

基于代数的 Transform 查询优化策略

王伟 郭青松 富丽贞 孟小峰

(中国人民大学信息学院, 北京 100872)

(wwcd2005@163.com)

Algebra-based Transform query optimization strategy

Wang Wei, Guo Qingsong, Fu Lizhen, Meng Xiaofeng

(Information School, Renmin University of China, Beijing 100872)

Abstract XQuery/Update defines a special Transform query, which is similar to be hypothetical query in relation databases, and can be expressed as: “Q when {U}”. In other words, the results of query Q are the same as the results after executing hypothetical update {U} on the original database, without actually updating database. The Transform queries need to copy the nodes in XML database and then update copied nodes, so it doesn't affect the database. But Transform queries will usually copy and update a lot of nodes which are useless for query Q and result in high cost. It is critical for query optimization to decrease the number of copied nodes and the update operation. In this paper, we propose a set of rules for Transform query optimization techniques based on OrientXA. Which are implemented in OrientX3.0.

Keywords XML; XQuery/Update; Transform query; Query Optimize

摘要 XQuery/Update 中定义了一种特殊的查询—Transform 查询。Transform 查询类似于关系数据库中的假设查询, 可以表示成假设查询的一般形式: “Q when {U}”, 即查询 Q 的查询结果是假设数据库执行了更新操作{U}以后的结果, 而更新操作 U 实际不修改数据库的状态。Transform 查询需要拷贝 XML 数据库中的结点, 并对拷贝的结点执行更新操作, 所以不影响数据库的状态。但该操作通常拷贝和更新了大量与查询结果无关的结点, 因此如何减少拷贝与更新操作的代价是 Transform 查询处理优化的关键。提出了基于 OrientXA 的 Transform 查询优化方法, 并在 Native XML 数据库系统 OrientX 3.0 里实现和验证了该方法。

关键词 XML; XQuery/Update; Transform 查询; 查询优化

中图法分类号 TP391

XQuery/Update 通过扩展 XQuery 以支持 XML 数据的更新操作[1~4], XQuery/Update 的处理必须有效利用 XQuery 处理技术。XQuery 查询处理主要分为基于导航和基于代数两种。代数式处理方法相对导航式处理方法在效率与优化上有诸多优势。

XQuery/Update 中的 Transform 查询通过拷贝数据库中的结点, 然后对拷贝结点进行更新, 但实际上并不修改数据库的状态。用户提交的 Transform 查询需要拷贝与更新和查询结果无关的结点。因此必须对

Transform 查询进行优化, 减少拷贝结点的数量, 提高查询效率。

本文针对 Transform 查询的特点, 提出了 Transform 查询的三种优化处理策略: Transform 的等价转换、“lazy”及“混合”策略处理 Transform 查询。主要贡献如下:

1) 提出了基于 OrientXA[5]代数的 Transform 查询优化方法。一方面通过等价转换将 Transform 查询转换成一般 XQuery 查询进行处理; 另一方面, 尽可能将 Transform 查询的拷贝与更新操作延后处理。

收稿日期:

基金项目: 国家自然科学基金项目(课题号: 60833005, 60573091), 国家 863 计划(课题号: 2007AA01Z155, 2009AA011904), 教育部博士点基金项目(200800020002)

本文通讯作者: 孟小峰(xfmeng@ruc.edu.cn)

2) 给出了以上三种查询优化方法的选择条件。即对于用户提交的一个 Transform 查询, 如何根据查询选择合适的优化方法进行查询重写;

3) 详细描述了 Transform 查询优化过程;

4) 通过实验验证了我们提出的查询策略的正确性和有效性, 并在 OrientX 中实现了这几种优化策略。

1 相关工作

XQuery/Update 草案是 W3C 组织 2007 年 1 月份提出来的, 以前的研究工作[6,7]关注于 XML 数据在存储上如何更新, 与本文讨论的 XQuery/Update 完全不同的。在 XML 数据库系统中目前有 galax[8]与 MonetDB/XQuery[9]支持 XQuery/Update, 但其未公开发表如何 XQuery/Update。

