软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(6):1561−1577 [doi: 10.3724/SP.J.1001.2012.04114]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

XML 数据流上 Top-*K* 关键字查询处理*

黎玲利+, 王宏志, 高 宏, 李建中

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

Efficient Top-*K* Keyword Search on XML Streams

LI Ling-Li+, WANG Hong-Zhi, GAO Hong, LI Jian-Zhong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

+ Corresponding author: E-mail: lwsbrr@gmail.com

Li LL, Wang HZ, Gao H, Li JZ. Efficient Top-*K* keyword search on XML streams. 2012,23(6):1561−1577. http://www.jos.org.cn/1000-9825/4114.htm

Abstract: Keywords are suitable for query XML streams without schema information. In current forms of keywords search on XML streams and rank functions do not always represent users' intensions. This paper addresses this problem in another aspect. In this paper, the skyline Top-*K* keyword queries, a novel kind of keyword queries on XML streams, are presented. For such queries, skyline is used to choose results on XML streams without considering the complicated factors influencing the relevance to queries. With skyline query processing techniques, two techniques, are presented to process skyline Top-*K* keyword single queries and multi-queries on XML streams efficiently. Extensive experiments are performed to verify the effectiveness and efficiency of these techniques presented in this paper. According to the experimental results, the algorithms are not sensitive to the parameters such as the number of keywords, the number of results, the number of queries, and the runtime is approximately linear to the size of document.

Key words: XML; streams; keyword search; Top-*K*; skyline

摘 要: 利用关键字可以在模式未知的情况下对 XML 数据进行查询.在当前的 XML 数据流上的关键字查询处理中,打分函数往往不能都满足各种用户不同的需求.提出了一种基于 skyline 的 XML 数据流上的 Top-*K* 关键字查询.对于这种查询,不需要考虑影响结果与查询相关性的复杂因素,只需利用 skyline 挑选与查询最相关的结果.提出了两种 XML 数据流上的有效的基于 skyline 的 Top-*K* 关键查询处理算法,包括对单查询和多查询的处理算法.通过扩展实验对两种算法的有效性和可扩展性进行了验证.经过实验验证,所提出的查询处理算法的效率几乎不受关键字个数、查询结果数量、查询数量等参数的影响,运行时间和文档大小大致呈线性关系.

关键词: XML;数据流;关键字查询;Top-K;skyline

中图法分类号: TP311 **文献标识码**: A

由于其灵活性和可扩展性,XML 成为越来越多互联网上数据的表示标准.在一类应用中,XML 数据表现为

* 基金项目: 国家自然科学基金(61003046, 61111130189); 国家重点基础研究发展计划(973)(2012CB316200); 高等学校博士学科点专项科研基金(20102302120054)

收稿时间: 2010-04-28; 定稿时间: 2011-09-02

源源不断到来的数据片断,在这类应用中的 XML 数据称为 XML 数据流.XML 数据流的应用包括网络信息订阅与发布、电子邮件监测等.

当前,研究人员提出了很多对 XML 数据流进行查询的技术.这些技术主要集中在 XML 数据流上的结构查询,即在 XML 数据流上匹配某个特定结构[1−4].然而在许多应用中,如果用户对 XML 数据流中数据的模式不了解,则难以写出合适的查询,因而利用关键字对 XML 数据流进行查询成为一种需要.XML 数据流上的关键字查询即用户给定一个关键字集合 Q,则该查询返回包含所有关键字的 XML 片段.

由于在 XML 数据流中会有源源不断的 XML 片段到来,导致可能会有过多的 XML 片段满足关键字查询.因此,我们需要找到最相关的结果并且限制返回的结果数量.如何判定结果与用户查询的相关性,仍然是有待研究的问题.当前,已有许多工作对如何在 XML 数据的关键字查询里识别与查询相关的结果和比较结果的相关性进行了研究,但是某些情况下,结果的相关性是无法比较的,因此很难找到统一的对结果相关性的打分函数.我们用下面的例子来说明这一点.

例如,考虑一个在 XML 片段上的关键字查询 Q={*Bob*,*database*,*engine*},如图 1 所示.对相同的查询 Q,用户 *A*,*B* 和 *C* 有不同的查询需求:用户 *A* 想要查询 *Bob* 参与开发的有关 *database* 或 *engine* 的工程;用户 *B* 想要查询由 *Bob* 开发的关于 *engine database* 的工程;用户 *C* 想要查询由 *Bob* 开发的关于 *database engine* 的工程.如图 2 所示,查询返回了 3 个结果 $result_1$,$result_2$ 和 $result_3$.从 3 个查询结果中可以看出,对用户 *A* 来说,$result_2$ 和 $result_3$ 是与查询相关的,而 $result_1$ 不是;对用户 *B* 来说,$result_2$ 是相关的,而 $result_1$ 和 $result_3$ 不是;对用户 *C* 来说,$result_3$ 是相关的,而 $result_1$ 和 $result_2$ 不是.

```
⟨company⟩
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨members⟩
      ⟨name⟩Kurt⟨/name⟩
      ⟨name⟩Madonna⟨/name⟩
    ⟨/members⟩
    ⟨project⟩database⟨/project⟩
  ⟨/department⟩
  ⟨department⟩
    ⟨manager⟩Madonna⟨/manager⟩
    ⟨members/⟩
    ⟨project⟩engine⟨/project⟩
  ⟨/department⟩
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨members/⟩
    ⟨project⟩engine database⟨/project⟩
  ⟨/department⟩
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨members/⟩
    ⟨project⟩database engine⟨/project⟩
  ⟨/department⟩
⟨/company⟩
```

Fig.1 An XML fragment

图 1 一个 XML 片段

```
result1:
⟨company⟩
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨project⟩database⟨/project⟩
  ⟨/department⟩
⟨department⟩
    ⟨project⟩engine⟨/project⟩
  ⟨/department⟩
⟨/company⟩
result2:
⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨project⟩engine database
    ⟨/project⟩
  ⟨/department⟩
result3:
⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨project⟩database engine
    ⟨/project⟩
⟨/department⟩
```

Fig.2 Query results of keyword query Q={*Bob*,*databse*,*engine*}

图 2 查询 Q={*Bob*,*databse*,*engine*}的查询结果

从上述例子可以看出,即使用户提交了相同的查询,对于不同的用户,结果与查询的相关性可能不同.在这种情况下,根据任意关键字查询,要找到一个确定的打分函数来为结果的相关性打分是不合适的.没有打分函数,如何根据用户的不同需求识别相关的结果,成为本文所要解决的问题.

考虑到用户需求不同会导致评价结果相关性的标准存在冲突,而 skyline[5]能够帮助用户做出智能的决策,我们发现 skyline 是解决结果的相关性不可比较这一问题的有效工具,因此,识别与用户查询最相关结果的问题

可转化为如下两个子问题:i) 如何定义能够满足多种用户查询需求的 skyline 上的属性;ii) 如何应用 skyline 查询来处理 XML 数据流上的 Top-*K* 关键字查询.本文提出了一种新颖的技术解决这两个问题.

本文的主要工作概括如下:

(1) 考虑到相同的关键字查询可能有不同的查询需求,skyline 被应用于 XML 数据流上的关键字查询,这是 XML 数据流关键字查询里对查询结果选择的一个新角度;

(2) 本文提出了一种新颖的 XML 数据流上的 Top-*K* 关键字查询:基于松弛 skyline 的 XML 数据流上的 Top-*K* 关键字查询,简称 LSK 查询;

(3) 提出了一种有效的基于松弛 skyline 的 XML 数据流上的 Top-*K* 关键字查询算法;

(4) 在 LSK 查询的基础上提出了多查询下的基于松弛 skyline 的 XML 数据流上的 Top-*K* 关键字查询,简称 MLSK 查询;

(5) 提出了一种有效支持 MLSK 查询的处理算法.

本文第 1 节给出预备知识.第 2 节介绍 XML 数据流上的基于 skyline 的关键字查询.第 3 节给出处理基于松弛 skyline 的 XML 数据流上的 Top-*K* 关键字查询的算法.第 4 节介绍多 LSK 查询并给出一种快速、有效的多 LSK 查询处理算法.第 5 节给出实验.第 6 节是相关工作.最后是全文总结.

1 预备知识

在 XML 数据流中,元素表示为固定顺序的序列.只有当它们被存储以后,才能被重复访问.

XML 文档可以表示为一棵有序、有标签的树,其中,文档中的元素对应树上的结点,文档中元素之间的嵌套关系对应树上的边.对于一棵 XML 树 $T=\{V,E,L\}$(V 是结点集合,E 是边集合,L 是叶子结点集合),我们定义元素 n 和 e 的距离为 n 到 e 的路径的边数,记为 $d(n,e)$.如果 d 和 e 是不可达的,则 $d(n,e)=\infty$.

