同时，降低真实系统的操作多样性[28]。真实的应用系统中有大量的查询和其他 DML 操作，其中许多查询因为查询消耗的时间，资源使用率以及执行的频率对于系统性能分析无足轻重而不被考虑。选择的业务操作具有以下特征：

- 操作具有很高的复杂度
- 数据访问模式多样，访问大量的数据
- 访问具有随机性（adhoc nature）
- 操作是互异的
- 查询参数是变化的

根据以上原则，分析应用模型，选取以下操作作为典型操作：

- 文本查询
- 统计查询

- 基于用户标识的查询

在数据传输的同时需要进行数据备份、数据加载等操作以模拟真实运行环境，因此，在多种操作进行的同时，UNION 查询可以在 benchmark 中进行模拟真实的业务负载下的测试。

5.1.2 测试环境与部署

在所有测试用例中，我们都是在大规模事务处理系统的 benchmark 中进行 UNION 查询服务的测试，其部署环境如图 5.1 所示：

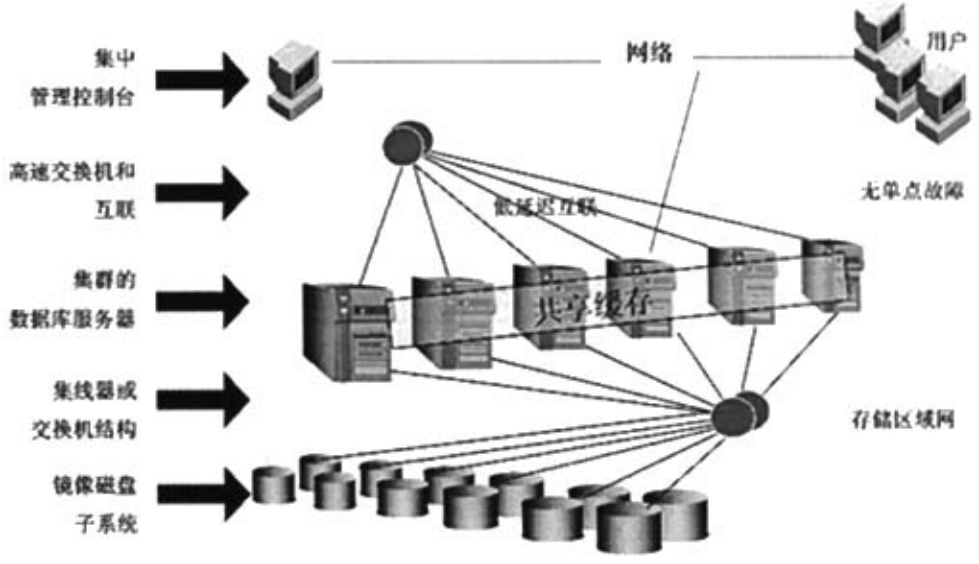


图 5.1 大规模事务处理系统测试环境

其中，数据库系统系统由 10 个 Cluster 构成，为保证系统的高可用性，避免单点故障问题，Cluster 之间通过两套千兆以太网络进行互连，并对应用服务器提供服务，两个交换机互为备份，并可支持负载平衡。

每个 Cluster 包含 6 个处理节点机，6 个处理节点机通过两套内部千兆以太网络实现 Cluster 的内部互连。每个内部以太网络对外提供 6 个互连端口，这 6 个对外互连端口分别接入到两套外部互连网络上。从 Cluster 的外部视图看，每个 Cluster 对外提供 12 个互连端口，分别接入到两套外部互连网络上。千兆网络除了作为外部接入网络外，还可用作系统的管理和监控网络。

