Software Engineering and Applications 软件工程与应用, 2016, 5(2), 103-113
Published Online April 2016 in Hans. http://www.hanspub.org/journal/sea
http://dx.doi.org/10.12677/sea.2016.52012

Hans汉斯

Tree Pattern Matching Method on XML Stream Data

Yao Lu, Husheng Liao, Hang Su, Hongyu Gao

Computer College, Beijing University of Technology, Beijing
Email: luyao6@emails.bjut.edu.cn

Received: Mar. 15th, 2016; accepted: Mar. 29th, 2016; published: Apr. 1st, 2016

Abstract

For the needs of high-performance processing on XML stream data in the internet age, this paper fully considers that streaming data query needs to process the continuous arrival of data, etc., for the majority of the existing studies have been limited to XPath query defects, this paper studies and implements XML Query core operation—stream data processing algorithm on tree pattern matching through experiments show that the algorithm achieves the performance demand of XML stream data processing.

Keywords

XML, Tree Pattern Matching, Stream Data Processing

面向XML流数据的树模式匹配方法

路　瑶，廖湖声，苏　航，高红雨

北京工业大学计算机学院，北京
Email: luyao6@emails.bjut.edu.cn

收稿日期：2016年3月15日；录用日期：2016年3月29日；发布日期：2016年4月1日

摘　要

面向互联网时代XML流数据处理的高性能处理需求，充分考虑流数据查询需要处理持续到来的数据等特

文章引用: 路瑶，廖湖声，苏航，高红雨. 面向XML流数据的树模式匹配方法[J]. 软件工程与应用, 2016, 5(2): 103-113.
http://dx.doi.org/10.12677/sea.2016.52012

点，针对多数现有研究局限于XPath查询的缺陷，研究并实现了XML查询核心操作——树模式查询的流数据处理算法，通过实验说明该算法达到了XML流数据处理的性能要求。

关键词

XML，树模式匹配，流数据处理

1. 引言

随着大数据时代的到来，来自社交网络、金融数据管理、网络监控等各种实时系统的流数据[1] [2]处理需求越来越多，XML 流数据得到了广泛的应用，XML 数据量也呈爆炸式增长。对于如何高性能的从持续到来的 XML 流数据中提取出重要的信息也成为了研究热点之一。

XPath 是从 XML 文档当中提取信息的重要的查询语言，许多研究工作的注意力都集中在了如何高效的支持 XPath 语言对于 XML 流数据的处理。树模式匹配是 XML 查询的核心操作，用于从 XML 文档树当中提取出所有出现的满足树模式的片段的查询，涵盖了 XPath 语言的主要功能。目前也有许多 XML 流数据查询算法(TwigM [3]、LQ [4]系列算法等)，但均不支持多返回结点的情形。

树模式查询算法 TwigList [5]是一种高效的 XML 查询算法，虽然支持多返回节点的查询，但其针对非流式 XML 数据，并且空间消耗大。对于流式 XML 数据，该算法并不适用。本研究基于传统算法 TwigList 进行了改造，在功能上支持对于 PC、AD 轴和多返回节点的查询，减少了时空开销，得到高效的 XML 流数据匹配算法。

论文组织如下：第 2 节介绍背景知识；第 3 节介绍了针对 XML 流数据的树模式匹配算法的设计思路；第 4 节介绍本文的实验部分；随后介绍了相关工作和总结。

2. 相关工作

关于半结构化数据的 Twig 查询算法和已经得到深入研究。本文仅介绍与本文关系密切的若干算法。

2007 年，Lu Qin 等人提出了一种 TwigList [2]算法，该算法针对非流式 XML 数据，按照后序遍历的顺序处理 XML 文档。由于 TwigList 使用了简单的数据结构，能够高效的处理 XML 查询。但其对与查询结点对应的 XML 文档结点都建立 List 元素，使得时空消耗很大。本文是在 TwigList 的基础上，针对其空间开销大的问题，通过边缘结点放弃建立 List，直接到其边缘元素流中找满足查询树的节点，达到节省空间需求的目的。

