ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.4, April 2007, pp.893−904
DOI: 10.1360/jos180893

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

针对 XML 流数据的复杂 Twig Pattern 查询处理*

杨卫东+, 王清明, 施伯乐

(复旦大学 计算机与信息技术系,上海 200433)

Complex Twig Pattern Query Processing over XML Streams

YANG Wei-Dong+, WANG Qing-Ming, SHI Bai-Le

(Department of Computing and Information Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: Phn: +86-21-65642219, Fax: +86-21-65642219, E-mail: wdyang@fudan.edu.cn, http://www.cit.fudan.edu.cn

Yang WD, Wang QM, Shi BL. Complex Twig Pattern query processing over XML streams. *Journal of Software*, 2007,18(4):893−904. http://www.jos.org.cn/1000-9825/18/893.htm

Abstract: The problem of processing streaming XML data is gaining widespread attention from the research community. In this paper, a novel approach for processing complex Twig Pattern with OR-predicates and AND-predicates over XML documents stream is presented. For the improvement of the processing performance of Twig Patterns, all the Twig Patterns are combined into a single prefix query tree that represents such queries by sharing their common prefixes. Its OR-predicates and AND-predicates of a node are represented as a separate abstract syntax tree associated with the node. Consequently, all the Twig Patterns are evaluated in a single, document-order pass over the input document stream for avoiding the interim results produced by the post-processing nested paths of YFilter. Compared with the existing approach, experimental results show that it can significantly improve the performance for matching complex Twig Patterns over XML document stream, especially for large size XML documents. Based on the prior works, the optimization of twig patters under DTD (document type definition) by using structural and constraint information of DTD is also addressed, which is static, namely, it is processed before the runtime of stream processing.

Key words: XML document stream; Xpath; Twig Pattern; query tree; DTD (document type definition)

摘 要: XML 流数据处理在研究领域引起了研究者的广泛兴趣.针对 XML 流数据的、具有嵌套 AND/OR 谓词的复杂 Twig Pattern 查询处理,提出一种新方法.为了提高查询处理性能,将所有 Twig Pattern 合并为一个共享前缀的查询树,其中,AND/OR 谓词被表示为单独的抽象语法树,因而能够以文档顺序、单遍地处理复杂 Twig Pattern 的匹配,并避免了 YFilter 中对嵌套谓词进行后置处理所产生的中间结果.实验结果表明,该方法能够有效改善 Twig Pattern 的处理性能,尤其是在处理大文档的情况下.基于已有的研究工作,讨论如何利用 DTD(document type definition)的结构和约束信息优化 Twig Pattern,即这种优化是在系统运行前进行的预处理.

关键词: XML 文档流;XPath; Twig Pattern;查询树;DTD(document type definition)
中图法分类号: TP311 文献标识码: A

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321905 (国家重点基础研究发展规划(973))

Received 2006-01-11; Accepted 2006-05-11

流数据处理系统与传统数据库管理系统不同.传统数据库管理系统的主要特点是数据持久存储,在某一时刻执行查询并通过稳定查询计划给出精确的回答;而流数据处理系统强调数据在线到达,查询持久存储.在流数据处理系统中,不可能控制数据到达的顺序,将所有到达的数据存储在本地进行管理和查询也是不现实的.流数据处理技术引起研究界的广泛兴趣,具有极其广阔的应用前景,可以应用在传感器网络、位置搜寻、网络监控、金融分析、在线拍卖等诸多领域.

由于 XML 已经成为 Web 上数据交换的标准,用于各种应用和信息源之间的数据交换,处理 XML 流数据的理论和技术目前成为流数据研究领域中的一个热点.XML 流数据处理系统通常运行在 Web 上,其上的用户会迅速增加到十万、百万级的数量.用户查询通常用 XPath[1]语言表示.由于一个用户可以提交若干查询,查询的数量更是十分巨大.XML 流数据处理研究的一个关键挑战是如何同时有效处理大量来自用户的查询并及时将结果返回给用户.

XML Twig 查询本质上是具有针对 XML 文档结构和内容的选择谓词的查询.将一组 Twig Pattern 与随时到达的 XML 文档进行匹配是 XML 流数据处理的核心操作.我们发现,对于具有嵌套 AND/OR 谓词的 Twig Pattern 的处理,目前还没有十分有效的处理方法.然而,实际应用中的一个查询,通常同时包含逻辑 OR 和逻辑 AND 谓词.

例如,

Q=/dblp/paper[title='XML Stream' or (year=2006 and conf='VLDB')]//author.

它要求选择 paper 的 author,其发表过 title 为 XML Stream 的文章或者在 2006 年的 VLDB 上发表过文章.这种查询,我们称为具有 AND/OR 谓词的复杂 Twig Pattern.

针对 XML 流处理系统,目前所采用的主要方法是基于自动机的方法[2−8];其他的有基于索引的方法[9]、基于 Bloom-Filter 的方法[10]、Fist 方法[11]等.基于自动机方法的一个瓶颈是随着查询的复杂性和数量的增加,状态数目会呈指数级增长.YFilter[3]是基于自动机方法的典型代表,具有较强的查询处理能力,该方法目前只实现了具有 AND 谓词的查询(它称为嵌套的 XPath).我们认为,从理论上讲,它也能够用同样的方法处理具有 OR 谓词的查询.但它采用的是后置处理的方法,会产生大量中间结果,不能单遍处理所有查询;尤其是在 AND/OR 谓词数量和嵌套层次增加时,处理性能明显下降.对此,我们提出基于查询树的处理方法,将所有具有 AND/OR 谓词的 Twig Pattern 合成为一棵查询树,并将 AND/OR 谓词单独表示为逻辑抽象语法树.采用自顶向下和自底向上相结合的方式,单遍同时处理所有 Twig Pattern.

利用已知的 DTD,可以有效简化 XPath,从而改善 XML 流处理系统的处理性能.基于已有的研究工作,本文概要讨论了相关优化技术,包括逻辑表达式的短路与共享计算以及基于 DTD 的优化技术.逻辑表达式的短路与共享计算只适用于我们提出的方法.基于 DTD 的优化技术是在系统运行前预先处理的,因此适合于所有用 XPath 表示用户查询的 XML 流处理系统.

