XSIEQ—一种立即计算的 XML 流查询系统

张 昱，吴 年

(中国科学技术大学 计算机科学技术系，安徽 合肥 230027)
(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：XSIEQ 是一种立即计算谓词并即时输出的 XML 流查询系统. 它利用前缀共享的方法由多个 XPath 式构造一个 NFA，并对 NFA 状态进行分类和添加索引，使得在运行时能快速确定谓词计算和数据缓存等的时机. XSIEQ 还提供在运行时惰性地构造 DFA 进行查询. 陈述了 XSIEQ 的查询机制以及多重匹配问题的解决方案. 最后给出了 XSIEQ 的两种自动机和 YFilter 的查询性能对比及分析.

关 键 词：XML 流；状态分类；索引；谓词计算；多重匹配
中图分类号：TP311 **文献标识码**：A **文 章 编 号**：1000-1220(2006)08-1514-05

XSIEQ—an XML Stream Query System with Immediate Evaluation

ZHANG Yu，WU Nian

(*Department of Computer Science & Technology*，*University of Science & Technology of China*，*Hefei* 230027，*China*)
(*Lab. of Computer Science*，*Institute of Software*，*Chinese Academy of Sciences*，*Beijing* 100080，*China*)

Abstract：XSIEQ(XML Stream Query with Immediate Evaluation) can evaluate predicates immediately and output in time. In XSIEQ，all XPath expressions are converted into a single NFA by prefix sharing，and the NFA states are labeled with type and index，so the opportunity of predicate evaluation and data cache can be quickly decided in runtime. Moreover，a lazy DFA is also implemented in XSIEQ. The query mechanisms and solution of multiple matching are represented. In the end，the query performance comparison among two automata of XSIEQ and YFilter are given and analyzed.

Key words：XML stream；state labelling；index；predicate evaluation；multiple matching

1 引 言

在企业内和企业间的系统集成中，采用面向消息的中间件是一种有前途的方法. 为了满足平台无关性，消息可以选择 XML 来描述. 这类 XML 消息代理系统的主要任务是实时地传递、查询或转换 XML 流信息，其中的关键问题是如何根据条件高效地查询 XML 流. 基本的查询条件通常是一组 XPath 式. 一个 XPath 式由若干位置步组成，每一位置步包含轴、节点测试和可选的谓词列表. XPath1.0 中共定义有 13 种轴，节点测试允许是通配′*′. XPath 式中去除谓词后剩余的部分称为主 XPath 式，它确定查询匹配的候选结果. XML 流查询的主流做法是根据 XPath 式构造查询自动机，如 YFilter[1,2]将 XPath 式构造为 NFA(Non-deterministic Finite Automaton)；XMLTK[3]从 NFA 构造 Lazy DFA(Deterministic Finite Automaton). XPath 式中的谓词、轴和通配等增加了 XPath 查询计算的难度和复杂度.

YFilter 支持带子孙轴′//′、通配′*′和部分谓词(包括存在性谓词、索引谓词、简单的嵌套谓词等)的 XPath 查询. 它将对同一 XML 文档查询的多个 XPath 式(包括主 XPath 式和谓词中的 XPath 式)连接合并为一个 NFA；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；在解析结束后通过后处理获得最终的查询结果. YFilter 不支持在解析 XML 时立即计算谓词并即时输出结果. 这对管道处理等实时性要求高的查询应用来说是欠缺的.

XMLTK 根据 XPath 式构造 NFA，然后在运行时惰性地从 NFA 建立 DFA，即 Lazy DFA，并引入流索引来提高查询效率. 它支持子孙轴、通配以及能在当前位置步计算的谓词，实现有限的立即计算和即时输出. YFilter 和 XMLTK 都是开源的，前者[4]用 Java 实现，后者[5]用 C++实现.

XSIEQ(XML Stream Query with Immediate Evaluation)是一个旨在支持复杂查询的、高效的、能立即计算的 XML 流查询研究项目. 当前它采用一种新的索引结构和缓冲机制实现两种支持多个 XPath 查询的自动机，即 NFA 和 Lazy DFA.

