ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.2032−2042
DOI: 10.3724/SP.J.1001.2008.02032

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

XML数据流上的高效聚集算法*

王宏志+, 李建中, 骆吉洲

(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 Aggregation Algorithms on XML Stream

WANG Hong-Zhi+, LI Jian-Zhong, LUO Ji-Zhou

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

+ Corresponding author: E-mail: wangzh@hit.edu.cn

Wang HZ, Li JZ, Luo JZ. Efficient aggregation algorithms on XML stream. *Journal of Software*, 2008,19(8): 2032−2042. http://www.jos.org.cn/1000-9825/19/2032.htm

Abstract: Each element and value in a XML stream can be accessed only one time. In this paper, efficient algorithms are proposed for processing aggregation on XML streams. These algorithms efficiently support the processing of the aggregation queries with complex structures and the aggregation queries on XML stream with recursion structures. Theoretical analysis and experimental results show that the proposed algorithms are able to process aggregation queries effectively and efficiently on XML stream with high scalability.

Key words: XML stream; aggregation; algorithm

摘 要: XML 数据流的特点是所有元素和值仅允许扫描 1 次.针对 XML 数据流上的聚集问题,提出了高效的 XML 数据流聚集算法.这种算法不但能够有效地支持 XML 数据流上具有复杂结构聚集查询的处理,而且能够有效地支持具有递归结构 XML 数据流上的聚集查询处理.理论分析和实验结果表明,算法能够有效地处理 XML 数据流上的聚集查询,并且具有很好的可扩展性.

关键词: XML 数据流;聚集;算法

中图法分类号: TP311 **文献标识码**: A

XML 已经成为因特网上信息表达和交换的重要标准,得到了广泛的应用.XML 数据的查询处理已经引起了人们的极大关注,取得了很多研究成果.与此同时,人们发现,在很多应用中,如网络信息订阅与发布、电子邮件监测等,出现了一类新的数据流,即 XML 数据流.XML 数据流是一种按照时间顺序实时到达的无限流数据.由于 XML 数据流仅允许 1 次并按照其数据到达的时间顺序进行存取(不允许多次和随机扫描),XML 数据流的查询处理成为一个挑战性的研究问题.最近几年,人们开展了一些相关研究,提出了一些 XML 数据流的查询处理方法[1−8].然而,目前的研究忽略了一类重要的查询,即 XML 数据流上的聚集查询.下面,我们以电子邮件监测为例来说明 XML 数据流上的聚集查询的基本思想.

电子邮件可以用 XML 来表示,其简化的模式如图 1 所示,其中 context 是邮件的内容,sender 是发件

* Supported by the National Natural Science Foundation of China under Grant Nos.60773068, 60773063, 60533110 (国家自然科学基金); the National Basic Research Program of China under Grant No.2006CB303000 (国家重点基础研究计划(973))

Received 2007-09-25; Accepted 2008-04-15

人,receiver 是收件人.对电子邮件监视系统来说,电子邮件是源源不断地发出的.因此,电子邮件监视系统面对的是一个 XML 数据流,其中每一个邮件是一个 XML 文档片断.潜在垃圾邮件的判定是电子邮件监视系统的一个重要功能.一种潜在垃圾邮件的判定方法是:如果系统发现同样内容的邮件被发送到很多用户,则判定该邮件为潜在垃圾邮件.这种判定需要对电子邮件流进行如下的聚集查询:“按照所接收的邮件之内容对邮件接收者分类,计算每个类中接收者的数量.”如果每个类中接收者数量很大,则表示该类对应的邮件内容被发送到很多用户,相应的邮件可以被判定为潜在垃圾邮件.上述查询可以表示为电子邮件流上的聚集查询: //mail/sum(/receiver) with count group by context.在此查询中,聚集函数 count 表示首先计算每个 mail 元素中包含 receiver 元素的个数,聚集函数 sum 表示计算每组中的 mail 元素包含的 message 元素个数的和.group by context 表示 mail 元素需要按照 context 元素进行分组.

```
〈!ELEMENT mail (context, sender, receiver*)〉
〈!ELEMENT context #PCDATA〉
〈!ELEMENT sender #PCDATA〉
〈!ELEMENT receiver #PCDATA〉
```

Fig.1 DTD of E-mail

图 1 电子邮件 DTD

XML 数据流上的聚集查询应用十分广泛.下面是另一个 XML 数据流上聚集查询的应用实例.在基于 XML 的信息定阅(subscribe)和发布(publish)系统中,XML 数据流是关键,用户常常提出类似于“每天发送的内容与‘database’相关的信息有多少?”、“每个类型的信息发送各有多少?”等查询.这些查询可以表示成聚集查询的形式.

从上面的讨论不难看出,XML 数据流上的聚集查询是一类应用广泛的重要查询.遗憾的是,虽然目前已经提出一些 XML 数据(不是 XML 数据流)上的聚集查询处理算法[9,10],却一直缺少有关 XML 数据流聚集查询处理的研究结果.由于 XML 数据流仅允许 1 次扫描,XML 数据上的聚集查询处理算法无法用来处理 XML 数据流上的聚集查询,因为这些算法需要多次扫描完整的 XML 数据.

为了有效地处理 XML 数据流上的聚集查询,本文提出了一种高效的 XML 数据流上聚集算法.该算法具有 3 个特点:第一,算法利用栈结构尽早处理聚集的中间结果,减少了中间结果的计算复杂性,提高聚集计算的效率;第二,针对具有递归结构的 XML 数据流,算法使用栈结构实现了利用后代结点的聚集结果计算祖先结点的聚集,能够有效地处理具有递归结构的 XML 数据流上的聚集查询;第三,算法采用位图、bloom filter、约束矩阵等技术,解决了 XML 数据流上具有复杂结构聚集查询的有效处理问题.分析和实验结果表明,本文提出的算法具有很高的效率和可扩展性,运行时间是数据量和查询中结点数的线性函数.本文的主要贡献如下:(1) 本文提出了 XML 数据流上聚集的问题,并给出了精确定义.(2) 本文提出了高效的 XML 数据流上聚集查询算法,既支持复杂的 XPath 表达式,又适用于多种类型的 XML 聚集,并且适用于基于 SAX 的 XML 文档上的聚集查询处理.

