

基于树模型的 XML 数据匹配查询算法的研究

蔡德山

河海大学计算机及信息工程学院, 南京 (210098)

E-mail: 2cds@163.com

摘要: 提出一种新的 XML 数据查询算法, Part-TreeMatch 算法。Part-TreeMatch 算法采用空间换时间的策略, 运用一种新的存储方法, 并在此基础上建立查询索引, 在一定程度上简化了 XML 模式树, 最后在查询索引中找到过程结果集, 并根据 Part-TreeMatch 算法进行模式树匹配, 从而减少了匹配结点的个数, 从根本上对查询算法进行了优化。并通过具体的性能分析阐述了算法的优点。

关键词: XML; 模式树; 模式树匹配; 过程结果集

中图分类号: TP301.6

1. 引言

随着可扩展标记语言 XML 迅速成长为 Web 上数据表示和数据交换的标准, XML 数据的查询变得越来越重要。可扩展性、灵活性、自描述性和简明性是 XML 的重要特性^[1], 正是这些重要的特性使得 XML 数据可以在不同的系统中进行传递交互, 从而成为一种网际语言。目前, Internet 平台上每天都会产生大量的 XML 数据^[2], 并不断地增长。大量 XML 数据的存在, 使得如何管理, 如何在这个庞大的数据集中查询到指定的信息成为尤为迫切的问题。

目前, Bruno 提出了 TwigStack 算法^[3], 它将过程结果集保存在一个堆栈链中, 每次匹配都从堆栈链中取, 这样 TwigStack 算法解决了在分解-匹配-合并步骤产生的大量无用的中间结果, 当匹配的元素为祖先孩子关系时, 匹配的路径很明确, 不会产生大量无用的中间结果。但是 TwigStack 算法在匹配某一个同时包含在几个路径中的节点时, 会对该节点进行重复的匹配。

H. Jiang, W. Wang 等人^[4]提出了结构化连接算法 Twig Join, 它是一种路径匹配算法, 可以很方便的进行带分支路径的 XML 查询, 尤其是只包含祖先子孙关系的路径匹配时, 不会产生任何的重复匹配, 效率较高, 但是, 当查询路径较为复杂, 尤其是在查询路径中包含分支时, 很容易造成重复的匹配操作, 降低 XML 查询的效率。

使用索引是一种有效处理路径表达式查询的方法, 它可以减少查询语言处理数据时连接操作的次数。但是从路径表达式的计算从本质上来说, 尽管这些查询语言在具体的处理路径表达式语法上都不甚相同, 但是它们都有一个共同的特征: 不但查询 XML 文档中具体的数据而且查询具体的结构, 这些查询语言的查询语句都可以表示成相应得查询模式树, 然后使用查询模式树去匹配 XML 文档中的数据, 这也是查询模式树在 XML 查询中广泛使用的重要原因。本文讨论了一种新型的存储模式, 并在该存储结构基础上建立索引结构的方法在 XML 文档模式树中进行查询树匹配的问题。

2. 模式树概念与模式树匹配

2.1 模式树概念

每一个 XML 文档都可以对应一颗模式树, 模式树是树和条件的组合 $P=(T,F)$, 其中 F 是和整个模式树对应的条件字符串, $T=(V,E)$ 是一棵结点和边都做了标记的树, V 是结点集

合, E 是相应的边集合。

具体说明如下:

- (1) 树 V 的每个结点都有一个标签(label)和一个索引号, 代表结点;
- (2) 每一条边都有类型, 如 PC(parent-child)、AD(ancestor-descendant)关系, 它们表示连接的两个结点之间关系;
- (3) F 是条件表达式, 对于查询匹配来说, F 是所有条件的集合;
- (4) ‘//’表示祖先—子孙边, ‘/’表示有着父子关系的边。XML 查询语言的查询使用(带标签结点)模式树来匹配 XML 数据库中相关部分的数据。模式树结点标签包含元素的标签 tag, 属性一值对或字符串值。模式树匹配边是父-子边(用单线表示)或者是祖先-后代边(用双线表示)。

