ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.2080−2088
DOI: 10.3724/SP.J.1001.2008.02080

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

XML流数据查询结果的缓存管理*

杨卫东+, 王清明, 施伯乐

(复旦大学 计算机科学技术学院,上海 200433)

Buffer Management of the Results of Queries over XML Streams

YANG Wei-Dong+, WANG Qing-Ming, SHI Bai-Le

(School of Computer Science, Fudan University, Shanghai 200433, China)

+ Corresponding author: E-mail: wdyang@fudan.edu.cn, http://www.cit.fudan.edu.cn

Yang WD, Wang QM, Shi BL. Buffer management of the results of queries over XML streams. *Journal of Software*, 2008,19(8):2080−2088. http://www.jos.org.cn/1000-9825/19/2080.htm

Abstract: This paper presents an approach for processing the buffer of XML stream systematically. In this approach, user interests are represented by XQueries, the recursive documents can be tackled, and the multiple queries can be processed simultaneously. This approach can determine the relationship between two nodes in results on the fly based on the binary code through runtime stack, which avoids the join among a large number of results and improves the system performance and memory usage efficiently.

Key words: XML stream; XQuery; buffer; Twig pattern

摘 要: 提出一种系统地处理 XML 数据流的返回结果集的方法.在该方法中,用户对数据的兴趣用 XQuery 表示,能够处理递归文档以及同时处理多个查询;通过运行时栈驱动的基于二进制的前缀编码,在运行时确定结果集中节点之间的关系,避免了大量结果集之间的连接操作,能够有效减少内存耗费,提高处理性能.

关键词: XML 流;XQuery;缓存;Twig pattern

中图法分类号: TP311 **文献标识码**: A

由于 XML 已经成为 Web 上数据交换的标准,用于各种应用和信息源之间的数据交换,因此,处理 XML 流数据的理论和技术目前已成为流数据研究领域中的一个研究热点.XML 流处理系统要求实时处理在线到达的 XML 流,用户兴趣用 XQuery[1]或 XPath[2]表示,XML 流可能具有复杂的递归、层次结构,因而系统的处理时间和内存耗费是判定系统质量的关键指标.

对每一个匹配的查询,如果需要将查询结果分发给用户,那么面对大量查询,要交付给用户的结果集可能非常巨大,对系统的空间耗费和性能具有重要的影响.例如,对于纳斯达克的实时股票行情服务系统,假定每秒到达的消息个数是 5 000 个,消息的数据量是 1KB,用户查询的数量是 1 000 万个,查询的平均匹配率是 0.001%,则结果集的数据量可能达到大约每秒 4G.基于 XPath 的 XML 流处理系统分为两类:过滤系统和带有谓词的全功

* Supported by the National Natural Science Foundation of China under Grant No.60773076 (国家自然科学基金); the Key R&D Program of Science and Technology Commission of Shanghai Municipality of China under Grant No.061111014 (上海市科委重点攻关项目)

Received 2006-10-09; Accepted 2007-02-12

能(full-fledge)演算系统,对于后一种情况,则需要使用缓存来管理结果或中间结果.然而在基于 XQuery 的 XML 流处理系统中,结果集的缓存管理通常是必需的,而且更为复杂.

本文的工作集中于 XML 流环境下 XQuery 查询的缓存结果管理,迄今为止,这方面的研究工作还未受到应有的重视.与已有研究相比,本文工作主要包括:

(1) 为了能够有效减少缓存的内存耗费,提高处理性能,指出 XQuery 结果集的缓存管理方法和应该遵循的基本原则;

(2) 通过运行时栈驱动的基于二进制的前缀编码,在运行时确定结果集中节点之间的关系,尽早构成返回结果元组,避免了中间结果集之间的连接操作;

(3) 支持递归文档的处理,通过定义一个查询的最低公共祖先谓词查询节点(lowest common ancestor predicate query node,简称 LCAPQN)来帮助尽早删除缓存中的节点,而不是等到整个文档结束之后.

本文第 1 节介绍相关研究工作.第 2 节介绍 XQuery 的表示以及查询匹配算法.第 3 节集中讨论 XQuery 结果集的缓存管理方法.第 4 节给出实验结果与分析.第 5 节给出本文的结论.

1 相关研究工作

目前,已提出各种方法来处理 XML 流数据,例如,基于自动机的方法[3−8]、基于索引的方法[9]、基于 Boom Filter 的方法[10]、基于 prüfer 序列的方法[11]以及其他方法[12−15],其中,大部分研究工作集中于 XML 数据流的查询匹配技术,主要考虑的是改进 XML 流的查询匹配性能和查询处理的空间耗费,而较少考虑查询结果集的缓存管理.