本文主要工作包括:

- 提出新的方法处理具有 AND/OR 谓词的复杂 Twig Pattern 查询.在我们的方法中,将 AND/OR 谓词作为单独的抽象语法树来处理,利用提出的基于运行栈的算法,结合自顶向下和自顶向上过程,有效地处理基于 XML 流的复杂 Twig Pattern 查询.
- 将所有 Twig Pattern 组合成单个可共享公共前缀的查询树,从而可以节省查询处理的空间和时间,并以输入的 XML 流顺序单遍处理所有 Twig Pattern.
- 分析影响 XML 流数据处理性能的关键因素以及基于 DTD 的优化技术,实现了基于 DTD 的优化组件.该组件在系统运行前进行预处理,不影响系统运行时性能.

本文第 1 节介绍相关研究工作.第 2 节重点研究针对 XML 流的多个复杂 Twig Pattern 的查询处理,包括 Twig Pattern 的表示、将多个 Twig Pattern 合成单个 Twig Pattern 以及查询处理的方法,逻辑表达式的短路计算;概要讨论如何充分利用 DTD 的结构和语义信息,对一组 Twig Pattern 进行优化.第 3 节给出实验结果与分析.第 4 节给出本文的结论.

1 相关研究工作

由于流数据处理系统所具有的广泛应用前景,以及 XML 已经成为 Web 上数据交换的标准,XML 流数据处理的研究引起了研究者的广泛兴趣.很多研究采用基于自动机的方法处理 XML 流数据[2−8].XFilter[2]首次利用基于有限状态自动机(finite state machine,简称 FSM)的方法过滤 XML 文档.XFilter 对每一个路径查询使用一个单独的 FSM,并在文档处理的过程中,同时运行所有的 FSM.YFilter[3]在 XFilter 的基础上进行了改进:将所有的 XPaht 查询合并成一个单独的非确定有限自动机(non-deterministic finite automaton,简称 NFA),并共享所有查询的公共前缀.YFilter 将 twig patter 视作嵌套路径表达式,并使用查询分解进行处理.在他们的方法中,当一个查询包含嵌套路径时,就被分为主路径和一组扩展路径,每一个扩展路径都用一个相对独立的 NFA 进行处理.对它的处理分为两步:路径匹配和路径匹配结果的后置处理(执行连接操作).针对嵌套路径,YFilter 主要考虑的是具有 AND 谓词的查询,并且,这种后置处理的方式可能会产生大量中间结果,从而影响系统性能.XPush[4]将所有的 XPath 表达式构造为单个定制的确定下推自动机(XPush 机).该方法首先将每一个 XPath 转换为等价的交替有限自动机(alternating finite automaton,简称 AFA).AFA 是一个非确定有限自动机,其中:每一个状态标记为 AND,OR 或者 NOT;然后,在此基础上构造自底向上的 XPush 机.XPush 主要强调原子谓词(基于值的谓词).与其他基于自动机的方法一样,随着 XPath 的增加,状态数目会呈指数级增长.为此,它也采用懒惰构造的方法在运行时构造 XPush 机.然而,这种情况下,当首次发现并计算该状态时,运行代价很高.Green[4]等人将 NFA 转换为确定有限自动机(deterministic finite automaton,简称 DFA),并使用懒惰 DFA 控制状态爆炸带来的运行时负载,以提高处理性能.Dan 等人[10]使用状态机处理 XPath 表达式,将一个 XPath 表达式转换为多个下推自动机构成的网络.这种方法难以处理大量 XPath 查询.文献[8]基于树自动机,采用自顶向下和自底向上的方式处理 XPath 表达式.

其他处理 XML 流的方法主要有基于索引的方法[9]、基于 Bloom Filter 的方法[10]以及 FiST 方法[11].Index-Filter[9]采用基于索引的技术处理 XML 流数据.Index-Filter 利用 XML 文档流的文档标记动态地建立 XML 文档的索引,从而避免处理一部分 XML 文档.与 YFilter 相比,他们通过实验表明:当查询数量相对较小、XML 文档相对较大时,Index-Filter 更有效(在不考虑建立索引所花费的代价的前提下);当查询数量相对较大、XML 文档相对较小时,YFilter 更有效.在 Index-Filter 的方法中,建立索引要花费一定的时间代价.另外,不能单遍处理 XML 文档.Index-Filter 没有考虑对 AND 或 OR 谓词的处理问题.基于 Bloom Filter 的 XML 包过滤器[10]是一种近似查询方法,利用 Bloom Filter,将 XPath 表达式作为字符串,将 XPath 与 XML 包之间的匹配转换为字符串之间的匹配,从而提高查询性能.它只是用来处理简单的 XPath 表达式(不包括谓词,只包括“/”,“//”,“*”,称为 $XP^{\{/,//,*\}}$),并且有一定的失误率.FiST[11]针对 Twig Pattern 提出一种有别于 YFilter 的方法,将一组 Twig Pattern 转换为 prufer 序列,并对一组 Twig Pattern 与 XML 流数据进行整体性(holostic)匹配.FiST 考虑的是具有 AND 谓词的 Twig Pattern,而没有考虑如何处理 OR 谓词.

上面提到的许多研究工作根据各自的方法提出了一些特定优化技术,不能适用于其他方法,在此不再赘述.我们将通用 XML 流数据处理优化技术分为两类:基于部分文档结构的优化和基于 DTD 的优化.

文献[5]使用轻量级的二进制数据结构预先编码 XML 文档的部分结构信息,称为流索引(stream index,简称 SIX),并把其加入 XML 文档流.这样,流处理引擎就可以根据流索引提前决定略去对哪些元素的解析.这种方法的一个缺点是,流处理引擎必须了解数据源端的流索引信息.文献[12]根据同样的思想,但使用不同的技术,用基于预先计算的视图来加速 XML 流数据的处理.该方法的关键是把对视图的回答进行编码,作为 XML 文档流的头.这样,当 XML 文档流到达时,流处理引擎通过解读视图就可以了解相关文档结构的信息,从而优化文档的解析和查询匹配.

在部分应用中,XML 文档具有 DTD.对于这类文档,可以有效利用 DTD 的信息来优化 XPath 表达式,从而优化 XML 数据流的处理.DTD 有两类信息可用于优化 XPath 表达式:结构信息和语义信息.利用结构信息,可以去除 XPath 中的“*”和部分“//”[8,13];利用 DTD 的语义信息[14],可以有效简化 XPath 表达式.已有的许多研究讨论了 XPath 表达式的最小化、等价、包含等问题,本文在此只讨论直接相关的研究工作.文献[13]利用树自动机(表示 DTD)和一般的自动机(表示路径)的乘积去除 XPath 表达式中的“*”,文献[14]利用 DTD 的语义信息给出相关优

化规则,并利用这些优化规则简化 XPath 表达式.这些研究成果同样可以应用在流处理的研究领域中:文献[8]利用树自动机的乘积去除“*”和部分“//”(在 DTD 包含环的部分,“//”不能去除),并将优化结果用于 XML 流数据的处理;文献[15]利用 XML Schema 的语义信息定义相关优化规则并用于 XML 流处理.我们在结合文献[8,14]的基础上,实现了基于 DTD 优化 XPath 的组件.