其中，每一个 Cluster 都有六个节点服务器，它们通过 SAN 网络共享一个物理数据库。每套 Cluster 的测试环境配置如表 5.1 所列。

表 5.1 测试环境配置

配置项	数量	配置明细
测试驱动节点机	2 台	CPU：Intel Pentinum4 3G， Memory：1G 操作系统平台：Windows 2000 SP4
中间件节点机	2 台	CPU：4* Intel Itanium-2 1.6G， Memory：48G 操作系统平台：RedHat AS3
数据库节点机	4 台	同中间件节点机 数据库配置成 ORACLE 10G RAC 集群
数据库系统		ORACLE 10G release 2
磁盘存储	1 套	存储网络，容量 20TB
StarBus		64 位，4.0 版本
网络环境		千兆以太网
测试数据		根据表结构模拟生成

测试用表 DataA 和 DataB 的结构如表 5.2 和表 5.3 所示：

表 5.2 DataA 表定义

列名	数据类型	约束	描述
TimeA	Date	Not null	数据传送开始时间
Client_FromA	Integer		发送用户 ID
Client_toA	Integer		接收用户 ID
DataCenter_FromA	Integer		发送数据中心 ID
DataCenter_ToA	Integer		接收数据中心 ID
ContentA	Variable text, size 256	Not null	数据内容

表 5.3 DataB 表定义

列名	数据类型	约束	描述
TimeB	Date	Not null	数据传送开始时间
Client_FromB	Integer		发送用户 ID
Client_toB	Integer		接收用户 ID
DataCenter_FromB	Integer		发送数据中心 ID
DataCenter_ToB	Integer		接收数据中心 ID
ContentB	Variable text, size 256	Not null	数据内容

5.1.3 数据生成与查询模板

大规模事务处理系统 benchmark 的数据生成和查询模板主要依据的是真实场景的模拟，以最大程度接近真实应用，更好的反映系统的性能。其中，测试用数据的产生主要依靠数据生成器，而查询生成器则负责根据查询模板产生出接近真实应用的用户查询。

5.1.3.1 数据生成

查询性能极度依赖于底层数据集。查询生成器只能揭示它的目标数据集的分布特性，查询工具必须依赖于能够生成可比较查询的数据集。创建和控制数据集的分布特性是生成可比较的查询集的前提条件。

人工合成数据生成器具有先天的缺陷。如果数据太过人为假设，比如假设数据完全服从均匀分布，显然它不能描述真实数据集的某些特性；另一个极端，完全使用采集自真实环境的真实数据，那么存在的问题是要么数据规模过小，要么对于研究人员来讲，没有太多的意义，因为它既不能产生可比较的工作负载结果，也不能扩展到回答某些感兴趣的假设问题。系统的 benchmark 在两者之间进行了折中。对于某些数据而言，采取常规的做法，比如每个数据中心发送出来的数据条目，可以认为服从均匀分布，但是每个数据条目中含有的汉语短文本的长度，则认为它服从某种经验分布。对于某些重要的分布，数据生成器依赖真实业务环境的数据产生更接近真实的数据。

5.1.3.1 查询模板

系统的 benchmark 的一个主要任务就是设计一个查询语句集合。该语句集合既能够反映真实世界的需求，同时能够在合理的时间内完成。

每一个查询模板包含以下几个部分：对业务问题的描述，描述了查询应用在怎样的业务情景下；替换标签的定义，定义了查询模板中替换变量的取值方式和取值范围；查询模板体定义：使用 SQL-92 语言，定义查询语句的功能。测试程序的实现者需要实例化查询模板以生成查询语句集。查询模板的内容定义及查询语句的生成使用查询语句生成器，测试选择的查询语句模板如下：

1) 用户收发数据查询

```
SELECT timeA, contentA
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND ( client_fromA = userid or client_toA = userid )
UNION
SELECT timeB, contentB
```

```
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND ( client_fromB = userid or client_toB = userid )
```

2) 双用户收发数据查询

```
SELECT timeA, contentA
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND ( client_fromA = userid1 or client_toA = userid2 )
AND (client_fromA = userid2 or client_toA = userid1)
UNION
SELECT timeB, contentB
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND ( client_fromB = userid or client_toB = userid )
AND contains(contentB, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')>0
```