就我们所知,迄今为止只有少量的 XML 流处理系统对相关缓存问题进行了初步研究.例如,XSQ[7]利用层次的增强有限自动机来处理 XPath 与 XML 流数据的匹配,其中也初步讨论了针对 XPath 的缓存管理问题.XSQ 使用缓存区暂存可能的结果,其缓存操作是由自动机驱动的;文献[16]针对基于 XPath 带有谓词的全功能演算系统讨论了缓存管理的最低边界,并给出相应算法,但是该方法只能处理非递归文档.上面两项研究工作都没有讨论基于 XQuery 的 XML 流处理系统的缓存管理问题,文献[16]称这是其下一步的工作.TurboXPath[17]讨论了 XQuery 的缓存问题,但缓存中存放了一些中间结果,并需要在最后对中间结果进行元组集合之间的连接操作.FluX[18]系统使用 XML 流的 DTD 模式来减少 XQuery 计算时的缓存耗费,因而只适用于特定的情况.需要指出的是,这些研究工作都只能处理单个查询的查询匹配和缓存管理,而没有能力同时处理多个查询.

2 XML数据流的查询匹配

XML 数据流的查询匹配算法是在我们以前工作的基础上[12],针对 XQuery 加入缓存管理操作.这里,我们简要介绍 XML 数据流的查询匹配算法.

2.1 XQuery的表示

我们将用户的查询(XQuery)抽象地表示为一棵紧凑查询树(查询节点之间的虚线表示“//”,实线表示“/”),图 1 所示的 XQuery 可用图 2 所示的查询树表示.其中,带有谓词的定位步称为谓词查询节点(PQNode),否则称为通常查询节点(OQNode).谓词查询节点通过 AND/OR 逻辑谓词将其子树连接起来,它关联一个逻辑表达式(在内部表示为抽象语法树).距离根节点最近的谓词节点称为该树所表示查询的接受节点(如果计算过程中接受节点相关逻辑表达式的值为真,则该文档与查询匹配).对于不带谓词的查询,其接受节点则是查询树的叶子节点(如果在匹配过程中到达接受节点,则该文档与查询匹配).用阴影表示的节点对应于 XQuery 中的 return 子句中的节点,称为返回结果节点(RNode),这些节点的元组集构成了用户查询的返回结果集.为了方便起见,将元素、元素的属性等统一视为节点.

```
for $b in doc("/bib.xml")//book
  where $b/publisher/text()="Addison-Wesley" and $b/@year>1991
  return
    ⟨book⟩
      {$b/title}
      {$b/@year}
    ⟨book⟩
```

Fig.1 A sample XQuery

图 1 XQuery 示例

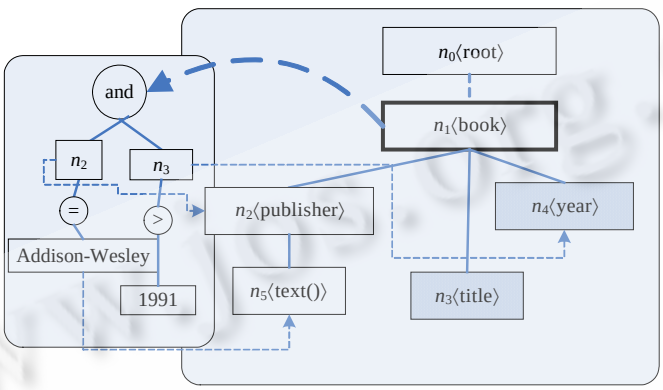


Fig.2 Compact query tree of the sample XQuery

图 2 XQuery 示例的紧凑查询树

2.2 查询匹配算法

在对 XML 流处理的过程中,我们对带有谓词的节点及其子节点采用自底向上的匹配过程;对于其他部分采用自顶向下的匹配过程.算法基于运行栈,分为 OPEN 和 CLOSE 两部分.

OPEN:

OPEN 事件通过回调函数调用该句柄,传入事件名字、元素名字和元素的文档层次.

- 对每一个到达的 XML 元素进行节点测试和文档层次检查.
- 如果节点检查返回 TRUE(事件名、节点名、文档层次匹配),则该节点被压入一个运行时栈.若遇到的节点是 PQNode 或谓词子树的子节点,则其状态标记的值设为 FALSE.若遇到的节点是一个查询的接受节点且不是 PQNode 节点,则一个查询匹配发生.如果该接受节点是 PQNode,则需要等遇到 CLOSE 事件时才能判定文档与查询是否匹配.