Transform 查询可以表示成“Q when {U}”的形式。在关系数据库上, 关于假设查询已经有不少相关工作, [9]给出了一种基于关系代数的假设查询处理框架。在关系数据库上执行假设查询“Q when {U}”时, 都是先物化数据库执行更新操作 U 后的结果, 再执行 Q 查询得到查询结果, 即“eager”方法。“eager”方法需要物化大量中间结果, 代价很高。为此, [9]提出了“lazy”方法, 通过关系代数的等价转换来减少物化中间结果, 从而减小假设查询处理的代价。

[10]对 XQuery/Update 中的 Transform 查询处理进行了详细讨论, 它的核心思想还是基于 XQuery 的导航式处理技术。通过构建自动机, 根据当前遇到的结点, 执行自动机上的相关操作, 但一次只能处理一结点, 并不适用于基于代数的查询处理, 本文将结合假设查询的思想讨论基于代数的 Transform 查询优化。

2 Transform 查询优化处理策略

Transform 查询可以表达成以下格式:

```
copy $a := T0 modify U($a) return Q($a)
```

其中 T_0 表示一棵 XML 实例树, 它可以是一个文档, 也可以文档中的某一片断, $\$a$ 表示对 T_0 的拷贝, $U(\$a)$ 表示对 $\$a$ 执行更新操作。可以将更新操作抽象地表示成以下形式[10]:

- (1) insert e into $\$a/p$
- (2) delete $\$a/p$
- (3) replace $\$a/p$ with e
- (4) rename $\$a/p$ as l

其中 p 为一个以 $\$a$ 为根结点的 XPath 表达式, l 为一 tag 名, e 为一个常量结点 (在实际查询中, 它也可能来自另外一个查询的结果)。

在后面的描述中, 用 $U(\$a)$ 表示 Transform 查询的更新操作, $Q(\$a)$ 称为 Transform 查询的目标查询, $Q(\$a)$ 的查询结果即为 Transform 查询的结果。根据前面的对

Transform 查询的描述, $U(\$a)$ 操作中更新的 path 可能不对 $Q(\$a)$ 的结果产生影响, 或者只影响 $Q(\$a)$ 的查询结果, 但不影响 $Q(\$a)$ 中的选择谓词。在此情况下, $Q(\$a)$ 中的选择谓词可以等价下推到 Q_c 中执行 (Q_c 为标准 XQuery 查询, 在此我们将 Q_c 的返回结果表示成 T_0)。

2.1 Transform 查询的等价转换

Transform 查询的等价转换是指: 对一个 Transform 查询 Q_t , 可以找到一个标准的 XQuery 语句 Q_s , 对任何被查询的文档 T , 使得 $Q_s(T) = Q_t(T)$ 。这时我们说可以将 Transform 查询 Q_t 等价转换成 Q_s 。当一个 Transform 查询 Q_t 等价于一个标准的 XQuery 查询 Q_s 时, 可通过执行 Q_s 从而避免执行 Q_t , 从而避免拷贝与更新操作, 减小查询代价。

2.2 Transform 查询的“lazy”处理策略

Transform 查询最直接的处理过程可描述为:“拷贝—更新—查询”, 即先拷贝 XML 文档的结点, 然后在被拷贝的数据上执行更新, 最后在此中间结果上执行查询。这种先拷贝再更新的方法称为“eager”方法, 符合 Transform 查询的语意, 实现简单, 但在执行过程中, 可能会拷贝与更新许多与查询结果无关的结点。

根据 Transform 查询的表达形式:“copy $\$a := T_0$ modify $U(\$a)$ return $Q(\$a)$ ”, “eager”方法的特点就是所有 $U(\$a)$ 的操作都在 $Q(\$a)$ 操作之前。 $Q(\$a)$ 中包含若干谓词: 值谓词 (限制查询结点的值满足的条件) 与结构谓词 (限制查询结点必须满足的结构)。对于一些 Transform 查询, $U(\$a)$ 操作不会影响 $Q(\$a)$ 中的谓词结果, 而 $U(\$a)$ 操作最终不会修改数据状态, 这样我们可以将 $Q(\$a)$ 中的谓词下推, 在 $U(\$a)$ 操作之前执行, 这样就可以提前过滤与查询结果无关的结点, 从而避免拷贝、更新这些与查询结果无关的结点。