本文第 1 节给出问题的定义和相关概念.第 2 节介绍 XML 数据流上的聚集算法及其性能分析.第 3 节给出算法的实验结果与分析.第 4 节对本文的内容进行总结.

目前 XML 数据流上查询处理方法主要分为 3 类.第 1 类方法是基于自动机的方法[1−8].这类方法首先为给定的 XPath 或 XQuery 查询建立一个自动机,然后把 XML 数据流作为自动机的输入,实现查询的处理.这类方法没有考虑聚集查询,采用了模式匹配技术实现查询的处理,得到的是未经分组的数据片段,无法处理聚集查询.第 2 类方法是面向优化的方法[11,12],重点是通过改写给定的 XQuery 查询,实现查询处理优化.第 3 类方法[13−15]是当查询数量很多时,对多个查询建立索引,实现多查询的有效处理.后两类方法都是对第 1 类方法的扩展,本质上仍然基于模式匹配技术,不能处理聚集查询.

关于普通数据流上的聚集查询处理已经有一些技术提出,文献[16]是对这些技术的综述.但是,这些技术不能用来处理 XML 数据流上的聚集查询.这是由于 XML 数据流上的聚集操作需要考虑对满足特定路径约束元素的聚集,传统数据流仅仅考虑数据的聚集而无法对数据的路径约束进行判断.

最近几年,人们研究了 XML 数据(不是 XML 数据流)上的聚集处理算法[9,10].文献[9]中提出的聚集算法首先需要对 XML 文档进行预处理,建立支持聚集的数据结构,然后进行查询处理,要求多次扫描 XML 文档,与 XML 数据流仅允许 1 次扫描的特点相矛盾,不能用来有效地处理 XML 数据流上的聚集查询.文献[10]中提出的方法用来处理带有约束条件的聚集查询,需要首先在 XML 文档上进行 XPath 查询处理,获得满足约束条件的路径,

然后在这些路径上进行聚集查询,至少需要扫描 XML 文档两次,因此这种方法不能通过 1 次扫描来处理 XML 数据上的聚集.

1 查询的定义

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.如果一个元素包含其他元素,则称这个元素为复合元素.对于元素 e,从根节点到 e 的路径上所有元素构成的序列称为 e 的进入路径.XML 数据流可以看作是源源不断到来的 XML 片段,在 XML 数据流上的处理仅仅允许对每个元素扫描 1 次.

XPath[17]表达式用来描述 XML 文档中一类特定的结点,XPath 描述可以分为一系列步骤,每个步骤均用标签描述,步骤间的关系用轴来描述,在 XPath 标准中定义了 13 种轴,本文考虑最常见的两种轴:child 和 descendent.其中,child 表示在 XML 树结构中两个结点之间存在父子关系(简称 PC 关系),用"/"来表示;descendent 表示两个结点间存在祖先后代关系(简称 AD 关系),用"//"来表示.

根据文献[9,10,18]中的定义,XML 上的聚集查询可以表示为如下形式:$path_a$ a/agg_fun_1($path_b$ b) with agg_fun_2 group by $path_{c_1}$ c_1,$path_{c_2}$ c_2,…,$path_{c_k}$ c_k using $group\_fun$.其中,a 是聚集操作的对象,称作聚集对象;b 是聚集对象 a 的一个后代,是聚集操作的具体属性,称作度量属性;c 是用来分组的元素,称为维属性;$path_a$,$path_b$ 和 $path_{c_1}$,…,$path_{c_k}$ 是 XPath 表达式,分别用来修饰元素 a,b 和 c_1,…,c_k;agg_fun_2 是用来对每个聚集对象的度量属性进行聚集操作的函数,称为内聚集函数;agg_fun_1 是用来对每个分组中聚集对象的内聚集函数结果进行聚集的函数,称为外聚集函数;$group\_fun$ 是用来定义分组方法的函数,称为分组函数.

XML 文档上的聚集查询与多维数据聚集查询有所不同.用于分组的元素既可以是简单元素也可以是复合元素,而多维数据上聚集查询的分组表示方法不能表达这种复杂的分组,故引入分组函数在分组过程中定义两个集合或复合元素相等.XML 文档上聚集对象的度量属性可以被多个数据元素匹配,而多维数据聚集查询的度量属性只能被一条记录匹配 1 次.内聚集函数恰当地表达了这个区别.

聚集查询的结果是一系列元组,每个元组对应一个分组,元组中包含维属性的值和外聚集函数的聚集结果.一个聚集查询的实例为 $a/b[k]//c[d[f][g]]/e/sum(//h)$ with count group by i/j.其含义是,对所有满足 XPath 表达式 $a/b[k]//c[d[f][g]]/e$ 的元素 e,按照 e 后代中满足约束 i/j 的 j 元素进行分组,计算每个分组中 e 的后代中 h 元素的个数.在这个查询中,e 是聚集对象,h 是度量属性,j 是维属性,$count$ 是内聚集函数,sum 是外聚集函数.

XML 上的聚集查询可以用树结构来表达,查询中的元素表示为树结构中的结点,查询中相邻元素之间的关系表示为树结构中的边.在树结构中,$path_a$,$path_b$,$path_{c_1}$,…,$path_{c_k}$ 上约束中的结点称为约束结点,$path_a$ 中除去约束结点的其他结点称为主干结点,$path_b$,$path_{c_1}$,…,$path_{c_k}$ 上除去约束结点的其他结点称为中间结点.对每个结点,其儿子结点中的约束结点称为该结点的约束结点.查询中的所有结点统称为查询结点.

本文考虑的是 XML 数据流上的聚集查询,其关键问题是如何在一段时间内对 XML 数据流上多个 XML 片段中指定的元素有效地进行分组并计算聚集结果.此时,这些 XML 片段可以看作一个具有虚拟根的 XML 文档,但每个结点只能扫描 1 次.

2 XML数据流上聚集查询处理算法

XML 数据流上的聚集查询处理算法包括两部分:一部分是查询对象的获取,得到相应的维属性和度量属性;另一部分将获取到的查询对象根据维属性聚集到相应的分组中.这两部分在XML数据流处理的过程中交替进行.下面,首先给出算法框架,然后分别给出这两部分的处理方法,最后分析算法的时间复杂性并对算法进行扩展.

