知识库系统中的一种并发控制方法*

曲云尧 施伯乐

(山东矿业学院计算机系,泰安 271019) (复旦大学计算机系,上海 210003)

摘要 本文给出了一个基于 DATALOG 环境的知识库并发控制算法,算法的基础是两段锁(TWO PHASE LOCKING),它使用了相关和覆盖的概念,减少了被锁对象的数目,从而提高了系统的效率.最后给出了算法的正确性证明.

关键词 知识库,并发控制,2PL.

知识库是近几年的热门领域,人们普遍认为在 90 年代,知识处理将是计算机科学中的一个活跃分支,知识库管理系统是知识处理的重要基础[1].目前,关于知识表示,获取和推理有了大量的研究,知识库设计与查询优化都已取得了重要成果.但是在多用户知识库环境中,有些方面的研究还不多,特别是多用户知识库环境中的并发控制方法研究的更少.在知识库系统中,知识的查询与更新是经常性的工作,例如,在专家系统的开发过程中,一个普遍的问题是专家们不能在建立专家系统之前完全地把知识描述出来,甚至在专家系统运行很长时间以后也是这样.有一个例子[3]可以很好地说明这一点,DEC 公司的 XCON 专家系统是用于设计新的计算机的,但公司面临的一个重要问题是新设备和规格的不断涌现,使公司不得不派专人不断地更新知识库,以保持知识的可用性.因此,为了不断地给系统充实新知识,需要经常对知识库进行修改与查询的工作.这样就带来了两个问题:当多个用户同时对知识库更新时,如何保持知识库的一致性?传统的数据库并发控制技术,在知识库中是否能直接应用?

1 问 题

设一个知识库中有内容如图 1.

* 本文 1991 年 3 月 11 日收到,1991 年 11 月 11 日定稿

作者**曲云尧**,33 岁,讲师,主要研究领域为数据库,知识库.**施伯乐**,56 岁,教授,主要研究领域为计算机软件,数据库理论,知识库等.

本文通讯联系人,曲云尧,上海 210003,复旦大学计算机系软件组

EDB(EXTENSIONAL DATABASE)

child

childname	fathername
sue	larry
carol	larry
mary	joe
tony	joe

person

name	sex	age
larry	m	40
carol	m	12
john	m	22

IDB(INTENSIONAL DATABASE)

r_1:grandchild(X,Y):−child(Z,Y),child(X,Z)

r_2:father(X,Y):−child(Y,X)

图 1

即数据库中有父子关系 child 和描述人的关系 person. child(sue,larry)表示 sue 是 larry 的孩子. r_1 表示如果 Z 是 Y 的孩子,且 X 是 Z 的孩子,则 X 是 Y 的孙子.

非过程化语言的一个显著特点是:只需指出"干什么",而不需指出"怎样干". 假设要查询 larry 的所有子女,则分别用关系查询语言 SQL 和逻辑查询语言 PROLOG 写的程序为:

```
SELECT childname
FROM child
WHERE fathername="larry"
```

程序 1

```
? father (larry,X)
```

程序 2

这样的查询给并发控制机制带来了一个很大的问题是:在编译时,无法确定一事务的所有 READ(X)和 WRITE(X)操作,X 为数据项名. 因此在传统的关系数据库或知识库中,系统对这种查询事务的加锁方式是:

方式一:锁住整个要使用的关系.

方式二:锁住数据库中当前满足操作条件的元组.

但是第一种方式降低了并发度,第二种方式是错误的. 下面用 PROLOG 例子来说明第二种方式的错误.

例 1:有以下两个事务.

T_1:grandchild(X,larry),commit

T_2:assert(child(john,sue)),assert(child(alice,carol)),commit

如果 T_1 在 T_2 开始运行之前运行结束,则结果为空,即 larry 没有孙子. 如果 T_2 在 T_1 运行之前结束,则结果为{john,alice},即 larry 有孙子 john,alice.

由 PROLOG 系统的推理过程知,T_1 是由许多子查询组成,图 2 是 T_1,T_2 的一个调度.

time	T_1	T_2
t_1	child(Z,larry)	
t_2	child(sue,larry)	
t_3	child(X,sue)	
t_4		assert(child(john,sue))
t_5		assert(child(alice,carol))
t_6		commit
t_7	child(carol,larry)	
t_8	child(X,carol)	
t_9	child(alice,carol)	
t_{10}	commit	

图 2

在 t_3 时，T_1 封锁 fathername 为 sue 的所有元组，而在 child 中没有满足条件的元组.

