计算机科学2000 Vol. 27 №. 2

面向对象数据库的并行查询处理*)

Research of Parallel Query Processing Algorithm In Object-Oriented Database

王意洁 王勇军 胡守仁

(国防科技大学计算机系并行与分布处理国家重点实验室 长沙410073)

Abstract In this paper, according to the features of the object-oriented database and its query, the stage-by-stage executing strategy, the class-based hybrid data placement strategy, the mark-based data operation parallel executing algorithm and the semi-join-based parallel query processing algorithm are proposed, the correctness of the algorithm is proved, and the results of performance evaluation are given.

Keywords Object-oriented database, Data placement, Data operation, Query processing, Parallel

一、引言

随着数据库规模日趋庞大，其查询日趋复杂，对数据库系统性能的要求也越来越高。另一方面，并行计算机系统迅速发展，许多商品化的高性能并行计算机系统相继投入市场，许多数据库研究者们认为，在并行计算机系统上实现数据库管理系统能够极大地提高数据库管理系统的性能，并行数据库管理系统有望成为未来的高性能数据库管理系统。

目前，国际上已经有几个研究机构涉及了并行面向对象数据库(POODB)这个研究领域，但还没有研制出比较成熟的原型系统。从现在的研究情况[1~4]来看，虽然研究者们的思路各不相同，但他们的研究存在一些共同问题，基本上是借鉴并行关系数据库(PRDB)的思想和技术，并根据面向对象数据库(OODB)的特点进行适当的修改和扩充。通过深入分析，我们发现，OODB与关系数据库(RDB)之间有着本质区别：①OODB处理的是复杂对象而不是规格化的关系元组。②RDB只处理数据的检索、修改、插入和删除；OODB除了处理这些传统数据库操作外，还处理用户定义的操作(即：对象的方法)。因此，对PRDB的思想和技术加以"改造利用"的方法是不能充分利用OODB及其查询的本质特点的。

针对OODB及其查询的本质特点，本文提出了查询处理的分阶段执行策略、基于对象类的混合式数据放置策略、基于合格标记的数据操作并行执行算法和基于semi-join的并行查询处理算法[5,6]，对有关算法的正确性进行了证明，并给出了性能评价的结果。

二、面向对象数据库及其查询

一个面向对象数据库既是一个类的集合，也是一个对象的集合。在内涵级和外延级两个层次上都可以用图来表示面向对象数据库。在内涵级，数据库可以定义为一个由相互关联的类构成的集合，采用模式图来表示。依据ODMG-93标准[7]，对应于现实世界中的实体的对象被划分为两类：对象和文字，其中，对象具有独立于内容的标识符(即：对象标识符Oid)，文字没有对象标识符，而是用它的内容作为标识。类的性质也分为两类：属性和关系，其中，属性的取值为文字或由文字构成的各种结构，关系在两个对象类之间定义，而且只支持二元关系。在外延级，数据库可以视为由属于不同类的相互关联的对象构成的网络，采用对象图来表示。

可以将OODB中的查询转化为查询图，查询图是模式图的进化子图，具体而言，查询图是由模式图的子图依据类与类之间的继承关系进化而成。查询图可以是线型结构、树型结构或网络结构。数据库中查询所涉及到的部分可以用查询对象图表示，它是对象图的进化子图，具体而言，查询对象图是由对象图的子图依据类与类之间的继承关系进化而成。分阶段执行策略第

*)本文得到国家高技术研究发展计划项目863-306-ZT01-06-1的资助。王意洁 博士，助理研究员，研究领域为面向对象数据库和并行分布处理技术；王勇军 博士，助理研究员，研究领域为虚拟实现、数据库和并行分布处理技术；胡守仁 教授，博士生导师，研究领域为面向对象数据库、知识库、高性能机体系结构及并行分布处理技术。

一步产生的前期结果可以用前期结果图来表示，它是对象图的进化子图，是查询对象图的子图。

下面，我们给出模式图、对象图、查询图、查询对象图和前期结果图的形式化定义。

设$(C_i,C_j)_k$表示类C_i与类C_j之间的第k个关系，O_{ip}表示类C_i中的第p个对象，$(O_{ip},O_{jq})_k$表示类C_i中的第p个对象与类C_j中的第q个对象通过类C_i与类C_j之间的第k个关系相连。

