VXMLR 系统存储模式的自适应调整

周傲英[1)] 胥正川[2)] 郭志懋[1)] 周水庚[1)]

[1)](复旦大学计算机科学与工程系 上海 200433)

[2)](复旦大学管理学院信息管理与信息系统系 上海 200433)

摘 要 XML 管理系统的查询处理效率很大程度上取决于系统中 XML 数据的存储模式.在用户查询已知或可预测的情况下,根据用户查询设计存储模式可以改善系统的查询处理效率.该文介绍 VXMLR 系统存储模式的自适应调整机制.根据历史查询信息,VXMLR 系统对其存储模式进行自适应调整,从而提高查询处理效率.其基本思路是:首先根据历史查询,推导出适当的映射规则,得到 XML 文档在关系数据库中的存储模式;然后,在给定的空间约束下,根据历史查询使用背包问题求解算法选择关系表进行垂直分割或冗余存储相关数据,使查询所访问的无关数据尽可能少.VXMLR 系统提供四种存储模式调整策略,其中两种策略可以实现自适应的存储模式调整.实验结果表明文中提出的方法是有效的.

关键词 XML 数据管理;存储模式;自适应模式调整

中图法分类号 TP311

Adaptation of XML Storage Schema in VXMLR

ZHOU Ao-Ying[1)] XU Zheng-Chuan[2)] GUO Zhi-Mao[1)] ZHOU Shui-Geng[1)]

[1)](*Department of Computer Science and Engineering*, *Fudan University*, *Shanghai* 200433)

[2)](*Department of Information Management & Information System*, *School of Management*, *Fudan University*, *Shanghai* 200433)

Abstract The efficiency of an XML management system depends mainly on its storage schema. Designing storage schema based on users' queries can improve its performance significantly. This paper introduces the technique of XML storage schema adaptation in the VXMLR system. Based on history queries, the VXMLR system automatically adjusts its storage schema for speeding up query processing. First, some appropriate mapping rules are derived from the history queries, and then initial storage schema is generated by using these mapping rules. Second, under given space constraint and with the history queries, some relational tables are selected for vertical partitioning or redundant storage so that these tables can be accessed as fast as possible. Four kinds of schema adaptation strategies are presented, in which two are used for automatic schema adaptation. Experimental results validate the practicability and effectiveness of our proposed approach.

Keywords XML data management; storage schema; schema adaptation

1 引 言

XML 已经成为 Web 上数据表示、集成和交换的标准①.用关系数据库存储 XML 数据是一种比较理想、实用的选择,因为这样不但可以利用已经很成熟的关系数据库技术(如查询优化、并发控制等),还可以使 XML 数据与传统的关系数据共存于同一应

收稿日期:2003-02-17;修改稿收到日期:2003-12-09.本课题得到国家自然科学基金(60228006,60003008)和国家"八六三"高技术研究发展规划项目基金(2002AA116020)资助.**周傲英**,男,1965 年生,博士,教授,博士生导师,研究方向为 XML/WEB 数据管理、WEB 服务、数据流与数据挖掘、P2P 计算. E-mail: ayzhou@fudan.edu.cn. **胥正川**,男,1973 年生,博士,讲师,研究方向为 XML 数据管理、基于 WEB 的信息系统. **郭志懋**,男,1978 年生,博士研究生,研究方向为 XML 数据管理、WEB 服务及其数据管理. **周水庚**,男,1966 年生,博士,教授,研究方向为信息检索和文本挖掘、空间数据库和地理信息系统、P2P 计算.

① W3C. Extensible Markup Language (XML) 1.0. October 2000. http://www.w3.org/TR/REC-xml

用之中.这种存储方法已经在数据库界内得到了广泛认同[1~4].

目前用关系数据库存储 XML 数据大致有两类方法:基于模型的方法和基于结构的方法[1].前者用固定的关系模式存储所有 XML 文档的结构,如 Florescu 等[2]提出的简单映射方法,他们根据 XML 文档的 DOM 模型,将 DOM 模型中的节点与边的关系映射为关系模式;后者则从 XML 结构中导出关系模式,如 Deutsch 等[3]提出的用数据挖掘的方法从 XML 文档中抽取出关系模式. Shanmugasundaram 等[4]提出了两种将 XML 文档的 DTD 映射为关系模式的结构映射方法,分别称为 HYBRID 方法和 SHARED 方法,它们的唯一差别在于对 DTD 图中入度大于 1 节点的处理:SHARED 方法将该类节点映射为独立关系表,而 HYBRID 方法则将该类节点内联到它父节点所在的关系表中.相比于基于模型的方法,基于结构的映射方法产生的文档片段较少,在遍历路径时需要的连接操作也少,因此查询效率优于基于模型的方法. Tian 等[5]对各种不同的存储方法作了性能比较,得出结论:当存在 DTD 时,将 DTD 映射为关系模式是最好的存储方法.

但是,SHARED 和 HYBRID 方法存在两个问题.首先,对入度大于 1 的节点的映射规则过于简单化,影响到查询效率;其次,在关系模式确定后,若某些查询所访问的关系表中无关数据过多,则查询效率就会降低.导致这些问题的原因在于上述方法只考虑了 XML 文档的结构信息,而没有考虑到作用于其上的查询信息.如果用户的查询是已知的或者可预测的,则可以根据 XML 文档结构和用户查询综合设计 XML 数据库系统的存储模式,从而改进系统的查询处理性能.根据 Zipf 原则[6],一个数据库系统最常用的第 i 个查询的发生概率是 $1/i$.因此,我们可以由历史查询记录大致推测未来可能发生的查询,并在此基础上设计 XML 文档管理系统的存储模式.

