基于相关性语义的高效 XML Twig 查询处理方法

朱金清[1] 王伟[1] 周军锋[1,2] 孟小峰[1]

[1] (中国人民大学信息学院 北京 100872)

[2] (燕山大学计算机科学与工程系 秦皇岛 066004)

(zhujinqing@ruc.edu.cn)

摘要 作为互联网上数据的表示和交换事实上的标准, XML 已经广泛应用于许多领域.虽然关键字查询方法容易使用,但其表达能力有限;而结构化查询语言在用户不了解模式信息的前提下无法正确地表达查询.提出一种基于相关性(related)语义来扩充 XPath 的表达能力,使得用户在不了解文档模式的情况下可以轻松表达自身的查询请求.提出基于 related 语义的 XML Twig 查询处理方法 rTwigStack,可以高效处理包含 related 语义的查询,在此基础上,提出基于 DTD 模式的优化算法 rTwigStack+来提高查询效率.通过实验根据不同的性能指标对本文提出的查询算法进行了实验验证.

关键词 XML 数据库, 查询处理, 相关性语义, Twig 模式

中图法分类号 TP391

Efficient Processing of XML Twig Pattern Based on Related Semantics

Zhu Jinqing[1], Zhou Junfeng[1,2], Wang Wei[1], Meng Xiaofeng[1]

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

[2] (Department of Computer Science and Engineering, Yanshan University, Qinhuangdao 066004)

Abstract As a de facto standard for information representation and exchange over the internet, XML has been used extensively in many applications. Though keyword search method can be used easily, it possesses the inherit feature of limited expressive capability. Structured query language has the powerful expressive ability, however, users must have a full understanding of the underneath schema information. We propose an extension to XPath by introducing a novel related semantics, which can help user to express their query requirements even if they are not familiar with the schema knowledge. We then propose an efficient algorithm, rTwigStack, to process a twig query involving related axis, and an optimized algorithm, rTwigStack+, based on DTD schema, to improve the efficiency. The experimental results show that our method can process a twig query with related axis efficiently according to different metrics.

Keywords XML database, query processing, related semantics, twig pattern

1 引言

随着 XML 应用的不断增加, XML 已经成为互联网上数据表示和交换事实上的标准,许多结构化或者半结构化的数据都以 XML 格式表示和传输.为了从大量的 XML 数据中找到感兴趣的信息,可以通过已有的关键字查询[1-3]或者结构化查询(如 XPath[4]、XQuery[5])方式获取需要的信息.

虽然关键字查询简单易用,但其有限的表达能力导致查询结果中包含大量与用户期待不一致的结果,而从大量结果中识别感兴趣的结果对用户来说几乎是不可完成的任务.结构化查询方法虽然可以准确表达查询需求,但前提是用户必须掌握文档的组织结构.而 XML 文档组织的灵活性导致用户难以完全掌握具体的文档结构,具体表现在：

(1) **XML 文档结构的复杂性.**通过文献[6]可知,XMark[7]数据集的 schema 中包含 327 个元素,仅了解 327 个元素对于普通用户来说已经是几乎不可能完成的任务,更不用说掌握这些元素之间

收稿日期:2008-05-30
基金项目:国家自然科学基金项目(课题号:60573091); 国家 863 计划(课题号:2007AA01Z155); 教育部新世纪优秀人才支持计划.