面向对象数据库D的模式图：$SG_D(C_D,A_D)$，其中：$C_D=\{C_i\}$，$A_D=\{(C_i,C_j)_k\}$。

面向对象数据库D的对象图：$OG_D(O_D,E_D)$，其中：$O_D=\{O_{ip}\}$，$E_D=\{(O_{ip},O_{jq})_k\}$。

作用于面向对象数据库D上的查询Q的查询图：$QG_Q(C_Q,A_Q)$，其中：$C_Q=\{C_i\}$，$A_Q=\{(C_i,C_j)_k\}$，它们分别是查询Q所涉及的对象类的集合以及对象类之间关系的集合。

作用于面向对象数据库D上的查询Q的查询对象图：$OG_Q(O_Q,E_Q)$，其中：$O_Q=\{O_{ip}\}$，$E_Q=\{(O_{ip},O_{jq})_k\}$，它们分别是查询Q所涉及的对象的集合以及对象之间关系的集合。

作用于面向对象数据库D上的查询Q的前期结果图：$OG_{res}(O_{res},E_{res})$，其中：$O_{res}=\{Oip\}$，$E_{res}=\{(O_{ip},O_{jq})_k\}$，它们分别是查询前期结果中的对象的集合以及对象之间关系的集合。

为了以后说明方便，假设任何两类之间至多存在一种关系，所以$(C_i,C_j)_k$中的下标k可以省略。

三、分阶段执行策略

由于面向对象数据库本身的特点，面向对象查询往往很复杂，查询时间开销大，尤其是包含多个连接操作的查询。为了避免访问和处理大量不必要的数据，我们在对面向对象查询处理的本质特点进行分析的基础上，提出了面向对象查询的分阶段执行策略[5]，其主要思想是：第一步，通过在对象类之间处理和传播对象标识Oid，以Oid的形式确定"合格"的对象，也就是查询的前期结果；第二步，将系统定义的或用户定义的函数作用于第一步得到的前期结果，从而产生最终结果。在分阶段执行策略中，把对大量描述数据的检索推迟到确定"合格"的对象之后，从而避免了对大量不必要数据的访问和处理；而且，为了减少处理和存储中间结果所需的时间和空间，通过对数据库中符合条件的对象进行标记来确定"合格"的对象而避免产生大量的中间结果。

四、基于对象类的混合式数据放置策略

在基于无共享结构的并行数据库系统中，数据并行性是并行性的主要形式，它要求数据库中的数据在系统的多个结点上划分，这就是数据放置。数据放置是数据划分与数据分配的组合。数据划分是将数据库中的数据分割为若干大小相同或不同的数据子集，数据分配是将这些数据子集合理地分配到各个结点上。分析表明，查询的并行执行时间是CPU时间、磁盘I/O时间和网络通信时间三者重叠的结果[4]。因此，数据放置策略的目标应该是：各处理机之间负载均衡，各处理机之间的通信开销达到最小，避免不必要的磁盘I/O请求。

目前，POODB的研究者们对数据放置问题进行了大量的研究[1-4]。现有POODB所采用的数据放置策略与并行关系库的很相近，对OODB的特点考虑得不够充分，不能兼顾负载均衡问题和通信开销问题。

基于上述考虑，我们提出了以下数据放置原则：(1)提高访问数据的速度；(2)能有效地开发各种并行性；(3)在保证负载均衡的情况下，考虑使通信开销趋于最小。基于上述原则，我们提出了基于对象类的混合式数据放置策略[5,6]，它包括混合式数据划分策略和基于对象类的数据分配策略。传统的数据划分策略可以大致分为两种：水平划分策略和垂直划分策略，水平划分策略有利于开发对象间并行性，垂直划分策略有利于开发对象内并行性，各有利弊。混合式数据划分策略依据数据放置原则的第一条和第二条，兼顾了水平划分策略和垂直划分策略二者的优点，它的具体思想是：首先，对数据库中的对象进行垂直划分，产生两种数据子集：一种是由二元组(Oid，属性值)构成的属性值集；另一种是由二元组(Oid，Oid)构成的关系集，它们分别存贮对象的属性值和对象与其它对象之间的关系；然后，对两种数据子集基于Oid进行水平划分，得到混合式数据子集。混合式数据子集由若干个分段的属性值集和关系集组成，同一对象的所有数据位于同一混合式数据子集中。它能避免访问大对象时所带来的大量不必要的I/O请求，从而提高数据访问的速度，而且能为开发各种并行性提供有力支持。基于对象类的数据分配策略[5,6]依据数据放置原则的第三条，对混合式数据划分策略的结果进行合理分配。