本文介绍 VXMLR 系统中存储模式的自适应调整机制.VXMLR 是我们研制的一个基于关系数据库的可视化 XML 文档管理系统[7~9].该系统采用将 DTD 映射为关系模式的内联技术作为基本存储方法,然后基于历史查询对存储模式进行自适应调整,并允许在调整中有选择地冗余存储相关数据.其具体思路是:

(1) 对于 DTD 图中入度大于 1 的节点,根据历史查询中该类节点与其父节点的关联程度决定采用内联或外联方法存储该节点,由此确定 DTD 到关系模式的映射规则.

(2) 为了减少关系表中的无关数据,先把一组 XML 历史查询转换为只跟单个关系表有关的 SQL 子查询和它们之间的连接操作,然后根据这些子查询对各个关系表进行自适应调整:冗余存储各子查询的相关数据或对相应关系表进行垂直分割,使得各子查询访问的关系表尽可能小.

(3) 计算上述调整对查询效率的影响及其存储代价,在给定空间约束条件下利用求解背包问题最优解的近似算法选择关系表作进一步调整,从而完成对存储模式的自适应调整.

需要指出的是:近一年来,国际上也有一些其他学者研究利用用户查询信息设计 XML 存储模式以改善 XML 查询的处理效率. Bohannon 等[10]提出了一个针对给定应用设计其存储模式的系统框架; Zheng 等[11]也研究了在给定用户查询的情况下如何基于查询代价选择 XML 文档在关系数据库中的存储模式.这两项工作的差别主要是具体实现的算法.与这两项研究相比,本文工作有明显的不同:

(1) 本文介绍的存储模式调整方法与技术已经集成到我们的 XML 文档管理系统 VXMLR 中.

(2) 本文研究的是动态模式调整,VXMLR 系统根据用户查询的变化自适应地调整存储模式,而前面的两项工作都是针对给定应用或用户查询进行模式调整.

(3) 本文给出的方法允许冗余存储数据,因此可以进一步改善查询效率,而前面的两项工作没有考虑冗余存储.

本文第 2 节简要介绍 VXMLR 系统和它的存储方法及其存储模式调整模块的构成;第 3 节分析了 XML 查询的代价模型,在此基础上提出了基于一组历史查询,对存储模式进行调整的两个算法;第 4 节给出进行存储模式调整的四种策略;第 5 节是相关实验结果和分析;第 6 节为结论部分.

2 VXMLR 系统及其存储方法

2.1 VXMLR 系统简介

VXMLR 系统通过关系数据库管理 XML 文档,其技术特点包括:

(1)采用内联技术将 XML 数据映射到关系表,使用关系数据库系统来存储数据;

(2)基于历史查询,对存储模式进行自适应调整;

(3)具有可视化查询界面,提供简易方法查询存储在关系数据库中的 XML 数据. XML 数据的结构被显示在界面上,用户可以通过点击界面上的数据条目并输入查询条件来生成查询,而不需要知道路径表达式的复杂语法;

(4)采用查询重写技术将带路径表达式的 XML 查询转换为 SQL 语句. 使用统计信息和新颖的路径目录来减少 SQL 语句的数目和连接操作的数目,显著提高了系统性能.

在 VXMLR 系统中,XML 文档按照一定的存储映射规则存储在底层关系数据库中,由关系数据库管理系统维护. 首先,输入的 XML 文档被解析成 DOM 树. 同时,系统抽取出它的 DTD. 然后,DOM 树被映射成关系表并被存储在关系数据库中. 用户通过可视化界面 DVQ(DTD-driven Visual Query)提交查询. 这些查询被转换成 SQL 语句并提交给底层的关系数据库管理系统. 为了从基于路径表达式的查询有效生成 SQL 语句,VXMLR 维护统计信息和一个路径目录(path directory). 统计信息和路径目录在查询重写过程中用来减少 SQL 语句的数目并简化连接操作. 查询结果从 DBMS 返回并重构成 XML 文档,然后通过 XSL 转换呈现给用户.

VXMLR 系统中的模式调整模块基于历史查询记录,对底层的关系存储模式进行自适应调整. 它包括如下子模块:

(1)记录查询. 将用户提交的查询记录下来,每条记录包括查询语句、查询发生的时间以及执行该查询的时间代价. 这些查询将作为存储模式调整的依据.

(2)确定映射规则. 该模块完成映射规则的确定,即根据入度大于 1 的节点与其父节点的关联程度确定其映射规则. 若该节点与其父节点的关联程度比较大,则将该节点内联到其父节点所在的关系表中,否则将该节点映射为独立的关系表.

(3)分解查询. 该模块完成对查询的分解,即将查询记录中的 XML 查询分解为一系列只跟单个关系表相关的子查询和它们之间的连接操作.

(4)估算调整收益代价. 该模块估算各种存储模式调整所带来的查询收益与存储代价,并计算其收益与代价比.

(5)调整关系模式. 该模块完成对存储模式的调整. 根据各种模式调整的收益与代价比,在存在空间约束的条件下,运用背包算法挑选出合适的存储模式,完成存储模式的自适应调整.

通过上述的模式调整模块,VXMLR 系统完成对存储模式的调整. 通过应用自适应的存储模式调整策略,使系统能够适应用户查询的变化,保持较高的查询处理效率.

