一种基于结构索引的 XML 模式匹配方法

乔 健 陈彤兵 汪 卫 施伯乐
(复旦大学计算机与信息技术系 上海 200433)

摘 要 XML 文档采用了树型的数据模型,对其查询通常是用带有选择谓词的模式树在 XML 数据中进行匹配。因此,找出 XML 文档中所有符合模式树结构的元素集,是 XML 查询处理的核心操作。本文提出了结构索引 JoinGuide,并在此基础上提出了一种新的 XML 模式匹配方法。它使用 JoinGuide 来对模式树进行预匹配,这样在 XML 文档上查询时可以利用索引上的匹配结果来忽略部分连接谓词和不必要的候选 XML 元素序列。本文还提出了三种具体算法来利用索引匹配结果进行进一步的查询。实验结果表明本文中的模式树匹配方法优于以往的匹配方法,并且索引所需的空间很小。

关键词 XML,模式树匹配,结构索引,JoinGuide

A Structural Index Based XML Pattern Matching Approach

QIAO Jian CHEN Tong-Bing WANG Wei SHI Bai-Le
(Department of Computing and Information Technology, Fudan University, Shanghai 200433)

Abstract XML document employs a tree-structured data model and its queries typically specify patterns of selection predicates to match XML data. So finding all occurrences of such a pattern in a XML document is the core operation of XMLquery processing. In this paper, a kind of structural index called JoinGuide is introduced and then a novel XML pattern matching approach based on it is presented. The approach utilizes JoinGuide to pre-match a pattern tree, then when querying XML documents, the pre-match result is used to avoid some join predicates and input XML elements list. Also three algorithms are presented to query XML documents using pre-match result. Experiments show that the approach outperforms the previous ones, and the index size is small.

Keywords XML, Pattern matching, Structural index, JoinGuide

1 引言

随着 XML 的广泛应用,如何高效地查询 XML 数据变得越来越重要。常用的 XML 查询语言有 XPath 和 XQuery 等。由于 XML 的树模型,它们也常被表示成树型结构的模式[8],因此找出模式树在 XML 文档中的所有匹配是查询处理的关键操作。

文[5,10,9,3]中提出了对 XML 数据建立结构索引的方法,可以较快地处理一般路径查询。但是,这类方法无法判断返回结点的子树条件,也无法返回具有结构关系的结点对。文[13,1,2,7]中提出将 XML 文档编码后使用结构化连接的方法进行查询,这类方法可以找出模式树在文档中的匹配,查询能力大于结构索引方法,但是没有利用结构索引,要进行大量连接操作,影响了查询效率。

本文结合这两类查询方法的优点,提出了一种新的 XML 模式树匹配方法,使用结构索引减少不必要的连接且查询能力等价于结构化连接方法。本文提出结构摘要索引 JoinGuide,在进行模式树匹配时首先在 JoinGuide 上进行预匹配,根据预匹配结果可以去掉原模式树中的一部分谓词条件,减少候选 XML 元素序列。本文还提出了三种具体算法来利用预匹配结果查询原 XML 数据。

2 相关工作

以前的工作已经对模式树匹配问题做了大量研究。文[1]中通过分解模式树为二元结构关系,分别连接,再把二元结果组合起来,处理模式树匹配,但是连接代价较大且会产生大量无用的中间结果。文[2]中的 TwigStack 算法可以将整棵模式树一起匹配处理,避免保存无用的中间结果,它对于只有祖先后代边的模式树匹配是最优的。但是,它没有使用索引信息,必须对模式树中所有的边都进行连接,而且它对于带有父子边的模式树不能保证最优。文[7]在 TwigStack 算法的基础上使用了 XR-Tree 索引,可以快速跳过输入序列中的无用结点,但这种索引不能减少连接次数。BLAS[4]中提出的 P-label 可以直接找出满足连续父子边路径的 XML 结点,从而避免了一部分父子边的连接,并减少须处理的结点数。它的缺点是没有利用更多的结构信息,而且 P-label 穷举了 XML 文档上所有可能路径,导致所需空间过大。

