对XML数据索引的回顾*)

刘振中　董道国　薛向阳

(复旦大学智能信息处理开放实验室,计算机科学与工程系　上海200433)

摘　要　随着Internet的迅速发展,XML已成为Internet网上数据表示与交换的事实标准,大量应用采纳了XML,例如Web Service中的数据表示和交换、MPEG7中定义的多媒体特征描述子等。目前,查询XML数据需要用XPath,由于查询语句的复杂性,很难找到一种通用的索引结构能有效支持任意查询。因此,在近十年的研究历程中,为了实现XML数据的快速查询,人们提出了大量索引结构。本文就是对已经提出的一些代表性的索引结构进行分类和总结,并指出其优缺点和所能支持的查询。

关键词　XML,索引结构,查询处理

Review of Indexing XML Data

LIU Zhen-Zhong　DONG Dao-Guo　XUE Xiang-Yang

(Dept. of Computer Science, Fudan University, Shanghai 200433)

Abstract　With the rapidly development of Internet, XML has become the new standard for data representation and exchange. More and more applications cannot run smoothly without XML, such as Web Service, MPEG7 etc. Currently querying XML data mostly relies on XPath, and it's difficult for us to build a general index structure for XML data because of the complexity of any query. After introducing some typical indices for XML data and analyzing their merits and drawbacks at length, this paper does a classification of XPath and subsequently offers appropriate index for each of them.

Keywords　XML, Index structure, Query processing

1　引言

可扩展标记语言(XML,Extended Markup Language)在Internet上占据越来越重要的地位,它已经成为Internet上数据表示与交换的新标准,同时也被认为是用来定义半结构化数据最有效的手段。XML允许用户引入新的标记来描述它们自己特定的应用,而这些引入的标记可以用来表征语义,比如XML元数据可以用来描述一个网站的结构,通过产生交互性的网站地图来方便网页间的导航[24]。不仅如此,XML格式数据在诸如Web Servce、MPEG-7、电子产品目录以及分类中均得到了非常广泛的应用。

XML数据的急剧增多给XML文档的维护工作带来了很大挑战,为了更好地对XML文档进行维护,使得其它应用程序更容易对XML文档中的数据进行操作,人们从存储到索引、再到查询做了积极深入的研究。这些研究促进了XML的发展,解决了现实中的诸多问题,许多优秀的研究成果已经有了一定规模的应用,比如Lore[18]。但是由于XML数据的多样性以及用户日益增长的查询需求,人们很难找到一种能够同时适应不同的数据来源(XML纯文本文档、关系型数据库以及其它各种应用数据等)并能够有效处理各种查询请求的通用索引结构。因此,针对不同的XML应用,人们提出了不同的索引结构,这些索引结构能够满足特定环境下的需求。本文详细分析了当前XML索引结构研究大致的几个走向,指出了它们的优缺点以及适用范围,希望能够对XML索引结构的研究做一个比较全面的回顾和总结。

2　基本概念

2.1　XML查询及XPath语言

XML格式数据能够得到广泛应用的前提在于能够在这种半结构化的数据中快速找到用户需要的数据,目前已经提出很多种XML查询语言,比如XPath[19]、XML-QL以及W3C标准XQuery等等。这些语言以一种路径的方式在层次结构的XML文档中进行导航,它们与关系数据库中SQL语言的主要差别在于引入了正则表达式,因此一条查询语句返回的结果将是一个集合,这个集合包含了查询文档对象中所有匹配查询语句的路径,这些语言的查询是一种布尔查询(Boolean Retrieval)。在这些查询语言中,XPath具有非常强大的查询表达能力,当前绝大多数XML索引结构的研究都是基于XPath的。下面是XPath语言的几个简单例子:

·employee[@secretary and @assistant]:选取当前节点所有具有secretary和assistant属性并且叫做employee的儿子节点;

·para [5]:选取当前节点的第五个叫做para的儿子节点;

·*/para:选取当前节点的所有叫做para的子孙。

总的来说XPath中的查询语句分为两大类:简单路径

*)资助项目:国家自然科学基金资助项目(60003017,69935010);国家863计划资助项目(2001AA114120,2002AA103065);上海市政府科技发展基金资助项目。**刘振中**　硕士研究生,主要从事XML数据索引结构研究;**董道国**　博士研究生,主要从事多媒体信息检索技术研究;**薛向阳**　教授,博导,主要从事多媒体信息检索与处理等技术研究。

(Simple Path)查询语句和正则路径(Regular Path)查询语句。两者的区别在于前者不含有通配符(*、+、?等等),而后者含有通配符。

2.2 XML索引结构

如果对XML文档不构建索引结构,那么针对XML数据的任何查询都很可能导致对整个文档树的遍历,随着XML数据集的增大,这种遍历所花费的开销是不可忍受的。XML索引结构的提出正是为了提高查询的效率,在速度与准确性两方面为查询提供更大的灵活性,通过减少访问那些与查询不相关的数据集来实现快速查询。