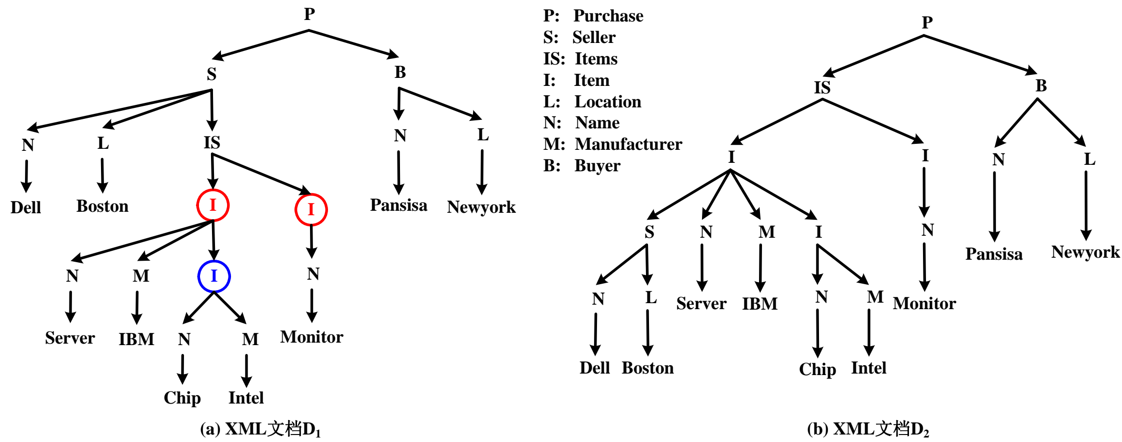

图 1. XML文档示例

的组织关系.

(2) **信息的对称性和 XML 文档组织结构的不对称性.**XML 数据组织的灵活性导致相同数据可能以不同方式进行组织,如图 1(a)和(b)所示,尽管语义相同,但用户必须针对不同文档构造不同查询.

(3) **XML 文档结构的不断演变性.**schema 信息可能随着企业业务的发展存在不断演变的现象,进而导致数据以不同形式进行组织,同样会导致查询不再适用的问题.

以上问题的存在使得普通用户在使用结构化语言表达自身查询请求时,面临无法有效获取信息的尴尬境地.

图 1 (a)中的文档 D_1 描述了一些购买信息.假设用户想知道 Dell 公司销售的一些具体的计算机部件,可以通过查询 Q_1: S[N=‘dell’]/IS/I 来获取相应的信息;如果用户不了解 S 和 I 之间的组织关系,可以使用查询 Q_2: S[N=‘dell’]//I 放宽查询处理的范围来获取相应的信息.但是这两种查询处理方式均有其不足之处.对于 Q_1,用户必须知道详细的 schema 信息.然而通过前面的分析我们知道,这么强的约束在实际使用的时候并不十分有效.对于 Q_2,查询结果为中包括了 3 个 item,即图 1(a)中两个红色 item 和一个蓝色 item.然而实际情况是 Dell 公司销售的 item 其实只有两个,即红色圆圈标识的两个 item,所以返回蓝色圆圈标识的 item 与用户的查询初衷并不相符.

为了克服已有方法在**用户不了解文档具体组织结构**的情况下进行查询处理时存在的不足,本文通过扩展 XPath 语法提出了一种新的相关性(related)语义.它更关注的是用户需要的数据,并非这些数据在文档中的具体物理组织关系,因此,用户使用 related 语义来表达查询请求,无需了解文档的具体组织形式.对于前面所述的 Q_1 和 Q_2,使用 related 语义,用户只需提交查询 Q: S[N=‘dell’]related::I 即可准确获取所需的数据.

这种扩充的好处有以下三点:(1)如果用户了解文档结构,则其可以准确表达自身查询请求.(2)如果用户不了解文档结构,则其可以通过 related 语义表达查询.(3)如果用户仅了解部分结构,则其可以通过 related 语义和 P-C, A-D 的组合来表达查询.

综上所述,本文的贡献可表述如下:

(1) 提出了一种新的 related 查询语义来扩充已有 XPath 语言的表达能力,这个扩充语义在用户不太了解文档的具体组织结构的情况下为用户提供了简单易用的查询表达方式.

(2) 提出了一种新的高效查询处理算法 rTwigStack,该算法只需扫描一遍元素即可得到所有满足条件的结果;提出了基于 DTD 模式信息的一种优化算法 rTwigStack$^+$提高查询效率.

(3) 通过丰富的实验数据对本文提出的算法根据不同的评价标准进行了验证,实验结果表明,本文所提出的方法可以高效处理包含 related 轴的查询,同时可以高效处理包含已有各种轴的查询.

本文剩余部分组织如下:第二部分主要介绍背景知识及相关工作;第三部分介绍相关性(related)语义及问题定义;第四部分详细介绍查询处理算法;第五部分介绍基于 DTD 模式的优化算法;第六部分通过实验对本文的方法进行了验证;第七部分进行总结.

