以目标节点为导向的 XML 路径查询处理*

王 静[1], 孟小峰[2], 王 宇[2+], 王 珊[2]

[1](中国科学院 计算技术研究所,北京 100080)

[2](中国人民大学 信息学院,北京 100872)

Target Node Aimed Path Expression Processing for XML Data

WANG Jing[1], MENG Xiao-Feng[2], WANG Yu[2+], WANG Shan[2]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Information School, Renmin University of China, Beijing 100872, China)

+ Corresponding author: Phn: +86-10-62515575, Fax: 86-10-62519453, E-mail: sandpiperwy@yahoo.com.cn

Received 2003-12-25; Accepted 2004-06-10

Wang J, Meng XF, Wang Y, Wang S. Target node aimed path expression processing for XML data. *Journal of Software*, 2005,16(5):827−837. DOI: 10.1360/jos160827

Abstract: XML query languages take complex path expressions as their core. To facilitate path expression processing, the processing strategy based on path decomposition and structural join operation needs to be investigated more deeply. In this paper, a target node aimed at path expression processing framework for XML data is proposed. This approach makes use of the extended basic operations to reduce the number of join operations. In the procedure of path decomposition and query plan selection, target node in the query tree is utilized to avoid the transfer of the intermediate results. In addition to decomposition rules and strategies, a set of extended basic operations and implementation algorithms are proposed. Preliminary experiments indicate this approach has good performance. It provides path query processing with more choices.

Key words: XML query processing; path expression; structural join; selective structural join; path index

摘 要: XML 查询语言将复杂路径表达式作为核心内容.为了加速路径表达式处理,基于路径分解和结构连接操作的处理策略需要更深入的研究.以目标节点为导向的 XML 路径查询处理框架被提了出来.该方法利用了扩展基本操作来减少连接操作的数目.在路径分解和查询计划选择的过程中,利用查询树中的目标节点来避免中间结果的传递.除了分解规则和策略以外,提出了一组扩展的基本操作和实现算法.初步的实验结果显示,该方法具有良好的性能.它为路径查询处理提供了更多的选择.

* Supported by the National Natural Science Foundation of China under Grant Nos.60073014, 60273018 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA116030 (国家高技术研究发展计划(863)); the Key Project of Ministry of Education of China under Grant No.03044 (国家教育部科学技术重点项目); the Excellent Young Teachers Program of Ministry of Education of China (教育部优秀青年教师资助计划)

作者简介: 王静(1975−),女,山西襄垣人,博士,主要研究领域为数据库与知识库,XML 数据管理;孟小峰(1964−),男,博士,教授,博士生导师,主要研究领域为数据库与知识库,Web 数据管理;王宇(1973−),女,博士生,主要研究领域为数据库与知识库,XML 数据管理;王珊(1944−),女,教授,博士生导师,主要研究领域为数据库与知识库,数据仓库.

关键词: XML 查询处理;路径表达式;结构连接;选择性结构连接;路径索引
中图法分类号: TP311 文献标识码: A

随着 XML[1]标准被广泛接收和采用,XML 数据的管理和查询问题也引起了人们的重视,成为研究的热点.尽管 XML 可以描述非常复杂的结构,其本质仍然是树状的数据.针对 XML 的查询,学者们已经提出了多种 XML 查询语言,例如 XPath[2],XQuery[3]等,这些查询语言都将路径表达式作为核心内容.针对路径查询的处理问题,人们已经进行了大量的研究工作.在树状的 XML 数据中匹配路径查询的基本方式是对数据进行导航式的遍历,文献[4,5]对这种方式进行了探讨,它简单、直接,但执行效率不能得到保证,尤其是在大数据量的情况下.导航式遍历方法的低效性促使了类似于关系数据库中"一次一集合"的路径查询计算策略的出现.目前被广泛接受的分解连接查询执行策略的基本思路是,首先定位路径查询树中每个节点的候选元素节点集合,然后通过结构连接操作组合这些中间结果来生成最后的结果.采用这种策略会产生大量的结构连接操作.目前,这方面的工作主要集中在高效的结构连接算法上[6−9],而对路径查询的整体处理框架的研究较少.文献[7]提出了对正则路径表达式的分解计算方法,但只针对没有分支的路径查询,而且大量的结构连接操作是该方法不可避免的.文献[10]从信息过滤的角度研究了如何对路径查询进行分解,建立对路径查询的索引,考虑的问题不同.在基本操作的基础上,设计合理的路径分解和计算框架是一个需要进一步研究的问题.

针对这个问题,结合在 Native XML 数据管理系统 Orient-X 中的实际考虑,本文提出了一个以目标节点为导向的路径查询处理框架.该方法充分利用基本操作的支持,增大了基本查询片段的粒度,从而减少了结构连接的数目.

本文第 1 节给出一些基本的概念.第 2 节描述路径查询的分解方法.第 3 节针对分解后的路径查询,探讨查询计划的生成.第 4 节描述查询中用到的扩展基本操作以及操作的具体实现.第 5 节分析实验结果.第 6 节对相关工作进行概述.第 7 节对全文工作进行总结,并展望未来的工作.

1 基本概念

在本节,我们首先给出一些基本概念的定义,这些概念在后面的描述中会用到.