XML索引结构与传统的文本索引存在很大的不同,它的研究并不关心XML文档中的文本内容,而是快速查找出满足某一特定路径结构的节点集。除了与查询需求密切相关,XML索引结构还在很大程度上依赖于被索引的XML文档集。比如这些文档的存储方式(纯文本或者存放在关系型数据库中等),是否有文档类型定义(DTD, Document Type Definition)、文档模式(Schema)以及数据集的动态性等。现在主要研究现实世界大量存在的、无DTD的XML文档,由于其结构的不可预测性导致索引结构的复杂性。

2.3 索引结构分类

人们在近十年中提出了很多索引结构,它们的存放媒质、构造索引的方法以及所能够支持的查询种类各不相同,根据各自特点,给出以下几种分类方法:

1. 根据索引的存放媒质不同,可分为文件系统、关系数据库和两者的混合方式。XML数据及其索引结构往往作为文本文件存放在文件系统中,如DataGuides[1]、XISS[10];为了充分利用关系数据库的成熟技术,将XML文档存放在关系数据库中以解决XML文档的查询和维护工作,如STORED[6];利用关系数据库和文件系统联合做索引,如文[15]。

2. 根据索引的组织形式不同,可以将XML索引结构分为结构型和离散型。结构型指索引结构中直接保存原始文档中的结构(路径),相当于XML文档的Schema,如1-Index[2]等;而离散型则是将原始文档中的节点聚类存储,原始文档中的结构通过索引结构中的附加信息计算获得,如XISS[10]。

3. 根据索引所支持的查询语句的种类,可以将XML索引结构分为可支持简单查询类和可支持正则查询类。前者指可有效支持简单查询语句的索引结构,如DataGuides[1]、A(k)-Index[3]等;后者则是那些能够支持正则查询语句的索引结构,如XISS[10]等。

3 XML索引结构

限于篇幅的原因,本文无法一一介绍所有XML索引结构,而是选取一些具有代表性的给予比较详细的描述。本节将根据索引结构所能支持的查询种类以及存放媒质的不同来总结这些索引结构。

3.1 可支持简单查询

可支持简单查询的索引结构能很好支持简单查询语句,但是对于正则查询语句则显得非常低效。下面将根据索引存放媒质的不同分别进行介绍。

3.1.1 *索引存放在文件系统中* 这一类索引也可称为Covering Index。它通过一些诸如节点等价、路径等价的技术来抽取XML文档中隐含的结构(类似于Schema),对原始文档进行无损或有损压缩,来得到比原始文档小很多的索引结构。其实Covering思想在其它研究领域早已有之,比如对于文本串AABBC,ABC从某种意义上讲就是原始文本的Covering Index。

构造这一类索引结构的过程实际上是一系列的节点合并过程,各种Covering Index的差异本质上在于所采用的节点合并策略。同时,为了使本文总结有一个统一的数据模型,我们将对某些基于OEM(Object Exchange Model)[20]的算法进行相应改动,使之适用于传统的图结构,这两种数据模型的主要差别在于一个用路径中的边来表示XML文档中的节点,而另一个则用路径中的点来表示。下面将介绍几种典型的Covering Index。

3.1.1.1 **DataGuides**[1] 它算是最早出现的一种Covering Index,并且这一算法在Lore DBMS中得到了实现。DataGuides首先提出了一种路径的合并策略:s表示原始文档数据对象,如果d是它的DataGuides,一定满足如下条件:s中的任何一条路径,在d中存在且只存在一条,同时d中的任何一条路径,在s中也一定存在。很明显,图1的(b)和(c)都是(a)的DataGuides。

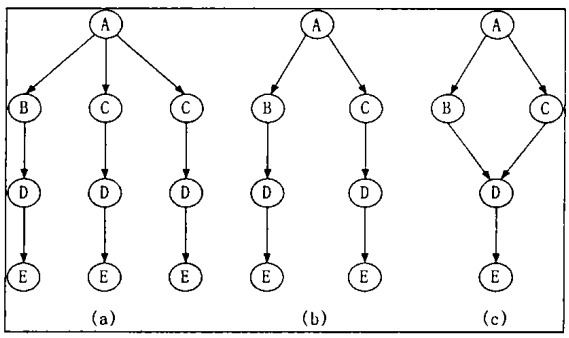

图1 DataGuides

可以看到,虽然(b)和(c)都是(a)的DataGuides,都能起到索引作用,而且由于(c)更小,因此直观感觉应该选择(c)作为索引结构。但是(a)的任何变更(如插入、删除等)都可能给索引结构带来全局影响,这种代价往往是不能接受的。因此,又有人提出StrongDataGuides概念,它与DataGuides的主要差别在于路径合并的策略:只有当s中两条路径完全相同时,才将这两条路径进行合并。可以看出,在(c)中,虽然路径A/B/D/E和路径A/C/D/E不完全相同,但是由于路径的部分相同导致了部分合并。因此很显然(b)是(a)的StrongDataGuides,而(c)则不是。还值得一提的是,将s转化为d与将一个非确定有限自动机转化为确定有限自动机是同构的。