CLOSE:

如果遇到的节点是 OQNode,则简单地将节点从栈中弹出.如果遇到的节点是 PQNode 或谓词子树的子节点,则执行下列步骤:

- 如果是叶子节点,则将其状态标记赋为 TRUE,意味着该节点匹配成功.从运行栈弹出该节点,并将当前栈顶节点(当前栈顶节点是其父节点)中相关的逻辑表达式中对应的项赋值为 TRUE.
- 如果是谓词子树中的中间节点,则计算其逻辑表达式(此时,它的孩子节点都已处理过).如果逻辑表达式的值为 TRUE,则意味着该节点匹配,将其状态标记赋值为 TRUE;否则,将其状态标记赋值为 FALSE.从栈中弹出该节点,并将当前栈顶节点的相关逻辑表达式中的对应项赋值为其状态标记的值(TRUE 或 FALSE).
- 如果是一个查询的接受节点,则计算其逻辑表达式,并将逻辑表达式的结果赋给状态标记.如果是 TRUE,则文档与该查询匹配;如果是 FALSE,则文档与该查询不匹配.

- 继续上述过程,直到所有 PQNode 得到处理节点.

另外,在 OPEN 事件中,若遇到的是返回结果节点(RNode),则将该节点放入缓存池中.若确定匹配成功或失败,则从缓存中删除相应节点.下一节,我们将集中讨论对返回结果的缓存的管理.

3 XML数据流查询的缓存管理

在查询的过程中,由于是单遍处理文档流,因此当遇到返回节点时,要将该节点放入缓存中.缓存中存储的是可能的查询结果集.为了优化 XML 流处理的缓存管理,我们提出需要遵循以下原则:

(1) 缓存中只存放可能构成结果元组的节点;

(2) 尽早删除缓存中不满足条件的返回节点;

(3) 尽早将缓存中满足条件的节点构造为结果元组分发给用户,并且从缓存中删除后面不会再使用的返回节点.

返回结果的缓存管理包括 3 个主要功能:向缓存中加入返回节点;构造满足用户查询条件的元组,并将结果元组分发给用户;从缓存中删除无用的节点.缓存管理最基本的操作是:向缓存中加入节点;结果元组的构造;从缓存中删除节点.

对于 XQuery,如果在 XML 流处理过程中遇到返回节点,则将该节点放入缓存,因此,遵循原则(1)是很自然的.构造结果元组的一个时机是在查询匹配过程完成后,通过不同节点集合之间的连接操作来确定哪些节点可以组成一个满足查询条件的结果元组(例如 TurboXPath).这样做有两点不足:其一,在元组构造时,缓存中可能还有中间结果,从而增加缓存的空间耗费;其二,节点集合之间的连接操作会增加系统的时间耗费.因此,我们提出在查询匹配的过程中构造满足查询条件的结果元组.

若要遵循原则(3),需要在两种情况下及时从缓存中删除相应节点:

a) 在查询匹配的过程中,当确定匹配不成功时,及时从缓存中删除相应的结果节点;

b) 在查询匹配的过程中,当确定一个查询匹配成功时,向用户分发查询结果元组,同时也从缓存中删除相应的节点.

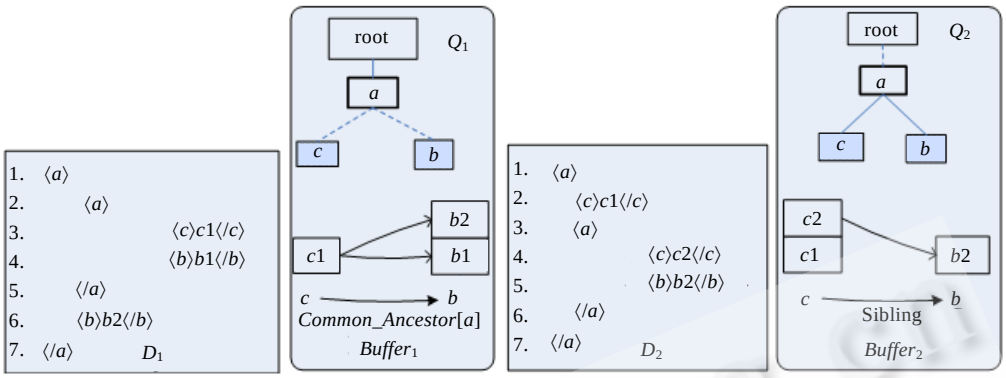
由于一个 XML 流中可能存在多个文档片断满足一个 XQuery 查询,且彼此之间会共用某些结果节点,因此并不总是能在查询匹配时立即删除某些节点,需要找出合适的时机进行删除操作,以改进缓存管理的空间耗费.