2 背景及相关工作

XML 文档可以用树模型来表示,其中节点表示文档中的元素、属性和值,边表示元素之间的嵌套关系.图 1(a)和(b)为两个树模型表示的 XML 文档.由于 XML 文档中的元素之间具有物理上的先序关系,为了进行查询处理,需要对所有的元素进行编码来标识不同元素之间的相对位置关系.已有的编码方式主要分为两种,即区间编码[8]和 Dewey Code[9].本文采用区间编码来表示元素,一个区间编码是一个三元组(start, end, level).

```
[26]  RelativePathExpr   ::=  StepExpr ( ("/" | "//" | "~>" ) StepExpr)*
[27]  StepExpr           ::=  FilterExpr | AxisStep
[28]  AxisStep           ::=  (ReverseStep | ForwardStep | RelatedStep) PredicateList
[n1]  RelatedStep        ::=  RelatedAxis NodeTest
[n2]  RelatedAxis        ::=  "Related" "::"
```

图2. 带有Related轴的扩展XPath的EBNF语法

Twig 查询作为 XML 查询处理的核心组件,已有的查询方法如 MPMGJN [8],Stack-Tree-Desc/Anc [10]等都是将 Twig 拆分成两两结构连接的方式进行处理,这将导致大量的无用中间结果.TwigStack 是第一个整体处理 Twig 查询的算法,当只考虑 A-D 关系时,它是 I/O 和 CPU 最优的.其它的方法 [11-14] 都是基于 TwigStack[11] 进行改进的,有其各自的侧重点,$TSGeneric^{+}$ [12] 使用 XR 树进行节点过滤,可以跳过一些与查询无关的节点,TwigStackList [13] 通过缓存一些节点,可以大量减少无用的单 path 结果输出,iTwigStack [14]根据不同标准提出了相应的索引,可以大幅度提高查询求解的效率.

3 related 语义及问题定义

3.1 related 语义

在介绍 related 语义之前,首先需要了解一种对称性语义——samepath.

定义 1:samepath 关系.给定两个元素 u, v, 若 u 和 v 之间满足 samepath 关系, 则 u 是 v 的祖先元素或者 v 是 u 的祖先元素.

从定义 1 不难理解,u 和 v 满足 samepath 关系,则二者位于同一条从根到叶子的路径上.本文用 ‘=>’表示 samepath 关系,它满足交换性,即任意两个节点 u, v 满足 samepath 关系,即 u=>v,那么 v=>u.

定义 2:RelatedStep.一个 RelatedStep 返回一系列从当前节点通过一个指定的轴(related 轴)可达的节点.RelatedStep 由两部分组成:(1)轴,定义求解的方向,(2)节点测试,定义节点选择的标准,即节点类型和名字.返回的节点序列按照文档顺序有序.

定义 3:related 轴(“~>”).related 轴包含一系列数据元素,这些数据元素是当前 context 节点的最邻近的后代或者祖先.给定 context 节点元素 n_1(tag 为 N_1),如果元素 n_2(tag 为 N_2)与 n_1 满足 related 轴关系,则 n_1,n_2 具有祖先后代关系且二者之间的路径上没有 tag 为 N_1 或者 N_2 的元素出现.

XPath 语法本身有 82 条规则[4],将 related 关系当作一种轴添加进来即可以完成对 XPath 语法的扩展.我们只需对第 26 条和第 28 条规则进行修改,再添加上两条规则即可,如图 2 所示.

需要注意的是,**在用户不了解文档具体组织形式的情况下,related 语义不能用已有的各种轴或者它们的组合来等价代替**,有了 related 语义,即使用户不了解文档结构,也可以通过 related 轴来获取需要的数据.例如,对于图 1(a)的文档 D_1,为了找 Dell 公司销售的 item,可以通过提交查询即可:S[N=‘dell’]~>I.

3.2 问题定义

本文主要解决包含 A-D、P-C 和 related 关系的 Twig 查询匹配问题,这个 Twig 查询可以表为 $P^{[/,\ //,\ \sim>]}$,其中“/”表示 P-C 关系,“//”表示 A-D 关系,而“~>”表示 related 关系.本文的问题可定义如下:

给定包含 related 轴的查询 Q 和相应的文档 D,从 D 中找出满足 Q 约束关系的所有数据元素,其中:(1) 满足条件的元素必须满足 Q 中查询节点的谓词约束,(2) 数据元素之间的关系必须满足相应查询节点之间的结构约束关系(P-C,A-D 或者 related).其匹配的结果可以用一个 n 元组来表示(e_1, e_2, e_3, …, e_n)来表示,这里 n 表示 Q 中的查询节点总数.

