



# XML数据流上的 关键字查询

王小锋



# 动机

- ❖ 以前的研究工作已经表明，在XML文档上进行关键字查询对用户来说是非常友好的，因为用户不用事先知道文档所包含的数据以及结构信息，也不需要学习复杂的查询语言如XPath或者XQuery等，但同样可以找到用户感兴趣的文档片断。
- ❖ 随着XML成为Internet环境中数据表示和交换的标准，也出现了大量的与XML数据流相关的应用，如股票交易，主动服务中的订阅和发布。与传统的应用环境相比，在流环境下，查询处理所使用的内存远远小于数据流本身；查询处理过程中数据仅仅能够被扫描一遍。
- ❖ 在XML文档上进行关键字查询以及用xpath等在xml文档上进行查询已经有很多文章讨论过，但是据我们所知，还没有人提出在XML流数据上进行关键字查询。

# 挑战

- ❖ 在传统的非流处理的环境下，XML结点可以在查询处理的过程中随机访问到，而在流环境下，每个结点只能被顺序地访问，并且只能访问一次，这样不可避免地必须缓存一些中间结点，因此如何较早得到查询结果，并减少中间缓存结点成了研究中的一个难点。
- ❖ 如果在流处理的环境下，用户指定查询语言，比如用XPath提交查询，那么我们可以用自动机等方法保存和查询相关部分的结点，但是如果只提交关键字，我们并不知道哪些结点可能是查询相关结点，。

## 相关工作

- ❖ Yi Chen. An Efficient XPath Query Processor for XML Streams.ICDE2005
- ❖ Yu Xu.Efficient Keyword Search for smallest LCAs in XML Databases
- ❖ Lin Guo.XRANK:Ranked Keyword Search over XML Documents

# Contributions

- ❖ 基于SLCA问题，我们提出了在XML数据流上处理关键字查询的算法Lookup。
  - ✎ 利用SAX接口，我们动态保存一个从根到当前结点的路径，并根据查询关键字标记结点是否可能构成解，一旦发现一个解，就立即输出，并释放缓存中不必要的结点。
- ❖ 我们实现了相关的算法，并用实验证明了我们的算法是高效的

# SLCA 问题描述

- ❖ 例如，[图1](#)所示的XML数据“dblp.xml”，用户可能对James和Richard之间的关系比较感兴趣，他只需要提交查询关键字“James”和“Richard”就可以了，搜索算法会找到article和Proceeding结点并返回给用户，其中的语义信息也是很明显的：这两个人合写了一篇文章，并共同主持了一个会议
- ❖ Slca（smallest lowest common ancestor）问题是找出这样的一个子树，它包含了所有的查询关键字，但是它的任意一个子树都不包含所有的关键字。

## slca问题描述

- ❖ **Slca问题定义**: 用 $T(V_T, E_T)$ 来表示XML流数据, 给定一个关键字集合 $\{k_1, k_2, \dots, k_n\}$ 和 $T$ 上的一组结点集合 $\{S_1, S_2, \dots, S_n\}$ , 存在一个映射 $f: k_i \rightarrow S_i$ , 满足下面条件: 对每一个关键字 $k_i$ , 在 $T$ 上都有一个集合 $S_i$ 与之对应,  $S_i$ 中的每一个结点都直接包含 $k_i$ 。用 $lca(k_1, k_2, \dots, k_n)$ 表示关键字集合的最近祖先结点集合, 给定一个结点 $v \in slca(S_1, S_2, \dots, S_n)$ , 对于任意的 $u \in lca(S_1, S_2, \dots, S_n)$ ,  $v$ 不是 $u$ 的祖先结点。
- ❖ Yu Xu等提出了三种算法来处理SLCA问题, 都是基于Dewey endcoding编码的, 而且算法处理有一个重要的前提, 就是事先已经得到了 $\{S_1, S_2, \dots, S_n\}$ 集合, 但是在流处理上这个集合是不能实现获得的, 因此上述算法对流文档是无效的。
- ❖ 关于返回值: 考虑用一个路径集合来表达结果(路径的叶结点是查询关键字), 这样用户可以得到结构信息

