文章编号:1001 - 9081(2005)02 - 0279 - 04

一种实时数据库查询执行方法的设计

刘云生,彭楚冀,廖国琼
(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)
(pcj_1976@ hotmail. com)

摘 要:在深入分析实时数据库常用的查询执行方法——指针法缺点基础上,给出了一种新的实时数据库查询执行方法——D/S 方法。该方法结合了指针法、实体化方法和流水线方法的优点,可有效地节省查询执行的内存需求,并扩展了实时数据库查询优化的空间。

关键词:实时数据库;查询执行;实体化;流水线
中图分类号: TP311.13 **文献标识码**:A

Design of a query execution method for real-time database

LIU Yun-sheng, PENG Chu-ji, LIAO Guo-qiong
(School of Computer Science and Technology, Huazhong University of Science & Technology, Wuhan Hubei 430074, China)

Abstract: In this paper, we analysed the weakness of the general query execution method, point method in real-time database, and then presented a flexible one, the D/S method which contains the virtues of materialization method, stream processing method and point method. Using the new method, the demand of memory decline effectively, and the space of query optimization will be enlarged.

Key words: real-time database; query execution; materialization method; stream processing method

0 引言

在实时数据库中,查询处理的设计目标是 CPU 时间和内存空间的高效使用,为此人们提出了一种与一般数据库查询执行方法不同的一种方法——指针法[1],它的优点是查询执行中间结果是一个指针集,很少占用内存,访问效率高,能够较好地满足查询处理的设计目标。但它也存在一些缺点,如投影通常必须延后执行、多次连接后元组形式复杂等;甚至在一定条件下,它占用的内存比普通的查询执行方法更多,从而违背了设计的初衷。

因此,本文将深入分析指针法缺点,并相应地结合传统数据库查询执行方法——实体化和流水线,设计一种新的实时数据库查询执行方法——D/S 方法,可有效地减少查询执行占用的内存空间,且占用的 CPU 时间跟指针法基本相当,从而更好地满足实时数据库查询处理的设计目标。

1 实时数据库现有典型查询执行方法

1.1 指针法分析

设 RTDB 是一个实时数据库,student、department 是 RTDB 中的两个基本关系,关系的元组如表 1、表 2 所示。注意表中每个关系中的 Ptr 属性都是伪属性,属性的值为该元组的起始地址。实际上,关系并不包含 Ptr 属性,这里为了使读者更容易理解,我们加入了 Ptr 属性。本文的所有范例都使用这种表示方法。

例 1 获取年龄不小于 24 岁的计算机系学生的信息,对应的 SQL 语句为:

```
select student. * from student, department
where 年龄 > =24 and student. 系号 = department. 系号 and 系名
  ='计算机';
```

表 1 Student

姓名	年龄	学号	系号	Ptr
贾明	22	100	1000	1221
刘明	24	101	1004	1241
周涛	23	102	1002	1261
彭浩	25	103	1004	1281
汪健	26	105	1004	1301

表 2 department

系名	系号	Ptr
数学	1000	2221
物理	1001	2241
化学	1002	2261
计算机	1003	2281
电子	1004	2301

result relation	
Addr	Ptr
1241	400
1281	4004
1301	4008

(a)

result description
student.姓名
student.年龄
student.学号
student.系号

(b)

图 1 操作结果关系

查询结果如图 1 所示,包括两个部分:一个新关系,存储

收稿日期:2004 - 07 - 22 **基金项目**:"十. 五"国防预研(413150403);国防预研基金项目(00J15.3.3.JW0529)

作者简介:刘云生(1940 -),男,教授,博士生导师,主要研究方向:现代数据库理、实时信息系统; 彭楚冀(1976 -),男,硕士研究生,主要研究方向:实时数据库系统; 廖国琼(1969 -),男,博士,主要研究方向:现代数据库技术、移动、主动、实时数据库系统等.