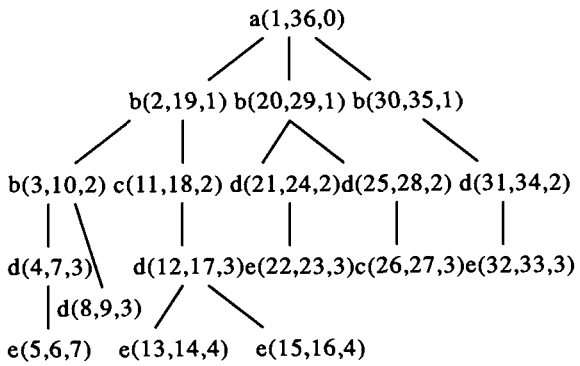

图 1 一棵 XML 文档树

对于 XML 文档建立结构索引也有很多相关研究,DataGuides[5]、1-index[10]、A(k)-index[9]、D(k)-index[3]中提出的结构索引分别建立了对 XML 文档的结构摘要。这样,在查询一般路径表达式时,只须在索引上进行查询,无须访问原 XML 数据。但是,这种索引方法查询能力较弱,只能处理一般路径查询,无法判断结点的子树条件,也无法找出结点集之

间的结构关系，不能处理模式树匹配。

3 基本概念

3.1 数据模型和模式匹配

XML 数据使用树型数据模型。图 1 为一个 XML 文档的树表示。对 XML 的查询一般表示为模式树，如 XPath 查询 a//b[//c]/d/e 可以表示为图 2 所示的模式树，返回结点为 e。本文中在模式树结点后加'*'，表示该结点需要返回。查询处理就是找出模式树在 XML 数据中的匹配，按下面形式定义匹配：

给定查询模式 Q 和 XML 数据 D，一个 Q 在 D 中的匹配表示为一个从 Q 中查询结点到 D 中 XML 结点的映射，使①查询结点上的谓词被对应的数据结点满足，②查询结点间的结构关系（如父子关系、祖先后代关系）被对应的数据结点间满足。

例如，图 2 的模式树在图 1 的 XML 文档中的匹配为(a:1,b:20,c:26,d:21,e:22)，返回的结果为(e:22)。

3.2 编码方式

为了能够在常数时间判定两个结点之间是否存在祖先-后代关系，常常将树中 XML 结点编码为〈StartPos, EndPos, LevelNum〉[13,1,2]。图 1 中对 XML 树进行了编码。本文使用 StartPos 来唯一标示一个 XML 结点。结构关系判断方法为：如果结点 a,b 编码是〈S_1,E_1,L_1〉,〈S_2,E_2,L_2〉，那么

1) b 是 a 的后代，当且仅当 $S_1 < S_2 < E_1$。

2) b 是 a 的孩子，当且仅当 $S_1 < S_2 < E_1$ 且 $L_2 = L_1 + 1$。

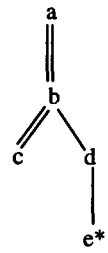

图 2 模式树

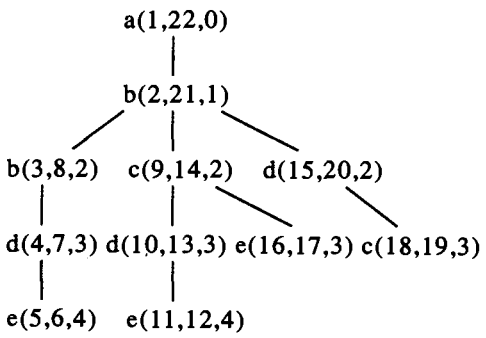

图 3 图 1 对应的 JoinGuide 索引

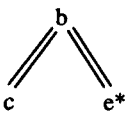

图 4 简化后的图 2 模式树

4 JoinGuide 索引

JoinGuide 索引是原 XML 文档的树型结构摘要，它可以用来简化查询模式树，以及选择输入的 XML 结点，从而提高 XML 上模式树匹配的效率。

一个 XML 结点的路径为从根开始到该结点所经过结点的标签序列 $l_1 l_2 \cdots l_k$。将具有相同路径的 XML 结点归入一个等价类。一个索引结点对应一个等价类，且该索引结点在 JoinGuide 树中具有与该等价类相同的路径。图 3 为图 1 的 JoinGuide 索引。

JoinGuide 对索引结点和 XML 结点都使用 3.2 节中的编码方式进行编码，然后用一个倒排表记录同标签的所有索引结点，每个索引结点有一个 extentSet 记录对应等价类中所有的 XML 结点。其中索引结点和 XML 结点都按照 StartPos 排序。