2.2　VXMLR 系统中的存储方法

VXMLR 系统中的基本存储方法是结构映射方法,即将 XML 文档的 DTD 映射为关系数据库模式,再对其数据进行存储. XML 文档的结构是通过其 DTD 说明的,DTD 相比关系模式有两个重要差异. 第一,关系模式是二维结构;第二,关系模式中不允许出现值为集合的属性和相互嵌套的属性,而 XML 文档中这些情形是常见的. 结构映射方法通过将子节点内联到其父节点中,解决第一个差异;通过将值为集合的元素和相互间有嵌套关系的元素映射为独立关系,解决第二个差异. 具体方法是,首先将 XML 文档的 DTD 简化成一个 DTD 图,然后将 DTD 图中的如下三类节点映射为独立的关系表,最后得到存储该 XML 文档的关系模式:

(1)入度为零的节点,即根节点和孤立节点;

(2)有一条入边标号为"*"的节点,这种节点反映了 XML 元素的集合特性;

(3)入度为 1,但相互嵌套的节点,选择其中之一映射为关系表.

对入度大于 1 的节点(即有两个或多个父节点的节点),结构映射中的 SHARED 方法将该类节点映射为独立关系,HYBRID 方法则将该类节点内联到它父节点所在的关系中. 其他的节点则被内联到由其最近祖先节点生成的独立关系中.

例 1.　图 1 给出一个示例 DTD 和它的 DTD 图. 按照上述规则,节点 PurchaseOrder, ItemsBought, Payments 将被映射成独立关系表,而节点 age, price, credit 则分别被内联入关系表 PurchaseOrder, ItemsBought 和 Payments 中. 该 DTD 图中,只有节点 name 和 SerialNum 的入度大于 1,且没有标号为"*"的入边. SHARED 方法将这些节点映射为独立的关系表,节点 name 被映射为关系表 name,节点 SerialNum 被映射为关系表 SerialNum,得到的关系模式如图 2 所示. HYBRID 方法把这些节点内联到它们父节点所在的关系表中,节点 name 被内联到关系表 PurchaseOrder 和 ItemsBought 中,而 SerialNum 则被内联入关系表 ItemsBought 和 Payments 中,得到的关系模式如图 3 所示. 由于查询转换和重构的需要,在每一个生成的关系表中添加连接信息,即两个属性列 ID 和 PARENTID.

```
〈!ELEMENT PurchaseOrder (Buyer, ItemsBought*, Payments*)〉
〈!ELEMENT Buyer (age, name)〉
〈!ELEMENT age (#PCDATA)〉
〈!ELEMENT name (#PCDATA)〉
〈!ELEMENT ItemsBought (name, price, SerialNum)〉
〈!ELEMENT price (#PCDATA)〉
〈!ELEMENT SerialNum (#PCDATA)〉
〈!ELEMENT Payments (SerialNum, credit)〉
〈!ELEMENT credit (#PCDATA)〉
```

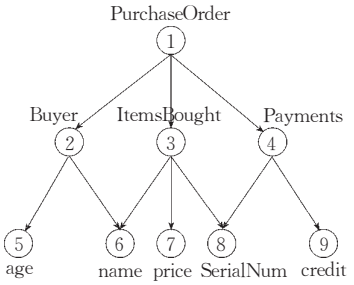

图 1 示例 DTD 和 DTD 图

这里对入度大于 1 的节点的处理过于简单,对于同时访问该类节点与它们父节点的查询,采用内联方法存储该类节点比用外联方法好,因为此时若内联该节点,则该查询只需访问一个关系表,而若采用外联方法存储该类节点,则要对两个关系表作连接操作. 对于只访问该节点及其子节点的查询,采用外联方法更好,因为此时若内联该节点,就必须查询它所有父节点所在的关系表,然后作并操作. 而采用外联方法,只需访问该节点所在的关系表就可以完成该查询.

我们根据 VXMLR 系统的历史查询信息决定入度大于 1 的节点的存储方法,确定在关系数据库中存储 XML 文档的模式,本文给出相应算法作为存储模式自适应调整的预处理. 然后基于历史查询信息,在有存储空间约束的限制条件下,通过垂直分割部分关系表或冗余存储部分数据,进一步调整该存储模式,提高 VXMLR 系统的总体查询效率.

3 存储模式的自适应调整

给定一组历史查询$\{Q_i \mid 1 \leq i \leq n_{wkl}\}$,$Q_i$的权重为$w_i$(即该查询的执行时间),频度为$f_i$(即该查询发生的频率),可用存储空间为 M. 在这些条件下,我们对存储模式进行调整,以提高 VXMLR 系统的查询处理效率. 我们通过两个步骤达到这个目标:(1)确定入度大于 1 的节点的存储映射规则;(2)调整已存在的关系存储模式.

