第36卷 第14期 计 算 机 工 程 2010年7月
Vol.36 No.14 Computer Engineering July 2010
·软件技术与数据库· 文章编号：1000—3428(2010)14—0046—03 文献标识码：A 中图分类号：TP311.13

基于缓存技术的XML代数查询优化

冯 琳[1]，顾进广[1,2]，李尧尧[1]，曹雅楠[1]
(1. 武汉科技大学计算机科学与技术学院，武汉 430081；2. 东南大学计算机科学与工程学院，南京 210096)

摘 要：针对XML代数查询优化，采用缓存技术将经常被查询到的模式树保存在缓存中，根据匹配规则判断查询模式树是否与缓存中的模式树匹配，对匹配的部分直接从缓存中获取部分查询结果，以提高查询效率。分析查询与缓存的模式匹配规则，并通过实验证明该规则的可行性和有效性。

关键词：模式树；缓存；模式匹配

Query Optimization of XML Algebra Based on Cache Technology

FENG Lin[1], GU Jin-guang[1,2], LI Yao-yao[1], CAO Ya-nan[1]
(1. College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430081;
2. College of Computer Science and Engineering, Southeast University, Nanjing 210096)

【Abstract】 To optimize the XML algebra query, this paper uses the cache technology to save the frequently queried XML pattern trees. It can determine whether the pattern trees of request can match the pattern trees in the cache. The results of the matched one can be got from cache directly to improve the query efficiency. It analyzes the rules of pattern match for query and cache, and proves the feasibility and effectiveness of the rules according to the results of experiment.

【Key words】 pattern tree; cache; pattern match

1 概述

随着Internet上信息的表示和存储不断向XML格式迁移，XML逐渐成为Internet上数据表示和交换的实际标准。对XML的查询、定位等需求不断增加，对XML数据的合理存储和快速查询要求随之增加。本文在XML数据库中使用缓存技术来提高查询效率，并引入一种查询代数树——TAX[1]作为缓存的数据存储模型。

2 XML查询缓存

在传统的关系数据库中，为了提高查询响应速度，可以进行查询语句改写、查询计划优化和基于代价的执行顺序选择，而缓存是查询优化的一项重要技术[2]。缓存技术最早在关系数据库中提出，是数据库系统的一个重要功能。数据库管理系统有选择地对查询结果进行缓存，把经常被查询的数据(即热数据)保存在缓存中。基于缓存的XML查询优化的处理过程如下：将经常被查询的XML节点信息存储到缓存中，当新的查询到来时，先比较缓存中的模式树与查询模式树，对匹配部分的查询结果直接从缓存中获取，而不必从原数据库中获得，从而有效提高数据库的查询效率和响应能力。当前XML数据库系统的缓存技术研究主要集中在语义缓存[2]。语义缓存是指把用户的查询请求与结果数据对应起来，一起保存到缓存中。语义缓存查询处理的基本思想如下：先确定是否有缓存项与查询请求间存在精确匹配关系，如果有，则查询请求可以由本地缓存处理。若没有，但存在缓存项与查询是包含匹配关系，则需要进行查询剪裁，得到缓存能够回答的探测查询和不能回答的剩余查询。剩余查询需要送服务器处理，查询的最终结果集由探测查询的结果和剩余查询的结果合并得到。在对XML数据的缓存过程中，主要研究以下3个方面：

(1)缓存数据的选择，选择并保存常用的数据。

(2)缓存数据的格式，即以什么样的形式来表示XML数据。

(3)缓存匹配算法，对新的查询需要恰当的匹配算法来确定已缓存的查询结果是否可以回答此查询。

3 基于语义缓存的代数查询优化

3.1 查询代数TAX

TAX是XML数据库TIMBER[3]采用的代数，它引入模式树和实例树的概念，用模式树表示查询的模式和相关的谓词限定，用实例树表示满足模式树的实例。一个模式树可以表示为一个二元组$P=(T,F)$，其中，T是一个带有节点和边标签的树；F是一个公式，它由一系列用于限定各节点值或属性值的谓词表达式组合而成。