JoinGuide 索引使 XML 数据树中出现的每种路径在 JoinGuide 中都会唯一出现一次，并且对应唯一一个索引结点，JoinGuide 中出现的路径都会出现在 XML 文档中。由于索引结点与路径一一对应，因此一个路径为 $l_1 l_2 \cdots l_k$ 的 XML 结点 a 如果属于索引结点 A，那么 a 的父结点路径为 $l_1 l_2 \cdots l_{k-1}$，属于索引树中 A 的父结点。利用这个性质，我们可以对查询模式树进行化简。

表 1 图 2 的模式树在图 3 的 JoinGuide 上的匹配结果

a	b	c	d	e
1	2	9	15	16
1	2	18	15	16

在处理模式树查询时，首先在 JoinGuide 上进行预匹配，如果预匹配结果中有索引结点对(A,B)满足查询中的二元结构关系，那么对 B 的 extentSet 中的每个 XML 结点 b，必然存在一个 XML 结点 a 属于 A 的 extentSet，并且(a,b)满足该二元结构关系。因此，在对 XML 文档的查询模式树中，只要保留原模式树中的叶子结点，就可以保证最终结果中从根到该叶子的路径条件被满足。另外还需要保留分叉结点来保证结果中该结点的子树条件被满足。这样，在 XML 文档上进行模式树匹配时，可以省略掉一些查询谓词。新的模式树上只需要保留返回结点、叶子结点和分叉结点。例如，对于图 2 的模式树，在 XML 文档上需要查询模式树如图 4 所示。

图 2 的模式树在图 3 的 JoinGuide 索引上的预匹配结果如表 1 所示。在 XML 文档上查询时，不需要对模式树处理满足结点谓词的所有 XML 结点，只需要处理预匹配结果中出现的索引结点的 extentSet 中的 XML 结点。这样就可以忽略大量无用的 XML 结点。

JoinGuide 省略模式树谓词和选择输入序列的能力要强于 BLAS[4] 中的 P-label。例如，对于图 2 的模式树，P-label 只能去掉模式树中的结点 d，在选择输入序列方面它会比 JoinGuide 多选出图 3 中 b(3)和 d(5)的 extentSet，而这部分结点没有匹配结果。而且，JoinGuide 不像 P-label 一样穷举 XML 文档中的所有路径，它只编码可能的路径，所以需要的空间远小于 P-label。本文 6.3 节的实验证实了这一点。

5 查询处理

当进行模式树查询时，首先在 JoinGuide 索引上进行预匹配，再根据预匹配结果对 XML 数据进行化简后的模式树

匹配得到最终结果。文[2]中的 TwigStack 算法，可以将编码后的有序结点序列作为输入，利用栈一次扫描连接出整树匹配结果，对于只有祖先后代边的模式树达到最优。本文使用 TwigStack 算法对索引进行预匹配。

本节提出三种使用 JoinGuide 索引的预匹配结果来对 XML 文档进行模式树匹配的算法，分别为 merge-match、match-union 和 multi-merge join。前两种算法各有侧重，而第三种算法结合了两者的优点，在不同情况都不比前两种算法差。

5.1 merge-match 算法

merge-match 算法只利用预匹配结果来选择输入 XML 结点序列，然后使用文[2]中的 TwigStack 算法连接出匹配结果。对于一个模式树结点，它将预匹配结果中出现的索引结点的所有 extentSet 利用多路归并排序得到输入序列。例如，对于表 1 中的预匹配结果，它首先对每列进行处理，将一列中的所有不同索引结点对应的 extentSet 作为下一步匹配的输入结点序列，对图 4 的模式树的 b 结点载入索引结点 2 的 extentSet，对 c 结点载入索引结点 9，18 的 extentSet 并排序。

这种算法的优点是继承了 TwigStack 算法对于输入结点序列的一次扫描，对于只有祖先后代边的模式树匹配保证最优，但是它仍然有 TwigStack 算法对于父子边的次优性。它的一个缺点是没有使用索引匹配结果中蕴涵的结构信息，这样会对两段没有结构关系的 XML 序列进行连接；另一个缺点是在选择输入结点序列时需要进行归并排序。

