2007 年 3 月 第 3 期
Vol. 28 No. 3 2007

一种逐层提升缓冲的 XML 流查询自动机

张 昱[1,2], 吴 年[1]

[1](中国科学技术大学 计算机科学技术系, 安徽 合肥 230027)
[2](中国科学院 计算机科学重点实验室, 北京 100080)
E-mail: yuzhang@ustc.edu.cn

摘 要: 如何在 XML 流上高效地执行大量 XPath 查询是当今研究的热点. 特别在管道处理等应用中还希望在解析流的同时尽早地输出查询结果. 定义了基本 XSIEQ (XML Stream Query with Immediate Evaluation) 机. 它是一个 XML 流查询框架, 是被索引化的、基于栈的自动机; 在其上可以扩展应用多种 XPath 查询算法. 在基本 XSIEQ 机上, 提出一种逐层提升缓冲 (promoting buffer, 简称 PBuf) 的查询算法, 形式地定义了基于 PBuf 的 XSIEQ 机并进行了实现和测试. 实验结果表明, 提出的方法能够支持复杂的 XPath 查询, 在执行效率方面优于传统算法.

关 键 词: XML 流; 状态分类; 索引; XPath; 谓词计算; 分层缓冲
中图分类号: TP311 **文献标识码**: A **文 章 编 号**: 1000-1220(2007)03-0456-06

XML Stream Query Automaton with Promoting Layered Buffer

ZHANG Yu[1,2], WU Nian[1]

[1](*Department of Computer Science & Technology, University of Science & Technology of China, Hefei* 230027, *China*)
[2](*Lab of Computer Science, ISCAS, Beijing* 100080, *China*)

Abstract: Much research has been done in evaluating massive XPaths set over an XML stream efficiently. In some applications especially pipelines, it is further required to output the results while parsing XML stream. In this paper, a basic XSIEQ (XML Stream Query with Immediate Evaluation) machine is defined, which is a framework of XML stream query and a kind of indexed automata based on stack. Various kinds of algorithms on XPath evaluation can be applied to extend the basic XSIEQ. Moreover, an algorithm based on promoting layered buffer (PBuf for short) is put forward. The XSIEQ machine based on PBuf is formally defined and also implemented and tested. Experimental results show that XSIEQ based on PBuf supports the complex XPath and outperforms the former work in efficiency.

Key words: XML Stream; state labelling; index; XPath; predicate evaluation; layered buffer

1 引 言

近年来出现一些 XML (eXtensible Markup Language) 流应用, 如信息选择分发[1-3]、数据集成[4]、管道处理[5]、连续查询[6]等. 其中的关键是如何高效地查询 XML 流以进行分发、传递或转换等. 在 XML 流上的 XPath 查询计算多数是根据 XPath 式构造自动机[1,2,8,9]或树模式[3], 通过一遍扫描流完成对流的解析和查询. 如果 XPath 式含有谓词, 则还需要有缓冲机制管理谓词尚未确定的候选结果[1]. 现有的流查询算法在支持的 XPath 式、自动机构造、索引和缓冲机制等方面有着不同的特点和局限性[10].

本文关注的重点是:

1) 如何对 XML 流进行复杂 XPath 式的高效计算, 所支持的 XPath 式片段包括子孙轴、通配以及含相对 XPath 式的复杂谓词(包括嵌套谓词);

2) 如何在解析 XML 流的同时尽早地输出已匹配的 XML 片段, 而不是在流文档解析完毕后才集中处理再输出. 第 2 点对于大型 XML 流的管道处理与连续查询尤为重要.

本文的主要贡献如下:

1) 定义基本 XSIEQ (XML Stream Query with Immediate Evaluation) 机, 它是状态被类型标记并索引化的自动机. 它提供了一个 XML 流查询框架, 在其上可以扩展实施多种 XPath 查询算法.

2) 提出并实现建立在基本 XSIEQ 机上的逐层提升缓冲 (Promoting Layered Buffer, 简称 PBuf) 查询算法. 该算法在 XSIEQ 机运行回溯到含谓词的位置步对应的自动机状态时, 才处理候选结果: 或加入到 XPath 分层缓冲区中, 或提升它在缓冲区中的层次, 或确定是匹配还是不匹配等.

3) 通过对大量的 XPath 式和 XML 文档测试用例实验, 结果表明基于 PBuf 的 XSIEQ 机优于 YFilter[2].

