对象代理数据库的双向指针存储优化方法

胡聪睿[1),2)] 刘 斌[1),2)] 冯 岭[3)] 王 飞[1),2)] 彭智勇[1),2)]

[1)](武汉大学软件工程国家重点实验室 武汉 430072)

[2)](武汉大学计算机学院 武汉 430072)

[3)](华北水利水电大学信息工程学院 郑州 450001)

摘 要 在对象代理数据库中,提出了两个新的概念:代理对象和代理类.一个对象可以有一个或多个代理对象,该对象也被称为代理对象的源对象.代理对象可以从其源对象中选择性继承属性和方法,也可以扩展定义代理类自己的属性和方法.代理对象继承自源对象的属性称为虚属性,虚属性不实际存储其值,而是在查询时根据源对象的实属性值计算得到.对象代理数据库利用双向指针表来实现源对象和代理对象之间的这种关联,双向指针表中每一条记录包含两个字段,分别对应于源对象 ID 和代理对象 ID.当存在多层代理关系时,查询代理对象需要对双向指针表执行多次查询操作,因而查询效率往往不是很好.文中针对代理对象查询效率低的问题进行了深入研究,发现其主要原因是双向指针表中具有相同代理类和源类的数据分布在不同的磁盘块中,查询双向指针表时需要多次 I/O 操作.基于此,文中首先设计新的对象代理数据库中空闲空间分配策略,在原有的空闲空间管理策略中加入新的索引信息,使得当需要添加记录时不仅使用空闲空间管理模块 FSM 查找空闲空间大小一个条件,而且结合使用索引中指定的聚簇信息寻找具有合适空闲空间的磁盘块,然后读取目标磁盘块到内存中,并将双向指针表中的元组以源类和代理类为单位进行分类写入磁盘块,从而实现基于代理关系的双向指针表的聚簇存储.同时针对具体的应用场景,文中根据数据库模式中代理层次信息、用户查询频率等因素设计一个 I/O 查询代价模型,利用该模型计算各级代理类与源类聚簇之后的查询代价,从而选择出一种查询代价最小,性能最优的聚簇策略,使得能够最大限度的减少 I/O 次数,提高代理对象的查询效率.实验结果表明,该聚簇方法在真实数据集上的优化效果比未优化平均提高 15%以上,比已有的双向指针表的优化方法平均提高 9%以上.

关键词 对象代理数据库;双向指针表;虚属性查询;存储优化

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2018.01752

Optimization of Bilateral Pointer Storage in Object Deputy Database

HU Cong-Rui[1),2)] LIU Bin[1),2)] FENG Ling[3)] WANG Fei[1),2)] PENG Zhi-Yong[1),2)]

[1)](*State Key of Laboratory of Software Engineering, Wuhan University, Wuhan* 430072)

[2)](*Computer School, Wuhan University, Wuhan* 430072)

[3)](*School of Information Engineering, North China University of Water Resources and Electric Power, Zhengzhou* 450001)

Abstract In Object Deputy Database, two new concepts are proposed: deputy object and deputy class. Each object uses a unique identity, multiple attributes, and multiple methods to describe its characteristics and behavior. Objects with common attributes and methods are clustered together and their schema is defined as a class. An object can have one or more deputy objects, and this object can also be called source object for these deputy objects. The deputy object can also have its own deputy objects. Deputy objects can selectively inherit attributes and methods from their source objects, or define their own attributes and methods. Attributes of a deputy

收稿日期:2016-11-20;在线出版日期:2017-04-20.本课题得到国家重点研发计划(2016YFB1000701)、国家自然科学基金(61232002)、湖北省科技支撑计划基金(2015BAA127)资助.**胡聪睿**,男,1991 年生,硕士研究生,主要研究方向为数据管理、数据挖掘.E-mail:hucongrui1991@163.com.**刘 斌**(通信作者),男,1975 年生,博士,讲师,主要研究方向为数据管理、数据挖掘.E-mail:binliu@whu.edu.cn.**冯 岭**,男,1986 年生,博士,讲师,主要研究方向为专利分析与挖掘.**王 飞**,男,1989 年生,博士研究生,主要研究方向为数据挖掘、信息检索.**彭智勇**,男,1963 年生,博士,教授,博士生导师,主要研究领域为复杂数据管理、Web 数据管理、可信数据管理.

object inheriting from its source object are called virtual attributes, which are not actually stored but obtained value according to that of the source object in the query. The Object Deputy Database uses a bilateral pointer table to manage the associations between source objects and deputy objects. Each record in the bilateral pointer table contains two fields, corresponding to the source object ID and the deputy object ID. When we get the virtual attribute value, we first find the source object of the deputy object according to the bilateral pointer table and get the real attribute value from the source object, and then calculate the virtual attribute value according to the switching expression and the real attribute value. When there are multilayer deputy relationships, multiple query operations are needed to perform on the bilateral pointer table to query deputy objects resulting in low query efficiency. In this paper, we have studied on the problem of low query efficiency and have found that the main reason is that the data of the same deputy class and source class are distributed in different disk blocks, thus resulting in multiple I/Os when querying the bilateral pointer table. Based on this, this paper first designs a blank space allocation strategy in the Object Deputy Database, which add new index information to the original free space management strategy. Therefore, when it needs to add new record, it can not only use the free space management module called FSM to find the free space with suitable size, but also should find the suitable disk block according to the clustering information specified in index. Then it reads the target disk block into memory, and tuples in the bilateral pointer table are written to disk as a unit of source classes and deputy classes to realize the clustering storage of the bilateral pointer table based on the deputy relation. At the same time, this paper designs an I/O query cost model considering the factors of deputy hierarchy and user query frequency in the database model aiming at the specific application scenario. The model is used to calculate the query cost after deputy class and source class clustering to select a clustering strategy with least query cost and optimal performance, and objects in the bilateral pointer table are also clustering stored as a unit of source classes and deputy classes in this strategy, which can make the Object Deputy Database reduce the number of I/O and maximize the query efficiency of deputy objects. The experimental results show that the clustering method improve more than 15 percent of the optimization results on the real data sets than before, compared with the previous study on the optimization of the bilateral pointer table increased by more than 9 percent.

Keywords　object deputy database; bilateral pointer table; virtual attribute query; storage optimization

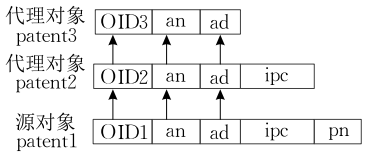
1　引　言

对象代理模型[1-3]是一种基于面向对象模型和关系模型提出的新的数据模型，其既具有处理传统关系型数据的能力，又具有处理复杂数据的能力. 对象代理模型扩展了传统的对象模型(OO Model)[4]，提出了两个新的概念：代理对象和代理类[5]. 一个对象可以有一个或多个代理对象(deputy object)，该对象也被称为代理对象的源对象(source object). 代理对象又可以拥有自己的代理对象. 代理对象可以通过切换操作从其源对象中选择性继承属性和方法，也可以扩展定义代理类自己的属性和方法. 代理对象继承自源对象的属性并不实际存储其值.

对象代理数据库 TOTEM 是一个基于对象代理(OD)数据模型的数据库系统[6-11]. TOTEM 对象代理数据库中定义了特有的结构：双向指针. TOTEM 数据库依据双向指针可以获取代理对象虚属性的值，双向指针记录了代理对象和源对象之间的关联关系，通过双向指针，代理对象可以查找其源对象，源对象同样可以查找其代理对象. 对象代理数据库 TOTEM 用双向指针表来存储双向指针，双向指针表包含两个属性：源对象 OID(SourceObjectOID)和

代理对象 OID(DeputyObjectOID);查询代理对象虚属性时,首先根据双向指针表查找到源对象,获取源对象的实属性值,然后依据实属性值计算得到代理对象虚属性的值.