5.2 match-union 算法

match-union 算法利用索引匹配结果中的结构信息，对索引匹配结果集(如表 1)的每一行，载入 XML 结点序列，然后使用 TwigStack 算法来计算匹配结果。在对每一个索引结果对应的输入序列分别匹配后，将所有匹配结果 union 在一起。例如对于表 1 中的预匹配结果，它分别根据索引结点(b:2,c:9,e:16)和(b:2,c:18,e:16)的 extentSet 载入输入 XML 序列进行查询，之后将两个结果集合并。

这种方法充分利用了索引匹配结果中的结构信息，对于不相关的两段 XML 序列不会进行连接。但是它有个很大的缺点是可能存在多遍扫描和处理。例如上例中它对索引结点 2 和 16 的 extentSet 都会多次处理。

5.3 multi-merge join 算法

考虑到 merge-match 和 match-union 算法的优缺点，本文提出了 multi-merge join 算法，它结合了两者的优点。它首先把索引预匹配结果组织起来，使每个索引结点的 extentSet 只出现一次，而又在数据流中保存相互之间的结构关系。这样，每个 XML 元素只须处理一遍，不须预先进行排序，而且充分利用了索引匹配结果中的结构信息，不会对两段不可能满足结构关系的 XML 序列进行连接。

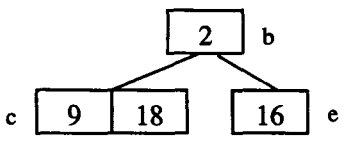

图 5　数据流之间的树关系

该算法将索引匹配结果组织成树的形式，例如对于模式树中的一个二元结构关系(anc，des)，假设 anc 是 des 的父结点或祖先结点，那么对于 anc 中的一个输入索引结点 i_a，des 中可能有一个或多个输入索引结点在预匹配结果中为 i_a 的后代。那么建立一棵子树，根为 i_a，des 中与 i_a 对应的索引结点作为 i_a 的孩子，这些索引结点被称为 i_a 的一簇孩子。

索引结点 i_a 有 k 簇孩子，其中 k 为模式树中结点 anc 的孩子数。例如对于图 4 的模式树结点 b，k=2。表 1 的匹配结果可以组织成如图 5 所示，其中索引结点 9 和 18 是索引结点 2 的一簇孩子，属于模式树结点 c；索引结点 16 是结点 2 的另一簇孩子，属于模式树结点 e。这样，就将索引匹配结果组织成了树型。

在查询时，一个输入索引结点作为一个 XML 数据流。一簇 XML 数据流被组织成一个最小堆，关键字为流中下一元素的 StartPos，该簇数据流中下一元素 StartPos 最小的一个称为当前数据流。查询时通过最小堆，可以将一簇数据流看作一个数据流，然后自底向上地使模式树结点的当前数据流满足子树条件，直到根的数据流满足子树条件时，对所有子树输出结果，然后移动根的当前数据流，重复直到根的数据流结束。

这里先定义一些数据结构和操作。

XML 结点序列，保存一个索引结点的 extentSet 中的 XML 结点，按照 StartPos 排序，每个结点有一个状态位表示该结点是否已被处理过。如果已被处理过，是否满足子树条件及在最终结果中。用此状态位来保证每个结点的子树只被处理一次。

XML 数据流，指向一个 XML 结点序列，还指向 k 簇子数据流 childstreamcluster，其中 k 为它对应的模式树结点的孩子数。每个 childstreamcluster 中包含若干个 childstream，组织成最小堆，childstreamcluster[0]指向其中 nextS 最小的 childstream，它就是该簇数据流中的当前数据流。另外介绍一些操作：isEnd()判断该数据流是否结束；gotoEnd()使该数据流结束；nextS()和 nextE()返回流中下一元素的 StartPos 和 EndPos；next()返回流中下一元素；adjust()函数调整当前数据流及子数据流使其就位，即满足子树条件；advance()函数使数据流前进；goNext()调用 advance()和 adjust()，找出数据流中下一个满足子树条件的位置。

