第34卷 第16期 计 算 机 工 程 2008年8月
Vol.34 No.16 Computer Engineering August 2008
·软件技术与数据库· 文章编号：1000—3428(2008)16—0063—03 文献标识码：A 中图分类号：TP311.5

基于自动机的XML流多查询处理

张兵令
(上海杉达学院教务处，上海 201209)

摘 要：XML流数据处理在研究领域引起广泛关注，该文针对XML流上的多查询处理提出一种算法，把多个查询合并为一个共享前缀的查询树，应用自动机和运行时栈相结合的方法，单遍扫描XML流处理数据流上的多个查询。该算法采用一种分层栈结构保存查询模式匹配候选集，利用XML节点的区间编码来确定节点之间的关系，返回整条匹配路径。

关键词：XML数据流；前缀共享；自动机

Automata-based Multiple Queries Handling over XML Streams

ZHANG Bing-ling
(Administration Office of Shanghai Sanda Institute, Shanghai 201209)

【Abstract】 The problem of processing streaming XML data is gaining widespread attention from the research community. This paper proposes a novel algorithm for handling of multiple queries over the XML streams. The approach merges the multiple queries into a single prefix sharing query tree, and combines the automata with the runtime stack structure to answer multiple XML path queries through scanning the XML stream at the same time. The algorithm returns the whole matching path by using a new hierarchical stack structure to store the result of query pattern matches, and confirms the relationship within the node through the position encodes in the XML data stream.

【Key words】 XML data stream; prefix sharing; automata

1 概述

随着网络技术的快速发展和广泛应用，XML流数据处理技术引起研究界和工业界的广泛兴趣，并迅速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。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。因此XML流数据与XML查询的高效匹配算法是XML流处理系统的关键研究问题之一。

XML数据流处理系统的输入数据以一种在线、瞬态、持续的数据流形式到达，这就意味着数据流的大小潜在无界；数据流只能单遍扫描，数据流中的某个元素要么被丢弃，要么被归档存储；系统无法控制将要处理的新到达数据元素的顺序；XML流中的标记会有多层递归嵌套结构，而XML查询又有祖先/后裔轴，两者的结合使得可能成为查询结果的组合成指数级地增长。

目前，针对XML流查询主要采用基于自动机的方法[1−2]；其他的有基于索引的方法[3]、基于谓词的方法[4]、基于Bloom-Filter的方法[5]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[2]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不带分支的单查询处理，没有考虑同时进行多个查询处理的情况。TwigM[1]在自动机的基础上，运用运行时栈解决了带分支的单查询处理，但是仍然没有解决多查询处理的问题。

2 本文主要工作和研究成果

本文主要的工作和研究成果如下：

(1)提出一种在XML流上进行多查询处理的新算法，算法将自动机与运行时栈结构相结合，单遍扫描XML数据流，有效地解决了在包含递归结构的XML数据流上同时进行多个查询的问题。

(2)与一般的XPath查询处理不同，本文提出的算法可以返回多个查询的整条查询匹配路径，而不只是返回节点。

(3)算法利用区间编码，记录、判断节点之间的关系。

3 算法介绍

在XML流数据处理系统中，XML数据集非常庞大的，且具有高度的流动性。而相对稳定的是用户查询，大量预先定义的用户查询被注册到处理系统中，等待数据的到来。一旦数据到来，将驱动查询的执行。

3.1 基本概念和相关约定

XML文档：一个XML文档包含了从根元素开始的嵌套元素结构，每个元素有一个标签与之相对应，除了字符串数据，它还可以包含属性值和嵌套子元素。可以把一个XML文档看作是一棵有根的、排序的、有标签的树，每一个节点与一个元素或一个值相对应，边则代表了节点之间的父/子关系(即元素-子元素关系或者元素-值的关系)或祖先/后裔关系，兄弟节点的顺序隐含定义了一棵树上的节点总顺序。如图1(a)所示。

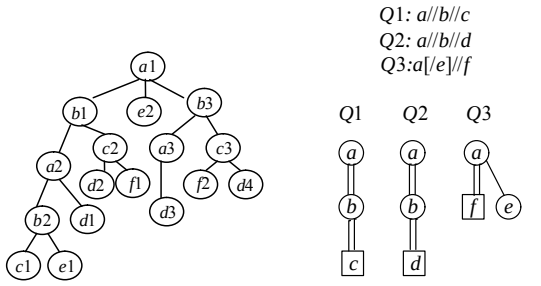
XML查询语言：在XML数据流中进行查询时使用的语言。XPath是W3C发布的用于对XML文档子集寻址定位的标准语言，XPath主要使用URL中路径的表示方法在XML文档的层次结构中搜索和定位。在后面的讨论中就使用Xpath作为XML的查询语言。如图1(b)所示。