满足条件的元组的指针;另一个用来描述查询结果的模式等信息,限于篇幅,这里我们只显示部分的模式信息。图 1(a)中包含了三条元组,每条元组都只包含一个指针,指向进行选择的关系 student 的某条满足选择条件的元组。

分析例 1,我们可以发现:

1) 指针法的操作结果只是一个指针集,不包含任何实际的属性值,故操作结果很小,因此通常人们认为该方法最节约内存。但是,该论断实际上是以分析单个运算为前提,若从一个完整 SQL 查询的整体角度来考虑,就会发现:由于指针法的中间结果中不包含任何属性值,只包含指向基本关系元组的指针,因此在整个查询语句处理结束之前,所涉及到的基本关系都必须被钉住(pinned),不能被交换出内存。即,指针法在一个查询语句执行过程中所占用的内存大小实际上是操作结果集的大小及所涉及的基本关系大小的和。

假设关系 student、department 的数据量都比较大,关系 student 的选择操作"年龄 > =24"的结果很小、department 的选择操作"系名 = 计算机"的结果也很小。如果该查询在执行完两个关系上的选择操作之后,在执行选择结果间的连接操作时,一个优先级更高的查询进入了数据库系统,由于实时特点,低优先级的事务需要被挂起,于是前面的查询被挂起,且在挂起过程中,student、department 这两个数据量大的关系始终必须被钉在内存中。

根据上述的分析可以得出:在实时数据库中,查询时若使用指针法则必然要求涉及的关系必须被钉在内存中,对于在数据量较大的关系上进行复杂查询的低优先级事务而言,可能会在查询过程中被高优先级的事务所挂起并占用大量的内存,在系统资源紧张时,这些低优先级的事务很可能会成为牺牲品而被杀掉,从而降低整个系统的吞吐量。

实际上,对于例 1,若某个策略可满足:在处理完 student(department)上的选择操作之后,关系 student(department)立即就可以被交换出内存,显然,基于该策略的复杂查询低优先级事务被牺牲的可能性会小很多。

2) 指针法很可能会限制生成更好的执行计划

由于指针法处理投影操作效率低,要求进行查询处理时,尽量把投影操作推迟执行,一般是最后才进行投影操作。而这样很可能会限制生成更好的执行计划。

3) 进行多次连接操作后,元组的表示形式比较复杂。

指针法在表示多次连接产生的元组时采用树的形式,连接的次数越多,树也就更复杂。因此,在表示一条多次连接所产生的元组时其形式比较复杂。

1.2 几种查询执行方法比较

为了解决指针法的上述缺点,我们对传统的实体化方法和流水线方法进行分析,并对三者进行比较。

在实时数据库中,实体化方法的缺点非常突出:极可能占用大量的内存以存储中间结果。由于内存是实时数据库最重要的资源之一,因此通常人们不考虑使用实体化方法。然而实体化方法具有如下的优势:首先,在一定条件下,若一个查询的某个操作使用实体化方法处理,那么一旦该操作处理完毕,被操作的关系继续驻于内存对于后续的处理没有必要:如果被操作的关系是基本关系,那么它可以被立即换出内存;如果是前面某个环节产生的实体化关系,那么它可以被立即释放。另一方面,对于连接操作的结果,实体化的表达形式非常简洁,其复杂程度不受连接次数多少的影响。另外,就处理效率而言,实体化与指针法主要区别在于:对于满足条件的每一条元组,实体化需要一次额外的内存拷贝,连续的元组访问是顺序内存访问;指针法则在读一条元组时需要访问两次内存,连续的元组访问实际上呈现为对内存的多次随机访问。所以,两种方法占用 CPU 的时间差别不大。

对于传统的流水线方法。它的最大优点是:操作产生的中间结果的大小可控制。通过流水线的输出缓冲区我们可以控制某个操作产生的中间结果大小的上限。不过,流水线也具有若干缺点:由于流水线的各个操作是并行的,因此在整个查询处理结束之前,所涉及到的关系很可能必须被一直钉在内存;此外,流水线要求在查询过程中各个流水线环节必须进行频繁的协调,在单 CPU 条件下,若流水线环节较多,则会降低查询处理效率。

实体化、指针法和流水线的主要特点综合比较如表 3 所示。

表 3 实体化、指针法和流水线的主要特点比较

类别	中间结果大小	操作关系在查询过程中必须常驻内存	处理效率	投影操作	多次连接后元组表示形式
指针法	通常很小	是	高	通常最后进行	复杂
实体化	有时会很大	否	高	不必最后进行	简单
流水线	大小可控制	是	单 CPU、SQL 复杂时效率低	不必最后进行	简单