如果 $Q(\$a)$ 上的所有谓词都能下推到 $U(\$a)$ 操作之前执行, 然后在 $U(\$a)$ 上直接构造查询结果, 我们称这种处理方法为 Transform 查询的“lazy”处理策略。采用“lazy”处理策略, 将 $Q(\$a)$ 中的谓词下推到 $U(\$a)$ 之前执行, 可以过滤许多查询与结果无关的结点, 从而减少拷贝与更新的代价。

2.3 Transform 查询的“混合”处理策略

并非所有查询都能采用上述两种方法, 对于这种情况, 选用“混合”方法处理 Transform 查询是个不错的选择, 即结合“eager”与“lazy”方法, 在保证查询结果的正确性的前提下将 $Q(\$a)$ 中的选择谓词下推, 先于 $U(\$a)$ 操作之前执行, 以减小拷贝与更新操作的代价, 我们称这种方法为“混合”处理策略。

3 Transform 查询处理方法的选择

3.1 相关概念的定义

基于代数的 XQuery/Update, 只有满足输入 pattern tree 的 XML 实例树才会执行更新操作或者作为查询结果返回。我们利用 pattern tree 来表示查询或者更新 XML 片断的结构。

1. Update Interesting pattern tree

我们可以对 pattern tree 进行模拟更新, 以反应 XML 实例树更新后的结构。为此, 对 pattern tree 上将被更新的结点设置相应的标志, 称为更新标记 (Update Flag(UF)), 来模拟对 pattern tree 进行更新。

更新标志 (UF) 的取值可以为以下几种:

- (1) #del: 结点将被删除
- (2) #ins: 新插入的结点
- (3) #ren: 结点将被重命名
- (4) #rep: 结点将被替换

值得注意的, 对于 rename,replace 操作, 更新前与更新后的结点都设置设置相应的 UF, 并将更新后的结点加入到 pattern tree 中相应的位置。

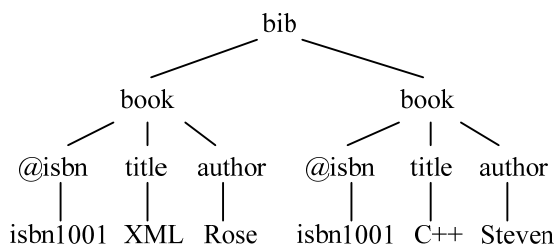


图 1 bib 文档结构

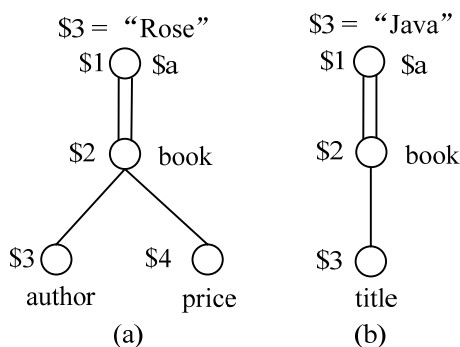


图 2 更新操作的 pattern tree

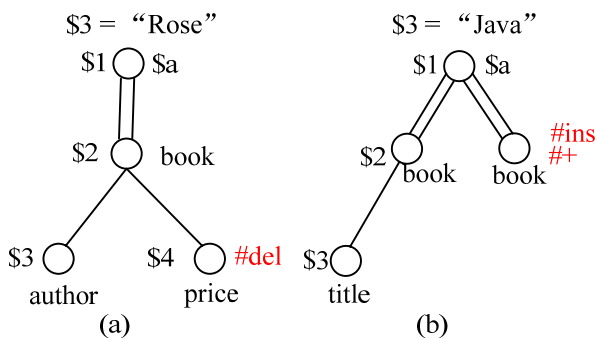


图 3 Update interesting pattern tree

在 pattern tree 中, 有些结点是叶子结点, 但在 XML 文档上对应的实例有可能不是叶子结点, 为了

在 Pattern tree 上反应这种关系, 我们对 pattern tree 上的结点设置其是否有后代结点标志 (Descendant Flag(DF)). DF 的取值有:

- (1) #+: 结点的实例不能确定是叶子结点;
- (2) #-: 结点的实例是叶子结点。

下面我们通过两个例子对 pattern tree 进行模拟更新操作, XML 文档为图 1 所示的 bib 文档。

- a. delete \$a//book[author = "Rose"]/price
- b. insert \$b//book after \$a//book[title = "Java"]

a,b 两个更新操作语句表示将被更新的 pattern tree 如图 2(a)、(b)所示。对图 2 中两个 pattern tree 进行相应的更新后, 并对被修改的结点设置相应的 UF, DF 后分别如图 3 (a)、(b)所示, 表示更新文档片断的模式结构称为 update interesting pattern tree.

定义 1: update interesting pattern tree (UIPT):

将更新操作作用于对应的输入 pattern tree 上, 并在更新的结点上设置相应的更新标志 (UF) 与实例结点是否有后代结点的标志(DF), 得到的即为 UIPT。

UIPT 的构建比较简单, 对每一个更新操作符, 把输入的 pattern tree 当成一个 XML 实例树进行更新操作 (但并不删除结点, 只是设置相应的更新标志, 如果是 Insert、Rename、Repalce, 更新后的结点将会被作为新的结点加入到 UIPT 中)。构建过程如下:

1) 对于 Delete 操作, 对 pattern tree 上将被删除的结点的 UF 设置为 "#del"。

2) 对于 Insert 操作, 直接将新来结点插入到 pattern tree, 并将新插入的结点的 UF 设置为 #Ins, 如果新插入的结点有后代结点, 所有的后代结点的 UF 都为 "#Ins", 新插入的结点中 (包括它的后代结点) 如果不是一个新构建结点, 则将其 DF 设置为 "#+"。

3) 对于 Rename 和 Replace 操作, 将被 Rename(Replace)的结点 a 的 UF 设置为 "#Ren" ("#Rep"), 对于 Rename 操作, 复制结点 a(包括其后代结点), 将 a 结点的 tag 名改为新的 tag 名, 并将复制的结点作为原结点 a 的右兄弟插入到 pattern tree 中 (复制结点的 UF 标志与源结点相同), 对 Repalce 操作也类似。

2. 路径间的关系

定义 2. 简单路径: 将 pattern tree (或 update interesting pattern tree) 上由根结点到叶子结点不带分支的路径称为简单路径。

例如如图 3(a)中的\$a//book/author 为简单路径, 而\$a//book[author = "Rose"]/price 则不是简单路径, 因为有分支 author 结点。

简单路径 P1、P2 间具有以下两种关系:

1) P₁跟P₂是完全独立关系: 当且仅当对任意 XML实例树T, P₁在T上的实例结点及其后代结点组成集合S₁, P₂在T上的实例结点及其后代结点组成的集合S₂, 对于S₁中的任意结点s₁, 在S₂中不存在结点s₂,

使得 $s_1=s_2$ ，那么称 P_1 跟 P_2 是完全独立关系。

2) P_1 是 P_2 的祖先路径：当且仅当对任意XML实例树 T ， P_1 在 T 上的实例结点集 S_1 ， P_2 在 T 上的实例结点集 S_2 ，对于 S_1 中的任意一结点 s_1 ，在 S_2 中不存在结点 s_2 ，使得 s_1 为 s_2 的后代结点，那么称 P_1 是 P_2 的祖先路径。

当 P_1 跟 P_2 是完全独立关系时 P_1 必是 P_2 的祖先路径。

3.2 Transform 查询处理策略的选择

下面将讨论对于一个Transform查询，如何选择合适优化策略。为了描述简单，假设Transform查询中 $U(\$a)$ 只有一个更新操作，即这个更新操作只用一个Update interesting tree即可表示，对于有多个更新操作只需要做相应的扩展。 $U(\$a)$ 操作表示成一个UIPT P_u ，其中 P_u 的根结点通常是一个拷贝变量。