另外,文献[16]也讨论了具有 AND/OR 谓词 Twig Pattern 的查询处理.由于他们研究的是针对通常 XML 文档而不是 XML 流数据的查询处理,因此,与本文采用的方法也不相同.

2 针对 XML 数据流的多 Twig Pattern 处理

本节主要讨论针对复杂 Twig Pattern 的 XML 流处理问题.问题定义如下:

给定一个包含复杂 Twig Pattern 的查询集合 Q 以及一个 XML 文档 D,找出 $Q'\subseteq Q$,满足对每一个 $q\in Q'$,都匹配文档 D.

第 2.1 节讨论 Twig Pattern 的内部表示;第 2.2 节讨论将所有 Twig Pattern 合并为单个共享前缀的查询树,以及基于运行栈的 Twig Pattern 处理算法:采用自顶向下和自顶向上相结合的方式,单遍处理 XML 文档.对于不含谓词的简单查询,可直接采用自顶向下的方法处理,而对于含有谓词的 Twig Pattern,需要结合自底向上的方法来处理.第 2.3 节讨论相关优化问题.

2.1 具有AND/OR谓词的Twig Pattern的表示

直观地,可将 Twig Pattern 表示为一棵查询树(我们称为通常的查询树),在树中插入相应的 AND 节点和 OR 节点,以表示相应的谓词.例如,对于 XPath 表达式 Q=/dblp/paper[title='XML Stream' or (year=2006 and conf/title='VLDB')]//author,其查询树如图 1 所示.但针对 XML 流数据处理问题,这种表示带来两点不便:树中含有大量 AND 节点和 OR 节点;当合并多个 Twig Pattern 时,AND 节点和 OR 节点难以处理,例如查询/a[b and c]和查询/a/[b or c]的合并.为此,我们将 AND 谓词和 OR 谓词剥离出来,单独表示为相关节点的抽象语法树,称其为紧凑 Twig Pattern 查询树.同样的例子,图 1 的紧凑查询树如图 2 所示.

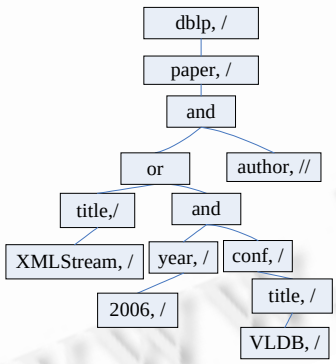

Fig.1 Parse tree of a Twig Pattern

图 1 Twig Pattern 分析树

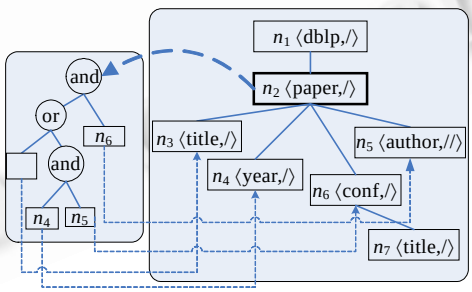

Fig.2 Compact parse tree of a Twig Pattern

图 2 Twig Pattern 紧凑分析树

定义 **1**. 一个紧凑 Twig Pattern 查询树是一棵表示 Twig Pattern 的查询树,查询树中的节点称为查询节点(QNode),每一个 QNode 具有唯一标识(如对于上面例子中的查询 Q,定位步/dblp 标识为 n_1),分为以下两种类型:

- OQNode:不带有谓词的定位步,称为通常查询节点(ordinary query node,简称 OQNode).在紧凑查询树中,OQNode 关联相关信息,包括节点名字(“*”的名字为“*”)和表示父子(“/”)或子孙(“//”)关系的算子,用两元组⟨name,“/” or “//”⟩表示.
- PQNode:带有谓词的定位步称为谓词查询节点(predicate query node,简称 PQNode).谓词查询节点是紧凑查询树中的特殊节点,它通过 AND/OR 逻辑谓词将其子树连接起来.除了节点标识、节点名和“/”或

者“//”以外,还关联一个逻辑表达式.该逻辑表达式在内部表示为抽象语法树.该抽象语法树的每一个叶子节点都维护一个到其对应节点(谓词节点的孩子节点)的引用.

2.2 Twig Pattern与文档流的匹配

定义 1 给出了与通常的具有 AND/OR 谓词的查询树等价的定义.在对 XML 文档流处理的过程中,我们对带有谓词的节点及其子节点,采用自底向上的匹配过程;而对于其他部分,采用自顶向下的匹配过程.为了适应这样的匹配过程,我们对定义 1 进行扩充.

定义 **2**. 谓词子树(predicate sub-tree). 在一个紧凑 Twig Pattern 查询树的表示中,以距离根节点最近的谓词节点为根所形成的子树称为谓词子树.同时扩充相应的节点结构:谓词子树中的 OQNode(叶子节点除外),也关联一个逻辑表达式,该逻辑表达式是只含有一个项,即其孩子节点的标识.而如果 OQNode 是叶子节点,则含有一个逻辑标记,初始为 FALSE,当遇到 CLOSE 事件后,将逻辑标记的值赋为 TRUE.

定义 **3**. 接受节点(accepting node). 在一个紧凑 Twig Pattern 查询树的表示中,距离根节点最近的谓词节点称为该树所表示查询的接受节点.如果在计算过程中,接受节点相关逻辑表达式的值为真,则该文档与查询匹配.对于不带谓词的 XPath 表达式($XP^{\{/,//,*\}}$),通常查询树与紧凑查询树相同.如果一个 XPath 表达式是不带谓词的一般查询($XP^{\{/,//,*\}}$),其接受节点则是其通常查询树的叶子节点.如果在匹配过程中到达接受节点,则该文档与查询匹配.

图 3 给出用于说明匹配过程的 3 个查询的例子,查询的接受节点用加粗的黑框表示.查询 Q_1 是一个不带谓词的查询,n_3 是其接受节点;查询 Q_2 是一个含有 AND/OR 谓词的 Twig Pattern(为简单起见,节点的相关逻辑表达式用文本表示),n_2 是它的接受节点,其相关逻辑表达式是[n_3 and (n_4 or n_5)];n_2 为根的子树是它的谓词子树.由于 n_5 是谓词子树中的 OQNode,它关联有包含其孩子节点的一个逻辑表达式;Q_3 的表示与 Q_2 类似.

