1000-9825/2004/15(05)0720 ©2004 Journal of Software 软 件 学 报 Vol.15, No.5

基于区域划分的 XML 结构连接*

王 静[1+], 孟小峰[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)

Structural Join of XML Based on Range Partitioning

WANG Jing[1+], MENG Xiao-Feng[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: jwang_lq@yahoo.com.cn
Received 2003-03-02; Accepted 2003-06-26

Wang J, Meng XF, Wang S. Structural join of XML based on range partitioning. *Journal of Software*, 2004, 15(5):720~729.
http://www.jos.org.cn/1000-9825/15/720.htm

Abstract: Structural join is the core operation in XML query processing, and catches the research community's attention. Efficient algorithm is the key of efficient query processing. There have been a number of algorithms proposed for structural join, and most of them are based on one of the following assumptions: the input element sets either have indexes or are ordered. When these assumptions are not true, the performance of these algorithms will become very bad due to the cost of sorting input sets or building index on the fly. Motivated by this observation, a structural join algorithm based on range partitioning is proposed in this paper. Based on the idea of task decomposition, this algorithm takes advantage of the region numbering scheme to partition the input sets. The procedure of the algorithm is described in detail, and its I/O complexity is analyzed. The results of the extensive experiments show that this algorithm has good performance and is superior to the existing sort-merge algorithms when the input data are not sorted or indexed. It can provide query plans with more choices.
Key words: XML query processing; path expression; numbering scheme; structural join

摘 要: 结构连接是 XML 查询处理的核心操作,受到了研究界的关注.高效的算法是高效查询处理的关键.目前已经提出了许多结构连接的算法,它们中的大多数都基于如下的前提条件之一:输入元素集合存在索引或者

* 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 the 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 (国家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,博士生,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系统,Web 数据管理,移动数据管理;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系统,数据仓库.

有序.当这些条件不成立时,由于对输入数据临时排序或建索引的代价,这些算法的性能会大大下降.基于这样的观察,提出了一种基于区域划分的结构连接算法.该算法基于任务分解的思想,利用区域编码的特点对输入集合进行划分.给出了详细的算法设计,并对算法的 I/O 复杂性进行了分析.大量的实验结果显示,该算法具有良好的性能,在输入数据无序或没有索引的情况下优于现有的排序合并算法,可以为查询计划提供更多的选择.

关键词: XML 查询处理;路径表达式;编码方法;结构连接

中图法分类号: TP311 **文献标识码:** A

XML(extensible markup language)[1]数据作为一种数据表现和交换的格式,正在赢得巨大的成功,特别是在 WWW 上被广泛接受和采用.随着 XML 的不断普及,XML 数据的管理和查询问题也越来越引起人们的重视[2,3].尽管 XML 数据表现形式灵活,可以描述非常复杂的结构,但其本质仍然是树状的模型.树中的每个节点对应于一个元素,树中的边描述了元素之间的父子关系.

针对这种树状数据的查询,学者们已经提出了多种 XML 查询语言,例如 XPath[4],XQuery[5]等.这些查询语言尽管各有特点,但它们都将路径表达式作为重要的组成部分.为了高效地计算路径表达式,人们提出了一种查询处理策略.其主要思想是将一个复杂的查询模式分解成为若干个二元基本结构关系的集合,首先计算二元基本结构关系,然后将基本的匹配结果组合起来.在这种处理策略下,基本结构关系(包括父子关系和祖先-后代关系)的计算成为查询处理的关键操作.这种操作被称为结构连接(或包含连接).

对结构连接算法的研究近年来备受重视,目前已经提出的算法大致可以分为两类:排序合并[6~10]和划分方法[11].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.该类算法在数据有序和存在索引的情况下有着很好的性能,但问题是输入集合有序和存在索引这样的前提条件在实际的查询处理中并不是总成立.在实际的 XML 数据库中,结构连接算法的输入集合可能来自元素索引、路径索引、值索引、中间结果等,并不是在所有的索引结构中元素都可以按照编码的顺序排列,其他操作的中间结果的情况也是多种多样的.当数据有序和存在索引这样的条件不成立时,这些算法需要临时对数据排序或建立索引,从而使算法的效率大大降低.文献[11]中提出的划分方法虽然不要求输入数据集合有序或存在索引,能够取得较优的 I/O 效率,但其算法只适用于特殊的 PBiTree 编码,而不能应用于被广泛采用的区域编码方法,应用范围非常有限.

基于这样的观察,我们在本文中提出了基于区域划分的结构连接算法.该算法基于目前被广泛采用的区域编码方法,不要求数据有序或事先存在索引,能够在任意的输入数据情况下取得较优的 I/O.我们的实验结果显示,该算法具有很好的性能,在数据无序和不存在索引的情况下优于排序合并类的算法.

本文的主要工作如下:

(1) 提出了一种新的基于区域划分的结构连接算法,适用于无序和没有索引的数据集合的结构连接.

(2) 对算法进行了代价分析和比较.

(3) 进行了大量的实验,比较并分析了实验结果.

