第32卷 第13期
Vol.32 № 13
计 算 机 工 程
Computer Engineering
2006年7月
July 2006
·软件技术与数据库· 文章编号：1000—3428(2006)13—0058—03 文献标识码：A 中图分类号：TP311

带谓词的XPath查询的即时处理

吴 年[1,2]，张 昱[1,2]
(1. 中国科学技术大学计算机科学技术系，合肥 230027；2. 中国科学院计算机科学重点实验室，北京 100080)

摘 要：介绍了一种立即计算谓词并即时输出的XML流数据查询系统XSIEQ。XSIEQ采用修改了的下推自动机技术，对多个XPath式按前缀共享的方式构造NFA，并对NFA状态进行类型标记和添加索引；从而在运行时能快速确定谓词计算和数据缓存等动作的时机，实现了即时处理；最后给出了XSIEQ和YFilter的查询性能对比及分析。

关键词：XML流；谓词计算；下推自动机

Immediate Processing of XPath Query with Predicates

WU Nian[1,2], ZHANG Yu[1,2]
(1. Department of Computer Science & Technology, University of Science & Technology of China, Hefei 230027;
2. Key Lab of Computer Science, Chinese Academy of Sciences, Beijing 100080)

【Abstract】 XSIEQ(XML stream query with immediate evaluation), a kind of XML stream query system, can evaluate predicates immediately and output in time. The design and implementation of XSIEQ are presented. XSIEQ uses a modified pushdown automata technique, and its idea is to convert all XPath expressions into a single NFA, then to attach type label and corresponding index info on NFA state. Accordingly, XSIEQ can quickly decide the opportunity of predicate evaluation and data cache in runtime, and implement immediate processing. In the end, the query performance comparison between XSIEQ and YFilter is given and analyzed.

【Key words】 XML stream; Predicate evaluation; Push down automata(PDA)

1 概述

在Web服务及其相关应用中，XML数据大多以流的形式进行传递、分发。XML流查询也被广泛用于信息选择分发、数据集成、管道处理、连续查询等应用中。

基本的XML流查询条件是一组XPath式。一个XPath式是对XML文档节点进行定位的一个字符串，并且可能包含对节点的谓词选择条件。XPath式中去除谓词后剩余的部分称为主XPath式，它确定XML查询匹配的候选结果节点。流查询的主流做法是根据查询XPath式构造查询自动机。如YFilter[1]将XPath式构造为NFA；XMLTK[2]从NFA构造Lazy DFA；SPEX[3]将单个XPath式转换为多个PDA(Push down automata)并连接为SPEX网。

在管道处理、连续查询等应用中，需要在查询过程中即时地输出当前查到的部分结果。实现即时输出的最大障碍是XPath式中存在的谓词。

例1 考虑XPath式P_1: /a[.//b=2] [e=3]/c[d]和XML片段：
<a>2<c><d/></c><e>3</e>

/a/c是P_1的主XPath式。当SAX解析获得XML片段中的/a/c节点时，谓词[e=3]尚未计算，故阻碍了查询结果的即时输出。

XMLTK只支持能在当前位置步计算的谓词，故实现有限的即时输出。YFilter在解析XML时缓存所有的候选结果节点和谓词中XPath式匹配的节点，在解析结束后通过后处理获得最终的查询结果。XPush[4]使用修改了的PDA，通过自底向上的计算对满足谓词的状态进行转换，到文档解析结束时根据最终匹配的状态集得到查询结果。

XSIEQ(XML Stream Query with Immediate Evaluation)旨在建立一套高效的、能立即计算谓词并即时输出的XML流查询引擎。当前，XSIEQ支持XPath式的孩子、子孙、自身和属性轴，支持含相对XPath式的复杂谓词(包括嵌套谓词)。XSIEQ采用索引技术和缓冲机制，通过构造PDA实现谓词的立即计算和查询结果的即时输出。