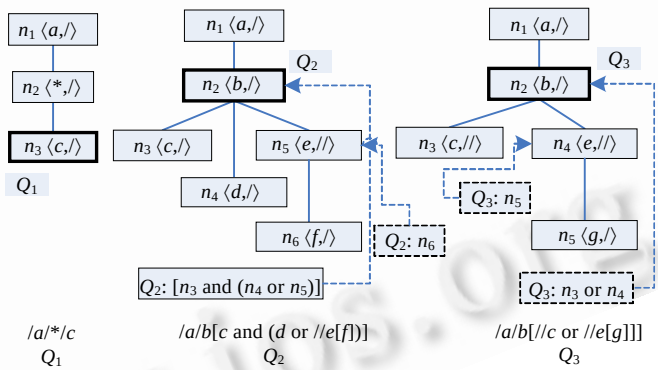

Fig.3 Examples

图 3 例子

XML 流数据处理系统应该能够同时处理大量用户查询.而在来自用户的大量查询中,可能会具有许多共同的部分,共享其共同部分可以节省系统的存储空间和执行时间,对改善系统性能非常重要.为此,我们将所有的 Twig Pattern 合并为一个单一结构,称为共享前缀的紧凑 Twig Pattern 查询树.我们的合并方法类似于 Index-Filter[9],节点名以及算子(“/”或“//”)相同的前缀可以合并.图 4 是图 3 中 3 个查询合并后的共享前缀的紧凑查询树.

在图 4 中,节点 n_4 是 Q_1 的接受节点;节点 n_3 是 Q_2 和 Q_3 的接受节点.这里需要注意的是,节点 n_3 关联有两个逻辑表达式,每一个逻辑表达式与其查询标识(查询标识由 XPath 解析器生成)相关联.

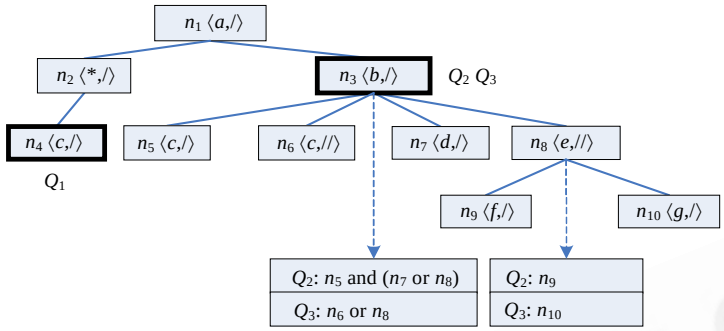

Fig.4 Prefix shared compact query tree

图 4 共享前缀的紧凑查询树

2.2.1 数据结构

每一个查询节点都有一个唯一的标识,并包含下面的基本信息:

- name.表示节点名字的字符串.
- axis.整数值:0 表示“/”,1 表示“//”.
- documentLevel.表示 XML 文档层次的整数.

每一个接受节点关联一组查询标识.如果一个节点是谓词节点或谓词节点的子节点,则对每一个查询(该节点可能被多个查询所共享),关联的信息是⟨QID,status,lExpression⟩.其中:

- QID 是查询标识;
- lExpression:该节点相关的逻辑表达式.逻辑表达式的每一项被其对应的孩子节点引用.如果是谓词子树的叶子节点,则它的逻辑表达式为空;
- status:该状态标记是一个布尔值,表示节点的匹配状态(逻辑表达式的计算结果).其初始值为 FALSE.

2.2.2 节点匹配和查询匹配

当文档到达系统时,通过 XML 解析器产生相应事件,流处理引擎通过回调函数对事件作出反应,将事件翻译为(Name,Type,DocumentLevel)的形式驱动匹配过程.其中:名字是节点测试名;类型是事件类型,包括 StartDocument,EndDocument,OPEN 和 CLOSE;DocumentLevel 通过计算 XML 文档元素的 OPEN 和 CLOSE 标记来计算,文档的根的 DocumentLevel 是 0.

对于一个查询,当遇到的节点是 OQNode 且不属于谓词子树的子节点时,如果事件名、节点名、文档层次相匹配,则一个节点匹配成功.当一个节点包含父子关系算子时,文档层次要求相同;当一个节点包含子孙关系算子时,若节点名相同,则节点匹配成功,这时,忽略文档层次的检查.当查询节点名字是“*”时,节点名字的测试永远返回 TRUE.这是自顶向下的过程,遇到 OPEN 事件时进行计算.

当遇到的节点是 PQNode 或谓词子树中的子节点且基本信息⟨Name,IsChild,DocumentLevel⟩的检查通过时,需要计算它所关联的逻辑表达式并检查状态标记.如果状态标记为真,则节点匹配成功.这是自底向上的过程.状态标记的初始值为 FALSE,当遇到 CLOSE 事件时,对相应逻辑表达式进行计算,并将结果值赋给状态标记.

如果节点匹配发生,则该节点通过,继续下一个节点.如果匹配的节点是接受节点(在此注意:一个节点可能对一个查询是接受节点,对另一个查询不是接受节点;或者对一个查询是 OQNode 节点,而对另一个查询是 PQNode),则该 XML 文档匹配查询.

2.2.3 算 法

算法基于运行栈,分为 OPEN 和 CLOSE 两部分.

OPEN:OPEN 事件通过回调函数调用该句柄,传入事件名字、元素名字和元素的文档层次.

- 对每一个到达的 XML 元素,进行节点测试和文档层次检查;

- 如果节点检查返回 TRUE,则该节点被压入一个运行时栈.若遇到的节点是 PQNode 或谓词子树的子节点,其状态标记的值设为 FALSE;若遇到的节点是一个查询的接受节点且不是 PQNode 节点,则一个查询匹配发生.如果该接受节点是 PQNode,则需要等到遇到 CLOSE 事件时才能判定文档与查询是否匹配.

CLOSE:如果遇到的节点是 OQNode,只是简单地将节点从栈中弹出;如果遇到的节点是 PQNode 或谓词子树的子节点,将执行下列步骤:

- 如果是叶子节点,将其状态标记赋为 TRUE,意味着该节点匹配成功,从运行栈弹出该节点,并将当前栈顶节点(当前栈顶节点是其父节点)中相关的逻辑表达式中对应的项赋值为 TRUE;
- 如果是谓词子树中的中间节点,计算其逻辑表达式(此时,它的孩子节点都已经处理过).如果逻辑表达式的值为 TRUE,意味着该节点匹配,将其状态标记赋值为 TRUE;否则,将其状态标记赋值为 FALSE,从栈中弹出该节点,并将当前栈顶节点的相关逻辑表达式中的对应项赋值为其状态标记的值(TRUE 或 FALSE);
- 如果是一个查询的接受节点,计算其逻辑表达式,并将逻辑表达式的结果赋给状态标记.如果是 TRUE,则文档与该查询匹配;如果是 FALSE,则文档与该查询不匹配.
- 继续上述过程,直到所有 PQNode 得到处理节点为止.