关于 XML 流数据的处理，采用自动机是常见的方法，Yfilter [6]和 XPush Machine [7]通过同时支持多个查询提高了查询效率，但是仅支持简单的 XPath 查询。Hakuta S.等人提出将 XQuery 的子集 MinXQuery [8]翻译成森林自动机，研究了基于宏森林自动机的 XML 流数据处理技术，使得查询执行效率均比之前的算法有较大改善，但不支持多返回节点。相比之下，本文提出的 XML 流数据匹配算法具有执行效率高、空间开销小的特点，模式匹配算法中采用了新型的数据模型和实现算法。

3. 背景知识

3.1. XML 文档树

XML 是一种树型结构的自描述的标记语言，通过定义元素、属性和文本等进行数据信息的说明。一个简单的 XML 文档示例如图 1 所示。

为了描述方便，一个 XML 文档可以看作是一棵有根、有序、节点带标记的树，树中的节点表示文

```
<r>
  <a>
    <a>
      <b></b>
      <c>
        <d></d>
      </c>
    </a>
  </a>
  <a>
    <a></a>
    <d>
      <c></c>
    </d>
  </a>
</r>
```

Figure 1. XML document
图 1. XML 文档

档中的素、属性、文本节点等，节点间的边表示元素之间或者元素与文本之间的嵌套关系，兄弟元素间的顺序由树的先序遍历得到。图 2(b)所示的文档树与图 1 所示的 XML 文档相对应。

定义 1：树模式是一种树型的模式，表示为五元组 Q = (Node, Lab, tag, axis, var)。其中：

Node 是查询节点的集合。

Lab 是 XML 标签集。

tag：Node→Lab 函数指定了查询节点的标签。

axis：Node→(Node X {PC, AD})函数定义给定查询节点的双亲或祖先，PC 表示双亲子女关系，AD 表示祖先后代关系。

var：Node→Var 函数指定了查询节点上绑定的变量。这些变量用于引用树模式匹配的结果。树模式可以采用图形化表示。例如，树模式 A->$A[//B->$B][//C/D]可以表示为图 2(a)；其中，文字符号表示查询节点及其 XML 标签；圆圈表示返回节点，也就是具有绑定变量的节点；节点之间的单线表示 PC 关系、双线表示 AD 关系。于是，该模式规定 A、B 是返回节点，A 节点须有子孙节点 B、C，C 节点必须有 D 孩子。见表 1。

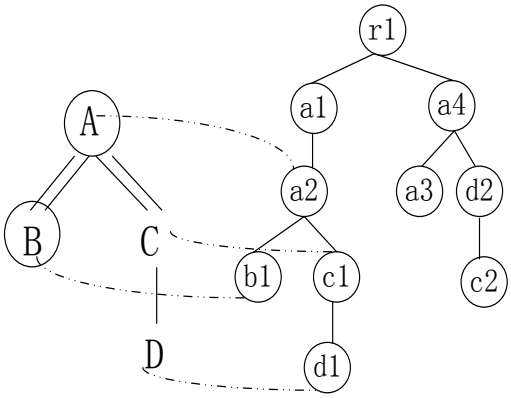
3.2. 树模式的匹配

树模式匹配就是从 XML 数据流中检索出满足树模式的文档片段。为了直观的显示树模式作用于 XML 数据流的匹配结果，这里把 XML 数据流建模为一颗有根的、有序的、有节点标记的文档树。设置一个虚拟的根节点作为树根，为每个 XML 数据流中的每个数据项建立一个子树，作为虚根的孩子。树节点代表 XML 元素，节点的内容代表 XML 元素标签，连接两个 XML 元素的边表示这两个 XML 元素是父子关系。

