对象代理数据库的查询处理与优化*)

Query Processing and Optimization of an Object Deputy Database Management System

张广舟　彭智勇　肖静静　王　桢

(武汉大学计算机学院　软件工程国家重点实验室　武汉 430072)

Abstract Query processing and optimizing strategies in a Object-Deputy Database Management System (ODDBMS) are proposed in this paper. Various path expressions for different types of deputy classes are designed. The cost model for query plans of deputy classes is presented and several other problems in query optimization are studied. Finally, an prototype implementation demonstrates the efficiency of our approach.

Keywords Object deputy model, Query processing, Query optimization

1　引言

对象代理模型[1]是针对数据库中管理复杂数据对象的要求，扩充传统面向对象数据模型(OODM)形成的一种新的数据模型。近年来，出现了利用对象代理模型解决工作流[2]、信息集成等领域问题的思想，但这些研究都只停留在理论层次上，未涉及这种模型在实际数据库系统中的实现和性能评估。我们提出，将对象代理模型在实际数据库系统中实现，来定义高效率的半结构视图，方便对现实世界概念的建模，形成一种新型的数据库——对象代理数据库管理系统。为实现对象代理数据库，至少应解决三方面的问题：(1)对象代理数据库语言的定义；(2)支持新模型上的查询；(3)支持新模型的存储方式的实现。文[3]讨论了问题(1)，作为这一工作的继续，本文将重点讨论问题(2)，给出对象代理数据库数据查询处理的策略。

2　对象代理数据库系统

文[1]提出的对象代理模型引入了代理类的概念。基于这种模型的数据库系统中，存在两种数据结构：基本类对象和代理类对象。基本类与一般的对象数据库中的类或对象关系数据库中的关系的概念类似；代理类是在基本类基础上通过“代理规则”派生出的(如图 1)。代理类有以下特征：1)代理类通过代理规则代理一个或多个源类。代理规则是源类上的一个简单的 SQL 查询语句，每个代理对象对应一个由执行此查询语句得到的对象，并且每当源对象被更新时，代理规则将被重新执行，以判断该源对象是否继续被代理。2)代理对象由实属性和虚属性构成。实属性是代理对象的自有属性。虚属性的值由源对象的一个或多个属性的值，经过一个表达式计算(我们称之为“切换”)得出，而代理对象本身并不存放这些属性的值。虚属性的表达式记录了取源对象的属性的路径和计算方式，称为“切换表达式”(switch expression)。3)每个代理对象都有指针指向对应的源类中的对象，每个源对象亦有指向其代理对象的指针。

根据代理规则的 SQL 语句类型的不同，代理类被分为四种类型：select 型，join 型，union 型和 group 型。它们的具体定义参见文[3]。我们允许在代理类上创建新的代理类，于是形成了一种代理关系的层次结构(如图 2)。