对于 P_u ，称从其根结点到标志了UF结点的简单路径组成的路径集合 $S_{UP} = \{up_1, up_2, \dots, up_n\}$ 为更新路径集， up_i 的叶子结点是标记了UF的结点。

Transform查询中的 $Q(\$a)$ ，可以用若干pattern tree表示，pattern tree中有些结点是谓词结点，有些是作为查询输出的结点。我们把由拷贝变量到各个谓词叶子结点的简单路径组成的集合 $S_{PP} = \{pp_1, pp_2, \dots, pp_n\}$ 称为谓词路径集。把拷贝变量到将作为查询结果输出结点的简单路径组成的集合 $S_{OP} = \{op_1, op_2, \dots, op_n\}$ 称为输出路径集。

1) Transform 查询可以采用等价转换优化策略

两种策略可用于判断一个 Transform 查询：

$copy \$a := T_0 \text{ modify } U(\$a) \text{ return } Q(\$a)$ 是否可以采用等价转换策略，从而转换成一个标准的XQuery查询。(a) $U(\$a)$ 中的更新操作对 $Q(\$a)$ 中的查询结果是无意义的，即执行 $U(\$a)$ 并不改变查询结果，此时Copy操作与更新操作可以删除；(b) $U(\$a)$ 操作可以用XQuery查询进行替换或者转换成相应的选择谓词，这种情况只有当更新操作是Delete时才成立。

对第一种情况：(a) $U(\$a)$ 与拷贝操作可以直接删除，直接在 T_0 上执行 $Q(\$a)$ 当且仅当Transform查询的 S_{UP}, S_{OP}, S_{PP} 满足以下条件：

$$\forall up \in S_{UP}, \forall op \in S_{OP}, \forall pp \in S_{PP} : up \text{ 跟 } op \text{ 是完全独立关系 } \wedge up \text{ 跟 } pp \text{ 是完全独立关系}$$

证明： 因为对于 $U(\$a)$ 中的UIPT P_u ，执行更新操作修改的结点和 $Q(\$a)$ 中的所有 S_{PP} 与 S_{OP} 均是完全独立关系，更新修改的结点不会影响到查询的谓词与结果，此时更新操作对查询结果是无意义的，则 $U(\$a)$ 操作可以不执行。从而拷贝操作也可以删除。证毕。

对于第二种情况：(b) $U(\$a)$ 操作可以转换成一个等价的XQuery查询或者选择谓词，只有当 $U(\$a)$ 操作

是Delete时，才可以进行等价转换。此时，Transform查询的 S_{UP}, S_{OP}, S_{PP} 必须满足以下关系：

$$\forall up \in S_{up}, \forall op \in S_{op}, \forall pp \in S_{pp} : up \text{ 是 } op \text{ 的祖先路径 } \wedge up \text{ 跟 } pp \text{ 是完全独立的关系}$$

证明：此时的更新操作必须是 Delete 操作，对任一 up 是 op 祖先路径，若不执行 $U(\$a)$ 操作， up 在 $\$a$ 上结点若同时满足 $Q(\$a)$ ，则将会做为查询结果输出，但因为 $U(\$a)$ 是删除操作，则 up 在 $\$a$ 上结点满足 $U(\$a)$ 输入 pattern tree 的那些结点将会从查询结果中删除，则 up 在 $\$a$ 的结果集为 S ，而满足 $U(\$a)$ 输入 pattern tree 的集和为 S' ，因为是 delete 操作，则只需要构造一个等价查询使其在 $\$a$ 的结果集为 $S - S'$ ，从而 $Q(\$a)$ 在 $S - S'$ 上构造查询结果。

2) Transform 查询可以采用“Lazy”处理策略

当Transform查询不能通过等价转换变成一标准XQuery查询时，应尽可能采用Lazy方法，将 $Q(\$a)$ 中的所有选择谓词下推至拷贝操作之前执行，以减少拷贝与执行更新的代价。当Transform查询的 S_{UP}, S_{OP}, S_{PP} 满足以下关系时可以采用“Lazy”方法：

$$\forall up \in S_{up}, \forall pp \in S_{pp} : pp \text{ 跟 } up \text{ 是完全独立的关系}$$

