XML 数据流上的关键字查询

王小锋[1] 张新[1] 谢敏[1] 孟小峰[1] 周军锋[1,2]
[1](中国人民大学计算机应用技术系，北京 100872)
[2](燕山大学计算机系，秦皇岛 066004)
(zzuwxf@ruc.edu.cn)

摘要 XML 数据流上的 XPath&XQuery 查询处理是目前研究者关注的热点问题，但由于 XPath&XQuery 查询语言相对复杂，在不知道模式信息的前提下，用户很难通过已有的查询接口得到自己感兴趣的数据片断，因此如何在数据流模型上根据 XML 数据的特点为用户提供最友好的查询接口就成为一个亟待解决的问题。针对这个问题，本文创新地提出了在 XML 数据流上做关键字查询的问题，给出了最小相关连通子树(SRCT)的概念用于处理返回的结果，并设计了一种新的基于栈的 Lookup 算法，可以有效解决在 XML 数据流上进行关键字查询的问题，最后通过实验从不同角度对 Lookup 算法的各项性能指标进行了验证。

关键词 XML 数据流, 关键字查询, 最小相关连通子树

中图法分类号 TP391

Keyword Search on XML Streams

Wang Xiaofeng[1], Zhang Xin[1], Xie Min[1], Meng Xiaofeng[1], and Zhou Junfeng[1,2]
[1] (Department of Computer Technology, Renmin University of China, Beijing 100872)
[2] (College of Computer Science, Yanshan University, Qin Huangdao, 066004)

Abstract XPath&XQuery query processing on XML streams has become the hot topic and attracted many researchers' attention. However, due to the relatively complex query language of XPath&XQuery , it is hard for users to get interesting data fragments via existing query interface without the prior knowledge of the schema information. As a result, it is becoming an impending issue to address the problem to provide uses with the most friendly query interface in XML streams according to the feathers of XML data. In this paper, we propose the problem of keyword search on XML streams for the first time and present the concept of smallest related connected subtree(SRCT), In addition, we design a novel stacked-based Lookup algorithm, which can efficiently solve the problem of keyword search on XML streams. Sufficient experiments show that our method is efficient in practice in various aspects.

Keywords XML Stream, Keyword Search, SLCA

1 引言

随着 XML 成为互联网上数据表示和数据交换的事实标准，许多结构化或半结构的数据都用 XML 格式表示和传输，如何在主存中或者数据流环境下高效地处理 XML 上的查询成了许多研究者关注的问题。

近年来，出现了大量的与 XML 数据流相关的应用，如股票交易信息，实时的新闻订阅和发布，监控关键设备的运行，分析商业数据，监控环境的变化情况[1,2]等。与传统的应用环境相比，在流环境下对数据的处理有以下特征：

（1）XML 数据流是以固定的顺序到来的。也就是说，数据只能被顺序地访问，而且只能被访问一次。

（2）查询处理所使用的内存远远小于数据流本身。

用户在 XML 数据流上进行查询，可以有两种方式，一种和传统的 DBMS 类似，需要定义复杂的查询语言，在 XML 数据上代表性的查询语言主要有 XPath 和 XQuey，用户向系统提交查询关键性

收稿日期：2006-05-25
基金项目：国家自然科学基金(60273018) 国家 973 项目(2003CB317000)

字，系统返回相应的查询结果，使用这种方式的前提是用户必须学习相应的查询语言；第二种方式吸收了 IR(information retrieval)的思想，用户通过向系统提供关键字来查找自己感兴趣的数据，这种方式对用户来说更加友好，因为用户不用事先学习复杂的查询语言，也不用事先知道文档的结构信息，缺点是返回结果不一定是用户所需要的。