2 基本 XSIEQ 机

收稿日期: 2005-12-22 基金项目: 国家自然科学基金项目 (60473068) 资助; 中国科学院计算机科学重点实验室开放课题基金项目 (SYSKF0502) 资助. 作者简介: 张 昱, 女, 1972 年生, 博士, 副教授, 主要研究方向为 XML 数据处理、程序设计语言理论和实现技术; 吴 年, 男, 1976 年生, 硕士研究生, 主要研究方向为 XML 数据处理.

[1]对一个 XPath 式 P, P 的主 XPath 式是指去除 P 中所有谓词后剩余的部分. 主 XPath 式匹配的 XML 节点称为候选结果.

2.1　支持的 XPath 式

XPath 是 W3C 发布的一种寻址 XML 文档的路径表达式语言. 一个 XPath 式由若干个位置步组成, 每个位置步由轴、节点测试和可选的若干谓词组成. 常见的轴包括孩子轴'/'、子孙轴'//'、属性轴'@'和自身轴'.'等; 节点测试可以是通配'*'; 谓词中允许有各种表达式存在.

```
[1]P: := /E |//E
[2]E: := E/E |E//E |E[Q] |label |text() |* |@* |@label
[3]Q: := E |E Op Const |Q and Q |Q or Q |not(Q) |func(Q*)
[4]Op: := < |≤ |> |≥ |= |≠ |* |div |+ |-
```

图 1　XSIEQ 支持的 XPath 片段

Fig. 1　XPath fragment supported by XSIEQ

XSIEQ 不仅支持 YFilter 所支持的所有 XPath 式特性, 而且支持更复杂的谓词表达式, 如逻辑和算术运算、函数等. 图 1 给出了 XSIEQ 支持的 XPath 式的形式定义.

2.2　基本 XSIEQ 机的构造

为支持谓词计算, 一个基本 XSIEQ 机是一个带栈的 NFA 或 Lazy DFA, 本文主要介绍 NFA. 它的输入是解析 XML 流产生的 SAX 事件: startDocument、startElement、character、endElement、endDocument; 输出是匹配的 XML 片段.

首先按照 YFilter 中的方法构造初始的基本 XSIEQ 机, 即将各查询 XPath 式中的主 XPath 式和谓词中的 XPath 式提取出来, 利用前缀共享的方法增量式地构造为一个 NFA.

然后对 NFA 中的以下特殊状态进行类型标记.

定义 1　结果状态是匹配主 XPath 式的 NFA 状态, 即主 XPath 式的最后一个位置步到达的状态. 叶子状态是匹配谓词中的 XPath 式的 NFA 状态. 分支状态是某 XPath 式对应的主路径上具有的、分支到谓词中 XPath 式的 NFA 状态.

显然, 到达结果状态(startElement 事件)可以启动相关 XPath 式候选结果的收集; 回溯到结果状态(endElement 事件)必须结束相关 XPath 式候选结果的收集; 到达叶子状态可以对相关的谓词进行计算; 回溯到分支状态时, 可以确定相关 XPath 式在该分支下的各谓词的最终计算状态.

如果在这些状态中保存相关的 XPath 式或谓词的引用信息, 则在 XSIEQ 机运行时, 就可以根据当前状态的类型和相关的 XPath 或谓词集, 快速实施相关的操作(如收集候选结果、计算谓词、确定查询结果等). 这里所保存的 XPath 式或谓词集即为状态的索引表.

定义 2　假设为 XPath 式集合 P 构造的 NFA 为 A, 对 A 中的每一状态 s, 可以建立如下三个索引表:

1) $LR(s)$: $LR(s) \subseteq P$. $\forall p \in P$, 若 s 是 p 的主 XPath 式所匹配的状态, 则 $p \in LR(s)$. 仅当 s 是结果状态时, $LR(s)$ 非空.

2) $LP(s)$: 是一个谓词表. $\forall p \in P$, 对 p 中的每一谓词 pr, 若 s 是 pr 中的 XPath 式所匹配的状态, 则 $pr \in LP(s)$. 仅当 s 是叶子状态时, $LP(s)$ 非空.

3) $LB(s)$: 是一个谓词表. $\forall p \in P$, 对 p 中每一位置步 ls 以及 ls 中的每一谓词 pr, 若 s 是 ls 到达的状态, 则 $pr \in LB(s)$. 仅当 s 是分支状态时, $LB(s)$ 非空.