3.1 缓存管理的基本方法

受到 TwigStack[19]的启发,我们定义一组缓存池来存放要缓存的节点.对于每一个返回节点,都定义一个缓存池与其相关联.根据用户提交的查询可以得知不同缓存池中的节点之间应该具有的语义关系,并用这样的关系将这些缓存池链接起来.这样,根据缓存池之间的语义关系,各缓存池中节点链接起来的一条路径,就构成了返回结果中的一个元组.

考虑图 3(a)的 XML 文档 D_1 和图 3(b)的查询 Q_1.对于 D_1 和 Q_1,要向用户返回所有满足 a 的子孙 c 和 b 构成的元组,即{⟨$c1$,$b1$⟩,⟨$c1$,$b2$⟩}.根据查询 Q_1,缓存池 b 中的节点与缓存池 a 中的节点之间的语义关系是具有共同的祖先 a.当查询引擎遇到 D_1 中的 $c1$ 时,将其放入与 c 关联的缓存池;当遇到 $b1$ 时,将其放入与 b 相关联的缓存池中,因为 $c1$ 和 $b1$ 具有共同的祖先 a,所以建立 $c1$ 到 $b1$ 的引用;当确定查询 Q_1 满足时,可将⟨$c1$,$b1$⟩元组返回给用户;在遇到 $b2$ 时,将 $b2$ 放入与 b 相关联的缓存池中(由于 $b2$ 和 $c1$ 具有共同的祖先 a,故建立 $c1$ 到 $b2$ 的引用),当确定查询 Q_1 再次得到满足时,可将⟨$c1$,$b2$⟩元组返回给用户.如图 3(b)中的 $Buffer_1$ 所示.

再考虑图 3(c)和图 3(d).希望返回所有由 a 元素的子元素 c 和 b 组成的元组.当查询引擎遇到 D_2 中的 $c1$ 时,将其缓存于 c 的缓存池中;$c2$ 也同样如此;当遇到 $b2$ 时,将其缓存于 b 的缓存池中.根据查询 Q_2,要求 c 的缓存池中的元素和 b 的缓存池中的元素是兄弟关系.现在,由于 a 是接受节点,故对其进行谓词逻辑表达式运算并确定结果为真,由此得到查询的一个匹配.此时,c 的缓存池中分别有 $c1$ 和 $c2$ 元素,b 节点缓存池中有一个 $b2$ 元素,因此,必须判定匹配的结果是⟨$c2$,$b2$⟩,而不是⟨$c1$,$b2$⟩.为此,我们采用基于运行时栈的前缀编码来进行判断.



(a) Document example D_1 (b) Q_1 (c) Document example D_2 (d) Q_2

(a) 文档例子 D_1 (b) Q_1 (c)文档例子 D_2 (d) Q_2

Fig.3 Samples of queries and documents

图 3 查询和文档示例

3.2 基于运行时栈的前缀编码表示

各缓存池中的节点之间的关系比较复杂,例如祖先后代关系、父子关系、兄弟关系、共同祖先关系等,利用基于前缀的编码方法可以快速判断(常量时间)它们之间的关系.在实际实现中,我们使用二进制串前缀编码方案[19],如图 4 所示(节点 10 不是节点 0.10 的祖先节点;节点 0.10 与 0.110 是兄弟节点).由于我们只需要在返回结果的缓存管理时才会使用文档节点的二进制编码信息,因此,我们不是对整个文档编码,而是只对进入运行时栈的节点编码,这样可以通过缩小编码的范围来提高处理效率.

定理 1. 在第 2.2 节的查询匹配算法中,对于 XML 文档 D(它对应于一棵文档树 T)和查询 Q,进入运行时栈查询节点所对应文档元素构成了一棵文档树 T',T'与原文档树 T 同根,并且是 T 的一棵子树.