XML 数据的路径查询可以描述为一棵查询树,其形式化的描述如下:

定义 **1**. 针对 XML 数据的路径查询可以表示为一棵查询树 $Q=(V,E,Root,predicate)$,V 是树中节点的集合,$Root$ 是查询树的根.E 是树中节点之间边的集合,边分为两种,分别表示父子包含关系和祖先后代包含关系.除了根节点之外,函数 $predicate$ 赋给查询树中的每个节点一个谓词条件,该条件针对元素的名字、属性值及文本值等.

这个查询树所表示的路径表达式是 XPath[2]的子集.在下面的章节中,为了简单起见,查询的定义简化为 $Q=(V_Q,E_Q)$,V_Q 表示节点,E_Q 表示节点间的边.

在实际的查询树中,往往只有一个节点在数据中的映射节点是查询需要的输出结果,其余节点之间的边只是对该节点的条件约束.基于这样的考虑,我们给出如下的一些定义.

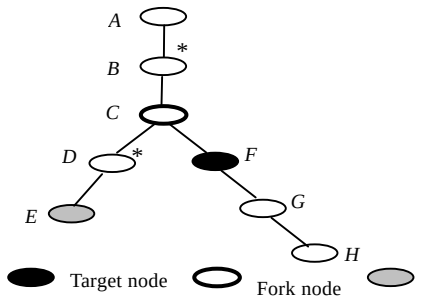
定义 **2**. XML 查询树中存在一个节点 n,它在数据中的映射是查询的最终输出结果,该节点称为查询的目标节点.

定义 **3**. 在 XML 查询树 Q 中,从根节点到目标节点之间的路径称为查询的主路径.

定义 **4**. 在 XML 查询树 Q 中,孩子节点数目大于 1 的节点(即出现路径分叉的节点)称为分支节点.

定义 **5**. 在 XML 查询树 Q 中,如果节点上除了针对元素名的谓词条件外,还定义了针对元素值或元素属性值的谓词条件,这样的节点称为值谓词节点.

考虑图 1 中给出的查询树例子,它试图找到研究领域包括数据库的教授在会议上所发表的论文.节点 F 是查询的目标节点,从节点 A 到 F 的路径是查询的主路径,节点 C 是分支节点,而节点 E 上带有针对元素文本值的谓词条件,是值谓词节点.



A.ElementName="department"
B.ElementName="researchgroup"
C.ElementName="professor"
D.ElementName="interests"
E.ElementName="area" and *E*.TextValue="DB"
F.ElementName="paper"
G.ElementName="conference"
H.ElementName="supporter"

Fig.1 An example of query tree

图 1 查询树例子

2 路径查询的分解计算

根据实际的查询需求以及底层访问方法的支持,我们提出了自己的查询计算框架.我们所研究的查询处理框架基于如下的两个前提:(1) 查询树的目标节点是查询的输出结果.在查询树中,只有目标节点在数据中的映射节点是查询的输出结果,整个查询树的计算是以目标节点为导向的.(2) 路径索引的支持.充分利用路径索引,尽量避免不必要的结构连接操作,从而减少计算代价,这是我们的一个指导思想.

2.1 查询分解状态

为了描述查询分解,首先给出两个定义:

定义 **6**. 给定一个路径查询 $Q=(V_Q,E_Q)$,一个查询片段 N 是满足如下条件的 V_Q 中节点的集合:

(1) $N \subseteq V_Q$;

(2) $\forall u,v \in N$,如果存在一个节点 w 位于从 u 到 v 的路径上,则 $w \in N$.

定义 **7**. 给定一个路径查询 $Q=(V_Q,E_Q)$,从 Q 中导出的一个查询分解状态是一棵树 $D=(V_D,E_D)$,V_D 中的每个节点 n 对应于 Q 的一个查询片段 N.D 满足如下的条件:

(1) $\forall n,m \in V_D, N \cap M = \varnothing$;

(2) $V_Q = \bigcup_{n \in V_D} N$;

(3) $E_D \subseteq E_Q$;

(4) $\forall n \in V_D, \forall u,v \in N, (u,v) \notin E_D$.

查询片段是可以借助索引等方式的支持进行快速计算的基本单位,各个查询片段之间通过结构连接操作来组合其执行的中间结果.对一个路径查询而言,根据查询片段划分的不同,可能的查询分解状态有很多.具体的查询片段的划分与底层的访问支持方式有着密切的关系.

2.2 简单路径分解

查询片段的确定是查询分解的关键.通过考虑查询的实际情况和已有的访问方法,我们采用如下的一些启发式规则来确定查询片段.

规则 **1**. 利用结构连接来处理不确定路径.

当查询树 Q 中出现了表示祖先-后代关系的边时,则两端节点之间可能的路径是任意的,例如图 1 中节点 A 和 B 之间带"*"的边.这种不确定的路径的匹配无论是在数据中,还是在路径索引中都是代价很大的.利用结构连接来直接判断候选节点之间的祖先-后代关系是解决不确定路径匹配的较好方法.基于这样的认识,分解产生的查询片段中不应当包含表示祖先-后代关系的边,该类边只能出现在查询片段之间.

规则 **2**. 查询片段不支持分支节点.

我们的基本访问方法不能直接支持带有分支节点的路径查询,所以分支节点不能作为查询片段所对应路径的中间节点.

规则 **3**. 值谓词节点作为查询片段的末端节点.

为了方便结合路径索引和值索引来计算值谓词条件,我们规定值谓词节点只作为查询片段的末端节点.多个值谓词节点之间的关系通过结构连接来实现.