数据流的 adjust()函数定义见算法 1。

算法 1：数据流的 adjust 函数

```
1 if( isLeaf() ) return
2 while ( ! isEnd() )
3     for each childstreamcluster
4         for each childstream in this childstreamcluster
5             while( childstream→nextS() < nextS() )
6                 childstream→advance()
7             childstream→adjust()
8         make min-heap for this childstreamcluster
9         if (childstreamcluster[0]→isEnd() )
10            gotoEnd()
11            return
12    max = max(childstreamcluster[0]→nextS() )
13    if ( nextE() > max ) return
14    while( nextE() < max) advance()
```

adjust()函数使当前数据流的下一元素及其所有子数据流满足子树条件。第 1 行判断是否对应叶子结点。如果是则直接返回，因为该数据流必然满足子树条件。2～14 行当数据流未结束时反复执行，直到流满足子树条件。3～6 行对于该数据流的每个子数据流簇，调整其中的每个子数据流，使其 nextS()大于当前流的 nextS()，即可能满足包含关系。7 行调用 adjust()调整子数据流，使其就位。8～11 行对该子数据流簇建最小堆，并判断是否该簇中所有子数据流结束。12～14 行找出各个子数据流簇中的最大 nextS，如果当前数据流

的 nextE()大于该值，则说明当前流已满足子树条件，返回，否则当前流前进。重复此过程，直到流结束或者就位。

算法 multi-merge join 如算法 2 所示。

算法 2：算法 multi-merge join

```
1 for each stream of root
2        stream→adjust()
3 make min-heap for streamcluster in root
4 while ( ! streamcluster [0]→isEnd() )
5        outputSubTreeResult(root, streamcluster [0] )
6        streamcluster [0]→goNext()
7        adjust min-heap of streamcluster
```

multi-merge join 算法首先调整模式树根结点中的每个数据流，使其就位，然后对这些数据流建最小堆。当根结点中的数据流未结束时，对每一就位位置输出子树结果。

算法 3：outputSubTreeResult 函数

函数 outputSubTreeResult(root, stream)

```
1 for each childstreamcluster of stream
2   while (! childstreamcluster [0]→isend()
    and childstreamcluster [0]→nextS() < stream→nextE() )
3     outputSubTreeResult(childnode, childstreamcluster [0])
4     output binary structural result ( stream→next(),
                    childstreamcluster [0]→next() )
5     childstreamcluster [0]→goNext()
6     adjust min-heap of childstreamcluster
```

函数 outputSubTreeResult 见算法 3。它输出当前流下一元素所有子树的二元结果，通过递归调用自身，首先输出它所有孩子的子树结果，然后输出父数据流和子数据流的二元匹配结果，直到父数据流当前元素与子数据流不再满足结构关系。

在 multi-merge join 算法后需要将所有二元结果连接成最终结果。

下面举个例子。对于图 5 的数据流树，索引结点 2，9，18，16 的输入 XML 序列分别为流 b1：(2-19，20-29，30-35)，c1：(11-18)，c2：(26-27)，e1：(22-23，32-33)。算法：首先调用 adjust()，使根结点的流 b1 就位，adjust 函数首先使流 c1，c2，e1 的 nextS()大于 2，然后选出 c1 和 c2 中 nextS()最小的为 c1，在 c 和 e 的数据流中选出 nextS()最大的为 e1，发现 b1 的 nextE()小于 e1 的 nextS()，移动流 b1，再移动流 c1，c2，e1，使其 nextS()大于 b1 的 nextS()。此时 c1 结束，然后调用递归，调用输出子树结果(b：20，c：26) (b：20，e：22)，然后 b1 调用 goNext()，移动流 b1 到(30)，移动子流 c2 和 e1，发现 c2 结束。这样，c1 和 c2 都已经结束。调用 gotoEnd()，使 b1 结束。算法结束。最终匹配结果为(b：20，c：26，e：22)，因为只需要返回结点 e，所以最终返回结果为 e：(22，23，3)。

multi-merge join 算法通过对子数据流维护最小堆，避免了 merge-match 算法中开始对于多个结点序列的多路归并，并且对于每个结点只处理一次。它充分利用了索引匹配结果的结构信息，没有输出多余中间结果。本文 6.2 节的实验证实了该算法的效率。

6 实验

我们在两个数据集上测试了本文中的算法，并且和文[2]中的 TwigStack 以及 BLAS[4] 中的基于文件系统的 Push-up 算法进行了比较，实验结果证明了本文中使用 JoinGuide 的算法优于这两者。我们还对不同大小文档的索引规模做了实验，说明了 JoinGuide 索引的大小不会太大。

6.1 实验设置

1) 数据集：实验中使用两个 XML 数据集，一个是 XMark[12]，它是一个人造基准测试数据，具有递归的 DTD，但是结构不太复杂，层数也不深；另一个是对 Nasa DTD[11] 用 IBM Generator[6] 生成的数据，它结构比较复杂，嵌套和递归比较多，层数也很深。数据集的属性如表 2 所示。

