第35卷 第14期 计 算 机 工 程 2009年7月
Vol.35 No.14 Computer Engineering July 2009
·软件技术与数据库· 文章编号：1000—3428(2009)14—0087—03 文献标识码：A 中图分类号：TP311.52

基于XQuery的异构数据源查询处理

严小泉，刘 渊
(江南大学信息工程学院，无锡 214122)

摘 要：异构数据源的集成问题是当前数据处理领域内研究的热点，它能更有效地利用信息资源，更好地实现数据共享。介绍一种基于Mediator-Wrapper中间层的异构数据源集成系统框架，对XQuery查询处理过程及其关键问题，如查询分解和优化技术进行深入研究，并结合实例进一步说明异构数据源中查询分解和优化的具体实现。

关键词：异构数据源；查询分解；查询优化

Query Processing in Heterogeneous Data Sources Based on XQuery

YAN Xiao-quan, LIU Yuan
(School of Information Technology, Jiangnan University, Wuxi 214122)

【Abstract】The integration of heterogeneous data sources, which can effectively make use of the information sources and realize data sharing better, is a hotspot within the field of current data processing research. This paper describes an architecture of heterogeneous data integration system based on a Mediator-Wrapper middleware, and makes a deep research on the XQuery query processing, especially the key issues, such as query decomposition and optimization technology. This paper illustrates the realization of query decomposition and optimization with examples.

【Key words】heterogeneous data sources; query decomposition; query optimization

1 概述

随着Internet和Web技术的飞速发展，许多企业和单位实现了信息的网络管理，极大地提高了工作效率，同时累积了大量的满足各自业务需求的数据源。由于平台差异、数据库技术以及通信协议等方面的不同，各数据源间的互操作非常复杂、困难，形成了“信息孤岛”现象。信息化应用领域的不断深入和扩展使企业迫切需要整合这些分布的、自治的、异构的信息系统，连通“信息孤岛”，实现数据集成、共享和有效一致的信息查询。异构数据源集成的目的是提供给用户访问多种数据源的统一接口，屏蔽它们的平台、系统环境、内部数据结构等方面的异构性，使用户不必考虑数据源的位置、数据抽取和合成等问题，只须提交针对全局视图的查询给集成系统，即可从中获取所需要的数据。

XML的自描述性、跨平台性、开放性、可扩展性等特点，使其能对各种数据源的信息进行标记并在任意系统间传递信息，消除了系统间的“异构性”，为异构数据源集成问题提供了很好的解决途径。XQuery作为W3C组织推荐的XML标准查询语言，不仅可查询XML文档，还可作为XML-Enabled和Native-XML数据库集成系统的查询语言。XML Schema是W3C正式推荐的标准，用XML进行描述，与XML文档的格式完全一致。

2 异构数据源集成系统

2.1 相关工作

本文讨论以XML作为公共数据模型，XML Schema作为数据源模式的异构数据源集成系统框架，并研究集成系统中XQuery的查询分解和查询优化问题。系统使用Mediator-Wrapper[1]的方式集成异构数据源。其中，Mediator是查询处理的核心，主要包括一个虚拟的中介模式以及对中介模式的查询处理机制；Wrapper提供统一的数据源输出模式以及Mediator访问数据源的接口，屏蔽了各数据源之间的差异，实现异构数据源的无缝连接。

Mediator将用户的全局查询分解为针对单个数据源的子查询，Wrapper将子查询转化为本地数据源支持的查询，然后将获取的数据返回给Mediator，Mediator对Wrapper返回的数据进行重组并将结果返回给用户。目前数据集成领域基于模式间映射关系构建的方法主要有2种[2]：GAV(Global-As-View)方法和LAV(Local-As-View)方法。GAV方法将各本地数据源的局部视图映射到全局视图，该方法的优点是查询效率比较高，缺点是可扩展性较差，不适合数据源存在动态变化的情况。LAV方法是将全局视图映射到各数据源上的本地局部视图，该方法的优点是可扩展性好，适合信息源变化比较大的情况，缺点是信息查询效率较低。

2.2 集成系统框架

本文描述的系统在全局模式到局部模式的映射关系上采用GAV方式，它为每个数据源建立一个局部数据源输出模式，全局模式根据各局部数据源输出模式动态集成，以此来简化查询分解[3]和转换过程。