对于给定的 XML 文档树 D 和树模式 P，同时满足以下条件的 XML 节点就是匹配的结果：

对于 P 中的任何查询节点，在 D 都可以找标签相同的 XML 节点。

对于 P 中的节点 x，若 axis(x) = (y,PC)，则 D 中对应的 XML 节点满足双亲子女关系，若 axis(x) = (y,AD)，则 D 中对应的 XML 节点满足祖先后代关系。



(a) 树模式 (b) XML 文档树

Figure 2. Tree pattern and XML tree
图 2. 树模式和 XML 文档树

Table 1. The grammar of tree pattern
表 1. 树模式的文法

树模式的文法表示
语法域：
Texp 树模式
Node 树模式节点
文法
Texp ->Node
Texp ->//Node
Node -> Tag 子女标签名
Node -> Tag->Var 子女标签名->绑定变量名
Node -> Node Preds
Preds ->'[‘Texp']'
Preds ->'['Texp']'Preds 谓词组

同时，树模式的定义为外部程序访问匹配结果提供了返回节点，也就是处于函数 var 定义域的节点。外部程序可以通过这些变量，获得匹配结果中不同的 XML 片段。本文将匹配结果叫做实例树。

图 2(b)所示的 XML 文档树中，节点的数字角标是为了区分同名节点。该文档树与图 2(a)所示正规树模式进行匹配将获得 2 个结果。其中，a1 及其子树满足树模式约束，因为 a1 存在子孙 b1、c1，c1 存在 d1 孩子。a2 及其子树也满足正规树模式约束，因为 a2 存在后代 b1、c1，并且 c1 存在 d1 孩子。

3.3. 查询节点的类型定义

为了算法描述的方便，正规树模式中的节点可以分为四类：

1) 叶子节点：不存在子节点的节点。
2) 单枝节点：子节点的数量为 1 的节点。
3) 分枝节点：子节点的数量大于 1 的节点。

此外，查询节点可以分为返回节点和非返回节点，或者边缘节点和非边缘节点。存在绑定变量的节

点叫做返回节点。边缘节点包括不带绑定变量的叶结点，以及从这种叶结点到最近的返回节点、最近的分枝节点之间的所有节点。

4. 树模式匹配算法

本节以 XML 数据流处理为例，介绍 XML 数据流的树模式匹配方法。

4.1. 流数据处理流程

如图 3 所示，XML 数据流的树模式匹配流程由三个阶段组成。在预处理阶段，系统根据给定的树模式分析输入的 XML 数据流，过滤掉与树模式无关的 XML 元素；然后通过边缘节点的开始结束标签驱动边缘分枝匹配算法，其中，startElem()针对边缘分枝节点的开始标签进行处理，endElem()针对边缘分枝节点的结束标签进行处理。这两个函数共同确定存在边缘分枝的非边缘实例树节点是否满足边缘分枝并为其做标记，为模式匹配阶段的筛选做准备，并且将非边缘节点的开始标签和结束标签交给模式匹配阶段处理。在模式匹配阶段，调用 toStack()函数处理非边缘节点的开始标签，toList()函数处理非边缘节点的结束标签。他们共同完成树模式的模式匹配，采用实例树保存匹配结果。在第三阶段，系统从实例树中逐个枚举出检测出的每个查询结果。

4.2. 边缘分枝数据模型

每条边缘分枝由边缘节点和边缘节点的最近非边缘祖先节点组成。对于边缘分枝上的每个节点 K，设置一个 stackK。

如果 K 是边缘节点，则 stackK 中的元素是一个三元组 enode=(level,point,flagB)。其中，level 保存了该元素所在层次，point 是指向父亲查询节点 V 的栈 stackV 中满足轴约束关系的最近祖先节点的指针，flagB 标识该节点是否满足该节点所在的边缘分枝。B 是 K 所在边缘分枝上距离 K 最远的边缘节点，标识该边缘分枝。