2.2 模式树匹配

当前, 模式匹配作为模式间的一项基本操作已成为信息集成、数据仓库、语义 Web、电子商务等许多应用领域的基本问题。在电子商务中, 它可以映射不同 XML 格式之间的信息; 在数据仓库中, 它可以把数据源映射到仓库模式中, 方便信息查询。也正是因为它是很多应用中的一个关键步骤, 是非常重要的研究基础, 所以模式匹配已经成为近年来数据库研究的一个热点, 同时, 模式匹配作为数据管理应用中的基础性问题受到了普遍关注, 出现了多种针对特定应用领域和通用化的模式匹配方法, 模式匹配已经成为模型管理的一个重要组成部分。

由于常用的模式匹配方式是通过该领域的专家应用某些图形工具或手工来完成的, 自动化程度极低, 因此, 最好是有这样一些工具, 它们可以自动检测到确切的候选匹配, 从而尽量减少人为的决策。

模式匹配是模式集成中的一个普遍而重要的问题, 然而它又是最困难的。由于模式匹配是各种应用的关键, 所以它应该作为一个独立的组成部分而建立, 并且尽量使之通用化。

3. 基于树模型的 Part-TreeMatch 算法

3.1 模式树的存储结构

借鉴三叉链表的存储方式, 因为本文的模式树是 K 叉树, 这要求在模式树存储时, 运用 $K+1$ 叉链表进行存储。

$K+1$ 叉链表是 K 叉树的另一种主要的链式存储结构。 $K+1$ 叉链表与三叉链表的主要区别在于, 它的结点比三叉链表的结点多一个指针域, 该域用于存储一个指向本结点双亲的指针。 $K+1$ 叉链表的结点形式如下:

Data	parent	Child1	Child2	Child3	childK
------	--------	--------	--------	--------	-----	-----	-----	--------

根据 $K+1$ 叉链表存储结构^[5], 只要明确一个结点, 很容易找到该结点的父亲结点和各个孩子结点。这为 Part-TreeMatch 算法提供了基础数据支持, 尤其是在后面提到的匹配算法。

当然, 根据模式树可知, XML 文档所对应的模式树一般都不会是满 K 叉树, 但根据 $K+1$ 叉链表的存储方法, 一个叶子结点也需要存储 K 个空指针。Part-TreeMatch 算法主要从算法的时间复杂度出发, $K+1$ 叉链表的存储方式尚且满足 Part-TreeMatch 算法的要求, 有兴

趣的可以做更深一步的研究。

3.2 Part-TreeMatch 算法查询索引结构

如第二章介绍,索引是一个单独的、物理的数据库结构,它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。

索引^{[6][7]}提供指向存储在表的指定列中的数据值的指针,然后根据您指定的排序顺序对这些指针排序。数据库使用索引的方式与您使用书籍中的索引的方式很相似:它搜索索引以找到特定值,然后顺指针找到包含该值的行。

在数据库关系图中,您可以在选定表的“索引/键”属性页中创建、编辑或删除每个索引类型。当保存索引所附加到的表,或保存该表所在的关系图时,索引将保存在数据库中。

下面给出基于基于 $K+1$ 叉链表存储方式, Part-TreeMatch 算法查询索引结构如图 3.4:

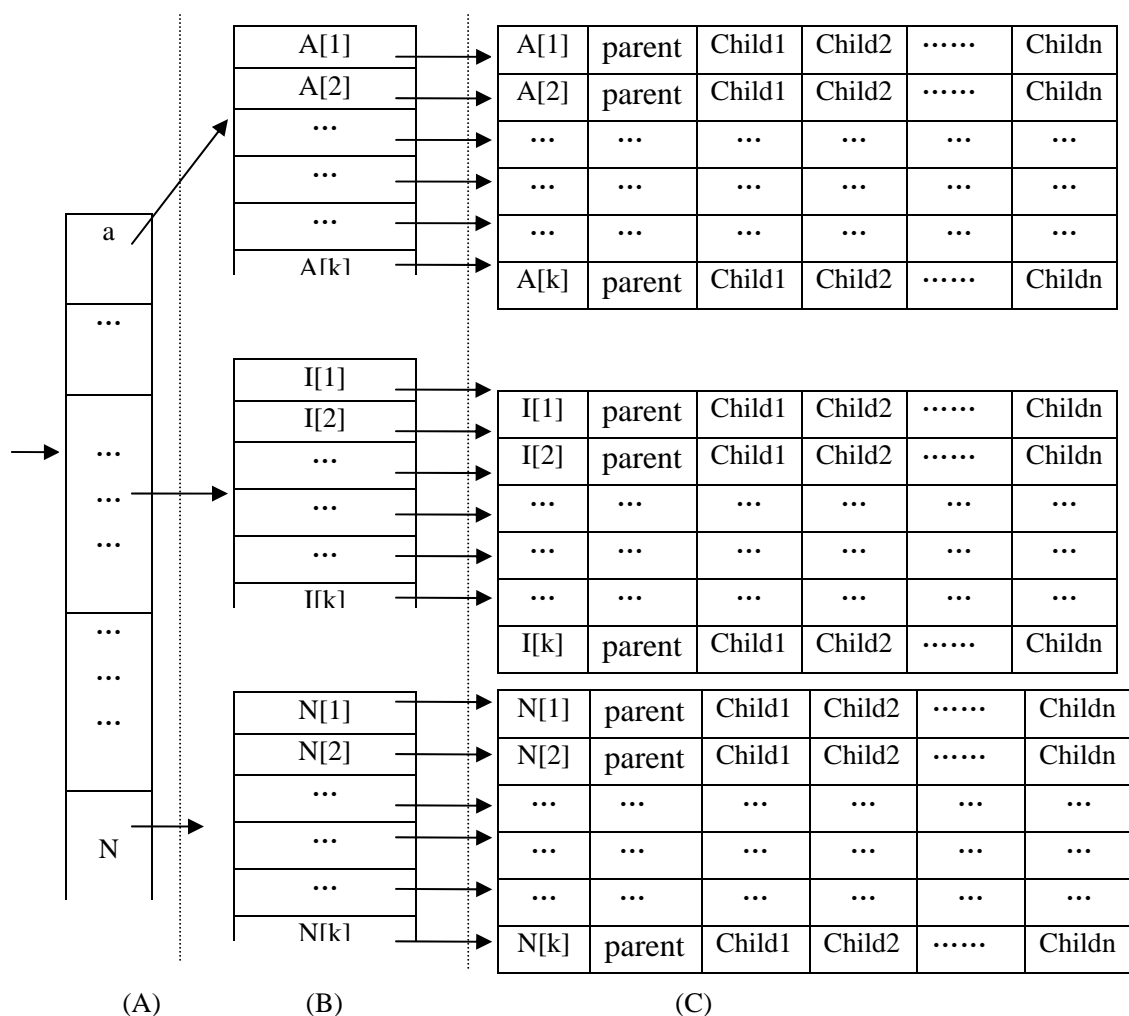


图 3.1 Part-TreeMatch 算法查询索引结构

该结构如图所示有三个层次:

(A) 中存储的是 XML 模式树中所有的不同结点索引, Part-TreeMatch 算法中也主要是通过这个索引减少了大部分匹配操作,对于提高 Part-TreeMatch 算法的时间复杂性有很大帮助;

(B) 中存储的是 XML 模式树中所有的结点信息, 具体信息如 (C) 中显示外, 还包括与 (A) 的关联指针。这保证了 Part-TreeMatch 算法进行树匹配时, 可以迅速检索到对应的结点;

(C) 中存储的是 XML 模式树中所有的结点信息, 使用 $K+1$ 叉链表的存储方式, 存储了指向本结点的双亲结点的指针、本结点的 data 和指向所有的孩子结点的指针。

3.3 基于查询树根结点的过程结果检索

任意给出一个查询树 T, 如图 3.6, 则 Part-TreeMatch 算法主要是依据查询树 T 的根结点, 在 Part-TreeMatch 算法查询索引结构中找到所有与查询树 T 的根结点相匹配的结点。所有这些结点都可能是最终结果树的根结点, 这个过程叫做过程结果检索。