在对XQuery查询的解析过程中，将XQuery查询语句转化为TAX代数表达式。代数表达式由代数操作符和其相应的模式树组成。查询频率较高的查询语句以模式树的形式保存在缓存中，同时保存其相应的查询结果和相关的逻辑操作。基于TAX代数表达式的特点，在检查缓存中的结果是否能回答新查询时，先要判断逻辑操作是否相同，再判断模式树是否匹配。即使2棵模式树完全相同，如果相关逻辑操作不同，这2棵树也不能进行匹配。

基金项目：国家自然科学基金资助项目(60803160)；中国博士后科学基金资助项目(20060400275)；江苏省博士后科学基金资助项目(0601009B)；武汉科技大学科研基金资助项目(2008RC04, 2008TD04)
作者简介：冯 琳(1984－)，女，硕士研究生，主研方向：XML数据库，智能信息处理；顾进广，副教授、博士；李尧尧、曹雅楠，硕士研究生
收稿日期：2010-02-15 **E-mail：**fenglin20042003@126.com

3.2 模式匹配

在查询过程中，缓存中的模式树为被匹配树，查询模式树为匹配树。当被匹配树包含于匹配树时，这 2 棵模式树是匹配的，即能够把被匹配树扩展成一棵能与匹配树完全匹配(节点及节点间的关系都匹配)的树。如果无需扩展就能完全匹配，那么这 2 棵模式树是完全相等的。

在模式树的匹配中，先使用路径集合表示查询模式树和缓存中的模式树，每条路径都是从根节点到叶子节点深度遍历产生的。先检查被匹配树的根节点是否与匹配树中的某一节点匹配，然后做进一步匹配。任意 2 条路径中的节点根据匹配规则自上而下逐一匹配，如果缓存模式树路径集合中的每条路径能包含于查询模式树中路径集合中的某条路径，则这 2 棵模式树是匹配的。图 1 中模式树 *P*1 和 *P*2 拆分而成的路径集合分别为(*ABC, ADE, ADF*)和(*DE, DF*)。根据匹配规则可知，模式树 *P*1 和 *P*2 是相匹配的，在匹配过程中，把模式树 *P*1 剪成 2 个部分，其中一部分是与 *P*2 完全相等的树，另一部分是模式树 *P*2 的补充树。*P*2 可以根据补充树扩展成与 *P*1 完全相等的树。

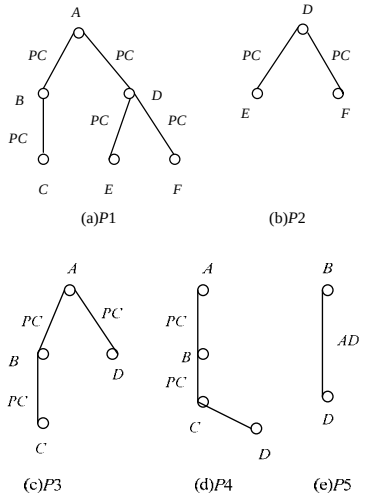


图 1　模式树的匹配与拆分

在 2 棵模式树的匹配过程中，除了对节点的匹配，还应对节点间的边进行匹配。模式树的匹配规则如下：

(1)节点匹配。节点类型、节点 tag、节点值必须相等。

(2)若 2 个节点匹配，节点间边的类型也相同，则这条边也是匹配的。

(3)对于匹配树和被匹配树上的 2 个节点，若类型和 tag 都匹配，且 2 个节点都有条件限定，对于等于或不等于条件，若要求条件表达式相同，则条件匹配。若匹配条件包含在被匹配条件中，如匹配条件为$3.content<1 990，被匹配条件为$3.content<2 000，那么这 2 个条件也是匹配的，查询结果可以在缓存中获得，只要去掉大于 1 990 且小于 2 000 的节点，加以修正即可。

(4)对于 2 棵相匹配的模式树，若匹配树都包含被匹配树，在可以将匹配树拆分成 2 个部分，其中一部分与被匹配树完全相等，其查询结果能完全从缓存中取得。另一部分是被匹配树的补充树，要保持树上节点信息和条件等都与匹配树上相应节点相同，最终的查询结果即这些查询结果的连接。拆分情况如图 1 所示，*P*1 为匹配树，*P*2 为被匹配树，可以把 *P*1 拆分成 *P*2 和 *P*3，*P*3 中应保留 *P*2 的根节点 *D*。