五、基于合格标记的数据操作并行执行算法

为了开发操作内的并行性，POODB的研究者提出了基于传统数据放置策略的并行数据操作算法[1~4]，为了便于区分，我们称其为传统的并行数据操作算法。虽然传统的并行数据操作算法比较有效地开发了操作内的并行性，但是，对OODB本质特点的考虑不够充分，而且磁盘I/O量和网络传输量都较大。针对传统并行数据操作算法的不足，提出了基于合格

标记的数据操作并行执行算法，它与传统并行数据操作算法的主要区别是：利用"合格标记"来记录数据操作的结果，而不是利用数据拷贝来记录操作的结果。

对于选择操作来说，"合格"意味着对象满足选择条件，那么，如何有效地进行"合格标记"呢？在一个由N个处理机结点通过高速互联网连接的无共享结构中，每个结点的磁盘中都有一张用于记录对象的"合格"标记的表格——标记表，存储于该结点的磁盘上的每一个对象都与标记表中的一项相对应，标记表中的每一项由对象的Oid和相应的标记构成。这个标记表由若干个子表构成，每个子表对应于存储在该结点的磁盘上的一个混合式数据子集。

从查询优化的角度考虑，在查询处理时，优先进行选择操作和投影操作，所以我们对连接操作的讨论也是在选择操作和投影操作的基础上进行的。对于连接操作来说，"合格"不仅意味着对象满足选择条件，而且对象之间存在一定的关系。所以，我们利用由两个对象的Oid构成的二元组(Oid,Oid)作为连接操作的"合格标记"。我们称这样的二元组为关联模式，所有的关联模式构成关联模式集，它也象标记表一样分布于各个处理机结点上。

六、基于semi-join的并行查询处理算法

6.1 基本思想

多元连接查询的执行效率直接反映了数据库系统的性能，并行数据库系统的许多研究工作都是围绕着多元连接查询的并行执行进行的。

多元连接查询的并行执行可以分为两步：1)对象匹配：利用并行查询处理算法得到查询结果，这些查询结果分布于各个处理机结点上；2)结果回收：回收分布于各结点上的查询结果，得到最终完整的查询结果。

现有的并行面向对象数据库(POODB)查询处理算法[1~3]大多是以基于树的查询执行计划模型为基础的，而且在算法执行过程中产生大量的中间结果，从而导致大量的系统开销，抵销了并行性带来的效率提高，以致影响POODB的性能。为便于区别，我们称现有的POODB查询处理算法为传统的并行查询处理算法。传统的并行查询处理算法以连接操作为基础，通过宏观控制各连接操作的执行次序来实现查询的有效处理。它的基本思想可以归结如下：

(设在查询图中，类C的相邻类数目为AN(C)，与其进行过连接操作的相邻类数目为JAN(C))
依次对每个类进行如下处理，直到所有类的AN等于JAN：
·IF (AN(C)－JAN(C))==1
  THEN (设类D是类C的相邻类，且类C没有与类D进行过连接操作)
    类C与相邻类D进行连接操作，即：C×D；
    JAN(C)++；
    JAN(D)++；

针对传统并行查询处理算法的不足，我们采用基于图的查询执行计划模型，在分阶段执行策略、基于对象类的混合式数据放置策略和基于合格标记的数据操作并行执行算法的基础上，进一步将一个连接操作转化成两个semi-join操作，并通过宏观控制所有semi-join操作的执行次序来实现查询的有效处理，这就是我们提出的基于semi-join的并行查询处理算法。

semi-join操作C→D的定义如下：给定类C和类D，找出所有满足下列条件的对象DO_i：对象二元组(CO_i,DO_i)满足连接条件，其中，CO_i和CO_i分别表示类C和类D中的对象。

在基于semi-join的并行查询处理算法中，semi-join操作的执行也是基于合格标记的，semi-join操作的执行算法与前面介绍的基本数据操作的执行算法相似。