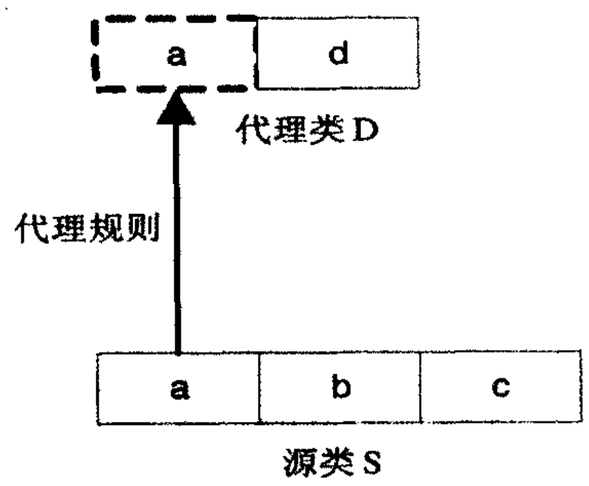

图 1　代理类示例

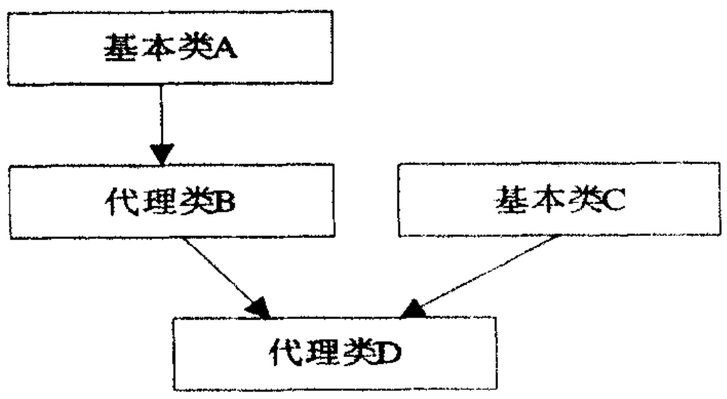

图 2　多层次代理实例

*)该研究得到国家自然科学基金(60273072, 60473076)，湖北省青年杰出人才基金项目(2002AC003)和教育部新世纪优秀人才计划项目(NCET-04-0675)资助。

图 2 中，B 代理了基本类 A，而 D 代理了 B 和 C（通过 join 或 union 型代理）。为了支持这种多层次的代理结构，完成复杂的切换操作，必须设计方便执行的切换表达式，作为系统元数据存储起来。我们发现，不论代理层次的类型和深度如何，切换表达式总是可以表示成一种表达式树，这种树的根节点是路径表达式或常数。例如，如果 A 的属性 v2 加上 3 得到 B 的虚属性 v1，而 B 的虚属性 v1 加上 2 得到 D 中的虚属性 v，则虚属性 D. v 的切换表达式可表示为：(D→B→A. v2 + 3) + 2，其中，D→B→A. v2 被称为“路径表达式”，数据库应能根据这种表达式和代理对象的源对象指针，取出被继承的源类属性 A. v2。需要说明的是，切换表达式中的路径表达式的计算方式却随代理类型的不同而不同，我们将在下一节具体介绍。

代理对象与源对象之间的指针可以采用地址映射表实现。每个映射表表项记录了对象的对象标识符(OID)、该对象的物理地址、该对象的相邻层次的代理对象 OID 和源对象 OID。若采用这种方式，在执行切换操作时，将根据对象的对象标识符，查找其对应的映射表项，得到它的源对象的 OID，即可查出源对象的物理地址。这种方式易于更新指针关系，且在多层次代理情况下，不必访问中间层的源类存在磁盘的物理对象，而只须多次查找映射表，即可得到最上层源类对象的物理指针，完成切换操作。在后面的讨论中，我们假设对象指针都是通过映射表实现的。

与现在流行的对象关系数据库相比，对象代理数据库在以下方面提高了性能：

1. 可以用代理类实现高效视图。对象关系数据库的视图，是一个虚表：数据库中只存放视图的定义，而不存放视图对应的数据。因此，定义视图并不能提高系统的效率。为了提高视图的查询效率，物化视图被提出。但一般的物化视图会产生数据冗余，并且当基本表被更新后，必须重新生成与它有关的所有物化视图，才能保证正确性。加入对象代理模型后，可以用代理类实现视图。代理类可以看作一种半结构化视图，它没有保存实际的属性值，通过指针存取真正的属性值，并且具备自动更新功能。这样，既可以提高效率，避免了冗余，又有效解决了更新问题。

2. 对象代理机制使数据库更易于建模。对象代理机制可以用来对已有对象进行分解，组合和扩充，增加了对象的柔软性，使数据库的建模更加方便。

3 对象代理数据库的查询处理

我们采用文[3]中提出的语言作为对象代理数据库的查询语言，它与 SQL92/99 类似。根据这种语言的定义，在对象代理数据库中，代理类可以像基本类一样出现在查询语句中。在用户看来，对代理类的查询与基本类没有区别，可以对代理类作投影、连接操作，或与其他类(代理类或基本类)进行连接，但在查询处理过程中，需要针对代理类作特殊处理，尤其是根据代理类的类型正确处理切换表达式和路径表达式。

3.1 查询处理的基本流程

对象代理数据库的查询流程，与一般数据库一样包括查询编译、查询优化和查询执行阶段。

在查询编译阶段，如果查询语句中出现了代理类和对虚属性的引用，查询编译器将通过查元数据，找出这些虚属性的切换表达式，放入查询树中，供查询优化器和执行器使用。此时，虚属性上的选择条件形成了一种以路径表达式和常数为叶节点的表达式树。例如，对于上节中提到的代理类 D，其虚属性 v 上的选择条件表达式树如图 3。