2.1 算法框架

为简单起见,我们仅考虑不包含复合约束条件的查询,带有复杂约束条件聚集查询的处理在第 2.5 节讨论.

聚集查询的处理是两个部分的复合,一部分是聚集对象的获取,另一部分是聚集的处理.前者在数据流中寻

找进入路径匹配每个查询结点 n 的元素 e,n 称为 e 的匹配结点.当访问到元素的结束标签时,根据 XML 数据流的性质,该元素的所有儿子和后代均已访问,则可以判定该元素是否满足查询结点对应的约束条件.聚集对象获取技术将在第 2.2 节中描述.聚集处理部分利用获取的聚集对象进行计算,实现聚集对象内部度量属性的聚集和聚集对象的分组与聚集.聚集处理技术将在第 2.3 节中描述.下面是基本数据结构和算法框架.

算法用一个栈 S 来保存 XML 数据流上正被处理的数据片段中根结点到当前扫描结点的路径,用一个同构于查询树的树结构 $T=(V,E)$来存储当前查询的临时结果.对每个查询结点 $v\in V$,用一个栈 $v.S$ 保存数据片断中当前访问路径上与查询结点 v 匹配的所有元素.

在扫描数据流的过程中,由于查询中存在"//"关系,XML 数据流上同一个结点可能与查询中多个路径中的结点匹配.但由于XML数据片断中每个元素只能被扫描1次,因此,当前被扫描的元素能否匹配查询中的多个路径必须同时得到判定.为此,用集合 *span_list* 存储当前需要匹配的查询结点.在接收一个数据片段之前,用查询的根结点初始化 *span_list*.在扫描过程中,当元素 e 匹配查询结点 n 时,如果 n 与其父亲的关系是 PC 或者 n 是根结点且 n 的轴是"/",则元素 e 的儿子或后代可以匹配 n 在查询中的儿子.此时,将 n 的儿子加入 *span_list*,同时将 n 从 *span_list* 中移除.如果 n 与其父亲的关系是 AD 或者 n 是根结点且 n 的轴是"//",由于递归的存在,e 的后代既可能匹配 n 的儿子,也可能匹配 n 本身.此时,将 n 的儿子加入 *span_list*,但无须将 n 从 *span_list* 中移除.

算法 1 是 XML 数据流上聚集查询处理算法的基本框架,其主要步骤是在扫描到数据流中每个元素的开始标签和结束标签时分别执行相应的动作.

算法 1.

```
processBegin(Element e)
1   for span_list 中的每个结点 n do
2       if e.tag=n.tag then
3           e.isMatched=true
4           n.S.push(e)
5               e.QNode=n
6               e.parent=n.parent.S.top()
7               if e.parent.child=NULL then
8                   e.parent.child=e
9           span_list.add(n.children)
10      if n.axis="/" then
11          span_list.remove(n)
12          S.push(e)
processEnd(Element e)
1   if e.isMatched=true then
2     e=S.pop()
3       e=e.QNode.S.pop()
4   switch(e 对应查询结点的类型)
5   case 聚集对象:processAggobject(e)
6   case 中间结点:processIntermeidate(e)
7   case 主干结点:processTrunk(e)
8   case 约束结点:processBranch(e)
9   case 维属性:processMeasure(e)
10  case 度量属性:processDinmension(e)
11  if e.QNode.S 非空 then  //恢复 span_list
12    for e.QNode.children 中的每个元素 q_d
13        if q_d.axis="/" OR q_d.S 非空 then
14            span_list.remove(q_d)
15    tn=S.top().QNode
16    for tn.children 中的每个元素 q_d do
17        span_list.insert(q_d)
```

结点 e 的开始标签在 *processBegin*()中处理.如果 e 的标签和 *span_list* 中的某个查询结点 n 相同,则表明 e 的进入路径能够匹配 n 的进入路径.此时,将 e 压入 S 中(第 12 步).同时,给 e 赋予一个指针 $e.QNode$ 指向 n(第 5 步),其主要用途是在 e 出栈时辨别 e 的类型.此外,还要将 e 压入 $n.S$(第 4 步),并将 e 与匹配 n 父亲的最近结点相连接(第 6~8 步).若 e 的标签与 *span_list* 中所有结点的标签都不相同,则表明 e 与任何查询结点不匹配.此时,不执行任何操作.

结点 e 的结束标签在 *processEnd*()中处理.若 e 在 S 中,则 e 分别从 S 和 $e.QNode.S$ 中弹出(第 2~3 步).同时,根据 $e.QNode$ 的类型执行相应操作,从而完成聚集(第 4~16 步).此外,还需要将 *span_list* 恢复到处理 e 之前的状态(第 17~23 步).若 e 不在 S 中,则 e 不匹配任何查询结点,不执行任何操作.

函数 *processAggobject*(e),*processIntermeidate*(e),*processTrunk*(e),*processBranch*(e),*processMeasure*(e) 和 *processDinmension*(e)是处理不同类型结点的终止标签时采取的不同动作,其实现将在第 2.3 节给出.根据 XML 数据流中元素到来的顺序,聚集对象识别和聚集执行这两个部分交替进行.

2.2 聚集对象获取的优化处理方法

算法 1 在 XML 数据流中获取查询对象及其对应的维属性和度量属性,它在得到每个结点的全部信息时对相应的约束进行判断,这可能会存储过多的中间结点和执行过多的判定.为加速处理过程和节省内存,我们给出聚集对象获取的优化处理方法.其核心思想是,对每个元素,尽早地判断该元素满足约束的情况,减少中间结果数量;对已经满足约束的元素,在处理其后的某些元素时不再对它们是否满足约束进行判定.

由于 XML 数据流只允许 1 次扫描,对于受结构约束的结点,从当前状态不能判断该元素是否满足所有的查询条件.为解决这个问题,对 $v.S$ 中的每个结点 v,用一个位图 $bitmap_v$ 来表示结构约束的满足情况,其中每个位表示 v 上的一个特定约束是否被满足.对不同类型的结点,其位图有不同结构和约束满足判定方法.假设元素 v 匹配查询结点 q_v,位图的结构和相应的判定方法如下:

(1) 如果 q_v 是约束结点或中间结点,q_v 的每个约束对应 $bitmap_v$ 的一个位,$bitmap_v$ 的长度等于 q_v 儿子中约束结点的个数.当 v 结点的开始标签到达时,$bitmap_v$ 的所有位被初始化为 0,这表示 v 上的所有结构约束均未被满足.在处理过程中,如果某结构约束被满足,则 $bitmap_v$ 中相应的位将被置 1.如果 $bitmap_v$ 的所有位置为 1,则表明 v 满足所有的约束条件.