把正在被SAX语法分析器处理的XML节点叫做“活跃

作者简介：张兵令(1973－)，女，讲师、硕士研究生，主研方向：XML流处理，数据挖掘

收稿日期：2007-09-21 **E-mail**：zbling@sandau.edu.cn

节点”，即那些开始标记已经被读取，而结束标记尚未到达的 XML 节点。



(a)XML 文档片断 D　　(b)XML 流上的 3 个查询 Q

图 1　1 个 XML 流上的多查询示例

用 *StartElement*(*tag*,*level*,*id*)表示某一节点的开始标记。其中，*tag* 表示开始被处理的节点的标记名；*level* 表示开始被处理的节点在相应的 XML 树上所处的层次；*id* 表示开始被处理的节点的唯一标识(XML 流中的标记有递归结构，具有同一标记的节点可能不止一个，因此 *tag* 和 *id* 有时并不相同)。例如：*StartElement*(*title*,3,*title*2)就表示遇到了图 1 中的第 2 个 *title* 节点的开始标记。

用 *EndElement*(*tag*,*level*)表示某一节点的结束标记，其中 *tag*, *level* 的含义与 *StartElement*(*tag*,*level*,*id*)相同。例如：*EndElement*(*title*,3)表示遇到了图 1 中的第 2 个 *title* 节点的结束标记。

把 XML 树中具有多于一个子节点的节点称为“分支节点”；把可能成为查询结果的节点集叫做“候选查询结果”；把查询语句中要求返回给用户的节点叫做“返回节点”，返回节点在 XML 树中通常以椭圆形或者长方形来表示。本文在 XML 树中以长方形来表示查询的返回节点。

3.2　多查询处理方案

为了加快 XML 流查询系统的查询速度，考虑单遍扫描 XML 流的过程中同时处理相近的一些查询，利用前缀共享的方法可以使近似查询共享一个查询计划。考虑将图 1(b)中的 3 个查询进行前缀共享处理，具体做法如下：首先，将 3 个查询从根开始的共同部分挑出来，在本例中的共同部分就是“*a*//”，把这部分内容构造一棵查询树；然后，把每个查询中剩余的部分依次添加到这棵查询树上，从而构造一棵包含所有查询的查询树。图 2 就是根据图 1(b)中的 3 个查询构造的一个共享查询树。

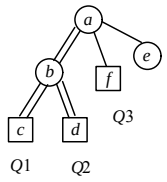


图 2　由图 1(b)构造的共享查询树

通过前缀共享把相似的一组查询合并成一个查询，现在还面临 2 个挑战：(1)和单查询不同，这里的查询树包含了多个查询，有多个返回节点。如何保存这些返回节点，使得将来返回查询结果给不同用户时，能够做到各取所需，而不是所有结果混淆在一起。(2)对于单查询来说，任何分支上的条件得不到满足，查询匹配就会失败，不需要保存中间结果。但是，对于多查询来说，分支条件可能代表的是不同的查询条件，当某一分支条件得不到满足时，不能轻易丢弃中间结果。因为这个中间结果对一部分查询是无意义的，但对一部分查询仍然是有意义的。如图 1(a)XML流中的节点d_2，虽然不能同时满足图 2 中共享查询树节点b的 2 个匹配条件，但是节点d_2对于子查询$Q1$ 仍然是是有意义的。所以，需要保存这个中间节点。

为了解决上面的问题，创建如图 3 中的自动机结构。具体如下：(1)与单查询处理相同，为共享查询树中的每一个查询节点创建一个对应的自动机机器节点，如：$v1$←→a, $v2$←→b等。这里有 3 个返回节点，均用实心的黑色圆表示。自动机机器节点之间的边上所标示的条件表示查询节点之间的关系，如“父/子”关系用“(=,1)”来表示，“祖先/后裔”关系用“(≥,1)”来表示。(2)为每一个机器节点创建一个栈，用来记录满足查询子条件的当前活跃节点的相关信息。如v_1的栈中存放的是可到达“//*a*”的所有活跃节点信息；v_2的栈中存放的是可到达“//*a*//*b*”的所有活跃节点信息；……；依此类推。栈中存放的活跃节点的信息内容由两部分组成，上面已经做出解释，这里不再赘述。当XML流数据节点满足相应的查询条件时，即被推入相应机器节点的栈中。(3)为每个子查询创建一个分层栈结构，用来保存查询候选结果集，这个分层栈结构可以在匹配过程中根据需要动态生成。