从查询优化的角度考虑，在查询处理时，优先进行选择操作和投影操作，所以我们对连接操作的讨论也是在选择操作和投影操作的基础上进行的。对于连接操作来说，"合格"不仅意味着对象满足选择条件，而且对象之间存在一定的关系。所以，我们利用由两个对象的Oid构成的二元组(Oid,Oid)作为连接操作的"合格标记"，我们称这样的二元组为关联模式，所有的关联模式构成关联模式集，它也象标记表一样分布于各个处理机结点上。对于多元连接查询来说，用作合格标记的关联模式应为对象多元组(Oid,…,Oid)，它由查询涉及的各类对象的Oid构成，随着查询处理的不断进行，关联模式中的Oid数目也由2增加至Num_Of_Class(Num_Of_Class为查询涉及的类的数目)。

基于semi-join的并行查询处理算法的基本思想是：

(设在查询图中，类C的相邻类数目为AN(C)，类C向JANS(C)个相邻类发送过消息并与它们进行semi-join操作，JANR(C)个相邻类向类C发送过消息并与类C进行semi-join操作)
·IF (AN(C)－JANR(C))==1
  THEN
  (设类D是类C的相邻类，且类D没有与类C进行semi-join操作，即：D→C)
    类C向相邻类D发送消息并与其进行semi-join操作(即：C→D)，其目的是进一步判定D类对象的合格性，更新D类的关联模式集；
    JANS(C)++；
    JANR(D)++；
·IF AN(C)==JANR(C) AND JANS(C)==0
  THEN 类C向所有相邻类发送消息并与它们进行semi-join；
  (即：
  FOR 类C的每一个相邻类D
    C向D发送消息并与其进行semi-join操作(C→D)，其目的是进一步判定D类对象的合格性，更新D类的关联模式集；

```
        JANS(C)++;
        JANR(D)++;
    ENDFOR)
·IF AN(C)==JANR(C) AND JANS(C)==1
THEN(说明类 C 仅向一个相邻类发送过消息并与
其进行了 semi-join 操作)
    类 C 向所有其它相邻类发送消息并与它们进
    行 semi-join 操作;
    (即:
    FOR  类 C 的每一个相邻类 D 且 C 未与其进
    行过 semi-join 操作
      C 向 D 发送消息并与其进行 semi-join 操作
      (C→D),其目的是进一步判定 D 类对象的
      合格性,更新 D 类的关联模式集;
        JANS(C)++;
        JANR(D)++;
    ENDFOR)
```

6.2 正确性证明

下面,证明基于 semi-join 的并行查询处理算法的正确性。

定理1 给定一个查询 Q 和一个数据库 D,存在唯一的前期结果图 OG_{res}。

证明:假设存在两个前期结果图 OG_{res-1} 和 OG_{res-2}。根据2.3节中的定义,OG_{res-1} 和 OG_{res-2} 均为所有合格子图的并集。

$\therefore OG_{res-1} \subseteq OG_{res-2}$ 且 $OG_{res-2} \subseteq OG_{res-1}$。

$\therefore OG_{res-1} = OG_{res-2} = OG_{res}$。

故给定一个查询 Q 和一个数据库 D,存在唯一的前期结果图 OG_{res}。证毕。

定理2 对于任意的查询 Q 和数据库 D,若查询图是树型结构的,那么算法能产生查询 Q 的前期结果图。

证明:首先,证明算法的可终止性。假设在 QG_Q 中有 m 个类,在 OG_Q 中有 n 个对象。根据算法,每个类将在最多得到(m-1)个来自其相邻类的消息之后完成所有的处理,也就是说,每个对象至多被处理(m-1)次。所以,算法在 O(mn)步内将完成所有处理并产生输出结果图 OG_{out}。

其次,证明算法能为一个树型结构的查询产生一个正确的前期结果图,即:$OG_{out} = OG_{res}$,由定理1所示,前期结果图 $OG_{res}(O_{res}, E_{res})$ 是 Q 的唯一解。

算法的输出结果图 $OG_{out}(O_{out}, E_{out})$ 是由合格对象和它们之间的边构成的集合。OG_Q 中的某个对象被标记为合格对象,是因为它被来自所有相邻类的消息所标记。所以,OG_Q 中的任一合格对象 O_{ip} 与其每个相邻类中的至少一个对象相关联。