2) 查询：实验使用了两组查询，分别对应 XMark 和 Nasa 数据集。每组三个查询从简单到复杂，查询使用 XPath 表示模式树，标签后加'*'的结点表示需要返回。查询及匹配结果数见表 3。

3)机器配置 cpu：CIII 1.2G 。内存：256M sdram，操作系统：Windows2000，编程环境：VC++6.0。

6.2 性能分析

本节比较文[2]中的 TwigStack 算法、BLAS[4] 中的 Push-up 算法和本文中提出的三种算法。这 5 种算法的实现都是基于文件系统的。查询处理时间和读入结点的个数见表 4。

从表 4 中可以看出，本文中算法利用 JoinGuide 索引忽略连接的能力要强于文[4]中的 P-label，所以查询处理时间和读入结点都要少于 BLAS。而 TwigStack 需要对每条边进行连接，效率最低。对于查询 QA1 和 QN1 的单路径查询，JoinGuide 索引可以直接产生出最终结果，无须连接，也不需对 XML 数据进行匹配，所以本文三种算法代价相同。QA2 和 QN2 查询的预匹配结果没有重合，所以 match-union 与 merge-match 算法性能相当。match-union 算法的优势没有体现出来的原因，是装载候选序列的次数过多、过碎，而 multi-merge join 算法避免了这一点，性能最好。从 QA3 和 QN3 查询的读入结点可以看出，match-union 算法在某些情况下对于结点的重复处理很多，这是它的主要缺点。而 multi-merge 没有这个问题，对 XML 结点只处理一遍。在不同情况下，multi-merge join 都是最优的，读入结点数等于 merge-match，处理时间最少。

6.3 索引大小

本节衡量索引大小随文档的变化情况。实验中用不同的参数生成了不同大小的 XMark 文档。参数从 0.05 到 1，间隔 0.05，生成了不同大小的文档，JoinGuide 索引的结点数变化见图 6。由图中可以看出，索引结点数在文档增大时逐渐趋于稳定。

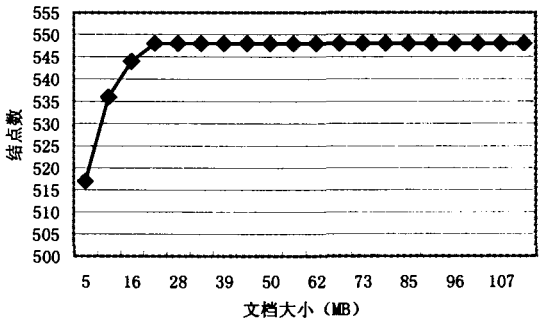

图 6 索引结点与文档大小的关系

表 2 XMark 和 Nasa 数据集的特性

	文档大小	XML 结点数	不同标签数	最深层数	JoinGuide 结点数
XMark	56.2M	1024073	83	12	548
Nasa	31.9M	838539	140	28	12774

表 3 模式树查询(QA 对应 XMark 集,QN 对应 Nasa 集)

	查询模式树(标签后加 * 的结点需要返回)	查询结果数
QA1	site/regions//item/description/text *	7729
QA2	site//item/description/parlist/listitem/text * [bold *][keyword *]/emph *	7450
QA3	site/regions/asia/item * [//mail/text [//bold]//keyword]/description * /text//emph	68
QN1	dataset//para/project//xlink:simple/@role *	5905
QN2	dataset//title//author * [middleName *][lastName *]/affiliation/xlink:simple/@role *	1634
QN3	dataset//title * //footnote * [para/address [homePage *]/street *]//author/lastName *	1210

BLAS[4]的 P-label 对于实验中选用的 XMark 数据集有 84^{12} 种可能的取值,需要 8 个 byte 才能表示一个 XML 结点的 P-label,索引一共需要 8M 的空间。而对于 Nasa 数据集,有 141^{28} 种可能取值,需要 26 个 byte 才能表示,一共需要 21M 空间,接近于原文档的大小。与之相比,JoinGuide 分别只需要 2k 和 51k 的空间。这说明,JoinGuide 需要的空间远小于文[4]中的 P-label,更具有实用性。