本文第 1 节给出了背景知识和相关工作.第 2 节阐述了算法中采用的划分方法.算法的具体描述和代价分析在第 3 节中给出.第 4 节对实验加以说明.第 5 节总结了全文.

1 背景知识和相关工作

在基于基本结构关系计算的 XML 查询处理中,有两个关键的问题:(1) 快速判断树中任意两个节点之间的结构关系,包括父子关系和祖先-后代关系;(2) 从给定的两个元素节点集合中高效地得到满足某种结构关系的所有数据.节点间结构关系的判断依赖于对节点的编码,而高效地计算两个集合中所有满足结构关系的节点对,则是结构连接所关注的问题.

1.1 节点编码

节点之间的结构关系可以根据编码进行快速判断.区域(region)编码方法[6]是目前所提出的编码方法中应用得最普遍的一种.它赋给每个元素节点一个编码(*start*,*end*),*start* 和 *end* 分别表示节点在树中的起始和结束位

置.对任意两个不同的元素节点 u 和 v,它们的编码所覆盖的区间不会部分相交,即 u 和 v 之间或者完全包含,或者不相交.本文所提出的结构连接算法基于区域编码方法,为了支持实际数据中多个文档的情况和父子结构关系的判断,将编码扩展为(*DocId*,*start*,*end*,*level*),*DocId* 是文档的标识,*level* 给出了元素节点在树中的层次.对任意的两个元素节点 u 和 v,如果满足 $u.DocId=v.DocId, u.start<v.start<u.end$,则 u 是 v 的祖先.如果还满足条件 $u.level=v.level-1$,则 u 是 v 的父亲.对元素节点的区域编码可以通过对文档树的前序遍历完成,编码可以与数据装入数据库同时进行.

与区域编码类似的还有其他一些编码方法.比如,文献[7]提出了一种基于扩展先序和后代范围的编码方法,该方法赋给树中的每个节点一个编码(*order*,*size*),*order* 与节点的前序遍历顺序一致,*size* 描述了节点可能的覆盖范围.

除了这些基于区域的编码方法以外,还有另一类编码方法.文献[12]中提出的 k-ary tree 编码方法将文档看作一个完全 k 分树,根据树的某个遍历顺序给树中节点编码.文献[11]提出了一种类似于 k-ary tree 的编码方法 PBiTree,通过增加虚拟节点将文档树嵌入到一个完全二叉树中.这种方法的优点在于,可以利用完全二叉树的优良特性来计算节点之间的结构关系,如给定节点在给定高度的祖先、节点自身在树中的层次等.

1.2 结构连接

结构连接是路径查询处理中的核心操作.给定一个潜在祖先(父亲)元素节点的集合 A 和一个潜在后代(孩子)节点的集合 D,结构连接操作的任务是找到所有的节点对(a_i,d_j),$a_i\in A,d_j\in D$,a_i 是 d_j 的祖先(父亲)节点.结构连接操作的输入数据可能来自索引扫描或其他操作所产生的中间结果,这取决于实际的 XML 数据存储和具体的查询执行计划,超出了本文的论述范围.以 native XML 数据库为例,它可能将带有编码信息的元素节点作为对象存放.为了高效地处理 XML 数据,针对元素名字的索引在存储 XML 数据的数据库中普遍存在.通过这种索引,可以找到所有具有某个元素名的节点集合.这样的集合可以作为结构连接算法的基本输入数据,而不需要访问源数据.为了描述的目的,本文只针对来自单个文档树的节点的结构连接进行了阐述,但所提出的方法同样适用于由多个文档组成的文档树集合.

目前基于区域编码所提出的结构连接算法都是基于排序合并的,要求输入集合 A 和 D 中的元素都是按照其 *start* 有序或有索引存在.文献[6]提出了一种排序合并连接算法——多谓词合并连接(multi-predicate merge join).它利用了所有的连接条件来指导合并的过程.该算法的缺点是可能会多遍扫描数据集.文献[7]中提出的 εε-Join 和εA-Join 也存在同样的问题.文献[8]中提出的 Stack-Tree 算法改进了以前提出的基于合并连接的算法,通过在内存中保留一个栈结构来达到对输入的数据集只扫描一遍的效果.文献[9]利用索引进一步改善了 Stack-Tree-Desc 算法,其主要思想是在元素的编码上建立特定的索引结构,以辅助跳过不可能发生连接的节点,从而避免对这些节点的处理.文献[10]针对文献[9]中的算法不能有效跳过祖先节点的问题,提出了一种索引结构 XR-Tree,以辅助跳过不参与连接的祖先节点.这两个算法依赖于索引的存在,而且只有在输入的祖先或后代集合中不参与连接的节点数目较多时,才会取得比 Stack-Tree 类算法更高的处理效率.

除了基于区域编码的结构连接算法以外,文献[11]针对其提出的 PbiTree 编码提出了基于划分的连接算法.划分的策略有两种:水平划分和竖直划分,分别根据元素节点在树中的高度和所在的分支进行子集合的划分.该算法不要求输入的数据有序,也不依赖于任何事先存在的索引结构,但它只适用于 PbiTree 类的编码方法,对被广泛采用的区域编码则不适用.