上述匹配过程将 2 条路径中的节点逐一匹配，要求对应的节点相同，节点间的关系也相同。查询语言 XQuery 的核心是路径表达式，本文根据路径表达式的等价规则[4]对模式树匹配规则做如下补充：如果匹配树中有 2 个节点 *B*、*D* 分别与被匹配树中 2 个节点 *B*、*D* 匹配，但在被匹配树中边类型为 *AD*，而匹配树中 *B* 有一个子节点 *C*，则 2 个节点的关系为 *PC*。在被匹配树中，*D* 是 *C* 的子孙节点。参照原 XML 文档树模型，若节点 *C* 是 *D* 唯一的父节点，则 2 个模式树是匹配的。或者节点 *C* 是 *D* 的关键祖先，即访问 *D* 必须先访问 *C*，则 2 棵模式树匹配。如图 1 所示，*P*4 为匹配模式树，*P*5 为被匹配模式树。若 *CD* 边为 *PC* 边，则当 *C* 为 *D* 的唯一父节点时，模式树匹配。若 *CD* 边为 *AD* 边，则当 *C* 为 *D* 的关键祖先节点时，模式树匹配，拆分规则同上。综合上述匹配规则，其相关算法描述如下：

```
Tree match()
{
  for each Qpath in the querypaths //querypaths 为查询模式树的
//路径表达式集合
{   int k=0,flag1=0;
      while(unmatch(root of cachetree, nodek of Qpath ))
{ k++;}
flag1=1;
break;
  }//判断被匹配树的根节点是否与匹配树中某一节点匹配
  if(flag1=1)
  {   for each Cpath in the cachepaths //cachepaths 为缓存模式树
//的路径表达式集合
       { int flag2=0;
       for each Qpath in the querypaths
       {int i=0,j=0;
       While(i<count(nodes of Cpath)&&j<count(nodes of Qpath ))
       {if (unmatch(nodei of Cpath,nodej of Qpath)) j++;
        else {i++;j++;}
}//匹配节点的方法是根据上面已定义的匹配规则
if(i= =count(nodes of Cpath)&&j= =count(nodes of Qpath ))
{flag2++;continue;}
  }
}
if(flag2= = count(paths of Cpaths)) return mached;
}
return unmached;
}
```

4　实验评估与分析

为了证明缓存策略能提高查询效率，在 XML 数据库系统——Timber 上进行实验。实验中使用 4 个 XML 文档，分别是 sbook.xml(4 KB)、mimi.xml(22 KB)、pepole.xml(280 KB)、afria.xml(1.39 MB)。根据对匹配规则的分析，可以把匹配分为 4 种类型：部分匹配(查询模式树部分包含缓存模式树)，条件匹配(匹配条件包含在被匹配条件中)，全部匹配(查询模式树与缓存模式树相同)，间接匹配(在路径等价的条件下匹配)。对每个 XML 文档都选用 4 组 XQuery 查询，分别针对上述 4 类匹配进行查询优化。每组查询包含 2 种 XQuery 查询：(1)简单 XQuery 语句，用“a”表示；(2)与第(1)种查询使用相同缓存模式树进行匹配但较复杂的 XQuery 查询，用“b”表示。

本文实验忽略了缓存技术中对于热数据的选取，探测查

询结果与剩余查询结果的连接等相关工作，重点针对匹配规则来验证优化策略的可行性和有效性。根据 TAX 查询代数的特点，实验中的模式树均为选择操作下的模式树。用准备好的 XQuery 查询用例在利用缓存和无缓存 2 种情况下分别对 4 个 XML 文档进行查询，记录 2 种情况下消耗的时间。利用缓存进行查询可以提高查询效率。图 2~图 5 分别给出了对大小不同的 4 个 XML 文档进行查询，复杂程度不同的 XQuery 对应不同的匹配类型所节省的时间。