2 XSIEQ自动机

XSIEQ根据要查询的XPath式构造自动机。为支持谓词，XSIEQ使用带栈的NFA，即下推自动机(PDA)。

2.1 从XPath式构造自动机状态

从XPath构造自动机状态分为两步。

(1)将XPath式构造为NFA。XSIEQ采用和YFilter类似的方法，将XPath式中的主XPath式和谓词中的XPath提取出来，通过前缀共享的方法增量式地构造NFA。在NFA中，XPath式的位置步被映射到状态，节点测试是状态间的转换条件。对于谓词中的相对XPath式，是以包含该谓词的位置步到达的状态为起始状态，进而构造为NFA中的分支路径。

(2)对NFA状态进行类型标记并添加索引信息。需要标记以下3类NFA状态：

定义1 结果状态是指由主XPath式的最后一个位置步到达的NFA状态。该状态将保存它所对应的主XPath式对象列表LR。

基金项目：国家自然科学基金资助项目(60473068)；中国科学院计算机科学重点实验室开放课题基金资助项目(SYSKF0502)
作者简介：吴 年(1976—)，男，硕士，主研方向：XML数据处理；张 昱，副教授
收稿日期：2005-07-10 **E-mail**：nwu@ustc.edu

定义 2 叶子状态是指谓词中 XPath 式的最后一个位置步到达的 NFA 状态。该状态将保存它所关联的谓词对象列表 LP。

定义 3 对于某个 XPath 式，主路径上具有分支到谓词中 XPath 式的 NFA 状态被称为分支状态。该状态将保存从该状态分支出去的谓词对象列表 LB。

例 2 按照上述转换步骤，对例 1 中的 P_1 构造出图 1 所示的带索引信息的 NFA。状态用圆圈表示，双圈表示结果状态；弧代表状态转换，其中虚线表示子孙轴类型的转换，实线表示孩子轴类型的转换，弧上的标记表示节点测试。状态 S1 由于包含子孙轴类型的转换弧而具有到自身的转换弧。

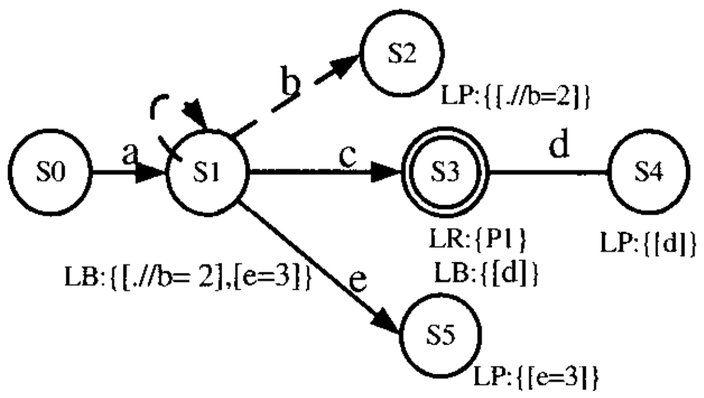


图 1 P_1 对应的带索引信息的 NFA

SAX 解析 XML 流时，SAX 事件触发自动机的状态转换。到达结果状态可以确定匹配待查 XPath 式的候选 XML 节点；到达叶子状态可以对谓词进行计算；分支状态是确定分支下各谓词最终计算状态的场所。通过标记这些状态就可以在 SAX 解析时快速确定谓词计算和数据缓存等动作的时机，实现即时查询。

2.2 定义

XSIEQ 自动机是一个修改了的非确定的 PDA，它包含普通的 PDA 状态、缓冲区状态和谓词计算状态；并且转换函数也包含对谓词进行计算的函数。以下∑为 XML 元素名的集合，V 表示字符串值集合，D 表示 XML 片段的集合。

定义 4 一个 XSIEQ 自动机是元组(Q^s, Q^b, Q^p, $q^s{}_0$, $q^b{}_0$, $q^p{}_0$, F, PF, B, δ)，其中：

(1)Q^s 是普通的 PDA 状态集合，Q^b 是缓冲区状态集合，Q^p 是谓词状态集合；

(2)$F \subseteq Q^s$ 是结果状态集合，$PF \subseteq Q^s$ 是叶子状态集合，$B \subseteq Q^s$ 是分支状态集合；