2 区域划分方法

区域划分结构连接算法的基本思想是任务划分,即通过将连接任务 $A\triangleright D$ 分解为若干个较小的子任务 $A_1\triangleright D_1,\ldots,A_m\triangleright D_m$ 来取得较优的 I/O 代价,同时保证 $A\triangleright D=\bigcup_{i=1}^{m}A_i\triangleright D_i$.该算法的关键是要选择合适的划分方法,以便最小化可能的元素比较次数和数据复制,同时确保所有的连接结果都被找到.本节首先给出了一些基本定义,然后对划分方法进行了具体阐述.

2.1 基本定义

在具体描述划分方法之前,我们先给出一些基本定义.

定义 1. 假设 XML 元素节点的编码范围为 $R(start,end)$,为了与元素的区域编码相区别,我们将编码范围 R 中的任意一个范围(s_i,e_i)称为一个区间 R_i.

区域编码与具体的元素节点相关,具有互不相交的特性,而区间是任意取的一段间隔,与元素没有直接的关系,如图 1 所示的 R_i 就是随意选取的一个区间.

定义 2. 给定一个区域编码为(s,e)的节点 E 和一个区间 $R_i(s_i,e_i)$,如果它们满足如下条件之一,则称 E 与区间 R_i 相交:

(1) $s<s_i<e_i<e$;

(2) $s_i\le s<e\le e_i$;

(3) $s<s_i<e<e_i$;

(4) $s_i<s<e_i<e$.

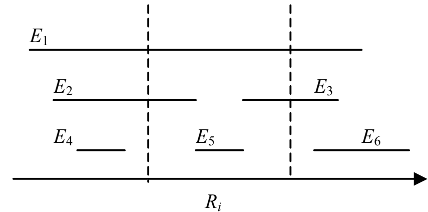

Fig.1 Relationship between numbering region and interval

图 1 编码区域和区间的关系

定义 3. 给定一个区域编码为(s,e)的节点 E 和一个区间 $R_i(s_i,e_i)$,如果它们满足 $s_i \le s \le e_i$,则区间 R_i 为 E 的开始区间.

定义 4. 给定一个节点 $E(s,e)$和一组区间 $R_i(s_i,e_i)$, $0\le i<n$. 所有与 E 相交的区间组成的集合称为 E 的相关集,所有 E 的开始区间组成的集合称为 E 的开始集.

上面的定义可以用如图 1 所示的例子来说明.图 1 中有 6 个节点和 1 个区间 R_i,其中节点 E_1,E_2,E_3 和 E_5 与 R_i 的关系都满足定义 1,即它们与 R_i 相交,此外 R_i 是 E_3 和 E_5 的开始区间.

2.2 数据集合的划分方法

基于区域编码的结构连接是基于多个不等谓词条件的,所以传统的哈希划分方法不适用于结构连接.考虑到 XML 数据区域编码和结构连接的特点,我们选择了如下的划分方法:

假设所选择的子集合个数为 m,整个元素节点的编码范围为 $R(start,end)$,则将编码区间 R 划分为 m 个子区间,每个子区间 R_i 的范围是(k_{i-1},k_i),$i=1,\ldots,m$,其中 $k_0=start$,$k_m=end$,对 $i=1,\ldots,m-1$,$k_i=k_{i-1}+\left\lfloor \frac{end-start}{m} \right\rfloor$.对 A(或 D)集合中的任意一个元素 e,如果 e 与 R_i 相交,则 e 被划分到 A_i(或 D_i)中,即 e 被划分到其相关集中的区间所对应的子集合中.

由于对编码范围的划分是连续的,所以 A 和 D 中的每个元素至少会与 1 个子区间相交,即至少会被分配到一个子集合中.此外,元素的编码范围可能与多个子区间相交,所以 1 个元素可能被分配到多个子集合中,即有数据复制的情况存在.

根据以上划分方法,A 和 D 可以被划分为指定数目的子集合,结构连接任务也转化为各个子集合之间的连接.对划分原则的一个最基本的要求是要保证所有的连接结果都能被找到.下面对划分方法的正确性进行说明.

定理 1. A 中可能与 D_i 中元素具有祖先-后代关系的元素一定存在于 A_i 中.

证明:假设对于 D 中的一个节点 $d(s_d,e_d)$,A 中的节点 $a(s_a,e_a)$是 d 的祖先节点,则 a 和 d 的编码满足条件 $s_a<s_d<e_d<e_a$,即 a 所覆盖的区域包含了 d 所覆盖的区域.如果 d 被划分到子集合 D_i 中,则说明(s_d,e_d)与(k_{i-1},k_i)相交,设它们的相交区域是(s_r,e_r),$s_d\le s_r<e_r\le e_d$,$k_{i-1}\le s_r<e_r\le k_i$.根据 a 和 d 的关系可以推出 $s_a<s_r<e_r<e_a$,所以 $a(s_a,e_a)$和(k_{i-1},k_i)相交,a 被划分到子集合 A_i 中. □

既然 A 中所有与 D_i 中元素有祖先-后代关系的元素都存在于 A_i 中,则 $A_i \rhd D_i$ 必然可以产生出所有包含 D_i 中元素的连接结果.又如前面所说明的,D 中的每个元素至少出现在 1 个子集合中,所以,所有包含 D 中元素的连接结果必然出现在 $\bigcup_{i=1}^{m} A_i \rhd D_i$ 中,即所有的连接结果都能够被发现.由此我们可以确保该划分方法的正确性.