图 1 是一个用树结构表示的 xml 文档，文档主要包含两部分信息，一部分是和 article 相关的信息，一部分是和 proceedings 相关的信息，假设这个 xml 文档是以数据流的方式流过，在用户不知道文档具体结构的情况下，想获得两个 person 的相关性信息，例如 “James”和“Richard”，则用户可以提交这两个关键字，通过关键字查询接口就可以得到用户最关心的文档片断。在图 1 中，虚线区域包含的文档片断就是我们最终返回给用户的结果。这样的语义信息也是很明显的：“James”和“Richard”共同写了一篇文章，并且共同主持了一个会议。仔细观察不难发现，我们返回给用户的恰是从包含所有关键字的最近祖先到这些关键字的路径组成的最小子树的集合。我们称这样的子树为最小相关连通子树，有关的定义我们将在 3.2 节中详细讨论。类似上面例子的应用在实际中也是很常见的，又如一个用 XML 数据流形式发布的新闻 news.xml，如果这个新闻是关于拍卖信息的，用户想查找几个拍卖产品在何处相关联，关联的内容又是什么，我们返回给用户的最小相关连通子树对用户来说也是很有帮助的。

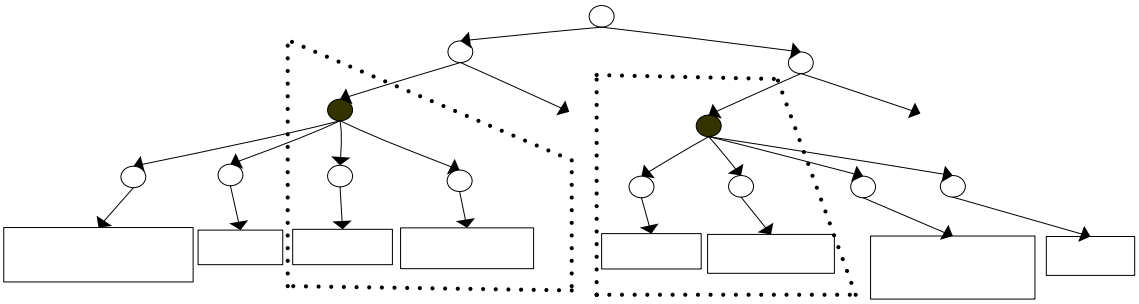


图 1 dblp.xml 文档片断

本文的贡献在于创新地提出了在 XML 数据流上做关键字查询的问题，给出了最小相关连通子树(SRCT)的概念用于处理返回的结果，并设计了一种新的基于栈的 Lookup 算法，可以有效解决在 XML 数据流上进行关键字查询的问题，最后通过实验从不同角度对 Lookup 算法的各项性能指标进行了验证。

本文后面部分的组织如下：第 2 部分是相关工作介绍。第 3 部分介绍数据流模型及相关概念。第 4 部分对基于栈的 Lookup 算法使用的数据结构及算法的实现进行了详尽的描述。第 5 部分展示实验结果。第 6 部分总结了本文的工作并对未来工作进行了展望。

2 相关工作

到目前为止，还没有文章在讨论如何在 XML 数据流上进行关键字查询。相关的工作主要有两个方面的，一个是在 XML 文档上进行关键字查询，代表性的文献主要有[3]和[4,5]， [3]主要讨论如何在 XML 树上快速找到包含关键字的 LCA(Lowest Common Ancestor)结点集合，并提出了如何对结果划分等级(Rank)的方法，文献[4,5]主要解决如何快速找到 SLCA(Smallest Lowest Common Ancestor)结点集合，文章认为这样的结果对用户是更有用的。在 XML 文档上进行关键字查询的关注点在于找到 LCA 或 SLCA 结点集合，而对返回的子树并没有深入的讨论；还有一方面的相关工作是讨论如何在 XML 数据流上进行有效的 XPath&XQuery 查询，文献[6,7,8]主要讨论在 XML 数据流上进行有效的 XPath 查询，方法主要有基于自动机的和基于栈的，用 XPath 表达的查询目标很明确，就是找出满足查询条件的一系列结点返回发给用户，文献[9,10]主要关注在 XML 数据流上进行 XQuery 查询，因为 XQuery 语法要比 XPath 复杂，因此处理起来也更困难一些。