根据如上的 3 条启发式规则,我们可以给出一个特定的查询分解状态.

定义 **8**. 给定一棵查询树 $Q=(V_Q,E_Q)$,如果 Q 中的一条路径 $p=\langle v_1,v_2,\ldots,v_n\rangle$满足如下的条件,则 p 是 Q 的一条简单路径:

(1) 对 $i=1,\ldots,n, v_i \in V_Q$, v_i 是 v_{i+1} 的父亲节点;

(2) 路径中相邻两个节点之间的边(v_i,v_{i+1})不表示祖先-后代关系;

(3) 如果存在 v_i 是 Q 中的分支节点或值谓词节点,则 $i=n$.

从定义 8 可以看出,查询树中的简单路径不包括祖先-后代结构关系,分支节点和值谓词节点只能出现在路径末端的路径,它的计算可以直接通过路径索引的查询来完成.

定义 **9**. 给定一棵查询树 $Q=(V_Q,E_Q)$,如果一个路径集合 $P=\{p|p$ 是查询树 Q 中出现的路径$\}$满足条件:

(1) p 是 Q 中的简单路径;

(2) 查询树 Q 中的每个节点至少包含在一条路径中,则 P 是 Q 的一个简单路径分解.

如果 P 还满足第 3 个条件:

(3) P 的每条路径 p 都是最长的,即 Q 中不存在另一个更长的简单路径包含 p,则 P 是 Q 的一个最小简单路径分解.

可以看出,最小简单路径分解是查询树的所有简单路径分解中包含简单路径数目最少的,而且最小简单路径分解是唯一的.图 2(a)和图 2(b)给出了两个可能的简单路径分解例子,图中虚线包围的区域是一个简单路径的范围.图 2(b)所给出的是最小简单路径分解,其中的每个简单路径都不可能被更长的简单路径所包含.

依据最小简单路径分解的概念,我们可以得到相应的查询分解状态.给定一棵查询树 Q 及其最小简单路径分解 P,P 中的每条简单路径对应于一个查询片段,查询片段之间的边则由它们所包含的查询节点之间的边来决定.图 2(c)给出了根据图 2(b)中的最小简单路径分解得到的查询分解状态.

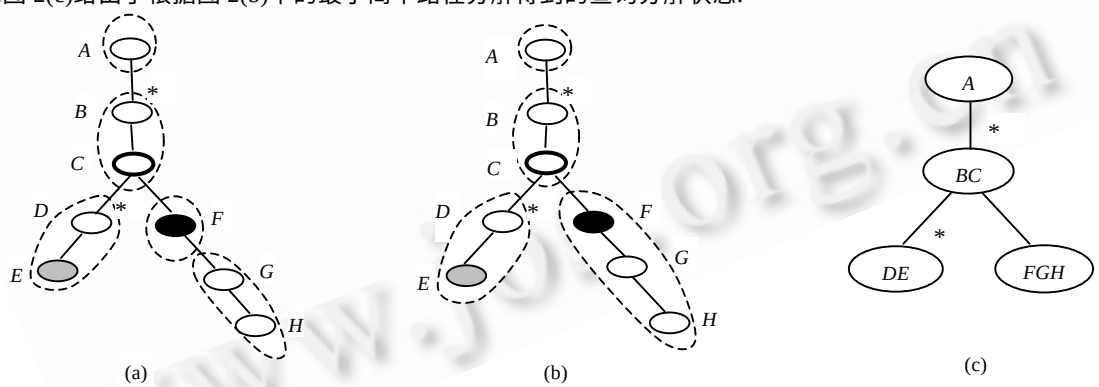


Fig.2 Path decomposition of query tree

图 2 查询树的路径分解

3 查询计划的选择

根据最小简单路径分解所得到的查询分解状态是查询执行的基础,需要经过进一步的转化才能成为查询计划.具体的查询计划的确定是一个复杂的优化问题,我们不作详细的探讨,只对其中的关键问题加以说明.

3.1 查询片段的状态确定

在我们的查询分解状态中,每个查询片段被看成是原子的,可以通过直接的访问方法得到中间结果.我们对查询片段的具体状态给出明确的描述,为其确定具体操作符的选择.每个查询片段的状态描述包括(LabelPath,

Output,Operator),其中 LabelPath 是查询片段对应的简单路径,Output 是查询片段所需输出的结果所包括的查询节点,而 Operator 则是根据前两者为该查询片段所选择的具体操作符.

查询片段输出的结果作为中间结果将会与结构相关的其他中间结果进行结构连接,所以查询片段的输出结果应当保留将用于进一步连接的元素节点信息,或者是查询最后输出结果的内容.查询片段输出的确定是根据查询片段在查询分解状态树中的边,按照如下的规则来进行.

给定一个路径查询 $Q=(V_Q,E_Q)$,从 Q 中根据最小简单路径分解得到的查询分解状态 $D=(V_D,E_D)$,D 中任意一个查询片段 N 的输出结果由所有满足如下条件的查询节点 u 所对应的元素组成:

(1) $u \in N$;

(2) $\exists v \in V_Q, v \notin N, (u,v) \in E_Q$;或者

(3) u 是 Q 的目标节点.

一旦确定了输出结果,对每个查询片段就可以根据其对应的简单路径的情况指定相应的基本操作.