例 1:图 2 描述了两个输入集合 A 和 D 中元素的编码情况,元素的编码范围划分为如图中虚线所示的 4 个子区间.按照基本的划分方法,A 和 D 被划分为表 1 中的 4 个子集合.由于元素 a_1 和所有的子区域相交,所以它被

分配到了 4 个子集合 A_1~A_4 中.此外,a_3 出现在两个子集合 A_2 和 A_3 中,d_3 出现在 D_2 和 D_3 中.对子集合进行连接的结果是 $A_1 \triangleright D_1 = \{(a_1,d_1),(a_2,d_1)\}$, $A_2 \triangleright D_2 = \{(a_1,d_2),(a_1,d_3),(a_3,d_2),(a_3,d_3)\}$, $A_3 \triangleright D_3 = \{(a_1,d_3), (a_3,d_3)\}$ 及 $A_4 \triangleright D_4 = \{\}$.各个子集合对所产生的连接结果有重复,例如 (a_1,d_3) 和 (a_3,d_3) 重复出现在了 $A_2 \triangleright D_2$ 和 $A_3 \triangleright D_3$ 的连接结果中.通过消除重复和合并可以得到最后的结果 $A \triangleright D = \{(a_1,d_1),(a_2,d_1),(a_1,d_2),(a_3,d_2),(a_1,d_3),(a_3,d_3)\}$.

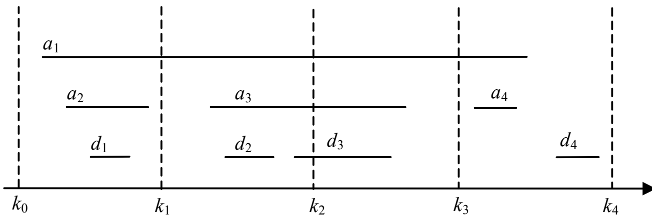

Fig.2 An example of structural join
图 2 结构连接例子

Table 1 Basic partitioning method
表 1 基本划分方法

Partition	A_i	D_i
1	{a1,a2}	{d1}
2	{a1,a3}	{d2,d3}
3	{a1,a3}	{d3}
4	{a1,a4}	{d4}

2.3 改进的划分方法

由第 2.2 节的划分方法所得到的子集合可能相互覆盖,由此导致了两方面的问题.首先,划分产生的子集合大小之和可能大于原集合的规模,中间结果的数据规模增大会导致更多的 I/O 和 CPU 代价.此外,子集合相互覆盖可能产生重复的连接结果.为了去除连接结果中的重复,需要额外的操作,增大了计算代价.

针对基本划分方法所存在的问题,在不破坏正确性的前提下,我们对划分方法进行了改进.从上一节中的例子可以看出,如果 D 中的任意一个元素 d 被分配到某个子集合中,则其所有的祖先元素节点必然也被分配到对应的子集合中.如果 d 存在于多个子集合中,包含 d 的连接结果必然出现多次.基于这样的观察,我们修改了对 D 的划分方法,新的划分方法是,对 D 集合中的任意一个元素 d,如果 R_i 是 d 的开始区间,则 d 被划分到 D_i 中,即 d 被划分到其开始区间所对应的子集合中.采用新的划分方法避免了 D 集合的数据复制,降低了数据复制的程度.更重要的是,由于每个后代元素只出现在一个子集合中,子集合连接不会产生重复的结果,从而避免了去除重复的运算.

例 1 中的数据采用新的划分方法所得到的子集合见表 2.D 的划分情况发生了变化,各个 D_i 之间互不相交.各子集合的连接结果为 $A_1 \triangleright D_1 = \{(a_1,d_1),(a_2,d_1)\}$, $A_2 \triangleright D_2 = \{(a_1,d_2),(a_1,d_3),(a_3,d_2),(a_3,d_3)\}$, $A_3 \triangleright D_3 = \{\}$ 以及 $A_4 \triangleright D_4 = \{\}$.可以看到,各个子集合连接所产生的结果之间不再有重复,可以直接合并产生最后的结果.