证明：因为任一 $up \in S_{UP}$ ，任一 $pp \in S_{PP}$ ， up 跟 pp 是完全独立关系，那么执行 $U(\$a)$ 更新并不会影响到 pp 对应的结点，这个 pp 对应的谓词可以下推至 $U(\$a)$ 之前执行。

3) Transform 查询可以采用“混合”处理策略

当Transform查询不能采用前面两种方法进行优化处理时，则可以进一步判断是否可以采用“混合”处理策略进行优化。当Transform查询的 S_{UP}, S_{PP} 满足以下关系时可以采用“混合”优化策略处理：

$$\forall up \in S_{up}, \exists pp \in S_{pp} : pp \text{ 对 } up \text{ 是完全独立的关系}$$

证明：(略)，参考“Lazy”优化策略的证明。

4 Transform 查询优化过程

假设copy操作后有更新操作： $Uop_1, Uop_2, \dots, Uop_n$ ，这些更新操作符是有序的。依次生成他们的UIPT： $UIPT_1, UIPT_2, \dots, UIPT_n$ ；而在更新操作符之后的查询pattern tree中，所有以拷贝变量为根的谓词简单路径集为 S_{PP} ，以拷贝变量为根的输出结点简单路径集为 S_{OP} 。依次执行以下几步：

1) 依次从 $UIPT_1, UIPT_2, \dots, UIPT_n$ ，抽取相应的更新路径集 S_{UP_i} ，判断 S_{UP_i}, S_{OP}, S_{PP} 是否满足Transform查询的等价转换条件，如果满足，则更新操作符 Uop_i 采用Transform查询的等价转换方法处理，并删除对应的更新操作符；若不满足，则将 S_{UP} 添

- 加到集合 SS_{UP} 。
- 2) 若 SS_{UP} 为空, 则删除所有copy操作符, 转到 6
- 3) 对于 SS_{UP} , S_{PP} 判断其是否满足Transform 查询“Lazy”处理策略的条件, 如果满足, 则用“Lazy”处理策略改写查询计划, 转到 5
- 4) 对于 SS_{UP}, S_{PP} 判断它们是否满足Transform 查询“混合”优化处理策略, 如果满足则用“混合”优化策略改写查询计划。
- 5) 对 copy 操作符中的每个拷贝变量 $\$var$, 判断是否还存在更新操作符作用于它, 如果不存在, 则删除对 $\$var$ 的拷贝, 并修改 Transform 查询中 $Q(\$a)$ 部分与变量 $\$var$ 相关的部分。

6) 返回重写后的代数执行计划。

对于查询 1:

```
let $doc := doc("bib.xml")
return copy $ndoc := $doc
      modify delete $ndoc/book[publisher =
        "Addison-Wesley"]/price
      return $ndoc/book[author = "Rose"]
```