图 1 刻画了一个两层代理情况下,包含源对象 patent1 和代理对象 patent2、patent3,对象代理数据库 TOTEM 虚属性的查询过程[12-15]:首先从 TOTEM 对象代理数据库的系统表中获得代理对象 patent3 的元信息,判断 an 是否为虚属性,patent3.an 是虚属性,因此需要读取双向指针表,查找双向指针表中的 patent3 的源对象,找到 patent3 的源对象 patent2 以及 patent3.an 对应的源属性 patent2.an,同样判断 patent2.an 是否为虚属性;patent2.an 同样为虚属性,继续寻找虚属性 patent2.an 对应的源属性 patent1.an,判断 patent1.an 为实属性;然后读取 patent1.an 的值,结合系统表中的 patent3.an 对应的切换表达式,并根据代理对象中 patent1.an 的值得出 patent3.an 的值.最后判断虚属性是否满足选择条件或直接输出.从上述例子可以看出,对象代理数据库 TOTEM 在进行虚属性查询过程中,当代理层次过多时,为了计算虚属性的值有可能多次读取磁盘,使得磁盘 I/O 次数增多,降低了虚属性查询效率,因此需要优化 OID 的存储.



(a) 对象与代理对象的关系

源对象OID	代理对象OID
OID1	OID2
OID2	OID3

(b) 双向指针表

图 1 对象代理数据库相关结构

在数据库存储优化方面[16],国内外也有很多相关的研究.数据库集群自动物理性分区[17]:可以在不中断数据交易的情况下在多个节点之间动态分配数据,提高各个节点的利用率以及均衡性,从而提高数据的查询性能.Oracle 内存数据库的分布式架构[18]:提出了一种高可用的具有容错性的分布式架构,能够使关系型数据库在数据库集群中内存容量和查询处理方面得到显式的拓展.大数据查询问题[19]:开发了一个具有查询处理模式和数据存储模型的系统,可以使 1.2 TB 的数据集在一台服务器级别的机器上不到三分钟的时间内得到查询结果,并且保持了良好的查询处理性能.基于列式存储技术的内存数据库引擎[20]:提出了一种基于列式存储和内存技术的数据库引擎,不需要部署列式数据库,能够在短时间内从行存储数据库中快速加载大数据,并返回查询结果.混合存储架构[21]:结合行存储和列存储两种存储架构更好地提高数据库的查询和更新性能.混合行列划分[22]:提出了一种基于数据分片的混合行列存储方案,无论是行、列或组合分片均由底层的文件系统存储层以相同的方式存储和处理,提高数据的查询性能.kudu 存储引擎①:一种支持低延迟随机存取和有效分析访问模式的一个开源的结构化数据存储引擎,兼顾了数据更新实时性和分析速度.基于一致性复制的可扩展的行存储[23]:提出了一种用于更新可扩展行存储的方法,包括接收对数据表中键的更新以及使用基于共识的复制算法在一组节点上复制更新.增强 SQL Server 列存储功能[24]:SQL Server 2012 针对数据仓库工作负载提出了两个创新:列存储索引和批处理模式,新的索引类型结合新的查询运算符大大提高了数据仓库的查询性能和决策支持的查询速度.可扩展的 SQL 和 NoSQL 的数据存储[25]:研究了一种旨在扩展多服务器上应用程序加载的 SQL 和 NoSQL 的数据存储,牺牲了数据库事务一致性以实现更好的可用性和扩展性.主内存混合存储引擎 HYRISE[26]:提出了一种主存储器混合数据库系统,根据数据表中列的访问方式,自动将表分为不同宽度的垂直分区,HYRISE 能够预测不同分区的性能,并使用数据库设计算法自动选择最佳分区.相对于全行或全列设计模型,性能得到较大提升,并且具有更高的可扩展性.在以上国内外的研究中,对于数据库存储优化的研究主要是基于关系型数据库多节点或者行存储、列存储等方面,但是这些方法不能适用 TOTEM 中双向指针表的优化,同时对于新型的非关系型数据库的存储架构方面的研究也不够深入,对于底层的存储研究更是很少涉及.

在 TOTEM 对象代理数据库查询优化方面,目前也有一些相关的研究[27-30].在双向指针表的存储优化方面,已有研究主要是基于列式的双向指针表存储方法和基于代理关系链的双向指针表存储方法.基于列式的双向指针表的存储方法,其基本原理是在使用列式存储结构时,使用参数 n 把关系垂直划分成 n 个子关系,各子关系以〈id,attribute〉的形式分别存储一条记录的对应属性,每一个子关系都包含关系的属性值和用于指定该属性值所属元组的指代标识.基于代理关系链的双向指针表存储方法,

① Kudu: Storage for Fast Analytics on Fast Data. https://kudu.apache.org/kudu.pdf

其基本原理是源对象与任意一个代理对象间都存在代理关系的类路径，使得两对象可达，在对象代理的相关操作中，寻找对象的源类和代理类是最基本的操作，这些操作间存在着相互的关联性，切换表达式和更新迁移均发生在代理关系链的范围内，如果将代理关系链中的关系聚簇存储，将加快关联对象的获取过程，从而提升整个查询的性能. 但是这两种聚簇方法也存在一定的不足. 在基于列式的双向指针表存储方法中，增加了寻道时间和插入操作的代价，同时也增加了元组重构成开销. 在基于代理关系链的双向指针表存储方法中，当一个关联关系发生改变时，可能需要同时移动多条记录，使得数据更新的性能降低，且需要封锁整个代理关系链的关系的查询和修改，对并发性也会带来不利的影响. 因此，需要设计新的双向指针表优化算法，有效地提高查询效率.

本文贡献在于根据双向指针表的存储特点，提出了基于代理关系的双向指针表的存储聚簇优化方法. 在此方法中，将源类和代理类之间在双向指针表中的所有指针关联聚簇存储在同一个磁盘块中. 在查询代理对象虚属性对应的实属性值时，一次性获取并缓存该磁盘块中的双向指针关联，根据缓存中指针关联直接找到代理对象虚属性对应的源对象的实属性的值，从而避免多次查询双向指针表的开销，提高了查询效率.

本文第 2 节对双向指针表的存储相关知识进行介绍；第 3 节介绍双向指针表聚簇存储优化方法；第 4 节通过实验验证方法的有效性；第 5 节对本文工作进行总结.

2　相关知识

数据库管理系统核心功能是数据的存储和数据的检索，因此数据库存储管理模块是数据库管理系统中的核心模块，其承担着数据存储和管理的基本功能. 由于作为存储介质的硬盘的种类以及质量的不同，内存设备同样质量参差不齐导致数据库的 I/O 速度存在着较大的差异，同时，不同数据库的设计理念、目标不尽相同，对数据在磁盘上的组织方式以及使用的索引类型也存在着差异，导致数据库 I/O 速度也存在着差异. 因此，良好的数据组织方式和读写策略能够有效减少数据库 I/O 次数，提高数据库性能.

在数据库管理系统中，为提高系统性能，会利用缓存减少内存和磁盘的数据交换次数. 数据库内存和磁盘以块为单位进行数据交换. 对于一次数据交换过程，即一次磁盘 I/O 操作，如果每次 I/O 操作从磁盘读入到内存的数据都能包含下次数据库需要访问的元组数据，则能够有效地提高缓存命中率，减少磁盘 I/O 次数. 对于数据库的查询事务，往往涉及多条符合条件的记录，如果满足条件的记录集中存储在尽量少的磁盘块中，也能够有效减少事务获取元组时的磁盘读取开销；对于复杂的数据库管理系统，在多个事务并发的情况下，影响事务执行效率的因素更加复杂，但是在数据库缓存大小相同的条件下，集中数据的存储，提高缓存命中率有助于提高数据库系统的事务的吞吐量.