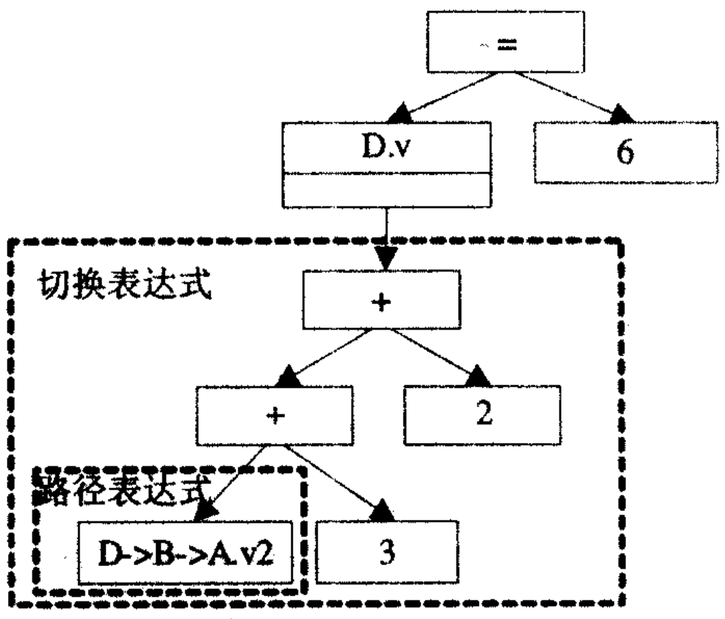

图 3 虚属性选择条件的表达式树

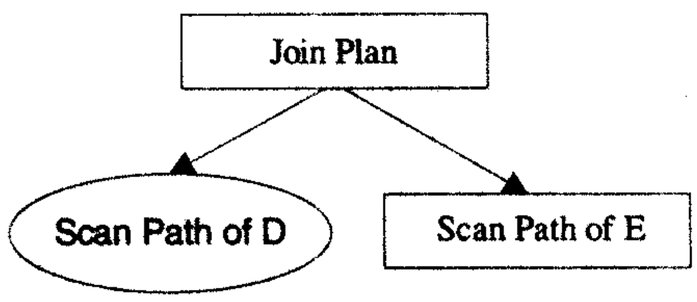

图 4 涉及代理类 D 与基本类 E 的查询计划树

查询优化阶段，由于代理类对象的属性并非像基本类那样，完全按照其模式存储，因此要针对代理类，生成特殊的(不同于基本类的)访问路径(即扫描代理类的计划)，以便执行器顺利执行(如图 4)。对于查询优化涉及的几个具体问题，将在下节介绍。

查询执行器的任务是执行整个查询计划树，并返回结果。当查询语句涉及代理类时，查询树的叶节点将可能是代理类的访问路径。下面我们讨论代理类访问路径的实现方式。我们定义了两种类似基

本类的访问路径。

一种是不利用索引的基本访问路径。这种路径类似于普通类的顺序扫描，但不同于普通类，因为代理类存于磁盘的物理对象只存储实属性值，故我们不能通过普通的顺序扫描，得到完整的(包含虚属性的)代理对象；而要在得到物理对象后，计算切换表达式才能得到虚属性的值。

另一种路径是使用索引的路径。若代理类上建有索引，可考虑利用索引进行存取。这种路径是在访问代理类的对象前，先访问它的索引，根据符合约束条件的对象的指针，取出代理类的物理对象，进而按与上述基本路径类似的方式生成虚属性及进行投影操作。

如果执行器中支持上述两种代理类访问路径，则在查询优化阶段应为代理类生成这两种路径，并计算出各自代价供选择。

3.2 路径表达式的计算

计算切换表达式的过程，关键是路径表达式的计算。第 2 节提到，切换表达式中的路径表达式的计算方法随代理类型的不同而不同，下面分别进行介绍。

select 和 join 型代理的路径表达式具有相同的计算方法。如计算 D→A.v 时，先取出 D 的实属性，再根据对象的 OID，查出其映射表项，从其源类 OID 中找出属于 A 类的源对象 OID，并通过此 OID 查映射表得到其物理地址，最后取出源对象和属性 v。

union 型的代理类的源对象可能来自于不同的类，因此，会有多个切换表达式，而对于每个具体的对象，只有其中一个是合法的。所以，需要针对每个代理对象选择其合适的切换表达式。为此，我们采用“试取”的方式进行选择：依次根据各切换表达式的路径表达式，试取源对象，如果可以取到，则说明此表达式是合法的切换表达式。

