文章编号: 1001-2486(2000) 05-0068-05

一个基于引用的高效连接算法*

阳国贵, 吴泉源

(国防科技大学计算机学院, 湖南 长沙 410073)

摘 要: 针对对象关系数据模型和查询语言的新特点, 提出了一个基于引用的高效连接算法 Sort-Loop。引用既是对象关系数据模型中一种重要的建模设施, 同时它也有利于连接算法的设计和高效实现, 如 Hash-Loops 就是基于指针、面向集合属性的连接算法。Sort-Loop 克服了 Hash-Loops 算法在数据访问方式和内存使用上存在的不足, 性能分析表明, 其性能优于 Hash-Loops。

关键词: 连接算法; 对象关系数据库; 算法分析

中图分类号: TP392 文献标识码: A

A Reference Based on High Performance Joining Algorithm

YANG Gue-gui, WU Quan-yuan

(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Based on the features of ORDM(Object-Relational Data Model) and the query language, a new reference based on join algorithm Sort-Loop is represented. Reference is not only one of the important modeling mechanism, but also beneficial to the design and implementation of joining algorithms, such as Hash-Loops. By overcoming the shortcomings of the Hash-Loops in the data access manner and the use of the main memory, Sort-Loop is of high performance. This is validated by the analysis in the paper.

Key words: join algorithm; ORDB; algorithm analysis

对象关系数据库技术具有管理复杂数据的能力, 提供了创建复杂结构类型的类型构造子, 如组合、集合和引用(Reference)等[1], 在查询方面, 将是 SQL 语言的一种自然发展, 如目前已经出现的 SQL: 1999。数据模型和查询语言的新特点, 将使得对象关系数据库系统的实现技术更为复杂, 在连接算法方面, 由于引用的引入, 需要研究新的基于引用的连接算法, 才能有效利用这一重要的建模设施。

在传统的关系数据库中, 关系表的联系靠码来进行, 或由查询语句中的连接条件临时确定, 但不管是哪种方式, 都是基于值的。当对关系表进行连接时, 由于是基于值的, 就必须在整个关系表中寻找, 或建立索引, 先通过索引找到查找范围, 然后在受限范围中进行查找。在关系数据库中, 连接运算十分费时, 为此, 对连接算法的研究也就十分重视, 研究出了许多的连接算法, 如嵌套循环、带索引的嵌套循环、排序归并、Hash Join、Hybrid Hash Join 等[2]。

对象关系数据库中的引用是传统 SQL 系统中主码-外码关系的自然替代, 在对象关系表中, 某列可以是引用类型, 用于指向某个类型的对象。同样, 关系表中的某列也可以是集合类型, 而集合的成员是引用类型。如在 Illustra 对象关系数据库中, 表中的每一行都有一个唯一的标识, 称为对象标识符(OID), 它是一个 64 位的唯一标识, 由系统赋予, 并且一经赋予就不再改变, OID 的组成包括两个部分: 其中的 24 位作为表标识, 另外的 40 位为表中的行标识。实现上, 引用类型列中所存储的就是所指对象的 OID。同样, Oracle8.x 对象关系数据库支持嵌套对象、可变数组、复杂结构类型, 关系表中的元组具有 ROWID, 该 ROWID 包含了元组所在的数据文件、块和块中的行序列等存储信息。

在面向对象数据库(如 Gemstone、ObjectStore、ORION、O++、O2 等)中, 尽管允许集合属性和对对象的直接引用, 但由于早期的面向对象数据库系统并不提供类似于 SQL 的查询语言接口, 仅提供导航式对象访问和操作方式, 所以, 对象间的关联访问(或连接)运算没有得到应有的重视。在以后的发展中,

* 收稿日期: 2000-02-1
基金项目: 国家部委基金项目资助(98J15.2.5.KG0133)
作者简介: 阳国贵(1964-), 男, 副教授, 硕士。

逐渐认识到了面向对象数据库系统也同样需要提供强有力的查询语言功能, 在 ODMG－93 中就定义了支持对象查询的 OQL 语言。

当对象的 OID 带上存储位置信息(如所在的物理页面) 时, 就可根据对 OID 的引用直接找到该对象了。为此, 利用该 OID 特点的一些连接算法已被提出, 如基于指针(pointer-based) 的 Hash-Loops 连接算法和 Hybrid-Hash 算法等[3]。一般地, 这些算法可以移植到对象关系数据库中。对 Hash-Loops 连接算法而言, 它对内表的访问不是按物理指针的顺序进行的, 这样, 将导致数据读取时磁头来回频繁移动, 使算法性能受到严重影响, 据此, 本文提出一种新的基于引用的连接算法 Sort-Loop, 该算法克服了上述不足, 比 Hash-Loops 具有更好的性能。

