文章编号:1671-9352(2007)11-0040-05

网络环境下 XPath 查询集的冗余去除

徐义静,张世栋,张群
(山东大学 计算机科学与技术学院,山东 济南 250061)

摘要:网络环境下 XML 数据库查询应用,目前国内外已存在多种优化技术,查询重写,语义缓存等,但在冗余去除方面却缺乏研究。在已有技术的基础上,从减少网络流量的角度改进原 XPath 查询集冗余去除方案,利用 XPath 树模式和 DTD 对查询集在不同 XML 文档结构下冗余度进行评估,并在算法中权衡网络流量和 XPath 查询复杂度,来满足用户需求。

关键词:XPath;查询集;冗余去除;XPath 树模式;DTD
中图分类号:TP311 **文献标志码**:A

Redundancy removal of XPath query set in the network XML database

XU Yi-jing, ZHANG Shi-dong, ZHANG Qun
(School of Computer Science and Technology, Shandong University, Jinan 250061, Shandong, China)

Abstract: At present, there are multiple optimization techniques in the application of querying XML database in the network environment at home and abroad, query rewriting and semantic caching technology, but the research on redundancy removal is poor. Based on the existing work, it improved the original redundancy removal solutions for XPath query set by reducing network traffic by using XPath tree pattern and DTD to evaluate the redundant degree of XPath query set in different XML document structures. In addition, the network traffic and XPath queries complexity were weighed to meet the actual user needs.

Key words: XPath; query set; redundancy removal; XPath tree pattern; DTD

XML 已成为目前信息交换和表达标准,网络方面也存在多种 XML 查询应用,如:客户端向远程数据库所在服务器提交 XML 查询,服务器端查询后返回结果。XML 查询主要是使用 XPath[1],XQuery 等 XML 查询语言进行查询,目前,网络 XML 数据库应用方面国内外已存在大量优化技术,包括查询重写[2,3],中间层使用物化 XML 查询视图来应答查询,减少访问数据库的频率,客户端可使用语义缓存[4]。但上述技术缺乏对冗余去除问题的研究,对于此问题,可对客户端提交的相关查询进行重写优化,减少结果集中可能存在的冗余,最大程度上减少网络流量。本文查询重写结合上述查询重写,语义缓存等优化技术和算法,基于冗余去除问题相关文献对算法进行了改进,已有的算法在减少网络流量方面比较有效,但改进后查询集比原查询集的查询代价要大。算法进行之前利用 DTD 对 XML 文档进行过滤,对查询集的冗余度进行估计,根据评估结果来决定采用哪个查询组,从而在网络流量和查询计算量之间取得一个均衡点。

1 背景知识及启发实例分析

XPath 查询分为简单查询和带谓词分支的复杂查询,前者查询节点在一条查询路径上,不存在谓词

收稿日期:2007-04-30
基金项目:国家自然科学基金资助项目(60673130);教育部科学技术研究重点资助项目(03102);山东省自然科学基金资助项目(Y2004G07,Y2006G29);山东省自然基金资助项目(11150005200327);省重大科技专项资助项目(2004GG4201022);山东省中青年科学家奖励基金资助项目(2005BS01002);山东省科技攻关计划资助项目(2005GG3201088);山东省科学技术发展计划国际合作资助项目(2006GG2201052)
作者简介:徐义静(1984-),女,硕士研究生,研究方向为 XML 数据库. Email:xuyijing@mail.sdu.edu.cn

分支。XPath 路径查询可描述为一棵查询树，称为 XPath 树模式。其中，往往只有一个节点的映射节点是查询需要的结果，其余节点之间的边是对该节点的条件约束. 该节点称为查询的返回节点。从根节点到目标节点之间的路径称为查询的主路径.其中，“/”或“//”的个数表示层次数；孩子多于 1 的节点称为分支节点；如果节点上除了针对元素名的谓词条件外，还定义了针对元素值或元素属性值的谓词条件，称为值谓词节点，谓词条件所在子树称为谓词子树。

XML 数据库网络查询模式中，考虑网络流量及 XML 数据库规模，必要信息才从服务器端传到客户端。用户提交的查询并不能保证结果的流量最优。例如用户向某个存储了应聘者信息的 XML 数据库提交如下 2 个查询，要查询一些满足条件的应聘者资料进行分析：