3.1.1.2 **1-Index**[2] 与DataGuides的出发点不一样,1-Index不是从路径而是从节点着手来考虑构造索引的,在1-Index中,采用了一种合并相似节点的技术来对原始文档中相似节点进行合并。下面给出1-Index中节点相似的定义。

DB表示原始文档数据(图结构),那么如果作用在DB节点上的二元关系"~"满足下面的四个条件,则称"~"为一种双向相似关系(bisimulation):

1. 如果$V \sim V'$,而且V是根节点,那么V'也是根节点
2. 相反,如果$V \sim V'$,而且V'是根节点,那么V也是根节点
3. 如果$V \sim V'$,而且对于任何$U \to V$边如果存在另一条边$U' \to V'$,则$U \sim U'$
4. 相反,如果$V \sim V'$,而且对于任何边$U' \to V'$如果存在另一条边$U \to V$,则$U \sim U'$

如果仅仅满足1、3条,则将两元关系"~"称为单向相似关

系(simulation)。设E是任意节点,定义$L(E)=\{l|l=E_0E_1\cdots E$,其中$E_0$是根节点$\}$,能够证明:

$U\sim V\rightarrow L(U)=L(V)$

因此也就容易证明根据这样一种相似的规则来对原始文档中的节点进行合并所得到的索引结构(仍然是图结构),其路径覆盖了原始文档中的所有路径,也就是说这样的索引结构是安全的,不会遗漏返回的结果。从图2中可以看出1-Index和DataGuides的区别和联系。

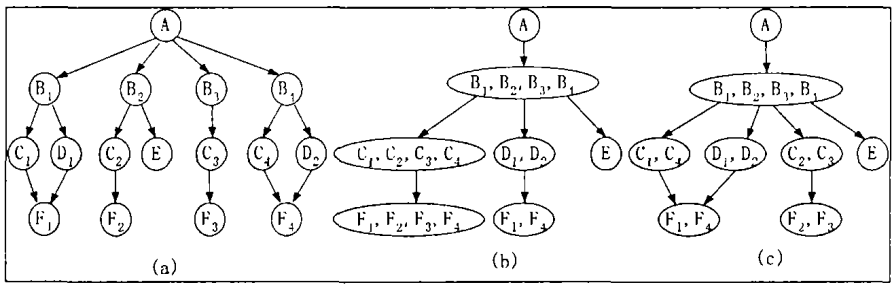

图2 Strong DataGuides和1-Index

Bisimulation是70年代图论中提出的概念,图2中(a)表示原始文档,(b)是(a)的Strong DataGuides,而(c)是(a)的1-Index。比较(b)和(c)可以看出,1-Index要比Strong DataGuides小,这种索引结构的减小是因为1-Index中不存在重复节点,而Strong DataGuides中会存在重复节点,如(b)中的F_1和F_4均出现了两次,但在1-Index中这些节点不会重复。

上面提到的1-Index仅仅适应简单路径表达式查询,不能有效地支持正则表达式查询语句,2-Index[2]是为了支持正则表达式查询语句而提出来的,但是这种索引结构有非常大的局限性,它仅仅能支持由节点、"-"以及"*"组成的非常简单的正则表达式查询语句,比如:*.A.-.C。

3.1.1.3 **A(k)-Index**[3] 容易看出DataGuides以及1-Index完全保存了原始文档中的所有信息,这样的索引结构可以看作对原始文档的一种无损压缩。有些时候,这种无损压缩的结果可能仍然会很大,达不到期望的大小,因为这种压缩完全决定于原始文档中相似节点的数量。如果说1-Index在节点相似算法上采取的是一种全局性相似,那么A(k)-Index则采取了一种局部相似算法,在索引结构大小与准确度这两方面提供一个折衷,使得可在损失一定准确度的前提下灵活控制索引结构的大小。

A(k)-Index在定义节点相似时采用k-similarity($\approx^k$)[21]概念:

1. 任何两个节点U和V,$U\approx^0V$当且仅当:U和V具有相同的节点名称(tag name);

2. $U\approx^kV$当且仅当:$U\approx^{k-1}V$,并且对于U的任何父亲U',存在V的一个父亲V',有$U'\approx^{k-1}V'$,反之对于V的任何父亲V',存在U的一个父亲U',有$U'\approx^{k-1}V'$。