3.1 入度大于 1 的节点的存储方式

如前所述,如果不加区分地内联或外联入度大于 1 的节点,则在某些情况下会降低系统的效率. 例如,查询在 purchase order 中出现过的所有 Serial number. 若 SerialNum 被内联到它父亲所在的关系表中(如图 3),则该查询必须查询 ItemsBought 和 Payments 这两个关系表,并对它们的结果作并操作. 但如果外联节点 SerialNum(如图 2),则该查询仅仅需要访问表 SerialNum,所以此时应该外联 SerialNum. 对另一查找所有 buyer 的 name 的查询,若内联节点 name,则该查询只需访问关系表 PurchaseOrder;若外联该节点,则该查询必须访问两个关系表 PurchaseOrder 和 name,并对它们作连接操作,所以此时应将 name 内联到它父亲所在的关系表中去. 基于这两个查询,我们可以设计一个新的存储模式,如图 4.

```
PurchaseOrder (ID, Buyer_age, PARENTID)
ItemsBought (ID, price, PARENTID)
Payments (ID, credit, PARENTID)
name (ID, name, PARENTID)
SerialNum (ID, SerialNum, PARENTID)
```

图 2 采用 SHARED 方法产生的关系模式

```
PurchaseOrder (ID, Buyer_age, Buyer_name, PARENTID)
ItemsBought (ID, name, price, SerialNum, PARENTID)
Payments (ID, SerialNum, Credit, PARENTID)
```

图 3 采用 HYBRID 方法产生的关系模式

```
PurchaseOrder (ID, Buyer_age, Buyer_name, PARENTID)
ItemsBought (ID, Item_name, price, PARENTID)
Payments (ID, credit, PARENTID)
SerialNum (ID, SerialNum, PARENTID)
```

图 4 新的存储模式

因此对于入度大于 1 的节点,本文根据它在历史查询中与其父节点的关联度,决定采用内联或外联方法存储该节点.

定义 1(关联查询). 对入度大于 1 的节点 n,若在查询 q 中,n 和某一个父节点同时出现,则称 q 为 n 的关联查询,否则称为非关联查询.

定义 2(关联度). 入度大于 1 的节点与其父节点的关联度定义为它所有关联查询的权重与频度乘积的和,而它所有非关联查询的权重与频度乘积的和被称为非关联度.

当一个入度大于 1 的节点与其父节点的关联度大于或等于非关联度时,将该节点内联到它父节点

所在的关系表中,否则采用外联方法存储该节点.设入度大于 1 的节点的集合是$\{N_j \mid 1\leq j\leq n_{\text{nodes}}\}$,一组历史查询如前所述,图 5 给出了相应的算法.通过这个算法可以确定入度大于 1 的节点的存储映射规则,由此完全确定从 DTD 到关系模式的映射规则.

```
算法:GenMap
输入:入度大于 1 的节点集合{Nj | 1≤j≤nnodes}和一组历
      史查询{Qi | 1≤i≤nwkl},Qi 的权重为 wi,频度为 fi
输出:Nj 的存储方式
过程:
for each Nj
  Win = 0; Wout = 0
  for each Qi
    if Qi 是 Nj 的关联查询
      Win += wi · fi
    else Wout += wi · fi
  end for
  if Win > Wout
    用内联方法对节点 Nj 进行存储
  else 用外联方法对节点 Nj 进行存储
end for
```

图 5　产生映射规则的算法

3.2　查询的分解

任何对 XML 文档的查询都可以表示为一条带简单路径表达式或正则路径表达式的查询,而带正则路径表达式的查询可以转换为几条带简单路径表达式的查询.将 XML 文档存储在关系数据库中后,采用查询重写规则可以将带简单路径表达式的查询转换为相应的 SQL 语句.因此,查询集合$\{Q_i \mid 1\leq i\leq n_{wkl}\}$中的每一条查询都可以转换为一系列关系表上的子查询和它们之间的连接操作,且每条子查询只访问一个关系表.为了表述方便,表 1 给出下文将要用到的相关符号表示,其中 R 为关系表.

表 1　符号及其含义

符号	含义
$\lvert R\rvert$	R 所占用的存储空间大小
$\lVert R\rVert$	R 的元组数
$\{Attr_k^R\}$	R 的属性,$1\leq k\leq col(R)$,$col(R)$为 R 的属性个数
$size(Attr_k^R)$	单个 $Attr_k^R$ 值所需空间
$\lvert R$ 的连接信息$\rvert$	R 中连接信息所占空间,即$(size(ID)+size(PARENTID))\cdot\lVert R\rVert$

设查询 Q_i 访问关系表 R_{i1}, …, $R_{in(i)}$.将 Q_i 转换为子查询 Q'_{i1}, …, $Q'_{in(i)}$,及它们之间的连接操作,其中 Q'_{ij} 只访问关系表 R_{ij}.本文用相关表属性的集合来表示每条子查询,即子查询 $Q'_{ij}=\{Attr_k^{R_{ij}}\}$,其中 $Attr_k^{R_{ij}}$ 是被 Q'_{ij} 访问的 R_{ij} 中的属性列.所以 R_{ij} 中与该子查询有关的数据量(记为$\lvert Q'_{ij}\rvert$)可以通过如下计算得到

$$\lvert Q'_{ij}\rvert=\sum_k size(Attr_k^{R_{ij}})\cdot \lVert R_{ij}\rVert+\lvert R_{ij}\text{ 的连接信息}\rvert \quad (1)$$

设 Q'_{ij} 的权重为 w'_{ij},频度为 f'_{ij},则有

$$f'_{ij}=f_i \quad (2)$$

$$w'_{ij}=\left(\frac{\lvert R_{ij}\rvert}{\lvert R_{i1}\rvert+\cdots+\lvert R_{in(i)}\rvert}\right)\cdot w_i \quad (3)$$

对历史查询$\{Q_i \mid 1\leq i\leq n_{wkl}\}$中的每条查询都作这种分解,则得到一个新的查询集合$\{Q'_{ij} \mid 1\leq i\leq n_{wkl},\ 1\leq j\leq n(i)\}$.合并该集合中完全相同的子查询,合并后的子查询权重等于被合并的所有子查询中权重的最大值,频度等于所有子查询的频度之和,可得相异子查询的集合.为了下文叙述的方便,亦将该集合记为$\{Q_i \mid 1\leq i\leq n_{wkl}\}$,注意这里的 Q_i 和上文的 Q_i 不一样了.我们根据这组子查询,进一步调整每个关系表的存储,减小查询代价.