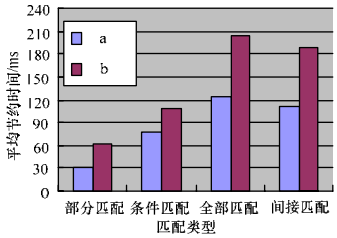


图 2　4 KB 文档的平均节省时间

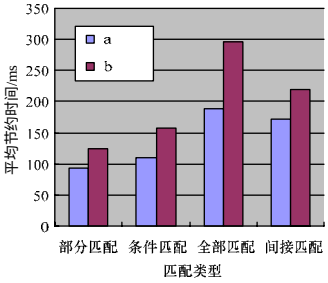


图 3　22 KB 文档的平均节省时间

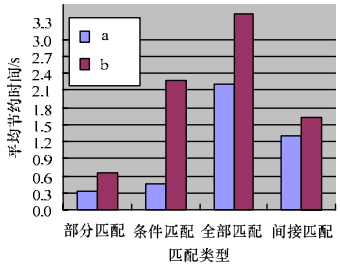


图 4　280 KB 文档的平均节省时间

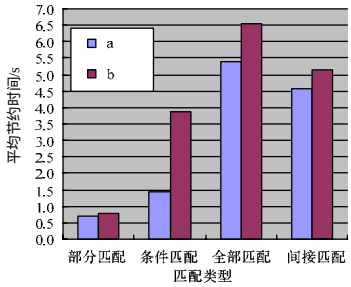


图 5　1.39 MB 文档的平均节省时间

从图 2~图 5 可以看出，在对同一个 XML 文档进行查询的过程中，若采用的缓存模式树和匹配类型相同，则 XQuery 查询越复杂，与无缓存查询相比，其节约的时间越多。

在不同的 XML 文档中查询，当 XQuery 语句的大小相似且缓存匹配类型相同时，与无缓存查询相比，节省的时间如图 6 和图 7 所示。可以看出，无论哪种匹配方式，当 XQuery 查询相似时，与无缓存查询相比，查询的文件越大，则节省的时间越多。

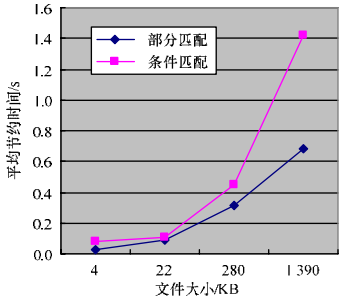


图 6　部分匹配与条件匹配节省的时间

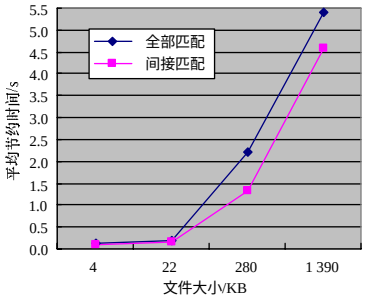


图 7　全部匹配与间接匹配节省的时间

5　结束语

本文将语义缓存技术运用到 XML 代数查询中，缓存区中存储用户的频繁查询模式，以 TAX 模式树的格式存储查询请求，以 TAX 实例树的格式存储查询结果。进行查询处理时，根据匹配规则从缓存中找到查询模式树的匹配部分，直接从缓存中获取该部分的查询结果，从而提高查询效率。如何更好地利用缓存优化查询有待进一步研究，如利用缓存区的缓存替换策略[5]，即选择一定数据，将其从缓存中淘汰出来，并将具有更大利用价值的数据写入缓存。“ 热数据 ” 的选择[6]也是今后可以研究的方向。

参考文献

[1] Jagadish H, Lakshmanan L, Srivastavad D. TAX: A Tree Algebra for XML[C]//Proc. of the 8th International Workshop on Database Programming Languages. Berlin, Germany: [s. n.], 2001: 149-164.

[2] 塔　娜, 冯建华, 李国良. 纯 XML 数据库语义缓存综述[J]. 计算机应用, 2006, 6(12): 2977-2981.

[3] Jagadish H, Lakshmanan L. TIMBER: A Native XML Database[J]. The VLDB Journal, 2002, 2(11): 274-291.

[4] 吕建华, 王国仁, 于　戈. XML 数据的路径表达式查询优化技术[J]. 软件学报, 2003, 14(9): 1615-1620.

[5] 邓　佳. XML 查询缓存中替换策略的研究与设计[D]. 济南: 山东大学, 2007.

[6] Ren Qun. Semantic Caching and Query Processing[J]. IEEE Trans. on Knowledge and Data, 2003, 15(1): 192-210.

编辑　陈　晖