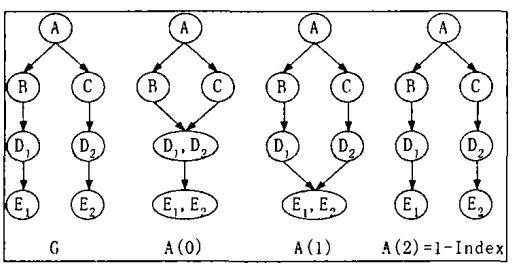

图3 A(k)-Index

可以看出,参数k越小,这种相似性越局部化,换句话说,相似节点的数量会越多,k等于零时,凡是具有相同元素名称的节点均相似;当k增大时,这种相似性就趋向于全局化,当k趋于无穷大时,由于这个时候不会损失原始文档中的任何结构信息(某些路径),因此此时的A(k)-Index便会与1-Index或者DataGuides相同。图3给了我们对A(k)-Index更直观的印象。

同时,A(k)-Index能够保证路径长度不超过k的返回结果是正确的。采用A(k)-Index虽然会损失返回结果的精度,但是能够比较好地控制索引结构的大小。

3.1.1.4 **相关研究** 文[12]提出了相关算法,能够有效地从原始文档中提取出符合文档的DTD,这种DTD也可以看作是一种索引,单纯从索引结构大小上来说,这样提取出来的DTD比上面提到的Covering Index小,因为DTD实际上是正则表达式的集合,因此对于一条路径A/A/A,Covering Index中会存在三个节点,而在DTD中只存在一个节点。另外文[14]也提出了一种提取Schema的算法。

由于DataGuides的获得需要很高的代价,特别是对于海量数据,要获取它的DataGuides代价是非常大的,文[13]提出了一种折衷方法,能够更有效提取出一种跟DataGuides相近的索引结构。

3.1.2 *关系数据库存放数据* 关系数据库已经非常成熟,利用关系数据库来存放XML数据,并以此来解决XML数据的存储及管理是一个非常自然的想法。通常,利用关系数据库解决XML文档的管理需要解决的问题有:(1)两种不同数据模型之间的映射;(2)XML查询语句向SQL的转换;(3)从关系数据库返回的结果向XML转换。

STORED(Semistructured TO Relational Data)[6]采用数据挖掘的技术,类似于上面Covering Index的思想,从原始文档中抽取出Schema(但与上面的Schema不同,这里的Schema主要是为了能够与关系数据库中的Schema相对应),尝试了将这两种不同的数据模型进行映射,对于不能映射的部分,采用overflow schema(图结构,实际上与Covering Index类似,极端情况就是原始的XML文档)来支持。

如图4所示,图4(a)是省略了值的原始文档,图4(b)是根据从图4(a)中抽取出来的Schema到数据库表的一个非常初级的映射。显然这里Schema的抽取是根据address的类型来进行的,虽然这里可以将每一个taxpayer映射为一张表,但是当XML数据太大时这种表的合并是必需的,同时在向数据库映射的过程中,会有空值或节点的重复出现,因此减小所占的磁盘空间也是非常重要的,这些在文[6]中都有比较详细的讨论。文[5]的映射模式与此不同,它将一个节点及其子孙(DTD中定义的可以重复多次出现的节点除外)放到一个关系中,这样可以更多地保留原始文档中对象的层次关系。

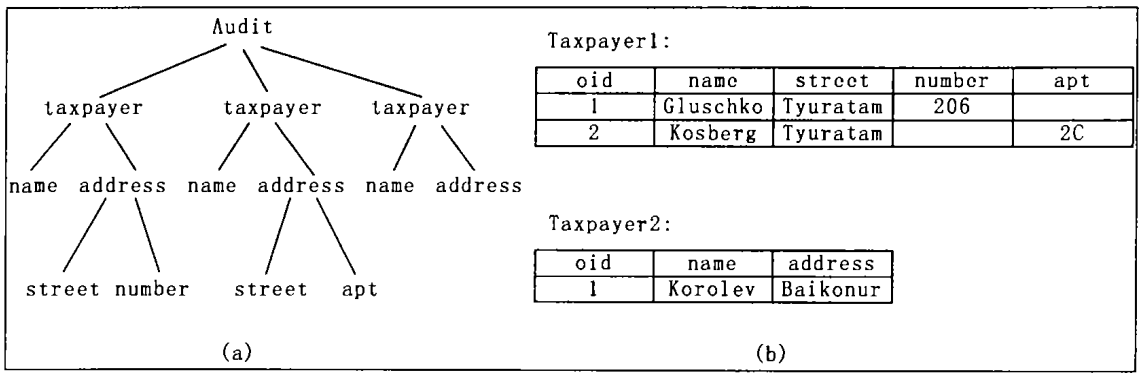

图4 Schema 的映射

查询语句的转换算法比较固定，依赖于数据库系统对 SQL 语句的支持。文[4,7]都基于当前标准 SQL 提出了转换算法，同时对现有 SQL 标准无法支持的转化（比如 XML 中存在的 IDREFs、查询语句中复杂的比较操作等）进行了总结。由于这种转化与数据库中的关系紧密联系，事实上是非常复杂的，文[5]在 XML 查询语句与数据库中的关系之间通过增加一个 XML 视图，使得这种转化不再直接依赖数据库中的关系，降低了复杂度。