其相应的代数查询计划如图 4 所示, 优化后的查询计划如图 5 所示。

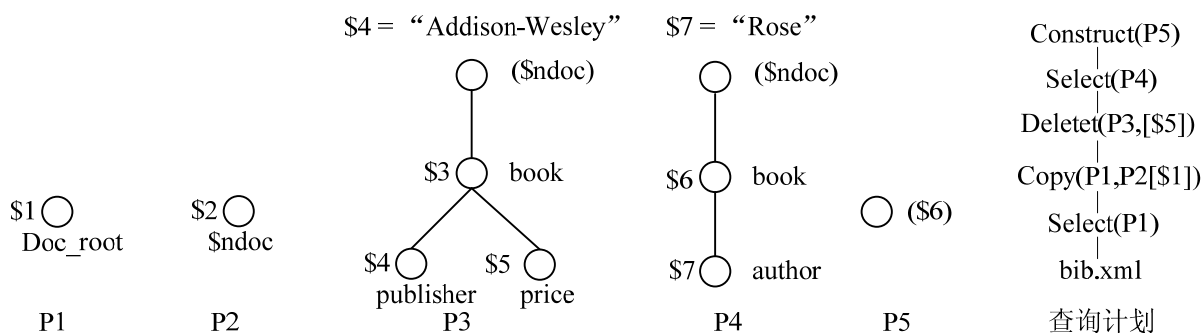


图 4 查询 1 的代数查询计划

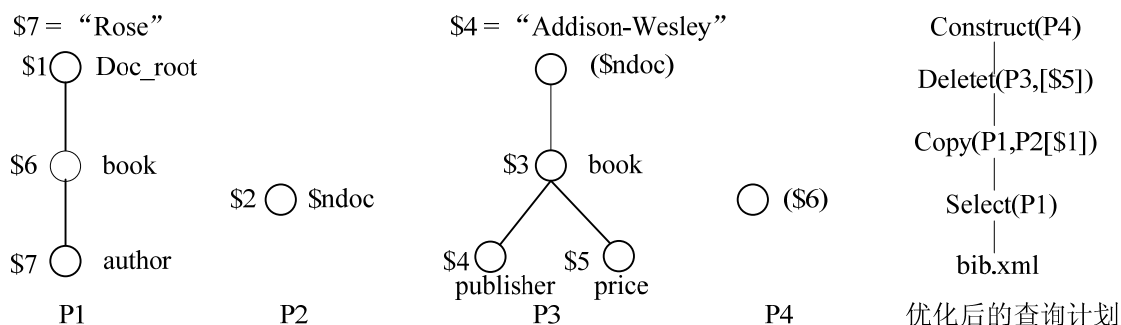


图 5 查询 1 优化后的查询计划

5 实验

实验基于 OrientX3.0[11], 实验系统平台为 P4 3.0G CPU, 512M RAM, Windows XP。实验采用的数据集为 XMark[12], 我们设计了 9 个 Transform 查询 Q1~Q9, 根据上文提出的优化规则, 可以将它们分为四组: G1 组{Q1,Q2}查询不满足优化条件、G2 组{Q3,Q4}满足等价转换条件、G3{Q5~Q7}符合“Lazy”方法处理、G4 组{Q8,Q9}符合“混合”处理方法条件。

评价指标为执行时间和拷贝结点的数量。

由图 6 可知, 除了 G1 组, 其它三组经过优化后, 执行时间上都低于未经优化的查询, 并且每一组的查询效率提高的程度大致相同。G1 组的两个查询 Q1,Q2 由于不符合优化规则, 其查询执行时间基本上是相同的。Q3~Q9 查询由于采用了优化策略, 优化后查询效率得到显著提高。

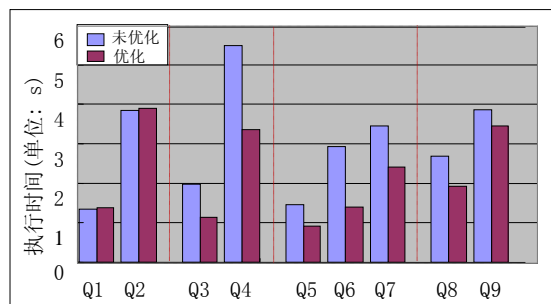


图 6 Q1~Q9 查询执行时间

图 7 表明除了 G1 组的两个查询拷贝的结点数目没有明显变化外, 其它三组查询中拷贝的结点数目都有很大程度的减少。G2 组查询 Q3、Q4 完全避免了结点的拷贝, G3 组查询满足“Lazy”方法处理条件, 可以将 $Q(\$a)$ 中的选择谓词下推至 $U(\$a)$ 之前执行, 从而可以过滤大量与查询结果无关的结点, 从而大幅度减少拷贝的结点数目。值得注意的是 Q5 和 Q8, 由于