(2) 如果 q_v 是主干结点或者聚集对象,任意主干结点的一个结构约束均对应 $bitmap_v$ 的一个位,$bitmap_v$ 的长度等于查询中所有主干结点上结构约束的个数.当 v 结点的开始标签到达时,$bitmap_v$ 中对应主干结点中已被满足结构约束的位被置为 1,其他位被置为 0.处理过程中,如果 v 上的某个结构约束被满足,则 $bitmap_v$ 中相应的位将被置 1;同时,v 结点所在的路径上的其他主干结点的位图中相应位也将被置为 1.如果 $bitmap_v$ 中对应 q_v 上结构约束的位为 1,则表明查询中 q_v 结点上的结构约束被满足.如果 $bitmap_v$ 的所有位置为 1,则表明 v 结点所在的路径满足查询中所有的结构约束.

为了加速判定结点 v 上的所有结构约束是否被满足,我们将判定条件表示成特殊的位图存储在匹配 v 的查询结点上.若 q_v 是约束结点或者中间结点,则 q_v 上这个特殊位图称为 $q_v.LMAP$,它是一个位数等于 q_v 儿子中约束结点的个数、所有位都是 1 的位图,$bitmap_v=q_v.LMAP$,则表明 v 满足 q_v 的所有结构约束.如果 q_v 是主干结点或者聚集对象,q_v 上的两个特殊位图分别记作 $q_v.LMAP$ 和 $q_v.FMAP$,前者用来判断 v 是否满足 q_v 结点上的结构约束,后者用来判断 v 所在路径是否满足查询中的所有结构约束.此时,$q_v.LMAP$ 是一个位数等于查询中所有主干结点儿子中约束结点个数的位图,其中对应 q_v 儿子中约束结点的位等于 1,其他位为 0;$bitmap_v\&q_v.LMAP=q_v.LMAP$ 表明 v 满足 q_v 结点上的所有结构约束.$q_v.FMAP$ 是一个位数与 $q_v.LMAP$ 位数相等且所有位都是 1 的位图,$bitmap_v= q_v.FMAP$ 表明 v 所在路径满足查询中所有结构约束.

下面讨论位图的更新.在扫描数据流的过程中,如果元素 e 的结束标签到达且 q_e 是叶子查询结点,则表明栈 $q_e.parent.S$ 中与 $q_e.parent$ 匹配的每个元素 e' 对应的结构约束已得到满足,这些元素的位图需要更新.如果 e' 的位图更新后等于 $q_{e'}.LMAP$,表明 e' 的所有结构约束均已满足,则需要进一步更新 e' 父亲或祖先的位图,该过程持续到遇到根结点或者聚集对象为止.算法 2 描述了位图更新过程.

算法 2. 路径约束条件的处理.

```
processBranch(e)
1 if q_e 是叶子 then
2   updateBitmap(q_e,e)
updateBitmap(QueryNode n, Element e)
1 if n 不是聚集对象或者根结点 then
2   e'=e.parent;
3   bitmap_e'[n]=1
4   if bitmap_e' & q_e'.LMAP=q_e'.LMAP then
5     updateBitmap(n.parent,e')
6       if n.axis="//" then
7         for n.parent.S 中的每一个结点 np do
8           bitmap_np[n]=1
9           if bitmap_np & n.parent.LMAP=n.parent.LMAP then
10            updateBitmap(np)
```

值约束也可以用类似的方法来处理.若查询结点 n 有值约束,则在构建查询树时为 n 分配变量 $n.value\_constraint$ 以保存这个值约束.如果 XML 数据流上的元素 e 匹配查询结点 q_e 且 q_e 有值约束,则当 e 的值 $e.value$ 到达时判断 $e.value$ 是否满足 $q_e.value\_constraint$.如果满足,则置判定变量 $e.satisfy$ 为 true,否则置为 false.当元素 e 的结束标签到达时,根据 $e.statisfy$ 的值执行相应的动作.

2.3 聚集处理

本节讨论聚集的处理方法.聚集过程首先在匹配查询条件的聚集对象上将度量属性按照内聚集函数进行聚集,再根据维属性对聚集对象进行分组并用外聚集函数聚集.在该过程中,可以进行如下优化:

(1) 无须等到聚集对象的所有元素到达后才进行聚集.对于聚集对象 a,当其度量属性元素 e 的结束标签到达时,即可将 e 的值用内聚集函数聚集后保存为 a 的临时结果.这样,度量属性的值就无须存储.

(2) 当聚集对象的结束标签到达后,尽早将其上保存的临时聚集结果用外聚集函数聚集到结果中,从而节省该聚集对象上存储这些临时结果的空间.

(3) 当聚集对象存在递归时,后代聚集对象上的聚集结果是其祖先聚集对象上聚集结果的一部分,这部分聚集结果在后代聚集对象上聚集得到后,无须再在祖先聚集对象上重新计算.这种聚集结果的重用可以提高计算效率.

为了达到这些优化目标,中间结点和主干结点需要相应的数据结构.给聚集对象中每个匹配中间结点的元素 e 赋予变量 $e.v$ 和序列 $e.dimension\_list$,$e.v$ 存储 e 后代中度量属性的临时聚集结果,$e.dimension\_list$ 存储 e 的后代中匹配维属性的元素的值,称为维序列.这是为了尽早计算维属性的值和内聚集函数的结果,并丢弃结束标签已到达的元素.为每个匹配主干结点的元素 e 赋予序列 $e.Agg\_list$ 保存 e 的后代聚集对象.当 e 的结束标签到达时,若 e 满足所有约束,则将 $e.Agg\_list$ 中的元素合并到结果中;否则,将 $e.Agg\_list$ 中的元素拷贝到其父亲的 Agg_list 中.此时,释放 e 结点以节省内存.