3.2 结构连接顺序的选择

在以路径查询的目标节点为查询结果的前提下,如果每个结构连接操作只产生下一步操作所需的数据作为结果,可以大大减小中间结果的规模,避免不必要的中间结果传递.基于这样的观察,我们以目标节点作为导向,只考虑满足这种特性的查询计划.

我们采用如下的启发式方法来选择查询计划:将目标节点所属的查询片段作为根节点,从而将查询状态树划分为若干个子树.对每个子树分别考虑以子树的根节点为输出结果的查询计划,选择最优的计划.然后考虑各个子树与根节点的可能连接计划.目标节点的每个子树形成了一个查询子树,其根节点是子树的目标节点.为每个子树选择查询计划是一个递归的过程.

这样产生的查询计划的数目是相当有限的,只与分支节点的不同连接顺序选择有关.图 3 描述了一个查询分解状态可能有的查询计划.图 3(a)是将目标查询片段作为根节点的查询分解状态树,*FGH* 对应的是目标查询片段.它对应的两个可能的查询计划如图 3(b)和图 3(c)所示.

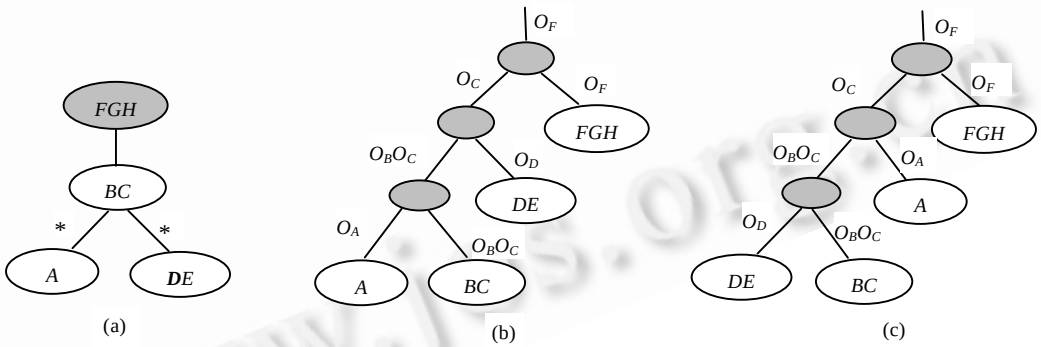


Fig.3 Selection of query plan

图 3 查询计划的选择

4 扩展的基本操作

在基于最小简单路径分解的计算框架中,由于以目标节点为导向和简单路径原子化的前提存在,原有的基本操作不能满足需求,需要引入新的操作.

4.1 扩展的索引查询操作

在我们基于简单路径的分解框架中,查询片段对应的简单路径是一个原子操作,需要路径索引的支持,直接得到满足简单路径关系的结果,从而避免多个二元结构连接操作.根据简单路径的定义,为了有效地支持其查询,需要扩展的索引查询操作包括:(1) SimplePath:对给定的简单标记路径 $L_1/L_2/\ldots/L_n$,返回满足路径约束条件的指定结果,包括:L_1 对应的元素节点集合,L_n 对应的元素节点集合,或者(L_1,L_n)对应的元素节点对集合.

(2) PathWithPredicate:对给定的标记路径 $L_1/L_2/\ldots/L_n$[值谓词条件],返回满足路径约束条件和值谓词条件的指定结果,包括:L_1 对应的元素节点集合,L_n 对应的满足值谓词条件的元素节点集合,或者(L_1,L_n)对应的元素节点对集合.

文献[11]中详细论述了利用 SUPEX 索引实现上述操作的方法.

4.2 选择性结构连接操作

结构连接是 XML 查询处理中的核心操作,目前所考虑的结构连接操作是一种完全结构连接操作,即结构连接所输出的结果是参加连接的两个输入集合中满足条件的元组的合并.在我们的计算框架中,查询树的计算是以目标节点为导向的,而不必给出全连接的结果.选择性结构连接操作只输出需要保留的部分作为结果,对无用查询结果的剔除可以尽早进行,其定义如下:

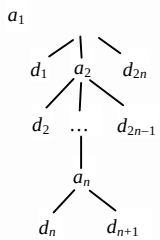
定义 **10**. 给定两个输入集合 $A=\{(a_1,a_2,\ldots,a_m)\}$和 $D=\{(d_1,d_2,\ldots,d_n)\}$,$A$ 和 D 分别是 m 维和 n 维元组的集合,对指定的潜在祖先 a_i,潜在后代 d_j 以及输出结果定义,选择性结构连接操作产生的结果集合为$\{(x_1,x_2,\ldots,x_p)|a_i$ 是 d_j 的祖先;$x_k\in\{a_1,\ldots,a_m\}$或者 $x_k\in\{d_1,\ldots,d_n\}$,且 x_k 在输出结果定义中,$k=1,\ldots,p\}$.

4.3 选择性结构连接算法实现

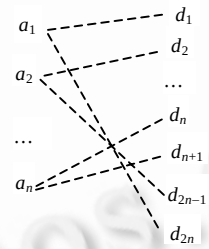
本节给出了两类选择性结构连接算法:排序合并和基于区域划分.