查询小于 25 岁的应聘者，返回其简历中的基本资料

查询小于 25 岁且教育程度满足“研究生”的应聘者，返回其整个简历信息

对于小于 25 岁且教育程度满足“研究生”的应聘者，其简历基本资料要被发给此客户端 2 遍，数据源整体数据量大时，会极大地增加网络负载流量。有时甚至一个 XPath 查询，结果也会存在冗余，如查询标题中含“XML”关键字的书的章节或子章节。如果一个章节标题中含有“XML”，它有一个子章节标题中也含有“XML”，查询结果中，子章节会被发送 2 次，单独发送一次，包含在父节点中发送一次，也造成了网络流量传输冗余。

对于某个用户，一段时间内往往会针对某方面感兴趣信息进行查询，这些查询会在结构或条件方面相关联，甚至相包含，消除这些查询结果的冗余对网络流量的优化起着重要作用，尤其对查询时网络流量需要计费的用户。

对如何消除这些冗余，本文在文献[5]查询重写和冗余去除算法基础上，分别针对简单 XPath 查询集和带谓词的复杂查询集进行讨论。算法基本思路是计算一组查询内各个查询的交集，交集即是冗余，通过重写 XPath 操作去除。文献[5]中去除冗余的查询集往往比原先的复杂，服务器端要消耗更多时间查询，用户也要消耗时间提取结果，虽然减少了网络流量，但这种耗时操作用户可能不能忍受，本文提出查询集冗余度概念，通过分析冗余度，在网络流量和查询复杂度之间做一个均衡，必要时让用户决定在这 2 者中优先满足时间或者流量。

2 简单 XPath 查询集的冗余去除

本文提出的算法对提交的相关查询进行重写优化，构造查询集的最小查询视图集。对一些查询集，存在两种优化选择，即使用原查询集和重写后查询集 2 种策略。本文使用 XPath 的树模式来辅助基本算法进行判断和优化。在查询之间需要上下文信息支持时，采用附加编码来存储上下文信息，代替复杂的查询计算。

本章用简单 XPath 查询的例子及对应消除冗余方法来阐述网络环境下 XML 查询的优化。基本算法在文献[5]中已有所阐述，采用自动机乘积方法构造出不存在冗余的查询组。图 1 为本章查询所用 XML 文档 books：

```
〈books〉
  〈book〉
    〈author〉Jack Herrington〈/author〉
    〈title〉PHP Hacks〈/title〉
    〈publisher〉O'Reilly〈/publisher〉
    〈section〉
      〈title〉1 Introduction 〈/title〉
      〈section〉〈title〉 background of php〈/title〉〈/section〉
    〈/section〉
  〈/book〉
  〈book〉
    〈author〉Jack Herrington〈/author〉
    〈title〉Podcasting Hacks〈/title〉
    〈publisher〉O'Reilly〈/publisher〉
    〈section〉
      〈title〉1 Introduction 〈/title〉
      〈section〉〈title〉 background 〈/title〉〈/section〉
    〈/section〉
  〈/book〉
〈/books〉
```

图 1 books 文档

Fig.1 Document of books

简单路径 XPath 查询集的冗余去除处理（只考虑树模式中从根节点到返回节点路径中不包含谓词子树的查询树模式，只有查询主路径），分为以下几种类型：不带递归的 XPath 查询集的冗余去除，带递归的 XPath 查询集的冗余去除。

2.1 不带递归的 XPath 查询集的冗余去除

例 1 对于如下查询集：$Q1$:/book/title，$Q2$:/book/*，$Q2$ 包含 $Q1$，若客户端提交这 2 个查询，服务器将 $Q1$ 和 $Q2$ 的结果同时发送给客户，总体上，$Q1$ 被发送 2 次，造成网络传输冗余。改进是将交集

$Q1$ 去除,客户端只提交 $Q2$,服务器进行查询后返回结果 Q(本文所有例子均假设返回的结果都存在于以〈Q〉……〈/Q〉为根节点的 XML 片断中),客户端收到 Q 后,进行如下操作即可得到原查询结果:$Q1 \leftarrow Q$/title,$Q2 \leftarrow Q$。