如果 K 是边缘节点的最近非边缘祖先节点，则 stackK 中的元素是一个一元组 enode = (flags)。其中，flags 是 n(n 是边缘分枝的个数)元索引数组。对于子节点 K_i(0 < I < n + 1)，若 Ki 是边缘节点，则数组元素 $flagK_i$ 标识该元素是否满足以 K_i 为最近边缘后代的边缘分枝。

4.3. 实例树数据模型

本节介绍保存树模式匹配结果的实例树模型。实例树模型采用与树模式相同的树结构。对于每个非边缘节点 K，设置一个实例树节点列表 L_K，这里假设 K 有 n 个非边缘子节点。

L_K 的表元素是四元组 node=(elem,starts,ends,brother)，表示一个实例树节点 qq。其中，elem 是标记为 K 的 XML 元素，starts、ends 都是 n 元索引数组。对于节点 K_i (0 < i < n + 1)，axis(K_i) = (K,AD)，则列表 L_{ki} 中从 starts[i]到 ends[i]所指向的每个 L_{Ki} 表元素包含了 elem 的所有 K_i 子孙元素；若 axis(K_i) = (K,PC)，则从 starts[i]到 ends[i]分别指向的 Ki 表元素按照 brother 指针组成一个兄弟链表，包含了 elem 的所有 K_i 子元素。

图 4 是 XML 流数据处理引擎针对图 2(b)所示的 XML 文档树执行图 2(a)所示的树模式查询得到的实例树。A、C、D 节点均为边缘分枝上的节点，故分别为其建立栈 S_A、S_C、S_D。L_A、L_B 分别是 A、B 查询节点对应的 List。List 中存放的元素均满足以该元素为根的树模式子树。L_A 中每个元素指向 L_D 的 $starts_D$，$ends_D$ 指针之间的 B 元素均为其后代。L_B 中 a2 指向 L_B 的 $starts_B$,$ends_B$ 指针之间的元素均为 a2 的后代。

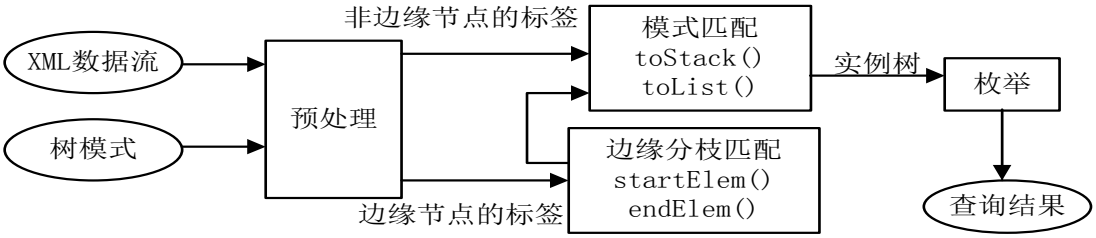


Figure 3. XML stream processing flow based on tree pattern
图 3. 基于树模式的 XML 流数据处理流程

Figure 4. Instance tree
图 4. 实例树

4.4. 边缘分枝过滤算法

在预处理阶段，边缘分枝过滤算法的输入是边缘节点的 XML 标签和其所在层次。使用到的数据结构是边缘分枝上的节点对应的栈 Stack。算法如下：

算法 1：startElem(label,level)

1. w 的树模式节点->W
2. W 的父亲查询节点->V
3. W 所在分枝上距离 W 最远的边缘节点->B
4. if S_V.empty()或者 S_V.top.flagB=true
5. return；
6. if axis(W)=(PC,V)并且(S_V.top().level+1)!=level
7. return；
8. 建立节点 w，
9. level->w.level，S_V.top()->w.point，false->w.flagB；
10. S_W.push(w)；
11. if W 是叶子节点 then
12. setEdgeFlag(w,W)

算法 2：setEdgeFlag(w,W)