处理 XML 数据流的一种常用的程序接口是 SAX(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 SAX 事件.在 XML 数据流的处理中,遇到一个元素的开始标签时,触发 BEGIN(tag)事件;遇到一个元素的结束标签时,触发 END(tag)事件;遇到某个元素的值时,触发 TEXT()事件.如果一个元素已经触发 BEGIN(tag)事件而未触发 END(tag)事件,则称对应 XML 树里的此结点为活跃结点.

以下给出关键字查询相关概念的定义.

定义 1(关键字匹配). 给定查询关键字集合 $W=\{K_1,K_2,\ldots,K_N\}$和 XML 数据流上某一叶子结点 u,如果 u 的文本内容中包含 W 中的某关键字 $K_i(1\leqslant i\leqslant N)$,则说明结点 u 和查询关键字 K_i 匹配,记做 $K_i\subseteq u$.

定义 2(SLCA). 在给定如下初始条件的情况下:一是对应 XML 数据的标签有向树 $G=(V,E,L)$,二是关键字集合 $W=\{K_1,K_2,\ldots,K_N\}$,SLCA 问题就是求解 G 中所有满足如下条件的子树的根结点:

(1) 子树必须包含关键字集合 W,即 W 中的任意关键字必然分布于该子树的叶结点;

(2) 后代中不存在更小的子树也包含 W.

这样的子树也称为满足关键字序列 W 的最紧致 XML 片段.

定义 3(SLCA 查询结果). 给定一个在 XML 流上的关键字查询 Q,在这个 XML 流上的 XML 片段 t 被称为一个 SLCA 查询结果,当 t 的根结点是查询 Q 的 SLCA,且不存在一个 t 子片段包含 Q 中所有关键字.显然,对于一个 SLCA 查询结果 t,t 的每个叶子都包含一个别的叶子所不包含的关键字;否则,去掉叶子 t,则存在比 t 更小的子树也包含 W.

2 基于 skyline 的 XML 数据流上的关键字查询处理

正如本文开始部分所讨论的,对于 XML 数据流上的关键字查询,skyline 能够用于帮助选择与查询最相关的结果.本节将给出基于 skyline 的 XML 数据流上的关键字查询的形式化定义,下一节我们将对这个问题返回的 SLCA 查询结果数量进行限制.

对于查询 $W=\{K_1,K_2,\ldots,K_N\}$和 W 的 SLCA 查询结果 T,我们用集合 R 来描述结果 T 与 W 的相关性,其中 $R=\{R(i,j)|R(i,j)$为关键字 K_i 和 K_j 在结果 T 中的相关性$\}$.

其合理性描述如下.

显然,对一个 SLCA 查询结果 T 的任意两个叶子 a,b 来说,设它们的最小公共祖先为 s,如果 $d(a,s)+d(s,b)$的值越小,则意味着 a 所包含的任意关键字与 b 所包含的关键字的相关性越大.根据距离的定义可知,$d(a,b)=d(a,s)+d(s,b)$.因此,这两个叶子的距离越小,它们所包含的关键字之间的相关性也就越大.

设 T 为查询 W 的某 SLCA 查询结果,对每个关键 K_i 来说,定义集合 $A_i=\{a|a$ 是查询结果 T 的叶子结点且 a 包含关键字 $K_i\}$,即 T 中所有包含关键字 K_i 的叶子集合.对任意 SLCA 查询结果 T,定义 T 中任意两个关键字 K_i 和 K_j 的相关性为 $\min_{a_i\in A_i,a_j\in A_j} d(a_i,a_j)$,记为 $R_T(i,j)$.显然,$R_T(i,j)=R_T(j,i)$.

对一个关键字查询 $Q=\{K_1,K_2,\ldots,K_N\}$,定义 $P_Q=\{(i,j)|K_i\in Q,K_j\in Q$ 且 $i<j\}$.P_Q 中所有二元组按照两个维度基数排序.记(i,j)的排序号为 $o(i,j)$,在 P_Q 中记为 k_{ij},即 $k_{ij}=o(i,j)$.对查询 Q 的任意 SLCA 查询结果 T 来说,$R_T(i,j)$又可以表示为 $R_{k_{ij}}(T)$.一个 SLCA 查询结果 T 可以表示为一个向量 $(R_1(t),R_1(t),...,R_{|P_Q|}(T))$.

定义 4(支配). 给定一个 XML 数据流上的关键字查询 $Q=\{K_1,K_2,\ldots,K_N\}$,和两个 SLCA 查询结果 T 和 T',称 T 支配 T',记为 $T\prec T'$,如果满足下述两个条件:(1) $\forall k(1\leqslant k\leqslant|P_Q|),R_k(T)\leqslant R_k(T')$;(2) $\exists i(1\leqslant i\leqslant|P_Q|),R_i(T)<R_i(T')$.

由相关性的定义来看,对任意关键字查询 Q 的两个 SLCA 查询结果 T_1 和 T_2 来说,T_1 支配 T_2 意味着结果 T_1 比结果 T_2 与查询更相关.例如在图 2 中,$result_2$ 支配 $result_1$,而从语义上来说,$result_2$ 确实比 $result_1$ 与 Q 更相关.因此,对查询 Q 的所有 SLCA 查询结果来说,最相关的 SLCA 查询结果一定不被任何其他 SLCA 查询结果所支配.基于这种考虑,我们定义关键字查询 Q 的 skyline 点.

定义 5(skyline 点). 给定一个 XML 数据流上的关键字查询 $Q=\{K_1,K_2,\ldots,K_N\}$,SLCA 查询结果 T 如果没有被任何其他的 SLCA 查询结果支配,则 T 被称为所接受到的 XML 片段的 skyline 点.

定义 6(基于 skyline 的 XML 数据流上的关键字查询). 给定一个 XML 数据流上的关键字查询 $Q=\{K_1,K_2,\ldots,K_N\}$,在这个数据流上的基于 skyline 的关键字查询,即获得目前接受到的 XML 片段中的全部 skyline 点的集合.

基于关键字集合 $Q=\{K_1=Bob,K_2=database,K_3=engine\}$在图 1 所示 XML 片段上的 SLCA 查询结果如图 2 所示,这些 SLCA 查询结果中关键字之间的相关性如下:

- $Ro(1,2)(result_1)=2,Ro(1,3)(result_1)=4,Ro(2,3)(result_1)=4$;
- $Ro(1,2)(result_2)=2,Ro(1,3)(result_2)=2,Ro(2,3)(result_2)=0$;
- $Ro(1,2)(result_3)=2,Ro(1,3)(result_3)=2,Ro(2,3)(result_3)=0$.

$result_2$ 和 $result_3$ 都不被其他任何 SLCA 查询结果所支配,因此他们是查询 Q 的 skyline 点.$result_1$ 被 $result_2$ 和 $result_3$ 支配,所以它不是 Q 上的 skyline 点.故基于 skyline 的关键字集合 Q 的查询结果是 $result_2$ 和 $result_3$.如引言中所分析的:对于所有的用户,$result_1$ 都不是一个好的结果;但是对用户 A 和用户 B,$result_2$ 是好的;对用户 A 和用户 C,$result_3$ 是好的.因此,$result_2$ 和 $result_3$ 应该返回给用户.这与基于 skyline 的关键字查询得出的结果是一致的.

3 基于松弛 skyline 的 XML 数据流上的关键字查询处理

在上一节中,我们描述了基于 skyline 的关键字查询.这种查询的不足之处是当用户可能想要更多查询结果时,用户不能对查询结果的数量进行设置.我们用一个例子来说明.

例如,在图 3 所示的 XML 片段上执行关键字查询 $Q=\{Bob,database,engine\}$.$Q$ 的两个 SLCA 查询结果 $result_1$ 和 $result_2$ 返回,如图 4 所示.对于 Q 来说,用户 A 想要找到 *Bob* 开发的有关 *database* 或 *engine* 的工程.对于这样的需求,$result_1$ 和 $result_2$ 都是好结果.然而,如果我们只返回 skyline 点的话,只有 $result_1$ 能返回,其他有用的 SLCA 查询结果就丢失了.