4.3.1 选择性排序合并结构连接算法

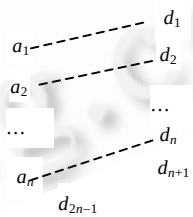
根据输出结果是 A 或 D 中的元素,选择性排序合并结构连接(selective sort merge join,简称 SSMJ)算法有两个:SSMJ-Anc 和 SSMJ-Des.它们利用了只输出一个输入集合中元素的特点,避免了对某些元素的处理,从而避免了完全结构连接的排序合并算法可能产生的对输入集合的多遍扫描.这两个算法在最坏情况下对两个输入集合各扫描一遍.图 4 和图 5 分别给出了完全结构连接的按祖先和后代排序合并算法的最坏情况,以及 SSMJ-Anc 和 SSMJ-Des 在这两种情况下的效率.在图 4(a)所描述的数据分布情况下,图 4(b)描述了按祖先排序合并算法进行完全结构连接的对 D 的多遍扫描,而图 4(c)中的 SSMJ-Anc 算法只需从 d_1 到 d_n 的扫描比较.图 5 中的情况也相类似.需要指出的是,SSMJ-Anc 和 SSMJ-Des 只适用于祖先-后代关系的计算.



(a) XML tree
(a) XML 树

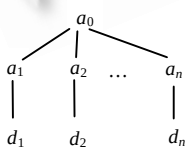


(b) Sort merge join by ancestor nodes
(b) 按祖先节点排序合并结构连接

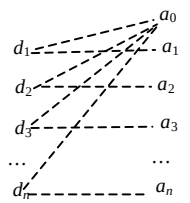


(c) SSMJ-Anc
(c) SSMJ-Anc

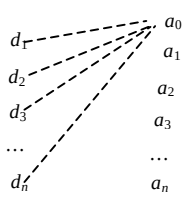
Fig.4 A case for SSMJ-Anc algorithm
图 4 SSMJ-Anc 算法的例子



(a) XML tree
(a) XML 树



(b) Sort merge join by descendant nodes
(b) 按后代节点排序合并结构连接



(c) SSMJ-Des
(c) SSMJ-Des

Fig.5 A case for SSMJ-Des algorithm
图 5 SSMJ-Des 算法的例子

4.3.2 基于区域划分的选择性结构连接算法

针对输入数据集合无序的情况,我们已经提出了基于区域划分的结构连接算法(range partitioning join,简称 RPJ)[12],实现了完全结构连接操作.在 RPJ 算法的基础上,针对选择性结构连接操作,我们提出了基于区域划分的选择性结构连接算法(selective range partitioning join,简称 SRPJ).该类算法包括两个:输出结果限定在 A 上的 SRPJ-Anc 和输出结果限定在 D 上的 SRPJ-Des.

4.3.2.1 SRPJ-Anc 算法

SRPJ-Anc 充分利用元素编码的特点,尽早地对 A 中节点进行判断,产生结果.算法分为 3 个阶段:

(1) 子集合划分阶段.首先对后代节点集合 D 进行划分,所采用的划分方法与 RPJ 相同.当对祖先节点集合 A 进行划分时,可以利用祖先节点的特点来产生部分结果.如果 A 中的节点 a 的区域编码完全覆盖了一个子区间 R_i,只要 R_i 对应的子集合 D_i 中的元素节点数目不为 0,D_i 中所有的节点都是 a 的后代节点.利用这个特点,我们在对 A 中的节点进行划分时就可以判断一些节点是否有后代节点,具体的判断规则为:假设 A 中的一个节点 a 的区域编码完全覆盖了 n 个子区间,如果这些子区间所对应的 D 的子集合中至少存在一个不为空,那么在 D 中一定存在 a 的后代节点,即 a 应当作为结果输出.对于确定为输出结果的 A 中节点,就不再划分到子集合中进行处理;对于那些不能确定为结果的 A 中节点,只需将其划分到与区域编码部分相交的子区间所对应的子集合中,这样的子集合的最大数目为 2.

(2) 连接阶段.对于那些在子集合划分阶段不能确定的 A 中的节点,需要实际的连接来进行检查.对每一对子集合 A_i 和 D_i,分别进行连接.由于 A 中的一个节点可能被划分到两个子集合中,所以它可能在结果中出现两次,即连接阶段产生的结果集中可能出现重复元素.

(3) 去重阶段.子集合划分和连接阶段已经产生了所有的输出结果,但连接阶段所产生的结果中可能会出现重复值.该阶段的主要任务是合并各个子集合对的连接结果,去掉重复值.

4.3.2.2 SRPJ-Des 算法

SRPJ-Des 的基本思想与 SRPJ-Anc 类似,不同的是考虑的目标是集合 D.算法分为两个阶段:

(1) 子集合划分阶段.首先对祖先节点集合 A 进行划分,所采用的划分方法与 RPJ 相同,不同的是对每个子集合 A_i 额外记录该子集合中区域编码完全覆盖其对应子区间的节点个数 V_i.如果 A 中节点 a 的区域编码完全覆盖了一个子区间 R_i,则 D_i 中所有的节点都是 a 的后代节点.利用这个特点,在对后代节点集合 D 进行划分时可以判断一些节点是否有祖先节点,判断规则为:假设 D 中的一个节点 d 应当划分到 D_i,如果对应的 A_i 的 V_i 值不为 0,则 A_i 中一定存在 d 的祖先节点,即 d 应当作为结果输出.对于确定为输出结果的 D 中节点,就不再划分到子集合中进行处理;对于那些不能确定为结果的 D 中节点,仍按照 RPJ 算法中的划分方法划分到子集合中,进行进一步的处理.