由于是不同的应用场景，上面提到的两部分工作均不能处理 XML 数据流环境下的关键字查询问题，因此我们需要新的处理方法和处理策略。

3 数据模型及相关概念

3.1 XML 数据流模型

文献[9]给出了 XML 数据流的一种常用程序接口是

SAX，全称是 Simple APIs for XML，即 XML 简单应用程序接口。SAX 提供了一种对 XML 文档顺序访问的模式，在访问的过程中，会产生一系列的 SAX 事件，应用程序需要在这些事件接口中提供处理器。解析 XML 文档时遇到特定的事件，就去调用相应的处理器进行处理。在本文中，XML 数据流被表示成一系列的 SAX 事件，将事件分为：BEGIN(tag)，END(tag)和 TEXT()，其中 tag 是指处理过程中遇到的结点的名称，BEGIN(tag)表示遇到一个 tag 的开始，END(tag)表示遇到一个 tag 的结束，TEXT()表示遇到文本。在第 4 部分介绍算法的时候我们会详细阐述各个事件的具体实现。[1]

3.2 相关概念

我们用有向树 T(VT,ET,)来表示一个 XML 数据流数据，其中 VT 表示结点的集合，ET 表示边的集合。当一个结点为叶子结点时，用 val(VT)表示叶子结点的值。当一个结点是非叶结点时，用 tag(VT)表示结点的标签名。

定义 1：KM(Keyword Match)，关键字匹配，给定查询关键字集合 W={$k_1,k_2,\ldots,k_n$}和 T 上的一个结点 u，用符号 KM(u,k_i)表示结点 u 匹配关键字 k_i,。若 KM(u,k_i)，当且仅当：tag(u)或者 val(u)直接包含查询关键字 k_i（i>=1&i<=n）。

定义 1 说明了用户提交的查询如何和数据流数据进行匹配。

在给出 SLCA 定义之前，先对一些符号进行一下说明。假设关键字集合 W={$k_1,k_2,\ldots,k_n$}，对于每一个 k_i，在 T 上都有一个集合 S_i 与之对应，若 $S_i \neq \phi$，则 $\forall v \in S_i$，都有 $KM(v,k_i)$。函数 LCA($v_1,v_2,\ldots,v_n$)[4]计算结点 $v_1,v_2,\ldots,v_n$ 的最近公共祖先($v_1,v_2,\ldots,v_n$ 的最长前缀路径对应的结点)，如果 $\exists v_1 \in S_1, v_2 \in S_2, ..., v_n \in S_n$，使得 v = LCA($v_1,v_2,\ldots,v_n$)，则 v 被成为集合{$S_1,S_2,\ldots,S_n$}的一个 LCA 结点，用 S=LCA($S_1,S_2,\ldots,S_n$)表示这样的 v 的集合。

定义 2：SLCA[4](**Smallest Lowest Common Ancestor)，最近最小公共祖先，**满足如下条件被成为一个 SLCA 结点：

如果一个结点 v ∈ SLCA ($S_1,S_2,\ldots S_n$), 则对于任意的 u ∈ LCA($S_1,S_2,\ldots S_n$), v 不是 u 的祖先结点。

简单地说，SLCA 就是找出至少包含一组查询关键字实例的结点集合，并且这些结点的后代结点都不能包含所有的查询关键字实例，即返回结果的结点之间没有祖先后代关系。如图 1 所示，当查询关键字是“James”和“Richard”时，SLCA 集合是{article, proceeding}，即在图中用实心圆标注的结点。

定义 3：SRCT（Smallest Related Connected subtree）最小相关连通子树，假设关键字集合是 W={$k_1,k_2,\ldots,k_n$}，则最小相关连通子树是指联结关键字 $k_1,k_2,\ldots,k_n$ 实例的最小子树。简单地说，就是以 SLCA 为根，到对应到文档上的关键字集合的一个实例的各个路径所组成的子树，如果以一个 SLCA 为根的子树里面包含一个关键字多次，则仍然作为一个最小连通子树返回。