对象代理数据库系统 TOTEM 在存储管理方面采用分页的存储方式，将磁盘文件以分页的形式读入内存中，文件页的大小和磁盘文件块的大小保持相同. 在内存方面，使用 LRU 算法对缓冲区进行管理和替换. 对象代理数据库存储管理系统的设计能够使物理存储和逻辑存储概念分离，达到数据库高效存储和处理的目的.

对象代理数据库系统 TOTEM 采用堆式文件组织方式，结构如图 2 所示. 当插入记录时，数据库系统首先查找具有合适空闲空间的文件块，然后将该文件块读入到内存中，将记录写入空闲空间，最后将缓存数据刷回磁盘. 为了实现空间管理功能，TOTEM 引入了空闲空间管理模块，该模块包括空闲空间管理、磁盘空间回收.

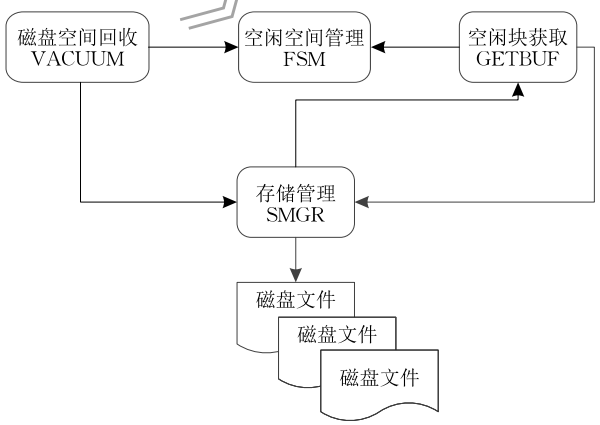


图 2　堆式文件组织管理

由于采用多版本并发控制，在删除记录时，记录不会被真正删除，只有在空闲空间回收时，对象代理数据库系统才会将文件块中标记删除的记录清除，并更新回收后的空闲空间信息. 由于 TOTEM 实现删除操作是通过标记的方式执行的，因此，只有在空

闲空间回收删除的记录后,被删除记录所占用的磁盘空间才真正的被回收.

在 TOTEM 数据库系统中,空闲空间管理为每一个类都创建一个空闲空间记录,当某个类需要创建对象时,调用空闲块获取模块,分配包含足够空闲空间的块的缓存,将对象写入缓存,最后将对象刷回磁盘.具体获取流程如下:

(1)读取空闲空间管理模块 FSM,判断是否具有足够空闲空间大小的磁盘块,如果没有则执行步骤(2),否则执行步骤(3);

(2)调用存储管理器 SMGR,为当前类分配一个新的文件块,并将该文件块读入缓存中,执行步骤(4);

(3)调用存储管理器 SMGR,将 FSM 记录的文件块读入缓存;

(4)执行创建对象操作后,该文件块的空闲空间大小更新为剩余空闲空间大小,并将其加入到空闲空间管理模块 FSM 的空闲空间记录中.

不难发现,在 TOTEM 系统中,当记录被创建时,随机获取具有合适空闲空间大小的磁盘块,然后将记录写入该文件块中.当 TOTEM 进行频繁的更新删除操作后,磁盘上的记录分布是无规律的.因此在查询双向指针表数据时会造成多次的 I/O 操作,降低查询的效率.

3 双向指针存储优化方法

3.1 基于源类和代理类(SD)的聚簇

数据库聚簇方法分为持久性聚簇和一次性聚簇[31-35].持久性聚簇在进行元组的插入,修改操作时会涉及比较复杂的计算,这种操作不适合频繁更新的应用,由于需要保证动态聚簇特性,因此需要在更新或插入时移动相关记录,从而降低了更新和插入的效率,但是这种聚簇方式可以实现自动聚簇,当对象被创建、更新时能保持聚簇的特性.从用户层面来说,用户使用一次性聚簇更加符合现实中用户的需求.一次性聚簇可以在查询效率比较低的情况下根据用户的查询习惯选择合适的聚簇方法,从而提高用户的查询效率.在用户的查询习惯发生变化时,一次性聚簇可以相应的调整聚簇方法,满足用户的需求.因此,基于以上分析,在本文中我们提出基于持久性聚簇和一次性聚簇的两种聚簇方法,从模式和用户角度最大限度的实现元组的聚簇存储,提高 TOTEM 中元组的查询效率.

根据上述分析,本文在 3.1 节中提出一种优化双向指针表的存储方法:基于源类和代理类的双向指针表的元组聚簇方法.为了便于聚簇方法的介绍,本文把源类(SourceClass)和代理类(DeputyClass)简称为 SD.基于 SD 的聚簇就是改变对象代理数据库的对象在磁盘块上随机存储方式,插入元组时,以 SD 为单位进行元组的存储.对双向指针表中的元组以 SD 为单位进行分组,具有相同 SD 的元组存储在相同的磁盘块中,在基于代理对象的虚属性查找源对象的属性时,就会尽可能多的把目标元组读进内存中,从而减少内存的 I/O 次数,提高缓存的命中率.

基于源类和代理类(SD)的聚簇存储通过修改空闲空间管理策略,在原有的空闲空间管理策略中加入新的索引信息,使得当需要添加记录时不仅使用空闲空间管理模块 FSM 查找空闲空间大小一个条件,而且结合使用索引中指定的聚簇信息寻找具有合适空闲空间的磁盘块,然后读取目标磁盘块并进行写入操作.

(1)空闲空间分配

对象代理数据库在分配空闲空间时,增加了新的索引信息,用于指向同一个源类和代理类的最后一个空闲块.如图 3 所示,双向指针表中记录着 4 个 SD 的对象之间的双向指针关联.其中 SD1 对应了两个磁盘文件块,根据数据库分配策略,SD1 对应的第一个文件块已经没有足够的空闲空间存储对象,则分配新的文件块存储对象,并且修改 SD1 索引指向新分配的文件块.

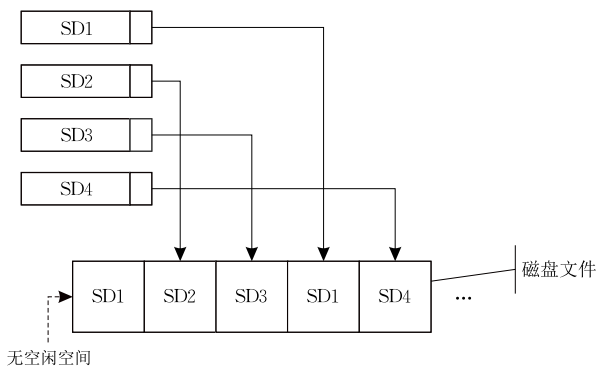


图 3 空闲块管理指针

为了加快查找每一个 SD 的具有空闲空间的磁盘块,我们使用 B+树对磁盘进行组织,每一个 B+树的叶子节点对应一个 SD 的空闲磁盘块,B+树以源类 ID(SourceClassID),代理类 ID(DeputyClassID)两个值作为键值,记录有关 SD 的信息.叶子节点记

录该 SD 对应的空闲磁盘块的信息，图 4 中 BlockNumber 代表空闲磁盘块块号，FreeSpace 代表空闲磁盘块空闲空间大小. 非叶子节点使用图 5 结构组织，其中每一个 SourceClassID，DeputyClassID 都包含一个 PagePointer 指针，指向下一层的 B+树指针，该结构称为块级聚簇空闲空间指针，简称为 BCFP.