根据上述的分析,在实时数据库中,几种查询执行方法都有各自的优点和缺点,任一种查询执行方法都不能很好地处理所有的情形:处理某些类别 SQL 语句时开销、效率都很好;处理另一些 SQL 语句时可能就不如人意。同时,我们可以发现这些方法的缺点存在互补性,因此有必要把这三种查询执行方法相结合,发挥各自的优点,以获得更好的系统性能。

2 D/S 方法的设计

2.1 查询执行规则的设计

D/S 方法的设计思想是将指针法、实体化以及流水线三种方法有机地结合起来,扬长避短,充分发挥每种查询执行方法的特点。根据这种思想,在处理具体的 SQL 语句时,先分析数据库的动态统计信息并进行估算,然后根据估算结果选择采用指针法、实体化或者流水线方法,以避免上述单纯使用指针法的缺点,降低查询的实际内存需求,并提高系统的吞吐量。D/S 方法的查询执行规则设计如下:

1) 对于包含两个或两个以上较大关系数据量的查询,如果某个操作产生的中间结果比进行操作的关系本身所需的空间(之和)小很多,那么对于该操作使用实体化方法;

2) 若不满足规则 1),则在进行选择、集合并、集合差等

运算时,将指针法和流水线法相结合,以使占用的内存最少;

3) 若不满足 1),则在进行连接、投影操作时将实体化和流水线方法相结合,使得结果关系元组表示形式相对简单,并可通过流水线来控制额外消耗的内存。

实际上,连接、投影运算都没有使用指针法,其他运算可根据不同条件相应地采用实体化方法、指针法或者与流水线相结合。因此,采用上述规则,就既可以避免处理某些查询时大量占用额外的内存,又可以获得较简单的表示形式,并且不需要总是最后才进行投影操作处理,执行计划可以灵活地改变投影操作的位置。

2.2 查询执行的算法

为了采用 D/S 方法实现各种查询操作,需要先进行两个基础工作:对传统的关系定义进行扩展和定义几个航行原语。

2.2.1 实时数据库中关系的表示

实时数据库 RMDB 的基本关系、查询操作产生的中间结果关系可以用巴克斯范式统一表示如下:

```
Rel: : =( <Rid>, <attr_list>, <result_set>, <Rel_type>)
Rid: : =the id of the relation;
attr_list: : = <attr>{, <attr>};
attr: : = a attribute of the relation;
result_set: : = <Tuple_set> | <Point_set>;
Tuple_set: : =the set of the tuples of the relation;
Point_set: : =the set of the points of that each points to a tuple of the
  relation;
Rel_type: : = <origin> | <point> | <materialization>
origin: : = the relation is a base one of the RMDB;
point: : =the relation is a temporary one produced by point method
materialization: : = the relation is a temporary one not produced by
point method
```

由于流水线方法只影响操作结果的产生过程,对关系的表达形式没有影响。因此关系的 Rel_type 不包括流水线方法。

2.2.2 航行原语

令 r 为关系,t 为 r 的元组,ptr 为元组位置指针。

1) Ptr(t)

输入:t,是 r 的元组;

输出:ptr,是一个指针,指向 t 的起始地址;

2) T(ptr, attr_list)

输入:ptr,是一个指针,指向某个元组的起始地址;

　　attr_list,是 r 的模式信息;

输出:t,是 ptr 所指向的 r 的一个元组;

3) TupleSet(r)

输入:r,是关系;

输出:r 的所有元组的集合;

算法:

```
if r. Rel_type. Status! = point
  TupleSet(r) = r. result_set;
Else // r. Rel_type. Status = point
  TupleSet(r) = {t| ptr r. result_set t = T(ptr, r. attr_list)};
```

2.2.3 查询操作实现

1) 选择操作 $\sigma_\theta(r,f)$:

输入:r,是关系

　　θ,是选择条件

　　f,是操作方法

输出:r_3,是临时关系,r_3 所有的所有元组都满足条件 θ;

算法:

```
r3. <attr_list> = r. <attr_list>;
if f = point
{
 r3. Rel_type = piont;
 r3. result_set = {ptr| ∀ptr ∀t ∈ TupleSet(r) θ(t) ∧
   (ptr = Ptr(t))};
}
else if f = materialization
{
 r3. Rel_type = materialization;
 r3. result_set = {t| ∀t ∈ TupleSet(r) ∧ θ(t)};
}
```