3.3　查询收益与存储代价

本文的工作是调整存储模式以提高查询效率,关心的是不同存储模式下查询代价的相对大小,对查询代价的准确值并不关心.对一个只访问单个关系表的查询,其查询代价是它所访问关系表大小的单调增函数[12].对 XML 文档的查询可以转化为一系列对单个关系表的查询和它们之间的连接操作,所以 XML 查询的代价就等于各子查询的代价和在它们之间作连接操作的代价之和.

为了减小 XML 查询的代价,提高查询效率,就要使每个子查询所访问的无关数据尽可能少.因此我们考虑将关系表中与某一子查询相关的数据存储在一个单独的关系表中.这将大大提高该子查询的效率,本文称这种方法为独立存储该子查询.子查询的独立存储有两种方式,一种是冗余独立存储,新建一个关系表,将原关系表中与该子查询相关的数据复制到新关系表中;另一种是非冗余独立存储,将该子查询的相关数据从原关系表中分割出来,存储在新的关系表中,即对原关系表作垂直分割.

设子查询 Q_i 的权重是 w_i,频度是 f_i,独立存储该子查询的查询收益表示为 $bft(Q_i)$,存储代价表示为 $cost(Q_i)$.Q_i 原来所访问的关系表是 R_i,新建的关系表为 R'_i,$\lvert R'_i\rvert$ 等于 $\lvert Q_i\rvert$.查询收益是该子查询所访问的关系表大小的差(即$\lvert R_i\rvert$与$\lvert R'_i\rvert$的差)乘以该子查询的权重与频度.因此 $bft(Q_i)$如下计算,

$$bft(Q_i)=(\lvert R_i\rvert-\lvert Q_i\rvert)\cdot w_i\cdot f_i \quad (4)$$

存储代价则是独立存储该子查询所占用的额外空

间.对于冗余独立存储,其存储代价就是新建关系表的大小,

$$cost_{\text{red}}(Q_i) = |Q_i| \tag{5}$$

而非冗余独立存储的代价只是连接信息所占用空间的大小,

$$cost_{\text{non}}(Q_i) = |R_i \text{ 的连接信息}| \tag{6}$$

3.4 独立存储方式的选择

如果仅仅从空间开销方面考虑,非冗余独立存储优于冗余独立存储.然而,如果不同的子查询要访问共同的表属性,非冗余独立存储某一子查询可能会影响其他子查询的效率.例如,子查询 $Q_1=\{Attr_1, Attr_2, Attr_3\}$ 和 $Q_2=\{Attr_3, Attr_4, Attr_5\}$,频度分别为 f_1, f_2,权重分别为 w_1, w_2,且 $w_1>w_2$,它们都只访问关系表 R.若非冗余独立存储 Q_1,则 R 分裂成两个关系表,当执行 Q_2 时就要访问这两个表,并作连接操作,代价反而增大.

因此,在对子查询作非冗余独立存储时,需要考虑它与其它子查询有无公共属性.若无公共属性,可以非冗余独立存储;若相互间存在公共属性,将它们合并后,非冗余独立存储合并后的查询.Q_1 和 Q_2 的合并结果是 $Q'=\{Attr_1, Attr_2, Attr_3, Attr_4, Attr_5\}$.$Q'$ 的频度是 $f'=f_1+f_2$,因为 $w_1>w_2$,所以 Q' 的权重 $w'=w_1$.由公式(4),$bft(Q')=(|R|-|Q'|)\cdot w'\cdot f'$,存储代价是非冗余独立存储 Q' 的空间大小,$cost_{\text{non}}(Q')=|R$ 的连接信息$|$.将 Q' 独立存储为关系表 R' 后,Q_1 和 Q_2 只需访问关系表 R' 而不必访问原关系表 R,提高了查询效率.

将集合 $\{Q_i \mid 1\le i\le n_{wkl}\}$ 分解为互相之间无公共属性的子查询集合 G 和互相之间有公共属性的子查询集合 H.在集合 H 上建立等价关系:如果两个子查询访问某一共同的表属性,那么它们两者等价.根据该等价关系可以得到集合 H 上的划分 $\{p_j\}$.针对给定分区 p_j,我们采用贪心算法合并 p_j 中的部分查询,得到 Q_j^m,同时删除已被合并的子查询,而未被合并的子查询集合仍然记为 p_j.这样就得到了 $\Omega=G\cup\{Q_j^m\}\cup\{p_j\}$,具体算法见图 6.