1 问题与算法

1.1 基本问题

假如在某公司的人事数据库中, 有岗位情况(工作) 表, 部门关系表 dept 和职员关系表 emp, 其中岗位情况表、部门表和职员表的元组类型分别为 job_t、dept_t 和 employee_t, 其结构定义分别为:

```
create type job_t( name varchar(30), skill_required varchar(50), …);
create table job of type job_t;
create type employee_t(name varchar(30), job_ref ref(job), salary int,
                       age int, state char(2), zipcode int);
create table emp of type employee_t;
create type dept_t (dname varchar(30), phone phone_t,
                    manager_ref ref(employee_t), workers setof ( ref (employee_t ) ) );
create table dept of type dept_t;
```

可以看到, employee_t 中的 job_ref、dept_t 中的 manager_ref 均为引用类型, 指向类型为 job_t 和 employee_t 的关系表中的元组, 而 workers 为集合类型, 集合成员为引用类型。

当查找职员表中年龄在 30 岁以下的职员及其工作情况时, 可以用如下的查询语句:

```
select name, age, job_ref.* from emp where age< 30;                    (1)
```

在 dept 表中, workers 属性是集合属性, 其集合成员为引用类型, 当连接属性为集合属性时, 如何高效求解该类查询? 如查找鞋帽部中工资高于其经理, 且起始工作日期在 95 年 10 月 1 日后的员工时, 套用 OQL 的写法, 可以表述为:

```
select distinct struct(name: x.name, employees :(select y
                                                 from y in x.workers
                                                 where y.salary> x.manager_ref-> salary
                                                 and y.startdate> '1/10/95'))          (2)
from x in dept;
```

更一般地, 我们可把上述查询问题归纳为:

```
for ( x1 of Set1; x2 of x1-> set ) suchthat ( Pred1 (x1,x2) ) S1。                    (3)
```

上式中, Pred1 为 x1, x2 应当满足的条件谓词, 而 S1 是语句中的其他部分, 如做属性投影运算等。下面将较详细地介绍 Sort-Loops 算法, 并且假定引用中包含了所指对象的物理位置信息, 如块号 PID (page identifier)。为了便于分析、比较, 对 Hash-Loops 算法、Probe-Loops 算法也将做简要说明。

1.2 Sort-Loops 算法

(3) 式中的 Set1 称为外连接关系, 而 x2 所在的关系称为内连接关系, 则 Sort-Loops 算法的基本思想是, 当外连接关系 Set1 在内存中放不下时, 分成若干遍进行, 每遍尽可能多地读入 Set1 中的元组(当算法可用内存为 M 块时, 为读内连接关系表 Set2 留出一块, 即在每遍中可读入 Set1 中的 M-1 块到内存。), 对每遍中所读入的 Set1 中的元组, 形成一组排序元素(Pid, Pointer), Pointer 为该元组在内存中的位置指针, 而 Pid 即为(3) 中各 X2(即引用) 中的 PID 部分(引用所指向的块号), 并按 Pid 对这些排序元素进行

排序, 对同一遍中所有的排序元素排序之后(具有相同 PID 的 Pointer 可放在一起) , 将按排序次序依次读取 Pid 所标识的数据块。当该块被读入后, 依据 Pointer 指针, 找到相应的 Set1 中的元组, 检查相应的元组对(X1, X2) 是否满足连接谓词 Pred1。若满足, 则进行 S1 语句的处理, 获得一个结果元组。当完成该遍中所有元组的处理工作后, 可开始下一遍的工作, 直至把 Set1 中的所有元组处理完毕。下面给出上述算法的伪代码描述:

```
Sort-Loops-algorithm ( )
    { while ( not eof ( Set1) ) do
        { phase1: Do get_ next_ page_ set1 ( pbuffer ) ;
                                /* get next page of Set1 Sequentially */
                  while ( not_ end_ of (pbuffer) )
                      {   object= get_ next_ object( ) ;
                          pointer= insert_ into_ mem_ table( object ) ;
                          pids= extract_ ref_ pid ( object ) ;
                          for each pid of pids
                              do insert_ into_ sort_ table( pid, pointer ) ; }
                  Until ( available memory reached threshold ) ;
          phase2: for ( i= 0; i++ ; i< Max_ Sort_ Table_ Length)
                  {       sort_ table_ entry= get_ next_ sort_ table_ entry( i ) ;
                          seq_ read_ page_ of_ set2 (sort_ table_ entry. pid, buffer) ;
                          set1_ objects= sort_ table_ entry. pointer;
                          for each object1 of set1_ objects
                                                  do join ( buffer, object1) ; }
        } /*  end of while */
    } /*  end of the algorithm */
```