例如,图 5 的右面是用户提交的查询 Q_3,左面表示的是对应的一棵 XML 文档树 D_3.在 XML 流处理过程中,对于查询 Q_3,进入运行时栈的节点对应的 XML 文档部分就是在文档树中用曲线勾勒出来的部分,它构成一棵与原文档的同根的子树.因此,由运行时栈驱动进行的前缀编码,只是对整个 XML 文档的一部分进行编码,而同时又能正确地反映文档节点之间的关系.

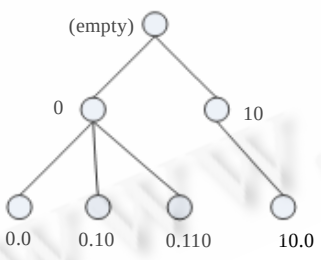


Fig.4 Binary-Code prefix

图 4 二进制前缀编码

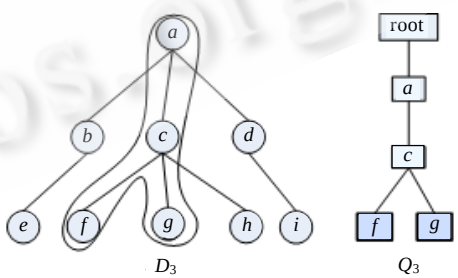


Fig.5 Runtime-Stack-Driven prefix code sample

图 5 运行时栈驱动的前缀编码示例

3.3 删除操作

在对 XML 流的查询处理过程中,由于查询匹配的失败,某些缓存的元素应该及时删除,以节省缓存的内存耗费以及提高性能.这一点很容易实现.如图 6 所示,当遇到 $b1$ 的结束标记时,可以确定 $b1$ 和 $c1$ 不满足查询匹配条件,因此,这时应该及时将 $b1$ 和 $c1$ 从缓存中删除(如图 6 中的 $Buffer_4$ 所示).

另一种需要执行删除操作的情况是,在查询匹配成功时,除了要向用户分发查询结果集以外,直观上讲,同时也应该从缓存中删除相应节点(如图 6 所示,在查询匹配成功时,⟨$b2$,$c2$,$d2$⟩构成返回结果的一个元组,则在向用

户分发后,可以将它们从缓存中删除).但是,实际情况要复杂得多.我们再回过头来考虑图 3 所示的例子:对于图 3(a)所示的文档 $D1$ 和图 3(b)所示的查询 Q_1:在 XML 流处理过程中,当遇到内层 a 元素的结束标记时(D_1 文档中的第 5 行),查询匹配成功,得到元组⟨$c1$,$b1$⟩,但是却不能删除缓存中的 $c1$ 和 $b1$,因为可能有多个与查询匹配的元组,即后面遇到的 b 还可能与 $c1$ 构成满足条件的元组.其原因是,D_1 是递归文档并且查询 Q_1 中 a 与 c 和 b 之间是子孙轴("//").而对于图 3(c)中的 D_2 和图 3(d)中的 Q_2,在遇到内层 a 的结束标记时(D_2 文档中的第 5 行),查询匹配成功,得到⟨$c2$,$b2$⟩,这时,显然可以将 $c2$ 和 $b2$ 从缓存中删除.

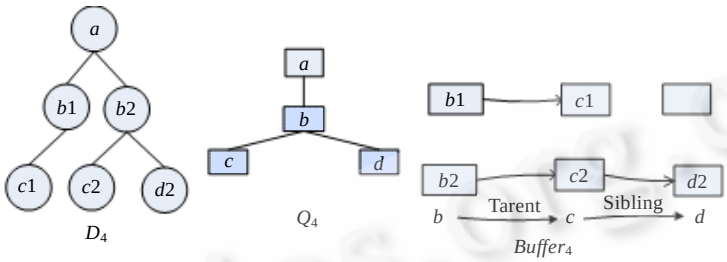


Fig.6 Elimination of nodes in buffer due to matching failed

图 6 匹配失败时缓存节点的删除

另一种存在多个匹配元组的原因是,文档的 DTD 中含有 1 对多的关系.考虑 DTD:⟨!ELEMENT a (b,c+)⟩,由该 DTD 可知,a 与 b 是一对一关系,a 与 c 是一对多关系,从而可推知,对于查询"$a[b][c]$"(假定 b 和 c 是返回元素),一个 b 元素可能与多个 c 元素构成多个返回元组.由上面的分析可知,造成这种复杂性的原因是递归文档和查询中的"//",以及元素之间的一对多关系.因此,通过 DTD 或 XML Schema 所提供文档的结构和约束信息(例如,是否允许递归文档,元素之间的基数约束、顺序等),可以对缓存管理进行优化(具体优化不在本文的讨论范围内).但是在有些应用中,我们并不知道关于 DTD 或 XML Schema 的任何信息,为了及时删除缓存中的已经"无用"节点(这里"无用"的含义是指将这些节点构成的元组分发给用户后,其中的节点不会与查询处理后面遇到的节点构成新的返回元组),我们分以下两种情况确定查询匹配成功后缓存节点删除的时机:

第 1 种情况:如果构成返回元组的节点属于通常查询节点,则在遇到其关闭标记时,从缓存中删除该节点.