图 4　索引叶子节点

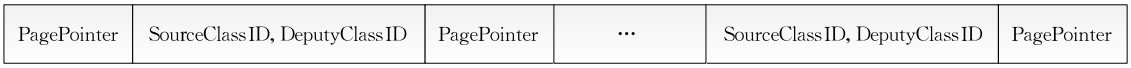


图 5　索引非叶子节点

新的空闲空间分配方法如下：

① 根据插入对象的 SD 检索 B+树，获取该 SD 对应的空闲磁盘块的 BlockNumber 和 FreeSpace；

② 若空闲磁盘块的空闲空间大小满足插入元组，则读取该磁盘块到缓存并更新该叶子节点的 FreeSpace 大小，最后返回，如不满足，则执行步骤③；

③ 若空闲磁盘块 FreeSpace 大小不满足对象的插入，则为该 SD 分配新的空闲块，更新 B+索引中叶子节点的 BlockNumber 和 FreeSpace，最后返回.

(2) 删除记录的空间回收方法

由于对象代理数据库采用了标记删除的策略，因此对于记录删除实际发生在空间回收过程中. 在空间回收过程中，需要保持聚簇特性，因此普通数据库将页面中有效数据向前移动的方式不再适用于对象代理数据库. 为了在回收过程中也能保持 SD 的聚簇，在 B+树叶子页面的末尾增加一个前驱指针，通过该指针使得所有具有相同 SD 的文件块形成一个链表结构. 如图 6 所示，对于每一个 SD 的索引记录，都对应了一条链表结构，其中 SourceDeputy1 对应两个文件块(3,1)，SourceDeputy2 对应两个文件块(5,2)，SourceDeputy3 对应一个文件块(4). 在空闲空间回收时，将建立新的表文件存储有效记录，并删除原有的双向指针表. 采用本方法回收空间以后，有效记录仍然保持着以 SD 为单位的聚簇. 回收后的效果如图 7 所示.

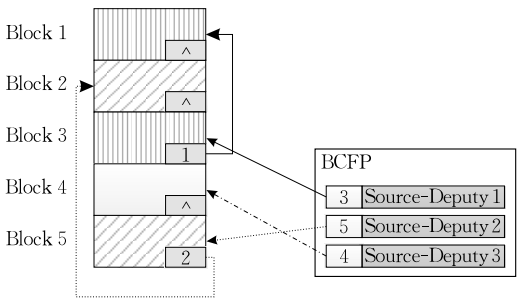


图 6　基于 SD 的文件块链表结构

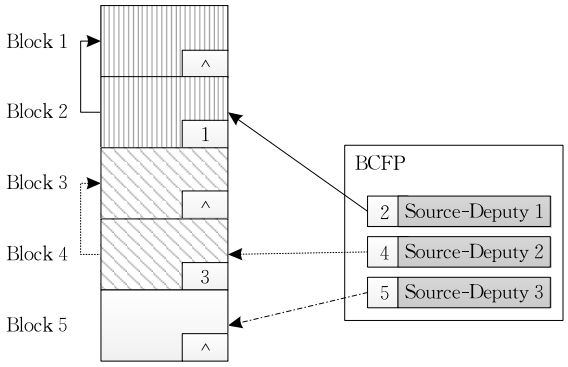


图 7　基于 SD 的聚簇空间回收结果

在进行空间空间回收时，将执行以下流程：

① 根据索引信息获取 SD 前驱指针指向的第一个文件块 oldBlock；

② 分配新的双向指针表，并为其分配一个新的文件块 newBlock；

③ 依次从原双向指针表 oldBlock 中读取有效记录，并将其复制到新的文件块 newBlock 中.

(i) 将 oldBlock 中有效元组移动到新分配的 newBlock 中，并获取 oldBlock 页尾指针指向的文件块，赋值给 oldBlock 中；

(ii) 当 newBlock 空闲空间不足时，在新的双向指针表中申请一个新的空闲块，并将新分配的空闲块的页尾指针指向 newBlock，同时将新的空闲块的块号赋值给 newBlock；

(iii) 当 oldBlock 的页尾指针指向 NULL 时，表示该 SD 对应的页面中所有的有效元组已经移动到新的双向指针表中，更新 newBlock 链表到块级聚簇空闲空间指针对应的 SD 索引项中.

例 1. 在专利查询系统中，我们创建一个 patent 类用于存储所有专利数据的基本信息，并且为其创建一个代理类 patent_common 用来存储专利的主要信息，同时在一级代理类 patent_common 基础上创建一个二级代理类 patent_simple 用来存储用户经常查询的专利数据信息. 双向指针表中存在

patent 类和 patent_common 类的代理关系，其在双向指针表中对象数为 100 条，以及存在相关的 patent_common 类和 patent_simple 类的代理关系，其在双向指针表中对象数为 10 条. 对象代理数据库中源类和代理类之间的代理关系如图 8 所示. 在对象代理数据库中查询二级代理类 patent_simple 时，根据代理关系查找一级代理类 patent_common，查找一级代理类时根据代理关系查找源类 patent 数据并返回.

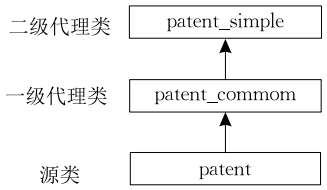


图 8 源类与代理类之间代理关系

在例 1 中，根据基于源类和代理类的聚簇方法可以实现图 9 的聚簇效果. 图 9(a)为 TOTEM 聚簇前随机存储效果，图 9(b)为聚簇以后存储效果.

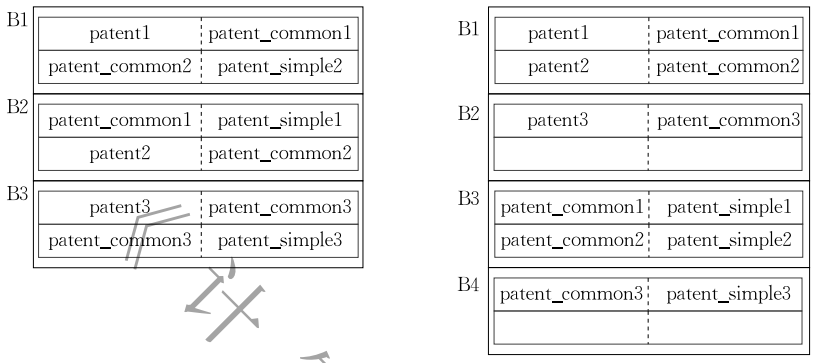


图 9 随机存储和聚簇记录组织结构对比图

3.2 基于用户查询频率的聚簇

3.1 节中提出的基于源类和代理类的聚簇方法实现了持久性聚簇. 对于一级代理关系能够很好的实现聚簇效果，在查询多级代理类时，也能很好地提高 I/O 查询效率，缩短查询时间. 这种聚簇方法是一种通用的聚簇方法，在 TOTEM 系统中可以实现对于任意数据表的聚簇. 但是对于 TOTEM 数据库中存在多级代理关系时查询效率不能达到最优，因此针对多级代理情况下，我们在 3.1 节的聚簇方法的基础上提出一种结合用户对数据库查询习惯的一次性聚簇方法：基于用户查询频率的聚簇方法. 此方法针对特定的应用环境可以使多级代理关系查询效率得到明显的提高. 在此方法中，用户可以执行聚簇命令实现聚簇优化，系统首先根据用户查询日志统计用户对于各个代理类的查询次数，然后 TOTEM 根据用户对于各个代理类的查询次数分别计算以不同源类和代理类为单位进行聚簇的代价，最终选择相较于 TOTEM 随机存储时查询代价减少最大的聚簇单位.

进行代价估计时，本文只考虑代价比较高的 I/O 代价，忽略在内存中其他操作的 CPU 代价，即采用较大粒度的代价估计方式，如果代价估计考虑的因素太多，代价估计本身也会引入较大的代价且意义不大.

在 TOTEM 存储中，查询双向指针表的代价如式(1)所示：

$$DeputyCost(t)=\sum_{i=1}^{t}Cost(i) \tag{1}$$

其中 t 代表基于源类的基础上存在 t 级代理类，$Cost(i)$为用户查询第 i 级代理类的查询双向指针表的代价，$DeputyCost(t)$为该用户查询的总代价.