Part-TreeMatch 算法的匹配算法时将查询树与以过程结果集中的结点为根结点的子树进行匹配, 并找到最终结果集。

3.4 基于树模型的 Part-TreeMatch 算法

在上述存储结构, 索引结构和过程结果集的基础上提出了基于树模型的 Part-TreeMatch 算法。

匹配算法的操作定义为:

若查询树 T 为空, 则空操作, 否则

比较过程结果集 Q 中的结点和 T 的根结点;

比较 T 的第一个子树和过程结果集 Q 的第一个子树;

比较 T 的第二个子树和过程结果集 Q 的第二个子树;

.....

.....

(K+1) 比较 T 的第 K 个子树和过程结果集 Q 的第 K 个子树。

Part-TreeMatch 算法如下:

```
Status Part-TreeMatch (Tree T, Tree Q, Status(* NodeMatch)){
```

```
//T 是过程结果集中的一个子树。
```

```
//遍历 K 叉树 T 的递归算法, 对每个数据元素调用函数 NodeMatch。
```

```
//调用实例: Part-TreeMatch (T, Q, NodeMatch);
```

```
If (Q){
```

```
    If(NodeMatch (T, Q)){ //在 3.3 中已经比较过, 当是根结点时恒成立
```

```
        For(i=1; i<=K; i++){
```

```
            If(Q→child[i])
```

```
                Part-TreeMatch(T→child[i], Q→child[i], NodeMatch);
```

```
        } //for 循环结束
```

```
    }
```

```
    }else return ok;
```

```
// Part-TreeMatch
```

Part-TreeMatch 方法中 NumberofChildren(T) 表示 T 节点的孩子节点总数, 根据 Part-TreeMatch 算法, 可以将过程结果集 Q 中的结点和查询树进行匹配, 并输出能否匹配作

为最终结果。其中 NodeMatch (q,T)主要进行两棵树的结点的比较，具体算法如下：

```
Status NodeMatch (BiTree q,BiTree T){ //判断两棵树相对应的两个结点是否相匹配
//采用 K+1 叉链表存储结构，NodeMatch 是对数据元素比较操作的应用函数，
if(q->data==T->data){
    return ok;
}else return error;
}
```

4. Part-TreeMatch 算法性能

文献[4]中的算法 TreeMatch 和本文算法 Part-TreeMatch 都属于匹配查询模式树的算法，本章主要是对这两个算法的时间复杂性进行比较分析。

算法 TreeMatch 的时间复杂性体现在在数据源中寻找匹配节点的主方法 ifnd0，时间复杂性取决于 ifnd0 函数的执行的次数方法。ifnd0 方法是一个递归方法，用 h 来表示递归的深度，每一次递归要调用 ifnd0 遍历所在节点的子节点假设查询模式树子节点个数为 n，则算法的复杂度为 n^h (n 的 h 次方)。

下面依据算法执行过程分析 Part-TreeMatch 算法的时间复杂性：

(1) 主要是对源 XML 模式树进行存储，并建立索引结构，为查询算法提供数据基础，这里假设该模式树为 K 叉树，深度为 h，若使用先序遍历，逐个检索 XML 模式树的结点，并为相同结点建立索引结构，则这一步进行 XML 模式树结点存储和建立索引的时间复杂性为 $O(K^h)$ ，即为 K 的 h 次幂。但是本文算法主要是处理 XML 文档模式树相对固定的情况，查询算法是一次存储，多次查询的情况，此时，此步中的预处理在时间复杂性中不再计算，因此此步的时间复杂性对响应速度的影响不大，时间复杂性为 $O(0)$ 。