在上述描述中, 外循环 while 的执行次数即为算法将外关系划分为多段执行时的遍数, 在每遍的 phase1 阶段, 依次读入外关系 Set1 的数据页面, 对页面中的每个元组而言, 先将该元组放到内存中的适当位置(由 insert_ into_ mem_ table 完成) , 然后从该元组的连接集合属性中获取指向 Set2 的引用, 并从中抽取出物理页面号 Pid, 把排序元素(Pid, Pointer) 插入到按 Pid 排序的排序表中(由 insert_ into_ sort_ table 过程完成) 。

需要特别指出的是, insert_ into_ sort_ table 过程的实现方式将直接影响到算法的性能, 在内存中实现排序的方法很多, 如基于优先队列(Priority Queue) 。但基于 Sort-Loops 算法中的 Pid 范围是已知的, 即关系 Set2 的页面范围, 据此, 可以预先分配一个排序表, 使得表项与 Pid 一一对应。在下面的算法分析中, 对单个引用属性的情况, 我们采用基于优先队列的排序算法, 而对集合引用属性的情况, 采用第二种方式, 以少量空间换取时间。

1.3 Hash-Loops 算法

当外连接关系 Set1 在内存中放不下时, Hash-Loops 算法同样分成若干遍进行, 每遍尽可能多地读入 Set1 中的元组, 对每遍中所读入的 Set1 中的元组, 形成一组元素(Pid, Pointer) , Pointer 为该元组在内存中的位置指针, 而 Pid 即为(3) 中各 X2(即引用) 中的 PID 部分(引用所指向的块号) , 并根据某个哈希函数对 Pid 进行计算, 由哈希值确定这些元素所在的位置, 完成同一遍中所有元素的处理并形成哈希表之后(具有相同 PID 的 Pointer 可放在一起) , 将依次按照哈希表中的 PID 读取相应的数据页面, 依据 Pointer 指针, 找到相应的 Set1 中的元组, 检查相应的元组对(X1, X2) 是否满足连接谓词 Pred1, 若满足, 则进行 S1 语句的处理, 获得一个结果元组, 当完成该遍中所有元组的处理工作后, 可开始下一遍的工作, 直至把 Set1 中的所有元组处理完毕。

1.4 Probe-Loops 算法

Probe-Loops 算法将把 M 块内存中的 M- 1 块用作保存 Set2 中的数据页面的缓冲区, 算法的基本思路是: 先从 Set1 中读入一个数据页面, 对该页面中的每个元组, 根据其连接集合属性, 找到各引用值, 对每个引用值, 先根据其 PID 值在内存中寻找相应的数据页面, 当该页面在内存时, 即可完成相应的处理工作。在内存找不到时, 将把 Set2 中的相应数据页面读入内存, 当内存有自由空间时, 将相应页面读入即可, 否则, 可按某种淘汰算法, 先选择将被覆盖的页面, 然后将相应页面读入。

2 性能分析与对比

假设 Set1 中的每个元组对 Set2 中元组的引用是均匀分布的, 且每个 Set1 元组中包含的引用个数的平均数为 m 个, 而一个 Set2 元组平均被 Set1 中的 n 个元组所引用。机器速度为 Mips, 执行比较、交换操作的时间分别为 compare 和 swap, 而排序元素的插入时间为 move, 完成一次磁盘读或写操作的时间为 IO, 内存页面数、磁盘页面大小为 P, |Set1|、||Set1|| 分别表示 Set1 的页面数和元组数, |Set2|、||Set2|| 分别表示 Set2 的页面数和元组数, 表示块号 Pid 所需的字节数为 K, 而在内存中指向一个 Set1 元组的指针所需的字节数为 L, Set1 和 Set2 中元组的大小为 Oset1 和 Oset2, 哈希表的膨胀因子为 F 等[2]。

当参数设置为: Mips= 100000000; IO= 0.002; P= 4096; ||Set1||= 100000; ||Set2||= 100000; n= 1; m= 1; Oset1= 400, Oset2= 100; F= 1.2; 不考虑磁盘顺序读取和随机读取的差异, 且采用基于优先队列的排序方法时的性能对比为下图 1 所示。而图 2 中的 IO= 0.02; Oset2= 400, 其余参数和假设不变。