2 XSIEQ 查询自动机的构造

2.1 支持的 XPath 式

收稿日期：2005-05-10 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，副教授，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，研究方向为 XML 数据处理.

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.

[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

2.2　NFA 的构造

XSIEQ 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(见算法 1). 在构造初始,NFA 只有一个初始(根)状态. 对于谓词(包括嵌套谓词)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.

算法 1. *IncNFAGen(xp, NFA, root)*: 增量式 NFA 构造

输入:XPath *xp*, *NFA*, *xp* 对应的初始 NFA 状态 *root*

输出:*NFA*

```
(1) current = root; next = null;
(2) for each location step ls in xp
(3)   if ( ls is self axis )
(4)     next = current;
(5)   else if (current.containTransition(ls))
(6)     next = current.move(ls);
(7)   else next = current.creatEdge(ls);
(8)   lspreds = ls.getPredicates( );
(9)   for each predication pred in lspreds
(10)      nestxps = pred.getNestXPaths();
(11)      for each XPath nestxp in nestxps
(12)        IncNFAGen(nestxp, NFA, next);
(13)   current = next;
```

例 1. 对 XPath 式

P_1:/a[.//b=2]//c[e]/d

P_2:/a/*[c]/b

P_3:/a//b[.//d > 3]/c

P_4:/a/*[b]/c

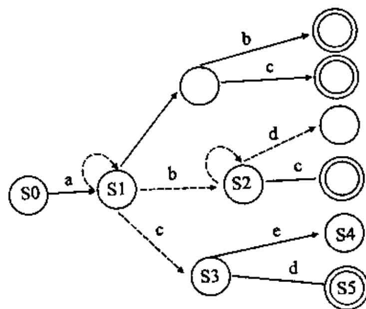


图 2　例 1 中 P_1～P_4 对应的 NFA

按算法 1 构造的 NFA 如图 2. 其中状态用圆圈表示,双圈表示接受状态;弧代表一个状态转换,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.

2.3　Lazy DFA 的构造

XSIEQ 在运行时惰性地由 NFA 构造 DFA,即 Lazy DFA. 此时,一个 DFA 状态对应 NFA 的一个状态集. NFA 和 DFA 的状态转换由解析 XML 文档产生的 SAX 事件来触发,主要包括 startDocument( ), startElement(*a*), endElement(*a*)和 endDocument( )四类事件. 为简化表示,本文将属性类似地看成元素,故上述的 *a* 既指元素标签,也指属性标签. 在自动机运行时,使用栈保存从根状态到当前状态所经过的状态信息. 算法 2 和算法 3 是两类自动机的状态转换处理.

算法 2. NFA 的状态转换

```
s0 为初态, stack 为栈,栈帧为 NFA 状态集

startDocument( )
(1) Push({s0}, stack);

startElement(a)
(1) sset = GetTop(stack);
(2) newset = {};
(3) for each s in sset
(4)   if s.containTransition(a)
(5)     add state s.move(a) to newset;
(6) Push( newset, stack);

endElement(a)
(1) sset=Pop(stack);

endDocument()
(1) stop or wait another XML document
```

算法 3. DFA 的惰性构造与状态转换

```
ds0 为初态, dstack 为栈,栈帧为 DFA 状态

startDocument( )
(1) ds0= newDState({s0});
(2) Push(ds0, dstack);

startElement(a)
(1) ds = GetTop(dstack);
(2) if ds.containTransition(a)
(3)   Push( ds.move(a), dstack);
(4) else
(5)   ds_nset = ds.getNFAStates( );
(6)   new_nset = ε-closure(ds_nset.move(a));
(7)   nds = newDState(new_nset);
(8)   add an edge from ds to nds labeled a
(9)   Push( nds, dstack);

endElement(a)
(1) ds=Pop(dstack);

endDocument()
(1) stop or wait another XML document
```

3　XSIEQ 的查询机制

本节以 NFA 为例说明 XSIEQ 的查询机制.