4 related 语义关系的问题分析

4.1 含 related 轴的 Twig 匹配的难点

对含有 related 轴的 Twig 处理却有它固有的挑战:

(1) related 轴的对称性

对于图 3 所示的查询 A~>B~>C,图中仅列出了其可能对应的具有 A-D 关系的具体查询,P-C 关系的查询限于篇幅没有列出.已有方法只能处理某个具体的查询,而不是同时处理所有的查询.

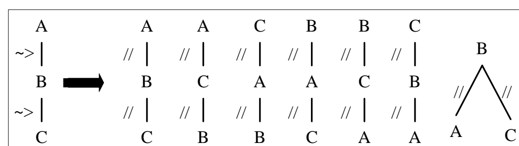

图3. 包含related轴的查询与普通twig查询的对应关系

(2) related 轴与 A-D 或 P-C 轴不可相互表示

当用户不了解具体的文档组织结构或者文档中元素之间含有递归嵌套的情况下,related 轴无法用已有的 P-C 或者 A-D 及其它们的组合方式进行替代.

4.2 扩展解决方案 *SE* (Solution Extension)

由于 related 关系是 samepath 关系的一种特殊形式,这里基于 samepath 关系定义扩展解决方案 *SE*.

定义4: 对于一个给定的查询 Q,如果 Q 中的一个节点 q 有 SE(记作 hasSE(q)),那么 q 必须满足下面两个条件中的任一条件：

(1) isLeaf(q) $\wedge$ $C_q \neq$ NULL，或者

(2) 对于任何的 $q' \in$ children(q)

(i) $q//q' \wedge$ hasSE(q') $\wedge$ $C_q//C_{q'}$[1]，或者

(ii) $q \sim> q' \wedge$ ((hasSE(q') $\wedge$ $C_{q'} => C_q$)，或者

(iii) $q \sim> q' \wedge (\exists e \in S_{q'} \wedge e => C_q)$[2]

这里的 SE 的定义与 TwigStack 中的 SE 的定义的区别在于：不但需要检测以 q 为根的查询节点对应的当前元素,而且可能需要检测以 q 为根的查询节点对应的栈中的元素.

5 rTwigStack 算法实现

查询处理算法 rTwigStack 的总体思路与已有的 TwigStack 算法类似.对于给定的查询,算法 rTwigStack 在第一阶段不断调用 getNext 函数以得到具有 SE 的查询节点 q,并对 C_q 进行处理,如果满足入栈条件,将其入栈,在每个元素出栈的时候同时输出与其相关的所有 path.第二阶段对第一阶段得到的所有 path 执行合并操作并得到最终满足条件的解.

5.1 数据结构

(1) Tag 索引

对于一个 XML 文档,通过 SAX 解析之后,将不同 Tag 名称的元素分别存储,从而可以得到每个 Tag 名称对应的 Tag 索引,并按照其中元素的 start 值进行升序.

(2) 栈的组织

在 rTwigStack 算法中,每个查询节点 q 对应一个栈 S_q,栈中元素从栈底到栈顶与 XML 文档中的从根到叶子的顺序一致,top(S_p)用以得到 S_p 的栈顶元素.如果 q 与 parent(q)之间是 related 关系,则 C_q 和 top($S_{\text{parent}(q)}$)满足 samepath 关系,为了找到满足 related 关系的元素,每个栈元素附设两个指针,p_1 和 p_2,分别代表指向 $S_{\text{parent}(q)}$中与 C_q 满足 related 关系的两个元素.