(2) 连接阶段.对于在子集合划分阶段没有确定的 D 中节点,连接阶段进行进一步的处理.根据装入内存的子集合的不同,可以采用相应的连接算法.由于 D 中每个元素最多只被划分到一个子集合中,所以连接阶段产生的结果中没有重复.两个阶段产生的结果可以直接合并生成最后的输出结果.

5 实验结果和分析

为了对本文中所提出方法的有效性进行验证,我们进行了初步的实验.本节对实验的结果进行了描述和分析.

5.1 实验设置

我们的实验在 Native XML 数据管理系统 Orient-X 的基础上进行,所有的算法都用 C++编程语言来实现.所有的实验在一台 Duron 1.0GHz,256M RAM,40G 硬盘的 PC 上运行,底层操作系统是 Windows XP.

我们选择执行时间为评价指标.这里所给出的执行时间都是运行实验多次,去掉最高和最低值后得到的平均执行时间.

5.2 选择性结构连接的有效性

选择性结构连接操作在我们的路径查询框架中具有重要的作用,本节我们结合所提出的多种实现算法对其进行性能上的分析.

我们采用 IBM XML Generator 生成了大小为 113M 的实验文档,所用的 DTD 如图 6 所示,其中各个元素的个数在表 1 中给出.我们采用了表 2 所示的 6 个查询,它们可以分为两类:Q1 到 Q4 是简单结构关系查询;Q5 和 Q6 是复杂查询,用来反映包含多个连接操作的查询的执行情况.除了人工生成的数据集以外,我们也在真实的数据集 DBLP 和 XMark 上进行了实验,实验结果类似.

```
⟨!ELEMENT manager (name, (manager | department | employee)+)⟩
⟨!ELEMENT department (name, email?, employee+, department*)⟩
⟨!ELEMENT employee (name+, email?)⟩
⟨!ELEMENT name (#PCDATA)⟩
⟨!ELEMENT email (#PCDATA)⟩
```

Fig.6 The DTD of synthetic data set

图 6 人工数据集的 DTD

Table 1 Description of synthetic data set

表 **1** 人工数据集的描述

Element	Number
Manager	38
Department	286 459
Employee	543 685
Name	1 111 390
Email	59 946

Table 2 Description of queries of synthetic data set

表 **2** 人工数据集的查询描述

Query	Path expression	Result (Ancestor)	Result (Descendant)
Q1	manager//employee	38	543 685
Q2	department//employee	286 459	543 631
Q3	department//email	34 223	59 929
Q4	employee//email	31 391	31 391
Q5	department//employee//email	10 477	31 374
Q6	manager//department//email	14	59 929

5.2.1 简单结构关系查询

这里我们用表 2 中的 4 个简单结构关系查询 Q1 到 Q4 来比较不同的选择性结构连接实现算法的性能.我们实现了 5 类算法:选择性排序合并连接(SSMJ),Stack-Tree-Filter(STF),Stack-Tree(ST),选择性区域划分连接(SRPJ)和区域划分连接(RPJ),每类算法包括结果限定在祖先和后代的两个算法.STF 算法是对 Stack-Tree 类的算法进行了改写,使其只输出指定的结果.ST 算法是在 Stack-Tree 类的算法后附加了一个投影到指定输出的过程,RPJ 也是类似.我们将这 5 种算法分为两类进行比较:基于排序合并和基于划分,这是因为我们在计算基于排序合并算法的执行时间时没有考虑排序的时间,在这种情况下,基于排序合并的算法效率明显高于基于划分的算法.此外,实验的目的是想反映选择性结构连接算法与其对应的完全结构连接算法加投影的性能比较.

图 7 和图 8 描述了排序合并类算法的性能,分别比较了输出结果限定在祖先节点和后代节点上的算法.从图 7 中可以看出,选择性结构连接算法 SSMJ 和 STF 的性能在各个查询上均优于 ST,这说明选择性结构连接算法的实现策略在性能上优于完全结构连接加投影的实现策略.对查询 Q1,Q2 和 Q3,两种策略的执行时间相差较大,而 Q4 的执行时间则比较接近,这是由于前 3 个查询所涉及的祖先节点元素 manager 和 department 是递归定义的,而 Q4 中的 employee 则没有递归定义.当祖先元素存在递归定义时,选择性结构连接算法由于可以避免对嵌套节点的多次处理,所以较大程度地提高了执行效率.此外,SSMJ 和 STF 在各个查询上的执行时间都比较接近,SSMJ 略优于 STF,这是由于两种算法本质上是相同的,而 STF 在内存中的栈和链表处理稍复杂一些.图 8 所

描述的性能比较表现了与图 7 类似的特征.

图 9 和图 10 描述了基于划分的算法的性能比较.从图中可以看出,选择性结构连接算法 SRPJ 在各个查询上都优于基于区域划分的完全结构连接加投影的方法.除 Q4 的优势不明显以外,其他 3 个查询上 SRPJ 都大大优于 RPJ.这个特征也与排序合并类算法的表现相一致.