3.1　缓冲机制

在 XML 流查询中,当自动机运行到匹配主 XPath 式的接受状态时,由于相应的谓词条件并不一定能完全确定,这时就需要缓存主 XPath 式所匹配的数据项. XSIEQ 在解析时实

时地维护各 XPath 式的谓词计算状态,并根据谓词计算状态即时地输出或清除缓存内容.

3.1.1 谓词计算状态的表示

XSIEQ 使用位组来标识某 XPath 式中各谓词的计算状态;对于嵌套谓词,进一步引入位组来标识该谓词及其内嵌谓词的计算状态.两类位组定义如下:

定义 1. 若 XPath 式 P 中含有 n 个顶层谓词,将这些谓词编号为 $0..n\text{-}1$,则 P 的谓词计算状态用 n 个 2 进制位位组 $b_{n-1}\cdots b_1b_0$ 表示,b_i($i\in[0,\ n\text{-}1]$)是编号为 i 的谓词的计算状态,b_i 取 1 表示确定为真,0 表示确定为假或未确定.

定义 2. 假设 Q 是 XPath 式 P 中的谓词,Q 内嵌有 m 个顶层谓词,这 m 个谓词相对编号为 $1..m$.内嵌谓词 Q' 在 P 中的绝对编号是其父谓词 Q 的绝对编号连上 Q' 在 Q 中的相对编号.若 Q 在 P 中的绝对编号为 $i_1\cdots i_d$,则 Q 的谓词计算状态用 $m+1$ 位位组 $b_{i_1\cdots i_dm}\cdots b_{i_1\cdots i_d1}b_{i_1\cdots i_d0}$ 表示,其中 $b_{i_1\cdots i_d0}$ 是 Q 的谓词计算状态,$b_{i_1\cdots i_dj}$($j\in[1,\ m]$)是 Q 中相对编号为 j 的内嵌谓词的计算状态.

例 2. 若 XPath 式 P_5 为/a[b]/c[d[e]/f[g[m]/n][h]/j]/k,则 P_5 包含 2 个谓词,即 Q_0:[b], Q_1:[d[e]/f[g[m]/n][h]/j];Q_1 内嵌有 3 个谓词,即 Q_{11}:[e], Q_{12}:[g[m]/n], Q_{13}:[h];而 Q_{12}又内嵌有一个谓词 Q_{121}:[m].这样 P_5 的谓词计算状态将由三个位组 b_1b_0、$b_{13}b_{12}\ b_{11}b_{10}$、$b_{121}b_{120}$组成,它们之间形成一种树型关系:位组 $b_{121}b_{120}$决定 b_{12}的状态,当 b_{121}、b_{120}均为 1 时,则置 b_{12}为 1;位组 $b_{13}b_{12}\ b_{11}b_{10}$决定 b_1 的状态.

3.1.2 缓冲区结构

例 3. 若 P_6 为//b[e][f]/d,待查询的 XML 流为:

<a><d>1</d><c><d>2</d><e/> <d>3</d></c><e/><f/>

表 1 列出当 SAX 解析到<e/>、<d>、和<f/>时 3 个候选数据项的缓存状态和谓词计算状态,位组自左至右表示谓词[e]和[f]的状态.

表 1 候选数据项及其状态

候选数据项		缓存状态/谓词计算状态			
节点位置	text()	到达<e/>	到达<d>	到达	到达<f/>
/a/b/d	1	C/00	C/00	C/00	11-输出
/a/b/c/b/d	2	C/10	C/10	11-清除	—
/a/b/c/b/d	3	U/-	G/10	11-清除	—

注:C-表示已缓存; G-正在收集; U-尚未缓存

从例 3 看出,在查询的某一时刻,可能需要缓存某 XPath 式的多个候选数据项,并且这些数据项的谓词状态可能不一致.当查询多个 XPath 式时,还可能出现同一数据项是多个 XPath 式的候选结果.为此,XSIEQ 在查询中缓存谓词条件尚未完全确定的候选数据项;并为每一 XPath 式建立缓冲区保存相关候选数据项的引用,具有相同谓词状态值的候选项被组在一个链表中.这样,一个 XPath 式的缓冲区将按谓词状态值的不同组织成多个链表.