计算 Ω 中所有元素的查询收益与存储代价,可得到它们各自的查询收益和存储代价.剩下来的问题就是在空间限制为 M 的情况下,如何挑选一部分查询进行独立存储.这是一个典型的 0/1 背包问题,我们采用近似算法解决它.对 Ω 中的任一查询 q,如果被挑选为独立存储,若 $q\in G\cup\{Q_j^m\}$,则非冗余独立存储 q;若 $q\in\{p_j\}$,则冗余独立存储 q.

```
算法:MergeQueries
输入:{p_j}
输出:{Q_j^m}, {p_j}
for each p_j
  Q_j^m = GreedyMerge(p_j)
end for

算法:GreedyMerge
输入:p
输出:Q^m
mq = ∅
while p ≠ ∅
   mqt_1 = ∅
   for each qt ∈ p
     mqt_2 = (mq ∪ qt)
     t = bft(mqt_2)/cost_non(mqt_2)
     if (t > bft(mqt_1)/cost_non(mqt_1) and t > bft(qt)/cost_red(qt)
           and cost_non(mqt_2) < cost_red(qt))
       mqt_1 = mqt_2; q = qt
   end for
   if (mqt_1 ≠ ∅)
     mq = mq ∪ mqt_1
     p = p - {q}
   else exit while
end while
return mq
```

图 6 合并子查询的算法

4 存储模式调整策略

前面我们讨论了存储模式自适应调整算法,这一节讲述存储模式调整策略.本文给出四种调整策略,其中,第一种策略由用户决定是否进行模式调整;第二种策略由系统周期性地自动进行模式调整;第三、四种策略根据用户查询的变化情况来决定是否对存储模式进行调整,因此是自适应的调整策略.VXMLR 系统实现了这四种策略,但后两种是 VXMLR 系统采用的主要调整策略.

按需调整:系统管理员或者用户决定何时进行存储模式的调整.当管理员或者用户觉得有必要进行存储模式的调整时,就启动这一功能.

周期性调整:按照这种调整策略,系统周期性地进行存储模式的调整.在 VXMLR 系统中,这种周期性调整分两种情况:基于时间的周期性调整和基于查询的周期性调整.前者是每隔一定的时间系统就自动进行模式调整;后者则是系统每处理完一定量的用户查询就自动进行模式调整.

基于查询效率的自适应调整:每隔一定时间,系统自动检测最近一段时间内查询处理的平均效率与前一次模式调整后紧接着的同等时间段里查询处理平均效率的变化情况.如果目前的平均效率降低了,且降低的程度超出给定的门限值,则启动存储模式调整程序.

基于查询内容的自适应调整:与第三种策略不同的是,这种调整策略根据查询内容的变化来决定是否进行模式的调整.将用户查询看作查询序列,在

查询序列上建立一个考察滑窗,其宽度为 w,表示滑窗内包含的查询个数.考察当前滑窗 W_i 覆盖的查询串 $\{q_{ik+1},q_{ik+2},\cdots,q_{ik+w}\}$ 与前一次调整使用的滑窗 W_0 对应的查询串 $\{q_1,q_2,\cdots,q_w\}$ 的差异程度,记为 $diff(W_0,W_i)$.基于查询内容的存储模式自适应调整策略表述如下:

给定一个查询内容差异度阈值 $Diff_0$,若 $diff(W_0,W_i)>Diff_0$,那么在 W_i 中的最后一个查询 q_{ik+w} 之后,根据 $q_{ik+1},q_{ik+2},\cdots,q_{ik+w}$ 启动新一轮存储模式调整.限于篇幅限制,$diff(W_0,W_i)$ 的具体计算方法不拟在此详述.

5　实验结果

为了检验本文方法的有效性,我们对 VXMLR 系统的存储模式调整机制进行了一系列的测试与验证,并对系统存储模式调整前后的性能进行比较.Shanmugasundaram 等认为对一般的 DTD 而言,HYBRID 和 SHARED 方法的区别不大[4].Tian 等在比较各种存储方法的性能时采用 SHARED 作为结构映射方法的代表[5].因此我们把用 SHARED 方法映射得到的存储模式作为调整前的关系模式,存储空间的约束定为调整后的总空间小于调整前所占空间的 110%.目前,用于测试 XML 数据库性能的一个常用基准测试集为 XMark①.我们采用 XMark 生成 133M 的 XML 文档作为实验数据,查询用例为用于检验 XML 查询性能的基准测试查询集① 中的 17 条查询.测试系统运行在 Windows 2000 环境下,处理器为 PIII 500MHz,128M 内存,底层数据库为 SQL Server 2000.

表 2 给出一组查询在模式调整前后的处理时间.其中,Shared 对应于采用 SHARED 方法存储 XML 文档(即调整前)的查询时间 t_{shared};而 Adapted 对应的是存储模式调整后的查询时间 t_{apt}.图 7 给出每条查询的执行效率改善因子,它的计算公式是 $Eff=\frac{t_{shared}-t_{apt}}{t_{shared}}$.图 8 是一组历史查询的总执行时间.使用效率改善因子计算公式可得,调整存储模式后,总查询效率提高了 41%.

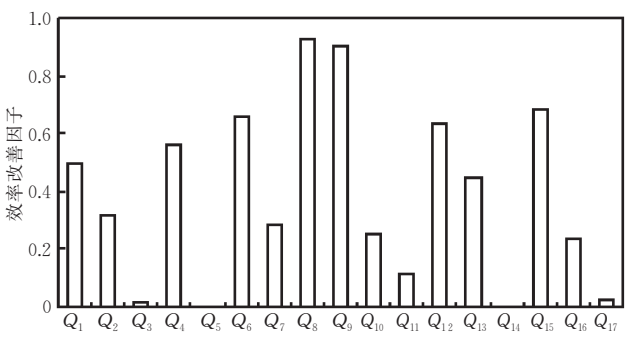

图 7　查询执行效率改善因子

表 2　一组查询的执行时间

Query	Shared (ms)	Adapted (ms)	Query	Shared (ms)	Adapted (ms)
Q_1	1603	811	Q_{10}	861	641
Q_2	480	326	Q_{11}	125240	111370
Q_3	2364	2333	Q_{12}	441	160
Q_4	841	371	Q_{13}	1182	650
Q_5	170	170	Q_{14}	721	721
Q_6	932	321	Q_{15}	1101	351
Q_7	421	301	Q_{16}	892	681
Q_8	22292	1593	Q_{17}	701	682
Q_9	53287	5107			