文献[11~13]描述了对关键字进行近似搜索的系统，文中指出，包含所有关键字实例越小的子树越能使关键字之间联系地更加紧密，因此这样的解被认为是一个更好的解。基于这样的假设，我们认为在返回结果给用户的时候，只返回最小相关连通子树的集合，而不返回与这些关键字无关的其它信息，就足以表达用户需要的语义信息了，如图 1 所示，当提交关键字是“James”和“Richard”时，我们只返回虚线框中的内容，而不返回 title、year 等信息，因为对于用户来讲，只关心“James”和“Richard”二者的关系，我们只把和用户查询关系最密切的信息返回给用户。

4 数据结构及算法描述

在本文中，我们提出的算法是基于栈的，任意时刻，栈内都保存一条到当前处理结点的路径（不一定是从根开始的路径），BEGIN 事件发生时，我们把结点推入栈中；END 事件发生时，判断当前结点是否构成解，如果已经构成解则直接输出。

4.1 数据结构

在我们的算法中主要用到的数据结构是栈。栈主要用户保存活跃结点(遇到了 BEGIN 事件，但还没有遇到 END 事件的结点)。栈元素是一个三元组（t,v,ct），其中 t 表示结点的标签名 tag，v 表示一个 bool 数组，用来标记关键字的匹配情况，数组的大小为待查找关键字的个数，并且数组每一项与一个关键字相对应，对于任意结点 u, 如果满足 KM(u,k_i)，则将 v 中与 k_i 相对应的元素置为 true。ct 表示候选子树 candidate subtree(所谓候选子树，

[1]在这里忽略了属性，实际上，处理和结点类似，在实验中我们不仅支持结点，而且也支持属性。

就是它可能构成一个查询结果或者是某个查询结果的子树)。每当遇到一个 END 事件时，判断以当前结点为根的子树是不是已经包含所有的关键字，即当前结点的数组 v 每项都为 true，构成解则输出，若包含部分关键字但不构成解则向父结点传递相关信息，若不包含任何关键字，则直接丢弃该结点。

4.2 基于栈的 Lookup 算法描述

在流上处理关键字查询的一个基本要求就是缓存较少的结点并及时输出结果。基于栈的 Lookup 算法充分考虑了这一点，算法可以描述为:

（1）使用一个栈来保存到当前结点的一条路径。

（2）在处理过程中，随着 BEGIN 事件，把新的结点入栈。

（3）当遇到 TEXT 事件时，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，并将其加到其父结点为根的候选子树中。要注意的是如果该文本中包含多个关键字，则只添加一次。

(4) 随着 END 事件，我们判断当前结点是否构成解，如果构成解，即相应的 v 数组所有项均为 true，则相应的 ct 所对应的子树就是一个作为解的最小相关连通子树，则立即输出，并清空栈。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。

定理 1：当遇到结点结束事件 END 时，如果与之对应的 bool 数组的各项已经被置为 true，则此结点就是一个 SLCA 结点。

证明：这个特性是由数据流的前序遍历顺序决定的的。假定此结点不是一个 SLCA 结点，那么一定存在一个结点 u，u 是此结点的后代且 u 对应的数组 v 各项已经被置为 true，但由于在处理到此结点时已经遇到了 v 结点的 END 事件，故不存在这样的 v，从而保证了输出结果的正确性和及时性。

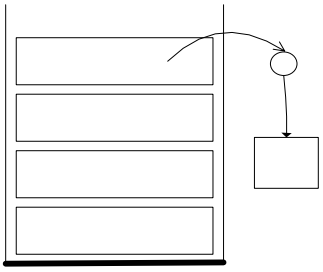
如果在处理的过程中，有一个结点对应的 bool 数组的各个位已经被置为 true，但是还未遇到此结点的 END 事件，此时不能判断此结点是否是 SLCA 结点，故不能输出候选子树给用户。这是因为可能存在一个结点 v 是此结点的后代，并且 v 也包含所有的查询关键字实例。