3.2 索引结构

首先定义 NFA 中的三类特殊状态及其索引信息.

定义 3. 结果状态是指匹配主 XPath 式的 NFA 状态,即由主 XPath 式的最后一个位置步到达的 NFA 状态.结果状态保存它对应的各主 XPath 式的引用,即主 XPath 式对象列表 LR.

定义 4. 叶子状态是指谓词中的 XPath 式的最后一个位置步到达的 NFA 状态.在叶子状态上保存它关联的各谓词的引用,即谓词对象列表 LP.

定义 5. 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;分支出去的路径为谓词中 XPath 式所对应的路径.分支状态保存从该状态分支出去的谓词对象引用列表 LB.

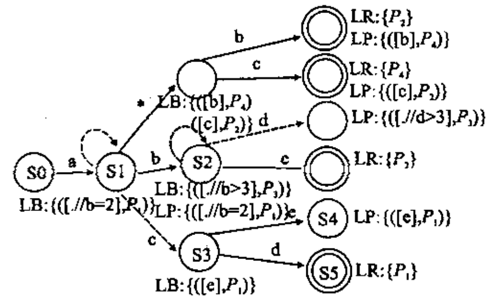


图 3 添加索引信息后的 NFA

按照上述定义,给例 1 的 NFA 加上索引信息,可得如图 3 的带索引的 NFA.假设待查 XPath 式的个数为 m,这些 XPath 式中谓词的个数为 n,则索引信息所占用的空间开销在最坏情况下为 $O(m+2n)$.

3.3 查询过程

观察 NFA 结构可知,到达结果状态将开始收集候选数据项;回溯到结果状态则结束收集,并根据其 LR 列表中各 XPath 式的当前谓词状态将收集的内容输出或缓存到相应的 XPath 缓冲区中;到达叶子状态可以计算 LP 列表中的各谓词;回溯到分支状态将确定 LB 列表中各谓词的最终计算状态.状态的到达和回溯分别由 SAX 解析产生的 startElement 事件和 endElement 事件来驱动.

因此,在 startElement 事件和 character 事件处理中,将根据栈顶各 NFA 状态的 LP 列表和当前面临的 XML 数据,立即计算 LP 中的各谓词.若 XPath 式 P 的某个顶层谓词 Q 的状态由 0 变为 1,则查找 P 缓冲区中 Q 的状态位为 0 的谓词状态值条目,将该条目下的数据项链表合并到 Q 状态位为 1 且其它状态位保持不变的谓词状态值条目下;若修改后谓词状态值的各位全为 1,表明该条目关联的数据项已经满足 P,可以立即输出.

在 endElement 事件处理中,检查栈顶状态 LB 列表中各谓词的计算状态.如果谓词 Q(属于 XPath 式 P)的状态位为 1,则将其置为 0 以等待下一次谓词计算;若为 0 则说明 Q 不满足,此时查找 P 缓冲区中 Q 状态位为 0 的谓词状态值条目,将该条目的数据项链表清除.

查询中对 P 缓冲区主要有三类操作：缓存数据项、更新谓词状态、清除数据项. 对含有 n 个顶层谓词的 XPath 式 P，若当前谓词状态取值有 m 种（$m\leqslant 2^n-1$），则以上操作一次执行的时间开销均为 $O(m)$. 若查询中，P 匹配的候选数据项数为 r，则操作 P 缓冲区的时间开销在最坏情况下为 $O(m\times n\times r)$.

3.4　多重匹配

定义 6. 在解析 XML 文档时，相互嵌套的不同深度的 XML 元素可能会匹配一个 XPath 式 P 中的同一个位置步，这种现象称为多重匹配. 在 NFA 查询 XML 流时表现为：某一时刻，同一 NFA 状态多次出现在栈的不同（也可能同一个）栈帧中，即该状态经过不同的 XML 文档路径得到了多次匹配. 这种现象发生的必要条件之一是 XPath 式中存在有子孙轴.