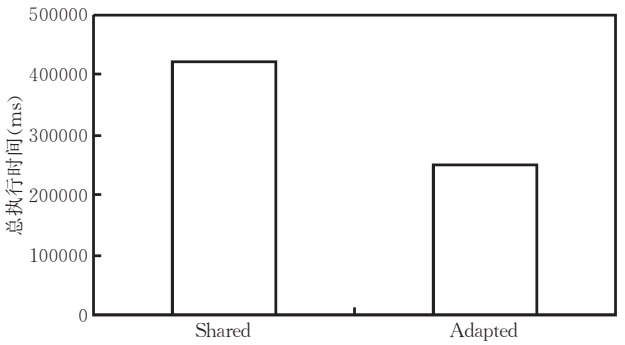

图 8　查询的总执行时间

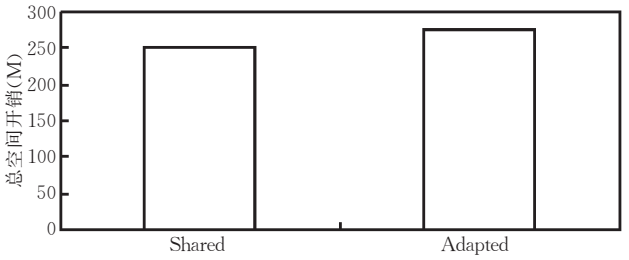

图 9　存储模式调整前后的总存储代价

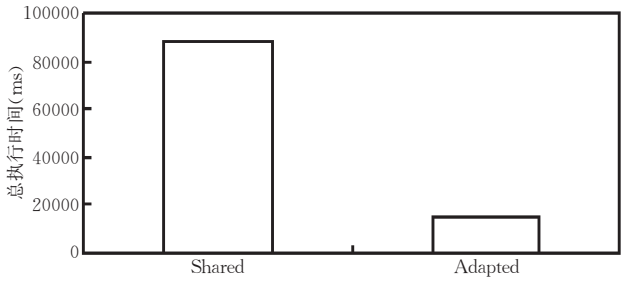

图 10　16 条历史查询的总执行时间(不包含 Q_{11})

① Schmidt A.,Waas F.,Kersten M. *et al*.. The XML Benchmark Project. CWI, Amsterdam, The Netherlands, Technical Report: INS-R0103, 2001. http://monetdb.cwi.nl/xml/

从表 2 和图 7 可以看到存储模式调整后几乎每条查询语句的执行时间都有所减少，但查询 Q_{11} 的执行效率没有太多改善. 这是因为 Q_{11} 的执行时间主要用于连接操作，而本文提出的算法主要针对数据库中单个关系表访问效率的提高，对查询中连接操作性能的改善不大. 所以在存储模式调整后，对此类连接操作比较多的查询影响不大，性能改善的效果不很明显. 图 9 是存储模式调整前后系统存储代价的比较图，可以看到调整存储模式前后所花的存储代价比较接近；图 10 给出除去 Q_{11} 后的总查询执行时间，总查询效率提高了 82.8%. 这表明本文的方法在几乎不改变存储空间的情况下，显著提高了系统的查询效率.

为了验证动态调整存储模式的效果，我们把实验中的 17 条查询语句分成两组：$Q_1 \sim Q_9$ 为第一组，$Q_{10} \sim Q_{17}$ 为第二组. 先根据第一组查询调整存储模式，测试 17 条查询的查询效率，再根据第二组查询调整存储模式，然后测试所有查询的查询性能. 图 11 所示为实验结果，其中，First 表示根据第一组查询调整存储模式后，执行 17 条查询的时间开销；Second 表示根据第二组查询调整存储模式后，执行所有查询的时间开销. 由图 11 可见，第一组查询（$Q_1 \sim Q_9$）在 First 曲线中的查询效率比在 Second 曲线中的查询效率高；而第二组查询（$Q_{10} \sim Q_{17}$）的情况恰恰相反，它们在 Second 曲线中的查询效率比在 First 曲线中的查询效率高. 这表明根据当前的查询调整存储模式后，查询性能可以得到显著提高. 这样在历史查询被更新后，如果存储模式不适应当前查询的执行，系统性能可能会下降，此时系统可以根据新得到的历史查询重新调整存储模式，改善系统的查询性能.

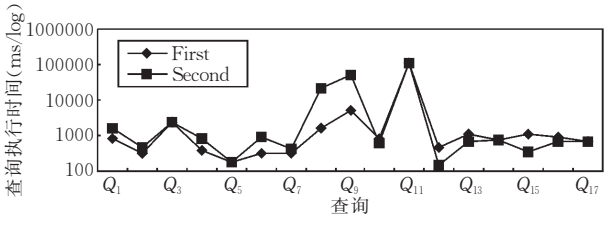

图 11 历史查询更新前后的存储模式调整效果对比

6 结 论

本文介绍了 VXMLR 系统中存储模式的自适应调整机制. 首先，分析介绍了当前在关系数据库中存储 XML 文档的方法，指出了它们的不足，然后给出了 VXMLR 中基于历史查询调整 XML 存储模式的方法，即先根据历史查询信息，确定 DTD 图中入度大于 1 的节点的映射规则，得到 VXMLR 中的基本存储模式；然后，在分析 XML 查询代价的基础上，将一组 VXMLR 查询记录中的历史查询转换为只与单个关系表相关的子查询及它们之间的连接操作；再根据这组子查询调整关系表的存储，冗余存储该关系表的相关数据或对该关系表作垂直分割；通过计算独立存储该组子查询的查询收益与存储代价，在有空间约束的条件下，将选择关系表自适应调整的问题转化为 0/1 背包问题. 本文还给出了四种调整 VXMLR 系统存储模式的策略，其中两种为自适应调整策略. 对系统的测试结果表明：对一组历史查询，使用本文的存储模式自适应调整算法后，系统总的查询效率得到显著提高；由于采用自适应调整，使得 VXMLR 系统在系统查询发生较大变化后可以及时调整存储模式以保持系统的查询处理效率.