Table 2 Improved partitioning method
表 2 改进的划分方法

Partition	A_i	D_i
1	{a1,a2}	{d1}
2	{a1,a3}	{d2,d3}
3	{a1,a3}	{}
4	{a1,a4}	{d4}

3 区域划分结构连接算法

3.1 结构连接算法

基于前面所描述的划分方法,我们设计了具体的区域划分结构连接算法 RangePartitioningJoin.该算法基于 XML 的区域编码,考虑输入数据集的大小和可用内存的大小,选择合适的划分子集合个数,以达到较优的 I/O 效率.设输入元素集合 A 和 D 中的元素个数为$|A|$和$|D|$,所占据的页面个数为 $\|A\|$ 和 $\|D\|$.当 A 和 D 中的任意一个能够完全装入内存时,对 A 和 D 进行连接的 I/O 代价是 $\|A\|+\|D\|$.选择划分的个数的原则是,要尽量使划分得到的子集合能够完全装入内存,从而达到较优的 I/O.假设可用内存的大小为 M,所选择的划分个数应当大于等于 $\left\lceil \frac{\min(\|A\|,\|D\|)}{M} \right\rceil$,这样产生的较小集合的子集合会有较大的可能小于可用内存的大小.

算法 RangePartitioningJoin 的具体步骤如图 3 所示.首先根据集合 A 和 D 的大小以及可用内存的大小来确

定子集合的个数 m,再依据第 2.3 节描述的划分原则将 A 和 D 分别划分为 m 个子集合 A_i 和 D_i,$1\le i\le m$.对所得到的每一对子集合 A_i 和 D_i,如果有任意一个可以完全装入内存,则可以对其调用内存的结构连接方法;如果两个都大于可用内存,则需要对它们进行进一步的划分,直到有一个可以装入内存中为止.

Algorithm RangePartitioningJoin(R,A,D,M)

Input:

R: encoding region, A: ancestor set, D: descendant set, M: available memory size

Description:

```
1. m = ⌈min(||A||,||D||) / M⌉;
2. partition A and D into m subsets according to the rule;
3. for   each A_i 和 D_i   do
4.       if   ||A_i||>M and ||D_i||>M   then
5.            RangePartitioningJoin(R_i,A_i,D_i,M);
6.       else if   ||A_i||!=0 and ||D_i||!=0
7.            IndexMemoryJoin(A_i,D_i,M);
8.       end if
9. end for
```

Fig.3 The RangePartitioningJoin algorithm

图 3 RangePartitioningJoin 算法

当 A 和 D 中的数据节点并不是均匀地分布在源数据树中的时候,即数据节点的分布相对于编码区域发生偏斜,此时所得到的子集合有的可能会大于可用的内存,有的可能很小,有的甚至可能为空.对于这种数据偏斜的情况,算法采用了如下的处理策略:(1) 对于 A_i 和 D_i 都大于可用内存的情况,递归调用 RangePartitioningJoin 进行进一步的划分;(2) 对于 A_i 和 D_i 中任意一个为空的情况,因为其产生的结果集合必然为空,所以该子集合对可以跳过,不必进一步处理.

3.2 内存中的结构连接算法

经过划分处理,原来的结构连接问题转化为多个子集合对之间的结构连接,每对子集合中有一个可以完全装入内存中.对于这种 A_i 和 D_i 中任意一个可以完全装入内存的情况,进行结构连接的 I/O 代价是$\|A_i\|+\|D_i\|$,这时影响算法效率的主要因素是所选择的内存结构连接算法的 CPU 代价.在各种内存结构连接算法中,最基本的是嵌套循环算法.但是正如我们已知的,嵌套循环连接算法的 CPU 代价是相当高的,为 $O(|A_i|\cdot|D_i|)$,并不可取.

在结构连接中,两个输入集合 A 和 D 的地位是不同的,对于 A_i 能装入内存和 D_i 能装入内存两种情况需要不同的考虑.我们设计了内存算法 IndexMemoryJoin,它根据子集合对的具体情况,分别进行不同的处理:

(1) 当 D_i 可以完全装入内存时,对 A_i 中的每个元素 $a(start,end)$,其后代节点是 D_i 中所有满足条件 $a.start<d.start<a.end$ 的元素 d.根据这个特性,我们首先对 D_i 中的元素根据其编码的 *start* 进行排序,构建内存中的索引结构,然后顺序扫描 A_i,对 A_i 中每个元素 a,查找 D_i 上的索引,得到满足条件的所有元素节点,输出连接结果.这是一种索引嵌套循环算法,其 CPU 代价是 $O(|A_i|\cdot\log|D_i|)$.

(2) 当 A_i 可以完全装入内存时,对 D_i 中的每个元素 $d(start,end)$,其祖先节点是 A_i 中所有满足条件 $a.start<d.start<a.end$ 的元素 a,而类似于 B+树的索引结构不能支持这种查询.为了支持基于 A_i 的索引嵌套循环算法,需要建立特殊的查找结构.然后顺序扫描 D_i,对每个元素 d,查找所有的祖先节点,产生连接结果.

在实验部分,我们给出了 IndexMemoryJoin 算法与基本的嵌套循环连结算法的性能比较,可以反映出它们之间的性能差异.