2) 笛卡儿积 $r_1 \times r_2$

输入:r_1,是关系;

　　r_2,是关系;

输出:r_3,是 r_1、r_2 的笛卡儿积所产生的临时关系;

算法:

```
r3. <attr_list> = r1. <attr_list> + r2. <attr_list>;
r3. Rel_type = materialization;
r3. result_set = {(t1, t2) | ∀t1 ∀t2 t1 ∈ TupleSet(r1) ∀t2 ∈
  TupleSet(r2)};
```

3) 投影运算 $\prod_{\alpha\beta}(r)$

输入:r,是关系;

　　α,是关系 r 的属性;

　　β,是关系 r 的属性;

输出:r_3,是对 r 进行 α、β 上的投影所产生的临时关系;

算法:

```
r3. <attr_list> = α, β;
r3. Rel_type = materialization;
r3. result_set = {(vα, vβ) | ∀t ∈ TupleSet(r), vα 是 t 对应属性 α
  的值, vβ 是 t 对应属性 β 的值};
```

3 代价分析

3.1 内存开销比较

在一定范围内,查询执行整个过程中每个操作所占用的内存可以使用权 $Wm(o,t)=\int_0^{t_0} M(t)\,dt$ 来衡量,其中 o 表示操作,t 是时间维,$M(t)$ 表示该操作需要的内存数量,t_0 表示该操作执行所需要的时间。$Wm(o,t)$ 越大,表示该操作消耗的内存量越多。

设实时数据库中,一个查询 Q 的执行时间为 T,令 R_1、R_2 表示查询 Q 涉及到的两个关系,且 R_1 的大小为 r_1、R_2 的大小

为 r_2，我们在 Q 中来考虑几种查询操作。

1）对 R_1 进行选择操作 σ_θ

如果使用指针法，由于在 S 执行的整个过程中，R_1 必须常驻内存，因此有 $Wm(\sigma_\theta,t) = \int_0^T M(t)\mathrm{d}t \geqslant r_1 \times T$；

如果使用 D/S 方法，那么我们可把时间 T 分为三段：$[0, t_1]$、$(t_1, t_1 + t_0)$、$(t_1 + t_0, T)$；其中 t_1 表示选择开始执行的时间，$t_1 + t_0$ 时选择操作执行完毕，r_1' 表示产生中间结果的小，相应地有：

$$Wm'(\sigma_\theta,t) = \int_0^T M(t)\mathrm{d}t \leqslant r_1 \times t_1 + (r_1 + r_1') \times t_0 + r_1' \times (T - t_1 - t_0);$$

于是：

$$Wm(\sigma_\theta,t) - Wm'(\sigma_\theta,t) \geqslant r_1 \times (T - t_1 - t_0) - r_1' \times (T - t_1 - t_0) - r_1' \times t_0 = (r_1 - r_1') \times (T - t_1 - t_0) - r_1' \times t_0;$$

如果把 σ_θ 操作提前，一开始就执行该操作，那么 $t_1 = 0$，则：

$$Wm(\sigma_\theta,t) - Wm'(\sigma_\theta,t) \geqslant (r_1 - r_1') \times (T - t_0) - r_1' \times t_0;$$

由于对于大部分选择操作，通常有 $r_1 \geqslant 2 \times r_1'$；并且对于复杂的查询而言，也往往有 $T \geqslant 2 \times t_0$；因此往往有 $Wm(\sigma_\theta,t)$ 的下界大于 $Wm'(\sigma_\theta,t)$ 的上界；实际上，我们在计算 $Wm'(\sigma_\theta,t)$ 的上界时，把它扩大了许多，因为 σ_θ 产生的临时结果在 σ_θ 处理完毕之后，可能在参与下一个操作后就被释放，或者进行一次投影后的内存需求量再一次降低。

根据上述推理，我们可以发现对于大部分选择操作，指针法实际消耗的内存更多。

2）对 R_1 进行投影操作 $\prod_{\alpha,\beta}$

在通常的实时数据库查询执行中，投影操作往往推后、合并；尽管投影操作产生的结果比 R_1 小，但是 R_1 仍然必须处于内存中，因此投影提前意义不大，反而需要消耗更多的内存。