例 1　图 2 是 *XPath* 式 P_1: /a[.//b=2][e=3]/c[b]/d 对应的索引化的 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.

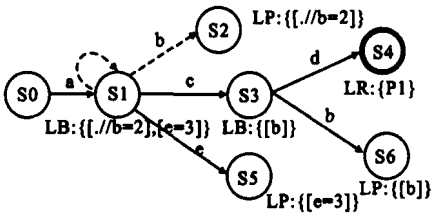

图 2　P_1 对应的索引化的 NFA

Fig. 2　The indexed NFA corresponding to P_1

2.3　基本 XSIEQ 机的定义

根据上节的构造思想, 我们给出基本 XSIEQ 机的形式定义, 它描述了 XSIEQ 机的静态特性.

定义 3　一个非确定的基本 XSIEQ 机 A 定义为

$$A = (Q^s, \sum, \delta, q_0^s, F, PF, B, S_s)$$

其中:

- Q^s 表示 NFA 状态集合;
- $\sum$ 为输入字符集合;
- q_0^s 是 A 的初始 NFA 状态, 也称根状态;
- δ 表示状态转换函数集合, 至少有 NFA 状态转换函数, 即 $t_{forward}: Q^s \times \sum \rightarrow 2^{Q^s}$;
- $F \subseteq Q^s$ 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;
- $\forall q^s \in Q^s$, 称 q^s 为 A 的一个 NFA 状态, $LR(q^s)$、$LP(q^s)$、$LB(q^s)$ 表示 q^s 的三个索引表(见定义 2);
- S_s 是 NFA 状态栈, S_s 的栈帧是 Q^s 的子集.

XSIEQ 机的动态特性由 SAX 事件处理描述, 需要反映 NFA 状态转换、谓词的计算、候选结果的收集、对收集到的候选结果的处理(谓词状态未确定时的缓存、确认后的输出或清除)等. 其中, NFA 状态转换以及谓词计算和候选结果收集的时机是可以固定的; 但是候选结果的缓冲机制可以有多种, 不同的机制会导致查询的不同时空代价, 不过缓冲机制的实施都将分散到对当前 NFA 状态集中各状态的原子处理(atomic process)中进行.

为此, 我们定义如下的 SAX 事件处理框架作为基本 XSIEQ 机的动态特性:

```
startDocument()
  push(Ss, {q0s}); other initialization
startElement(a)
  qsset = {}; //current NFA state set
  for each qs in getTop(Ss)
      merge tforward(qs, a) into qsset
    push(Ss, qsset);
    if exist qs in qsset and qs∈F
      start to collect candidate results
    for each qs in qsset{
  if(qs∈PF)
```

```
        evaluate each pred in LP(q^s)
        other atomic process
    }
character(str)
    do collection & predicate evaluation
endElement(a)
    q^s_set = pop(S_s);
    if exist q^s in q^s_set and q^s ∈ F
        end the collection
    for each q^s in q^s_set{
        other atomic process
        if(q^s ∈ B)
        reset status of each pred in LB(q^s)
        other atomic process
    }
endDocument()
    pop(S_s);
```

由此可见, 基本 XSIEQ 机是一种 XML 流查询自动机框架, 在它上面需要扩展具体的缓冲查询算法, 从而得到不同的 XSIEQ 机. 建立在基本 XSIEQ 机上的各 XSIEQ 机具有相同的静态结构和部分动态特性.

3 基于 PBuf 查询的 XSIEQ 机

在文[11]中我们提出即时输出(immediate output, 简称 IOut)的缓冲查询算法, 即一旦谓词状态发生变更, 就立即处理候选结果缓冲区. 这种方案会频繁操作缓冲区, 使得当同时查询的 XPath 式数目接近 1000 或谓词数偏多时, 查询效率显著降低. 本节给出一种新的称为逐层提升缓冲(PBuf)的查询算法.

3.1 PBuf 查询思想

定义 4 假设 XSIEQ 机 A 中包含查询 XPath 式 P, P 有 m ($m>0$) 个带谓词的位置步, 这些位置步按在 P 中的出现次序自左至右地标记为 $ls_0, \cdots, ls_{m-1}$, 它们分别对应于 A 中的 m 个分支状态. 称 ls_0 对应的分支状态是 P 的最左分支状态, ls_{m-1} 对应的分支状态是 P 的最右分支状态.