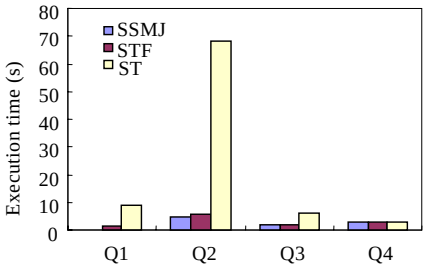


Fig.7 The result of sort merge algorithms by ancestor

图 7 输出祖先节点的排序合并类算法的结果

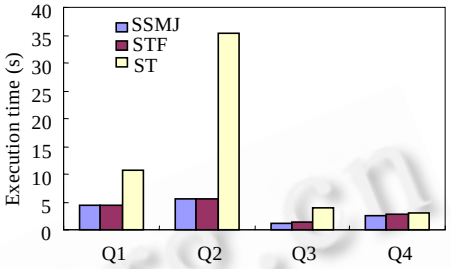


Fig.8 The result of sort merge algorithms by descendant

图 8 输出后代节点的排序合并类算法的结果

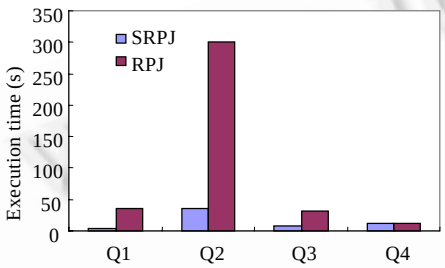


Fig.9 The result of partitioning algorithms by ancestor

图 9 输出祖先节点的划分类算法的结果

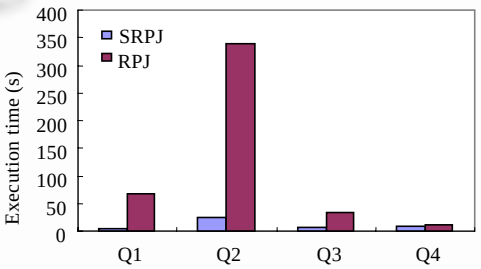


Fig.10 The result of partitioning algorithms by descendant

图 10 输出后代节点的划分类算法的结果

5.2.2 复杂查询

表 2 中的 Q5 和 Q6 是两个较为复杂的查询,每个查询包含了两个连接.我们用五类算法分别基于结果为祖先节点和后代节点实现了这两个查询.SSMJ,STF 和 SRPJ 同上节所述,而 Path-S(Path-R)则表示先采用 Stack-Tree(RPJ)算法完成连接,最后对完全连接结果进行一次投影.表 3 和表 4 分别给出了排序合并类算法和划分类算法的实验结果.从表 3 中可以看出,SSMJ 和 STF 执行时间相近,都大大优于 Path-S.而表 4 则反映了 SRPJ 与 Path-R 相比所取得的性能优势.

Table 3 The result of queries using sort merge algorithms

表 3 排序合并类算法的查询结果

Query	Ancestor (s)			Descendant (s)		
	SSMJ	STF	Path-S	SSMJ	STF	Path-S
Q5	3.1	2.94	4.83	7.68	8.19	35.78
Q6	1.08	1.29	9.11	3.07	3.43	8.63

Table 4 The result of queries using partitioning algorithms

表 4 划分类算法的查询结果

Query	Ancestor (s)		Descendants (s)	
	SRPJ	Path-R	SRPJ	Path-R
Q5	12.22	21.63	30.13	122.81
Q6	6.66	25.92	8.36	45

5.3 查询执行计划的性能

我们用初步的实验来验证所提出的查询分解处理策略的可行性.实验采用 XMark 数据集,选择了大小分别为 1M,10M 和 20M 的文档.表 5 给出了实验中所用的查询例子 QP1 和 QP2.QP1 是一个不带分支的路径查询,存在不确定路径.而 QP2 则带有分支路径,是一个树状的路径查询.对 QP1 和 QP2,我们分别采用了两个不同的执行计划来实现.连接计划通过单步的结构连接操作来完成查询,连接顺序采用逐步向目标节点靠拢的顺序.分解计划是采用本章所提出的分解策略所产生的执行计划.两个查询例子的执行结果分别见表 6 和表 7.从表 6 可以看到,除了数据为 1M 的情况以外,QP1 的分解计划的执行时间大大小于连接计划.对 QP2 来说,这种执行时间的差距更为明显.

虽然这里的连接计划并不是通过优化而选出的最优计划,但从实验结果仍然可以看出,基于分解策略的执行计划具有一定的优势,可以成为一种优先考虑的选择.

Table 5 Description of path queries

表 5 路径查询描述

Query	Path expression
QP1	/site//open_auction/bidder/increase
QP2	/site/open_auctions/open_auction[annotation/description/text]/bidder/name

Table 6 The execution time of QP1 (ms)

表 6 QP1 的执行时间(ms)

Dataset (M)	Join plan	Decomposition plan
1	15.33	<1
10	114.33	31.33
20	328	151

Table 7 The execution time of QP2 (ms)

表 7 QP2 的执行时间(ms)

Dataset (M)	Join plan	Decomposition plan
1	15.67	15.67
10	104	83.33
20	250.33	161.67

6 相关工作

路径查询的处理方面已经有大量的研究工作.继承了半结构化数据领域的研究,文献[7,8]对导航式遍历的路径查询匹配方法进行了研究.导航式遍历方法简单、直接,但执行效率不能得到保证,尤其是在大数据量的情况下.