2.2.4 例 子

我们的方法将自顶向下和自底向上的过程结合起来.下面使用例子来说明查询匹配的执行过程.

图 5 使用图 4 的 3 个查询以及一个 XML 片断作为例子,展示了查询匹配的过程.

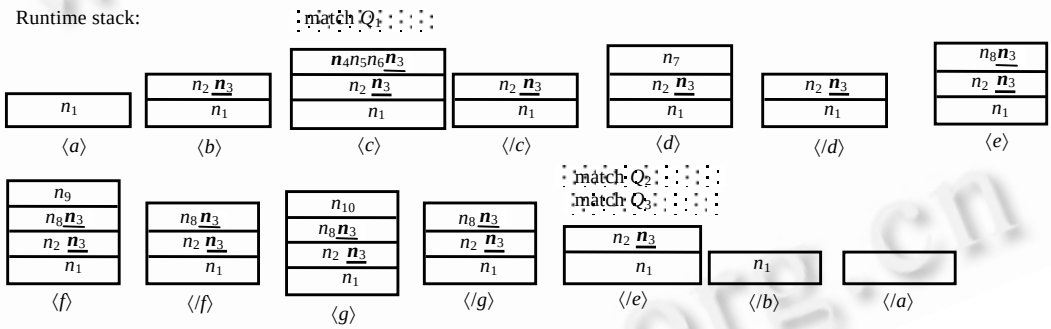

Fig.5 Processing of examples

图 5 查询处理过程

图 5 中,运行栈中的节点对应于图 4 共享前缀紧凑查询树的节点标识.加粗斜体字表示接受节点,下划线节点表示接受节点是 PQNode.当遇到 b 元素的 OPEN 事件时,节点 n_2 和 n_3 被压入栈中.这时,n_3 的状态标记的值是 FALSE.当遇到 c 元素的 OPEN 事件时,n_4,n_5 和 n_6 被压入栈中.由于节点 n_4 是 OQNode 且是查询 Q_1 的接受节点,这时,可以判定文档与查询 Q_1 匹配.节点 n_5 和 n_6 的状态标记的值为 FALSE.当遇到 c 元素的 CLOSE 事件时,将节点 n_5 和 n_6 的状态标记值赋为 TRUE,弹出节点 n_4,n_5 和 n_6(这时,当前栈顶节点是 n_2 和 n_3),并将 n_5 和 n_6 的状态标记值赋给 n_3 中对应的逻辑表达式中的项,即⟨Q_2,FALSE,(TRUE and (n_7 or n_8))⟩和⟨Q_3,FALSE,(TRUE or n_8)⟩.类似地,当遇到 b 元素的 CLOSE 时,计算 b 元素相关的逻辑表达式,并将计算结果赋给对应查询的状态标记,即⟨Q_2,TRUE,(TRUE and (TRUE or TRUE))⟩和⟨Q_3,TRUE,(TRUE or TRUE))⟩.由于 b 元素是 Q_2 和 Q_3 的接受节点,可以判定 Q_2 和 Q_3 与文档匹配.

2.3 优 化

XML 流数据处理系统通常面对大量用户,需要及时处理百万级数量的查询.因此,处理性能是判定系统质量的一个关键指标.对于给定的文档流 D,查询集合 Q,用户集合 U,我们将 XML 流数据处理系统的处理时间定义为

$$T_D=T_{sax}(D)+T_{matc}(D,Q)+T_{dissemination}(D,U),$$

其中:$T_{sax}(D)$表示对文档 D 的解析时间;$T_{matc}(D,Q)$表示文档 D 与查询集合 Q 的匹配时间;$T_{dissemination}(D,U)$表示将匹配的文档 D 分发给用户集 $U'(U'\subseteq U)$所需的时间.

根据已有研究[3,17],除了查询匹配时间外,文档的解析时间和分发文档给用户的时间(信息分发在本文讨论范围之外)对流处理系统的性能同样有较大影响.在此,我们概括讨论两种优化技术:逻辑表达式的短路与共享计算以及基于 DTD 的优化技术.

2.3.1 逻辑表达式的短路计算与共享计算

对于一个逻辑表达式,例如 $E_1=e_1$ and e_2,$E_2=e_3$ or e_4,如果 e_1 为 FALSE,则不用计算整个表达式就可得知 E_1 为 FALSE;如果 e_3 为 TRUE,则可得知 E_2 为真.通过对逻辑表达式适时地进行短路求值计算,可以节省逻辑表达式的计算时间,同样可以略过部分文档的解析,从而也可节省文档解析时间.

针对大量的查询,在共享前缀的谓词查询树中,一个谓词查询节点可能会关联很多的逻辑表达式.在这种情况下,可能会存在许多相同的逻辑表达式子项,如果对这些相同的逻辑表达式子项进行共享——即共享的逻辑表达式子项只存储且只计算一次,不仅可以降低逻辑表达式的空间复杂度,而且可以加速逻辑表达式的计算.为此,我们将逻辑表达式表示成一棵抽象语法树,对公共逻辑表达式子项进行共享处理就是合并多棵语法树中相同的节点或分支.

例如:如图 6 所示,有 Q_1:$n_1/n_2[(n_3$ and $n_4)$ or $n_5]$,Q_2:$n_1/n_2[n_3]/n_4$,Q_3:$n_1/n_2[n_3$ and $n_4]$这样 3 个查询,由它们建立的共享前缀紧凑查询树的简化表示(只用元素名表示相应节点)如图 6 左面部分所示,其中,n_2 为接受节点,关联有这 3 个查询各自的逻辑表达式.不难发现:对 Q_2 和 Q_3 查询来说,逻辑表达式都是 n_3 and n_4,而这也是 Q_1 的逻辑表达式(n_3 and n_4)or n_5 的子项.因此,只需为 n_2 节点建立一棵共享逻辑分支的抽象语法树.