从这个例子中可以看出,仅仅返回查询中的 skyline 点可能是不够的.有时候,用户需要更多的结果.因此在

本节中,基于 skyline 的关键字查询被扩展到返回一定数量结果的基于 skyline 的关键字查询.我们称这种查询为基于松弛 skyline 的 XML 数据流上的关键字查询.

```
⟨company⟩
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨members/⟩
    ⟨project⟩database engine
    ⟨/project⟩
  ⟨/department⟩
⟨department⟩
    ⟨manager⟩Madonna⟨/manager⟩
    ⟨members⟩
      ⟨name⟩Bob⟨/name⟩
    ⟨/members⟩
      ⟨project⟩engine database
      ⟨/project⟩
  ⟨/department⟩
⟨/company⟩
```

Fig.3 An XML fragment

图 3 一个 XML 片段

```
result1:
  ⟨department⟩
    ⟨manager⟩Bob⟨/manager⟩
    ⟨members/⟩
    ⟨project⟩database engine
    ⟨/project⟩
  ⟨/department⟩
result2:
⟨department⟩
    ⟨members⟩
      ⟨name⟩Bob⟨/name⟩
    ⟨/members⟩
      ⟨project⟩engine database
      ⟨/project⟩
  ⟨/department⟩
```

Fig.4 Query results of keyword query Q={*Bob*,*databse*,*engine*}

图 4 查询 Q={*Bob*,*databse*,*engine*}的查询结果

定义 7(基于松弛 skyline 的 XML 数据流上的关键字查询). 给定查询 $Q=\{W,K\}$,$W=\{K_1,K_2,\ldots,K_N\}$为关键字集合,K 为对返回结果的数量限制.在 XML 数据流上 Q 的结果为 W 的 SLCA 查询结果 D 的一个子集,记为 LS_D,该子集满足:i) $\forall u\in LS_D$ 不被 D/LS_D 中任意 SLCA 查询结果所支配;ii) $|LS_D|=K$.我们称查询 Q 为基于松弛 skyline 的 XML 数据流上的关键字查询.为方便,我们称这种查询为 LSK 查询.

为了处理 XML 数据流上的 LSK 查询,我们提出了一种有效的算法.该算法的基本思想是:对一个 LSK 查询 $Q=\{W,K\}$,维护一个包含 LSK 查询结果集合的超集,称为中间结果集合.当有新的 SLCA 查询结果生成时,需要更新该中间结果集合.

首先,我们提出中间结果集合的更新算法,然后提出利用更新算法求解 LSK 查询结果的算法.

3.1 中间结果更新算法

本节提出了 LSK 查询中间结果的更新算法.中间结果集合必须包含 Q 的 LSK 查询结果;且为了减少从中间结果集中挑选最终的查询结果的处理时间,应使得该中间结果集尽量地小.在定义中间结果集之前,我们定义 skyline 层作为中间结果的单元.

定义 8(skyline 层). 给定一个关键字查询 $Q=\{K_1,K_2,\ldots,K_N\}$和 XML 数据流的一个 SLCA 查询结果集合 M,我们称 M 的一个子集为 skyline 层 i($i\geqslant 1$),记为 L_i 如果它满足下述条件:

i) $\forall a,b\in L_i$,a 不被 b 支配,b 也不被 a 支配;

ii) $\forall a\in L_i$:if $i\geqslant 2$,$\exists b\in L_{i-1}$,$b\prec a$;if $i=1$,$\forall b\in L_j(j\geqslant 1)$,$a$ 不被 b 支配.

由此可知,M 可被分成从 skyline 层 L_1 到 skyline 层 $L_n(n\geqslant 1)$,n 个 skyline 层.利用这个定义,我们给出分层规则从 SLCA 查询结果集合 M 中选择 LSK 查询 $Q=\{W,K\}$的中间结果,记为 L.

分层规则:初始化,$L=\varnothing$.对于关键字查询 Q 的 SLCA 查询结果集合 M,首先 L_1 被加到 L 里.如果$|L|$大于 K 或者所有的 SLCA 查询结果都已经处理了,这个选择结束;否则,下一个 skyline 层 L_2 被选择加入到 L 中.如此不断处理,直到处理到某一层 L_t,满足下面的约束:

i) $|L_1\cup L_2\cup\ldots\cup L_t|\geqslant K$;

ii) if $t>1$,$|L_1\cup L_2\cup\ldots\cup L_{t-1}|<K$.

由上述选择可看出,$L=L_1\cup L_2\cup\ldots\cup L_t$.

我们用一个例子来说明这个分层规则.对于一个关键字查询 W,用点表示其 SLCA 查询结果如图 5 所示(横纵坐标均代表 skyline 查询的属性).我们用圈、星和三角形来区分在不同 skyline 层上的 SLCA 查询结果,圈结

点为 L_1 层的结点,星结点为 L_2 层的结点,三角形结点为 L_3 层的结点.对于一个 LSK 查询 Q 来说,设 W 为其关键字集合,K=5.基于分层规则,L_1 首先被选中,因此点 7 被加入 Q 的中间结果集 L.由于目前$|L|$=1<K,skyline 层中第 2 层中的点集合 L_2 被加入 L.这时,$|L|$=4<K.因此,第 3 层的点集合 L_3 也被加入 L.此时$|L|$=7>K,终止条件满足,因此该过程停止.

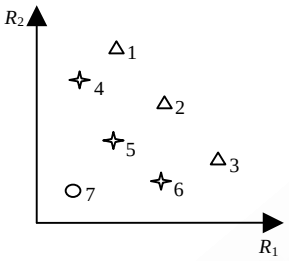

Fig.5 An example of garlic rule

图 5 一个分层规则的例子

当根据分层规则生成查询 Q 的临时结果集 L 后,查询 Q 的结果就能够很容易地通过从 L 的最后一个 skyline 层 L_t 里随机删除若干结点而生成.也就是说,如果 t=1 且$|L_1|$>K,则从 L_1 中随机选择 K 个结果作为最终结果;否则,选择从 L_1~L_{t-1} 层的所有结果,并从 L_t 层中随机选择 $K-|L_1\cup L_2\cup\ldots\cup L_{t-1}|$个结果,记为 L_t',则查询 Q 的最终结果为 $L_1\cup L_2\cup\ldots\cup L_{t-1}\cup L_t'$.这样的选择规则称为最底层选择规则.因此,对 XML 数据流上的一个 LSK 查询 Q 来说,如果 Q 的中间结果集被维护,则最终的查询结果很容易就生成了.为有效处理 XML 数据流上的 LSK 查询,我们提出了一种对中间结果集 L 的更新算法,具体描述见算法 1.

算法 1. 中间结果更新算法.Update_intermediate(Q,L,m)

```
Input: (Q,L,m)   //Q={W={K1,K2,…,KN},K},L 是中间结果集,m 是新生成的 SLCA 查询结果;
Initialization: Isdominated=false
for i=t to 1 do
    for each r∈Li do
        if r dominates m then
            Isdominated=true
                Break;
    if Isdominated=true then
        l=i+1
        Break;
if Isdominated=false then
    l=1
if l=t+1 then
    if |L|<K then
        add a new layer Lt+1={m} to L
else
    add m to Ll
    E={u|u∈Ll∧m dominates u}
    while l≠t do
        Ll=Ll–E
            E={u|u∈Ll+1∧∃v∈E,v dominates u}
```

L_t=L_t–E

if |L|–|E|<K **then**

add a new layer L_{t+1}=E to L

在这种算法中,一个新生成的 SLCA 查询结果 m 需要和 L 里的结果从最低层 L_t 开始逐层向上进行比较,直到找到某一层 L_i 中的一个结果 u 能够支配 m 或者发现没有任何中间结果支配 m 为止.如果 m 被某一层 L_i 的结点所支配,就意味着 m 应该处于 L_i 层的下一层.如果 L_i 是 L 的最底层且|L|≥K,则直接删除 m;否则,m 被插入 L_{i+1} 层,L_{i+1} 层中被 m 支配的结点被插入 L_{i+2} 层.这个将被支配结点下推的过程一直持续到下推结点集在所在层,记为 E,没有被支配的结点为止或结点集合 E 被插入到最后一层 L_t.一旦 E 被插入最后一层,被 E 中结点所支配的结点集 E'应该插入到 L_t 的下一层.如果|L|+1–|E'|≥K,则删除 E';否则,建立 L_t 的下一层,结点集合为 E'.如果顶层 L_1 中没有任何结点支配 m,则说明 L 中没有支配 m 的结点,则 m 应该被插入 L_1,L_1 中由 m 支配的结点按照上述过程处理.为了能提高将结点下推的效率,对中间结果集中每个结点 e,设 e 在 L_i 层,都维护一个到 L_{i+1} 层的指针集,指向 L_{i+1} 层中被 e 支配的结点集.