# 一个实例

```
<dblp>
<articles>
<article key="journals/toplas/ArcherCS84">
<author>James</author>
<author>Richard</author>
<title>User Recovery and Reversal in Interactive Systems.</title>
<year>1984</year>
...
</articles>
<proceedings>
<proceeding>
<author>James</author>
<author>Richard</author>
<title>An Industrial Success in Formal Development.</title>
<pages>26</pages>
<year>1998</year>
</proceeding>
...
</prceedings>
```

# 树模型

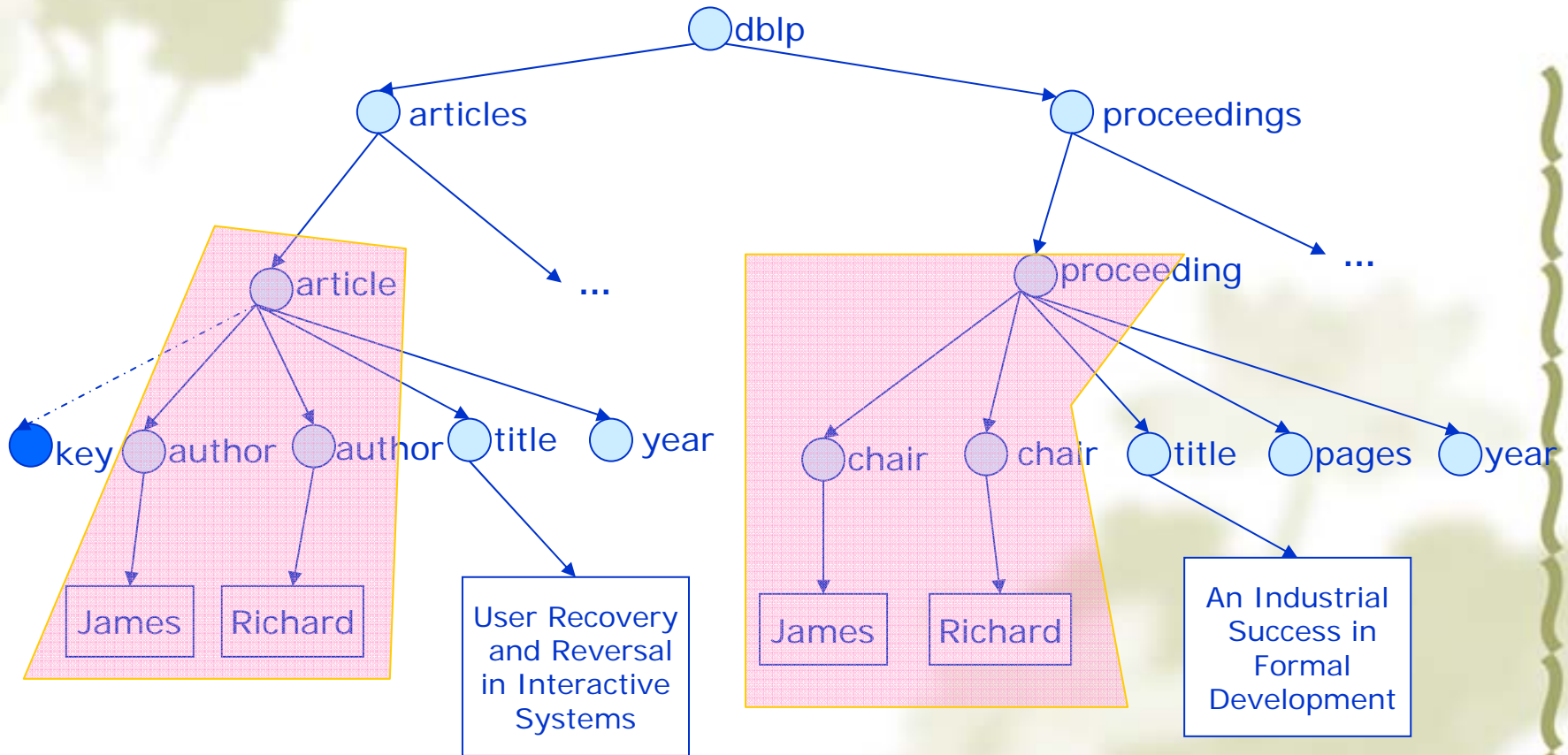


图1 dblp.xml

# XML Stream操作

## ❖ Sax接口

↪ Startelement()