基于上述优化策略和数据结构,聚集过程在元素 e 的开始标签到达时为 e 分配相应的数据结构.当 e 的结束标签到达时(此时,e 的所有后代均已处理完毕),根据 q_e 的类型和 e 数据结构中的信息执行相应的动作.如果 q_e 是维属性,则将 e 的值加入 $e.parent$ 的维序列.如果 q_e 是度量属性,则用内聚集函数将 e 的值聚集到 $e.parent.v$ 中.

如果 q_e 是中间结点,则需要根据 q_e 的轴和位图进行相应的处理.如果 $bitmap_e=q_e.LMAP$,表明 e 及其后代的结构约束均已被满足,此时,将 $e.v$ 和 $e.dimension\_list$ 合并到 $e.parent$ 的相应数据结构中.否则,如果 q_e 的轴是“/”,则丢弃 e,因为 e 的所有后代已扫描完毕,e 的条件不可能得到满足;如果 q_e 的轴是“//”,则将 $e.v$ 和 $e.dimension\_list$ 拷贝到其祖先上去,因为尽管 q_e 的约束没有被满足,但(由于递归)e 的祖先有可能满足条件.这个过程见算法 3.

如果 q_e 是聚集对象,则根据 e 是否满足结构约束执行相应的动作.如果 $bitmap_e=q_e.FMAP$,则将 $e.v$,$e.dimension\_list$ 和 $e.Agg\_list$ 中的元素合并到最终结果去.否则,如果 $bitmap_e\&q_e.LMAP=q_e.LMAP$,则将⟨$e.v$,$e.dimension\_list$⟩拷贝到 $e.parent$ 的 Agg_list 中;待 $e.parent$ 处理完成后,根据 $e.parent$ 是否满足结构约束再进行相应的处理.上述过程正是第 2 个优化策略在聚集对象结点上的具体实现.在上述过程中,如果 q_e 的轴是“//”,则 e 所对应的度量属性和维属性同时也是其祖先聚集对象 e'(如果存在,则在 $q_e.S$ 弹出 e 后,e' 位于 $q_e.S$ 的栈顶)对应的度量属性和维属性,因此,此时还要将 $e.v$ 和 $e.dimension\_list$ 合并到 e' 的数据结构中.这是第 3 个优化策略在聚集对象结点上的具体实现.聚集对象的处理过程见算法 4.

算法 3. 中间结点结束标签的处理.

```
processIntermediate(Element e)
1 if bitmap_e=q_e.LMAP then
2   copyDimension(e.dimension_list,e.parent)
3   mergeMeasure(e.v,e.parent)
4 else if q_e.axis="//" then
5   copyDimension(e.dimension_list,q_e.S.top)
6   mergeMeasure(e.v,q_e.S.top)
```

算法 4. 聚集对象结束标签的处理.

```
processAggregationobject(Element e)
1 if bitmap_e=q_e.FMAP AND e.dimenstion_list!=EMPTY
2   AND e.v!=EMPTY then
3   mergeResult(e)
4   for e.Agg_list 中的每个结点 r do
5     mergeResult(r)
6 else if bitmap_e & q_e.LMAP=q_e.LMAP then
7   copyAggObject(e,e.parent)
8   if q_e.axis="//" & q_e.S!=EMPTY then
9     mergeAggobject(e,q_e.S.top)
```

若 q_e 是主干结点,则同样根据 e 是否满足结构约束来执行相应的动作.如果 $bitmap_e=q_e.FMAP$,则表明 $e.Agg\_list$ 中所有聚集对象的路径约束均已被满足,则将 Agg_list 中的元素用分组函数分组并根据外聚集函数

聚集到最终结果中;否则,如果 $bitmap_e$&q_e.*LMAP*=q_e.*LMAP*,则表明 *e* 元素已满足 q_e 上的结构约束但不满足 q_e 祖先结点上的结构约束,此时将 *e.Agg_list* 合并到 *e.parent* 上.上述过程正是第 2 个优化策略在主干结点上的具体实现.如果上述条件均不成立但 q_e 的轴是"//",那么,尽管 *e* 元素不满足约束,但其祖先中匹配 q_e 的元素仍有可能满足约束,故将 *e.Agg_list* 合并到匹配 q_e 的元素 *e* 的最近的祖先中.主干结点的处理过程见算法 5.

算法 5. 主干结点结束标签的处理.

```
processTrunk(Element e)
1 if bitmap_e=q_e.FMAP then
2   for e.Agg_list 中的每个结点 t do
3     mergeResult(t)
4 else if q_e.LMAP & bitmap_e=q_e.LMAP then
5   copyAggObject(e,e.parent)
6 else if q_e.axis="//" then
7   copyAggObject(e,q_e.S.top)
```

在算法 5 中,*mergeResult*()的功能是把聚集对象按分组函数分组,用外聚集函数将内聚函数计算的临时结果聚集到相应分组的聚集结果中.这里讨论两类常见的分组函数的实现.一类是基于值相等的分组函数,与关系数据库中的分组类似,将 *Agg_list* 和最终结果中的所有分组的维属性值按字典序排序即可实现分组元素的快速定位和合并.另一类是基于集合相等的分组函数,可以用 bloom filter[19]来描述 *Agg_list* 和最终结果中所有分组中维属性值的集合,并将 *Agg_list* 和最终结果中的所有分组按 bloom filter 排序,但合并两个 bloom filter 相同的分组前需要确认它们的维属性集合相等.基于集合相等的分组处理方法还可以处理维属性是复合元素的情况,因为复合元素可以用树来表达,而树可以看成是结点和边的集合.如果分组函数是其他种类,则只需修改 *mergeResult*()的具体实现即可.

需要注意的是,聚集函数包含分布聚集函数、代数聚集函数和整体聚集函数[20].对于分布聚集函数,本文讨论的优化策略可以直接处理.对于代数聚集函数,在每个分组中对聚集函数的每个参数保存一个聚集值,本文的优化策略同样适用.对整体聚集函数的高效处理将另文加以解决.

2.4 复杂性分析

算法的复杂性与聚集函数的类型以及分组函数的类型密切相关.为使算法的复杂性易于讨论,将操作分为两类.一类是简单操作,包括进栈、出栈、位变换等,将一次简单操作的代价记为 $cost_S$.另一类是复杂操作,主要指聚集结果的合并,将一次复杂操作的代价记为 $cost_C$,依赖于聚集函数的类型和分组函数的类型.