(3)$Q = Q^s \times Q^b \times Q^p$ 为状态集合，状态 $q = (q^s, q^b, q^p) \in Q$, $q^s \in Q^s$, $q^b \in Q^b$, $q^p \in Q^p$；

(4) $(q^s{}_0, q^b{}_0, q^p{}_0) \in Q$ 是初始状态；

(5) $\delta = \{t_{forward}, t_{collect}, t^p{}_{eval}, t^p{}_{reset}, t^b{}_{add}, t^b{}_{set}, t_{accept}\}$ 是一组状态转换函数，定义如下：

$t_{forward}$: $Q^s \times \sum \to Q^s$

$t_{collect}$: $D \times F \times (\sum \cup V) \to D$

$t^p{}_{eval}$: $Q^p \times PF \times (\sum \cup V) \to Q^p$

$t^p{}_{reset}$: $Q^p \times B \to Q^p$

$t^b{}_{add}$: $Q^b \times D \times Q^p \to Q^b$

$t^b{}_{set}$: $Q^b \times Q^p \to Q^b \times Q^b$

t_{accept}: $Q^p \to P(I)$

若 XPath 式 P 没有谓词，则 $Q^b(P)$和 $Q^p(P)$均为空。若 P 包含 m 个顶层谓词，则用 m 位位组标识 P 中各谓词的计算状态(即谓词状态)。单个谓词的状态分未确定、确定为假、确定为真 3 种。当某谓词确定为假后，与该谓词相关的缓存数据将被清除，这样谓词又处于未确定状态。为此，位组中各位用 0 表示未确定和确定为假，1 表示确定为真。当实现 n 个 XPath 式的查询时，每个 XPath 式 P_i 有自己的谓词状态 $Q^p(P_i)$和缓冲区状态 $Q^b(P_i)$，则

$$Q^b = Q^b(P_1) \cup \ldots \cup Q^b(P_n)$$

$$Q^p = Q^p(P_1) \cup \ldots \cup Q^p(P_n)$$

$t_{forward}$ 为状态转换操作。$t_{collect}$ 为候选结果的收集操作，它在进入到结果状态时启动一次收集过程，回溯到结果状态时结束。$t^p{}_{eval}$ 表示到达叶子状态时根据当前 XML 数据计算谓词并更新谓词状态；$t^p{}_{reset}$ 表示回溯到分支状态时对谓词状态进行复位。t_{accept} 是在谓词状态确定时标记候选结果是否是最终要输出的结果并执行输出。$t^b{}_{add}$、$t^b{}_{set}$ 是对缓冲区的操作，在下节中说明。

2.3 缓冲区管理

在 XML 流解析过程中，收集到主 XPath 式匹配的候选结果时，相关的谓词条件可能尚未完全确定，这时需要缓存候选结果。由于 XPath 式的谓词状态是实时变化的，而缓存数据项的谓词状态与当前 XPath 式的谓词状态变化不同步，因此缓存数据项必须保存它所关联的谓词状态，即每个缓存的数据项是一个二元组<*data*, *pStatus*>，$data \in D$, $pStatus \in Q^p$。

自动机定义了两种缓冲区操作：

(1) $t^b{}_{add}$ 将收集到的候选结果和当前谓词状态<*data*, q^p>加入到缓冲区 q^b 中；

(2) $(q^b, out) \leftarrow t^b{}_{set}(q^b, q^p)$表示根据 q^p 对 q^b 进行维护，维护后缓冲区状态为 q^b，并将已确定的结果通过 *out* 带回。分两种情况：

1)当 q^p 中谓词 *pred* 的计算为真时，执行 $t^b{}_{set}$ 将缓冲区中 *pStatus* 的状态为 0 的 *pred* 修改为 1。若某数据项的 *pStatus* 各位均为 1，则将其移入 *out* 中；