例 1 中，查询双向指针表中对象的总代价 $DeputyCost(2)=Cost(1)+Cost(2)$，即查询一级和二级代理类的总代价.

在随机存储方式下，TOTEM 对于查询第 n 级代理类的代价如式(2)所示：

$$Cost(n)=S_n\times C_{io}\times\left(\sum_{i=1}^{i=n-1}P_i+D_n\right) \tag{2}$$

其中 S_n为查询第 n 级代理类的次数，P 为查询第 i 级代理类所需花费的 I/O 代价，C_{io}为从磁盘顺序读取一页的代价，D_n为第 n 级代理类与其源类在双向指针表中的对象所在的磁盘块数.

例 1 中，假设查询一级代理类的次数 $S_1=100$，查询二级代理类的次数 $S_2=100$，$C_{io}=1$，$D_1=10$，$D_2=10$. 因此查询一、二级代理类代价分别为

$Cost(1)=100\times1\times10=1000$,

$Cost(2)=100\times1\times(10+P_1)=1000+100P_1$.

文献[36]提出一个经典的代价估计公式，用于估计数据库系统中管理缓存时使用 LRU 替换算法的索引计划的代价，本文根据该公式设计了 P_i 的计算如式(3)所示：

$$P_i=\begin{cases}\min\left(\dfrac{2T_iT_{i+1}}{2T_i+T_{i+1}},T_i\right), & T_i\le b\\ \dfrac{2T_iT_{i+1}}{2T_i+T_{i+1}}, & T_i\ge b \text{ and } T_{i+1}\le\dfrac{2T_ib}{2T_i-b}\\ b+\left(T_{i+1}-\dfrac{2T_ib}{2T_i-b}\right)\times\dfrac{T_i-b}{T_i}, & T_i\ge b \text{ and } T_{i+1}\ge\dfrac{2T_ib}{2T_i-b}\end{cases}\tag{3}$$

其中 T_i 为第 i 级代理类与其源类在双向指针表中的存储的对象数，b 为缓冲区的块数.

例 1 中，假设 $b=10$，当查询二级代理类时，此时计算出 $T_1=100>b=10$，且 $T_2=100<(2000/190)$，因此计算出 $P_1=2\times100\times10/210=2000/210$，所以 $Cost(2)=1000+20000/21$.

在一次性聚簇中，对于每一个代理类的查询的代价如式(4)所示：

$$Cluster(n)=S_n\times Q_n\times C_{io}\tag{4}$$

其中 S_n 为查询第 n 级代理类的次数，Q_n 为查询第 n 级代理类所需花费的 I/O 代价，C_{io} 为从磁盘顺序读取一页的代价. 而聚簇是基于源类和代理类 n 聚簇的，所以在读入缓冲区中的数据都是查询所需要的，不需要进行多次的磁盘块的替换操作，其 I/O 代价如式(5)所示：

$$Q_n=\frac{\sum_{i=1}^{i=n}T_i}{m}\tag{5}$$

其中 T_i 为第 i 级代理类与其源类在双向指针表中的存储的对象数，m 表示每个磁盘块中存储的双向指针表中的对象数. 由于在磁盘中，每个对象所占的空间是一定的，所以 m 为定值.

例 1 中，假设 $m=50$，则聚簇以后查询一、二级代理类的代价分别为

$Cluster(1)=1\times100\times100/50=200$,

$Cluster(2)=1\times100\times(100+10)/50=220$.

基于以上的代价公式，在基于源类的基础上存在 t 级代理类的情况下，源类和第 n 级代理类的聚簇之后的总的查询代价如式(6)所示：

$$ClusterCost(n)=\sum_{i=1,i\ne n}^{t}Cost(i)+Cluster(n)\tag{6}$$

例 1 中，以源类和一级代理类为单位聚簇，查询代价为 $ClusterCost(1)=Cluster(1)+Cost(2)$.

以源类和二级代理类为单位聚簇，查询代价为 $ClusterCost(2)=Cost(1)+Cluster(2)$.

其中当基于 n 级代理类进行聚簇时，其他层次的查询代价可以认为与聚簇前的查询代价相同，因此根据聚簇前的代价 $DeputyCost(t)$ 和聚簇后的代价 $ClusterCost(n)$，可以计算经过对基于源类和第 n 层代理类的聚簇之后，计算 I/O 效率的提高值如式(7)所示：

$$Improve(n)=DeputyCost(t)-ClusterCost(n)\tag{7}$$

对于用户查询的所有代理类经过效率提高公式的计算，从中选择出效率提升最高时对应的代理类 i，此时可以得出当基于源类和代理类 i 聚簇时，即把源类和代理类以及它们之间的代理类在双向指针表的对象聚簇在同一个磁盘块中，查询双向指针表的效率能够达到最优.

算法 1 实现了基于用户查询频率的代价计算以及双向指针表的聚簇过程.

算法 1.　双向指针表的聚簇存储.

输入：双向指针表所有的源类集合 Y，所有代理类集合 D，原双向指针表 *oldTable*

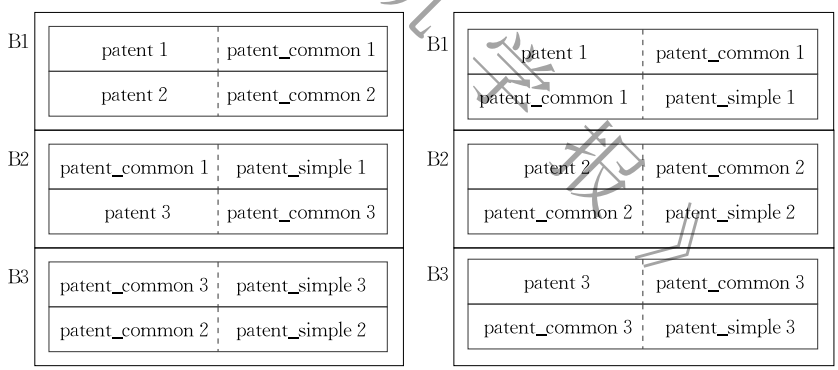
输出：新的双向指针表 *newTable*

```
1.  初始化源类计数器 i=0；初始化数据集 DS={}；
2.  FOR j=1→N
        根据源类寻找与源类相关的代理类，并且把 oldTable 表中源类和相关的代理类划分到相同的数据集 DS 中 DS=Add_Q(Q_j)；//构建数据集
3.  分配相应的内存块用来存储数据 C；
4.  WHILE (DS!=NULL) DO
5.    Q=GetFromDS()；
6.    Y=ComputeSource(Q)；//获取数据集 Q 中的源类
7.    D=ComputeDeputy(Q)；//计算 Q 中代价最小的代理类
8.    DH=GetDeputyObject (D)；//构建元组集合 DH
9.    RemoveFromDS(Q)；
10.   WHILE (DH!=NULL) DO
11.     Object=GetDeputyTuple(DH)；//获取 DH 中的元组
12.     WriteToC(Object)；
13.     RemoveFromDH (Object)；//删除 DH 中的元组
```

```
14.    WHILE(!ContainsSource(Y,Object)) DO
15.      Object=GetSourceObject(Object);//获取源对象
16.      WriteToC(Object);
17.    END WHILE
18.  END WHILE
19.  WHILE (Q!=NULL) DO;
20.    Object=GetObject(Q);
21.    WriteToC(Object);
22.    RemoveFromQ(Object);
23.  END WHILE
24. END WHILE
25. WriteToDisk();
26. newTable=GetNewTable(); //返回新的双向指针表
27. RETURN newTable;
```

在构造数据集 DS 时，函数 $Add\_Q()$ 通过遍历把具有关联性的源类代理类构造成集合 Q 然后加入到 DS 中，通过 $ComputeSource()$，$ComputeDeputy()$ 函数计算出集合 Q 中聚簇代价最小的源类 Y 和代理类 D. $GetDeputyObject()$ 函数返回在双向指针表中包含最小聚簇代价的代理类 D 在双向指针表中的关联对象集合. 在计算出代价最小的源类 Y 和代理类 D 之后，通过 $GetDeputyTuple()$ 函数依次返回代理类 D 在双向指针表中的关联对象元组，$ContainsSource()$ 函数中参数 Y 的作用是在查找代理类 D 及其源类在双向指针表中的关联对象时，判断查找是否已经回溯到源类 Y，若已经回溯到源类 Y，则表明回溯查找已经完成，若没有回溯到源类 Y，则需继续回溯，直到回溯查找到源类 Y 成功返回. 在聚簇完成后，新的双向指针表 $newTable$ 替换旧的双向指针表 $oldTable$，返回新的双向指针表 $newTable$.