(2) 主要是将查询树存储下来，在查询匹配时调用，这样的存储过程也就是一个简单遍历的过程。设查询树的结点数为 n，则算法的时间复杂性也应该为 $O(n)$ 。

(3) 主要是根据查询模式树的根结点（设为'a'），在目标 XML 模式树上检索出所有和'a'相匹配的结点，并记录下类似节点的个数，设为 p 个，同时记录下目标 XML 模式树中所有的互不相同的结点总数，如图 3.4 中（A）阶段不同结点的个数，设为 q。接下来 Part-TreeMatch 算法只需要在这个范围内进行检索匹配即可。这个过程中，因为目标 XML 树种所有的不同结点已经在第一结点建立索引时，存放在图 3.4 的(A)阶段中，只需要在这 P 个不同的节点中检索到与查询树根结点'a'相匹配的结点，所以 Part-TreeMatch 算法的时间复杂度为目标 XML 树中所有的互不相同的结点总数，即为 $O(q)$ 。

(4) 首先在进行模式树匹配之前，根据（3）中的过程结果集和查询模式树的深度，确定匹配结点的个数，根据第三章中的例子，匹配算法实际上是要在如图 4.1 中的结果树集合中找到能够完全匹配查询模式树 T 的结果树的过程。

设本文 Part-TreeMatch 算法的总的复杂性为 N，则有

$$\begin{aligned} N &= O(0) + O(n) + O(q) + O(n * p) \\ &= O(0 + n + q + n * p) \\ &= O(n + q + n * p) \end{aligned}$$

综上，得到算法 PattemMatch 的时间复杂性为 $O(n + q + n * p)$ ，算法 PaternMatch 比算法 TreeMatch 复杂性有了很大提高，由于算法 PatemMatch 复杂性只和查询模式树节点个数相关，和查询模式树深度没有直接关系，不会随着深度的增加带来时间损耗的显著增加。

5. 总结

本文提出了一种基于XML文档树的查询算法Part-TreeMatch, 它的结构模式是在XML模式树的基础上建立相应的索引, 并根据索引直接与过程结果集进行匹配, 结果相匹配才继续, 否则, 跳出此次匹配, 因此, 大大减少了匹配节点的数量, 并迅速在查询匹配结果中找到最终查询模式树的匹配结果。最后通过查询算法Part-TreeMatch的性能分析得知查询算法Part-TreeMatch是一种较为高效的XML查询算法。

参考文献

- [1] M. Marchiori. XML Query.<http://www.w3.org/XML/Query>,2002
- [2] [美]Murdoch Mactaggart. Enabling XML security.
http://www-900.ibm.com/developerWorks/cn/xml/s-xmlsec/index_eng.shtml,2001.
- [3]N.Bruno, N.Koudas, D.Srivastava.Holistic Twig Joins: Optimal XML Pattern Matching [C] .SIGMOD, 2002:310-321
- [4] H. Jiang, W Wang, H.Lu. Holistic twig joins on indexed XML documents [C]//VLDB2003:273-284
- [5] 张晓琳, 王国仁. 《面向对象的 XML 数据的存储模式研究》. 小型微型计算机系统, 2004,25:11.13.
- [6] 路燕, 张亮, 段起阳. 《一种基于 DTD 的 XML 索引方法》. 计算机研究与发展, 2005,42(1):31-35
- [7] 包小源, 宋再生, 唐世渭. 《Suffindex 一种基于后缀树的 XML 索引结构》. 计算机研究与发展, 2004,41(10):1794-1780

XML data match inquiry algorithm research based on tree model

Cai Deshan

Department of Computer Science and Technology, Hohai University, Nenjing (210098)

Abstract

This paper focuses on a new XML data query algorithm, Part-TreeMatch algorithm. Part-TreeMatch algorithm uses the strategy of trading the time by the space, that is to say, uses a new storage methods, and sets up an indexed inquiries on the base of the new storage methods, to some extent, the XML-tree is simplified, at the last, the inquiries set to find the course from the indexed queries and in accordance with Part-TreeMatch algorithm pattern tree matching, thereby reducing the number of matching nodes, fundamentally on the query optimization algorithm. Through specific performance analysis on the merits of the algorithm.

Keywords: XML; Tree pattern; Tree pattern matching; Collection of process result