输入：边缘分枝上的节点 w 和节点类型 W

1. if W 是边缘节点 then
2. W 的父查询节点->V；
3. w 指向 S_V 中的元素 w.point->cur；
4. if axis(W)=(V,AD)
5. while cur 存在并且 cur.flagB=false do
6. true->cur.flagB；
7. setEdgeFlag(cur,V)；
8. S_V 中 cur 的下一个节点->cur；

9. else
10. true ->cur.FlagB；
11. setEdgeFlag(cur,V)；

算法 3：endEBranch(label,level)

1. label 的树模式节点->W
2. if S_W 不空&&S_W.top().level=level
3. S_W.pop()；

算法 1 用于响应边缘节点开始标签的处理，当 w 的开始标签到来的时候，如果 w 与其对应的查询节点 W 的父亲节点 V 的栈顶元素 stackV.top()满足轴约束关系，并且 stackV.top()还不满足边缘分枝时，则执行入栈操作。同时，如果 w 是叶子节点，会递归调用 setEdgeFlag()函数为此分枝上的满足边缘分枝的所有元素设置标识这些元素满足边缘分枝的布尔值。算法 3 用来响应边缘节点的结束标签。

4.5. 模式匹配算法

如上所述，树模式的模式匹配的输入是非边缘节点的 XML 标签，输出是保存匹配结果的实例树模型。模式匹配过程中使用了几个数据结构：正规树模式 Q、栈 STK、存在边缘子节点的非边缘节点对应的 Stack 以及非边缘节点对应的 List。两个核心算法如下：

算法 4：toStack(label)

1. 为 label 构造实例树节点 v；
2. v 对应的查询节点 →V；
3. 构造标识边缘分枝的栈元素->flags
4. for each child W of V do
5. if W 是边缘节点 then
6. flags.add(flagsW = false)
7. else
8. $v.starts_W = length(L_W)+1$;
9. if flags.size!=0 then
10. S_V.push(flags)
11. STK.push(v);

注释：$v.starts_W$ 表示实例树节点 v 中 starts 数组中子节点 W 所对应的元素。

算法 5：结束标签的处理算法 toList(label)

1. v = STK.pop();
2. v 对应的查询节点 →V
3. if check(v, V)=false then
4. return;
5. 将 v 添加到列表 Lv；

算法 6：check(v,V)

输入：XML 元素 v、其对应的查询节点 V

输出：布尔值

1. if V 存在边缘分枝
2. S_V.pop->flags

3. if flags 中存在值不等于 true 的元素
4. return false
5. for each child W of V do
6. if W 是非边缘节点 then
7. $v.ends_W$= length(L_W);
8. if $v.ends_W$<$v.starts_W$ then
9. return false;
10. if axis(W)=(V,PC) then
11. 在 $v.starts_W$ 和 $v.ends_W$ 之间构造兄弟链表;
12. if 兄弟链表为空 then
13. return false;
14.return true;

算法 5 所示的 toList 函数用于响应非边缘节点结束标签的处理；从栈中弹出当前标签对应的 XML 元素 v 后，得到对应的查询节点 V，调用 check 检查以 v 为根的子树中是否存在以 V 为根的查询子树的查询结果。当节点 v 通过上述检查时，则作为中间结果保存到实例树节点列表 L_V 中。