例 1 中，当以源类和一级代理类为单位聚簇时，聚簇后的查询代价减少了：$Improve(1)=800$，当以源类和二级代理类为单位聚簇时，聚簇后的查询代价减少了：$Improve(2)=780+20000/21$. 可以明显看出 $Improve(2)>Improve(1)$，因此选择以源类 patent 和二级代理类 patent_simple 为单位进行聚簇，能够实现最好的聚簇效果. 根据代价公式的计算，选择以源类 patent 和二级代理类 patent_simple 为单位进行聚簇，实现图 10 的聚簇效果，图 10(a)为 TOTEM 聚簇前随机存储效果，图 10(b)为聚簇后存储效果.



(a) 随机存储 (b) 聚簇存储

图 10 随机存储和聚簇记录组织结构对比图

4 实 验

本文提出的基于代理关系的双向指针表的存储聚簇优化方法已经在 TOTEM 数据库系统中实现. 本节通过具体的实验比较本文提出的基于代理关系的双向指针表中对象聚簇优化方案与原方案的查询效率的差异，已经在 TOTEM 2.2 数据库系统中完成实验.

4.1 实验环境和实验数据

实验测试环境包含硬件环境和软件环境两部分. 硬件环境配置如下：4 GB 内存，1 TB 硬盘，Intel Core i5-4460 3.20 GHz 处理器，软件环境配置如下：Ubuntu 15.04 操作系统，对象代理数据库系统 TOTEM 2.2.

实验采用的测试数据库是一个专利数据库，实验的数据集来自中国专利局，数据集中的专利数据主要是钢铁，建筑，造船，以及材料等方面的专利信息. 数据集包含源类 patent 以及十级代理类. 其中使用一级代理类 patent_common、五级代理类 patent_five 以及十级代理类 patent_ten 作为实验中查询的代理类进行实验. 3 种代理类分别代表了用

户在使用专利系统过程中所进行的查询频率最高的实验测试数据.实验采用 5 种规模不同的数据集进行测试,表 1 中的数据表示了不同规模的数据集的源类(代理类)中源对象(代理对象)的数量.在设计实验参数表时,考虑到本文所提方案在数据集、代理类的层次和代理类查询次数 3 种实验参数影响下的优化效果.表 2 中的数据表示了以上 3 种实验参数在测试实验中的取值大小和范围.

表 1　测试数据集

数据集	基本类对象数	一级代理类对象数	五级代理类对象数	十级代理类对象数
	size of patent	size of patent_common	size of patent_five	size of patent_ten
D1	100000	100000	100000	100000
D2	200000	200000	200000	200000
D3	500000	500000	500000	500000
D4	1000000	1000000	1000000	1000000
D5	2000000	2000000	2000000	2000000

表 2　实验参数表

	实验	代理层次	查询次数
a	{D1,D2,D3,D4,D5}	1	100
b	{D1,D2,D3,D4,D5}	5	100
c	D4	{1,5,10}	100
d	D4	5	{100,200,300,400,500}

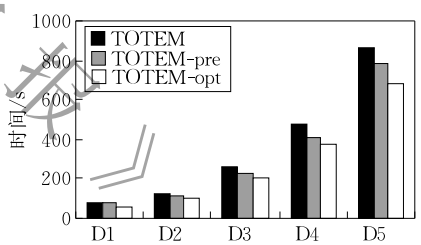
4.2　实验结果与分析

本节通过实验分析基于代理关系的双向指针表中对象的聚簇对虚属性查询的影响.实验参数表的查询次数为各级代理类查询的总次数.

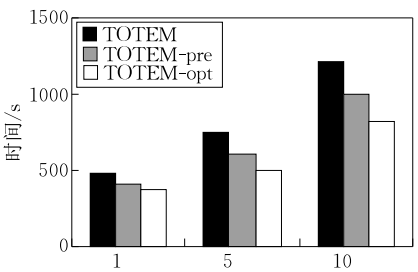
在实验过程中,对于每一个实验参数的变化,实验均进行十次,统计十次查询时间,求出平均值即为该参数变化时的查询时间.

在实验中,主要进行本文优化方法和未优化之前的对比,本文优化方法和以往对双向指针表的优化方法的对比.在已有的双向指针表的优化过程中,基于列式的双向指针表存储方法适用于持久性聚簇,在本文中使用基于源类和代理类的聚簇方法与之进行对比.基于代理关系链的双向指针表存储方法由于其数据更新性能和并发性能不好,只适用于一次性聚簇,因此在本文中使用基于用户查询频率的聚簇方法与之进行对比.

在基于源类和代理类的聚簇方法的实验中,实验结果如图 11 所示(纵坐标为查询时间,单位 s).TOTEM 代表未使用优化存储的查询时间,TOTEM-pre 表示使用基于列式的双向指针表存储方法的查询时间,TOTEM-opt 表示使用基于源类和代理类的聚簇方法的查询时间.由图 11(a)可知,使用基于源类和代理类的聚簇方法优化了双向指针表中对象在磁盘上的存储方式,对于不同的数据集,随着数据集数据量的增加,优化后的查询时间比优化前的查询时间明显减少,并且和基于列式的存储方法相比,使用基于源类和代理类的聚簇方法查询时间更短,并且随着数据量的增大,优化效果更加明显,有效提高了查询效率.由图 11(b)可知,对于同一个数据集,用户对于不同的查询的代理层次不同时,当代理层次较低时,由于双向指针表中数据量比较小,和聚簇前相比,聚簇优化以后优化效果不是很明显,但是当代理层次增多以后,双向指针表中的数据成倍的增加,聚簇优化以后查询时间明显减少.和基于列式的存储方法相比,理论上在一层代理类时,使用基于源类和代理类的聚簇方法的优化效果更明显,但是一层代理类时,数据量比较少,所以效果不是很明显,但是总体的聚簇效果优于基于列式的存储方法的优化效果.