2)当回溯到分支状态时，若 q^p 中存在有谓词 *pred*，且状态为 0，则说明该谓词不满足；执行 $t^b{}_{set}$ 将清除缓冲区中 *pStatus* 的 *pred* 状态为 0 的数据项。

自动机在两种场合会调用 t_{accept} 处理输出：一是上述的 1)；二是在候选结果收集完毕，q^p 已完全确定时。

3 XSIEQ 自动机的运行

3.1 运行过程

XSIEQ 自动机在运行时维持一个当前状态 $q=(q^s, q^b, q^p)$、当前收集的数据 d 和栈 s，其中 q^s 由 NFA 状态集组成，q^b 和 q^p 为各 XPath 式相应状态的并置。自动机的运行由 SAX 事件驱动，初始时 $q=(q^s{}_0, q^b{}_0, q^p{}_0)$，$q^s{}_0$ 由 NFA 根状态组成。

startDocument( )
  $q^s \leftarrow q^s{}_0$; $q^p \leftarrow q^p{}_0$; $q^b \leftarrow \phi$; push(s, q^s); $d \leftarrow \phi$;
startElement(a)
  $q^s \leftarrow t_{forward}(q^s, a)$; push($s$, q^s);
  if ($q^s \in PF$) { $q^p \leftarrow t^p{}_{eval}(q^p, q^s, a)$;
    $(q^b, out) \leftarrow t^b{}_{set}(q^b, q^p)$; $t_{accept}(out)$; }
  if ($q^s \in F$) $d \leftarrow t_{collect}(\phi, q^s, a)$;
character(str)
  if ($q^s \in PF$) { $q^p \leftarrow t^p{}_{eval}(q^p, q^s, a)$;
    $(q^b, out) \leftarrow t^b{}_{set}(q^b, q^p)$; $t_{accept}(out)$; }
  if ($q^s \in F$) $d \leftarrow t_{collect}(d, q^s, str)$;
endElement(a)
  if ($q^s \in F$){ $d \leftarrow t_{collect}(d, q^s, str)$;
    if (q^p is true) t_{accept} (<d, q^p>);
    else $q^b \leftarrow t^b{}_{add}(q^b, d, q^p)$;}
  if ($q^s \in B$) { $q^p \leftarrow t^p{}_{reset}(q^p, q^s)$;
    $(q^b, out) \leftarrow t^b{}_{set}(q^b, q^p)$;}
  $q^s \leftarrow$pop(s);
endDocument( ) do nothing.

以上代码定义了各主要 SAX 事件发生时自动机的执行过程。startElement(*a*)事件处理中根据 *a* 更新 q^s 并将 q^s 入栈，同时计算存在性谓词并更新 q^p；character(*str*)事件处理中，计算普通谓词并更新 q^p。q^p 更新后将设置缓冲区状态，并将确定的缓存数据项进行即时输出处理。endElement(*a*)事件处理中重置谓词状态，并将收集到的结果状态数据进行输出或缓存，最后将栈顶状态出栈。

3.2 运行举例

例 3 对例 1 中的查询构造 XSIEQ 自动机并执行，执行时的状态转换和缓冲区操作过程如表 1 所示，栈的变化如图 2 所示。图 2 中的每个栈帧由一个 q^s 组成，直接使用 NFA 状态集表示。注：E- t^p_{eval}, R- t^p_{reset}, C- $t_{collect}$, S- t^b_{set}, A- t^b_{add}, O- t_{accept}。

表 1 运行时的状态转换和操作

Q	$\sigma \in \Sigma \cup V$												
	start	<a>		2		<c>	<d>	</d>	</c>	<e>	3	</e>	
q^s	S0	S1	S2	S2	S2	S3	S4	S4	S3	S5	S5	S5	S1
q^p	000	000	000	001	001	001	101	101	001	001	011	011	000
q^b									b1	b1			
输出											b2		
操作				ES		C	CES	C	CARS		ESO		