3.3 算法的代价分析

目前已经提出的基于排序合并的结构连接算法或者要求数据有序,或者要求在输入集合上存在对编码的索引.当这些先决条件不满足时,需要对数据进行排序或建立索引的预操作.当数据集大于内存大小时,对两个数据集进行排序所导致的 I/O 代价为 $\|A\|\cdot 2\log_M\|A\|+\|D\|\cdot 2\log b_M\|D\|$.如果对数据集建立索引,所需的 I/O 代

价会更高,因为装载数据前需要对数据进行排序.当数据有序时,该类算法实际连接时的 I/O 代价一般为 $\|A\|+\|D\|$,只需对数据集进行一遍扫描.

我们的算法所需要的I/O代价包括对数据集合的划分和连接时读入的代价.不考虑数据复制和数据偏斜的情况,对数据集进行划分的 I/O 代价包括将数据集读入一遍和将划分结果写出一遍的代价,其大小为 $2\cdot(\|A\|+\|D\|)$.在对子集合进行连接运算时需要将整个数据集合读入内存,其 I/O 代价是 $\|A\|+\|D\|$,所以整个算法的 I/O 代价是 $3\cdot(\|A\|+\|D\|)$.由于我们的划分方法不能防止数据复制的出现,所以实际的 I/O 代价大于 $3\cdot(\|A\|+\|D\|)$.假设数据复制的因子为 $\alpha=\frac{\sum_{i=1}^{m}\|A_i\|}{\|A\|}$,则 A 集合划分的 I/O 代价为 $\|A\|+\alpha\|A\|$,连接时 A 集合读入的 I/O 代价为 $\alpha\|A\|$.整个算法的 I/O 代价为 $(1+2\alpha)\cdot\|A\|+3\cdot\|D\|$.假设 $\|S\|$ 为 $\|A\|$ 和 $\|D\|$ 中较大的一个,可以推出当 $2\log_M\|S\|+1>2+\alpha$ 时,即 $\|S\|> M^{\frac{1+\alpha}{2}}$ 时,我们的算法具有较好的 I/O 效率.一般情况下,α 只是略大于 1,所以只要 A 和 D 中任意一个大于可用内存的大小,RangePartitioningJoin 算法的 I/O 代价就会小于排序合并算法.

4 实验结果和分析

为了评价算法的性能,我们进行了大量的实验.本节描述实验的结果,并对其进行分析.我们选择了STD(Stack-Tree-Desc)算法作为排序合并类算法的代表来与本文提出的 RPJ(RangePartitioningJoin)算法进行比较,因为它是目前提出的排序合并算法中性能较好的.我们在人工生成的数据集和真实的数据集上进行了实验,实验的输入集合 A 和 D 中的元素都是无序的,而且没有索引存在.基于排序合并的连接算法需要临时对数据集进行排序,排序的时间也被计算在算法的执行时间之中.

4.1 实验设置

我们在 native XML 数据库系统 Orient-X 的基础上实现了算法,选择了它的存储和缓冲子系统来构建实验系统.存储子系统被修改为直接对磁盘上进行 I/O 操作,以消除操作系统文件缓冲的影响.所有的算法都用 C++编程语言来实现,所有的实验都在一台 Pentium IV 1.40GHz,256M RAM,60G 硬盘的 PC 上运行,底层操作系统是Windows XP.实验结果都在固定的缓冲区大小(100 个页面)下获得.

我们的实验系统对实验数据需要进行预处理和装入.它通过调用一个基于事件的 XML parser 来分析 XML 文档,实现对元素节点的编码,同时将带编码的元素节点装入到对应的元素集合中,每个元素节点集合包含具有相同 Tag 的元素节点记录,是连接算法的输入数据.

为了评价算法的性能,我们选择执行时间为评价指标.STD 算法的执行时间包括对数据排序的时间和合并连接的时间,RPJ 算法的执行时间包括划分的时间和内存连接的时间.

4.2 人工生成的数据集

我们选用人工生成的数据集来进行实验,以便能够方便地控制 XML 文档的结构和特性.我们采用 IBM XML Generator[13]生成实验文档,所用的 DTD 如图 4 所示.

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

Fig.4 The DTD of synthetic data set

图 4 人工数据集的 DTD

4.2.1 整体性能

为了对两种算法的整体性能进行评价,我们选择大小为 106M 的文档作为实验数据,其中各个元素的个数

在表 3 中给出.表 4 给出了实验用的 8 个查询例子.因为父子关系是祖先-后代关系的特例,所以只选择了祖先-后代查询进行实验.实验结果如图 5 所示.从图中可以看出,对给出的 8 个查询例子,RPJ 方法都优于 STD 方法,具有更好的整体性能.