例 2　对于如下查询集:$Q3$:/book/section/title,$Q4$:/book/*/title,$Q4$ 包含 $Q3$,但返回节点 title 的父节点所在层存在包含关系,如果只查询 $Q4$,客户端接收后,缺少 title 父节点上下文信息,不能提取 $Q3$,从 $Q4$ 中减去 $Q3$,提交查询:$Q3$:/book/section/title,Q:/book/{section}'/title,其中{section}'表示{section}的补。

XPath 不支持节点补操作,查询可用 not 函数实现,上例用/book/*[not(self:: section)]/title。提交的查询中,$Q3 \cup Q = Q4$,去除了冗余 $Q3$,得到最小查询集,在客户端提取结果:$Q3 \leftarrow Q3$,$Q4 \leftarrow Q \cup Q3$。

归结上述类型规律,对于 2 个 XPath 简单路径查询,若一个是另一个的子查询,即存在路径包含,且查询层数相同,则可只提交较大查询,在客户端通过操作获得子查询结果。对于 XPath 路径包含测试,可使用 XPath 包含测试算法进行,文献[3]已经指出,当 XPath 支持{//,/,*,[]},包含测试能在多项式时间内实现。

该类型可以扩充到不存在包含关系的查询类型,经验证,此类型冗余去除过程可归结为:

(1) 查询间主路径层数相同的查询集的冗余去除。对于 $Q1, Q2 \cdots\cdots Qn$,n 个层次相等的查询,其重写后的最小查询视图集合 $V(S)$为:设 $S \subseteq \{1,2,3,\cdots\cdots n\}$,$S \neq \Phi$,$V(S) = \bigcap Qi - \bigcup Qj$,其中 $i \in S$,$j \in \{1,2,3,\cdots\cdots n\} - S$,上式逻辑含义:对集合中每个查询 Q,将 Q 与其余几个查询的交集加上 Q 去掉与其它查询交集后的部分,得出最小查询集,保证交集部分在结果中出现一次,总体上结果最小。得到 $V(S)$后,可得原查询结果 $Qi \leftarrow V(S)/*$。

(2) 前期冗余度判断。对主路径元素完全不同的 2 个查询 $Q1, Q2$,其本身 $Q1 \cap Q2 = \Phi$,而 $(Q1 - Q2) \cup (Q2 - Q1) = Q1 \cup Q2$,上面公式并没有消除冗余,用户根据公式所得查询与原查询结果占用网络流量相同,只是增加了服务器端查询复杂性,所以在查询之前对要提交的查询集进行过滤判断很重要,这种判断将在下面进行说明。

2.2　带递归的 XPath 查询集的冗余去除

例 3　对于单个 XPath 查询 $Q7$://section,如果所要进行查询的 XML 文档如图 1,存在递归结构(即 section 节点存在后代子孙节点元素 section),则结果冗余,既返回以 section 祖先节点为根的子树,又会返回以 section 后代节点为根的子树,前者包含后者,后者会被发送多次。改进是提交查询 Q://section-//section//*,即只保留最上层的 section,将以子孙 section 节点为根的分支去掉。客户端接收结果后,进行如下操作,从结果集中获得所有 section 分支:$Q7 \leftarrow Q$//section。

而对于"//"查询(//a)后面还存在多个"/"查询(/b/a/b),客户端提取结果时,要考虑"/"查询所在节点的获取(由于查询比较复杂,不以图 1 的 books 文档为例进行分析)。

例 4　$Q8$://$a/b/a/b$,客户端要提交查询来去掉以 $a/b/a/b$ 分支所在的子树所造成的冗余 Q://$a/b/a/b$-//$a/b/a/b$//*,在客户端,不能只是取如下查询结果:$Q8 \leftarrow Q$//$a/b/a/b$.因为对于 Q 中某个 //$a/b/a/b$ 开始的子树,若其分支包含/a/b,例//$a/b/a/b/a/b$,则此子树的第二层节点 b 也满足查询要求(其下路径中含有/$a/b/a/b$),同理,$b/a/b$ 分支也满足,所以以下 2 个查询结果是原先查询的组成部分(即 $Q8$ 由 3 部分组成):$Q8 \leftarrow Q/b$,$Q8 \leftarrow Q/b/a/b$。经验证,带有递归的查询集类型冗余去除计算过程归结为:带递归的 XPath 查询的冗余去除。