↪ Endelement()

↪ Characters()

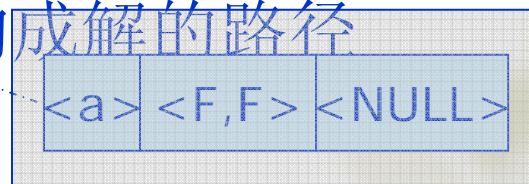
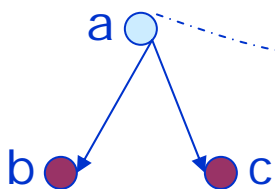
# 我们的方法

## ❖ 重要思想

- ↪ 使用一个栈来保存从根到当前结点的一个路径
  - ↪ 在处理过程中，随着`startelement()`事件，我们把新的结点推入栈中。
  - ↪ 当遇到`character()`事件时，我们判断文本结点中是否包含查询关键字，如果有，则向父结点对应位置标记为`true`，并添加从父结点到正处理结点的一条路径。
  - ↪ 随着`endelement()`事件，我们判断当前结点是否构成解，如果构成解，则立即输出，并清除缓存的中间结点。否则要么向父结点标记，要么直接丢弃，保证处理过程中占用较少的内存。
- ❖ 为什么第一次遇到满足条件的结点一定在结果集合里呢？
- ↪ 这是由流遍历的顺序决定的，先序遍历保证了结果的正确性。

# 数据结构

- ❖ 查询处理过程中，给每个入栈的非叶结点都关联一个三元组,用  $(t,v,cp)$  来表示，其中  $t$  表示 **tag**， $v$  表示 **vector**， $c$  表示 **candidate path**。
- ❖  $t$ —结点的标签名
- ❖  $v$ —**bool**数组，数组大小和提交的关键字个数相等  
 $v[i]$  — 对应位置的关键字是否存在
- ❖  $c$ —候选路径，可能构成解的路径



# 数据结构

- ❖ 为什么要给每个非叶结点关联一个 **bool vector**?
  - ↪ 当在流上遇到一个匹配关键字的时候，我们并不能判断此结点的父结点是否是解，因此我们只需将父结点的 **vector** 对应此关键字的位置上标记为 **true** 即可。因此，当在流上遇到一个结点结束的时候，就可以通过 **vector** 来判断此结点是否是一个解（当且仅当 **vector** 各个项都为 **true**）
- ❖ 什么是 **cp**?
  - ↪ **cp** 记录了当前所匹配的候选子树，当一个结点结束时，如果 **vector** 全为 **true**，则输出 **cp** 给用户。