第 2 种情况:如果构成返回元组的节点是谓词子树的子节点(包含谓词节点),则给出如下定义:

定义 1. 最低公共祖先谓词节点(LCAPQN):如果查询中的一个节点是 LCAPQN,则需满足下列条件:

- 该节点是一个谓词节点;
- 该节点是所有返回节点的一个公共祖先;
- 在所有返回节点的全部公共祖先中,该节点不是其他任何一个的父亲节点.

根据 LCAPQN,在文档处理过程中,当遇到文档中对应于 LCAPQN 的节点的关闭标记时,即可将缓存中的返回节点从缓存中删除.在图 7 中,Q_5 中的 LCAPQN 是"c",返回节点是"d"和"e",因此,在遇到文档 D_5 中的 c 的关闭标记时(即⟨$c1$/⟩和⟨$c2$/⟩),从缓存中删除节点.

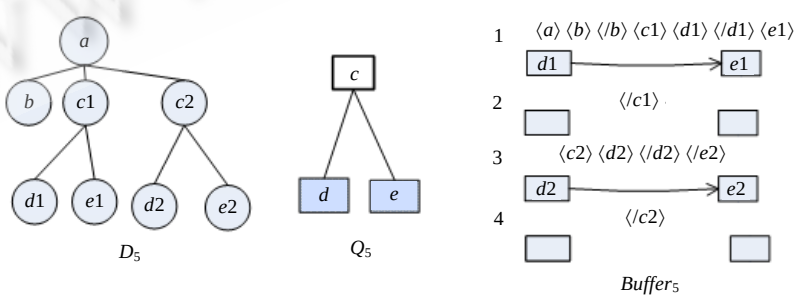


Fig.7 Examples of buffer nodes elimination

图 7 缓存节点删除例子

对于递归的嵌套文档,并且查询中的 LCAPQN 与返回节点之间包含“//”,就可能会出现这样一种情况:文档中存在着处于不同层次上的多个节点与 LCAPQN 对应,这时,只能在遇到相应的最外层节点的关闭标记时执行缓存的删除操作.

3.4 多查询的缓存管理

前面讨论了针对单个查询的缓存管理问题.在实际系统中,XML 流数据处理系统通常面对的是大量用户,因此要求系统具有同时处理大量查询的能力.而来自用户的大量查询中,可能会具有许多共同的部分,共享其共同部分可以节省系统的存储空间和执行时间,对改善系统性能非常重要.我们将所有 XQuery 的紧凑查询树合并为一个单一的共享前缀的紧凑查询树.其中,节点名以及算子(“/”或“//”)相同的前缀可以合并.另外,若返回结果中的节点是一个多个查询共享的节点,则缓存结果节点的缓存池被这些查询所共享(合并方法类似于我们前面的工作[12]).与共享前缀的紧凑查询树中的其他节点一样,共享的结果节点的缓存池关联有查询标识.如果两个缓存池被多个查询共享,则两个缓存池构成的缓存池链也同样被共享.缓存结果的共享可以有效节省缓存空间以及优化构造缓存结果元组的性能.但是,缓存的共享可能延迟共享缓存节点的删除时间,因为对于一个查询而言,可以删除的缓存节点可能还会被其他查询使用,而只有当共享该缓存节点的所有查询都认为这个节点“无用”时,才可将其从缓存中删除.

4 实 验

我们用 Java 实现了原型系统,称为 LeoXSSII.其中,使用 IBM 的基于事件的 XML 解析器[20].系统运行的环境是 Eclipse3.1,机器主频为 2.7G,内存为 512M.所有的实验都运行于 Window XP 专业版.我们使用 XMark[21],DBLP[22]两个数据集进行实验.在处理单个查询的情况下,我们通过改变查询的谓词节点个数(表示查询的分支数目)、路径长度、“//”和“*”出现的个数来产生查询实例,数据集的大小是 8M~32M 不等.实验结果表明,在单查询情况下,几乎对所有情况,内存耗费基本保持在常量(不到 1M),文档大小对其影响不大,处理时间随着 XML 文档大小的逐步增加(从 83ms 增加到 165ms).可以看出,单查询情况下系统通常表现良好.