3) 用户收发内容过滤查询

```
SELECT timeA, contentA
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND ( client_fromA = userid or client_toA = userid )
AND contains(contentA, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')>0
UNION
SELECT timeB, contentB
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND ( datacenter_fromB = datacenter_id or datacenter _toB =
    datacenter_id )
AND contains(contentB, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')
```

4) 数据中心收发内容过滤查询

```
SELECT timeA, contentA
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND ( datacenter_fromA = datacenter_id or datacenter _toA =
    datacenter_id )
AND contains(contentA, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')
UNION
SELECT timeB, contentB
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND ( datacenter_fromB = datacenter_id or datacenter _toB =
    datacenter_id )
AND contains(contentB, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')
```

5) 数据内容精确匹配查询

```
SELECT timeA, client_fromA, client_toA
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND ( datacenter_fromA = datacenter_id or datacenter _toA =
    datacenter_id )
AND contains(contentA, 'datacontentA')
AND contentA = 'datacontentA'
UNION
SELECT timeB, client_fromB, client_toB
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND ( datacenter_fromB = datacenter_id or datacenter _toB =
    datacenter_id )
AND contains(contentB, 'datacontentB')
AND contentB = 'datacontentB'
```

6) 数据内容全文检索查询

```
SELECT timeA, contentA
FROM   DataA
```

```
WHERE DataA.timeA between timeA_value
AND timeA_value +30
AND contains(contentA, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')>0
UNION
SELECT timeB, contentB
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
AND contains(contentB, 'keyWord1 and keyWord2 and keyWord3 and
    keyWord4')>0
```

7) 数据中心统计查询

```
SELECT datacenter_to, count(*)
FROM   DataA
WHERE DataA.timeA between timeA_value
AND timeA_value +30
GROUP BY datacenter_toA
UNION
SELECT datacenter_to, count(*)
FROM   DataB
WHERE DataB.timeB between timeB_value
AND timeB_value +30
GROUP BY datacenter_toB
```

5.2 功能测试

UNION 查询服务为大规模事务系统提供了面向海量信息多数据库的 UNION 查询的功能，本节将利用大规模事务处理系统的 benchmark 对其进行功能测试。

5.2.1 测试方案

对 UNION 查询服务的功能测试分为以下四步：

1) 对测试系统配置查询模板，如上述模板中七类查询；
2) 配置查询的分布和取结果的记录数；
3) 设定查询参数范围；
4) 使用进行测试，记录各类查询查询语句运行情况；
5) 编写客户端程序，调用 UNION 查询服务接口和各个方法，测试正确性。

5.2.2 测试结果与分析

UNION 查询服务对七类模板语句能正常执行，得到正确的查询结果；调用 UNION 查询服务的接口，程序均能正确处理。

5.3 性能测试

UNION 查询服务通过查询优化提高查询的并发执行度，并通过查询后处理策略提高合并查询结果的效率，本节将对 UNION 查询服务的性能进行测试。

5.3.1 测试环境

UNION 查询系统测试环境如表 5.1 所示，测试用表如表 5.4，5.5 所示，TestA 与 TestB 中数据集大小约为 3000 万条。

表 5.4 TestA 表定义

列名	数据类型	约束
TimeA	Date	Not null
Id	Integer	Not null
Name	Char	Not null

表 5.5 TestB 表定义

列名	数据类型	约束
TimeA	Date	Not null
Numid	Integer	Not null
Username	Char	Not null

针对两个测试数据集，两采用了如下三种查询语句，分别代表普通查询、带约束条件的查询以及使用聚合函数的查询：

- 查询语句 1：select timeA , id, name from testA union select timeB, numid, username form testB;
- 查询语句 2：select timeA ,id from testA where id<3 union select timeB, numid form testB where numid>15 order by timeA;
- 查询语句 3：select timeA, avg(id), name from testA group by name union select timeB, avg(numid), username from testB group by username;