在最坏情况下,XML 数据流上每个新到达的元素均能匹配查询中的每个结点.此时,数据流中的每个元素都要和查询中的每个结点进行匹配,简单操作的次数为 $O(Q\times N)$,其中 *N* 是 XML 数据流中扫描的元素个数,*Q* 是查询中元素的个数.

在最坏情况下,复杂操作的次数为 $O((D+M)\times H_{\max})$,其中 *D* 是数据流中维属性的个数,*M* 是数据流中度量属性的个数,$H_{\max}$ 是当前扫描过的 XML 片段中高度的最大值.最坏情况发生在数据流中除去维属性和度量属性而其他结点都是递归的聚集对象时.此时,每个维属性的分组和度量属性的聚集需要进行的最多次数都是与其相关的聚集对象的个数,由于可能存在递归,相关聚集对象的个数最多为 $H_{\max}$.

2.5 复杂约束条件的处理

复杂约束条件指的是含有多个 AND,OR 或 NOT 的约束,如 *a*[*b* or [*c* and *d*] and *e*]/*f*.经简单修改,本文提出的算法可以如下处理这类查询:对每个查询结点 *q*,将该结点上约束的复杂逻辑表达式转换为析取范式,再根据变量在每个合取子式中出现的形式建立一个约束矩阵 A_q,用 A_q 可以判定 *q* 结点上的复杂约束是否被满足.假设析取范式有 *m* 个布尔变量(复杂约束中的第 *i* 个原子条件对应第 *i* 个布尔变量)和 *n* 个子式,且在每个子式中一个变量和其补变量不会同时出现,则所建立的矩阵 *A* 有 *m* 行 *n* 列,其中 a_{ij} 表示第 *i* 个变量在第 *j* 个子式中出现的情况.如果该变量本身出现在第 *j* 个子式中,则 a_{ij}=1;如果该变量的补变量出现在第 *j* 个子式中,则 a_{ij}=−1;否则,a_{ij}=0.每个匹配 *q* 结点的数据元素 *e* 上存储一个 *m* 维的布尔向量 B_e,它表示每个原子条件是否被满足,其中,b_i=1 表示第 *i* 个原子条件被满足,而 b_i=0 表示第 *i* 个原子条件未被满足.下面的命题表明可以用 B_e 和 A_q 来判断 *e* 是否满足 *q* 结点上的复杂约束.

命题. 设元素 *e* 匹配查询结点 q_e,q_e 上的复杂约束对应的矩阵为 A_{q_e},A_{q_e} 的列 *a* 中 1 的个数记为 c_a.在 *e* 的结

束标签到达之前,如果 A_{q_e} 中存在不包含–1 的列 a 使得 $B_e{\cdot}a{=}c_a$,则 e 满足 q_e 上的复杂约束.当 e 的结束标签到达时,如果 A_{q_e} 中存在一列 a 使得 $B_e{\cdot}a{=}c_a$,则 e 满足 q_e 上的复杂约束.

由上述命题可知,对于具有复杂约束的查询结点 q,A_q 中每列是否出现–1 和 1 的个数在约束判定过程中起着重要作用.为了加速判定过程,这些信息需要抽取出来,存储在 q 对应的数据结构中.将 A_q 每列中由 1 的个数构成的 n 维整数向量 C_q 称为 q 的判定向量.用 f_i=1(0)表示 A_q 中第 i 列是(否)出现–1,向量 $F_q=\langle f_1,..,f_n\rangle$ 称为 q 的示性向量.

当 XML 数据流上元素 e 的开始标签到达时,如果 e 匹配具有复杂约束的查询结点 q_e,则根据 A_{q_e} 的维数初始化 B_e 为 m 维零向量,同时初始化 D_e 为一个 n 维零向量,D_e 的第 i 个分量用来保存当前 B_e 与 A_{q_e} 的第 i 列的内积.在处理 e 的某个后代元素时,如果某个原子条件被判断为真,则 B_e 中相应的位被置为 1,同时用 A_{q_e} 的相应行与 D_e 相加的结果来更新 D_e,然后根据命题结合 D_e、判定向量 C_{q_e} 和示性向量 F_{q_e} 判断复杂约束是否被满足.判定过程见算法 6.如果此时复杂约束被满足,则在处理 e 的其他后代元素时不再进行判断.如果处理 e 的后代元素时,复杂约束始终未被满足,则在 e 的结束标签到来时最后进行判断.

算法 6. 根据约束位图判定.

```
bool changeConstraint(int k, Bool is_end)
1 if Be[k]!=1 then
2   Be[k]=1
3   for i=1 to n do
4     De[i]=Aqe[k][i]*Be[k]+De[i]
5     if De[i]=Cqe[i] AND (fi=0||is_end=true) then
6       return true
7 return false
```

对每个元素 e,向量 B_e 的每个分量在整个过程中最多被更新 1 次,只有 B_e 的分量被更新时 D_e 才会被更新,每次更新的时间复杂性为 $O(n)$.所以,处理有复杂约束结点的最坏时间复杂性为 $O(m{\times}n)$.

3 实验结果

影响本文提出算法的因素包括 XML 数据流中接收到结点的个数、XML 数据流中 XML 数据片段的最大高度、查询中约束的个数和维属性的个数.为了研究本文所提出的算法的性能,我们从这些方面来对本文提出的算法进行研究.

3.1 实验配置

我们实现了本文提出的 XML 数据流上的聚集算法.实验的硬件环境是 PC 机,P4 3.0G,1G 内存,150G、7200 转 IDE 硬盘.操作系统是 WindowsXP professional,算法用 Visual C++6.0 实现.