表 4 TwigStack, BLAS, merge-match, merge-union, multi-merge 五种算法执行查询的运行时间和读入结点个数

运行时间(ms)/读入结点数	TwigStack	BLAS	merge-match	match-union	multi-merge join
QA1	18500/ 85807	730 / 7730	80 / 7729	80 / 7729	80 / 7729
QA2	4200 / 232043	1340 / 20574	410 / 20573	430 / 20573	340 / 20573
QA3	1320 / 254310	550 / 118543	50 / 4901	210 / 98235	50 / 4901
QN1	1660 / 142714	930 / 58431	90 / 5905	90 / 5905	90 / 5905
QN2	1610 / 216608	900 / 85849	140 / 7846	130 / 7846	100 / 7846
QN3	550 / 61221	380 / 23446	90 / 2548	140 / 10358	40 / 2548

结论 本文对模式树匹配问题提出了一种新的结构索引 JoinGuide,并且提出了使用 JoinGuide 的三种查询匹配算法,实验证实了本文中算法的优越性。

以后,我们打算使用局部相似性来改进 JoinGuide,使它避免对路径信息过于细化,对复杂结构的 XML 也能保持合适的大小,同时能够提高在索引上的查询速度。

参 考 文 献

1 Al-Khalifa S, et al. Structural joins: A primitive for efficient XML query pattern matching. In ICDE, 2002

2 Bruno N, et al. Holistic twig joins: Optimal XML pattern matching. In SIGMOD, 2002

3 Chen Q, et al. D(k)-index: An adaptive structural summary for graph-structured data. In SIGMOD, 2003

4 Chen Y, et al. BLAS: An efficient XPath processing system. In SIGMOD, 2004

5 Goldman R, Widom J. Dataguides: Enabling query formulation and optimization in semistructured databases. In VLDB, 1997

6 IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator

7 Jiang H, Wang W, Lu H. Holistic twig joins on indexed XML documents. In VLDB, 2003

8 Jagadish H V, et al. TAX: A Tree algebra for XML. In DBPL, 2001

9 Kaushik R, et al. Exploiting local similarity for efficient indexing of paths in graphstructured data. In ICDE, 2002

10 Milo T, Suciu D. Index structures for path expressions. In ICDT, 1999

11 NASA XML Group. Available at: http://xml.gsfc.nasa.gov/.

12 XMARK. The XML-benchmark project, April 2001. http://monetdb.ewi.nl/xml/index.html

13 Zhang C, et al. On supporting containment queries in relational database management systems. In SIGMOD, 2001

(上接第 67 页)

7 Kuntz M, Siegle M. A stochastic extension of the logic PDL. In: Sixth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS6), Monticello (IL), 2003. 58～61

8 Ramani S, Trivedi K S, Dasarathy B. Performance analysis of the CORBA Event Service using stochastic reward nets. In: Proc. of the 19th IEEE Symposium on Reliable Distributed Systems, Nurnberg, Germany, 2000. 238～247

9 Ramani S, Trivedi K S, Dasarathy B. Performance analysis of the CORBA notification service. In: Proc. 20th IEEE Symposium on Reliable Distributed Systems, New Orleans, USA, 2001. 227～236

10 Ramani S, Go℃seva-Popstojanova K, Trivedi K S. A framework for performability modeling of messaging services in distributed systems. In: the 8th IEEE Intl. Conf. on Engineering of Complex Computer Systems, Greenbelt, MD, 2002. 229～238

11 Trivedi K S, Ramani S, Fricks R. Recent Advances in Modeling Response-Time Distributions in Real-Time Systems. In: Proc. of the IEEE 91(7), 2003. 1023～1037

12 Lin C. Performance Evaluation of Computer Network and Computer System. Press of Tsinghua University, Beijing. 2001, ISBN 7-302-04265-5

13 Lin C. Quality of Service of Computer Networks. Press of Tsinghua University, Beijing, 2004. ISBN 7-302-08076-3

14 Mainkar V, Trivedi K S. Transient analysis of real-time systems using deterministic and stochastic petri nets. In Quality of Communication-Based Systems. Dordrecht, Netherlands: Kluwer, 1995. 69～84

15 Ciardo G, Muppala J, Trivedi K. SPNP: Stochastic Petri Net Package. In: Proc. of Third International Workshop on Petri Nets and Performance Model, Kyoto, Japan, June 1989. 142～151

16 Liu X M, Li S X. Performance Modeling and Analysis of a Grid Monitoring Service. To appear in Journal of Information and Computational Science