5.3.2 测试方案

由于 UNION 查询涉及到大量的网络开销和频繁的 CPU，I/O 操作，所以性能

测试时构造数据集较小的表 TestA 和 TestB，测试步骤如下所示：

1) 设定表 TestA 和表 TestB 的分布域（分别为 1、2 和 3 个节点）；
2) 向表内加载数据；
3) 启动客户端程序进行 UNION 查询（多次取平均值）；
4) 记录查询时间。

5.3.3 测试结果与分析

测试结果如图 5.2 所示：

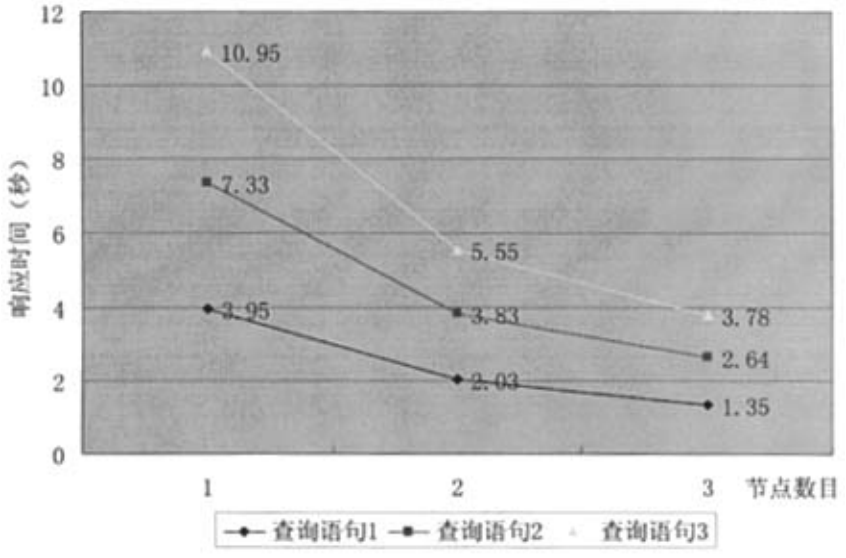


图 5.2 UNION 查询性能测视图

从实验结果看，针对不同的 UNION 查询语句，其查询耗时均随着节点数目的增加成线性收敛，说明 UNION 查询取得了良好的效果。不过随着节点数的增加，控制开销也随之增长，所以加速比和节点数还是有一定的差距。如何减少这种差距正是我们日后要做的工作。

第六章 结 束 语

进入新世纪以来，随着现代科学技术与信息产业的迅猛发展，网络环境下的应用领域不断扩展，越来越多的应用呈现出数据量大，持续倾灌等特点，多数据库成为海量数据存储管理的主要技术手段之一。作为海量信息多数据库的最主要业务之一，查询服务扮演着十分重要的角色，而其中的 UNION 查询更是涉及到大量数据的处理与传输，从而对 UNION 查询的执行效率以及查询性能提出了更高的要求。

本文在充分比较和分析各种解决方案优劣和适用场合的基础上，针对海量信息多数据库中 UNION 查询的处理与优化，利用 CORBA 中间件技术，结合增量多路连接算法等多种优化手段，构建了一套高性能的面向海量的信息多数据库 UNION 查询服务。

首先，本文在对目前海量多数据库中的 UNION 查询时所面临的问题进行分析的基础上，设计了一套面向海量多数据库的 UNION 查询服务体系结构。然后提出一组新的并行 UNION 查询算法，在中间件层支持并行海量数据库 UNION 查询，并在 ORACLE 10G 集群上基于 CORBA 技术实现了该算法。新算法通过对查询进行全局优化与子查询优化来提高查询执行的并行性，并选取增量多路连接、双缓冲并行处理、多路并行归并等策略和算法来提高查询后处理的相应时间。文章的最后，对 UNION 查询服务进行了测试，证明系统在海量多数据库环境下的优越性。

但本文所提出的 UNION 查询系统还有如下方面有待改进：

1) 系统支持的 UNION 查询语句集还有待扩充，如嵌套子查询等，使系统可以适应于更广泛应用的业务需求；
2) 查询优化还有潜力可以挖掘，如子查询语句的代数优化等。