对于 group 型代理类，每个代理对象代理一组对象，它记录了组中每个源对象的 OID 和物理地址。它的虚属性有些是通过聚合函数(如 SUM()，AVG ())定义的，执行时，需要将组中的源对象都取出来，来计算聚合函数的值。

4 对象代理数据库的查询优化

代理类的引入要求优化器进行特殊的优化操作。本节讨论对代理类访问路径优化的问题，涉及虚属性选择条件的选择因子，选择优化和路径表达式的代价计算等。

4.1 虚属性选择条件的选择因子

为了计算查询计划的代价，查询优化器需要估计各种选择和连接条件的“选择因子(selectivity factor)”，也就是使此表达式值为真的对象占类的所有对象的比例。为此，优化器需要从系统中取出类的统计信息(statistics)，包括记录类的属性值分布情况的直方图(histogram)信息。

在对象代理数据库中，代理类的某些属性是虚属性，采集这些属性的直方图信息会比较复杂。我们发现，在某些情况下，可以直接利用源类对应的属性的统计信息，计算虚属性上的条件表达式的选择因子。例如，如果代理类 D 的属性 v 直接继承了源类 A 的属性 v1，且代理规则没有对 v1 属性值的分布造成影响，则可以直接利用 v1 上的直方图信息计算虚属性 v 的选择因子。具体地说，假设有一个在代理类 D 上的条件表达式 e(D.v)，如果下列条件满足：

(1)D 的代理规则中，不涉及源类的连接、合并，聚集等操作，即它是 select 型代理类；

(2)代理规则中的谓词条件中不涉及属性 A.v1；

(3)D.v 直接继承至 A.v1，即继承时切换表达式没有＋、－等表达式计算；

则设 F()是利用属性的统计信息计算选择因子的函数，我们有下面的公式：

$$F(e(D.v))=F(e(A.v1)) \quad (1)$$

上述公式是合理的。考虑一下关系数据库中计算选择因子时的情况。我们总是假设不同属性上的选择和连接条件相互独立，不影响彼此值的分布情况，所以可以将它们的选择因子相乘，得到总的选择因子。上面提出的条件(2)使虚属性所继承的属性和代理规则中的约束条件涉及的属性不同，从而保证了独立性，属性值的分布情况不被影响；条件(1)、(3)保证了源类属性被继承时值没有变化；因而，公式(1)成立。

4.2 选择优化问题

在一般的关系数据库中，计算选择条件的选择操作被认为不需磁盘 IO，并且只需很少的 CPU 周期，其代价对总的执行代价影响很小。但在对代理类的选择操作中，生成虚属性的切换操作可能需要访问磁盘，从而使选择操作有很大开销。这使得代理类计划的代价与选择条件的计算次序相关。

例如，对某一代理类 D 有一个查询，其中 D 有两个查询条件：D.v1＜10∧D.v2＝3，其中 v1，v2 是 D 上的虚属性，且来自不同的源类。这两个条件的切换操作的可能代价不同；另一方面，第一个条件的选择因子将影响计算第二个选择条件的机会。所以代价最小的计算次序与它们的代价和选择因子的有关。因此，为了得到较优的查询计划，需要慎重考虑如何对选择条件进行排序，也就是要解决“选择优化”问题。

其实，支持抽象数据类型(ADT)的对象关系数据库也有类似问题。带有 ADT 方法的条件计算需要很大开销，所以要考虑对选择条件的排序。一种做法是，将选择条件基于递增的 rank 值排序，其中，对每个条件，rank=(选择因子−1)/代价。

与之不同的是，在计算代理类虚属性上的各个条件时，各虚属性值可能来自同一源对象，从而使这个源对象被引用多次。由于系统的缓冲机制，被第二次引用的同一源对象，已经在缓冲区了，因此，取源对象时无需 IO 操作了，代价随之降低。为了考虑这种影响，使代价估计较为准确，我们采用一种经过修改的基于 rank 值对选择条件进行排序的方法。