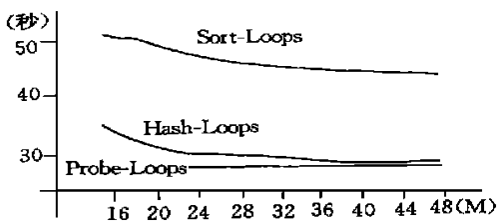

图 1 对比分析之一

Fig. 1 Result one of comparison

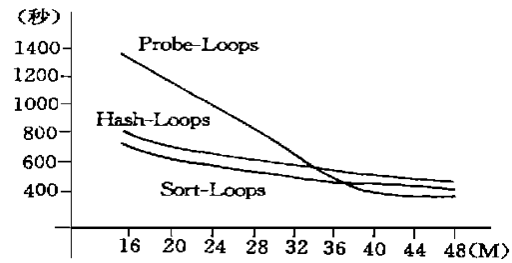

图 2 对比分析之二

Fig. 2 Reslt two of comparison

从上可以看出, 当关系 Set2 较小, 可以全部放入内存时, Probe-Loops 算法的性能将好于 Sort-Loops 与 Hash-Loops, 但当 Set2 较大, 在内存放不下时, 性能有些下降, 见图 2。由于采用基于优先队列的排序方法, 故排序时间不容忽视, 导致图 1 中 Sort-Loops 的性能不如 Hash-Loops。但当降低 I/O 速度时(相当内存排序时间降为次要因素时), Sort-Loops 的性能好于 Hash-Loops, 如图 2。而当采用预留排序表时, Sort-Loops 所花的排序时间进一步降低, 将表现出更好的性能。

下面的分析主要针对集合引用属性, 参数设置为: IO= 0.02; n= 50; m= 50; Oset1= 400, Oset2= 100; 其余不变, 但采用预留排序表方式(图 3)。当 ||Set2||= 100000; M= 10; n= 100, 其余参数和假设不变时, 性能分析结果见图 4。

从上可以看出, Sort-Loops 算法采用预留排序表空间方式实现集合引用属性的连接处理时, 具有很好的性能。

3 总结

对象关系数据库技术已被认为是未来数据库技术发展的大趋势[4], 一些厂商也积极推出初步具有对象关系数据模型特点的数据库产品, 如 OracleV8.0、Illustra、IBM 公司的 UDB 等。尽管如此, 对象关系数据库实现技术仍有待进一步研究和完善。Hash-Loops 是在面向对象数据库环境下提出来的, 是基于指针的、面向集属性的连接算法。对象关系数据库中提供了引用机制, 它既是数据建模的一种有效手

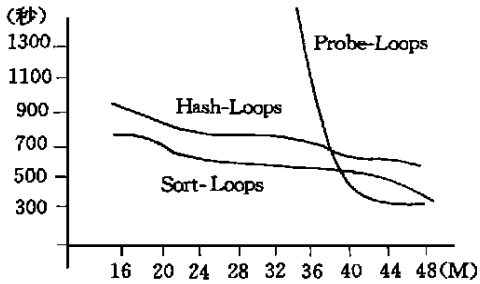

图 3 对比分析之三

Fig. 3 Result three of comparison

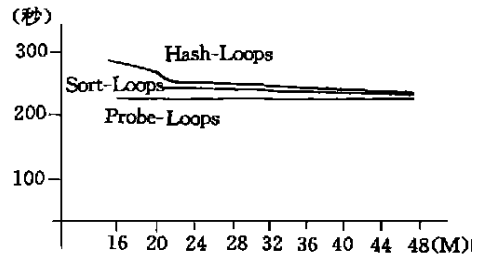

图 4 对比分析之四

Fig. 4 Result four of comparison

段, 也为某些连接运算的高效实现提供了支持, 一般地, 可在引用的表示中带上物理页面信息, 此时, Hash-Loops 原则上可移植到对象关系数据库中。但 Hash-Loops 具有两个主要的不足: 对数据的访问难以按物理地址顺序进行, 造成数据访问的低效; 另外, 哈希办法也将造成一定的空间浪费。为此, 本文提出了 Sort-Loop 算法, 它克服了 Hash-Loops 算法的上述不足。通过分析表明, Sort-Loop 算法在性能上优于 Hash-Loops 算法。Probe-Loops 算法在 Set2 能放入内存时才具有好的性能, 否则, 性能很差。

参考文献:

[1] 阳国贵等. 对象关系数据库系统与技术[J]. 计算机科学, 1998, 25(6).

[2] Shapiro L D. Join Processing in Database Systems With Large Main Memories[J]. ACM TODS. 1986, 11(3): 239-264.

[3] Shekita E. and Carey M J. A Performance Evaluation of Pointer-Based Joins. Proc. 1990 ACM SIGMOD, June 1990.

[4] Stonebraker M, et al. Objete-Relational DBMSs: The Next Great Wave [M]. Morgan Kaufmann Publishers, Inc. 1996.