基于栈的 Lookup 算法尽量较少缓存的结点个数，当输出一个结点时，由 SLCA[4]的定义可知，此结点的祖先结点不可能构成解，因此当输出解时，还可以同时清空栈，从而保证占用较少量的内存。

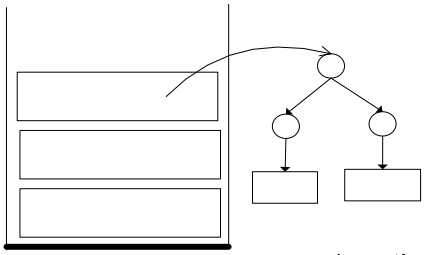
算法 1 是基于栈的 Lookup 算法所涉及的 SAX 处理函数，其中是 s 是一个全局的数据结构，用来表示栈，s.push(e)，s.pop()和 s.clear()是标准的栈操作，分别表示将元素 e 入栈，将栈顶元素出栈和清空栈操作。不难看出，算法的时间复杂度为 O(n)。

下面举例说明算法的处理过程，如图 1 所示，当用户提交关键字“James”和“Richard”时，根据 SAX 接口先序遍历文档，，依次把 dblp、articles、article、title 入栈，和结点关联的 bool 数组初始值全为 false，这时遇到 TEXT 事件，但是 title 里面不包含关键字，因此到 title 结点结束的时候，直接把 title 弹出栈，处理 year 和 title 类似。接下来，遇到了 author 结点的 BEGIN 事件，把三元组(<author>,<F,F>,<NULL>)推入栈中，然后处理 TEXT 事件，遇到了“James”，和其中的一个查询关键字相匹配，于是把 author 结点对应的 bool 数组的第一个位置置为 true，并修改候选子树 ct，此时栈的情况如图 2(a)所示。当 author 结点结束时，它对应的 bool 数组只有一个为 true，因此把它对应的 ct 加入到其父结点 article 对应的 ct 位置，并置父结点对应的 bool 数组的第一个位置为 true。接着用同样的方法处理第二个 author 结点，它包含第二个查找关键字，因此当这个 author 结束时，应把父结点对应的 bool 数组的第二个位置置为 true。此时，栈的情况如 2(b)所示。此时 article 对应的 bool 数组都被置为 true，但是此时并不能判断 article 是

否已经是解，因为 article 未遍历到的后代结点可能构成解。继续处理数据流，当遇到 article 结点的 END 事件时，发现 article 结点对应的 bool 数组的各个位置都已经为 true，输出一个解 ct 给用户，同时将栈清空，因为根据 SLCA 问题定义，此结点的祖先结点不可能构成解。后面的处理和前面类似，故不再赘述。



(a) 第一个 author 结点的 text 结束时



(b) 第二个 author 结点的结束时

图 2 处理过程中栈的变化情况

5 实验

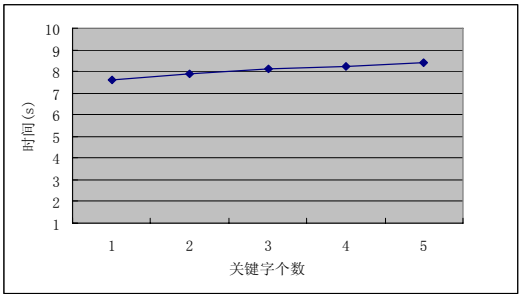
实验环境： 所有的实验都是在处理器为 Pentium 4 1.7G，内存为 512M，操作系统为 Windows XP 的机器上进行的。