假设代理类有 n 个选择条件，可按如下方法排序：

先计算所有的条件的 rank 值，并将它们按 rank 值递增次序排列；然后假设排在第一的选择条件涉及的源类对象已取入内存，重新计算第 2 至 n 个条件的 rank，并根据 rank 值对第 2 至 n 个条件重新排序；类似地，再假设排在前两位条件涉及的源类对象已取出，对后面的条件重新排序；依次类推，即可得到一个这些条件的排列。

4.3 代理类的路径的代价

代价计算的准确性对优化器的性能至关重要，这就要求对象代理数据库的优化器准确估计代理类访问路径的代价，并根据代价大小从中选择一个放入最终的计划树。代理类路径的代价计算与一般的关系最大的不同是，要计算执行路径表达式所需 IO 次数。下面我们讨论路径表达式代价的计算方法。

路径表达式的执行过程，主要是通过代理对象的 OID，通过查映射表，得到源对象的指针，再去磁盘中取出源对象的过程。而映射表一般常驻内存，查一次映射表的代价是一定的，故 IO 代价花费在用指针从磁盘取源对象上。另一方面，观察如图 5 所示的索引结构和图 6 所示的代理类的结构，可以发现，代理类的指针指向的源类所有对象中的全部或一部分；代理类的指针类似于全部索引指针的一部分。顺序访问代理类、取出所有源对象的过程，与对象关系数据库中用某些键值匹配对应索引项后，用索引指针取对应元组类似。也就是说，可以将代理类看作匹配了某些键值的部分索引，因此，其所需的 IO 次数，可以采用相似的方法计算。

为计算利用索引中的指针，取元组所需的 IO 次数，文[4]提出了一个经典的估计公式，被广泛用于计算在使用 LRU 方式管理缓存的数据库中计算索引计划的代价。现假设我们的系统采用 LRU 缓存管理方式，参照文[4]中的公式，可以得出路径表达式的代价公式，具体方法如下。