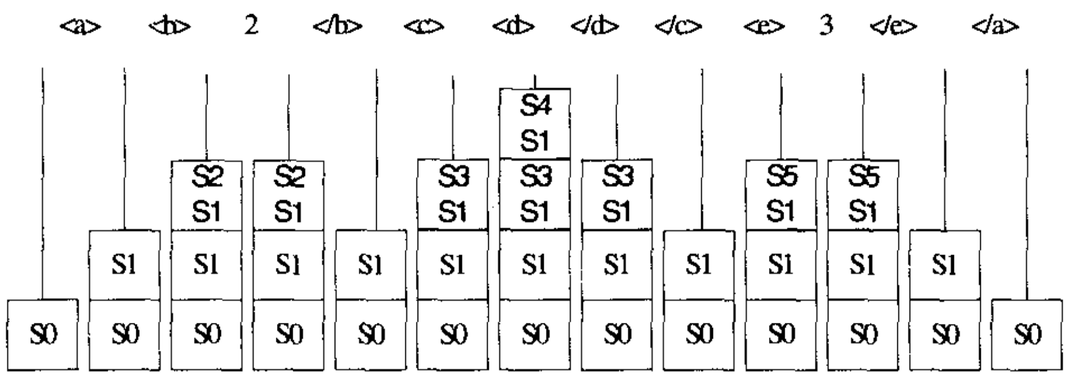


图 2 运行时的状态栈

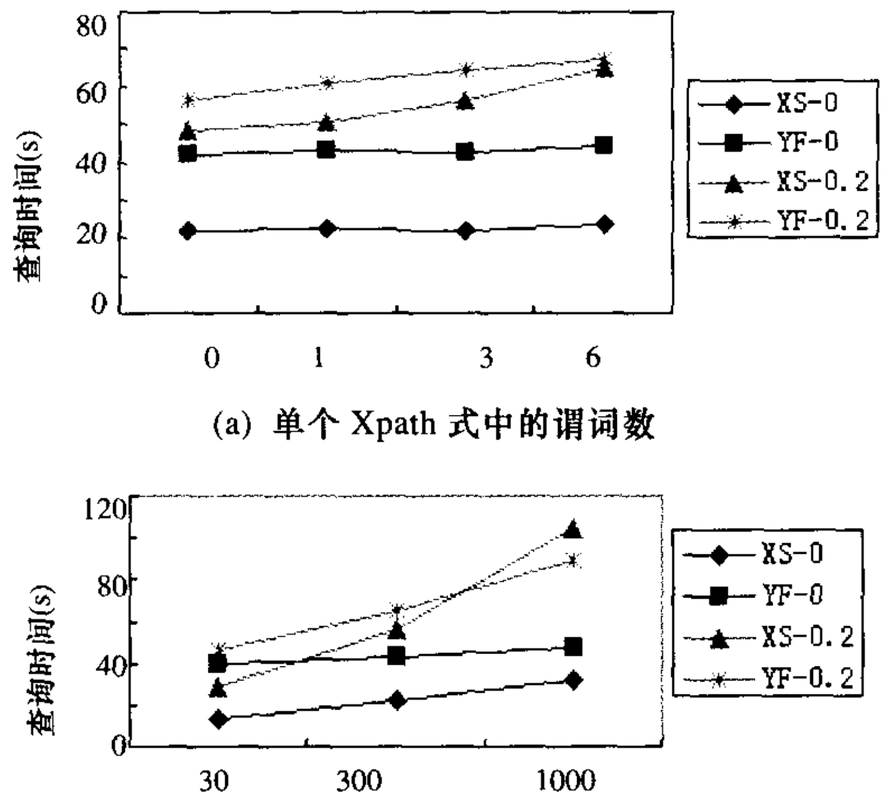
P_1 的 3 个谓词自左至右依次编号为 0,1,2，初始 q^p 为"000"。character(2)中谓词 0 计算为真，q^p 变为"001"，由于缓冲区为空，因此 set 动作没有影响；在 startElement(c) 中将到达结果状态 S3，开始收集候选结果 *d* 为"<c>"；在 startElement(d)中继续收集，*d* 变为"<c><d>"，同时谓词 2 计算为真，q^p 变为"101"，set 动作对缓冲区仍无影响。在 endElement(c)中，候选结果收集完成，*d* 为"<c><d/></c>"，由于 q^p 为"101"，即谓词 1 状态未确定，故将项 b1=("<c><d/></c>","101") 加入缓冲区；由于 S3 也是分支状态，故将 q^p 复位为"001"，由于缓冲区中不存在谓词 2 状态为 0 的数据项，因此无清除操作。在 character(3)中，谓词 1 计算为真，q^p 变为"011"，执行 set 操作使项 b1 中的谓词状态更新为"111"，即成为项 b2，由于 b2 的谓词状态全部为 1，故从缓冲区中去除并调用 t_{accept} 操作输出"<c><d/></c>"。