数据集：我们将在三个数据集上进行实验。第一个是 XMark[14]数据集，产生的数据符合特定 DTD 模式定义。第二个数据集是从 TPC-H 关系数据库的 benchmark 转换过来的，是关于产品供求信息的 lineitem.xml。最后一个数据集 treebank.xml[15]是由英文句子组成的，这个数据集的特点是结构复杂，嵌套层次很深。这些数据集的特征如表 1 所示。

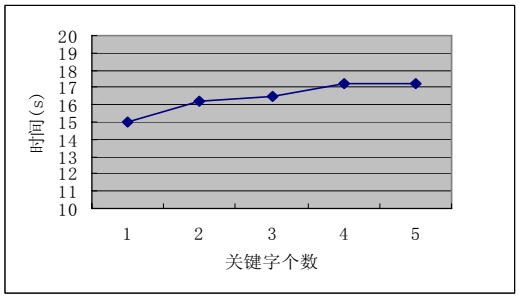
表 1 数据集描述

数据集名称	大小	结点个数	深度
XMark	20.3M	301161	12
Lineitem	30.7M	1022976	3
Treebank	82.0M	2437666	36

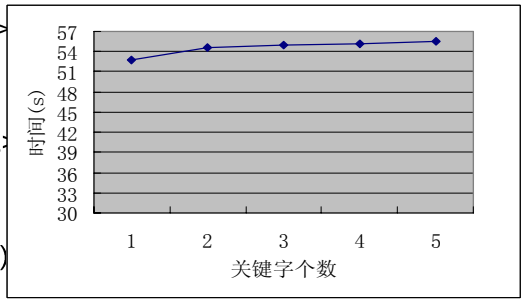
我们先来看一下查询处理时间随关键字个数的变化情况。图 3(a),(b),(c)是在上面三个数据集上的查询处理时间，其中横轴表示关键字的个数，纵轴表示查询处理时间。实验中，查询关键字是以增量方式添加的，即保留前一步的查询关键字，每一步添加一个新的关键字，观察查询处理时间的变化。从图中可以看出，关键字个数不同时，查询时间基本上变化不大，可以认为是一个常量。从(a)到(c),文档大小从 20M 增加到 82M，查询处理时间也随之增加，从 8s 左右增加到 54s 左右。



(a)XMark



(b)Lineitem



(c) Treebank

图 3 不同数据集查询处理时间比较

下面的实验主要是验证当数据增大时算法的可扩展性，实验中选取了 10M、20M、30M、40M、50M 不同大小的 XMark 文档。图 4 展示了随着文档的增大算法执行时间的变化情况，在实验过程中，关键字是保持不变的。从图 4 可以看出，查询执行时间随文档增大线性增加。

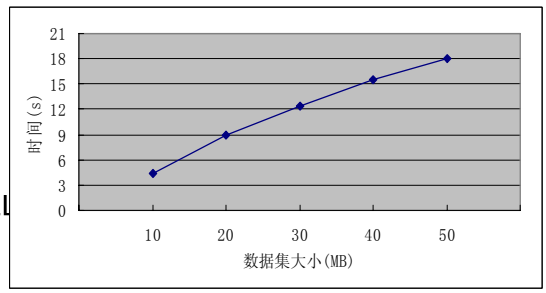


图 4 查询执行时间随数据集大小变化关系

衡量数据流环境下算法的好坏还有一个重要的因素就是内存使用率，图 5 是在 XMark 数据集上随文档增大内存利用率情况，实验数据同上(10M~50M)。从图中可以看出，内存占用率和文档大小关系不大，几乎维持在一个恒定的值。