(1) 普通递归类查询(以"/"开始):/$P1$//…//Pm。客户端要提交 Q:(/$P1$//…//Pm) - (/$P1$//…//Pm//*) 并通过 Q//* 得到查询结果。

(2) 对以"//"开始的递归类查询//$P1$//…//Pm,定义子查询序列集 $S = \{*/Pm(1, k-1), */*/Pm(1, k-2), \cdots */\cdots/*/Pm(1, 2)\}$,$k$ 是 Pm 的长度,$Pm(i, j)$是 Pm 从位置 i 到 j 的子查询序列,提交与 1 相同的查询,客户端通过以下操作来得查询结果:每个 $T \subseteq S$,对应查询结果:$V(T) = (/P1//\cdots// (Pm \cap p - \cup q)) - (/P1//\cdots//Pm//*)$,$p \in T$,$q \in S - T$。$S$ 每个查询 p 和 Pm 有相同长度 k,说明这些查询结果不可能是其他结果的子元素,所以使用 $\cap$ 和 $-$。

3　算法改进

对于分支模式这种比较复杂的查询模式,目前受到越来越多的重视,因此应该对带谓词的复杂查询加以考虑,下面对有复杂谓词子树的 XPath 路径表达式进行分析和优化重写。

文献[5]算法(暂称为最小化视图集算法)的不足是提交查询前,未对文档结构特点进行分析,以确

定提交的查询集是否存在冗余,不存在冗余时重写没有减少冗余,还增加了服务器端查询复杂度。本扩展算法的改进是在进行查询集优化重写之前,先判断查询集结果是否存在冗余,存在冗余才进行改造,考虑了查询集的冗余度。

3.1 查询预处理

最小化视图集算法并没有对要查询的文档进行结构分析,由于改进后的查询操作往往比原先的要复杂,增大了服务器端查询的压力,所以在服务器端利用 XML 文档的 DTD, schema 等对 XML 结构特点进行分析,根据不同结构特点决定是否进行重写优化,提交不同的查询集。

改进算法中,对上述例 3 类型,提交 2 个查询组,组 1 为未重写的原查询 $Q7$,组 2 为重写的查询 Q,服务器端先利用 DTD 判断文档中有无递归 section 元素,若没有就使用原始查询 $Q7$,对于非递归的文档来说,$Q7$ 并不会产生冗余。由于 Q 比 $Q7$ 复杂性高,这种过滤大大减少了服务器端的查询复杂度。操作时,扫描 DTD 树进行判断,针对要进行判断的元素 A 如本例的 section 元素,对 DTD 树(图 2)前序遍历,进行具体判断。

DTD 是 XML 文档模式定义,包含未重复的文档结构,与对应的 XML 文档树相比,结点少得多,图 2 所示 books 文档的 DTD 树,模式相同的一组 XML 使用同一 DTD,比对文档进行复杂查询,扫描整棵文档树,效率要高。