至于从数据库返回的结果向 XML 数据格式的转换算法，则依赖于所希望得到 XML 数据的 Schema，文[16,17]都提出了非常有效的将存放于关系数据库中的数据转化为 XML 格式文档的算法，同时许多商业的关系数据库也都支持了这一转化，比如 Oracle、Access2002等等。

除了关系数据库本身的优势，利用关系数据库存放 XML 数据还可以解决上面 Covering Index 不好解决的“序”问题[4]，比如 XPath 中的 following、following-sibling 以及 position操作等等。

将图5(a)中的元素按某种“顺序”（比如前序）给出标号，可在数据库中相应地建立如下一张表：Edge(id, parent_id, end_desc_id, path_id, value)，用来存放(a)中的节点，其中 id 是上面给出的标号，end_desc_id 是当前节点按前序排列的最后一个孙子，而 path_id 是因为效率原因引进的一个标志当前节点所在路径的字段。这样，XPath 中的 following 操作就可以如(b)那样转化为 SQL 语句。

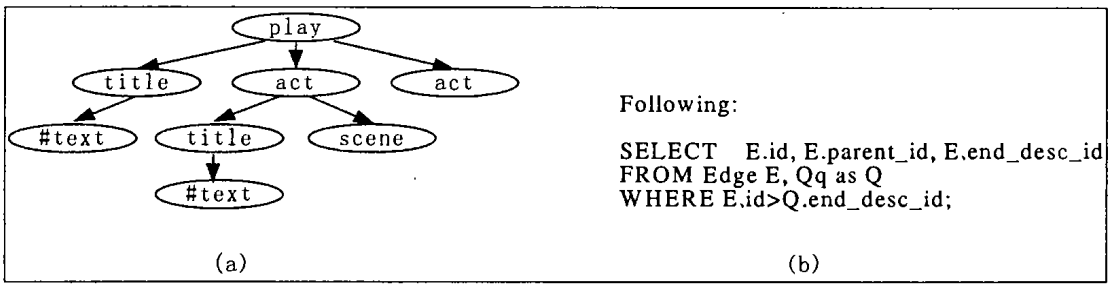

图5 XPath 中的 following 操作与 SQL 语句

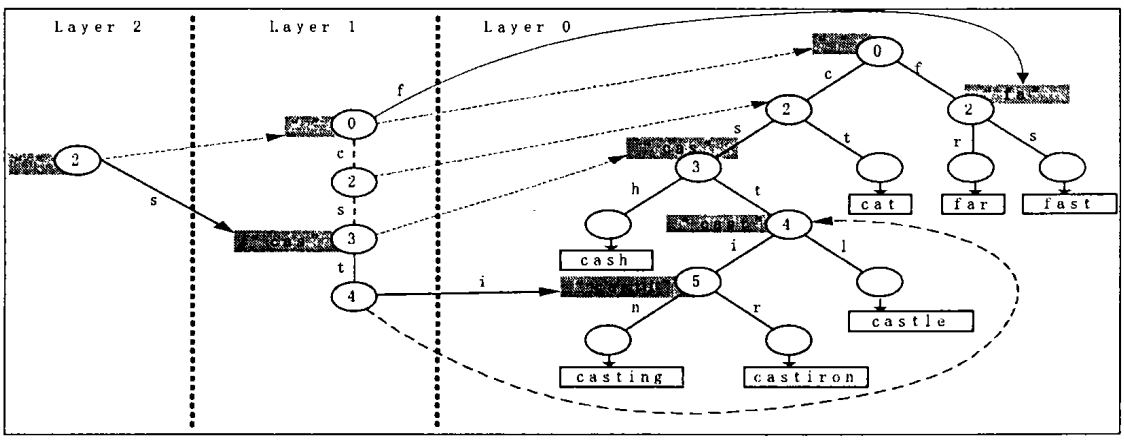

图6 Index Fabric

3.1.3 *混合存储* 一个典型的例子就是 Index Fabric[15]，它与上面提到的几种索引结构有本质的区别，Covering Index 将一条路径看作由许多元素构成，元素的定位根据路径来导航，而 Index Fabric 将任何一条路径编码成字符串，然后将这些字符串插入到一种特殊的索引结构(Patricia trie)中，从而大大优化复杂查询语句的处理，同时将 trie 中的每一条路径及其对应的值存放在数据库中以加快检索。