PBuf 查询方案的设计是基于这样的一个事实: 对含谓词的 XPath 式, 当 XSIEQ 机运行回溯到该 XPath 式的最左分支状态时, 一定能确认在此前收集到的该 XPath 式的候选结果是否匹配. 下面重点讨论 PBuf 中候选结果的收集、处置和缓冲机制.

3.1.1 收集机制

当不同结果状态匹配到不同深度的 XML 节点时, 它们需要收集的 XML 内容可能存在嵌套关系. 为节约空间, 对相互嵌套的多个候选结果进行统一收集并保存到收集区中.

定义 5 匹配某结果状态的 XML 内容的收集过程从 XSIEQ 机运行到达该结果状态开始, 直至回溯到该结果状态时终止, 该过程称为一次单个收集过程. 当 XSIEQ 机运行处在收集时, 称收集区是活跃的. 收集区从开始活跃到变为不活跃的过程, 称为一次完整收集过程. 一次完整收集过程可能包含多次单个收集过程.

在基于 PBuf 的 XSIEQ 机中, 收集区保存一次完整收集过程所收集的 XML 内容的串值 content. 为区分收集区中各次单个收集过程, 我们为 XML 文档中的各元素、属性和文本节点按先序遍历的次序进行编号, 并为每一单个收集过程建立标识其收集内容在 content 中位置的标识项. 标识项列表 items, content 以及 startID (收集区的起始 XML 节点编号)等形成一个收集区的基本信息.

一般地, 一次单个收集过程所收集的内容可用三元组<xmlID, startOffset, endOffset>标识, 其中 xmlID 为对应的 XML 节点编号, 后两项为所收集的内容在 content 中的起止位置. 当收集的是复杂 XML 元素的 text 时, 文本内容由该元素及其子孙元素的文本顺次组成, 它们在 content 中可能是不连续的, 即由 content 中的多个离散子串组成, 这时可以用五元组<xmlID, startOffset, endOffset, next, itemNum>来标识其中的每一子串, next 指向下一个子串, itemNum 为组成该文本节点内容的子串数.

3.1.2 候选结果的处置与缓冲机制

为降低对候选结果缓冲区的操作频率, PBuf 算法在 XSIEQ 机运行回溯到分支状态时检测该状态的 LB 表中各谓词的计算状态, 确定候选结果是被保留还是被清除. 这样, 候选结果的确认是按 XPath 式对应的分支状态逐层进行的. 考虑到在运行的某一瞬时, 某 XPath 式的各候选结果的确认层次可能不一致, 故为每一 XPath 式引入一个分层缓冲区缓存其候选结果(为节约空间, 实际缓存的是对应的 XML 节点编号).

定义 6 假设 XPath 式 P 含有 m 个带谓词的位置步, 自左至右依次标记为 $ls_0, \cdots, ls_{m-1}$, 则 P 的缓冲区含有 m 层, 第 i ($0 \le i \le m-1$) 层保存的候选结果满足 ls_i 到 ls_{m-1} 中的所有谓词.

在缓冲区上有以下几种基本操作:

1) add 操作: 当回溯到 ls_{m-1} 对应的分支状态(即 P 的最右分支状态)时, 调用 add 操作将检查 ls_{m-1} 中各谓词的状态, 若均为真则将相关的候选结果项添加到缓冲区的第 $m-1$ 层;

2) promote 操作: 当回溯到 ls_i ($0 \le i < m-1$) 对应的分支状态时(设回溯到的 XML 节点编号为 xid), 调用 promote 操作将检查 ls_i 中各谓词的状态, 若均为真则将缓冲区的第 $i+1$ 层中 XML 节点编号大于 xid 的候选结果项提升到第 i 层, 同时清除第 $i+1$ 层;

3) accept 操作: 当回溯到 ls_0 对应的分支状态(即 P 的最左分支状态)时, 在执行 promote 操作后, 可调用 accept 操作输出缓冲区中第 0 层的数据项, 即最终匹配的数据项.