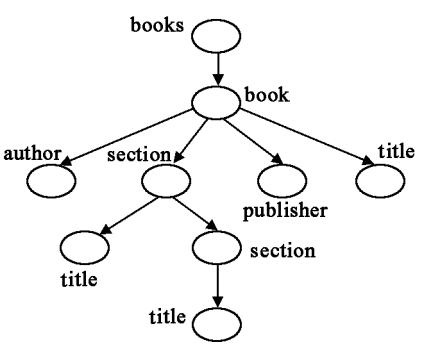

图 2 books 文档对应的 DTD
Fig.2 DTD of document books

3.2 存在谓词子树的路径查询集的冗余去除

本节对谓词包含子路径的情况进行讨论。没使用原算法计算公式,而是基于 XPath 树模式,针对树模式特点讨论.重写优化的关键是先处理主路径,再处理谓词子树,主路径满足上一章中路径包含类型的,可以对其按上面的算法进行处理,主路径完全不同的,不会存在冗余,下面考虑不同谓词子树的情况。查询重写前,先对主路径相似度进行判断,利用 XPath 树模式,对 2 个 XPath 查询及树模式(图 3)$Q9$://$a[c/d]/b/e$, $Q10$://$a[g/l]/b/f$ 进行冗余去除。其中, $Q9$, $Q10$ 为树模式,黑色节点为返回节点,只考虑主路径,即$/a/b/e$ 与$/a/b/f$,按照树的操作对其进行特殊合并操作,将 2 个树模式的相同部分合并,其他仍按原来分支,得新树模式结构 Q,查询时只考虑 Q 中未分支部分,即有一个子节点的路径($/a/b$ 路径)为相似主路径的结果。

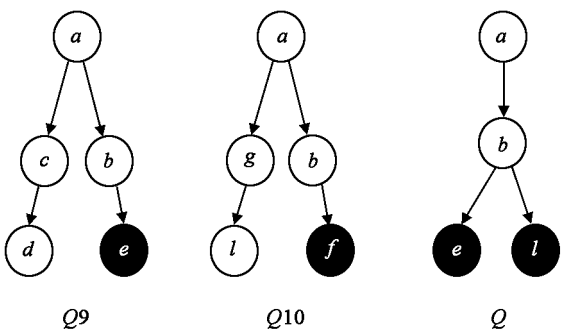

图 3 XPath 查询 $Q9$, $Q10$ 的树模式及合并后的树模式 Q
Fig.3 Tree pattern of XPath query $Q9$, $Q10$ and the united tree pattern Q

例 5 对于 2 个查询 $Q11$://$a[c/d]/b/e$, $Q12$://$a[g/l]/b/e$ 考虑其树模式如图 4 所示。

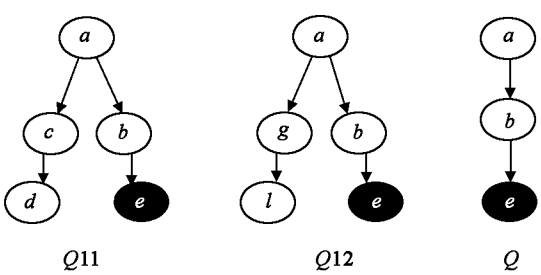

图 4 XPath 查询 $Q11$, $Q12$ 的树模式及合并后的树模式 Q
Fig.4 Tree pattern of XPath query $Q11$, $Q12$ and the united tree pattern Q

从图 4 合并后的树模式 Q 判断出 2 个树模式主路径是相同的,均是从根节点 a 到返回节点 e。对这类主路径相同的查询组,改进方法是对 2 个查询的谓词子树进行合并,即提交查询 Q://$a[c/d|g/l]/b/e$。

冗余去除说明:将 a 原先的 2 个谓词路径合并成路径并集$[c/d|g/l]$,避免既满足$[c/d]$又满足$[g/l]$的节点在结果中的 2 次出现,就减少了相应后代节点 e 的子树,去除了冗余。且服务器查询时,只扫描一遍文档树即可得查询结果,客户端只要极少的操作即可,大大减少了查询时间和流量。

结果减少了冗余,但缺少 e 的祖先节点 a 的信息,所以在返回的 XML 片断中的根节点 e 上加入附加编码,如编码 c 表示满足$[c/d]$的 a, g 表示满足$[g/l]$的 a,编码可作为根节点的一个属性 PRE。客

户端进行如下操作,获得查询结果:$Q11$ <—Q 中根节点 PRE 属性为编码 c 和 $Q12$ <—选择 Q 中根节点 PRE 属性为编码 g 的分析查询特点,2 个谓词分支均存在于节点 a 下,不同特点文档冗余度不同。优化方案是客户端提交原查询组和重写的查询组,服务器端查询前,根据 DTD 判断子树 c/d 与 g/l 是否同在一个节点 a 下,如图 5 所示。

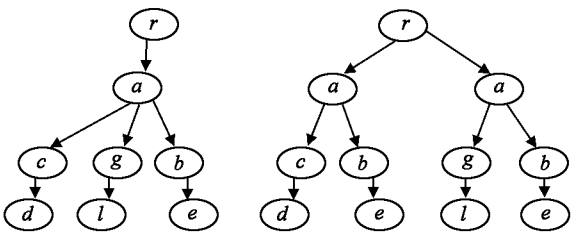

图 5　针对查询 $Q11$, $Q12$ 的不同特点文档的 DTD 图
Fig.5　Different documents' DTD for XPath query $Q11$, $Q12$