例如,在图 5 所描述的中间查询结果上选取 skyline 点,设 K=5,每个结点的 id 是他们被处理的顺序.当处理 $result_4$ 时,$result_1$,$result_2$ 和 $result_3$ 都已经生成了.这时,L=L_1={$result_1$,$result_2$,$result_3$},t=1.由于 $result_4$ 支配 L_1 中的 $result_1$ 并且 $result_4$ 不被 $result_2$,$result_3$ 支配,因此它被插入 L_1,$result_1$ 被下推.由于|L|<K,建立新的一层 L_2={$result_1$}.当处理 $result_5$ 时,$result_5$ 支配 L_2 中的 $result_1$、L_1 中的 $result_2$,且不被 $result_3$ 和 $result_4$ 支配,因此它被插入 L_1,$result_2$ 被推向下一层 L_2.由于 $result_2$ 不被 L_2 中的结点支配,$result_2$ 被插入 L_2 中.当处理 $result_6$ 时,其操作与 $result_5$ 的操作类似.当处理 $result_7$ 时,L_1={$result_4$,$result_5$,$result_6$}且 L_2={$result_1$,$result_2$,$result_3$},$result_7$ 支配 L_1 中的 $result_4$,$result_5$ 和 $result_6$.由于没有结点支配 $result_7$,故 $result_7$ 被插入 L_1.$result_4$,$result_5$ 和 $result_6$ 被下推到 L_2.由于 L_1 和 L_2 中结点个数少于 K,因此新的一层 L_3={$result_1$,$result_2$,$result_3$}被创建.

3.2 在XML数据流上的LSK查询处理算法

根据上一节所提出的中间结果更新算法,在本节,我们提出在 XML 数据流上的 LSK 查询处理算法.

如第 1 节中所介绍的,处理 XML 数据流的一种常用的程序接口是 SAX,即 XML 简单应用程序接口.我们算法的基本框架是:当 n 的 BEGIN 事件被触发时,我们对结点 n 进行初始化;当 n 的 TEXT 事件被触发时,我们找到结点 n 所包含的关键字并且更新结点 n 的状态;当 n 的 END 事件被触发时,我们求解以 n 为根结点的 SLCA 查询结果,并且更新中间结果集 L 和生成当前 LSK 查询的结果集,记为 LS.

在算法处理过程中,为了表示 XML 数据流中结点之间的结构信息,我们采用 Dewey 编码[6].每个活跃结点 n 的状态用一个 N 位二进制串表示,记为 s_N.如果 n 的任何后代包含关键字 K_i,则 s_N 的第 i 位置为 1;否则置为 0.每个结点的初始化状态 s_N 的每一位均为 0.

在描述该算法之前,我们首先介绍支持算法的数据结构.在算法中,我们用一个栈来存储当前活跃结点的信息,按照结点开始事件被触发的顺序记录活跃结点的编码和状态信息.

数组 P 用来记录每个当前非叶子的活跃结点为根的子树所包含的关键字信息.对每个非叶子结点 i,P[i][j]记录结点 i 的后代中包含关键字 K_j 的所有后代.数组 L 存储当前的中间结果集.数组 LS 存储查询的当前结果集.

每个结点 n 的初始化包括对结点 n 的状态 s_N 的初始化,对 n 的 Dewey 编码和将结点 n 压栈.

当结点 n 的 END 事件被触发时,以 n 为根的临时结果集通过数组 P 中的 P[n][1]~P[n][N]的笛卡尔积生成,即 P[n][1]×P[n][2]×…×P[n][N].中间结果集 L 根据新生成的结果按照最底层规则进行更新.算法的形式化描述见算法 2.

算法 2. LSK 查询处理算法.

Input: Q={W,K};

Output: LS //LS 的查询 Q 的结果

BEGIN (**node** ***n***)

$n.state$=00…0

Stk.push(*n*)
endfunction
TEXT (**node** ***n***)
for each $K_i \in W$ **do**
if $K_i \subseteq n.v$ **then**
$n.state=n.state$ OR 2^i
$P[m][i].insert(n.id)$ //*m* is the parent of node *n*
endfunction
END (**node** ***n***)
if $n.state=2^N-1$ **then** //the descendants of node *n* contain all the keywords
Generate all the query result set *M* with root *n*
for each result *m* in *M* **do**
Update_intermediate(*Q*,*L*,*m*)
Select *LS* from *L* with the lowest layer rule
else
Stk.top(·).*state*=*Stk.top*(·).*state* OR *n.state*
Stk.pop(·)
endfunction

我们用一个例子来说明算法 2 的流程.设查询 $Q=\{W=\{K_1=a,K_2=b,K_3=c\},K\}$.XML 数据流上每个结点的 Dewey 编码如图 6 所示.当结点 1.1 的 TEXT 事件被触发时,我们发现结点 1.1 包含关键字 K_1 和 K_2,因此结点 1.1 的状态更新为 011,我们把结点 1.1 放入 *P*[1][1]和 *P*[1][2]中.当结点 1.1 的 END 事件被触发时,我们将结点 1 的状态更新为 011.当结点 1.2 的 TEXT 事件被触发时,我们将结点 1.2 的状态更新为 100,将结点 1.2 放入 *P*[1][3]中.当结点 1.2 的 END 事件被触发时,我们将结点 1 的状态更新为 111.当结点 1 的 END 事件被触发时,由于结点 1 的状态的每一位均为 1,故生成所有以结点 1 为根结点的 SLCA 查询结果,更新 *L*,再根据 *L* 生成 *LS*.

⟨1⟩
⟨1.1⟩*ab*⟨/1.1⟩
⟨1.2⟩
⟨1.2.1⟩*c*⟨/1.1.2.1⟩
⟨/1.2⟩
⟨/1⟩

Fig.6 An example of Algorithm 2
图 6 算法 2 的一个例子

3.3 算法复杂度分析

在这节中,我们分析了算法的时间复杂度和空间复杂度.

由于时间复杂度受 XML 的模式影响,我们只分析最坏情况下的时间复杂度.

对算法 1 来说,设关键字的个数为 *N*,因此,skyline 属性的个数为 $N\times(N-1)/2$,故两个结果进行比较的时间复杂度为 $N\times(N-1)/2$.设 *L* 的最大规模为 $L_{\max}(L_{\max}\geqslant K)$,在最坏情况下,每个 SLCA 查询结果与 *L* 中的每个元素进行比较,故算法 1 的最坏时间复杂度为 $O(L_{\max}\times N^2)$.

下面我们逐步分析算法 2 的时间复杂度.

显然,对 BEGIN 事件的处理的复杂度 $O(1)$.当 TEXT 事件被触发时,对每个关键字 *k*,利用 KMP 算法判断结点 *n* 是否包含 *k*,时间复杂性为 $O(L_n)$,其中,L_n 为结点 *n* 内容的长度.仍假设关键字的个数为 *N*,则 TEXT 事件被触发所需要的执行的操作的时间复杂度为 $O(N\times L_n)$.当结点 *n* 的 END 事件被触发时,如果 *n* 是个叶子,最坏情况

下叶子包含所有的关键字,则更新 L 的时间复杂度为 $O(L_{\max}\times N^2)$;且在最坏情况下,生成 LS 的时间复杂度为 $O(K)$ ($K\leqslant L_{\max}$),因此,叶子结点的 END 事件被触发后所执行的操作时间复杂度为 $O(L_{\max}\times N^2)$.若 n 不是叶子,则需要生成以 n 为根结点的所有 SLCA 查询结果,设 $\max\{|P[n][i]||1\leqslant i\leqslant N\}=M$,则 $P[n][1]\times P[n][2]\times\ldots\times P[n][N]$的时间复杂度为 $O(M^N)$.在最坏情况下,每个新生成的 SLCA 查询结果均要插入 Q,则更新 L 和生成 LS 的时间复杂度为 $O(L_{\max}\times N^2)$.因此,非叶子结点的 END 事件被触发后所执行的操作的时间复杂度为 $O(M^N\times L_{\max}\times N^2)$.故在最坏情况下,END 事件被触发后所执行的操作的时间复杂度为 $O(M^N\times L_{\max}\times N^2)$.

设 XML 数据流中非叶子结点个数为 M_1,叶子结点个数为 M_2,结点内容长度最大值为 L_{node},则算法 2 的时间复杂度为 $O(\max\{M_1\times M^N\times L_{\max}\times N^2, M_2\times N\times L_{node}\})$.

设 XML 数据流中 XML 片段最大深度为 H,则在最坏情况下,栈的空间复杂度为 $O(H)$.设 $\max\{|P[i][j]|\}$为 M,则 P 的空间复杂度为 $O(H\times M\times N)$.设 L 的最大规模为 $L_{\max}$,每个 SLCA 查询结果对应的树至多$(N\times H+1)$个结点,故 L 的空间复杂度为 $O(L_{\max}\times N\times H)$.因此,这个算法的空间复杂度为 $\max\{O(H\times M\times N), O(H\times N\times L_{\max})\}$.