由第 2 节知, 在回溯到结果状态时即可完成候选结果的收集. 而按上述思想, 对含谓词的 XPath 式 P, 其候选结果只有在回溯到分支状态时才能被处理. 为此, PBuf 算法引入缓冲栈 S_b 暂存并传递候选结果, S_b 与 NFA 状态栈 S_s 同步, S_b 的栈帧为一哈希表, key 是 NFA 状态集, 值为对应的候选结果的 XML 编号集. 对 S_b 的操作主要有:

1) 在启动单个收集过程时, 将当前 NFA 状态集和对应的候选结果 XML 编号组成二元组加入到栈顶;

2) 在状态回溯时, 出栈并将弹出内容合并到的栈顶中, 从而使候选结果向上传递;

3) 在回溯到某 XPath 式的最右分支状态时, 从 S_b 栈顶取得相应的候选结果执行缓冲区的 add 操作.

3.2　基于 PBuf 的 XSIEQ 机的形式定义

定义 7. 一个基于提升缓冲(PBuf)的不确定 XSIEQ 机 $A_{PBuf}=(Q^s, Q^p, Q^b, D, I, \sum, \delta, q_0^s, q_0^p, q_0^b, F, PF, B, S_s, S_b), Q^s, \sum, q_0^s, F, PF, B$ 含义同定义 3, 其中

· Q^p 是所有谓词状态的集合; Q^b 是 XPath 缓冲区的状态集合;

· D 是收集区集合; I 是 XML 节点编号的集合; $P(I)$ 是 I 的幂集;

· q_0^b、q_0^p 是缓冲区和谓词的初始状态;

· S_b 是缓冲栈, 栈帧为一组二元组, 二元组类型为 $2^{Q^s}\times I$, 即含结果状态的 NFA 状态集与相应的 XML 节点编号;

· $\delta=\{t_{forward}, t_{collect}^d, t_{accept}^d, t_{eval}^p, t_{reset}^p, t_{add}^b, t_{promote}^b, t_{accept}^b\}$ 是一组状态转换函数集合, 它们是满足以下定义的偏序函数:

$t_{forward}: Q^s\times\sum\to 2^{Q^s}$

$t_{collect}^d: D\times\sum\times\{SE, CH, EE\}\to D$

$t_{accept}^d: D\to P(I)$

$t_{eval}^p: Q^p\times PF\times\sum\to Q^p$

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

$t_{add}^b: P(2^{Q^s}\times I)\times Q^b\times Q^p\to Q^b$

$t_{promote}^b: Q^b\times Q^s\times Q^p\to Q^b$

$t_{accept}^b: Q^b\times Q^s\to P(I)\times Q^b$

当实现 n 个 XPath 式的查询时, 每个 XPath 式 $P_i(1\le i\le n)$ 有自己的谓词状态集合 $Q^p(P_i)$ 和缓冲区状态集合 $Q^b(P_i)$, 则 $Q^p=Q^p(P_1)\cup\cdots\cup Q^p(P_n), Q^b=Q^b(P_1)\cup\cdots\cup Q^b(P_n)$; 若某个 XPath 式 P 没有谓词, 则 $Q^p(P)$ 和 $Q^b(P)$ 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(简称谓词状态), 0 表示谓词未确定和确定为假, 1 表示确定为真.

$t_{forward}$ 是一个 NFA 状态的状态转换操作, 为便于表示, 我们用 $t_{forward}^s$ 表示一组 NFA 状态的状态转换操作, 即 $t_{forward}^s: 2^{Q^s}\times\sum\to 2^{Q^s}$. $t_{collect}^d$ 为候选结果的收集操作, SE, CH, EE 等分别标识收集的是元素起始标签、字符文本、元素结束标签; t_{accept}^d 是回溯到结果状态时, 输出当前收集到的、与不含谓词的 XPath 式相匹配的结果. t_{eval}^p 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_{reset}^p 表示回溯到分支状态时对谓词状态进行复位. t_{add}^b、$t_{promote}^b$ 和 t_{accept}^b 是对 XPath 缓冲区的操作, 含义见定义 6.

基于 PBuf 的 XSIEQ 机在运行时维持一个当前状态 $q=(q_{set}^s, q^b, q^p)$、当前的收集区 d、栈 S_s 和 S_b, 其中 q_{set}^s 是一个 NFA 状态集, q_b 和 q_p 分别为 XPath 缓冲区和谓词的状态; 初始时 $q=(\{q_0^s\}, q_0^b, q_0^p)$.