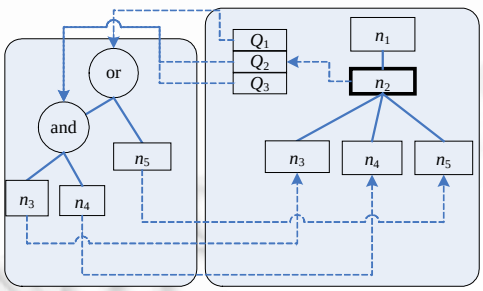

Fig.6 Sharing of common logic expression

图 6 公共逻辑表达式的共享

2.3.2 基于 DTD 的优化技术

利用 DTD 信息可以简化 XML 查询以及降低流处理系统的时间和空间复杂度.

文献[9]用基于树自动机的方法表示 DTD,用一般自动机表示 XPath,然后对两个自动机进行乘积操作去除“*”.文献[8]利用类似的方法讨论了对“//”的去除.我们的优化第 1 步是利用 DTD 的结构信息,采用与参考文献[8]相同的思路,不同的实现方法,简述如下:

步骤 1. 用正则语法树(regular tree grammar,简称 RTG)形式化定义 DTD,并构造 DTD 树,构造 Twig Pattern 的紧凑查询树.

步骤 2. Twig Pattern 路径分解.以 Twig Pattern 紧凑查询树的 PQnode 作为分界点,可以将 Twig Pattern 分

解为若干简单路径,针对每一个简单路径,进行去除“*”和“//”的处理.

步骤 3. 去除“*”.对每一个简单路径,若包含“*”,例如 *a***b*,用其对应 DTD 图中 *a* 到 *b* 的所有可能路径替换 *a*//*b*.

步骤 4. 去除“//”.对每一个简单路径,若包含“//”,例如 *a*//*b*,

a) 若其对应的 DTD 图中包含环时,则“//”不能去除;

b) 否则,用 DTD 图中 *a* 到 *b* 的所有可能路径替换 *a*//*b*;

步骤 5. 重构 Twig Pattern.将简化后的简单 XPath 路径重新合并为 Twig Pattern.

我们的优化第 2 步是充分利用 DTD 的语义信息,预先简化 Twig Pattern.实际上,DTD 中包含丰富的语义信息,见表 1.

Table 1 Constraints of DTD

表 **1** DTD 的约束信息

Constraints	Mean
Optional (?)	An element can have either zero or one sub-element
Only (default)	An element must have one and only one sub-element
Any (*)	An element can have zero, one and more sub-element
At least (+)	An element can have one or more sub-elements
Exclusion (|)	An element “*r*” can have either “*a*” or “*b*” as its sub-element but not both at the same time

为此,我们结合已有工作[16,18]实现基于 DTD 的优化,对 XPath 表达式进行预处理.优化规则如下:

父子规则:如果一个元素一定有一个子元素作为它的孩子,则它与该子元素满足父子规则(父子规则可由约束 only 和 at least 得到).

子孙规则:如果一个元素一定有一个子元素作为它的后代,则它与该元素满足子孙规则(子孙规则可由父子规则推导出).

兄弟规则:如果一个元素有一个子元素作为其孩子,那么,一定有该子元素的兄弟(另外一个子元素)作为其孩子.

排斥规则:如果一个元素有一个子元素作为其孩子,那么,一定不会有另外的子元素作为其孩子.

$n\rightarrow(b,(c^*,d)|e))$

$n\rightarrow(k)$

$n\rightarrow(f|g)$

$n\rightarrow(h?,i)$

$n\rightarrow(i)$

例如:假定给出以上 DTD,根据父子规则,*a*[*b*][*e*/*i*]可以简化为 *a*[*e*];根据子孙规则,*a*[//*i*]/*b* 可简化为 *a*/*b*;根据兄弟规则,*a*/*d*[*h*][*j*]可简化为 *a*/*d*[*h*];根据排斥规则,*a*/*c*[*f*][*g*],*a*/*c*[*f*]/*g*,*a*[*d*][*e*],*a*[*d*]/*e* 都将返回空集.

利用 DTD 优化 XPath 表达式,可以有效改善流处理系统的性能:在现有的流处理方法中,“//”和“*”的出现都会增加系统处理的复杂性.例如:NFA 状态表的大小随着“//”的数目呈指数级增长[3];DFA 的状态数目随着“//”数目的增长而呈指数级增长[5].同样,它们也会增加对 XML 文档的解析时间.通过去除“*”和部分去除“//”以及利用 DTD 语义信息简化 Twig Pattern,可以减少系统查询处理的空间和时间复杂度,略去对一些不匹配元素的解析,增加多个查询之间的共享程度,提前去除不符合 DTD 的 XPath 表达式.

3 实 验

类似于 YFilter,我们用 Java 实现了系统并与之进行实验比较.系统运行的环境是 Eclipse3.1,机器的主频是 2.7G,内存为 512M.构成实验的组件包括:文档生成器、DTD 解析器、XPath 生成器、基于事件的 XML 解析器、LeoXSQ 的 XML 流数据处理引擎以及 YFilter 系统.YFilter 用 Java 实现,其源码可从 Berkeley 大学的网站 http://yfilter.cs.berkeley.edu 中获得.我们使用 IBM 的文档生成工具[19]生成文档;DTD 解析器来自 WUTKA[18];

XPath 解析器使用 JXPath[20],使用的 DTD 来自于 Xmark[21],使用的文档解析器来自于文献[22].

下面我们用几个指标进行实验比较和分析:(1) Twig Patterns 的总数量;(2) Twig Patterns 的分支数目;(3) 输入文档的大小.实验分为两类:针对小 XML 文档(10K,30K,50K)的实验和针对大 XML 文档(500K,1M,2M)的实验.由于 YFilter 不支持 OR 逻辑谓词,我们选择具有嵌套的 AND 逻辑谓词的 Twig Pattern 进行实验.包括:针对不同的分支节点的数目;针对不同的数据集(例如 Xmark[21],DBLP[23],这里的实验结果使用的是 Xmark,使用 DBLP 的效果相类似);针对不同的文档大小等.结果表明:在处理 Twig Pattern 时,性能比 YFilter 有所提高,尤其是在处理大文档的情况下.

首先给出针对小文档的部分实验结果.以查询的分支节点(PQNode)数是 3 为例,我们借用 YFilter 的 XPath 生成器,生成 Twig Pattern 的参数是:6 0.2 0.2—num_nestedpaths=3—distinct=TRUE,其中:6 表示查询深度;两个 0.2 表示"//"和"*"出现的概率;3 表示分支节点数;TRUE 表示每个 Twig Pattern 都不相同.实验结果如图 7 所示,其中:纵轴表示时间(ms×10),横轴表示查询个数,分别为 5 000,10 000,50 000,100 000;文档的大小为 10K,20K,50K.