Index Fabric 采用 Patricia trie 来为编码后的字符串（对应于查询表达式）做索引，从图6中可以看到，Layer 0中 节点（以圆圈表示）以字母在原字符串中的位置来标志，因此 Patricia trie 的大小并不取决于插入字符串的长度，事实上，

任何字符串的插入最多增加一个节点的开销。由于篇幅的原因，有关 Patricia trie 的细节参看文[9]。

Patricia trie 的非平衡性，使得它的应用受到很大制约。Index Fabric 为了克服 trie 结构的这一缺陷，引入了纵向层的概念，如图6所示，将原始的 trie 结构按照某种合理的大小分成多个块，在这些块的基础上再用 trie 做索引，使原先的 trie 结构最终达到平衡。图6中的虚箭头称为 far link，表示箭头两端一种等同的关系；实箭头称为 direct link，表示箭头的始端经过某个字符导航到箭头的末端。

Index Fabric 的缺点在于块的大小不固定，给定一个 trie 结构，通常难以给出一个合理的块划分。而且基于将路径表达式编码成字符串的方法本身限制了这种索引结构不可能支持含有通配符的查询语句，但是由于 Index Fabric 比其它索引结构要小很多，因此可以将节省下来的开销做更多的优化，比如可以留出一块缓冲区来存放那些经常出现的查询路径。

3.2 可支持正则查询类

到目前为止，我们介绍的所有索引结构都不能够或者不能很好地支持含有通配符的查询语句，Covering Index 仅仅能够支持非常有限的简单正则查询语句，单纯的用关系数据库存储 XML 文档以及 Index Fabric 都不能解决这一问题，然而在现实当中更有价值的是正则查询表达式。

XISS[10]是可支持正则查询的索引结构中最具代表性的一种。XISS 通过 B^+ 树做索引，将文档中具有相同节点名称的节点存储到一起，能够通过节点名称快速找到所有具有该名称的节点，这样通过将查询表达式进行分解，然后利用所谓的 Numbering Schema 来判断节点之间的祖孙关系从而获得满足查询条件的节点。XISS 与其它索引结构本质的不同在于提出了 Numbering Schema，这种 Schema 实际上是为每一个节点附加信息，通过这些信息能够有效地判断任何两个节点之间是否存在祖孙关系，用(order，size)对来标记树中的节点，规则如下：

1. 如果 x 是 y 的父亲，则有 $order(x)<order(y)$ 并且有 $size(y)+order(y)\leqslant size(x)+order(x)$；

2. x 和 y 是兄弟节点，且在前序排列中 x 在 y 的前面，那么有 $order(x)+size(x)<order(y)$。

可以得出如下结论，任何两个节点 x 和 y，x 是 y 的祖先当且仅当：

$order(x)<order(y)\leqslant order(x)+size(x)$；

有了这种 Numbering Schema，XISS 提出子表达式的概念：

·一个单独的元素或者属性是一个子表达式；

·具有一个属性的元素是一个子表达式(例如：figure [@caption="Tree Frogs"])；

·具有两个元素的表达式是一个子表达式(例如：chapter/figure 或者 chapter/-*/figure)；

·一个子表达式的 Kleene 闭包(+，*)是一个子表达式；

·两个子表达式的联结是一个子表达式。

通过将原始查询语句分解为子表达式，然后分别针对这些子表达式实现查询，最后对这些中间结果进行联结获得查询结果集。这里中间结果的联结操作正是利用了上面的 Numbering Schema，例如对正则表达式 A/-*/B，首先将其分解成为子表达式 A 和 B，通过 B^+ 树有效地获取所有的 A 节点和所有的 B 节点，然后利用上面的 Numbering Schema 获取那些具有祖孙关系的 A、B 节点的匹配。图7给出了 XISS 中元素-元素联结算法，其中 did 是文档编号：

```
EE-Join:Element and Element Join
Input:{E1,…,Em},Ei 是有相同文档标记的元素集合;
      {F1,…,Fn},Fj 也是有相同文档标记的元素集合.
Output:(e,f)集合,其中 e 是 f 的祖先。
    //Sort-merge {Ei} and {Fj} by doc.identifier;
1:for each Ei and Fj with the same did do
  //Sort-merge {Ei} and {Fj} by parent-child relationship;
2:        foreach e∈Ei and f∈Fj do
3:            if(e is an ancestor of f) then output (e,f);
              end
      end
```