为了有效地测试算法性能,我们采用了 XMark 测试集[21]、DBLP 测试集[22],以及根据图 2 中所示的模式由 XMLGenerator[23]自动生成的一系列数据(下文简称为 GD).XMark 数据包含了递归等复杂的结构,用来测试算法对于多种聚集查询的效率;DBLP 是真实数据,用来测试算法在真实数据上的执行效率;由于在执行带有"//"的聚集查询时,本文所提出的算法的效率受到 XML 片段高度与递归深度的影响,利用 GD 可以根据参数控制 XML 片段的高度,从而测试在不同高度和递归深度的情况下本文所提出算法的效率.对于 Xmark,我们生成了 10M~100M 的数据;对于 GD,我们生成了文档高度为 10~14 的数据.我们采用从磁盘上读取 XML 文档、对其进行 1 次扫描的方式来模拟 XML 数据流.我们仅仅与用同样方法扫描 1 次 XML 文档但不进行任何操作(在下文中简称 SCAN 操作)的时间进行比较.

```
⟨!ELEMENT a (b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT b (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT c (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT d (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT e (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT f (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT g (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT h (#PCDATA|b|c|d|e|f|g|h|i)*⟩
⟨!ELEMENT i (#PCDATA|b|c|d|e|f|g|h|i)*⟩
```

Fig.2 The DTD of the randomly test data

图 2 随机测试数据的 DTD

为了有效地测试算法性能,我们在数据集 XMark 上选取了 8 个查询(QX1~QX8),这组查询覆盖了带有 PC 和 AD 关系的查询以及存在各种结构约束的查询;在数据集 DBLP 上选取了 2 个查询 QD1 和 QD2;在数据集

GD 上,选取了 1 个查询 QG.为了测试查询形态对查询效率的影响,我们设计了模式相同但参数不同的查询.具体的配置将在相应的实验中说明.

3.2 对比实验

本小节给出了查询在 XMark100M 数据、DBLP 数据和高度为 15 的 DG 数据上的执行时间和 SCAN 操作执行时间的比较.比较结果见表 1.可以看出,在 XMark 和 DBLP 数据上执行聚集查询的代价是扫描文档代价的两倍左右.可见,对于一般的文档和查询,与扫描相比,本文所提出的算法具有较高的效率.值得注意的是,算法在 DG 数据集上的执行时间是扫描时间的 4 倍左右,这是因为其对应的查询树的边全都是 AD 边且文档存在多层递归,这些因素均直接影响了算法的效率.

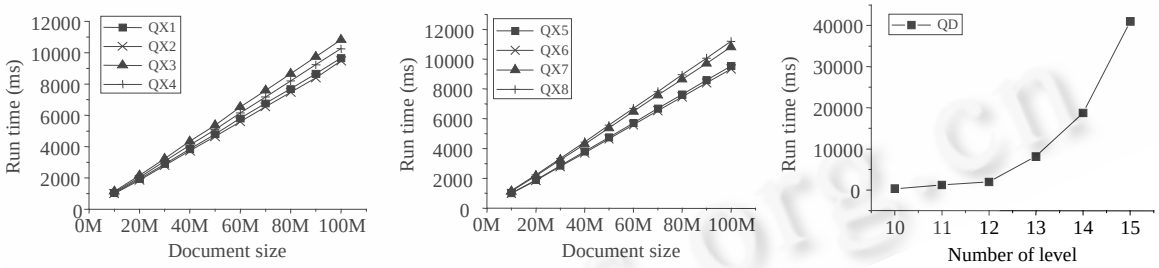
Table 1 Comparison of query execution time and document scan time

表 1 查询执行时间和文档扫描时间的对比

Query	QX1	QX2	QX3	QX4	QX5	QX6	QX7	QX8	QD1	QD2	QG
Execution time (ms)	9 656	9 473	10 824	10 246	9 542	9 339	10 829	11 201	28 146	28 226	40 986
Scan time (ms)	5 683	5 683	5 683	5 683	5 683	5 683	5 683	5 683	13 017	13 017	11 250

3.3 可扩展性

本节给出两组实验,测试了算法的可扩展性.一组使用 XMark 数据,生成的参数从 0.1 变化到 1;另一组使用 GD,其生成参数固定结点最大扇出 6,层数从 10 变化到 15.两组实验的结果分别显示在图 3(a)、图 3(b)和图 3(c)中.可以看出,对 XMark 数据(图 3(a)、图 3(b)),查询处理时间和文档大小成正比,因此,本文提出的算法具有良好的扩展性.对 GD 数据(如图 3(c)所示),可以看出,查询处理时间随文档高度的增加呈指数增长,这是因为文档中的结点个数随文档高度的增加呈指数增长;而在相应的查询中,复杂操作的代价与加法操作类似,其真实代价和简单操作相差不多.因此,文档中的结点个数比文档层数对算法效率的影响更大.



(a) Document size vs. run time (QX1–QX4) (b) Document size vs. run time (QX5–QX8) (c) Document height vs. run time (QD)

(a) 文档大小 vs. 运行时间(QX1–QX4) (b) 文档大小 vs. 运行时间(QX5–QX8) (c) 文档高度 vs. 运行时间(QD)

Fig.3 Experimental results of scalability

图 3 可扩展性实验结果

3.4 变化查询参数

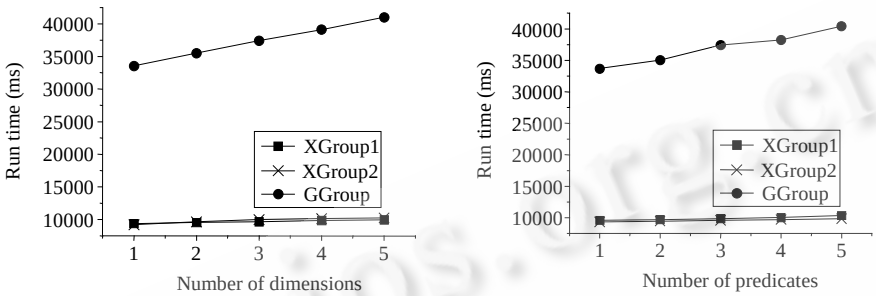
本节通过改变查询中维属性的个数和约束的个数来观察算法运行效率随查询参数变化的趋势.实验分别在 XMark100M 和高度为 15 的 GD 数据上运行.在 XMark 数据集上,通过改变维属性个数和约束个数分别进行了两组实验.在 GD 数据集上进行了一组实验.算法运行时间随维属性个数变化的结果如图 4(a)所示.算法运行时间随约束个数的变化结果如图 4(b)所示.可以看出,运行时间随维属性个数和约束个数的增加呈线性增长,这符合第 2.4 节中关于算法复杂性的结论,同时也说明本文提出的算法关于查询结点个数这一参数具有良好的扩展性.