参 考 文 献

1 Yoshikawa M., Amagasa T., Shimura T. *et al*.. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology, 2001, 1(1): 110～141

2 Florescu D., Kossmann D.. Storing and querying XML data using an RDMBS. IEEE Data Engineering Bulletin, 1999, 22(3): 27～34

3 Deutsch A., Fernandez M., Suciu D.. Storing semistructured data with stored. In: Proceedings of SIGMOD, Philadelphia, Pennsylvania, USA, 1999, 431～442

4 Shanmugasundaram J., Tufte K., Zhang C. *et al*.. Relational databases for querying XML documents: Limitations and opportunities. In: Proceedings of VLDB, Edinburgh, Scotland, 1999, 302～314

5 Tian F., DeWitt D. J., Chen J. *et al*.. The design and performance evaluation of alternative XML storage strategies. SIGMOD Record, 2002, 31(1): 5～10

6 Breslau L., Cao P., Fan L. *et al*.. Web caching and zipf-like distributions: Evidence and implications. In: Proceedings of Infocom, New York, 1999, 126～134

7 Zhou A., Lu H., Zheng S. *et al*.. VXMLR: A visual XML-relational database system. In: Proceedings of VLDB, Rome, Italy, 2001, 719～720

8 Zhang L., Zheng S., Zhou A. *et al*.. DVQ: A DTD-driven visual query interface for XML database systems. In: Proceedings of VDB, Bisbane, Australia, 2002, 385～399

9 Liang Y., Zhou A., Zheng S. *et al*.. Enhancing XML data processing in relational system with indices. In: Proceedings of

WAIM, Xi'an, China, 2001, 168～178

10 Bohannon P., Freire J., Roy P. *et al*.. From XML schema to relations: A cost-based approach to XML storage. In: Proceedings of ICDE, San Jose, California, USA, 2002, 64～75

11 Zheng S., Wen J., Lu H.. Cost-driven storage schema selection for XML. In: Proceedings of DASFAA, Kyoto, Japan, 2003, 337～344

12 Selinger P., Astrahan M., Chamberlin D.. Access path selection in a relational Database Management System. In Proceedings of SIGMOD, New York, USA, 1979, 23～34

ZHOU Ao-Ying, received his B. S and M. S. degree in computer science from Sichuan University, Chengdu, in 1985 and 1988 respectively, and his Ph. D. degree in computer software from Fudan University in 1993. He is currently a professor at the Department of Computer Science and Engineering of Fudan University. His main research interests include XML/Web data management, Web services, data stream and data mining, and peer-to-peer computing.

XU Zheng-Chuan, received his Bachelor degree and Master degree, both in Electrical Engineering, from University of Science and Technology of China in 1995 and Chinese Academy of Sciences in 2000 respectively, and his PhD in Computer Science from Fudan University in 2003, and is currently a faculty member at the School of Management. His research interests include XML data management, Web-based information systems.

GUO Zhi-Mao, received his Bachelor degree in computer science from Shanghai Jiaotong University in 2000, and is currently a Ph. D. candidate at the Department of Computer Science and Engineering of Fudan University. His specialty is Web database and P2P computing. His research interests include XML data management, Web services and related data management.

ZHOU Shui-Geng, received his B. E and M. E degree both in electromagnetic theory and microwave technology, from Huazhong University of Science & Technology, Wuhan, in 1988 and University of Electronic Science & Technology of China, Chengdu, in 1991 respectively, and his Ph. D. degree in computer software from Fudan University in 2000. He is currently an professor at the Department of Computer Science and Engineering of Fudan University. His research interest lies in information retrieval and text mining, spatial database and geographic information systems, and peer-to-peer computing.

Background

This paper is supported by National Natural Science Foundation of China under projects titled "Publishing, Exchange and Integration of Data on the WEB Based on XML" and "XML Documents Management Systems".

This projects are focusing on the edging field in database and information process, including, 1) Storage and Query of XML documents; 2) Transformation between XML data and traditional data; 3) Publishing of XML documents; 4) Data Integration and exchange based on XML.

In pursuit of the goal, methods and techniques for efficiently storing, retrieving and normalizing XML data has being developed in our research group. For example, we have get some achievements on efficient and effective methods on XML data storage and querying, transformations between XML data and relational databases, publishing of relational data in XML style, and etc..

As for XML storage and querying, We take relational databases as the underlying storage bases, and transform users' queries on XML into SQL statements, then the SQL statements are executed by RDBMS and the result XML document can be achieved by reconstructing the record set returned. Further, we studied the transformation process from XML queries to SQL, and implemented a novel XML-Relation management system, VXMLR. This paper presents some techniques which could adjust automatically the storage schema of VXMLR system based on historical queries.