图7 XISS 中元素-元素联结算法

XISS 中将这种中间结果的联结归结为三类，即元素-元素联结、元素-属性联结和闭包联结(实质上是元素-元素联结)，显然对每一个中间结果进行联结后可得到最终结果。虽然这样一种方法的确能够解决所有的通配符问题，但是也必须清醒地看到，这种中间结果的联结很有可能是非常耗时的，特别是对于长路径的简单表达式，因此文[11]给出了另外一种 Numbering Schema，并在此基础上引进了高维索引中 R-Tree[22]的概念，避免了许多不相关结果的处理，下面是文[11]的 Numbering Schema：

将文档中的任何节点与一个区间段(start，end)相对应，start 和 end 的赋值遵循如下原则：

1. 如果节点 x 是 y 的父亲，则有区间 $(start_x,end_x)\in(start_y,end_y)$；

2. 如果 x 是 y 的兄弟，并且在前序排列中 x 在 y 的前面，那么有 $end_x<end_y$。

根据上面两条原则，可以为每一个节点用一个区间来标记，那么上面提到的任何两个节点间祖孙关系的确定是非常容易的：节点 x 是 y 的祖先当且仅当 $start_x<start_y<end_x$。

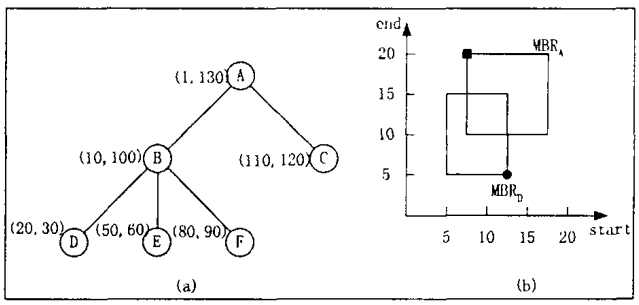

图8 节点记号与二维空间表示

图8中(a)给出了这样一种标记实例，在(b)的坐标系中，以横轴表示 start，纵轴表示 end，那么一个节点 A 是另一个节点 B 的祖先，当且仅当 A 在 B 的左上方。MBR_A 和 MBR_D 分别表示两个中间结果集和的最小外接矩形，那么这两个中间结果至少存在一对匹配的条件是 MRB_A 左上角(图中的小黑正方形)是 MBR_D 右下角(图中的小黑圆圈)的祖先。

与 XISS 不同，文[23]采用有限自动机的理论来处理正则表达式语句，其中一个比较重要的贡献就是将正则表达式化简(比如 A/-*/-/B 化为 A/-*/B)，这一化简算法实际上可以用在 XISS 的查询处理中。对于一输入串，文[24]能够找到与这一输入字符串相匹配的正则表达式，因此它还可以应用在诸如 XML 数据过滤等方面。

总结 Covering Index 能很好胜任简单查询语句的处理，在 Covering Index 中，由于 DataGuides 的获取代价比较大，同时由于 DataGuides 会存在节点的重复，因此总体性能不如1-Index，A(k)-Index 实际上可看作是1-Index 的扩展，它

使得在正确性与索引结构的大小这两者的权衡上更加灵活。至于 Index Fabric 也能很好适应这一类查询语句，但是它有一个前提，就是基于 STORED[6]的，因此效率要受到 XML 数据模型与关系数据库数据模型两者之间如何映射的影响。XISS[10]能够很好处理正则表达式语句，但是这种将查询语句分解的方式对简单查询语句开销过大，因为比如 A/B 的处理，需要找出所有的 A 和所有的 B 然后进行两两匹配，随着路径长度的增加，这种处理方式的 IO 代价可能要比 Covering Index 大许多，因此 XISS 也只适应正则表达式的处理。当然，XISS 可以在 Covering Index 的基础上做索引，这样可以使得它的 IO 代价与 Covering Index 相差不太大。用数据库存放 XML 数据优点不言而喻，但其缺点也很明显，首先需要额外地增加格式转换，另外，两种数据模型之间不能有效找到一个好的映射、XPath 向 SQL 的转换算法不通用以及原始 XML 文档中的对象层次关系在关系数据库中难以维持都是有待研究的。

XISS[10]能非常有效支持正则查询，但是 XISS 还有一个缺陷，就是它的 Numbering Schema 仅仅适应于树结构的 XML 文档，因此 XISS 不能适应图结构的数据对象。同时可以看到，XISS 易存放到关系型数据库中，虽然会降低效率，但是数据的维护会变得非常方便。

通过上面总结也可以看到当前 XML 索引结构研究的困境，今后 XML 索引结构的研究必须加强如下两方面：(1)研究适用范围更加广泛的索引结构，能更加全面地支持 XPath 查询语句；(2)目前的索引结构基本上都是静态的，今后应该加强动态索引结构的研究。

参 考 文 献

1 Goldman R, Widom J. DataGuides: Enabling query formulation and optimization in semistructured database. In: Twenty-Third Intl. Conf. on Very Large Data Bases, 1997. 436～445