今后， UNION 查询系统在完善功能的基础上还要提高 UNION 查询性能，通过进一步的优化调整来更好的适应海量信息数据库的需求。

致 谢

在我的研究生阶段的学习即将结束之际，谨向在攻读硕士学位的过程中曾经指导过我的老师，关心过我的朋友，关怀过我的领导，和所有帮助过我的人们致以崇高的敬意和深深的感谢！

首先要衷心感谢我的导师贾焰教授！在硕士期间，每一次您对我的点拨都让我受益匪浅，在我消沉的时候，是您的谆谆教诲让我重新振作。在我对问题迷惑的时候，又是您一针见血地指出问题所在。您高深的学术造诣，对问题敏锐的直觉，严谨的生活态度以及对事业的孜孜不倦追求是我以后学习的楷模。在我遇到困难的时候，是您无私的帮助，使我有勇气度过一次次的难关。

衷心感谢杨树强老师。在做课题的日子里，杨老师带领我们小组在一起研究问题，攻克难关，一年之内我们的工程水平和理论功底都有了明显的提高。杨老师对问题的宏观把握能力，敏锐思维，迅速定位和解决问题的能力都让我受益良多。杨老师的编程功力和对事业的热爱是我学习的榜样。

衷心感谢吴泉源教授、王怀民教授、邹鹏教授，你们高深的学术造诣、纵观全局的洞察能力、刻苦敬业的治学态度对我产生有极大影响。

感谢韩伟红老师，从进教研室学习起，韩老师对问题的把握能力和解决能力给我留下了深刻的印象；韩老师接人待物的态度和治学态度是我永远的榜样。

感谢计算机学院 613 教研室的史殿习、李爱平、郭长国、刘惠、隋品波、高洪奎等老师给予的支持和帮助！

感谢梁妍大姐热心的帮助，为我的课题研究提供了很多方便。

非常感谢王永恒、邓波、袁志坚、杜凯、王乐、苏亮、白小波等师兄，你们的指导和帮助使我受益匪浅。

非常感谢贾艳燕、张丽、王博、万智俊、王晓鹏、韩毅、张英武、饶翔等同学，和你们共同学习的这段时间是我人生中一段快乐而充实的乐章，和你们的讨论和交流是我学习和提高的重要途径。

真心感谢我深爱的 Ryan 对我生活无微不至的关心照顾，在我寂寞与疲劳时陪伴着我。

最后，深深的感谢我的父母、亲人和其它所有关心我的人。你们的关心和支持是我最大的学习动力，在我成长的每一个足印里，都倾注了他们的心血和汗水，再次感谢我远方的亲人们。

参考文献

[1] 贾焰，王志英，分布式数据库技术，第 1 版，国防工业出版社，2000

[2] 韩伟红，贾焰，王志英，杨晓东，多数据库系统中的关键技术，计算机工程与科学，21(6)，1999，p1 - 4

[3] 贾焰，分布式对象事务处理中间件技术研究，国防科技大学研究生，2000

[4] 国防科技大学计算机学院网络所，分布计算软件平台 StarBus3.0 程序员指南手册，2002

[5] 朱其亮，郑斌，《CORBA 原理及应用》，北京邮电大学出版社，2001，1 月

[6] 齐勇，马莉等，分布式事务处理技术及其模型，计算机工程与应用，2001，37(9)

[7] 钟午，胡守仁，一种分布式数据库查询优化算法，计算机学报，1997 年第 20 卷第 11 期

[8] 卢正鼎，李兵，肖卫军，李瑞轩，基于 CORBA/XML 的多数据库系统研究与实现，计算机研究与发展，Vol.39，No.4，Apr，2002

[9] 李霖，周兴铭，分布式数据库研究的新方向，计算机应用与软件，2000，17(6)

[10] 施伯乐，丁宝康，汪卫，数据库系统教程(第 2 版)，高等教育出版社，2003 年 8 月

[11] S.M.Deem，Distributed Databases：some problems，1980，International conference on Database，PP.277 - 281