异构数据源集成系统实现基本框架如图1所示，其中每个数据源对应一个包装器，中介器通过包装器和各个数据源交互。用户提出针对全局模式的查询请求，中介器将全局查

基金项目：2007年度高等学校科技创新工程重大项目培育基金资助项目

作者简介：严小泉(1984－)，男，硕士研究生，主研方向：网络安全，网络信息系统；刘 渊，教授

收稿日期：2008-11-04 **E-mail：**suki_yan2004@yahoo.com.cn

询分解为单个数据源能够处理的子查询，并对此过程进行优化，将各个子查询发送给包装器，由包装器来与其封装的数据源交互，执行子查询请求，并将结果返回给中介器，以提高查询处理的并发性，减少响应时间。

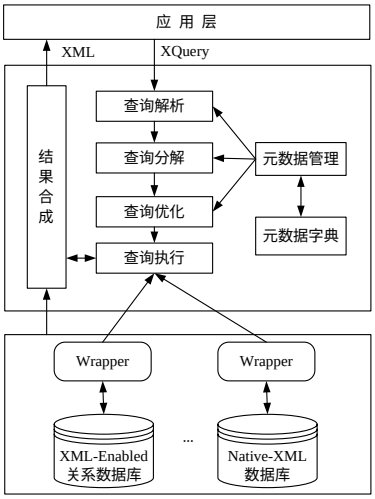

图 1　异构数据源集成系统基本框架

该体系结构框架分为 3 层，每个下层功能的实现都是其上层功能实现的前提，它们相互依赖。主要的模块功能如下：

(1)应用层。由具体业务应用程序组成，是异构数据源集成系统的具体使用者。用户向处理逻辑层发出 XQuery 查询，同时将查询结果返回给用户。

(2)查询解析。查询解析模块接受用户提出的 XQuery 查询请求，并对用户提出的请求进行语法解析，利用元数据管理模块检查查询请求的合法性，并将查询请求转换为规范的查询表示形式，并将合法的 XQuery 传递给查询分解模块。

(3)查询分解。查询分解模块利用元数据管理模块将全局查询分解为针对单个数据源的子查询，同时记录查询分解后的各个子查询之间的关系(如 Union 和 Join)和子查询的执行顺序。

(4)查询执行。该模块负责将子查询分派给各数据源包装器，包装器解析 XQuery 子查询，将其转换为本地查询语言执行；并利用各子查询之间的关系，将子查询结果重组为目标 XML 文档，并将其传送给应用层。

(5)元数据管理和元数据字典[4]。元数据管理模块负责系统中所有数据源的管理和操作。元数据字典记录各数据库的位置、模式等信息，并保存全局视图、局部数据源 Schema 和全局 Schema 以及它们之间的映射关系，提供给元数据管理模块。

3　查询处理关键技术

3.1　查询分解

查询分解是集成系统中查询处理的关键技术之一，它将针对异构数据源的全局查询分解为多个单数据源子查询。查询分解算法应该在保证转换前后的查询结果相一致的条件下，使发送到特定数据源的查询尽可能准确，从而减小从数据源返回的数据集的大小，提高查询执行效率。FLWR 表达式是 XQuery 查询的核心；XPath 用于文档定位；元素构造表达式将查询结果重构为用户所需的 XML 文档结构。XQuery 的这些特性使其非常适合承担异构数据集成系统中查询分解和结果重构的任务。

本系统的查询分解算法建立在 XQuery 语言的层次上，查询分解需要从全局查询和映射文件中提取以下 3 个方面的信息：数据源查询的路径表达式，相对于全局路径的若干相对路径，谓词提取(查询的筛选条件)。查询分解算法将这些信息收集起来，并将分解后的 XQuery 子查询传递给 Wrapper。