4 多 LSK 查询处理算法

当系统中用户提出了多个 LSK 查询时,对于同一个输入数据流,需要对这些 LSK 查询都进行处理,为每个查询选择符合要求的结果.本节将多个 LSK 查询看成多 LSK 查询,统一进行处理,其目的是在接收到数据流中的数据片段时,快速地确定此片段满足哪些查询.多 LSK 查询定义如下:

定义 9(MLSK 查询(基于松弛的 skyline 的 XML 数据流上的关键字多查询)). MLSK 查询是一个查询集合 $S=\{Q_1,Q_2,\ldots,Q_n\}$,其中,$Q_i=\{W_i,K_i\}$,$1\leqslant i\leqslant n$,$W_i=\{k_{i1},k_{i2},\ldots,k_{iN}\}$为查询 Q_i 的关键字集合,K_i 为对查询 Q_i 返回结果的数量限制,W 为所有查询的关键字集合,$W=\bigcup_i W_i$.在 XML 数据流上 MLSK 查询 S 的结果是由 n 个查询 Q_1~Q_n 的 LSK 查询结果集合组成的集族 R.

对于 MLSK 查询,最直接的方法是线性地调用第 3 节中提出的 LSK 查询算法.该方法在效率和有效性上存在着如下不足:i) 效率方面:该方法的效率与查询个数呈线性关系,因此随着用户数量增加,查询效率会线性降低;ii) 有效性:由于内存有限,若新的数据流片段到来的速度比处理所有用户查询的效率快的话,会导致数据丢失,且不能保证查询结果的正确性.因此,如何高效处理 MLSK 查询成为 MLSK 查询处理的关键,具体地说,即如何建立有效的数据结构对所有关键字加以管理,并对每个 LSK 查询的包含的关键字信息加以管理,从而快速判断接收到的数据片段满足哪些查询中的关键字约束.在第 4.1 节中,我们提出了有效管理 MLSK 查询的索引结构,第 4.2 节中提出了基于该结构的 MLSK 处理算法 MULSK.

4.1 MLSK查询的索引结构

考虑到很多应用环境中,MLSK 查询更新不频繁,因此我们对 MLSK 查询建立索引,有效管理关键字和查询,其目的是快速判断一个 XML 元素满足哪些查询.该索引包含两个部分:KIndex 和 QIndex,KIndex 用于在 TEXT 事件触发时判断结点的内容所包含的关键字,QIndex 用于在 END 事件触发时判断该结点是哪些查询的 SLCA 结点.

当对应结点内容 n 的 TEXT 事件触发时,MLSK 查询处理需要判断 n 包含哪些关键字,直接的实现方法是依次判断每个关键字是否被 n 包含.显然,这个方法的时间复杂度与关键字个数成正比.而实际应用中,MLSK 查询可能包含大量的关键字.为了快速确定 n 包含的关键字集合,我们提出了一种支持关键字快速匹配的索引结构 KIndex.KIndex 基于 Trie 树[7]建立,为了节省空间,我们选用双数组[8]实现 Trie 树.基于 KIndex,判断 n 所包含的关键字集合所需要的时间只与关键字的长度和结点内容的长度成正比,而与关键字集合大小无关.设 n 的长度为 L_n,关键字的最长长度为 $L_{\max}$,则采用索引 KIndex 所需要的判断时间的时间复杂度为 $O(L_{\max}L_n)$,而不采用索引结构时,利用 KMP 算法判断 n 是否包含关键字 k 的复杂性是 L_n,所需要的判断时间复杂度为 $O(|W|L_n)$.考虑到包含大量关键字的集合中$|W|$的值很大而每个关键字的长度相对较小,KIndex 适用于包含大量关键字的 MLSK 查询.

我们用一个例子来说明 KIndex 的结构和操作.设给定查询集合 $S=\{Q_1,Q_2\}$,Q_1={blue,moon},Q_2={bus,map},则 W={blue,bus,map,moon}.根据 W 建立的 KIndex 如图 7 所示.

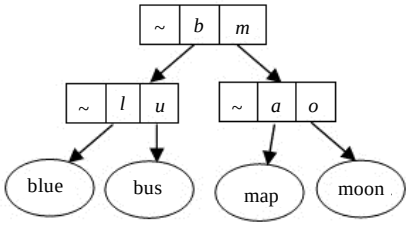

Fig.7 An example of KIndex

图 7 一个 KIndex 的例子

设当接收到结点 a,a 的内容为“wide bus”时,查找 KIndex 索引树,从根结点开始对 a 的内容一次一个字母进行比较,根据 a 当前位的字母决定所走的支.例如,a 内容第 1 位是‘w’,KIndex 根结点中无该字母,继续将 a 的内容与根结点一次一个字母比较,直到在 KIndex 根结点中找到与 a 的当前位相同的字母‘b’,因此决定走 KIndex 当前结点的左支,a 的下一位是 u,故决定走 KIndex 当前结点的右支,遇到叶子结点值为“bus”,发现 a 的后一位也是‘s’,因此判断 a 的内容包含关键字“bus”.

a 的内容长度 L_a=8,W 有 4 个词,故$|W|$=4.利用 KMP 算法判断 a 包含哪些关键字需要的复杂性为$|W|L_a$,则判断 a 包含哪些关键字需要的时间为$|W|L_a$=4×8=32;若采用上述索引结构(如图 7 所示),W 中关键字最长长度 L_{max}=4,则需要的时间不超过 $L_{max}L_a$=4×8=32.可以看出,当$|W|>L_{max}$ 时,采用 KIndex 的时间复杂性小于直接字符串匹配算法的时间复杂性.考虑到大量查询中的关键字集合很大,即$|W|>>L_{max}$,KIndex 在实际中会大大提高匹配效率.

当 END 事件触发时,LSK 查询处理需要判断结点 n 是哪些查询的 SLCA 结点(满足该条件的查询集合记为 Q_n),如果为每个查询 q 检验 n 是否是 q 的 SLCA,则处理时间随查询个数线性增加.因此,我们设计了基于编码的快速判定方法,并提出 QIndex 对编码进行索引,从而根据编码高效地查找其对应的查询.

为了有效地描述结点 n 中的内容包含哪些关键字,我们提出了基于位图的编码 C_n,C_n 包含$|W|$位,若第 $C_n[i]$=1 表示 n 的内容包含关键字 K_i,$C_n[i]$=0 表示 n 的内容中不包含关键字 K_i.类似地,我们为 LSK 查询 q 设计位图编码 C_q,$C_q[i]$=1 表示 $K_i\in W_q$,$C_q[i]$=0 表示 $K_i\notin W_q$.对于结点 n 和查询 q,如果 $C_n\cap C_q=C_q$,则 n 满足 q.

QIndex 利用哈希函数 H 为所有的 C_q 建立索引,使得当处理结点 n 的 END 事件时,可以快速根据 C_n 查找到满足 $C_n\cap C_q=C_q$ 的 q.

为了保证查询结果是 SLCA,我们为每个结点 n 定义状态 T_n,表示 n 的后代中的结点已满足了哪些查询.T_n 可以用一个$|S|$位的位图来表示,其中,$T_n[i]$=1 表示存在 n 的后代结点 n'满足查询 Q_i,$T_n[i]$=0 表示任何 n 的后代结点都不满足查询 Q_i.

根据如下性质,我们可以利用上述编码判断 n 是否是查询 Q_i 的 SLCA 结点.

性质. 若结点 n 和查询 Q_i 满足如下两个条件,则 n 为查询 Q_i 的 SLCA 结点:

1) $C_n\cap C_q=C_q$;

2) $\overline{T_n}\cap 2^i\neq 0$.

性质的条件 1)说明结点 n 的内容包含查询 Q_i 的所有关键字;条件 2)说明结点 n 的后代中没有 Q_i 的 SLCA 结点.根据该性质,当处理对应结点 n 的 END 事件时,按照如下步骤计算 Q_n:

第 1 步:求出 $Q'_n=\{Q_i|Q_i\in S\wedge C_n\cap C_{Q_i}=C_{Q_i}\}$;

第 2 步:根据 Q'_n 求出 $Q_n=\{Q_i|Q_i\in Q'_n\wedge\overline{T_n}\cap 2^i\neq 0\}$.

考虑到在大多数情况下 n 仅包含少数查询中的关键字,我们为查询建立基于哈希表结构的索引 QIndex,以

快速查找 n 所满足的查询.QIndex 利用哈希函数 H 可以根据编码 C_q 快速地找到对应的查询 q,即 $QIndex(C_q)=q$.对于一个结点 n,首先生成编码集合 $S_{C_n}=\{C|C\wedge C_n=C\}$,再对于 S_{C_n} 中的每个元素 C 求得的查询 $q_C=QIndex(C)$,这样得到的 q_C 的集合即是 Q'_n.