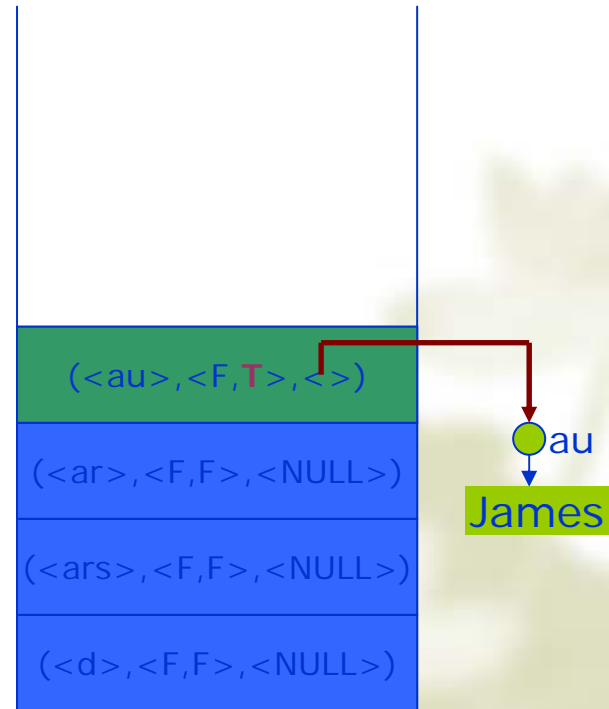
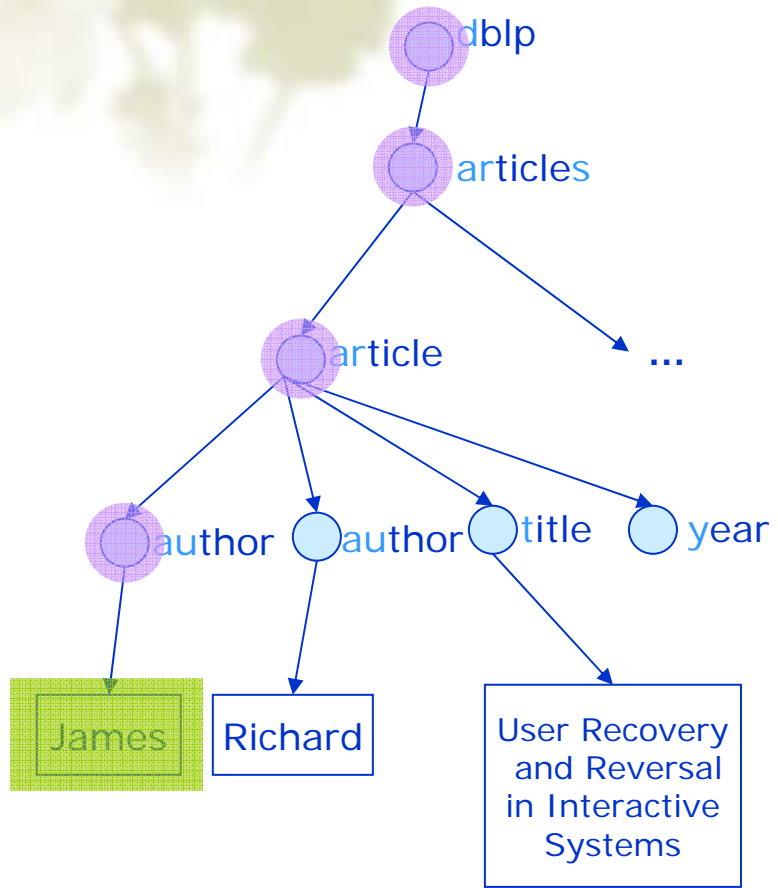
## 举例说明

- ❖ 如图1所示，我们给每个非叶结点关联一个二维bool数组  $v[2]$  和候选路径  $cp$ ，初始的时候数组中的每一个值均为 **false**， $cp$  为空。假定查询关键字为 **{James, Richard}**，当遇到 **article** 结束时，数组中的值均为 **true**，输出此结点对应的候选路径  $cp$ ，并从栈中弹出所有的祖先结点，因为它们不可能构成解。最后，输出两个文档片断：**article** 和 **proceeding**。注意，结果并不输出 **title** 等信息。