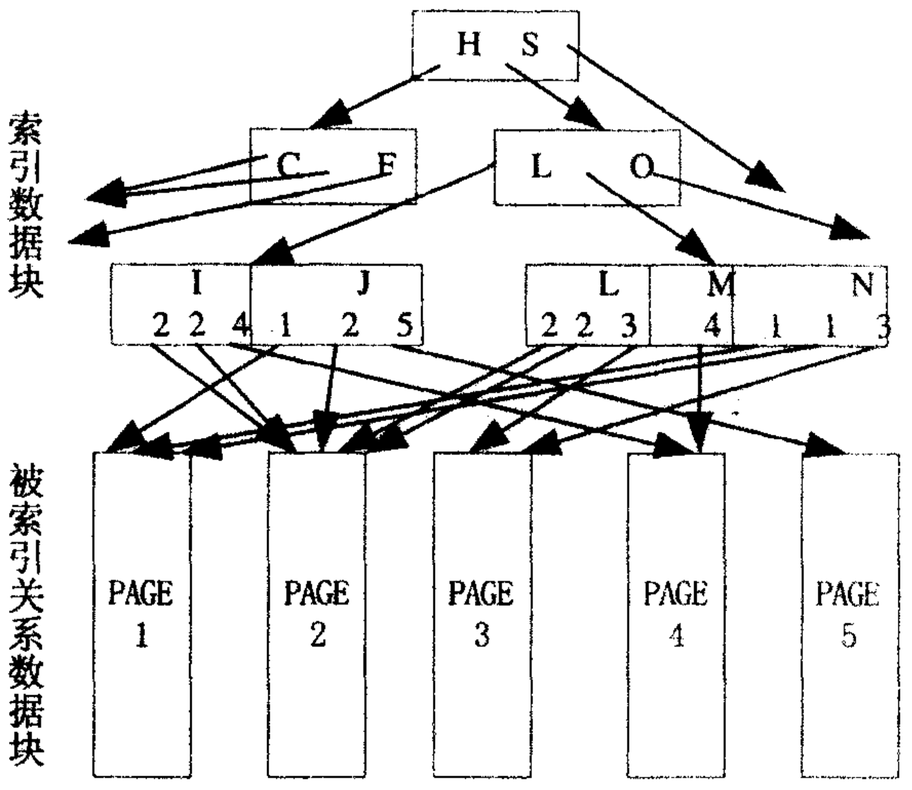

图 5 索引的指针结构

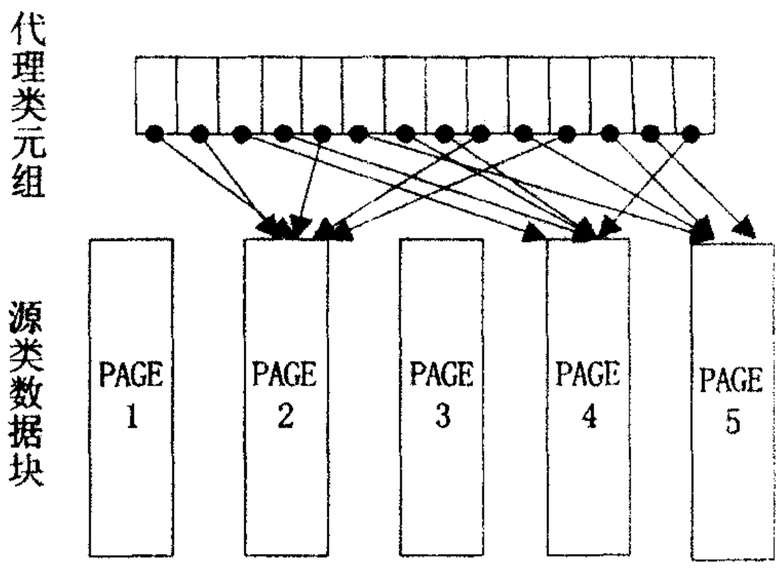

图 6 代理类指向源类对象的指针

文[4]中的估计公式 Y_{APP} 等价于下面的形式：

利用一个非聚簇索引扫描关系时，(访问被索引关系)所需 IO 次数

$$PF=\min(2TDx/(2T+Dx),T),$$

当 $T<=b$；

$2TDx/(2T+Dx)$，

当 $T>b$ 且 $Dx<=2Tb/(2T-b)$；

$b+(Dx-2Tb/(2T-b))\times(T-b)/T$，

当 $T>b$ 且 $Dx>2Tb/(2T-b)$；

其中，T＝被索引关系的对象(元组)数；N＝被索引关系的对象(元组)所磁盘块数；D＝平均每个索引键值的对应的被索引对象(元组)数；x＝表示匹配的键值的个数；b ＝ 系统的缓冲区(buffer)的数量。

假设代理对象的指针地址是均匀分布的，则可以将代理类中指向源类对象的指针，设想为源类上的一个非聚簇密集索引的指针中的一部分(注意，上述公式是针对密集索引的；而代理类因为可能只代理一部分源类对象，其指针只指向源类所有对象中的一部分)。假设所有代理对象共对应了 x 个键值，平均每个键值对应 D 个指针，则 Dx 值正好为代理

(下转第 120 页)

ACM International Workshop on Grid Computing (GRID'04), 2004
9 Kwon O K, Hahm J, Kim S, Lee J. GRASP—a grid resource allocation system based on OGSA. In: IEEE Intl. Symposium on High Performance Distributed Computing (HPDC'04), 2004
10 OGSA(Open Grid Services Architecture) Documents: http://www.globus.org/ogsa
11 Foster I, Kesselman C. The globus project: A status report. In: Proc. The Heterogeneous Computing Workshop, IEEE Press. 1998. 4～18
12 http://www-unix.globus.org/toolkit/ downloads/3.2/
13 Ester M, Kriegel H -P, Sander J, Xu X. A Density-Dased Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining(KDD'96), Portland, OR, Aug. 1996. 226～231

(上接第 100 页)

对象数。对代理类的顺序扫描取源类对象,相当于在索引扫描时正好匹配了这 x 个键值。T、N、Dx 的值对应如下:(D 表示代理类,S 表示源类)T=T(S),S 的对象数,N=N(S),S 占的磁盘块数 Dx=T(D),T(D)表示 D 的对象数。代入上面的公式,即可得取出所有代理类 D 的对象的源对象所需的 IO 次数的值,即路径表达式的 IO 代价的计算公式:

PF=min(2T P/(2T+ P),T),
当 T<=b;
2T P /(2T+ P),
当 T>b 且 T(D)<=2Tb/(2T−b);
b+(P−2Tb/(2T−b)) * (T−b)/T,
当 T>b 且 P>2Tb/(2T−b);