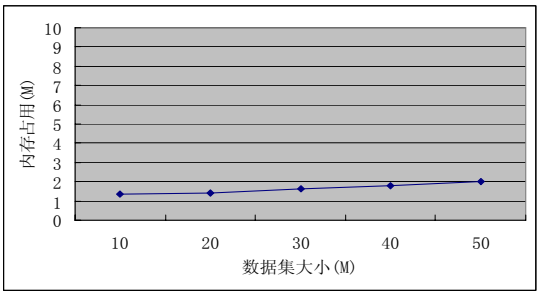


图 5 内存占用随数据集大小变化关系

6 结语和展望

在 XML 数据流上做关键字查询是很有意义的，可以给用户提供一个更加友好的界面，本文提出了在 XML 数据流上做关键字查询的想法，并给出了最小连通子树的概念，设计了一个有效的算法来计算最小连通子树，最后用详尽的实验验证了我们的算法是高效的。

之前的工作已经有在 XML 文档上处理关键字查询的方法，其中重要的一点就是对结果进行等级划分(Rank)，在本文中并未给予考虑，在以后的工作中我们会将二者结合起来进行研究。

参考文献

[1] L Golab, M Ozsu. Issues in data stream management. In: SIGMOD Record, 2003, 32(2) : 5-14

[2] B Babcock, S Babu, M Datar, .R Motwan, J Widom. Models and Issues in Data Stream Systems. In: Proceedings of PODS. Wisconsin: ACM Press, 2002. 1-16

[3] L Guo, F. Shao, C Botev, J.Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In: Procedings of SIGMOD. California : ACM Press, 2003.16-27

[4] Y Xu, Y Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In:ACM SIGMOD Conference. Maryland: ACM Press, 2005. 537-538

[5] Y Li, C Yu, H V Jagadish. Schema-Free XQuery. In: Proceedings of VLDB. Toronto: Morgan Kaufmann, 2004.72-83

[6] F.Peng, S.S.Chawathe. XPath queries on streaming data. In:ACM SIGMOD Conference. California: ACM Press, 2003. 431-442

[7] C Y Chan, P Felber, M N Garofalakis, R Rastogi. Efficient filtering of XML documents with XPath expressions. In: Proceedings of ICDE. California: IEEE Computer Press, 2002. 235-244

[8] Y Chen, S B. Davidson, Y F Zheng. An Efficient XPath Query Processor for XML Streams. In: Proceedings of ICDE. Georgia: IEEE Press, 2005. 79

[9] V. Josifovski, M. Fontoura and A. Barta. Querying XML streams. The VLDB Journal, 14(1): 2005: 197-210

[10] D Florescu,,C Hillary, D Kossmann, P Lucas, T Westmann, M.Carey, A Sundararajan. The BEA streaming XQuery processor. VLDB Journal, 13(1) 2005: 294-315

[11] G Bhalotia, C Nakhey, A.Hulgeri, S.Chakrabarti, S.Sudarshan. Keyword searching and browsing in databases Using BANKS. In: Proceedings of ICDE. California: IEEE Computer Press, 2002. 431-440

[12] R Goldman, N Shivakumar, S Venkatasubramanian, H Garcia-Molina. Proximity search in databases. In: Proceedings of VLDB. New York: Morgan Kaufmann 1998. 26-37

[13] V Hristidis, Y Papakonstantinouand A ABalmin. Keyword proximity search on XML graphs. In: Proceedings of ICDE. Bangalore:IEEE Computer Press, 2003. 367-378

[14] http://monetdb.cwi.nl/xml/index.html, 2001-04

[15] http://www.cs.washington.edu/research/xmldatasets/www/repository.html,2003-10

王小锋，女，1980 年生，硕士研究生，研究领域：xml 数据库。

张新，男，1983 年生，硕士研究生，研究领域：xml 数据库。

谢敏，男，1983 年生，硕士研究生，研究方向：xml 数据库。

孟小峰，男，1964 年生，教授（博导），研究领域：Web 数据管理，XML 数据库，移动数据管理。曾先后在香港中文大学、香港城市大学、新加坡国立大学访问研究。主持或参加过十多项国家科技攻关项目、国家自然科学基金以及国家 863 项目。

周军锋，男，1977 年生，博士研究生，研究方向：xml 数据库。