根据算法,与 O_{ip} 相关联的对象 O_{jq} 必须满足下列条件之一:(1)C_j 接收到除 C_i 之外的其它所有相邻类的消息,O_{jq} 被来自除 C_i 之外的所有相邻类的消息标记;(2)O_{jq} 是一个合格对象。若在条件(1)下 O_{jq} 还不是最终合格对象,那么算法将用来自 C_i 的消息标记 O_{jq} 并使其最终成为合格对象。因为 O_{ip} 和 O_{jq} 均为合格对象,所以边 (O_{ip}, O_{jq}) 也将被包括在 OG_{out} 中。所以 OG_{out} 中任一对象 O_{ip} 与 C_i 的每一个相邻类的至少一个对象相关联。在 OG_{out} 中,可以从 O_{ip} 开始遍历并且从每一个相邻类中找到一个相关联的对象,从而形成一棵树,产生的这棵树是 OG_Q 的一个子图,并且可以象第2节中定义的那样映射到查询图 QG_Q 上。这样 OG_{out} 中的每一个对象也就在 OG_{res} 中。同理,可以证明 OG_{out} 中的每一条边也是 OG_{res} 中的边。

所以,$OG_{out} \subseteq OG_{res}$。

现在,假设 $O_{ip} \in O_{res}$ 且 $O_{ip} \notin O_{out}$。

由 $O_{ip} \in O_{res}$,$\therefore$ 存在 $OG_{subl} \subseteq OG_Q$,$O_{ip} \in O_{subl}$ 且 OG_{subl} 能被映射到 QG_Q 上。

若将算法单独作用于 OG_{subl},则 O_{ip} 将被标记为合格对象。这与假设矛盾。

$\therefore OG_{res}$ 中的每一个对象也都在 OG_{out} 中,同理可证 OG_{res} 中的每一条边也在 OG_{out} 中。

$\therefore OG_{out} = OG_{res}$。

所以,算法能为一个树型结构的查询产生一个正确的前期结果图。证毕。

6.3 算法的改进

分析算法的执行过程,可以发现:

1)算法结束后,各类的最终关联模式集是相同的;

2)当类 C 收到来自其所有相邻类的消息后(即:AN(C)==JANR(C)),就已经得到了类 C 的最终合格对象和反映查询结果的最终关联模式集。

所以,当 AN(C)==JANR(C)时,类 C 与其相邻类的 semi-join 操作可以简化,即:不需进行繁琐的建表操作和探询操作,只需利用类 C 的关联模式集去更新其相邻类的关联模式集,简化后的 semi-join 操作被称为伪 semi-join 操作。本文的后续部分将详细介绍 semi-join 操作和伪 semi-join 操作的执行算法。

鉴于上述考虑,对基于 semi-join 的并行查询处理算法进行改进,改进后的算法的基本思想如下:

```
·IF (AN(C)-JANR(C))==1
  THEN
  (设类 D 是类 C 的相邻类,且类 D 没有与类 C 进行
  semi-join 操作,即:D→C)
      类 C 向相邻类 D 发送消息并与其进行 semi-
      join 操作(即:C→D),其目的是进一步判定 D
      类对象的合格性,更新 D 类的关联模式集;
          JANS(C)++;
          JANR(D)++;
·IF AN(C)==JANR(C) AND JANS(C)==0
  THEN 类 C 向所有相邻类发送消息并与它们进行
  伪 semi-join 操作;
      (即:
        FOR  类 C 的每一个相邻类 D
        C 向 D 发送消息并与其进行伪 semi-join 操
        作(C→D),其目的是进一步判定 D 类对象
        的合格性,更新 D 类的关联模式集;
```

```
        JANS(C)++;
        JANR(D)++;
    NEDFOR)
IF AN(C)==JANR(C) AND JANS(C)==I
  THEN  (说明类C仅向一个相邻类发送过消息并
  与其进行了 semi-join 操作)
      类C向所有其它相邻类发送消息并与它们进
      行伪 semi-join 操作;
      (即:
        FOR 类C的每一个相邻类D且C未与其进
        行过 semi-join 操作)
        C向D发送消息并与其进行伪 semi-join 操
        作(C→D),其目的是进一步判定D类对象
        的合格性,更新D类的关联模式集;
          JANS(C)++;
          JANR(D)++
        ENDFOR)
```

6.4 性能评价

多元连接查询的并行执行包括对象匹配和结果回收两部分,在对象匹配中采用的并行查询处理算法不同,相应的结果回收算法也不同。所以,为了对基于 semi-join 的并行查询处理算法进行较为全面的评价,我们对查询的整个并行执行过程进行了性能分析,既考虑了对象匹配也考虑了结果回收。由于实验环境等客观条件的限制,我们采用理论分析与模拟实验相结合的方法来对比分析基于 semi-join 的并行查询处理算法与传统的并行查询处理算法的性能。由于篇幅的限制,这里只给出性能评价的结果,更细致的算法描述和详细的代价公式推导参见文[5]。我们模拟的工作环境是:多台 SGI Challenge 服务器(MIPS R4400芯片,128MIPS)通过网络互联,网络启动时间为0.05ms,传输率为75Mb/s。