“一次一集合”的路径查询计算策略目前被广泛接受,基于该策略的研究工作包括多个方面,结构连接算法的研究是其中的重点.目前已有的工作大体上可以分为两类:基于排序合并的算法[6−8]和基于划分的方法[9].排序合并类的算法依赖于一定的前提条件:数据集合是有序的,或者集合上存在索引.当条件不成立时,算法的效率会大大降低.文献[9]中的划分方法虽然不要求输入数据集合有序或存在索引,但只适用于其提出的 PBiTree 编码,应用范围非常有限.在结构连接操作的基础上,对路径查询的整体处理框架的研究目前还比较少.文献[7]提出了对正则路径表达式的分解计算方法,但只针对没有分支的路径查询,而大量的结构连接操作在该方法是不可避免的.文献[10]从信息过滤的角度研究了如何对路径查询进行分解,建立对路径查询的索引,从而实现对 XML 文档的高效过滤.此外,文献[13,14]对结构连接的结果估计问题进行了研究,文献[15]则针对结构连接的顺序选择问题提出了多种优化算法.

除了基于结构连接的策略以外,还有一些研究工作从其他角度出发对路径查询处理进行了探讨.文献[16]针对路径查询的匹配,提出了新的整体树状连接算法,不会产生中间结果.采用这种策略处理路径查询的问题在于将整个的执行由连接算法控制,不能进行优化和选择.

7 总 结

路径查询的计算是 XML 查询处理中的关键问题,本文结合实际的系统,提出了路径查询的计算框架.首先给出了路径查询的一些相关定义,在此基础上提出了对查询树的最小简单路径分解.针对由最小简单路径分解导出的查询分解状态,提出了一些扩展的操作符,包括选择性结构连接操作和扩展的索引查询操作,并分别给出了具体的实现方法.

我们的工作是在 Native XML 数据管理系统中查询计算的环境下进行的.在路径查询的研究领域中,仍然有许多问题有待于进一步的探讨,如基于代价的查询优化、更多的基本操作(如多路结构连接等)、新的访问方法等.未来我们将在路径查询的基础上,对 XQuery 的查询处理进行进一步的研究.

References:

[1] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, eds. Extensible markup language (XML) 1.0 (2nd Edition). W3C Recommendation, 2000. http://www.w3.org/TR/2000/REC-xml-20001006

[2] Clark J, DeROse S, eds. XML Path language (XPath) Version 1.0. W3C Recommendation, 1999. http://www.w3.org/TR/1999/REC-xpath-19991116

[3] Chamberlin D, Florescu D, Robie J, Simeon J, Stefanescu M, eds. XQuery: A query language for XML. W3C Working Draft, 2001. http://www.w3.org/TR/2001/WD-xquery-2001215

[4] Goldman R, McHugh J, Widom J. From semisturctured data to XML: Migrating the lore model and query language. In: Proc. of the 2nd Int'l Workshop on the Web and Databases (WebDB'99). 1999. http://www-rocq.inria.fr/~cluet/WEBDB/lore.ps

[5] McHugh J, Widom J. Query optimization for XML. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1999. 315−326.

[6] Zhang C, Naughton J, DeWitt D, Luo Q, Lohman G. On supporting containment queries in relational database management systems. In: Timos S, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2001. 425−436.

[7] Li QZ, Moon B. Indexing and querying XML data for regular path expressions. In: Apers PMG, Atzeni P, Ceri S, Paraboschi S, Ramamohanarao K, Snodgrass RT, eds. Proc. of the 27th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2001. 361−370.

[8] AI-Khalifa S, Jagadish HV, Koudas N, Patel JM, Srivastava D, Wu YQ. Structural joins: A primitive for efficient XML query pattern matching. In: Agrawal R, Dittrich K, Ngu AHH, eds. Proc. of the 18th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2002. 141−152.

[9] Wang W, Jiang H, Lu H, Yu X. PBiTree coding and efficient processing of containment join. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2003. 391−402.

[10] Chan C-Y, Felber P, Garofalakis M, Rastogi R. Efficient filtering of XML documents with XPath expressions. In: Bernstein PA, *et al*, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2002. 354−379.

[11] Wang J, Meng XF, Wang S. SUPEX: Schema guided path index for XML data. Computer Science, 2002,29(8A):25−38 (in Chinese with English abstract).

[12] Wang J, Meng XF, Wang S. Structural join of XML based on range partitioning. Journal of Software, 2004,15(5):720−729 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/720.htm

[13] Wu YQ, Patel J, Jagadish HV. Estimating answer sizes for XML queries. In: Jensen CS, *et al*, eds. Proc. of the 8th Int'l Conf. on Extending Database Technology. Prague: Springer-Verlag, 2002. 590−608.

[14] Wang W, Jiang H, Lu H, Jeffrey XY. Containment join size estimation: Models and methods. In: Halevy AY, Ives ZG, Doan A, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 145−156.

[15] Wu YQ, Patel J, Jagadish H. Structural join selection for XML query optimization. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2003. 443−454.

[16] Nicolas B, Nick K, Divesh S. Holistic Twig joins: Optimal XML pattern matching. In: Franklin MJ, *et al*, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 310−321.

附中文参考文献:

[11] 王静,孟小峰,王珊.SUPEX:一种基于模式的 XML 路径索引.计算机科学, 2002,29(8A):25−38.

[12] 王静,孟小峰,王珊.基于区域划分的 XML 结构连接.软件学报,2004,15(5):720−729 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/720.htm