优化过程中将一些满足条件的查询谓词提前到 copy 操作之前,而满足这个谓词的结点集合为空,从而使得拷贝的结点数为 0。

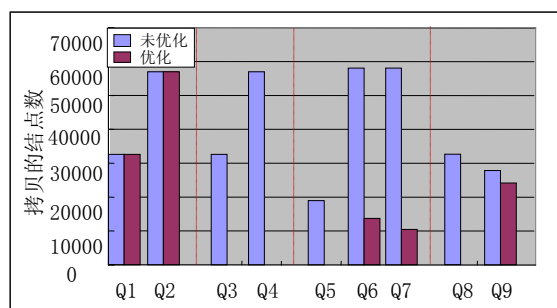


图 7 Q1-Q9 查询执行中拷贝的结点数

实验结果表明,由于我们分析了 Transform 查询中 U(\$a)与 Q(\$a)的关系,将 Q(\$a)查询谓词尽可能地推下执行,或者将 Transform 查询等价转换成一个标准的 XQuery 查询,从而提前过滤掉不是查询结果的结点,极大地减少了拷贝与更新操作的代价,使得 Transform 查询效率得到了较大程度的提高。

6 结论

本文提出了 Transform 查询的三种优化处理策略: Transform 查询等价转换, Transform 查询“Lazy”处理策略及“混合”策略,这三种处理策略适用于不同的 Transform 查询,本文给出了选择合适优化处理策略的规则与过程。并通过实验验证了这些策略均可有效提高 Transform 查询的效率与减少查询执行过程中的拷贝与更新操作的代价。

参 考 文 献

[1] Scott Boag, Don Chamberlin, Mary F. Fernández, et al. XQuery1.0: An XML Query Language[S].Boston:World Wide Web Consortium(W3C),2008. <http://www.w3.org/TR/xquery>

[2] Anders Berglund, Scott Boag, et al. XPath2.0[S].Boston:World Wide Web Consortium(W3C),2008.<http://www.w3.org/TR/xpath>

[3] Mary Fernández, et al. XQuery 1.0 and XPath 2.0 Data Model [S].Boston:World Wide Web Consortium(W3C),2008. <http://www.w3.org/TR/xpath-datamodel>

[4] Tim Bray, Jean Paoli, et al. Extensible Markup Language (XML) 1.1[S]. Boston:World Wide Web Consortium(W3C),2008. <http://www.w3.org/TR/XML>

[5] Meng Xiaofeng, Wang Yu, et al. OrientX: A Native XML Database System[C]// Proc of the 20th National Data Base conf(NDBC), Beijing:China Computer Federation,2003: 111-115
(孟小峰,王宇,罗道锋等. OrientX: 一个 Native XML 数据库系统的实现策略[C]//第 20 届全国数据库学术会议,北京:中国计算机学会,2003:111-115)

[6] Zhang Xin, Meng Xiaofeng, et al. OrientStore+:A native XML storage

strategy for efficient update[J].Journal of Computer Research and Development, 2007,44(Suppl):368-373(in China)

(张新, 孟小峰,等: OrientStore+: 一种支持高效更新的 Native XML 存储方法[J]. 计算机研究与发展,2007, 44(增刊): 368-373)

[7] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S.Weld. Updating XML[C]//Proc of the 2001 ACM SIGMOD conf. Santa Barbara : ACM, 2001:413-424

[8] Galax Project[DB]: <http://www.galaxquery.org/>, 2008-04

[9] MonetDB/XQuery project[DB]. Availble at <http://monetdb.cwi.nl/>. 2008-04

[10] Wenfei Fan, Gao Cong, Philip Bohannon: Querying XML with Update Syntax[C]. //Proc of the 2007 ACM SIGMOD conf. Beijing : ACM, 2007: 293 - 304

[11] A. Schmidt, F. Waas, M. Kersten, et al. XMark: A benchmark for XML data management[C]//Proc of the 28th Int'l conf on VLDB.San Francisco:Morgan Kaufmann, 2002:974-985

王 伟, 男, 1985 年生, 硕士研究生, 研究方向: XML 数据管理。

郭青松, 男, 1983 年生, 硕士研究生, 研究方向: XML 数据管理。

富丽贞, 女, 1982 年生, 博士研究生, 研究方向: XML 数据管理。

孟小峰, 男, 1964 年生, 教授, 研究领域: Web 数据集成, XML 数据库, 移动数据管理。