Table 3 Description of synthetic data set
表 3 人工数据集的描述

Element	Number
manager	216
department	270 574
employee	511 725
name	1 048 951
email	63 608

Table 4 Description of queries of synthetic data set
表 4 人工数据集的查询描述

Query	Path expression	Result cardinality
QD1	manager//department	409 038
QD2	manager//employee	772 529
QD3	manager//email	95 492
QD4	department//employee	3 446 609
QD5	department//name	6 784 805
QD6	department//email	362 209
QD7	employee//name	778 161
QD8	employee//email	33 359

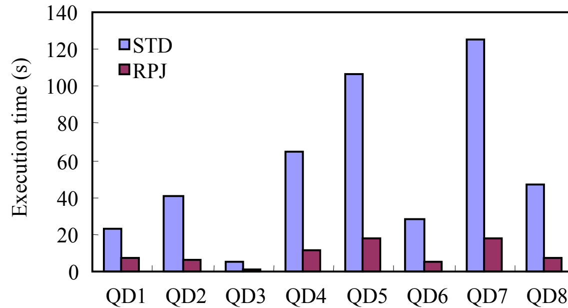

Fig.5 The experimental result of synthetic data set
图 5 人工数据集的实验结果

4.2.2 可扩展性实验

可扩展性实验仍然采用上面给出的 DTD,通过调整参数分别生成了大小为 20M,40M,60M,80M 和 100M 的文档.从上一节的 8 个查询中,我们选择了 QD4 和 QD7 两个查询.实验结果如图 6 所示,STD 算法和 RPJ 算法的执行时间都随着数据规模的增大而呈线性增长,而 RPJ 算法的增长趋势比 STD 平缓很多.由此可以看出,RPJ 算法有很好的可扩展性,适用于大规模数据集的结构连接运算.

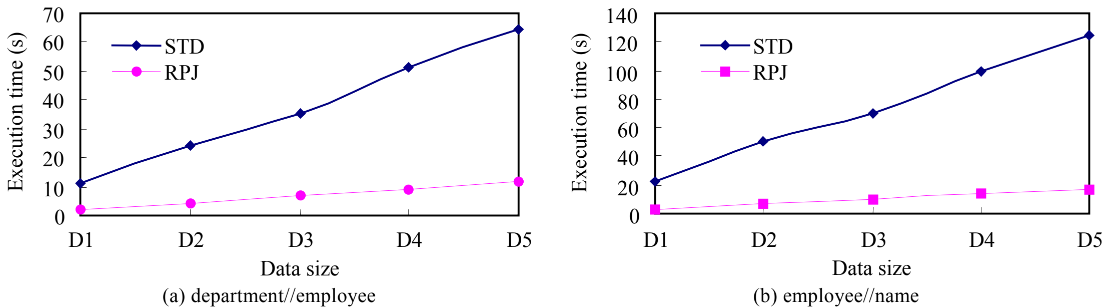

Fig.6 The result of scalability test
图 6 可扩展性实验结果

4.2.3 内存结构连接算法的性能

为了说明内存结构连接算法的性能问题,我们进行了实验,所选择的两个算法分别是嵌套循环算法(nested loop join,简称为 NLJ)和 IMJ(IndexMemoryJoin).我们生成了较小的数据集(见表 5),以保证在我们的缓冲设置下该数据集中的所有元素集合都能被装入内存中,这样查询的执行时间就完全反映了内存连接算法的执行时间,而不涉及集合的划分.实验结果在表 6 中列出.可以看到,对 QD1 查询,NLJ 算法快于 IMJ 算法,而对 QD2 和 QD3 两个算法的执行时间相同.之所以出现这种情况,是因为这 3 个查询的祖先集合 manager 只有 3 个元素,所以 NLJ 和 IMJ 没有太大的性能区别,甚至由于构建索引的代价,IMJ 执行时间可能长于 NLJ.除了这 3 个查询以外,对于

QD4 到 QD8,IMJ 算法都远远快于 NLJ 算法.实验结果说明,不同内存结构连接算法的性能差别是非常大的,除了极端情况以外,IMJ 算法的性能远远优于 NLJ 算法.

Table 5 Description of data set
表 5 数据集描述

Element	Number
manager	3
department	3 053
employee	5 551
name	11 606
email	1 075

Table 6 The experimental result of memory algorithms
表 6 内存算法的实验结果

Query	Result cardinality	NLJ (s)	IMJ (s)
QD1	3 725	0.020	0.035
QD2	6 790	0.015	0.015
QD3	1 330	0.005	0.005
QD4	32 171	9.604	0.036
QD5	63 987	19.789	0.060
QD6	5 000	1.838	0.020
QD7	8 550	35.882	0.045
QD8	585	3.320	0.030

4.3 真实的数据集

我们也在真实的数据集上进行了实验,所选择的数据集是 DBLP[14]和 XMark[15].DBLP 是一个关于参考文献的数据集,我们采用的原始文档的规模为 116M.XMark 来源于一个针对 XML 的基准测试项目,我们选择参数为 1 生成了大小约为 115M 的文档.对两个数据集我们分别选择了 8 个查询例子,这些查询的情况分别在表 7 和表 8 中给出.XMark 测试集中输入数据集 *A* 和 *D* 的规模比较接近,差别不是很大.DBLP 测试集中各个查询的输入集合普遍规模比较大,此外还有 *A* 和 *D* 规模都很小的情况,如 QB7,以及 *A* 和 *D* 规模相差较大的情况,如 QB8.