应用 D/S 方法时，同样可把时间 T 分为三段：$[0, t_1]$、$(t_1, t_1 + t_0)$、$(t_1 + t_0, T)$；其中 t_1 表示投影开始执行的时间，$t_1 + t_0$ 时投影操作执行完毕，产生的中间结果大小为 r_1'，相应地有：

$$Wm'(\sigma_\theta,t) = \int_0^T M(t)\mathrm{d}t \leqslant r_1 \times t_1 + (r_1 + r_1') \times t_0 + r_1' \times (T - t_1 - t_0);$$

如果投影操作推后、合并，那么关系 R_1 消耗内存的权：

$$Wm(\sigma_\theta,t) = \int_0^{T-t_0} M(t)\mathrm{d}t \geqslant r_1 \times (T - t_0);$$

$$Wm(\sigma_\theta,t) - Wm'(\sigma_\theta,t) \geqslant r_1 \times (T - t_1 - t_0) - r_1' \times (T - t_1 - t_0) - (r_1 + r_1') \times t_0 = (r - r_1') \times (T - t_1 - t_0) - (r_1 + r_1') \times t_0;$$

我们可以发现只有 r、r_1' 差距比较大，并且 T、t_0 的差距也比较大时投影才能获取较好的效果。

3）对关系 R_1、R_2 进行 θ 连接操作 $R_1 \times_\theta R_2$

由于直接使用实体化方法进行 θ 连接操作很可能产生庞大的中间结果，为了避免这一点，在设计 D/S 方法的查询执行规则时，我们使用结合实体化的流水线方法。流水线方法的并行的特点，决定了连接操作的执行时间接近于整个查询 Q 的执行时间。令使用流水线方法后查询 Q 的执行时间为 T_1，连接操作的输出缓冲区大小为 s_1，那么有：

$$Wm'(\times\theta,t) = \int_0^{T_1} M(t)\mathrm{d}t \leqslant (r_1 + r_2 + s_1) \times T_1;$$

连接时如果使用指针法，有：

$$Wm(\times\theta,t) = \int_0^T M(t)\mathrm{d}t \geqslant (r_1 + r_2) \times T;$$

s_1 是流水线输出结果缓冲区的大小，通常设置较小，因此当 T、T_1 接近时，Wm、Wm' 也差别较小。实际上，对于 T_1，单纯从查询 Q 的角度来看，结合实体化的流水线方法所消耗的时间，与单一的实体化方法所需时间相当接近，即 T、T_1 差别较小。

3.2 CPU 开销的比较

前面已经分析了在通常情况下，几种查询方法占用 CPU 的时间差别较小。D/S 方法是由三种查询执行方法综合而成，从整体上来看，与指针法相比，本文方法需要额外占用的 CPU 时间主要用于分析动态统计数据，以决定使用具体的查询执行方法。另一方面，无论使用哪种查询方法，在生成物理计划时同样需要分析统计数据、计算代价以选择一个较优的查询路径，最主要的区别在于 D/S 方法把分析、计算的步骤提前了。因此，可以认为使用新方法的 CPU 开销与指针法差别很小。

4 结语

写普通的模拟程序很难全面地反映指针法与新方法的内存、CPU 开销比较，应该在一个真正的实时数据库环境中进行比较。作者参与开发的实时数据库系统 arts-edb 已进入集成阶段，集成后，可在 arts-edb 中进行具体的试验以获得相关的数据并进行比较，这是下一步的工作。

实时数据库通常的查询执行方法过于单一，在处理包含多个数据量较大的关系的查询时往往会占用大量的内存。D/S 方法将指针法与传统的实体化、流水线查询方法相结合，并根据动态统计数据进行灵活运用，从而可以获得更好的系统性能，同时查询优化的空间也得到进一步扩展。

参考文献：

[1] LEHMAN TJ, CAREY MJ. Query Processing in Main Memory Database Management Systems [A]. Proceedings of the 1986 ACM SIGMOD international conference on Management of data[C], 1986. 239-250.

[2] SILBERSCHATZ A. 数据库系统概念[M]. 北京：机械工业出版社, 1999.

[3] WHANG K-Y, KRISHAMURTHY R. Query Optimization in a Memory-Resident Domain Relational Calculus Database System[J]. ACM Transactions on Database Systems, 1990, 15(1): 67-95.

[4] 刘云生. 现代数据库技术[M]. 北京：国防工业出版社, 2001.