在 check 函数的处理过程中，对边缘节点和非边缘节点分别进行的检查。前者检查是否满足边缘分枝，后者则检查 V 子女的实例树节点列表是否存在相应的元素。

图 5 展示了图 2(a)的树模式对图 2(b)的 XML 文档树匹配过程的片段。图中的 T、F 分别表示 true 和 false。初始时，STK 初始化为空，S_A、S_C、S_D、L_A、L_B 也初始化为空。当 a1 的开始标签到来时，创建 a1 表元素，设置 $a1.starts_B$ 为 length(L_B)+1 并将其入栈 STK，同时，创建 a1 栈元素，设置 a1.flagB = F 并将其入栈 S_A。当 a2 的开始标签到来时，执行的操作与 a1 相同。当 b1 的开始标签到来时，创建 b1 表元素，并入栈 STK。如图 5(a)所示。当 b1 的结束标签到来的时候，b1 出栈 STK,并加入到 L_B。当 c1 的开始标签到来时，由于 c1 和 S_A 的栈顶元素 a1 满足轴约束关系，故创建 c1 栈元素，设置 c1.point 指向 a2,c1.level 为 c1 的层次，flagC 为 F。当 d1 的开始标签到来时，同样设置 d1.point 指向 c1，d1.flagC 为 F。由于 d1 是叶子节点，依次设置 b1.flagC、c1.flagC、a2.flagC、a1.flagC 为 true。如图 5(b)所示。当 d1、c1 的结束标签到来时，d1、c1 分别从 S_D、S_C 出栈。当 a2 的结束标签到来时，设置 $a2.ends_B$ 为 length(L_B)，由于 $a2.ends_B$ >= $a2.starts_B$ 并且栈元素 a2.flagC = T，所以 a2 存在 B 类的后代，并且满足以 C 为最近后代的边缘分枝。故将 a2 加入 L_A 的后代，由于 streamB 中的 b1 是 a1 的孩子，故 a1 满足树模式约束被加入到 L_A，如图 5(c)所示。

4.6. 算法分析

在复杂事件检测方法的开销集涉及边缘分支的处理和非边缘节点的处理。

在时间复杂度方面，非边缘节点的 check 匹配算法，仅涉及 XML 子树的一遍扫描，且没有针对 XML 元素的循环处理，可以在线性时间内完成。边缘节点的匹配算法中，时间开销主要集中在 setEdgeFlag，为 O(d*n)。其中，d 是分支个数，n 是边缘分支的元素个数。因此计算复杂度与元素个数成正比。在最坏的情况下，对于 XML 文档 D，树模式的时间复杂度可以达到 O(m|D|)，其中，m 是最大分枝度数。因此，树模式匹配可以在线性时间内完成。

在空间复杂度方面，边缘分枝栈中的元素所占内存空间为 O(N1)，其中 N1 是 XML 数据流中边缘分枝的深度，并且对于非边缘节点元素来说，占用内存空间为 O(b*N2)，其中 b 为分枝节点的个数，N2 是 XML 数据流中非边缘节点元素的个数。因此，空间复杂度为 O(N1 + b*N2)。鉴于分支节点的数量有限，流数据匹配过程中将随时释放已经枚举的结果，算法的空间开销主要取决于正规节点匹配结果的需求，

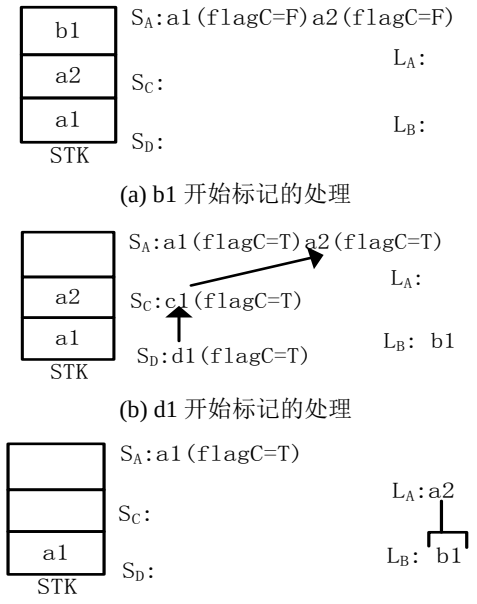


Figure 5. State change in matching process
图 5. 匹配过程的状态变化

与匹配元素序列的长度和匹配结果的数量成正比。

5. 流数据匹配结果的获取

每当处理正规树模式的根节点的结束标签时，则调用算法 Enum 获得并输出匹配结果，通过 Enum(x,v)可以获得实例树 x 中指定变量 v 的绑定值，具体算法如下：