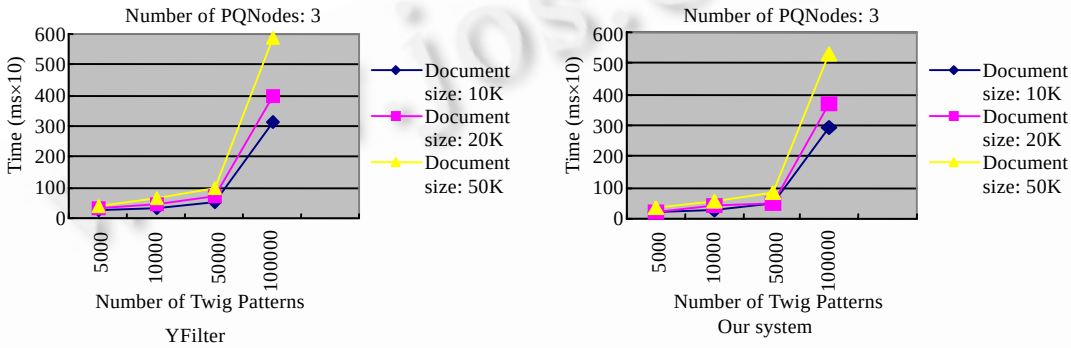

Fig.7 Experiments for small XML documents

图 7 针对小 XML 文档的实验

YFilter 的查询处理时间分为基本处理时间和后置处理时间.其中,后置处理时间是处理 Twig Pattern 谓词部分的耗时.在处理小文档和少量查询时,YFilter 的后置处理时间占非常小的比例.例如:在分支为 3,查询数量为 5 000,文档大小为 10K 时,处理时间为 265ms,而后置处理时间为 51ms,约占 20%;随着查询数量和文档的增大,后置处理时间所占比例会增大.例如,同样条件下,当查询数为 100 000,文档为 50K 时,后置处理时间超过 55%.正如实验结果表明的,随着查询数量的增大和 XML 文档的增大,性能的提高越来越明显.

下面给出针对大文档的实验部分结果.我们使用与上面相同的参数生成 Twig Pattern.实验的结果如图 8 所示,其中:纵轴表示时间(ms×100),横轴表示查询个数,分别为 5 000,10 000,50 000,100 000;所使用的文档大小分别为 500K,1MB,2MB.在处理大文档的情况下,无论是对大量查询还是少量查询,我们的系统都具有非常明显的优势.在文档为 1MB 时,性能提高 2 倍以上.例如,在文档为 1MB,查询数目为 5000 时,我们的处理时间是 4 406ms,而 YFilter 的处理时间是(3594+8422)=12016ms,括号中的前一个数字 3 594 是基本处理时间,后一个数字 8 422 是后置处理时间.可以看出,YFilter 仅后置处理的时间就是我们系统的近两倍.当文档大小不变(1MB),查询数量增加到 100 000 时,我们的处理时间是 86 062ms,而 YFilter 的处理时间是(14733+168844)=183577ms,其后置处理时间已经超出两倍.当文档为 2M 时,分别对 5 000 和 10 000 个查询,我们的系统与 YFilter 的性能对比是 8969/(7047+33906),17218/(9703+71188).可以看出,仅后置处理时间就已经超出 4 倍左右.因此,YFilter 的方法不适合于大文档和复杂查询的应用,而仅适合于处理简单查询的消息过滤.由于我们针对复杂 Twig Pattern 查询,采用单遍处理 XML 流的方法,避免了中间结果的产生和后置处理的过程,因此,也有利于处理大文档的复杂查询.

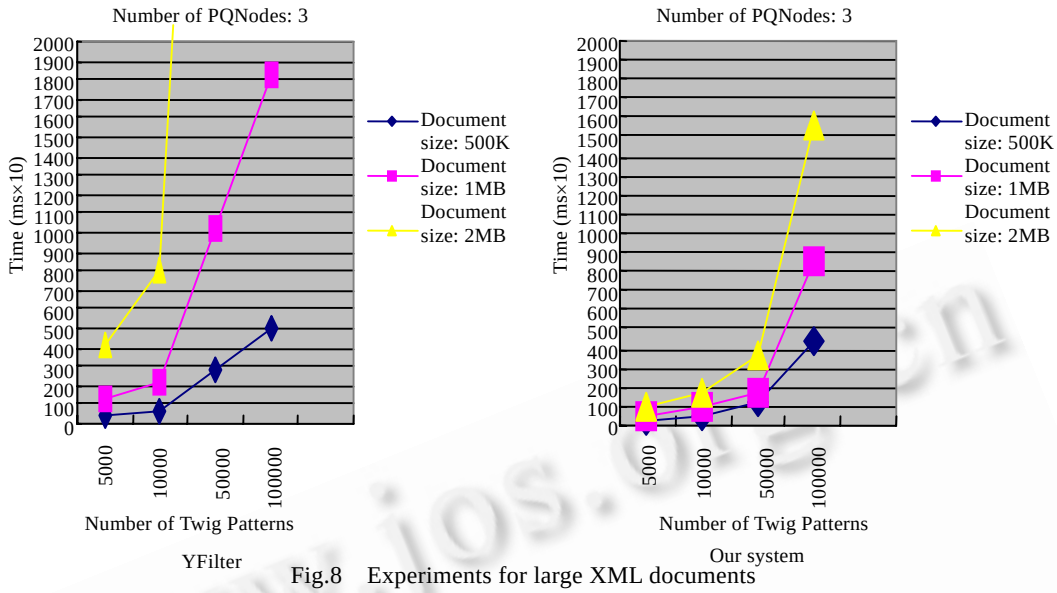

Fig.8 Experiments for large XML documents

图 8 针对大 XML 文档的实验

4 结束语

Twig Pattern 由于通常具有嵌套谓词,查询处理时相对较为耗时.针对 XML 流数据的、具有嵌套 AND/OR 谓词的复杂 Twig Pattern 查询处理,我们提出一种新的方法:为了提高查询处理性能,将所有 Twig Pattern 合并为一个共享前缀的查询树,其中,AND/OR 谓词被表示为单独的抽象语法树.因而,可以依据文档顺序,单遍地处理复杂 Twig Pattern 的匹配.限于篇幅,文中算法没有讨论递归 XML 文档的处理,但稍加扩充就很容易实现这一点.递归文档处理的关键问题是:文档是递归的,并且查询树的谓词子树中含有“//”.解决该问题的关键是区分文档中嵌套递归的同名元素(处于文档中的不同层次),因而只需在进入运行时栈时,将嵌套递归元素对应的查询树中的节点作为不同的状态加以处理.与已有的后置处理方法相比,我们所提出的方法能够改善 Twig Pattern 的处理性能,针对大 XML 文档(1M 以上),具有非常明显的改善.我们的系统能够处理自由 XML 文档(没有 DTD 的 XML 文档)及具有 DTD 的 XML 文档.针对具有 DTD 的情况,讨论并将已有研究工作应用于 XML 流处理系统,对一组 Twig Pattern 进行预先优化,以提高系统处理性能.