我们重点考虑在在同时处理大量 XQuery 的情况下,LeoXSSII 在性能和内存使用方面的情况.实验的度量指标如下:(1) 输入文档的大小(10K,20K,50K,1M,3M);(2) 查询的数目(5 000~100 000);(3) 谓词节点的数目(3个);(4) 返回结果所包含的元素个数(1个~3个元素).这里使用的数据集是 DBLP,使用 Xmark 数据集的结果与此类似.

图 8 首先给出针对小文档的部分实验结果.以查询的分支节点(PQNode)数是 3、返回节点数是 3 为例,我们用查询生成器生成 XQuery(扩充 YFilter 的查询生产器),其参数是:6 0.2 0.2—*num_nestedpaths*=3—*distinct*=TRUE,其中,6 表示查询深度,两个 0.2 表示“//”和“*”出现的概率,3 表示谓词节点数,TRUE 表示每个查询都不相同.实验结果如图 8 所示,其中,纵轴表示时间(ms×10),横轴表示查询个数,分别为 5 000,10 000,50 000,100 000,文档的大小为 10K,20K,50K.图 9 显示了在文档大小为 3M 的情况下,系统内存的使用情况:随着查询数目的增加,系统使用的内存逐步增加.当查询数目很大时,每一个查询数目的返回结果数目虽然不同,但整体分布情况大致相似,因此,对内存的使用影响不大.

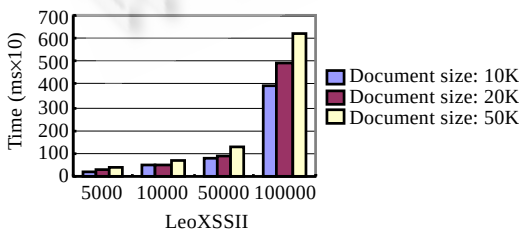


Fig.8 Experiment 1

图 8 实验 1

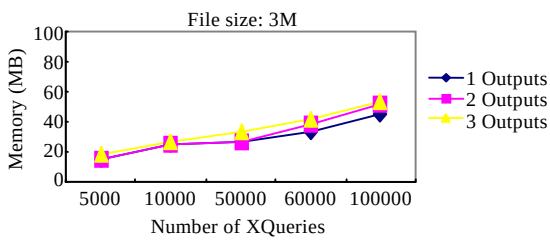


Fig.9 Experiment 2

图 9 实验 2

在图 10 中,文档的大小分别是 1M 和 3M,*X* 轴表示查询的数目从 5 000 递增到 100 000,*Y* 轴表示性能的变化情况,图表反映的是随着查询数目的递增(从 5 000 到 100 000),查询的返回结果所包含元素的数目的变化,在输入文档流分别为 1M 和 3M 的情况下,系统性能的变化情况.由实验结果可以看出,随着查询数据的增加以及文档大小的增加(从 1M 到 3M),系统的运行时间逐步递增.当查询数目从 50 000 增加到 100 000 时,系统处理时间增加幅度较大.而输出结果包含的元素个数(这里没有考虑输出结果的分布情况,即它们之间是祖先关系还是兄弟关系等),对系统性能的影响也不明显.

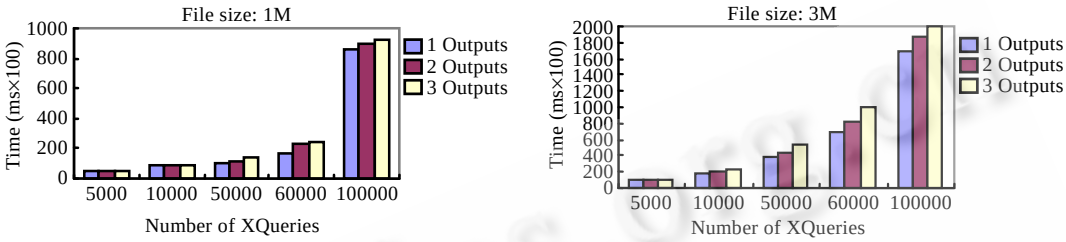


Fig.10 Experiment 3

图 10 实验 3

5 结束语

目前,XML 流数据环境下,针对 XQuery 的返回结果的缓存管理还缺乏系统的研究,只有少数文献对此进行了初步讨论.与已有的研究工作相比,我们的方法具有如下的特点:利用运行时栈驱动的二进制前缀编码及早构造结果元组,避免了大量中间结果节点之间的连接操作;能够及早删除“无用”节点;支持多查询的缓存管理.进一步的工作是完善我们的系统,并研究如何在分布环境下有效地向用户点对点地分发返回结果.