下面用一个例子来说明如何计算 Q_n.设查询集合为 $S=\{Q_1,Q_2,Q_3\}$,$Q_1=\{\{k_1,k_2\},3\}$,$Q_2=\{\{k_1,k_4\},3\}$,$Q_3=\{\{k_3,k_4\},3\}$.当结点 n 的 END 事件被触发,如果此时 C_n=1011,T_n=001,表示 n 的内容包含关键字集合$\{k_1,k_2,k_4\}$且 n 有后代结点满足查询 Q_1.首先生成编码集合 $S_{C_n}=\{0001,0010,1000,0011,1010,1001,1011\}$,再对于 S_{C_n} 中的每个元素 C 查询 QIndex,得到结果 $Q'_n=\{Q_1,Q_2\}$.考虑 Q_1,由于 $\overline{T_n}\cap 2^1=0$ 不满足条件,故过滤掉 Q_1;类似地,可以判断 Q_2 满足条件,故 $Q_n=Q'_n/\{Q_1\}=\{Q_2\}$.

考虑到关键字或查询很多的情况下编码占用的空间很大,为节省存储空间,我们提出了编码压缩策略.该策略可以根据给定的压缩比对编码进行压缩.对于给定的压缩比 CR,在编码 C 对应压缩的编码 PC 中,$PC[i]=\bigvee_{j=1}^{CR}C[(i-1)\cdot CR+j]$.从而对于结点 n,$PC_n[i]$=1 表示$\exists q\in\{k_j|(i-1)\cdot CR+1\leqslant j\leqslant i\cdot CR\}$且 q 包含关键字 k.基于压缩编码进行查询查找的方法与上述方法一致,但根据 QIndex 对每个编码 C 查找到的结果是查询集合 Q_C,考虑到 Q_C 中可能有查询不满足条件,需要对 Q_C 中查询的编码进行进一步验证.

4.2 MLSK查询处理算法

为了处理 XML 数据流上的 MLSK 查询,我们提出了 MULSK 查询处理算法.该算法的基本框架与第 3.2 节中的 LSK 查询处理算法一致.在处理 TEXT 事件时,使用第 4.1 中所提出的 KIndex 查找内容所包含的关键字,从而更新结点的状态;在处理 END 事件时,利用 QIndex 查找当前结点是哪些查询的 SLCA,当前的元素即是这些查询的结果.算法的伪代码见算法 3.

算法 3. MULSK 查询处理算法.

```
Input: S={Q1,Q2,…,Qn};
Output: R={LS1,LS2,…,LSn}    //LSi is the LSK result set of query Qi.
Initialization: build a Keyword Index of W, KIndex; build a Query Index of S, QIndex
BEGIN (node n)
    Cn=00…0
    Tn=00…0
    Stk.push(n)
endfunction
TEXT (node n)
    keyword set K(w)=KIndex.find(n.value)
    Update Cn according to K(w)
    for each ki∈K(w) do
        P[m][i].insert(n.id)    //m is the parent of n
endfunction
END (node n)
    Query set Q(n)=QIndex.find(Cn)
    for i=1 to n do
        if Tn[i]=1 then
            delete Qi from Q(n)
    for each Qi∈Q do
        Generate the query result set M(Qi) with root n
        Tn[i]=1
```

for each result m in $M(Q_i)$ **do**

Update_intermediate(Q_i,L_i,m)

Select LS_i from L_i with the lowest layer rule

$C_{Stk.top(\cdot)}$=$C_{Stk.top(\cdot)}$ OR C_n

$T_{Stk.top(\cdot)}$= $T_{Stk.top(\cdot)}$ OR T_n

Stk.*pop*(·)

endfunction

我们用一个例子来说明 MULSK 算法.设给定查询集合 S={Q_1,Q_2},Q_1={{blue,moon},3},Q_2={{bus,map}, 3},则 W={k_1=blue,k_2=bus,k_3=map,k_4=moon}.待处理的 XML 片段如图 8 所示.当结点 b 的 TEXT 事件被触发时,利用 KIndex 查找出 b 所包含的关键字为 k_1 和 k_4,因此结点 b 的状态 C_b 更新为 1001.当 b 的 END 事件被触发时,根据 C_b,生成 S_{C_b} ={0001,1000,1001},再根据索引 QIndex 查找 S_{C_b} 中每个元素编码值所匹配的查询集合组成的集合 Q'_b ={Q_1}.由于 T_b=00,表示 b 的后代不满足任何查询,故结点 b 为查询 Q_1 的 SLCA 结点.此时,将 b 的父结点 a 的状态 C_a 更新为 1001,T_a 更新为 01.当结点 c 的 TEXT 事件被触发时,根据 KIndex 查找出 c 所包含的关键字为 k_2 和 k_3,c 的状态 C_c 更新为 0110.当结点 c 的 END 事件被触发时,根据 C_c,生成集族 S_{C_c} ={0010,0100,0110},再根据索引 QIndex 查找 S_{C_c} 中每个元素编码值所匹配的查询组成的集合 Q'_c ={Q_2}.由于 T_c=00,故结点 c 为查询 Q_2 的 SLCA 结点.此时,将 c 的父结点 a 的状态 C_a 更新为 1111,T_a 更新为 11.当结点 a 的 END 事件被触发时,根据 C_a,生成集族 S_{C_a} ={0001,0010,0100,1000,…,1111},再根据索引 QIndex 查找 S 中每个元素编码值所匹配的查询组成的集合 Q'_a ={Q_1,Q_2},而 T_a=11,表示 a 结点的后代中已经有满足查询 Q_1 和 Q_2 的 SLCA,故结点 a 不是任何查询的 SLCA 结点.

⟨*a*⟩
⟨*b*⟩blue moon⟨/*b*⟩
⟨*c*⟩bus map⟨/*c*⟩
⟨/*a*⟩

Fig.8 An example of MULSK

图 8 一个 MULSK 的例子

下面分析算法 3 的时间复杂度.设结点内容最大长度为 L_n,关键字最大长度为 L_k,查询的关键字个数最大数为 L_q,结点内容包含关键字最大个数为 $L_{\#k}$,变量 M_1,M 和 $L_{\max}$ 与单查询中描述一致,其余变量描述已在上述分析中给出.处理每个结点 n 的 BEGIN 事件的复杂度均为 $O(1)$.当结点 n 的 TEXT 事件被触发时,利用 KIndex 索引,可以在 $O(L_nL_k)$时间内求得结点所包含的关键字,处理 TEXT 事件的时间复杂度为 $O(L_nL_k)$.当结点 n 的 END 事件被触发时,S_{C_n} 大小至多为 $2^{L_{\#k}}$,故查找集族内每个元素的编码值在索引 QIndex 里对应的查询所需要的时间复杂度为 $O(2^{L_{\#k}})$,之后对查询结果更新的操作与算法 2 一致,最坏情况下,每个查询的查询结果都要更新,故最坏情况下,处理 END 事件的时间复杂度为 $\max\{O(2^{L_{\#k}}),O(|S|\times M^{L_q}\times L_{\max}\times L_q^2)\}$.故算法 3 的时间复杂度为 $\max\{O(L_nL_k),O(2^{L_{\#k}}),O(|S|\times M^{L_q}\times L_{\max}\times L_q^2)\}$.一般地,由于每个结点的内容长度有限,故 $L_{\#k}$ 很小,通常不大于 10;而用户提交的关键字个数较少,L_q 通常在 3~7 之间.

5 实 验

本节给出实验结果和对我们算法的分析.

5.1 实验环境

实验的硬件环境是:PC 机,Pentium Processor 3.20 GHz,512M 内存.操作系统为 Microsoft Windows XP,算法用 VC++ 6.0 实现,并且使用了 SGI 的 STL 库.我们实现了本文中提出的所有算法,我们采取从磁盘上读取 XML

文档,对其进行一次扫描来模拟 XML 数据流.

为有效地测试算法性能,我们采用了 XMark 测试集和 DBLP 测试集.XMark 数据包含了递归等复杂的结构,用来测试算法对于多种查询的效率;DBLP 是真实数据,用来测试算法在真实数据上的执行效率.我们使用 XMark benchmark 数据生成器生成参数从 0.1~1.0 对应的大小不同的文件,其基本信息见表 1.为了有效地测试多种形式查询的处理效率,我们为每个不同 N 值设计了关键字出现较频繁的查询集(Q_1~Q_6)和出现较稀疏的查询集(Q_7~Q_{12}),查询集合见表 2.