(a) 不同数据集

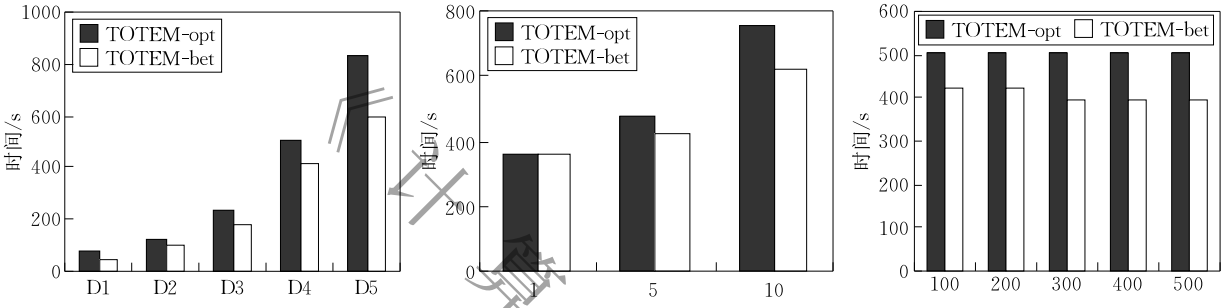


(b) 不同代理层次

图 11　双向指针聚簇存储实验结果

在基于用户查询频率的聚簇方法的实验中,实验结果如图 12 所示(纵坐标为查询时间,单位 s),TOTEM-opt 代表使用基于源类和代理类的聚簇方法的查询时间,TOTEM-bet 表示使用基于用户查询频率的聚簇方法的查询时间.由图 12(a)可知,在

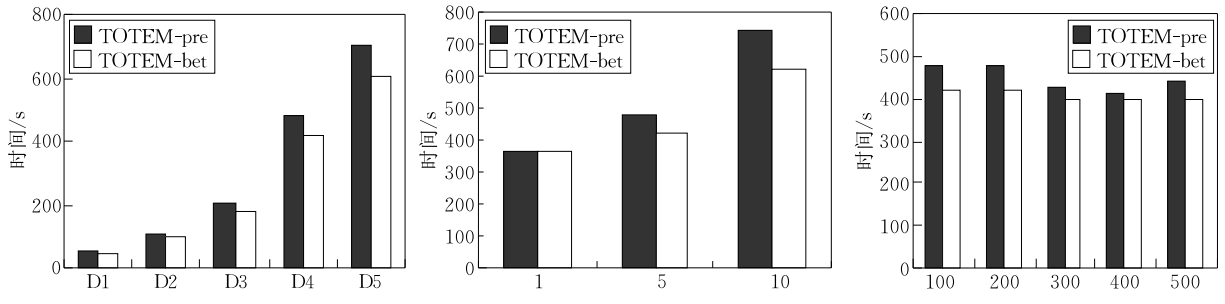
查询五级代理类时,针对用户特定的查询,在对于不同数据集上的查询使用基于用户查询频率的聚簇方法比使用基于源类和代理类的聚簇方法有效的减少了查询时间.由图 12(b)可知,对于同一个数据集,使用基于用户查询频率的聚簇方法和使用基于源类和代理类的聚簇方法在一级代理关系上查询时间基本相同,但是在多层次的代理关系查询时可以有效提高效率.由图 12(c)可知,对于同一个数据集,当用户查询次数不同时,系统根据对于不同层次的代理类的查询次数决定经过代价公式的计算以哪一种代理类和源类作为聚簇单位进行聚簇,经过计算对于以相同的源类和代理类为单位的聚簇,查询时间相同.经过代价公式的计算调整了聚簇的源类和代理类,则查询时间会随之变化,在代理层次较少并且查询高层次代理类的次数较高时,查询效率会相对提高,其原因是当以较高层次的代理类和源类为单位聚簇时,相当于对低层次的代理类和源类做了一个部分聚簇.通过对图 12 进行分析可以看出在特定的用户查询习惯下,经过基于用户查询频率的聚簇方法优化存储以后,对于多级代理类的查询效率明显比使用基于源类和代理类的聚簇方法优化查询效率得到提高,有效的减少了用户查询虚属性带来的 I/O 次数,提高了查询效率.



(a) 不同数据集 (b) 不同代理层次 (c) 不同查询次数

图 12 双向指针聚簇存储实验结果

由于基于代理关系链的双向指针表存储方法只适用于一次性聚簇,所以和本文中的基于用户查询频率的聚簇方法做单独的对比实验.实验结果如图 13 所示(纵坐标为查询时间,单位 s).TOTEM-pre 代表使用基于代理关系链的双向指针表存储方法的查询时间,TOTEM-bet 表示使用基于用户查询频率的聚簇方法的查询时间.由图 13(a)可知,在查询五级代理类时,针对用户特定的查询,在对于不同数据集上的查询使用基于用户查询频率的聚簇方法比基于代理关系链的双向指针表存储方法有效的减少了查询时间,并且数据量越大,两种方法之间的优化的效果对比越明显.由图 13(b)可知,对于同一个数据集,使用基于用户查询频率的聚簇方法和基于代理关系链的双向指针表存储方法在一级代理关系上查询时间完全相同,这是因为在一层代理模式下,两种方法的存储策略完全相同.但是在多层次的代理关系查询时可以有效提高效率.由图 13(c)可知,对于同一个数据集,当用户查询次数不同,查询代理层次较高的代理类次数比较多时,使用基于用户查询频率的聚簇方法的聚簇效果更加稳定,并且在总体上的优化效果上比基于代理关系链的双向指针表存储方法的效果好.



(a)不同数据集 (b) 不同代理层次 (c) 不同查询次数

图 13 双向指针聚簇存储实验结果

5 总　结

在 TOTEM 中，虚属性查询需要查找双向指针表中对应的源类和代理类的对象，因此在查询代理对象中的虚属性时会耗费大量的时间. TOTEM 对象代理数据库系统需要提出一种新的双向指针的存储方法. 因此本文提出了双向指针表存储优化方法——基于代理关系的双向指针表的存储聚簇优化方法. 在基于源类和代理类(SD)聚簇方法中，对双向指针表中的元组以源类和代理类为单位进行分组，具有相同源类和代理类的元组存储在相同的磁盘块中. 在基于用户查询频率聚簇方法中，系统根据用户查询日志得出用户查询各个代理类的频率，通过计算各级代理类和源类聚簇之后的查询代价，选择出一种最适合，查询代价最小，性能最优的代理类和源类的聚簇单位. 两种聚簇方法把对象聚簇在同一个磁盘块中，从而在查询双向指针表时，减少磁盘读入内存的 I/O 次数，提高查询双向指针表的效率. 本文对两种聚簇方法在数据集、查询次数以及代理层数三个参数的影响下进行了测试，和以往研究成果进行对比，通过数据对比，本文中双向指针表的聚簇方法优化效果比未优化之前提高 15%以上，比以往研究对双向指针表的优化效果提高 9%以上，验证了基于代理关系的双向指针聚簇方法的有效性.

致　谢　作者感谢武汉大学计算机学院彭煜玮副教授和王黎维副教授对本文研究内容的指导和对文字的精心修改. 本文在写作过程中还得到了刘梦兰和闫伟霞的帮助，以在此一并深表感谢！

参　考　文　献

[1] Peng Zhiyong. An Object Deputy Model for Advanced Database Applications. Kyoto, Japan: Kyoto University, 1994

[2] Wang L, Peng Z, Luo M, et al. A scientific workflow framework integrated with object deputy model for data provenance//Proceedings of the International Conference on Web-Age Information Management. Hong Kong, China, 2006: 569-580

[3] Kim W. A model of queries for object-oriented databases//Proceedings of the 15th International Conference on Very Large Data Bases, Amsterdam, Netherlands, 1989: 423-432

[4] Kambayashi Y, Peng Zhiyong. Object deputy model and its applications//Proceedings of the 4th International Conference on Database Systems for Advanced Applications. Singapore, 1995: 1-15

[5] Ye Juan, Wang Zhen, Peng Zhi-Yong. Extension of ODMG with deputy class//Proceedings of the China National Computer Congress. Beijing, China, 2003: 856-864(in Chinese)
(叶娟，王桢，彭智勇. 通过引入代理类扩展 ODMG. 中国计算机大会. 北京，中国，2003：856-864)

[6] Peng Z, Kambayashi Y. Deputy mechanisms for object-oriented databases//Proceedings of the 11th International Conference on Data Engineering. Taipei, China, 1995: 333-340

[7] Peng Zhiyong, Shi Yuan, Zhai Boxuan. Realization of biological data management by object deputy database system//Proceedings of the Transactions on Computational Systems Biology. Berlin, Germany, 2006: 49-67