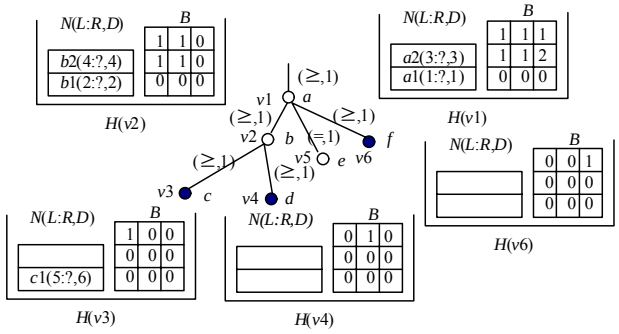


图 3　自动机和栈相结合处理多查询的快照结构 1

再来看这个带有栈结构的自动机对多查询共享树的处理过程，当 SAX 语法分析器发现一个 XML 节点的开始标记时，由 *StartElement*(*tag*,*level*,*id*)触发对与 *tag* 对应的机器节点 *v* 的验证。若 *v* 满足下列 2 个条件之一，则将 XML 元素推入 *v* 的栈中，作为从根到 *v* 的一个子查询候选解决方案。

条件 1 *v* 是根节点，且 XML 元素满足进入 *v* 边上的条件；

条件 2 *v* 不是根节点，且 XML 元素在 XML 树上所处的层次与 *v* 的父节点栈顶元素 *u* 的层次之差满足进入 *v* 的边上的条件。

在图 3 中，*StartElement*(b,4,b_2)事件将触发对v_2的验证，根据多查询共享树可知：v_2不是根节点，v_2的父节点栈顶元素是a_2，a_2的层次 3，XML节点b_2的层次 4。因此，$b_2.D$-$a_2.D$=4-3=1≥1，符合v_1和v_2之间的条件，故将相关的信息推入v_2的栈中。b_1在XML流中开标记所处的位置和层次信息已经确定，直接放入相应栈结构中。注意，此时b_1在XML流中开标记所处的位置尚无法确定，先作为空缺，不填入数据。v_2参与子查询$Q1$ 和$Q2$，因此，$b_2.B(1,1)=b_2.B(1,2)=1$。v_2不参与子查询$Q3$，$b_2.B(1,2)=0$。v_2只需要验证子查询$Q1$ 和$Q2$ 的一个分支，故$b_2.B(2,1)=b_2.B(2,2)=1$。v_2不需要验证子查询$Q3$ 的任何分支，故$b_2.B(2,3)=0$。此时，b_2还没有验证任何子查询的分支，$b_2.B(3,1)=b_2.B(3,2)=b_2.B(3,3)=0$。图 3 就是XML流经过节点$c_1$的开始标记后，自动机各机器节点栈中信息的快照。

当SAX语法分析器遇到一个XML节点的结束标记时，由*EndElement*(*tag*,*level*)触发对与*tag*对应的机器节点*v*的验证。若XML节点的*level*值*D*与机器节点*v*的栈顶元素的*level*值*D*相等并且*v*的栈顶元素的$B(2,1)=B(3,1),\ B(2,2)=B(3,2),\ \cdots$中的任何一个成立，则将标记计数器中的当前值赋给*v*的栈顶元素的*R*值，并把*v*的父栈中所有元素的*B*数组第三行相应元素值增加 1，同时把该栈顶元素从*v*的栈中弹出到相应的子查询分层栈结构中。若XML节点的*level*值*D*与机器节点*v*的栈顶元素的*level*值*D*相等，但数组*B*中的信息都不符合要求，则直接弹出*v*的栈顶元素(即抛弃该节点)。如图 4，当SAX语法分析器遇到c_1的结束标记时，*EndElement*(*c*,5)事件将触发对v_3的验证。v_3的栈顶元素的*level*值*D*正好是c_1的*level*值*D*，并且此时$c_1.B(2,1)= c_1.B(3,1),\ c_1.B(2,2)= c_1.B(3,2),\ c_1.B(2,3)= c_1.B(3,3)$都成立。因此，作如下的设置：6——〉$c_1.R,\ b_2.B(3,1)+1$——〉$b_2.B(3,1),\ b_1.B(3,1)+1$——〉$b_1.B(3,1)$，$c_1$从$v_3$的栈顶弹出到*HD*(*C*)。图 4 是XML流经过节点c_1的结束标记后，自动机各机器节点栈中信息的快照。