4 实验结果与分析

笔者用 Java 实现了 XSIEQ，并与 YFilter 进行了对比测试。实验平台为：Windows 2000，CPU AMD Athlon XP 2500+，内存 512MB DDR。测试用的 XML 文件用 XMark[5]生成，XPath 式使用 YFilter 自带的 XPath 式生成工具生成。

实验考核的因素包括 XPath 式的个数、单个 XPath 所含的谓词数、通配符和子孙轴的出现概率等。图 3 为实验结果。

图 3 中查询时间忽略了 XPath 式解析和自动机构造所花的时间；图例中的 XS 和 YF 分别表示 XSIEQ 和 YFilter，"-0"和"-0.2"分别表示通配符和子孙轴概率均为 0 和均为 0.2。图 3(a)为 300 个 XPath 式，1M*100XML 文档，图 3(b)为单个 Xpath 中的谓词数为 3，1M*100XML 文档。



(a) 单个 Xpath 式中的谓词数

(b) 查询的 Xpath 式个数

图 3 XSIEQ 与 YFilter 的查询时间比较

从图 3(a)中看出在对 300 个 XPath 查询"-0"时，XSIEQ 的查询时间约是 YFilter 的一半，随谓词数的增加而趋于平稳；当通配符、子孙轴概率较高时("-0.2")，XSIEQ 与 YFilter 的差距缩小，且随着谓词数的增加，XSIEQ 的查询时间增长得更快。这是因为谓词数增加将导致 XSIEQ 对缓冲区操作的增加；而 YFilter 先对 XML 预解析并在内存中保留完整的信息，因此在查询时不需要操作缓冲区。

从图 3(b)中看出，随着 XPath 式个数的增长，XSIEQ 查询时间的增长趋势较 YFilter 快，特别是在子孙轴、通配符概率较高时。这是由于 XSIEQ 对各个 XPath 式的谓词计算和缓冲区的管理都是分别进行的。

5 结论

通过定义一种修改了的 PDA，本文给出了带谓词的 XPath 式查询引擎的设计与实现，即 XSIEQ。它最大的特色是对谓词的立即计算和结果的即时输出，特别适用于对查询实时性要求较高的系统以及对不间断数据的连续查询系统。

XSIEQ 通过前缀共享构造 NFA，解决了 XPath 式状态共享问题；然而，查询中可能存在大量相同的谓词计算，并且，对于含有多个谓词的查询，谓词计算仍是影响性能的瓶颈。下一步的工作是考虑如何共享处理谓词计算和缓冲区，以进一步提高查询性能。

参考文献

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

2 Green T J, Miklau G, et al. Processing XML Streams with Deterministic Automata[C]. Proc. of the 19th ICDT, 2003-01.

3 Olteanu D, Kiesling T, Bry F. An Evaluation of Regular Path Expressions with Qualifiers Against XML Streams[C]. Proc. of the 19th ICDE, Bangalore, India, 2003-03.

4 Gupta A, Suciu D. Stream Processing of XPath Queries with Predicates[C]. Proc. of SIGMOD'03, San Diego, CA, 2003-06.

5 Schmidt A, Waas F. XMark: A Benchmark for XML Data Management[C]. Proc. of the 28th VLDB, Hong Kong, China, 2002.