算法 7 枚举算法 Enum(x,v)

输入：实例树节点 *x*, 绑定变量 *v*

输出：匹配结果

1. if *v* 与 *x* 的子女 *y* 绑定 then
2. return *x* 中 *y* 对应的实例树节点列表
3. 建空集合 *s*;
4. for each *z* in *x* 的子女
5. 将 Enum(*z*, *v*)添加到 *s*;

return *s*;

6. 实验

本节对基于正规树模式匹配的复杂事件检验方法进行不同的对比实验，并对实验结果进行简要分析，说明本方法的正确性和高效性。案例见表 2。

6.1. 实验环境

实验环境为一台配有 Intel Core i7-2600 3.40GHz CPU，12G 内存的 Windows 7 PC，Java 运行环境（JRE）版本 1.6。实验数据采用的数据集是 3 个 XML 基准数据集，分别是两个真实数据集 DBLP 和 TreeBank，以及一个根据参数人工生成的数据集 Xmark (见表 3 所示)的流数据集。

Table 2. Tree pattern test case
表 2. 树模式测试案例

查询标号	数据集	树模式
Q1.1	XMark	//item[/location]/description//keyword
Q1.2	XMark	//people//person[//address/zipcode]/profile/education
Q1.3	XMark	//item[/location][//mailbox/mail//emph]/description//keyword
Q1.4	XMark	//people//person[//address/zipcode][/id]/profile[//age]/education
Q1.5	XMark	//open_auction[//annotation[//person]//parlist]//bidder//increase
Q2.1	DBLP	//dblp/inproceedings[/title]/author
Q2.2	DBLP	//dblp/article[/author][//title]//year
Q2.3	DBLP	//dblp/inproceedings[//cite/label][/title]/author
Q2.4	DBLP	//dblp/article[/author][//title][//url][//ee][//year]
Q2.5	DBLP	//article[//mdate][//volume][//cite//label]//journal
Q3.1	TreeBank	//S/VP//PP[//NP/VBN]/IN
Q3.2	TreeBank	//S/VP/PP[/IN][/NP/VBN]
Q3.3	TreeBank	//S/VP//PP[//NN][//NP[//CD]/VBN]/IN
Q3.4	TreeBank	//S[//VP][//NP]/VP/PP[/IN]/NP/VBN
Q3.5	TreeBank	//EMPTY[//VP/PP//NNP][//S[//PP//JJ]//VBN]//PP/NP//_NONE_

Table 3. Three XML baseline data set
表 3. 三个 XML 基准数据集

数据集	大小(MB)	节点数(百万)	最大平均深度
DBLP	127	3.3	6/2.9
XMark	115	1.7	12/5.5
TreeBank	82	2.4	36/7.9

待测试的树模式查询用例是包含 PC、AD 轴和存在谓词的 XPath 查询。采用流式的 TwigList 算法和本论文提出的流数据匹配方法作对比。其中，流式的 TwigList 算法是本课题研究为了进行对比实验，对 TwigList 算法进行了改造，使 TigList 算法能够针对流数据进行查询处理。

6.2. 测试结果及分析

本测试以 XML 数据流的查询执行时间和内存消耗作为衡量算法高效性的依据。其中，有效执行时间 $T = t_{total} - t_{in} - t_{out}$，其中，$t_{total}$ 为 XML 数据流处理的总时间，t_{in} 为从磁盘上读入 XML 文档并对其进行解析的时间，t_{out} 为从内存向磁盘中输出查询结果的时间。

图 6、图 7 分别反映了针对 Q1.1~Q3.5 分别执行流数据匹配方法和流式 TwigList 算法的执行时间、内存消耗对比。从图中可以看出，流数据匹配方法的执行速度优于流式 TwigList 算法，并且内存消耗也小于 TwigList 算法。