下面是基于 PBuf 的 XSIEQ 机的 SAX 事件处理规则, 其中 n_{active} 表示当前激活的单个收集过程的数目.

```
startDocument()
  q_set^s ← {q_0^s}; q^p = q_0^p; q^b = q_0^b;
  push(S_s, q_set^s); resetDMgr();
  n_active = 0; d ← ∅;
startElement(a)
  push(S_b, ∅);
  q_set^s ← t_forward^s(q_set^s, a); push(S_s, q_set^s);
  if (exist q^s in q_set^s and q^s ∈ F) {
    n_active++;
    addtop(S_b, <q_set^s, getID(a)>);
  }
  if (n_active > 0) d ← t_collect^d(d, a, SE);
  for each q^s in q_set^s
    if (q^s ∈ PF) q^p ← t_eval^p(q^p, q^s, a);
character(str)
  for each q^s in q_set^s {
    if (q^s ∈ PF) q^p ← t_eval^p(q^p, q^s, a);
    if (n_active > 0) d ← t_collect^d(d, str, CH);
  }
endElement(a)
  q_set^s = pop(S_s);
  if (n_active > 0) d ← t_collect^d(d, a, EE);
    if (exist q^s in q_set^s and q^s ∈ F) {
    n_active--;
    t_accept^d(d);
  }
  for each q^s in q_set^s
    if (q^s ∈ B) {
      if (q^s is the rightest branch state of an xpath)
        q^b ← t_add^b(fetchCandidates(S_b, q^s), q^b, q^p);
      else q^b ← t_promote^b(q^b, q^s, q^p);
      if (q^s is the leftest branch state of an xpath)
        t_accept^b(q^b);
      q^p ← t_reset^p(q^p, q^s);
    }
  bs = pop(S_b); addTop(S_b, bs);
endDocument()
  pop(S_s);
```

3.3　运行举例

例 2 对 P_1: /a[.//b=2][e=3]/c[b]/d 构造 XSIEQ 机(图 2)并按 PBuf 算法执行, 待查询的 XML 片段为:

〈a〉〈c〉〈b〉3〈/b〉〈d/〉〈/c〉〈e〉3〈/e〉〈b〉2〈/b〉〈/a〉

表 1　运行时的状态转换和操作

Table 1　State transitions and operations in runtime

XML	start	<a>	<c>		3		<d>	</d>	</c>	<e>	3	</e>		2		
Id		1	2	3	4	3	5	5	2	6	7	6	8	9	8	1
q^p	000	000	000	001	001	001	001	001	000	000	010	010	010	110	110	000
q^b 0																5
q^b 1									5	5	5	5	5	5	5	
操作				E	E		C	C	AR		E			E		POR

注: E-t_{eval}^p, R-t_{reset}^p, C-$t_{collect}^d$, A-t_{add}^b, P-$t_{promote}^b$, O-t_{accept}^b

则运行时的各种状态转换及操作如表 1, 状态栈 S_s 和缓冲栈 S_b 的变化如图 3. 假设 S_s 的栈顶为 q_{set}^s. P_1 中有三个谓

词, 自左至右依次编号为 0～2, 初始 q^p 为"000"; P_1 缓冲区有 0 和 1 两层

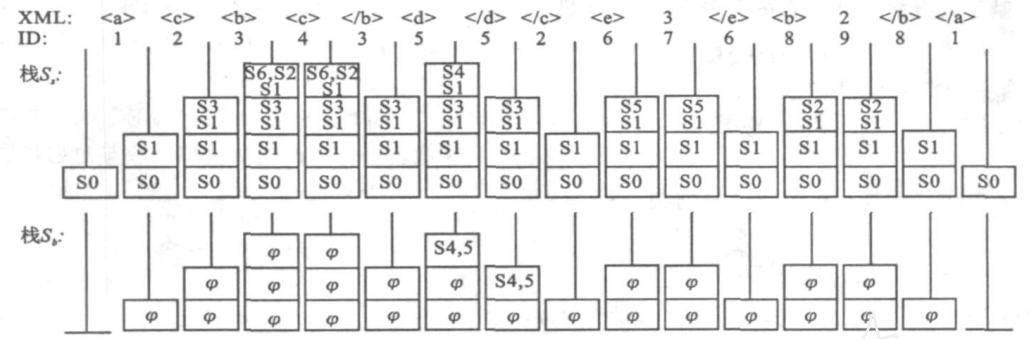

图 3 运行时的栈

Fig 3 Stacks in runtime

在第1个startElement(b)中谓词2计算为真, q^p 变为"001" 在 startElement(d)中由于到达的状态集 q^s_{set} 中包含结果状态 S4, 故开始收集候选结果, 并将当前状态和开始收集的 XML 节点编号 5 组成二元组(S4, 5)加入到 S_b 栈顶 endElement(d)时结束候选结果的收集, 并在 S_b 退栈时将原栈顶内容