通过改变重要参数(结点数目 N、Master 结点上的磁盘数目 size_of_disk_array、每个类所含大性质数目 m、大性质的大小 S_big)的取值,观察两种算法的执行时间比值 ratio1(传统的并行查询处理算法的执行时间/基于 semi-join 的并行查询处理算法的执行时间)和两种方法的执行时间比值 ratio2(传统方法的执行时间/基于 semi-join 的方法的执行时间)的变化。

从模拟测试结果可以看出,在综合考虑多元连接查询的并行执行的整个过程中,基于 semi-join 的方法优于传统方法,并且当类的大性质数目越多,大性质占用的存储空间越多,基于 semi-join 的方法就越优于传统方法(反映在方法的执行时间比值 ratio2越大)。但是,ratio2的值明显小于 ratio1的值,这是因为两种方法的结果回收时间相当且在两种方法的总执行时间中都占有较大的份额(尤其是基于 semi-join 的方法),也就是说,结果回收已成为基于 semi-join 的方法的"瓶颈"部分。基于 semi-join 的并行查询处理算法对查询并行执行的对象匹配阶段进行了改进,查询并行执行的结果回收阶段也需要有所改进才能更大地提高查询并行执行的效率。结果回收阶段的改进可以从服务器端的工作模式入手,将主从模式改为端对端(peer-to-peer)模式。在端对端模式中,我们可以选择不同的结点来完成不同查询的结果回收工作,这样就可以在一定程度上解决结果回收的"瓶颈"问题。

结论 分阶段执行策略通过把对大量描述数据的检索推迟到确定"合格"的对象之后,从而有效地避免了对大量不必要数据的访问和处理;而且,通过对数据库中符合条件的对象进行标记来确定"合格"的对象,从而避免了产生大量的中间结果,有效地减少了处理和存储中间结果所需的时间和空间。

基于对象类的混合式数据放置策略能避免访问大对象时所带来的大量不必要的 I/O 请求,从而提高了数据访问的速度,而且能为开发各种并行性提供有力支持。

基于合格标记的数据操作并行执行算法利用"合格标记"来记录数据操作的结果,而不是利用数据拷贝来记录操作的结果,从而有效地降低了磁盘 I/O 量和网络传输量。它比较依赖于最初的数据放置,而且不涉及结果数据的放置,这不仅减少了对后续的数据操作的影响,而且减少了数据偏斜产生的可能性。

基于 semi-join 的并行查询处理算法将传统算法中的连接操作分解为两个 semi-join 操作,通过对它们的宏观控制有效地实现查询的并行处理。通过两个定理对该算法的正确性进行了证明。

总之,分阶段执行策略、基于对象类的混合式数据放置策略、基于合格标记的数据操作并行执行算法和基于 semi-join 的并行查询处理算法都是针对 OODB 及其查询的本质特点提出的,具有一定的实用性和有效性。

参考文献

1 Haddleton R F. An Implementation of a Parallel Object Oriented Database System:[Technical Report]. University of Virginia,1995

2 Bassiliades N, Vlahavas I. Hierarchical query execution in a parallel object-oriented database system. Parallel Computing,1996,22(3):1017～1048

3 Gesman M. Mapping a Parallel Complex-Ojbect DBMS to Operating System Processes. In: Proc. of the Second Interl. Euro-Par Conf. Euro-Par' 96. 1996. 852～861

4 Lieuwen D F, et al. Parallel Pointer-based Join Techniques for Object-Oriented Databases. In: Proc. of the Second Int' l Conf. on Parallel and Distributed Information Systems. 1993. 172～181

5 王意洁. 面向对象数据库的并行查询处理与事务管理:[博士学位论文]. 国防科技大学研究生院,1998年10月

6 王意洁,王勇军,胡守仁. 并行面向对象数据库中的基于对象类的混合式数据放置策略. 计算机学报,1998,21(增刊):178～183

7 Cattell R, et al. The Object Database Standard. ODMG-93. Morgan Kanfmann,1994