[8] Peng Z, Peng Y, Zhai B. Using object deputy database to realize multi-representation geographic information system//Proceedings of the ACM International Symposium on Geographic Information Systems. Washington, USA, 2007: 43-46

[9] Ishikawa H, Yamane Y, et al. An object-oriented database system Jasmine: Implementation application and extension. IEEE Transactions on Knowledge and Data Engineering, 1996, 8(2): 285-303

[10] Fishman D H, Beech D, Cate H P, et al. Iris: An object-oriented database management system//Stonebraker M eds. Readings in Object-Oriented Database Systems. San Francisco: Morgan Kaufmann Publishers Inc., 1989: 216-226

[11] Bagui S. Achievements and weaknesses of object-oriented databases. Journal of Object Technology, 2003, 2(4): 29-41

[12] Jiang Lian, Li Rong-Rong, Peng Zhi-Yong. A cross-class query optimization method of object deputy database. Computer Engineering and Science, 2016, 38(7): 1425-1433 (in Chinese)
(蒋廉，李蓉蓉，彭智勇. 一种对象代理数据库的跨类查询优化方法. 计算机工程与科学，2016，38(7)：1425-1433)

[13] Liu Dong-Ming, Wang Liang, Wang Li-Wei, et al. Query optimization for virtual attributes in object deputy database. Computer and Digital Engineering, 2014, 42(10): 1792-1797 (in Chinese)
(刘东明，王梁，王黎维等. 对象代理数据库的虚属性查询优化方法. 计算机与数字工程，2014，42(10)：1792-1797)

[14] Shi Yuan, Peng Zhi-Yong, et al. An OID recycle mechanism for object-relational Database. Computer Science, 2004, 31(10): 566-568(in Chinese)
(施源，彭智勇等. 对象关系数据库中 OID 回收机制. 计算机科学，2004，31(10)：566-568)

[15] Calvanese D, De Giacomo G, Lenzerini M, et al. Rewriting of regular expressions and regular path queries//Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. New York, USA, 1999: 194-204

[16] Hellerstein J M, Stonebraker M. Predicate migration: Optimizing queries with expensive predicates//Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. Washington, USA, 1993: 267-276

[17] Schall D, Härder T. Dynamic physiological partitioning on a shared-nothing database cluster//Proceedings of the 31st IEEE International Conference on Data Engineering. Seoul, South Korea, 2015: 1095-1106

[18] Mukherjee N, Chavan S, Colgan M, et al. Distributed architecture of oracle database in-memory. Proceedings of the VLDB Endowment, 2015, 8(12): 1630-1641

[19] Kargin Y, Kersten M, Manegold S, et al. The DBMS- your big data sommelier//Proceedings of the IEEE International Conference on Data Engineering. Seoul, South Korea, 2015: 1119-1130

[20] Liu W, Zhou Y. A kind of memory database engine based on column-storage techniques//Proceedings of the 2013 8th International Conference on Computer Science & Education. Colombo, Sri Lanka, 2013: 436-440

[21] Apte T, Ingle M, Goyal A K. Hybrid storage architecture: A survey. International Journal of Computer Science and Information Security, 2013, 11(9): 97

[22] Al-Kateb M, Sinclair P, Au G, et al. Hybrid row-column partitioning in teradata®. Proceedings of the VLDB Endowment, 2016, 9(13): 1353-1364

[23] Rao J Shekita E J, Tata S. Scalable row-store with consensus-based replication. USA, 2015-6-2

[24] Larson P A, Clinciu C, Fraser C, et al. Enhancements to SQL server column stores//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 1159-1168

[25] Cattell R. Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 2010, 39(4): 12-27

[26] Grund M, Krüger J, Plattner H, et al. HYRISE: A main memory hybrid storage engine. Proceedings of the VLDB Endowment, 2010, 4(2): 105-116

[27] Bertino E, Negri M, et al. Object-Oriented query languages: The notion and the issues. IEEE Transactions on Knowledge Data Engineering, 1992, 4(3): 223-237

[28] Zhang Guang-Zhou, Peng Zhi-Yong, Xiao Jing-Jing, et al. Query processing and optimization of an object deputy database management system. Computer Science, 2005, 32(5): 97-100(in Chinese)
(张广舟，彭智勇，肖静静等. 对象代理数据库的查询处理与优化. 计算机科学，2005，32(5)：97-100)

[29] Bertino E, Guglielmina C. Path-Index: An approach to the efficient execution of object-oriented queries. Data and Knowledge Engineering, 1993, 10(1): 1-27

[30] Kifer M, Kim W, Sagiv Y. Querying object-oriented databases. ACM SIGMOD Record, 1992, 21(2): 393-402

[31] Huang Ze-Qian. Studies on the Clustering Strategy and Query Optimization Issues of the Object Deputy Database[Ph. D. dissertation]. Wuhan University, Wuhan, 2011(in Chinese)
(黄则谦. 对象代理数据库聚簇策略与查询优化技术研究[博士学位论文]. 武汉大学，武汉，2011)

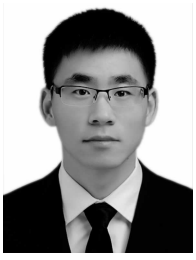
[32] Jenq B P, Woelk D, Kim W, et al. Query processing in distributed ORION//Proceedings of the International Conference on Extending Database Technology. Berlin, Germany, 1990: 169-187

[33] Benzaken V. An evaluation model for clustering strategies in the O2 object-oriented database system//Proceedings of the International Conference on Database Theory. Berlin, Germany, 1990: 126-140

[34] Benzaken V, Delobel C. Enhancing performance in a persistent object store: Clustering strategies in O//Proceedings of the 4th International Workshop on Persistent Objects. Massachusetts, USA, 1990: 403-412

[35] Bertino E, Saad A A, Ismail M A. Clustering techniques in object bases: A survey. Data and Knowledge Engineering, 1994, 12(3): 255-275

[36] 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



HU Cong-Rui, born in 1991, M. S. candidate. His research interests include data management and data mining.

LIU Bin, born in 1975, Ph. D., lecturer. His research interests include data management and data mining.

FENG Ling, born in 1986, Ph. D., lecturer. His research interests include patent analysis and mining.

WANG Fei, born in 1989, Ph. D. candidate. His research interests include data mining and information retrieval.

PENG Zhi-Yong, born in 1963, Ph. D., professor, Ph. D. supervisor. His research interests include complex data management, Web data management, and trusted data management.

Background

Over the past 20 years, a large variety of data has been generated, which includes structured and unstructured data. We entered a big data Era. Traditional database management system cannot manage those data effectively. Then, we need a novel method to process those data. Object-deputy model is a novel data model, which integrate the relational model and the object model together. Many researchers adopted object-deputy model into their researches and have achieved good results.

Totem is a database management system based on object-deputy model. The great effect of deputy is that we can use it to model complex semantic relations.

In order to retrieve semantic relations, Totem uses a bidirectional pointer table to manage the association between the source object and deputy object. Some work has been done to optimize the bidirectional pointer table storage only considering one-layer-deputy. If we allow multi-layer-deputy, for example A is a source object, A_1 is a deputy object of A, and A_2 is a deputy object of A_1, and so on, A_n is a deputy object of A_{n-1}. So for a given deputy object An, we need access bidirectional pointer table n times in the worst case that we can get its source object A, which will be a performance bottle neck of Totem. In this paper, we put forward a bidirectional pointer table clustering method using schema information. Furthermore we optimize clustering method according to user's query content and frequency. We apply our method on synthetic data and real dataset. Experiment results show that this clustering method can decrease I/O cost effectively. In general, our work solves the bidirectional pointer table storage problem more comprehensively which will help to improve Totem performance.

This work is supported by the National Key Research and Development Plan under Grant No. 2016YFB1000701, the National Natural Science Foundation of China under Grant No. 61232002, and the Science and Technology Support Program of Hubei Province under Grant No. 2015BAA127.