# 算法描述

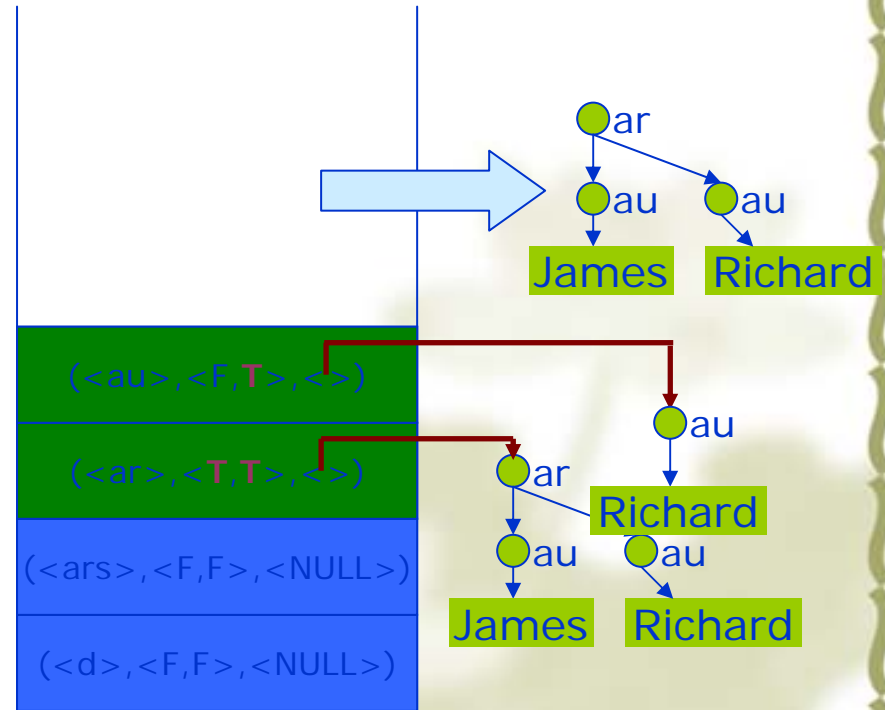
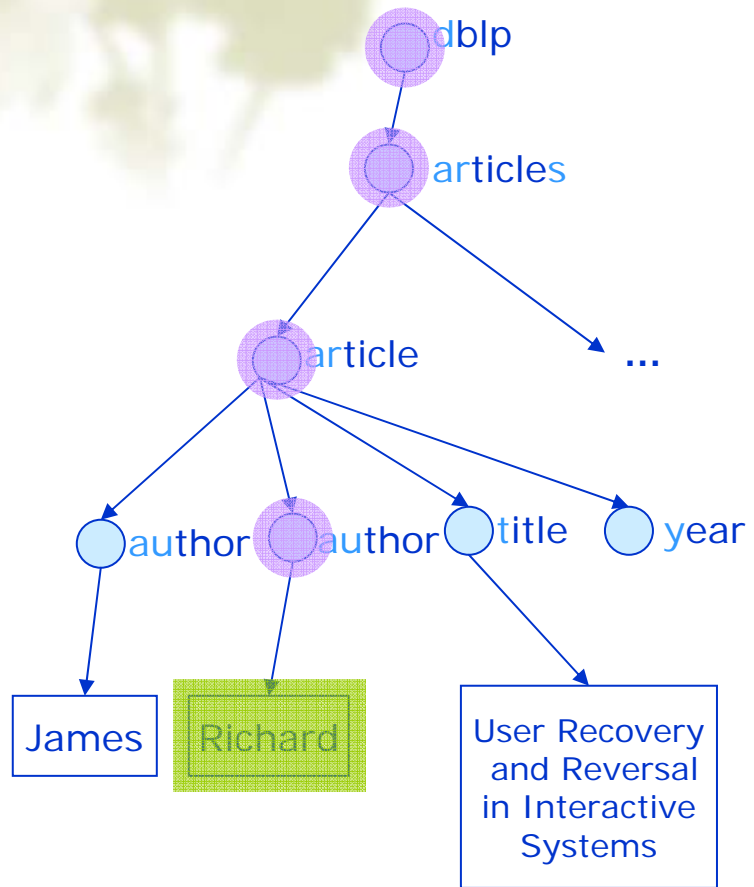
- ❖ 算法：找到keyword $\{k_1, k_2, \dots, k_n\}$ 的slca结点
- ❖ 输入：XML数据流
- ❖ 输出：slca结点集合
- ❖ StartDocument()
  - ↪ 初始化栈s
- ❖ StartElement()
  - ↪ 生成一个结点e,  $e.v[0 \sim n]=\text{false}$ ,  $e.cp=\text{NULL}$
  - ↪ s.push(e)
- ❖ EndElement()
  - ↪ If( $e.v[0 \sim n]=\text{true}$ )
  - ↪   outputSolution()
  - ↪   clearmidresult()
  - ↪ Else if( $e.v[i]=\text{false}$ ) //  $0 < i < n-1$
  - ↪   put e.cp to its parent
  - ↪ End if
- ❖ Characters()
  - ↪ If contains keyword
  - ↪   mark parent's corresponding position true
  - ↪ End if
- ❖ EndDocument()
  - ↪ Clear all temporary variables

# 处理过程实例



查询关键字 “james” “Richard”

# 处理过程实例



查询关键字 “james” “Richard”



# Experiments





The End