Table 1 Basic information of testing data

表 1 实验数据的基本信息

Document	Size of document (M)	No. of nodes	No. of leaves	Depth
Xmark10	11	167 865	122 026	12
Xmark20	22	336 244	244 275	12
Xmark30	34	501 498	364 481	12
Xmark40	45	667 243	484 978	12
Xmark500	56	832 911	605 546	12
Xmark60	68	1 003 441	729 179	12
Xmark70	79	1 172 640	852 384	12
Xmark80	90	1 337 383	972 501	12
Xmark90	102	1 504 685	1 094 333	12
Xmark100	113	1 666 315	1 211 774	12

Table 2 Basic information of queries

表 2 实验所用的查询

Document	Q_{ID}	Query
Xmark	Q_0	{United States}
dblp	Q_1	{IBM,Institute}
dblp	Q_2	{IBM,IWBS,Knowledge}
dblp	Q_3	{IBM,Germany,IWBS,Knowledge}
dblp	Q_4	{IBM,Germany,Center,IWBS,Knowledge}
dblp	Q_5	{IBM,Germany,Center,IWBS,Institute,Knowledge}
dblp	Q_6	{IBM,Germany,Center,IWBS,Institute,Knowledge,Science}
dblp	Q_7	{Gehlen,Reyle}
dblp	Q_8	{Gehlen,-Lattice,Reyle}
dblp	Q_9	{Gehlen,-Lattice,Pletat,Reyle}
dblp	Q_{10}	{Gehlen,-Lattice,Pletat,Reyle,Pnbbenow}
dblp	Q_{11}	{Gehlen,SortLattice,Pletat,Reyle,Pnbbenow,Nayeri}
dblp	Q_{12}	{Gehlen,SortLattice,Pletat,Reyle,Pnbbenow,Nayeri,Czymmeck}

5.2 单查询对比实验

由于本文是第一篇讨论基于松弛 skyline 的 XML 数据流上 Top-*K* 关键字查询的论文,我们仅和用同样方法扫描一次 XML 文档但不进行任何操作的时间进行比较.在此实验中,设 K=5,查询是 Q_0,比较结果见表 3.由表 3 可看出,随文档增大,查询执行时间增加随扫描文档时间线性增长,其查询执行时间增加的速度大约是扫描文档时间增加的速度的 5.5 倍.从中可以看出,相对于扫描数据流,本算法的多余代价较小.

Table 3 Comparison between run time and scanning time

表 3 查询执行时间和扫描时间对比

Document size (M)	Scanning time (s)	Execution time (s)
11	1.248	6.975
22	2.470	14.00
34	3.690	20.81
45	4.923	27.71
56	6.474	34.41
68	7.698	41.70
79	8.779	48.84
90	10.51	55.53
102	11.24	63.23
113	12.50	69.71

5.3 单查询可扩展性实验

为了测试本文提出算法的可扩展性,我们生成具有不同参数的 XMark 文档(从 0.1 变化到 1.0).我们也将查询结果的数量 K 从 6 变化到 18.实验结果如图 9 所示.从实验结果中我们可以看出,程序运行时间和数据流中元素个数大致呈线性关系.另一个观察结果是,程序运行时间基本不受 K 值的影响.这和第 3.3 节的时间复杂度分析是一致的.

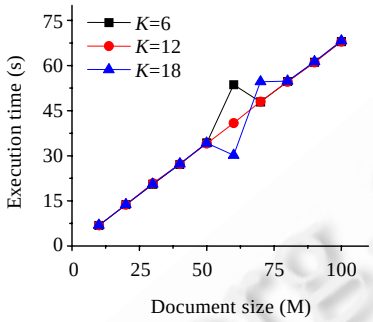

Fig.9 Execution time vs. document size

图 9 查询时间 vs.文档大小

5.4 单查询修改参数

根据我们的分析,查询效率受关键字个数 N 和需要返回结果个数 K 的影响,因此我们对这两个参数取不同值的算法执行效率进行了测试.实验结果如图 10、图 11 所示.由图 10 可知,无论是查询出现频率低的关键字集合还是查询出现频率高的关键字集合,算法运行时间基本不随 N 变化.这是因为我们在第 3.3 节所分析的是最坏情况下的时间复杂度,在大多数情况下,SLCA 结点的后代中只有少数叶子结点满足包含关键字.由于笛卡尔积的基数总是很小,因此生成笛卡尔积的代价也几乎不受 N 的大小的影响.从图 11 可知,无论是查询出现频率低的关键字集合还是查询出现频率高的关键字集合,算法运行时间基本不随 K 变化.这也是因为我们在第 3.3 节所分析的是最坏情况下的时间复杂度,在大多数情况下,新生成的 SLCA 查询结果并不是全部需要被插入 L 中,而只是一小部分.

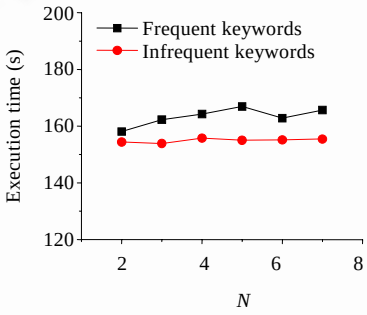

Fig.10 Execution time vs. N

图 10 查询时间 vs. N

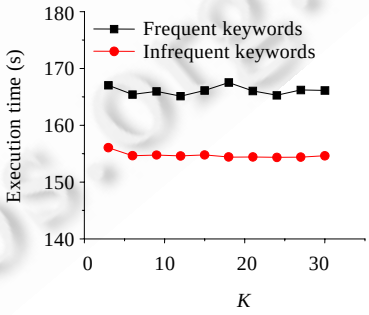

Fig.11 Execution time vs. K

图 11 查询时间 vs. K

5.5 多查询对比实验

根据我们的分析,多查询效率受关键字总个数(#keywords)和查询总个数(#queries)的影响,因此我们对这两个参数取不同值,通过随机产生的若干关键字和查询,对 MULSK 算法执行效率进行了测试.此外,为了体现 MULSK 的优势,我们将其与采取串行执行多个 LSK 来处理 MLSK 的策略(简称为 SQ)相比较.

在图 12 的实验中,固定查询个数为 64.由实验结果可看出:随着关键字总个数增大,采用 KIndex 索引的 MULSK 算法执行时间基本不随关键字个数的增大发生变化;而串行处理的 SQ 算法执行时间均随关键字个数的增大而增大,中间存在的一些波动与随机选择关键字出现的频率有关.在图 13 的实验中,我们固定关键字个数为 32.由实验结果可看出,随着查询个数增大,采用 QIndex 索引的 MULSK 算法的执行时间基本不随查询个数的增大发生变化,而串行处理的 SQ 算法的执行时间随关键字个数的增大呈线性增加.

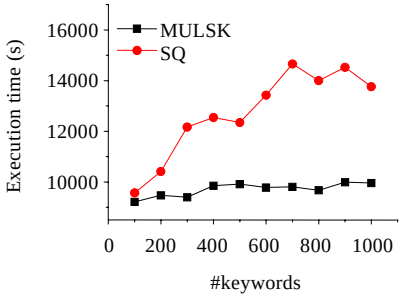

Fig.12 #keywords

图 12 关键字总数

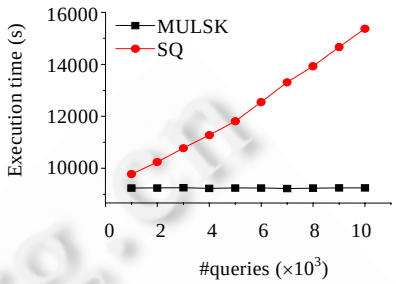

Fig.13 #queries

图 13 查询个数

6 相关工作

目前,对 Skyline 查询处理已有了很多的研究.研究者考虑了在数据不具有索引结构[9,10]和具有索引结构支持[11−14]情况下对数值型数据的 skyline 查询的评估.最近,更多的工作集中于当维数数量增大导致 skyline 点过多时对 skyline 查询进行变体处理,从而向用户提供最有意义的 skyline 点:文献[15]根据元组在不同子空间里成为 skyline 点的频繁度,提出了查找前 k 个最频繁的 skyline 点的方法.文献[16]提出了 k 维支配的 skyline 点,即若 p 是这样的点,不存在其他结点 q,满足存在某 k 维的子空间 R,q 在 R 上支配结点 p.文献[17]提出向用户返回前 k 个最有代表性的 skyline 点的 Top-K skyline 查询,使得至少被这 k 个 skyline 点中某点支配的结点个数最大化.这些研究主要集中于对 skyline 操作的研究.文献[18]提出了在不确定数据上如何计算 skyline 概率的方法来寻找前 K 个概率最高的点,该研究集中于对不确定数据上 skyline 概率的计算.上述工作均没有考虑 XML 数据流关键字查询的特点.