其中,T=T(S),S 的对象数,
N=N(S),S 占的磁盘块数
P=Dx=T(D),D 的对象数
b=系统的缓冲区(buffer)的数量

在用这个公式计算查询路径中所含所有路径表达式的代价后,便可以按照一般的代价模型计算整个的查询计划的代价了。

5 实验结果

我们实现了基于对象代理模型的数据库。为说明我们提出的查询处理的方法的可行性和有效性,我们在对象代理数据库和 PostgreSQL 数据库中分别定义了三个类,并分别用视图和代理类实现一个连接这三个类的查询,响应时间与其中一个源类大小变化的关系如图 7(我们的实验配置是 Intel CPU P4 2.0G,内存 512M,操作系统 Red Hat Linux 9.0)。可以看出用代理类实现视图使查询效率大大提高,而且源类越大效率提高越明显。这是因为,用视图实现此查询时,每次查询此视图,视图在查询处理时被重写,查询变成了对三个类的进行连接,从而耗费较长的时间;而使用代理类实现时,每次查询时可以通过访问代理类,通过属性切换直接得到结果,避免了类的连接操作,耗费的时间大大减少;另外,随着源类的对象数的增加,造成连接三个类时所需的时间增长很快;而查询结果增长相对较慢,使对应的代理类大小变化不大,从而代理类的查询的时间变化不大,因此,源类越大时,使用代理类时的效率提高越明显。

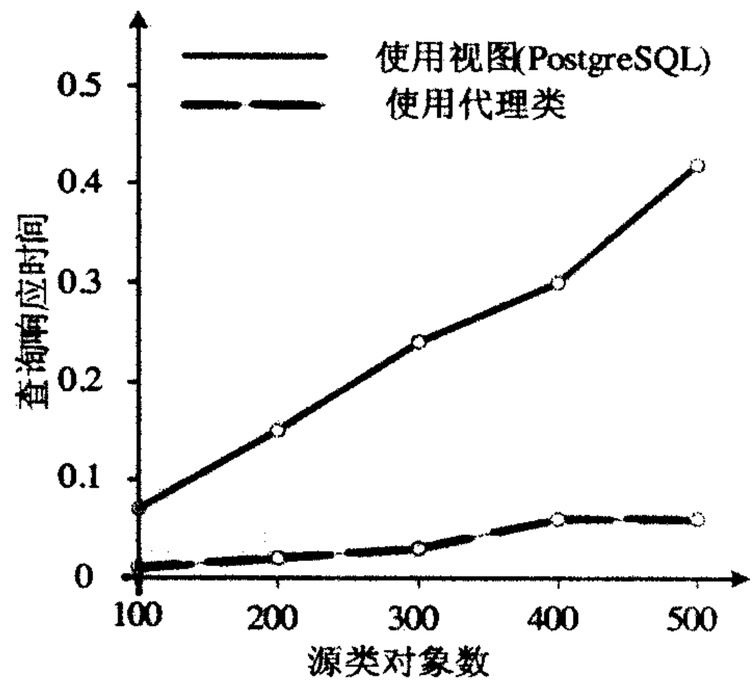

图 7 使用代理实现视图的效率比较

结束语 本文讨论了在对象代理数据库中,处理涉及代理类查询的过程,以及与代理类相关的特殊的查询优化问题。根据本文提出的实现策略,可以很方便地实现对象代理模型数据的查询处理,使其成为对象代理数据库。我们的实践表明,这种对象代理数据库,在性能上比传统的对象关系数据库有很大改进。

参 考 文 献

1 Peng Zhiyong, Kambayashi Y. Deputy Mechanisms for Object-Oriental Databases. In: Proc. of IEEE 11th ICDE, 1995. 333～340
2 Shan Zhe, Long Zhiy, Luo Yi, Peng Zhiyong. Object-oriented realization of work-flow views for web services - an object deputy model based approach. In: The Fifth Intl. Conf. on Web Age Information Management, WAIM 2004, LNCS 3129. Springer-Verlag, 2004
3 翟博馥, 彭智勇, 庄继峰. 对象代理数据库语言. 见:第 21 届全国数据库学术会议. 厦门,2004
4 Marckert L F, Lohman G M. Index Scans Using a Finite LRU Buffer: A Validated I/O Model. ACM Transactions on Database Systems, 1989, 14(3): 401～424