5.2 算法 rTwigStack

第一阶段(1-10 行)重复调用 getNext(第 2 行)得到返回节点 q,如果 q 有 SE(第 3 行),而且它不是根节点(第 4 行),那么将 q 的父节点的栈 $S_{\text{parent}(q)}$中不是 C_q 的祖先的节点弹出栈(第 5 行);然后如果 q 是根节点,或者不是根节点但 q 的父亲节点的栈非空(第 6 行),那么将栈中不是 C_q 的祖先的节点弹出(第 7 行),然后将 C_q 节点入栈(第 8 行),但是此时的入栈与 TwigStack 的不一样,必须保证在 C_q 节点之前的孩子查询节点对应的祖先元素要先处理并考虑入栈等;然后将 C_q 后移指向下一个元素(第 9 行).第二阶段将每条路径对应的 path 进行合并,得到所有满足条件的结果(第 11 行).

算法 1 rTwigStack(Q)

1: **while** (!endAll(Q)) **do**
2: q = getNext(Q);
3: **if** q.hasSE **then**
4: **if** not isRoot(q) **then**
5: cleanStack($S_{\text{parent}(q)}$, C_q);
6: **if** isRoot(q) or (not empty($S_{\text{parent}(q)}$)) **then**
7: cleanStack(S_q, C_q);
8: pushElementsToStack(S_q, C_q);
9: Advance(C_q);
10: **end while**
11: MergeAllPathSolutions();

Function endAll(q)

1: return $\forall q_i \in subtreeNodes(q): end(C_{qi})$

Procedure cleanStack(S_q, C_p)

1: Pop all elements that are not ancestor of C_p from S_q, and output all related path solutions with the popped elements

Procedure push(S_q, C_q)

1: Push C_q to the Stack S_q; and if q is not the root, modify related pointers.

Procedure pushElementsToStack(S_q, C_q)

1: **if** isLeaf(q)=TRUE **then**
2: push(S_q, C_q);
3: return;
4: **for** $p \in$ children(q) **do**
5: **while** p.start < q.start **do**
6: bPushLater = true;
7: r = getNext(p);
8: **if** r.hasSE **then**
9: cleanStack($S_{parent(r)}$,C_r);
10: **if** r = p **then**
11: **if** (not empty(S_q) or (C_p.end>C_q.end)) **then**
12: cleanStack(S_p,C_p);
13: pushElementsToStack(S_p, C_p);
14: **else if** (not empty(S_q)) **then**
15: cleanStack(S_r,C_r);
16: pushElementsToStack(S_r, C_r);
17: Advance(C_r);
18: **end while**
19: **end for**
20: push(S_q, C_q);
21: **if** bPushLater **then**
22: Modify related pointers

下面介绍 getNext 算法,如果 q 是叶子节点,那么直接返回即可(第 1 行);如果不是叶子节点,对于每个孩子节点 p(第 2 行)递归调用 getNext,如果 p'与 p 不等,直接返回 p'(第 4 行),如果 p'与 p 相等但 p'没有 SE,那么直接返回 p'(第 5 行).如果验证了所有孩子都有 SE 之后,需要检查这个 q 是否有 SE(7-19 行).$n_{\min}$ 与 $n_{\max}$ 都是用来检查 A-D 关系的(7-8 行),C_q 首先应该跳到

---

[1] $q//q'$ 表示 q 和 q' 满足 A-D 关系,$C_q//C_{q'}$ 表示 C_q 是 $C_{q'}$的祖先.

[2] $q \sim> q'$表示 q 和 q' 具有 related 关系，$e \sim> C_q$ 表示 e 和 C_q 之间满足 related 关系。

C_{nmax} 的祖先位置(9-11 行),如果 C_{nmin} 在 C_q 之前,返回 n_{min}(第 12 行).然后对所有的 related 边(第 13 行),如果 C_p 在 C_q 之前,那么返回 p(第 14 行),如果 C_p 在 C_q 之后且 S_p 中没有与 C_q 满足 samepath 的元素,那么 q.hasSE 为 false 并返回 q(15-17 行).其它情况返回 q(19 行).