4 结 论

本文提出了 XML 数据流上有效的聚集查询处理算法.本文提出的算法的基本思想是用位图保存数据元素

满足结构约束的情况,利用栈结构来进行递归情况下“//”轴的处理.本文提出的算法适用于具有复杂结构的聚集函数和具有递归结构的 XML 数据流.理论分析和实验结果表明了本文所提出方法的有效性和可扩展性.除了可以有效地处理 XML 数据流上的聚集以外,本文提出的算法还适用于在仅允许扫描 XML 文档 1 次的情况下,基于 SAX 的聚集查询处理.

本文提出的方法是针对单个聚集查询的,在实际应用中,如果可以把多个相关的聚集函数的查询进行合并处理,则可以提高系统的整体效率.因此,未来的工作包括 XML 数据流上的多聚集查询处理和优化等.



(a) Number of dimensions vs. run time (b) Constraint number vs. run time

(a) 维属性个数 vs. 运行时间 (b) 约束个数 vs. 运行时间

Fig.4 Experimental result of changing query forms

图 4 改变查询形态的实验结果

References:

[1] Diao YL, Fischer PM, Franklin MJ, To R. YFilter: Efficient and scalable filtering of XML documents. In: Agrawal R, Dittrich K, Ngu AH, eds. Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE Computer Society, 2002. 341−344.

[2] Diao YL, Franklin MJ. Query processing for high-volume XML message brokering. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann Publishers, 2003. 261−272.

[3] Diao YL, Altinel M, Franklin MJ, Zhang H, Fischer PM. Path sharing and predicate evaluation for high-performance XML filtering. ACM Trans. on Database Systems, 2003,28(4):467−516.

[4] Gupta AK, Suciu D. Stream processing of XPath queries with predicates. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM, 2003. 419−430.

[5] Peng F, Chawathe SS. XPath queries on streaming data. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 431−442.

[6] Diao YL, Rizvi S, Franklin MJ. Towards an Internet-scale XML dissemination service. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 612−623.

[7] Gao J, Yang DQ, Tang SW, Wang TJ. Tree automata based efficient XPath evaluation over XML data stream. Journal of Software, 2005,16(2):223−232 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/223.htm

[8] Yang WD, Wang QM, Shi BL. Complex twig pattern query Processing over XML streams. Journal of Software, 2007,18(4): 893−904 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/893.htm

[9] Gokhale C, Gupta N, Kumar P, Lakshmanan VS, Ng R, Prakash BA. Complex group-by queries for XML. In: Chirkova R, Dogac A, Ozsu T, Sellis T, eds. Proc. of the 23rd Int'l Conf. on Data Engineering. Istanbul: IEEE Computer Society, 2007. 646−655.

[10] Wang HZ, Li JZ, He YZ, Gao H. Xaggregation: Flexible aggregation of XML data. In: Dong GZ, Tang CJ, Wang W, eds. Proc. of Advances in Web-Age Information Management. Chengdu: Springer-Verlag, 2003. 104−115.

[11] Koch C, Scherzinger S, Schweikardt N, Stegmaier B: Schema-Based scheduling of event Processors and buffer minimization for queries on structured data streams. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 228−239.

[12] Su H, Rundensteiner EA, Mani MM. Semantic query optimization for XQuery over XML streams. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim: ACM, 2005. 277−288.

[13] Kwon J, Rao P, Moon B, Lee S. FiST: Scalable XML document filtering by sequencing twig patterns. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim: ACM, 2005. 217−228.

[14] Gong XQ, Yan Y, Qian WN, Zhou AY: Bloom filter-based XML packets filtering for millions of path queries. In: Toyama M, Sasaki S, eds. Proc. of the 21st Int'l Conf. on Data Engineering. Tokyo: IEEE Computer Society, 2005. 890−901.

[15] Hou S, Jacobsen HA. Predicate-Based filtering of XPath expressions. In: Liu L, Reuter A, Whang KY, Zhang JJ, eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Atlanta: IEEE Computer Society, 2006. 53−64.

[16] Jin CQ, Qian WN, Zhou AY. Analysis and management of streaming data: A survey. Journal of Software, 2004,15(8):1172−1181 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1172.htm

[17] Clark J. XML Path language (XPath). W3C, 1999. http://www.w3.org/TR/XPath

[18] Beyer SK, Chamberlin DD, Colby LS, Özcan F, Pirahesh H, Xu Y. Extending xquery for analytics. In: Özcan v, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM, 2005. 503−514.

[19] Burton H. Bloom. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970,13(7):422−426.

[20] Li JZ, Wang S. Principles of Database Systems. 2nd ed., Beijing: Publishing House of Electronics Industry, 2004. 365−366 (in Chinese).

[21] Schmidt A, Waas F, Kersten ML, Carey JM, Manolescu I, Busse R. XMark: A benchmark for XML data management. In: Bressan S, Chaudhri AB, Lee ML, Yu JX, Lacroix Z, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 974−985.

[22] Ley M. DBLP DTD. 2001. http://www.acm.org/sigmod/dblp/db/about/dblp.dtd

[23] Diaz AL, Lovell D. XML Generator. 1999. http://www.alphaworks.ibm.com/tech/xmlgenerator

附中文参考文献:

[7] 高军,杨冬青,唐世渭,王腾蛟.基于树自动机的 XPath 在 XML 数据流上的高效执行.软件学报,2005,16(2):223−232. http://www.jos.org.cn/1000-9825/16/223.htm

[8] 杨卫东,王清明,施伯乐.针对 XML 流数据的复杂 Twig Pattern 查询处理.软件学报,2007,18(4):893−904. http://www.jos.org.cn/1000-9825/18/893.htm

[16] 金澈清,钱卫宁,周傲英.流数据分析与管理综述.软件学报,2004,15(8):1172−1181. http://www.jos.org.cn/1000-9825/15/1172.htm

[20] 李建中,王珊.数据库系统原理.第 2 版,北京:电子工业出版社,2004.365−366.

王宏志(1978－),男,黑龙江哈尔滨人,博士,讲师,CCF 学生会员,主要研究领域为 XML 数据管理,信息集成.

李建中(1950－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库系统实现技术,数据仓库,半结构化数据,传感器网络,压缩数据库技术,Web 数据集成,数据挖掘,计算生物学.

骆吉洲(1975－),男,博士,讲师,主要研究领域为压缩数据库.