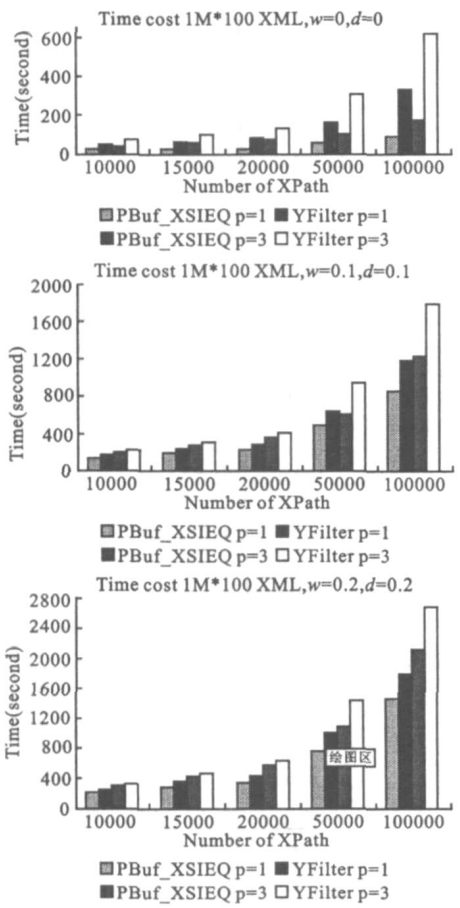

图 4 查询时间比较

Fig 4 The time cost

(S4, 5)合并到新栈顶 endElement(c)时, 由于 q_{sset} 中包含分支状态 S3 且由此分支出去的谓词 2 状态为真, 故将 S_b 栈顶暂存的候选结果添加到 P_1 缓冲区的第 1 层, 同时更新 S_b 栈顶内容, 再将 q^p 复位为"000". 在 startElement(e)之后的 character(3)中谓词 1 计算为真; 在 character(2)中谓词 0 计算为真 在 endElement(a)中当前由 S_s 出栈的 q^s_{set} 中包含分支状态 S1, 且分支出去的谓词 0 和 1 的状态均为真, 故对 P_1 缓冲区执行提升操作, 将第 1 层提升到第 0 层; 此时形成最终的匹配结果并输出

4 实验结果与分析

笔者用 Java 实现了基于 PBuf 的不确定 XSIEQ 机(记为 PBuf-XSIEQ), 并与 YFilter 进行时间性能对比测试 实验平台为 Windows 2000 操作系统, CPU 为 P4, 主频为 2 8G, 内存为 512M. 我们使用 XMark[12] 生成 100 个 1M 大小的 XML 文件供测试用, 使用 YFilter 自带的 XPath 式生成工具分别生成不同特性组合的 XPath 查询集合. 实验考核的因素包括 XPath 式的个数、单个 XPath 所含的谓词数 p、XPath 式中通配的概率 w 和子孙轴的概率 d 等.

图 4 为实验的部分结果, 其中的时间为查询 100 * 1M 大小的 XML 文件共花的时间, 忽略了 XPath 式解析和自动机构造的时间 从中可以看出, PBuf-XSIEQ 和 YFilter 的查询时间均随 XPath 式个数的增加而增长; 在 XPath 查询集具有同等特性时, PBuf-XSIEQ 的查询时间比 YFilter 要快 10% 以上, 特别是在通配、子孙轴概率均为 0 时, PBuf-XSIEQ 的查询时间不到 YFilter 的一半. 从图中还可以看出, 单个 XPath 式所含谓词数的增长也会引起查询时间的增长; 而通配、子孙轴概率的提高则会引起查询时间的急剧增加, 这是因为通配、子孙轴的增加导致自动机状态转换次数的增加, 并增加了需要进行谓词计算等处理的状态数 与 YFilter 相比, PBuf-XSIEQ 能较早地输出查询结果