查询分解的算法如下：

输入 XQuery 和 Mapping 文件
输出 子查询 $q1, q2, \cdots, qn$
Step1 解析 XQuery xq;
Step2 获取 XQuery 中的全局路径表达式 g1, g2, …, gn;
Step3 读取 Mapping 文件;
Step4 确定全局查询涉及的局部数据源 s1, s2, …, sn;
Step5 确定全局查询涉及的绑定变量 b1, b2, …, bn;
ForBoundVariable=FORclause.getBoundVariable();
LetBoundVariable=LETclause.getBoundVariable();
Step6 for (每个数据源 Si){
//获取 XQuery 中 for 和 let 的绑定变量对应的局部路径表达式
for(绑定变量 bj) { 获取其对应的全局路径表达式 gj,
并通过映射条件找出 gj 对应的局部路径表达式 locj;
if (局部路径表达式 locj 非空){
将绑定变量名修改为路径表达式的最后一个元素名称，
并按 for 和 let 子句的格式构造局部子语句；}
else{没有对应 Si 的路径表达式; }
//对每个数据源 Si 分别用 loc1 替换 g1,loc2 替换 g2, ...,locn 替换 gn
createWhereClause(){//operator 为 where 子句中的操作符
if(operator 的右边是常量){
获得 operator 左边的绑定变量,找到数据源 Si 对应变量并将它添加到 where 子句中; }
else{分别获取 operator 两边的绑定变量 b1 和 b2;
找到 b1 和 b2 对应的局部数据源 local1 和 local2;
if(local1 和 local2 相同){将此条件添加到 where 子句中作为连接条件;}
else{将 local1 和 local 2 上的子查询间的连接条件加入到它们的连接条件列表;}
} }//createWhereClause 结束
createReturnClause(){根据绑定变量信息分解 return 语句，产生 Si 的 return 语句;}
createLocalXQuery(){构造完整的 FLWR 表达式，产生数据源 Si 对应的子查询 qi;}
}//for(绑定变量 bj)结束
}// for (每个数据源 Si)结束

3.2　查询优化

Native-XML 数据库和许多 XML-Enabled 数据库都支持 XQuery 查询，使 XQuery 逐渐成为获取异构数据库资源的标准语言，其查询效率倍受关注。查询优化是异构数据集成中须解决的关键问题之一，但局部数据源的分布性、自治性以及异构性使查询优化非常困难。

XQuery 中 FLWR 表达式本身的复杂性提供了优化的余地[5]。在 XQuery 语义实现过程中，充分考虑 FLWR 表达式语义的特点来减少不必要的变量绑定，重复寻址等计算过程，来达到优化的目的。FLWR 的 for 子句用来创建一个表示节点集多次求值的变量，可以循环且嵌套使用，而 let 子句绑定一个节点集到中间变量中去，如果节点集则无须多次求值，用 let 代替 for 即可。for 和 let 子句的迭代次数是影响整个 XQuery 查询效率的重要因素。本文的查询优化方法主要考虑 FLWR 子句本身的优化来提高 XQuery 的执行效率。

3.2.1　for 子句优化

最外层 for 子句中的绑定变量出现情况有以下 4 种：

(1)where 和 return 子句都包含了 for 子句中的变量。

(2)只有 where 子句包含 for 子句中的变量。

(3)只有 return 子句包含 for 子句中的变量。

(4)where 和 return 子句中都不包含 for 子句中的变量。

其规则如下：

规则 1：在情况(1)和情况(2)下，where 子句是绑定变量的约束条件，return 子句返回的结果是受 for 其影响的。

规则 2：在情况(3)和情况(4)下，for 子句只是引起 return 子句中返回结果的迭代。

规则 1 和规则 2 下的优化如图 2 所示。

规则 1 下的优化		规则 2 下的优化	
原查询	优化后查询	原查询	优化后查询
for $x in E1 for $y in E2 where C1($x) and C2($y) return E3($y)	for $y' in E1 for $y in E2 where C2($y) return E_3($y) for $x in E1 where C1($x) return $y'	for $x in E1 for $y in E2 where C1($x) and C2($y) return E3($y)	for $y' in E1 for $y in E2 where C2($y) return E_3($y) for $x in E1 where C1($x) return $y'

图 2　规则 1 和规则 2 下的优化

若第 1 个 for 子句和第 2 个 for 子句的迭代次数都是 100，那么优化前表 1 和表 2 中总的迭代次数为 10 000 次。而优化后规则 1 中的 C2 只要执行 100 次，C1 执行的次数是 y'的个数乘以 100 ;规则 2 中优化后的总迭代次数为 y'个数乘以 100，都明显减少了计算量。

假设 2 个数据源 S1(bib.xml 和 reviews.xml) 和 S2(authors.xml 和 prices.xml)，它们的模式、全局模式和映射关系如图 3 所示，其中，实线表示 2 个元素间的映射关系，虚线表示连接条件。