[12] M.Jarke，J.Korch，Query Optimization in Database System，No.2，1984 ACM Computing Surveys

[13] P.H.G.Apers，A.R.Henver，S.B.Yao，Optimization Algorithms for Distributed Queries，IEEE Trans，On Software Engineering，Vol.SE-9，No.1，1983，PP.57-68

[14] C. T. Yu，C. C. Chang，Y. C. Chang，Two Surprising Results in Processing Simple Queries in Distributed Databases，IEEE COMPSAC，1982，PP.377 - 384

[15] Bernstein PA，Goodman J，Query processing in a system for distributed database(SUD-1)，1981，ACM TODS

[16] Babb.E，Implementing a relational database by means of specialized hardware，ACM Trans. Database Syst，4，1(Mar. 1979)，1-29

[17] Berntstein P.A，Chiu D.W，Using Semijoins to Solve Relational Queries，Journal of the ACM，Volume 28，Number 1，January 1981

[18] Bitton D.，Boral H.，DeWitt，D.J.，and Wilkinson，W.K. Parallel algorithms for the execution of relational database operations，ACM Trans，Database Syst. 8，

3(Sept. 1983), 324-353

[19] Cyrus Shahabi, Latifur Khan, Dennis McLeod, A probe-based technique to optimize join queries in distributed Internet databases, Knowledge and Information Systems, 2000

[20] E.LChong, Query Optimization in Distributed Database Systems and Multidata base Systems, Ph.D Thesis, Northwestern University, 1994

[21] W.Du, R.Krishnamurthy, M-C.Shan, Query Optimization in Heterogeneous DBMS, Proc.of the 18th Int'l Conf, Very Large Databases, August 1992

[22] C.Evrendilek, A.Dogac, Query Decomposition, Optimization and Processing in Multidatabase Systems, Technical Report MD-2, Software R&D Center, METU, Dec, 1994

[23] Y.Breitbart, H.Garcia-Molina, A.Silberschatz, Overview of Multidatabase Transacoon Management, VLDB Journal, l(2):181-240, Oct.1992

[24] P.A.Bernstein, V.Hadzilacos, N.Goodman, Concurrency Control and Recovery in Database Systems, Assison-Wesley Reading, MA, 1987

[25] R.Koo, S.Tueg, Check pointing and Rollback-Recovery for Distributed Systems, IEEE Transactions on Software Engineering, Vol.SE-13, No.1, January 1987, pp23-31

[26] Adali S. et al, Query caching and optimization in distributed mediator systems, In:Proc.of SIGMOD, 1996, 137--148

[27] Papakonstantinou I G, Query processing heterogeneous information sources: [Doctor thesis], 2000

[28] Thuraisingham and H.-P.Ko, "Concurrency control in trusted database management systems: a survey", SIGMOD Record 22:4, 1993

[29] Henry F. Korth and Greg 5peegle, "Formal Aspects of Concurrency Control in Long-Duration Transaction Systems Using the NTIPV Model", ACM TODS 19, No.3, 1994.9

[30] C.H.Papadimitriou, The Theory of Database Concurrency Control, Computer Science Press, Rockville, 1986

[31] Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Middle-tier database caching for e-business [C], SIGMOD Conference 2002: 600-611

[32] Oracle, Oracle High Availability Architecture and Best Practices 10g Release 1 (10.1) Part Number B10726-01

攻读硕士学位期间发表的论文

[1] 王佳，贾焰，杨树强 面向海量数据的并行 UNION 查询技术研究与实现，全国开放式分布与并行计算学术会议，西安，2006

攻读硕士学位期间参与的科研项目

[1] 863 国家高技术研究发展计划：网络环境的新一代中间件核心技术及运行平台（No.2004AA112020）

[2] 国家“九七三”重点基础研究发展规划基金项目虚拟计算环境聚合与协同机理研究（No.2005CB321804）

面向海量数据的多数据库UNION查询研究与实现

作者：王佳
学位授予单位：国防科学技术大学