算法 2 getNext(*q*)

```
1:  if isLeaf(q)=TRUE then return q;
2:  for p ∈ children(q) do
3:      p' = getNext(p);
4:      if p' ≠ p then return p';
5:      if not p'.hasSE then return p';
6:  end for
7:  n_min=minarg_p{C_p.start | q//p};
8:  n_max=maxarg_p{C_p.start | q//p};
9:  while (C_q.end < C_nmax.start) do
10:     Advance(C_q);
11: end while
12: if C_q.start > C_nmin.start then return n_min;
13: for p ∈ children(q) and p~>q do
14:     if C_p.end<C_q.start then return p;
15:     if (C_p.start>C_q.end) and (not ∃e ∈ S_p:e=>C_q) then
16:        q.hasSE = false;
17:        return q;
18: end for
19: return q;
```

定理一:对于含有 A-D 边和 related 边的 Twig 查询,算法 rTwigStack 的时间复杂度和所有处理的 tag 索引的大小以及输出解的大小成比例.空间复杂度为 $O(|Q| \cdot H_D)$, 其中$|Q|$表示查询节点的个数, H_D 表示被查询文档的深度.*P-C* 边的处理需考虑 level 信息即可.

6 rTwigStack 算法优化

在 schema 信息可获取的情况下,本文提出一种优化方法 rTwigStack⁺,尽量将原来查询中的 related 边转化成 A-D 边.这样就可以减少查询中的 related 边数量.

6.1 优化规则

假定 schema 图为一个有向图,元素之间的包含关系用有向边来表示.那么给定 A~>B,有三个转化规则:

(1) 如果 schema 图中 A 和 B 之间只有一条路径,假设路径从 A 到 B(或 B 到 A),那么 A~>B 边可以转化为 A//B(或 B//A).

(2) 如果 A 和 B 之间有多条路径,且都是从 A 到 B(或 B 到 A),而且中间不再经过 A 或 B,也就是没有经过 A 或 B 的环,那么 A~>B 边可转化为 A//B(或 B//A).

(3) 如果 schema 图中 A 与 B 之间存在环, ,那么需要转化成包含 P-C 或 A-D 边的若干个查询.

6.2 优化算法

算法 3 为基于 schema 的优化算法 rTwigStack⁺:

算法 3 rTwigStack⁺(*q*)

```
1:  S_Q = optimizeQuery(q);/*利用三个规则进行优化,S_Q
    表示优化之后的查询集合*/
2:  if | S_Q | > 1 then /*转化后有多个查询*/
3:      rTwigStack(q);
4:  else /*"~>"边已变为"//" */
5:      rTwigStack(p') //其中 p' ∈ S_Q
```

7 实验

7.1 实验设置

本实验的硬件环境是 Intel Core 2 主频为 2.00GHz 的 CPU,1G 内存和 120GB 的硬盘,操作系统为 Windows XP.

本文实现了算法 rTwigStack 和 rTwigStack⁺算法.如果在有 schema 的情况下,由于包含 related 边的查询,根据 schema 转化后的查询可能有多个,因此利用已有方法 TwigStack 求解时,所用时间为各个查询求解时间之和.这 3 种算法分别记为 rTS、rTS+和 TS,都使用 VC++ 6.0 实现.实验的数据集为 XMark [7].实验所用查询如表 1 所示,主要分为两组查询:第一组是不包含 related 轴,如 Q1-Q3;第二组包含 related 轴,如 Q4-Q6.

表 1. 实验测试的查询

查询	XPath 表达式	等价查询数量	数据集
Q1	/site/closed_auctions/closed_auction[//description]//person	1	XMark
Q2	/site/people/person/name	1	XMark
Q3	//site//people//person[//name][//age]	1	XMark
Q4	/site~>open_auction~>privacy	1	XMark
Q5	//closed_auction[.//seller]//parlist~>listitem	2	XMark
Q6	//parlist//text~>keyword	5	XMark

本实验主要考虑两个性能指标来对比这三种方法:(1) 运行时间,(2)扫描的元素数量.

7.2 性能比较和分析

对于包括 related 轴的第二组查询,如图 4 所示,对于 Q5 和 Q6,转化后的查询不止一个,可以看出 rTS 和 rTS+所用时间少于 TS,而对 Q4 来说, TS 运行时间少的原因在于优化后只有一个不包含 related 边的查询.

对于不包含 related 轴的第一组查询,从图 5 可以看出,对于不包含 related 轴的查询,rTS 和 rTS+的运行时间相同,而且效率与 TS 没有太大差距.

对于扫描元素数量而言,如果等价转化后的查询只有一个,那么扫描元素的数量一样,如图 6 中的 Q1-Q4;而对于转化后查询有多个的情况,TS 扫描元素数量急剧增加,如图 6 中的 Q5-Q6,因为它需要处理多个查询才能完成查询求解.