为了区分状态在不同层次上的匹配，在状态转换时，需要记录每个状态的匹配层次. NFA 状态的匹配层次计算规则为：

1）设置全局变量 m 保存当前的匹配层次；

2）文档解析开始时，NFA 初始状态的匹配层次为 1，$m=1$；

3）startElement(a)中，如果存在从层次为 n 的栈顶状态 S 转换到状态 Q，转换边为子孙轴且 S 在层次 n 上已经进行过到 Q 的转换，那么 Q 的匹配层次为 $m+1$，且修改 $m=m+1$；其它情况下，Q 的匹配层次为 n；

4）如果在 startElement(a)中更改了 m，则在 endElement(a)中将 m 恢复为更改前的值.

以上规则保证：

1）NFA 中相邻的状态，如果其中没有发生多重匹配，则它们的匹配层次相同；

2）NFA 不同分支中的状态，如果在同一次状态转换中都发生多重匹配，它们的匹配层次一定不相同.

例 3. 使用例 1 中 XPath 式 P_1：/a[.//b=2]//c[e]/d，查询如下 XML 片段：

片段 1：<a>2<c><c><e/><d/></c></c>；

片段 2：<a><c><e/><d/></c>2.

表 2　状态转换和匹配层次

SAX 事件	栈顶 NFA 状态集
startDoc	(S0,1)；m=1
startElem(a)	(S1,1)；m=1
startElem(b)	(S2,1),(S1,1)；m=1
endElem(b)	(S1,1)；m=1
startElem(c)	(S3,1),(S1,1)；m=1
startElem(c)	(S3,2),(S1,1)；m=2
startElem(e)	(S4,2),(S1,1)；m=2
startElem(d)	(S5,2),(S1,1)；m=2

表 2 列出解析片段 1 时的部分状态转换和状态的匹配层次. S1 由于包含子孙轴类型的出边，所以匹配后一直存在于栈顶.

匹配层次的概念贯穿整个查询处理过程，谓词计算和缓冲区都按照不同的匹配层次分别进行管理. 对匹配层次为 n 的叶子状态，其 LP 列表中各谓词的计算结果放入第 n 层的谓词状态位组中；对匹配层次为 n 的结果状态，候选数据项放入 XPath 式的第 n 层缓冲区. 层次为 n 的谓词计算结果只能作用于第 n 层缓冲区.

根据 XPath 式中多重匹配发生的位置，会导致如下两种问题.

1）多重匹配发生于主 XPath 式中. 如例 3 中 P_1 的位置步"//c". 查询片段 1 时，叶子状态 S2（见图 3）在层次 1，其 LP 表中的谓词[.//b=2]确定为真；在解析第 2 个 c 元素时发生多重匹配，进入层次为 2 的 S3 状态；到结果状态 S5 时收集 d 元素内容并放入 P_1 的第 2 层缓冲区中；当首次回溯到 S3（即第 2 个 c 元素结束）时，为使查询结果正确，必须将第 2 层缓冲区中的数据项合并到第 1 层缓冲区中，即将高层缓冲区中的内容合并到低层缓冲区中；

2）多重匹配发生于谓词的 XPath 式中. 如例 3 中 P_1 的谓词"[.//b=2]". 考虑片段 2，解析第 2 个 b 元素时在层次 2 到达叶子状态 S2，此时谓词确定为真，但分支状态 S1 在层次 1 匹配. 所以当回溯到 S1 时，需要将 LB 表中各谓词的高层计算状态传播到 S1 当前匹配层次的谓词状态中.

由上可知，在状态回溯时，对于每一个栈顶状态 S，如果 S 是多重匹配的发生点，则需要合并缓冲区；如果 S 是分支状态，则还需要将高层谓词状态传播到当前匹配层次.

多重匹配的发生将极大地降低 XSIEQ 的查询效率. 但是，除非 XPath 式中包含"//*"，否则查询一般的 XML 文档很少会发生多重匹配，这是查询效率得到保证的一个重要前提.