References:

[1] XQuery 1.0: An XML query language. 2007. http://www.w3.org/TR/xquery/

[2] Berglund A, Boag S, Chamberlin D, Fernandez MF, Kay M, Robie J, Simeon J. XML path language (XPath) 2.0. W3C. 2004. http://www.w3.org/TR/xpath20

[3] Altinel M, FrankliN MJ. Efficient filtering of XML documents for selective dissemination of information. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, Whang KY, eds. Proc. of the VLDB 2000. San Francisco: Morgan Kaufmann Publishers, 2000. 53−64.

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

[5] Gupta A, Suciu D. Stream processing of XPath queries with predicates. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the ACM SIGMOD Conf. San Diego: ACM Press, 2003. 419−430.

[6] Green TJ, Miklau G, Onizuka M, Suciu D. Processing XML streams with deterministic automata. In: Calvanese D, Lenzerini M, Motwani R, eds. Proc. of the ICDT 2003. Berlin: Springer-Verlag, 2003. 173−189.

[7] Peng F, Chawathe SS. XSQ: A streaming XPath engine. ACM Trans. on Database Systems, 2005,30(2):577−623.

[8] 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

[9] Bruno N, Gravano L, Koudas N, Srivastava D. Navigation- vs. index-based XML multiquery processing. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the ICDE 2003. Los Alamitos: IEEE Computer Society, 2003. 139−150.

[10] Gong XQ, Qian WN, Yan Y, Zhou AY. Bloom filter-based XML packets filtering for millions of path queries. In: Proc. of the ICDE 2005. Tokyo: IEEE Computer Society, 2005. 890−901.

[11] 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 PÅ, Ooi BC, eds. Proc. of the VLDB 2005. Trondheim: ACM, 2005. 217−228.

[12] Yang WD, 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

[13] Hou S, Jacobsen HA. Predicate-Based filtering of XPath expressions. In: Reuter A, Whang KY, Zhang JJ, eds. Proc. of the 22nd IEEE Int'l Conf. on Data Engineering. Piscataway: Institute of Electrical and Electronics Engineers Computer Society, 2006. 53.

[14] Chen Y, Davidson SB, Zheng YF. An efficient XPath query processor for XML streams. In: Liu L, Reuter A, Whang KY, JJ Zhang, eds. Proc. of the 22nd IEEE Int'l Conf. on Data Engineering. Piscataway: Institute of Electrical and Electronics Engineers Computer Society, 2006. 79.

[15] Candan KS, Hsiung WP, Chen ST, Tatemura J, Agrawal D. AFilter: Adaptable XML filtering with prefix caching and suffix clustering. In: Dayal U, Whang KY, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim YK, eds. Proc. of the VLDB 2006. Seoul: ACM, 2006. 559−570.

[16] Yossef ZB, Fontoura M, Josifovski V. Buffering in query evaluation over XML streams. In: Li C, ed. Proc. of the PODS 2005. New York: Association for Computing Machinery, 2005. 216−227.

[17] Josifovski V, Fontoura M, Barta A. Querying XML streams. The VLDB Journal, 2005,14(2):197−210.

[18] 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 VLDB. Toronto: Morgan Kaufmann Publishers, 2004. 228−239.

[19] Li C, Ling TW. An improved prefix labeling scheme: A binary string approach for dynamic ordered XML. In: Zhou LZ, Ooi BC, Meng XF, eds. Proc. of the DASFAA. Beijing: Springer-Verlag, 2005. 125−137.

[20] Megginson D. Simple API for XML. 2000. http://sax.sourceforge.net

[21] Busse R, Carey M, Florescu D, Kersten M, Manolescu I, Schmidt A, Waas F. XMark: An XML benchmark project. 2001. http://monetdb.cwi.nl/xml/index.html

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

附中文参考文献:

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

[12] 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

杨卫东(1967－),男,陕西西安人,博士,副教授,主要研究领域为 XML 数据管理,软件开发方法,面向对象技术.

王清明(1981－),男,硕士生,主要研究领域为 XML 数据流处理.

施伯乐(1935－),男,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,知识库.