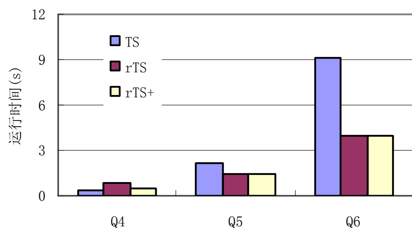

图 4. 200M XMark 的运行时间对比

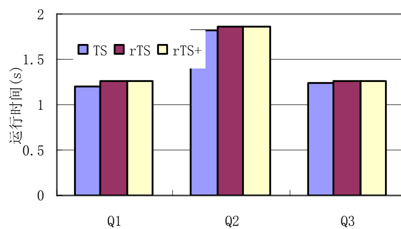

图 5. 200M XMark 上的运行时间

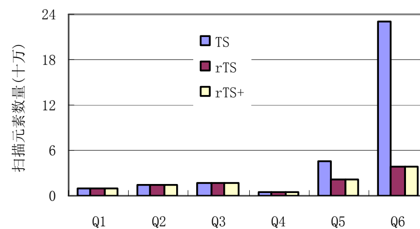

图 6. 6 个查询所需要扫描的元素

综上所述,与 TS 相比,rTS 在处理不包含 related 轴时仍具有很高的效率.如果 schema 已知且根据 schema 转化后的查询只有一个,那么 rTS+的效率比 rTS 高,但比 TS 相对低,如果文档中有循环嵌套的情况,那么 rTS 和 rTS+的性能远高于 TS.

8 结论和展望

本文提出了一种新的 related 查询语义来扩充已有 XPath 语言的表达能力.同时,针对 related 语义,提出了一种高效查询处理算法 rTwigStack 和基于 DTD 提出一种优化算法 rTwigStack+.通过实验可以看出,本文提出的算法不但可以高效处理包含 related 轴的查询,而且可以高效处理不包含 related 轴的查询.本文下一步将考虑将相关性语义扩展到 XML 图上查询.

参考文献

[1] Y. Li, C. Yu, and H.V. Jagadish. Schema-free XQuery. In VLDB, pages 72-84, 2004

[2] S. Cohen, J. Mamou, Y. Kanza and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In VLDB, pages 45-56, 2003

[3] G. Li, J. Feng. J, Wang, and L. Zhou. Effective Keyword Search for Valuable LCAs over XML Documents. In CIKM, pages 31-40, 2007

[4] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007.

[5] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanascu. XQuery 1.0: An XML Query. W3C Recommendation 23 January 2007.

[6] C. Yu and H. V. Jagadish. Schema summarization. In VLDB, pages 319{330, 2006.

[7] XMark. http://monetdb.cwi.nl/xml.

[8] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In Proceedings of SIGMOD 2001, pages 425-436, 2001.

[9] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasun daram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In Proceedings of SIGMOD 2002, pages 204-215, 2002.

[10] Shurug A K, Jagadish H V, Jignesh M P, et al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. ICDE 2002: 141-152.

[11] N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: Optimal XML pattern matching. In Proceedings of SIGMOD 2002, pages 310-321, 2002.

[12] H. Jiang, W. Wang, H. Lu, and J. X. Yu. Holistic twig joins on indexed XML documents. In Proceedings of VLDB 2003, pages 273–284, 2003.

[13] J. Lu, T. Chen, and T. W. Ling. Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In Proceedings of CIKM 2004, pages 533–542, 2004.

[14] T. Chen, J. Lu, and T.W. Ling. On Boosting Holism in XML Twig Pattern Matching using Structural Indexing Techniques. In Proceedings of SIGMOD 2005, pages 455-466, 2005.

朱金清,男,1984 年生,硕士,研究领域: XML 数据库

王 伟,男,1983 年生,硕士,研究领域: XML 数据库

周军锋,男,1977 年生,博士, 研究领域: XML 数据库

孟小峰,男,1964 年生,教授,博士生导师,主要研究方向为 Web 数据管理、XML 数据库、移动数据管理

联系信息:
朱金清
E-mail: zhujinqing@ruc.edu.cn
中国人民大学品园 4-218 室 100872
手机：13426207220
座机：010-62519440