7. 结论

针对大数据时代半结构化流数据处理中数据量巨大、价值密度低等特点，本文提出一种新型的 XML

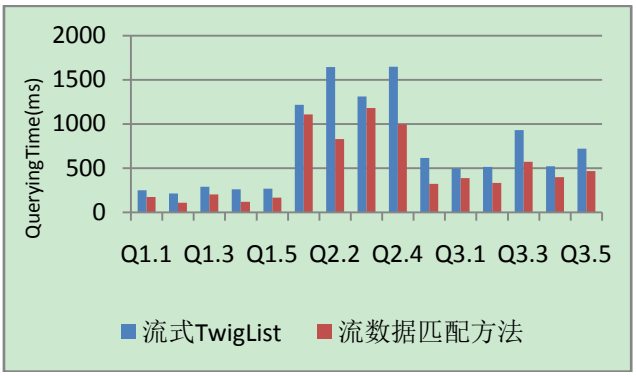


Figure 6. Query time comparison chart
图 6. 查询时间对比图

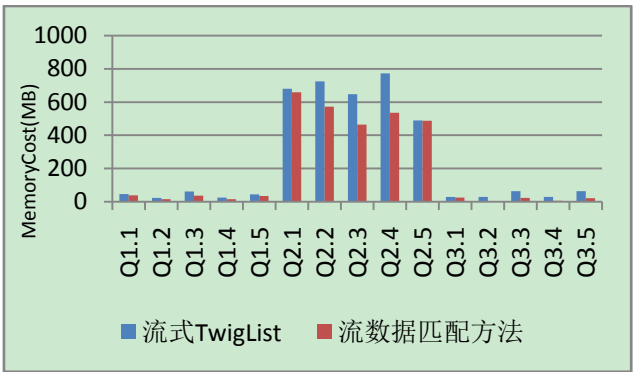


Figure 7. Memory consumption comparison chart
图 7. 内存消耗对比图

流数据匹配算法，捕获 XML 数据流中指定树模式，满足了流数据处理的性能需求。实验表明，本方法在满足树模式查询空间开销小的同时，保证了查询的执行效率。

参考文献 (References)

[1] Yang, W.D. and Shi, B.L. (2009) A Survey of XML Stream Management. *Journal of Computer Research and Development*, **10**, 18.

[2] Barzan, M., Kai, Z. and Carlo, Z. (2012) High-Performance Complex Event Processing over XML Streams. *SIGMOD*'12, Scottsdale, 20-24 May 2012, 253-264.

[3] Peng, F. and Chawathe, S.S. (2003) XPath Queries on Streaming Data. *Proceedings of the* 2003 *ACM SIGMOD International Conference on Management of Data*, 431-442.

[4] Olteanu, D. (2007) SPEX: Streamed and Progressive Evaluation of XPath. *IEEE Transactions on Knowledge and Data Engineering*, **19**, 934-949. http://dx.doi.org/10.1109/TKDE.2007.1063

[5] Qin, L., Yu, J.X. and Ding, B. (2007) TwigList: Make Twig Pattern Matching Fast. *Proceedings of DASFAA*'07, Springer, Thailand, 850-862.

[6] Diao, Y.L., Fischer, P., Franklin, M. and To, R. (2002) YFilter: Efficient and Scalable Filtering of XML Documents. *Proceedings of the* 18*th International Conference on Data Engineering*, IEEE Computer Society, Washington DC, 341-342.

[7] Ashish, K. and Gupta, D.S. (2003) Stream Processing of XPath Queries with Predicates. *Proceedings of the* 2003 *ACM SIGMOD International Conference on Management of Data*, 419-430.

[8] Hakuta, S., Maneth, S., Nakano, K. and Iwasaki, H. (2014) XQuery Streaming by Forest Transducers. *ICDE*, 417-428. http://dx.doi.org/10.1109/icde.2014.6816714