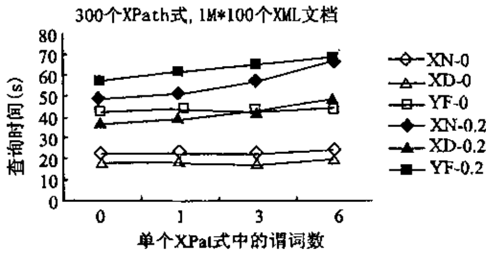
4　实　验

笔者用 Java 实现了两种 XSIEQ 查询自动机—NFA 和 Lazy DFA，分别记为 XN 和 XD，并与 YFilter（记为 YF）进行了对比测试. 实验平台为：Windows 2000，CPU AMD Athlon XP 2500+，内存 512DDR. 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. 图 4 是部分实验结果（见下页）.

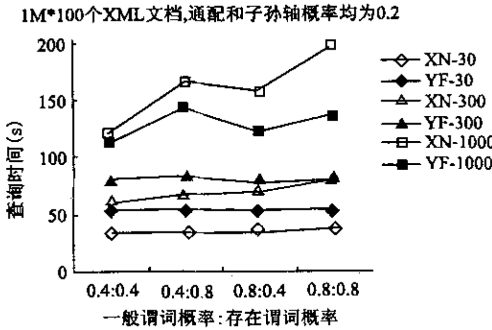
图 4(a) 图考察在查询 300 个 XPath 式时，单个 XPath 式所含的谓词数（0，1，3，6）、XPath 式中通配和子孙轴的出现概率（均为 0 或均为 0.2）对三种自动机的查询时间影响. 显然，XD 查询最快，但它是有空间代价的. 当 XPath 式中无通配、子孙轴时，XN 的查询时间约是 YF 的一半，并且随谓词数的增加而趋于平稳；当通配、子孙轴概率较高时（0.2），XN 与 YF 的差距缩小，且随着谓词数的增加，XN 的查询时间增长得更快. 这是因为谓词数增加将导致 XN 对缓冲区操作的增加；而 YF 先对 XML 预解析并在内存中保留完整的信息，在查询时不操作缓冲区，它是以空间来赢取时间.

图 4(b)图考察在通配和子孙轴概率均为 0.2 时，XN 和 YF 对不同数目（30，300 和 1000）的带嵌套谓词的 XPath 式

的查询性能，X 轴为一般谓词概率对存在谓词概率的四种取值. 图中曲线表明，在XPath式数比较少(300)时，XN查询性能比



(a)



(b)

图 4 查询性能比较

YF 高，并且一般谓词概率和存在谓词概率对查询性能的影响比较小. 随着 XPath 数的增多，处理存在谓词会比一般谓词更耗时间，因为存在谓词确定为真的几率比一般谓词要高得多，从而导致 XN 更频繁地处理缓冲区.

5 结 论

通过采用索引结构和缓冲机制以及有限自动机技术，本文给出了一种能立即计算的 XML 流查询处理解决方案，即 XSIEQ. 当前 XSIEQ 适用于查询的 XPath 数较少(≤300)、且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 通过前缀共享解决了 XPath 式状态共享问题；然而，查询中可能存在着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. 从实验结果也看到，频繁更新缓冲区也是制约 XSIEQ 性能的一个重要因素. 下一步将考虑谓词计算和缓冲区的共享等问题.

References:

[1] Yanlei Diao, Peter Fischer, Michael Franklin, Raymond To. YFilter: efficient and scalable filtering of XML documents[C]. ICDE 2002, February 2002:341-344

[2] Yanlei Diao, Mehmet Altinel, Michael J. Franklin et al. Path sharing and predicate evaluation for high-performance XML filtering[J]. ACM TODS, December 2003,28(4):467-516.

[3] Green T J, Miklau G et al. Processing XML streams with deterministic automata and stream indexes[J]. ACM TODS, Dec. 2004,29(4):752-788.

[4] YFilter 1.0 code release[EB/OL]. http://yfilter.cs.berkeley.edu/code_release.htm, April 2005.

[5] XMLTK 2.0 code release[EB/OL]. http://www.cs.washington.edu/homes/suciu/XMLTK/xmltk-v2.0.zip, April 2005.

[6] Albrecht Schmidt, Florian Waas. etc. XMark: a benchmark for XML data management[C]. Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.