Table 7 Description of queries of Xmark
表 7 XMark 的查询描述

Query	Path expression	|*A*|	|*D*|
QX1	item//keyword	21 750	69 969
QX2	item//quantity	21 750	43 500
QX3	item//incategory	21 750	82 151
QX4	open_auction//description	12 000	44 500
QX5	open_auction//bidder	12 000	59 486
QX6	person//interest	25 500	37 689
QX7	person//name	25 500	48 250
QX8	description//text	44 500	105 114

Table 8 Description of queries of DBLP
表 8 DBLP 的查询描述

Query	Path expression	|*A*|	|*D*|
QB1	article//title	105 754	294 470
QB2	article//author	105 754	635 154
QB3	article//cite	105 754	171 071
QB4	inproceedings//crossref	184 465	85 186
QB5	inproceedings//author	184 465	635 154
QB6	inproceedings//pages	184 465	286 061
QB7	proceedings//editor	2 326	4 969
QB8	book//author	801	184 465

实验结果如图 7 所示,RPJ 算法在两个数据集上的表现都优于 STD.从实验可以看出,RPJ 对于各种不同的数据情况都表现出了良好的性能,不会因为输入数据集合的情况不同而产生性能上的巨大变化.

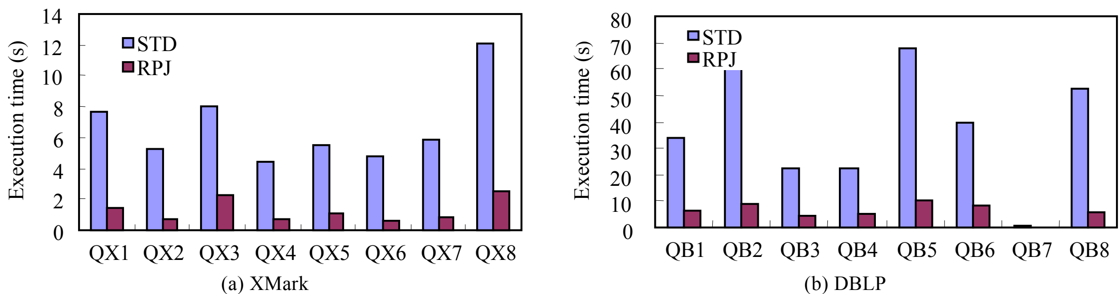

Fig.7 The experimental result of real data set
图 7 真实数据集的实验结果

5 结 语

近年来,XML 数据的管理和查询问题吸引了众多学者的关注,成为数据库领域的研究热点.本文针对 XML 路径表达式计算中的核心操作——结构连接的有效处理问题进行了研究.在区域编码方法基础上,我们提出了一种基于区域划分的连接算法.该算法不要求数据有序或索引的存在,基于编码的覆盖区间对元素节点进行划分,针对磁盘 I/O 进行了优化.实验结果显示,该算法在输入数据无序的情况下优于基于排序合并的结构连接算法,而且在各种数据分布情况下具有稳定的表现以及良好的可扩展性.

结构连接作为XML路径表达式处理中的关键问题,其高效的解决对整个查询处理有重大的影响.多种不同的连接算法可以为查询优化提供多种选择,形成不同的查询计划.我们目前正在 XML 数据管理原型系统 Orient-X 的基础上进行查询处理和优化的进一步研究工作.

References:

[1] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, eds. Extensible markup language (XML) 1.0 (second edition). W3C Recommendation 6, 2000. http://www.w3.org/TR/2000/REC-xml-20001006

[2] Shanmugasundaram J, Tufte K, He G, Zhang C, DeWitt D, Naughton J. Relational databases for querying XML documents: Limitations and opportunities. 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. 302~314.

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

[4] Clark J, DeRose S, eds. XML path language (XPath) Version 1.0. W3C Recommendation 16, 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

[5] Chamberlin D, Clark J, Florescu D, Robie J, Simeon J, Stefanescu M. XQuery: A query language for XML. W3C Working Draft 07, 2001. http://www.w3.org/TR/2001/WD-xquery-20010607

[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] Chien SY, Vagena Z, Zhang DH, Tsotras VJ, Zaniolo C. Efficient structural joins on indexed XML documents. In: Bernstein PA, *et al*., eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2002. 263~274.

[10] Jiang HF, Lu HJ, Wang W, Ooi BC. XR-Tree: Indexing XML data for efficient structural joins. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2003. 253~264.

[11] Wang W, Jiang HF, Lu HJ, Jeffrey XY. PBiTree coding and efficient processing of containment joins. 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.

[12] Lee YK, Yoo SJ, Yoon K. Index structure for structured documents. In: Edward AF, Gary M, eds. Proc. of the 1st ACM Int'l Conf. on Digital Libraries. New York: ACM Press, 1996. 91~99.

[13] IBM Corporation. XML data generator. http://www.alphaworks.ibm.com/tech/xmlgenerator

[14] DBLP. http://dblp.uni-trier.de/xml/

[15] Schmidt AR, Waas F, Kersten ML, Florescu D, Manolescu I, Carey MJ, Busse R. The XML benchmark project. Technical Report, INS-R0103, Amsterdam: CWI, 2001.