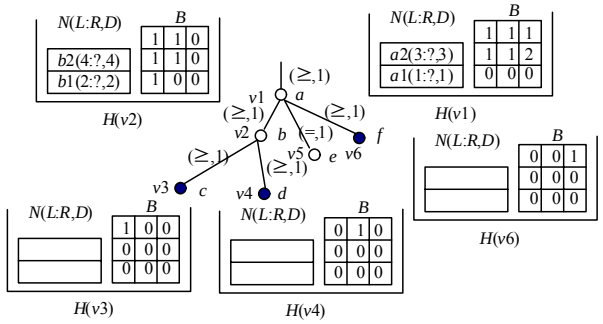


图 4　自动机和栈相结合处理多查询的快照结构 2

按以上思路，重复上述操作，处理好每一个 XML 流中节点的开始、结束标记，直到根节点，输出根节点中的候选集。

在上面的讨论中，没有考虑查询中的“ * ”节点。事实上，没有必要专门为“ * ”节点创建自动机，可以用查询节点之间的关系约束来表达查询中的“ * ”节点。现在假设有这样一个查询：*Q*4: *a*/*/*b*，只需要为这个查询创建 2 个自动机v_1和v_2，分别对应查询中的*a*和*b*节点。自动机v_1和v_2之间表示两者关系的条件改成“ (=,2) ”即可。同样的道理，对于查询*Q*4: *a*//*//*b*，只要把自动机v_1和v_2之间的条件改成“ (≥,2) ”即可。

3.3　实验

下面的实验结果是在主频为 550 MHz 的 Pentium III 处理器上运行得出的，内存为 256 MB ，操作系统是 Windows 2000；程序设计语言为 Java。

在实验过程中使用的数据集是 DBLP ；使用 BookDTD 作为输入的 DTD；查询集从 100 条~100 000 条；XML 文档流从 1 MB~5 MB；XML 文档树的最大深度 20；XML 文档树的最大递归深度 9。查询生成工具采用 XPath 自动生成工具，设置每个查询的最大深度为 6，最大分支节点数是 4，查询中“ // ”出现的概率为 0.2，“ * ”出现的概率为 0.1，查询之间的相似度设为 0.75。

在实验数据的统计过程中，将每组查询运行 10 次，取平均的处理时间。

下面给出了分别针对 500 KB, 1 MB, 2 MB, 5 MB 的 XML 文档进行查询处理的实验结果。具体实验结果如表 1 所示。

表 1　XML 文档上的多查询所需时间统计　　s

文档大小	查询个数/个					
	500	1 000	5 000	10 000	50 000	100 000
500 KB	0.08	0.12	0.27	0.86	1.92	4.76
1 MB	0.19	0.30	0.74	1.55	5.60	10.91
2 MB	0.53	0.97	2.21	4.45	11.75	27.54
5 MB	1.92	3.46	4.58	16.69	44.36	96.83

4　结束语

本文研究的是 XML 流查询相关技术，其主要目的是改进现有的流查询技术以加速查询匹配过程，同时可以支持对多个查询的优化。本文对现有的 XML 流查询技术进行了总结和改进，最终提出了一种新的高效处理多查询的方法。

参考文献

[1] Yi Chen, Davidson S B, Zheng Yifeng. An Efficient XPath Query Processor for XML Streams[C]//Proc. of ICDE'06. Atlanta, USA: IEEE Computer Society, 2006: 1-12.

[2] Diao Yanlei, Altinel M, Franklin M J, et al. Path Sharing and Predicate Evaluation for High-performance XML Filtering[J]. ACM Trans. on Database System, 2003, 28(4): 467-516.

[3] Bruno N, Gravano L, Koudas N, et al. Navigation vs Indexed-based XML Multi-query Processing[C]//Proc. of VLDB'04. Toronto, Canada: [s. n.], 2004: 1-12.

[4] Hou Shuang, Jacobsen H A. Predicate-based Filtering of XPath Expression[C]//Proc. of the 22nd IEEE International Conference on Data Engineering. Atlanta, USA: IEEE Computer Society, 2006. 53-64.

[5] Gong Xueqing, Qian Weining, Yan Ying, et al. Bloom Filter-based XML Packets Filtering for Millions of Path Queries[C]//Proc. of the 21st Int'l Conf. on Data Engineering. Tokyo, Japan: IEEE Computer Society, 2005: 890-901.

(上接第 62 页)

参考文献

[1] Gray J. The Dangers of Replication and a Solution[C]//Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Montreal, Canada: [s. n.], 1996: 173-182.

[2] Phatak S H. Multiversion Reconciliation for Mobile Databases[C]// Proc. of the 15th Int'l Conf. on Data Engineering.Sydney, Australia: [s. l.], 1999: 582-589.

[3] Phatak S H. Conflict Resolution and Reconciliation in Disconnected Databases[C]//Proc. of the 10th Int'l Workshop on Database and Expert Systems Applications. Florence, Italy: [s. n.], 1999: 76-81.

[4] 丁治明, 王　珊, 孟小峰. 移动复制数据库系统冲突检测及消解策略[J]. 计算机学报, 2002, 25(3): 297-305.

[5] 祝　庆. 移动数据库系统同步机制的研究与实现[D]. 北京: 中国科学院研究生院, 2004.

[6] Cormen T H. 算法导论[M]. 2 版. 潘金贵, 译. 北京: 机械工业出版社, 2006.