进一步的工作是完善我们的系统,包括返回节点缓存策略以及如何有效地将结果集重构为 XML 文档,并及时分发给用户等.

References:

[1] Berglund A, Boag S, Chamberlin D, Fernandez MF, Kay M, Robie J, Simon J. XML path language (XPath) 2.0 W3C working draft 16. Technical Report, WD-xpath20-20020816, World Wide Web Consortium, 2002. http://www.w3.org/TR/2002/WD-xpath20-20020816/

[2] Altinel M, Franklin MJ. Efficient filtering of XML documents for selective dissemination of information. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, Whang KY, eds. Proc. of the 26th Int’l Conf. on Very Large Data Bases (VLDB 2000). Cairo: Morgan Kaufmann Publishers, 2000. 53−64.

[3] Diao YL, Altinel M, Franklin MJ, Zhang H, Fischer P. Path sharing and predicate evaluation for high-performance XML filtering. ACM Trans. on Database System (TODS 2003), 2003,28(4):467−516.

[4] Gupta A, Suciu D. Stream processing of XPath queries with predicates. In: Halevy AY, Ives ZG, Doan A, eds. Proc. of the 2003 ACM SIGMOD Int’l Conf. on Management of Data (SIGMOD 2003). San Diego: ACM Press, 2003. 419−430.

[5] Green TJ, Miklau G, Onizuka M, Suciu D. Processing XML streams with deterministic automata and stream indexes. ACM Trans. on Database Systems (TODS 2004), 2004,29(4):752−788.

[6] Olteanu D, Kiesling T, Bry F. An evaluation of regular path expressions with qualifiers against XML streams. In: Dayal U, Ramaritham K, Vijayaraman TM, eds. Proc. of the 19th Int’l Conf. on Data Engineering (ICDE 2003). Bangalore: IEEE Computer Society, 2003. 702−704.

[7] Ludscher B, Mukhopadhyay P, Papakonstantinou Y. A transducer-based XML query processor. In: Bressan S, Chaudhri AB, Lee ML, Yu JX, Lacroix Z, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases (VLDB 2002). Hong Kong: ACM Press, 2002. 227−238.

[8] Gao J, Yang DQ, Tang SW, Wang TJ. Tree-Automata based efficient XPath evaluation over XML data stream. Journal of Software, 2005,16(2):223−232 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/223.htm

[9] Bruno N, Gravano L, Koudas N, Srivastava D. Navigation—vs. index-based XML multi-query processing. In: Dayal U, Ramaritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering (ICDE 2003). Bangalore: IEEE Computer Society, 2003. 139−150.

[10] Gong XQ, Qian WN, Yan Y, Zhou AY. Bloom filter-based XML packets filtering for millions of path queries. In: Proc. of the 21st Int'l Conf. on Data Engineering (ICDE 2005). Tokyo: IEEE Computer Society, 2005. 890−901. http://ieeexplore.ieee.org/iel5/9680/30564/01410201.pdf

[11] Kwon J, Rao P, Moon B, Lee S. FiST: Scalable XML document filtering by sequencing twig patterns. Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB 2005). Trondheim: VLDB Endowment, 2005. 217−228.

[12] Gupta A, Halevy A, Suciu D. View selection for XML stream processing. In: Fernandez MF, Papakonstantinous Y, eds. Proc. of the 5th Int'l Workshop on Web and Databases (WebDB 2002). Madison: ACM Press, 2002. 83−88.

[13] Chidlovskii B. Using regular tree automata as XML schemas. In: Proc. of the IEEE Advances in Digital Libraries. Washington, 2000. 89−104. http://citeseer.ist.psu.edu/chidlovskii99using.html

[14] Peter TW. Minimising simple XPath expressions. In: Giansalvatore M, Jérôme S, Mecca, G, Siméon J, eds. Proc. of the 4th Int'l Workshop on Web and Databases (WebDB 2001). Santa Barbara: ACM Press, 2001. 13−18.

[15] Su H, Rundensteiner EA, Mani M. Semantic query optimization for XQurey over XML. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB 2005). Trondheim: VLDB Endowment, 2005. 277−288.

[16] Jiang HF, Lu HJ, Wang W. Efficient processing of XML twig queries with or-predicates. In: Weikum G, König AC, Deßloch S, eds. Proc. of the 2004 ACM SIGMOD Int'l Conf. on Management of data (SIGMOD 2004). Paris: ACM Press, 2004. 59−70.

[17] Uchiyama H, Onizuka M, Honishi T. Distributed XML stream filtering system with high scalability. In: Proc. of the 21st Int'l Conf. on Data Engineering (ICDE 2005). Tokyo: IEEE Computer Society, 2005. 968−977. http://ieeexplore.ieee.org/iel5/9680/30564/01410208.pdf

[18] Wutka. DTD parser. 2000. http://www.wutka.com/dtdparser.html

[19] Luis Diaz A, Lovell D. XML generator. 1999. http://www.alphaworks.ibm.com/tech/xmlgenerator

[20] JXPath. http://jakarta.apache.org/commons/jxpath/

[21] Busse R, Carey M, Florescu D, Kersten M, Manolescu I, Schmidt A, Waas F. Xmark: An XML benchmark project. 2001. http://monetdb.cwi.nl/xml/index.html

[22] Megginson D. Simple API for XML. 2000. http://sax.sourceforge.net

[23] Ley M. DBLP DTD. 2001. http://www.acm.org/sigmod/dblp/db/about/dblp.dtd

附中文参考文献:

[8] 高军,杨冬青,唐世渭,王腾蛟.基于树自动机的 XPath 在 XML 数据流上的高校执行.软件学报,2005,16(2):223−232. http://www.jos.org.cn/1000-9825/16/223.htm

杨卫东(1967－),男,陕西西安人,博士,副教授,主要研究领域为 XML 数据管理,软件开发方法.

施伯乐(1935－),男,教授,博士生导师,CCF 高级会员,主要研究领域为数据库理论与应用.

王清明(1981－),男,硕士生,主要研究领域为 XML 流数据处理.