5 相关工作

YFilter[2] 在解析 XML 时缓存所有的候选结果和谓词中 XPath 式匹配的节点; 在文档解析结束(endDocument 事件)时, 再通过后处理获得结果并输出 当 XML 文档很大时, YFilter 不能很快地输出结果并且需要大量的空间缓存 XMLTK[8] 只能支持在当前位置步计算的谓词(即只能包含

对当前元素属性或文本的引用), 实现有限的谓词立即计算和结果即时输出. XMLTK 是基于 Lazy DFA 的, 我们根据它的思想同时实现了基于 Lazy DFA 的 XSIEQ 机 XPush[9]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. 它支持多种谓词以及谓词计算的共享处理. SPEX[13]给出 XPath 的反向轴重写为前向轴的方法, 这使现有的支持前向轴的 XML 流查询系统能比较容易地支持反向轴. [14]研究 XML 流查询中的缓冲问题, 指出可以避免空间是查询大小的指数级关系.

6　结　论

本文提出一种提升缓冲的 XSIEQ 机来解决带谓词、通配、子孙轴等的复杂 XPath 式在 XML 数据流上的高效查询. 因篇幅所限, 本文没有一一陈述 XSIEQ 机中的所有细节, 如嵌套谓词的处理、NFA 状态在不同 XPath 路径上的多重匹配等. 目前实现的 XSIEQ 机不仅在查询的总时间上优于 YFilter, 而且能在解析 XML 流时尽早地输出查询结果, 特别适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.

我们也注意到目前实现的 XSIEQ 机在空间上还有很多改进的余地, 这可以作为下一步的工作之一. 此外, 我们还将开展: 1) 谓词的共享计算; 2)) 支持 XQuery 查询.

References:

[1] Altinel M, Franklin M. Efficient filtering of XML documents for selective dissemination of information [C]. In Proc. 26th VLDB 2000, 53-64.

[2] Diao Y, Altinel M, Franklin M, et al. Path sharing and predicate evaluation for high-performance XML filtering [J/OL]. ACM Transations on Database Systems, 2003, 28(4): 467-516. http://yfilter.cs.berkeley.edu/code-release.htm

[3] Chan C Y, Felber P, Garofalakis M N, et al. Efficient filtering of XML documents with XPath expressions[J]. In VLDB Journal, Special Issue on XML, 2002, 11(4): 354-379.

[4] Ives Z G, et al. An XML query engine for network-bound data[J]. In VLDB Journal, Special Issue on XML, 2002, 11(4): 380-402.

[5] McGrath S. XPipe. Available at [EB/OL]. http://xpipe.sourceforge.net/.

[6] Chen J, DeWitt D J, Tian F, et al. NiagaraCQ: A scalable continuous query system for internet databases[C]. In Proc of the 2000 ACM SIGMOD Intl Conf. on Management of Data, May 2000, 379-390.

[7] Clark J. XML path language (XPath). 1999. Available from the W3C [EB/OL]. http://www.w3.org/TR/XPath.

[8] Green T J, Miklau G, etal. Processing XML streams with deterministic Automata and Stream Indexes [EB/OL]. In ACM TODS, 2004, 29(4): 752-788. http://www.cs.washington.edu/homes/suciu/XMLTK/xmltk-v2.0.zip

[9] Gupta A, Suciu D. Stream processing of XPath queries with predicates[C]. SIGMOD 2003, San Diego, CA. June 2003: 419-430.

[10] Zhang Y, Wu N. Filter and query technique of XML stream [J]. Computer Science, 2004, 31(10A): 458-461.

[11] Wu N, Zhang Y. Immediate processing of XPath query with predicates[J]. Computer Engineering, 2006, 32(13): 58-60.

[12] Albrecht Schmidt, Florian Waas, etal. XMark: A benchmark for XML data Management [C]. Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002: 974-985.

[13] Dan Olteanu. Evaluation of XPath queries against XML streams [D]. Institute for Informatics, University of Munich, 2005.

[14] Ziv BarYossef, Marcus Fontoura, Vanja Josifovski. Buffering in query evaluation over XML streams [C]. In Proceedings of. PODS, 2005: 216-227.

附中文参考文献:

[10] 张　昱, 吴　年. XML 数据流的过滤与查询技术[J]. 计算机科学, 2004, 31(10A): 458-461.

[11] 吴　年, 张　昱. 带谓词的 XPath 查询的即时处理[J]. 计算机工程, 2006, 32(13): 58-60.