左图,2 个谓词子树在同一节点 a 下,则满足条件的 2 个查询的 e 节点是同一节点,结果存在冗余,这时按照新查询组查询,右图返回节点 e 在不同的 2 个子树上,这时 2 个查询的结果不会存在重复,可以按原查询集进行,2 端都可以减少计算量。文档 DTD 树采用前缀编码,对节点 c 与 g,若编码前缀和 a 编码相同且 2 节点编码位数相同,可判断 2 节点是否兄弟节点,从而判断子树 c/d 与 g/l 是否同在一个节点 a 下。

4　实验结果及分析

实验环境为 2 台 P IV CPU 3.0 GHz, 1 G 内存, 160 G 硬盘的 PC 机上,一台为服务器,存放 XML 数据库,提供查询接口,另一台为客户端,进行查询并对查询结果做一定处理。采用 xmark 合成 XML 数据集,使用 Benchmark 数据生成器产生 160 M 左右的数据集。实验目的是证明冗余去除后的查询集的查询结果比原先的结果所占用的网络流量少,以例 2 类型的查询集优化为例,针对 xmark 的数据模式选取如下查询集:

$Q1$:/site/region/namerica/item;　$Q2$:/site/region/europe/item;　$Q3$:/site/region/ * /item/name　$Q4$:/site/region/ * /item/descript。

使用本文算法改进后的查询集为:$Q1$:/site/region/namerica/item;　$Q2$:/site/region/europe/item;　$Q3$:/site/region/{namerica, europe}'/item/name;　$Q4$:/site/region/{namerica, europe}'/item/descript。

其中{namerica, europe}'表示{ namerica, europe }的补。

分别用 2 个查询集查询,监控网络流量及查询时间见表 1,实验结果中,查询时间没增加的基础上,原查询组查询结果约为 100 M,优化后只需 72 M,节省约 28 M(28%)的流量,证明冗余去除算法大大减少了网络流量。

表 1　查询集优化前后时间与流量比较
Table 1　Comparison of time and network flow before and after query set optimization

	查询所需时间/s	网络流量/kb
原查询集	314.54	100 123.09
优化后的查询集	300.08	71 869.06

5　结论及展望

本文在原有查询重写和冗余去除算法基础上,提出了新的 xpath 查询集冗余去除算法,算法中利用 XPath 树模式和 DTD 对查询集在不同 XML 文档结构下冗余度进行评估。经实验分析,对简单 XPath 查询集和带谓词的复杂查询集两种类型查询集,算法都提高了查询效率。算法还考虑了 XPath 查询中存在谓词子树的情况下的处理。

未来的研究主要考虑可以通过研究 XPath 查询集内各个 XPath 之间的关联来快速推断此查询组是否存在冗余。

参考文献:

[1] CLARK J, DEROSE S. XML path language(XPath). W3C recommendation[DB/OL].(1999-11-16)[2007-04-15]. http://www.w3.org/TR/XPath.

[2] GAO Jun, YANG Dongqing, WANG Tengjiao. Efficient XML query rewriting over the multiple XML views[C/OL]// Porc. of the 17th Data Engineering Workshop of IEICE (DEWS2006). Ginowan: ACM Press, 2006. http://www.ieice.org/~de/DEWS/DEWS2006/doc/4A-v2.pdf.

[3] 高军,唐世渭,杨冬青,等. 数据集成中 XML 查询语义重写[J]. 计算机研究与发展, 2002, 39(4):435-442.

[4] 塔娜,冯建华,李国良. 纯 XML 数据库语义缓存综述[J].计算机应用, 2006, 26(12):2977-2981.

[5] Keishi Tajima, Yoshiki Fukui. Answering XPath queries over networks by sending minimal views[C]// Proceedings of the 30th International Conference on Very Large Databases. Toronto, Canada: Morgan Kaufmann, 2004: 48-59.

(编辑:孙培芹)