2 Milo T, Suciu D. Index structures for path expressions. In: ICDT: 7th Intl. Conf. on Database Theory, 1999

3 Kaushik R, Shenoy P, Bohannon P, Gudes E. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data. In: Proc. of ICDE, 2002

4 Tatarinov I, et al. Storing and Querying Ordered XML Using a Relational Database System. In: Proc. of SIGMOD, 2002

5 Shanmugasundaram J, et al. A General Technique for Querying XML Documents using a Relational Database System. SIGMOD Record, Sep. 2001

6 Deutsch A, Fernandez M, Suciu D. Storing Semistructured Data with STORED. In: Proc. of SIGMOD Conf. 1999

7 Shanmugasundaram J, et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB, 1999

8 Zhang C, et al. On Supporting Containment Queries in Relational Database Management Systems. In SIGMOD 2001

9 Knuth D. The Art of Computer Programming. Vol. Ⅲ, Sorting and Searching. Third Edition. Addison Wesley, Reading, MA, 1998

10 Li Q, Moon B. Indexing and Querying XML Data for Regular Path Expressions. In: Proc. of VLDB, 2001

11 Shu C, Zografoula V, Zhang D, et al. Efficient Structural Joins on Indexed XML Documents. In: Proc. of VLDB, 2002

12 Gionis A, et al. XTRACT: A system for extracting document type descriptors from XML documents. In: Proc. of ACM-SIGMOD 2000 Intl. Conf. on Management of Data, 2000

13 Goldman R, Widom J. Approximate DataGuides. In: Proc. of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jan. 1999. 436～445

14 Nestorov S, Abiteboul S, Motwani R. Extracting schema from semistructured data. SIGMOD Record, 1998, 27(2): 295～305

15 Cooper B F, Sample N, Franklin M J, Hjaltason G R, Shadmon M. A Fast Index for Semistructured Data. In: Proc. of VLDB, 2001

16 Fernandez M F, Morishima A, Suciu D. Efficient Evaluation of XML Middle-ware Queries. In SIGMOD, 2001

17 Shanmugasundaram J, et al. Efficiently Publishing Relational Data as XML Documents. In VLDB 2000

18 McHugh J, et al. Lore: A Database Management System for Semistructured Data: [Technical Report]. Stanford University Database Group, Feb. 1997

19 http://www.w3.org/TR/XPath20/: XML Path Language (XPath) 2.0, W3C Working Draft 15 Nov. 2002

20 Papakonstantinou Y, Garcia-Molina H, Widom J. Object Exchange Across Heterogeneous Information Sources. In: Proc. of the Eleventh Intl. Conf. on Data Engineering, Taipei, Taiwan, 1995. 251～260

21 Milner R. A Calculus for Communicating Processes, volume 92 of Lecture Notes in Computer Science. Springer Verlag, 1980

22 Guttman A. R-trees: A Dynamic Index Structure for Spatial Searching. In: Proc. of SIGMOD, 1984

23 Chan C, Garofalakis M, Rastogi R. RE-Tree: An Efficient Index Structure for Regular Expressions. In: Proc. of the 28th VLDB conf. Hong Kong, China, 2002

24 Liechti O, Sifer M J, Ichikawa T. Structured graph format: XML metadata for describing web site structure. Computer Networks and ISDN Systems, 1998, 30:11～21

(上接第57页)

设计，并对最终实现的仿真系统 STSNS 作了全面的阐述。在仿真建模过程中，首先用用例(use case)对仿真系统的需求进行了分析，然后建立了 STSNS 的静态模型和动态模型，并用 UML 中相应的图例来描述。最后给出了 STSNS 的详细实现技术。然后，分别采用不同的通信模型对基于时空的路由算法进行了仿真，从仿真结果来看，该算法能够适应卫星网络的特点，对用户提供较好的通信质量，具有较短的时延。

参 考 文 献

1 李军. 卫星网上的动态路由:[南京邮电学院硕士论文]. 2003,3

2 Booch G, Rumbaugh J, Jocobson I. The Unified Modeling Language User Guide. Addison Wesley Longman Inc, 1999

3 Jocobson I, Booch G, Rumbaugh J. The Unified Software Development Process. Addison Wesley Longman Inc, 1999

4 Booch G, Jacobson I, Rumbaugh J. UML Notation Guide (Version1.1). Rational Corporation, Santa Clara, 1997

5 Fowler, Martin, Kendall S. UML Distilled: Applying the Stand Object Modeling Language, Reading, MA: Addison-Wesley, 1997

6 Richter J. Programming Applications for Microsoft Windows, Fourth Edition, Microsoft Press, 1999

7 Echel B. Thinking in C++, Prentice Hall, 1995

8 MSDN Library-July 2001, Microsoft Corp., 2001