由于 XML 数据关键字查询的广泛应用引起了研究人员的重视,已经有很多 XML 数据上的关键字查询技术[1,3,4,19−25]提出,确定和关键字查询相关的结果是研究的重点之一.其中,XKeyword[1]和 Précis[19]允许一个系统管理者和用户基于模式图定义相关性.文献[20]根据 XML 数据的特性和输入关键字自动地推断出与查询相关的结果;XSearch[3]和 XRank[4]扩展了信息检索技术来衡量结果与查询的相关性大小.对于 XML 数据上的 Top-K 关键字查询,大都利用计算结果与查询相关性的打分函数对结果排序.其中,文献[3]结合 tfidf 的排序技术及树的大小和结点关系信息来对查询结果进行排序;文献[4]通过一般化 Google 的 Page-Rank 算法对 XML 文档中的元素进行排序,并通过将元素的排序技术和临近字符串查询结合起来对结果与查询的相关性排序.XML 关键字查询的相关工作还包括:文献[20,24]考虑了查询的关键字之间存在 AND 和 OR 两种关系的情况,文献[22]研究了利用搜索实体化视图来优化 XML 关键字搜索技术.

目前存在的排序方法都基于预先定义的打分函数,根据此函数计算查询结果与关键字查询的相关性.当用户对查询的需求不一致从而导致查询结果的相关性无法比较时,很难建立合理的打分函数来表示结果与查询的相关性.针对打分函数的不足,本文采用 skyline 技术帮助用户做智能的决策,我们发现,skyline 是处理不同结果的相关性不可比较这一问题的有效工具.

本文的前期工作[26]发表在 DASFAA 2009 上,提出了在 XML 数据流上的 LSK 查询,并给出了 LSK 查询算法.本文在其基础上进行了扩展,提出多查询下的 LSK 查询,并给出了多 LSK 查询处理算法.

7 总 结

随着 XML 数据流的越来越广泛的应用,Top-*K* 关键字查询已成为 XML 数据流上重要的查询.考虑到相同的关键字查询对于不同用户存在不同的查询需求,本文提出了基于松弛 skyline 的 XML 数据流上的关键字查询(简称为 LSK 查询)和有效地处理这类查询的算法.并将 LSK 查询扩展,提出了多查询环境下的 LSK 查询(简称为 MLSK 查询),设计了 MLSK 查询处理算法.经实验验证,所提出的查询处理算法的效率几乎不受关键字个数、查询结果数量、查询数量等参数的影响,运行时间和文档大小大致呈线性关系.分析和实验结果证明,对于 LSK 查询和 MLSK 查询,本文所提出的技术都能够快速有效地获得查询结果,并且具有良好的可扩展性.我们未来的研究工作是处理基于滑动窗口的 XML 数据流上的 Top-*K* 关键字查询.

References:

[1] Hristidis V, Papakonstantinou Y, Balmin A. Keyword proximity search on XML graphs. In: Dayal U, *et al*., eds. Proc. of the 19th Int'l Conf. on Data Engineering. Bangalore: IEEE Computer Society, 2003. 367−378.

[2] Barg M, Wong RK. Structural proximity searching for large collections of semi-structured data. In: Proc. of the 2001 ACM CIKM Int'l Conf. on Information and Knowledge Management. Atlanta: ACM Press, 2001. 175−182. [doi: 10.1145/502585.502615]

[3] Cohen S, Mamou J, Kanza Y, Sagiv Y. XSEarch: A semantic search engine for XML. In: Freytag J, Lockemann P, *et al*., eds. Proc. of 29th Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 45−56.

[4] Guo L, Shao F, Botev C, Shanmugasundaram J. XRANK: Ranked keyword search over XML documents. In: Halevy AY, Ives Z, Doan AH, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 16−27. [doi: 10.1145/872757.872762]

[5] Borzsonyi, Kossmann D, Stocker K. The skyline operator. In: Proc. of the 17th Int'l Conf. on Data Engineering. Heidelberg: IEEE Computer Society, 2001. 421−430. [doi: 10.1109/ICDE.2001.914855]

[6] Tatarinov I, Viglas S, Beyer K, Shanmugasundaram J, Shekita E, Zhang C. Storing and querying ordered XML using a relational database system. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 204−215. [doi: 10.1145/564691.564715]

[7] Fredkin E. Trie memory. Communication of the ACM, 1960,3(9):490−499. [doi: 10.1145/367390.367400]

[8] Aoe J. An efficient digital search algorithm by using a double-array structure. IEEE Trans. on Software Engineering, 1989,15(9): 1066−1077. [doi: 10.1109/32.31365]

[9] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with presorting. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Bangalore: IEEE Computer Society, 2003. 717−816. [doi: 10.1109/ICDE.2003.1260846]

[10] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets. In: Böhm K, Jensen CS, *et al*., eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim: ACM Press, 2005. 229−240.

[11] Tan KL, Eng PK, Ooi BC. Efficient progressive skyline computation. In: Apers PMG, Atzeni P, *et al*., eds. Proc. of 27th Int'l Conf. on Very Large Data Bases. Roma: Morgan Kaufmann Publishers, 2001. 301−310.

[12] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 275−286.

[13] Papadias D, Tao Y, Fu G, Seeger B. An optimal and progressive algorithm for skyline queries. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 467−478. [doi: 10.1145/872757.872814]

[14] Lee K, Zheng B, Li H, Lee WC. Approaching the skyline in z order. In: Koch C, Gehrke J, *et al*., eds. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. ACM Press, 2007. 279−290.

[15] Chan CY, Jagadish HV, Tan KL, Tung AKH, Zhang Z. On high dimensional skylines. In: Ioannidis Y, Scholl M, *et al*., eds. Proc. of 10th Int'l Conf. on Extending Database Technology. Munich: Springer-Verlag, 2006. 478−495. [doi: 10.1007/11687238_30]

[16] Chan CY, Jagadish HV, Tan KL, Tung AKH, Zhang Z. Finding *k*-dominant skylines in high dimensional space. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Chicago: ACM Press, 2006. 503−514. [doi: 10.1145/1142473.1142530]

[17] Lin X, Yuan Y, Zhang Q, Zhang Y. Selecting stars: The *k* most representative skyline operator. In: Proc. of the 23rd Int'l Conf. on Data Engineering. Istanbul: IEEE Press, 2007. 86−95. [doi: 10.1109/ICDE.2007.367854]

[18] Zhang Y, Zhang WJ, Lin XM, Jiang B, Pei J. Ranking uncertain sky: The probabilistic Top-*k* skyline operator. Information Systems, 2011,36(5):898−915. [doi: 10.1016/j.is.2011.03.008]

[19] Koutrika G, Simitsis A, Ioannidis YE. Précis: The essence of a query answer. In: Liu L, Reuter A, *et al*., eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Atlanta: IEEE Computer Society, 2006. 69−78. [doi: 10.1109/ICDE.2006.114]

[20] Liu Z, Chen Y. Identifying meaningful return information for XML keyword search. In: Chan CY, *et al*., eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Beijing: ACM Press, 2007. 329−340. [doi: 10.1145/1247480.1247518]

[21] Li G, Feng J, Wang J, Zhou L. Effective keyword search for valuable LCAs over XML documents. In: Silva MJ, *et al*., eds. Proc. of the 16th ACM Conf. on Information and Knowledge Management. Lisbon: ACM Press, 2007. 31−40. [doi: 10.1145/1321440.1321447]

[22] Li G, Ooi BC, Feng J, Wang J, Zhou L. Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: Wang JTL, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Vancouver: ACM Press, 2008. 903−914. [doi: 10.1145/1376616.1376706]

[23] Li YY, Yu C, Jagadish HV. Enabling Schema-Free XQuery with meaningful query focus. The VLDB Journal, 2008,17(1):355−377. [doi: 10.1007/s00778-006-0003-4]

[24] Sun C, Chan CY, Goenka A. Multiway SLCA-based keyword search in XML data.In: Williamson CL, *et al*., eds. Proc. of the 16th Int'l Conf. on World Wide Web. Banff: ACM Press, 2007. 1043−1052. [doi: 10.1145/1242572.1242713]

[25] Xu Y, Papakonstantinou Y. Effcient keyword search for smallest LCAs in XML databases. In: Özcan F, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM Press, 2005. 527−538.

[26] Li LL, Wang HZ, Li JZ, Gao H. Efficient algorithms for skyline Top-*k* keyword queries on XML streams. In: Zhou XF, Yokota H, Deng K, Liu Q, eds. Proc. of 14th Int'l Conf. LNCS 5463, Brisbane: Springer-Verlag, 2009. 283−287. [doi: 10.1007/978-3-642-00887-0_24]

黎玲利(1986－),女,四川广元人,硕士,主要研究领域为 XML 关键字搜索,数据质量.

高宏(1966－),女,博士,教授,博士生导师,主要研究领域为数据管理,传感器网络,图数据库.

王宏志(1978－),男,博士,副教授,主要研究领域为 XML 数据管理,图数据库,数据质量,信息集成.

李建中(1950－),男,博士,教授,博士生导师,主要研究领域为数据库,并行计算,无线传感器网络.