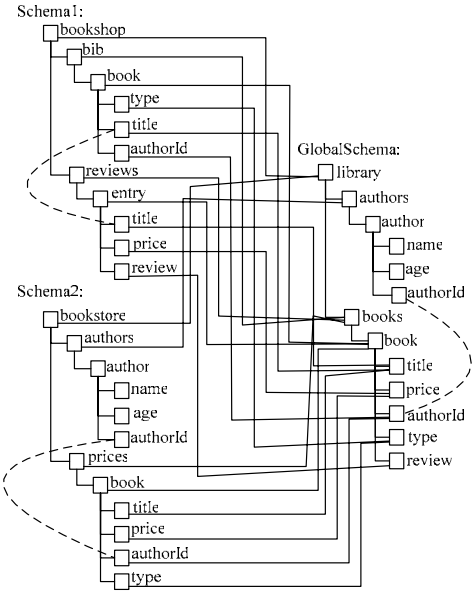

图 3　S1 Schema, S2 Schema, Global Schema 之间的映射关系

例 1 查询书籍类型为计算机类且价格低于 50 元的书名、作者及作者 ID 号。该例应用查询分解算法对全局 XQuery 查询语句进行分解，并且说明了如何对分解后的子查询进行优化。全局查询分解和优化如图 4 所示。

局部数据源 S1 和 S2 的子查询优化分别采用图 2 和图 4 的优化方法。全局查询中的条件 $book/authorId=$author/authorId 是 2 个数据源进行连接运算的连接条件，Mediator 利用它来完成对数据源查询结果的连接。

全局查询	for $book in view("library")/library/books/book for $author in view("library")/library/authors/author where $book/type="Computer Science" and $book/price<50 and $book/authorId=$author/authorId return <title>$book/title</title> <name>$author/name</name> <authorId>$book/authorId</authorId>	
	局部数据源 S1 子查询	局部数据源 S2 子查询
子查询	for $entry in doc("reviews.xml")/reviews/entry for $book in doc("bib.xml")/bib/book where $entry/price<50 and $book/type="Computer Science" and $book/title=$entry/title return <title>$book/title</title> <authorId>$book/authorId</authorId>	for $author in doc("authors.xml")/authors/author for $book in doc("prices.xml")/prices/book where $book/price<50 and $book/type="Computer Science" and $book/authorId=$author/authorId return <name>$author/name</name> <authorId>$book/authorId</authorId>
优化后的子查询	for $entry in doc("reviews.xml")/reviews/entry for $book' in for $book in doc("bib.xml")/bib/book where $book/type="Computer Science" and $book/title=$entry/title return <title>$book/title</title> <authorId>$book/authorId</authorId> where $entry/price<50 return $book'	for $author in doc("authors.xml")/authors/author let $book' := for $book in doc("prices.xml")/prices/book where $book/price<50 and $book/type="Computer Science" and $book/authorId=$author/authorId return <name>$author/name</name> return {$book'} <authorId>$book/authorId</authorId>

图 4　查询分解优化

3.2.2　let 子句和 where 子句优化

一个 let 子句可以包含一个或多个变量，每个变量有一个关联表达式。与 for 子句不同的是，let 子句将每个变量与其关联表达式的结果绑定，并不迭代。

例 2 查询所有价格低于 50 的计算机类书籍的书名及相关评论(假设只涉及单数据库查询)，如图 5 所示。

原查询	优化后查询
for $entry in doc ("reviews.xml")/reviews/entry let $title:=doc ("bib.xml")/bib/book/title where $entry/title=$title and $entry/price<50 return <book> <title>$title</title> <review>$entry/review</review> </book>	let $title:=doc ("bib.xml")/bib/book/title for $entry in doc ("reviews.xml")/reviews/entry [title=$title and price<50] return <book> <title>$title</title> <review>$entry/review</review> </book>

图 5　let 子句和 where 子句优化

let 子句中的绑定变量对于 for 循环是独立的。将 let 子句从 for 循环中提取出来，并将 let 子句生成的变量绑定添加到由 for 子句的路径表达式中，从而省去 for 循环的每次遍历时 let 子句重新赋值的操作，达到优化的目的。

将 where 子句中的简单条件写在相应的 Xpath 路径表达式中，通过 Xpath Engine 过滤结果，省去了变量路径迭代的中间过程，使 Xquery Engine 减少了 for 循环迭代的次数，达到二次优化的目的，这样可以明显减少计算量。对于多个条件的复合查询，合理地安排谓词表达式的顺序可以减少中间结果集的大小。由于等式结果的唯一性要强于不等式，因此将等式放在谓词表达式的最左边，使其最先执行，减少查询匹配的时间、提高执行效率。

3.2.3　查询执行优化

查询执行模块将根据查询的内容选择处理策略，如根据不同数据源之间的关系决定查询的并行与串行顺序，无关的数据源查询采用并行方式，同时制定局部查询结果合并计划。主要的处理策略为：(1)合并查询过程。在查询过程中，将相同数据源的查询语句合并，一次查询得到结果。(2)并行查询。

(下转第 107 页)