在 t_4 时，T_2 可插入一个元组(john,sue)，从而 T_1 的查询结果为{alice}，而没有 john.

例 2：r_3：father(X,Y):—child(Y,X),person(X,m,AGE),AGE＞30

T_1，T_2 如下：

T_1：father(larry,X),commit

T_2：retract(r_2),assert(r_3),commit

如果 T_1，T_2 串行执行，则结果为{sue,carol}.

图 3 是 T_1，T_2 的一个调度

time	T_1	T_2
t_1		retract(r_2)
t_2	father(larry,x)	
t_3	commit	
t_4		assert(r_3)
t_5		commit

结果为{}，而没有 sue,carol.

2　冲突类型

在 PROLOG 系统中，事务对 IDB，EDB 的操作有三种：查询、插入或删除事实、插入或删除规则. 我们分别用 Q,F,R 表示它们. 根据这些操作，可把冲突类型分为以下四种：1. 查询—事实(Q—F)冲突，即一个事务访问的事实正是另一个事务要插入或删除的事实；2. 查询—规则(Q—R)冲突，即一个事务要使用的规则正是另一个事务要插入或删除的规则；3. 事实—事实(F—F)冲突，即一个事务要插入或删除的事实正是另一个事务要插入或删除的事实；4. 规则—规则(R—R)冲突，即一个事务要插入或删除的规则正是另一个事务要插入或删除的规则.

在例 1 中为 Q—F 冲突，在例 2 中，为 Q—R 冲突.

在传统的基于锁(LOCKING)的数据库并发控制机制 2PL 中，事务要对某一数据项操作(读或写)时，必须先申请其锁，只有得到锁后才能操作，并且[4]已证明，只要事务遵守

2PL 协议,其运行结果是可串行化的.我们能否直接把这种方法搬到 PROLOG 系统中呢?即事务每对一个事实或规则操作时,都先申请其锁,在得到锁后才操作.回答是否定的.我们可以从例 1 中得到证实:假设 T_1,T_2 都遵守 2PL 协议,图 2 是一个合法的调度,但这个调度是非串行化的.

为什么会出现这种情况呢?这是因为在传统的数据库并发控制机制的研究中,我们总是假设事务的每一个操作(读或写)仅仅是对当前数据库中满足操作条件的元组操作.可是满足操作条件的元组集合在事务的运行过程中可能是变化的.因此,我们必须给事务建立一更严格的锁,这种锁可以锁住当前或将来(在事务运行结束之前)数据库中满足操作条件的元组或规则.

从上面的分析可得出,在 PROLOG 系统中,传统的数据库并发控制技术必须改进,其中主要问题是选择合适的加锁对象.

3　相关和覆盖

在介绍概念之前先给出一个假设:谓词的变元或者是一个变量或者是一个常量.在大多数知识库环境中这种假设是完全满足要求的.为了减少被锁对象,[2,3]给出了相关和覆盖的概念.直观上说,谓词 P_1 和谓词 P_2 相关是指 P_1,P_2 可合一.

例 child(X,larry)和 child(X,Y)是相关的,和 child(sue,Y)也是相关的,和事实 child(carol,larry)也是相关的.谓词 P_1 覆盖谓词 P_2 是指 P_2 查询的事实集是 P_1 查询的事实集的子集.例 child(X,Y)覆盖 child(X,larry),当然也覆盖事实 child(sue,larry).

定义 1. 谓词 P_1 和谓词 P_2 相关,如果:

(1)P_1 和 P_2 是同一谓词名,且

(2)P_1 和 P_2 的第 i($i\in\{1\cdots n\}$,n 为 P_1 或 P_2 的目数)个变元不能为两个不同常量,且

(3)如果 $P_1(P_2)$两个变元为同一变量,则 $P_2(P_1)$相应位置的两个变元不为不同常量.

定义 2. 谓词 P_1 覆盖谓词 P_2,如果:

(1)P_1 和 P_2 是同一谓词名,且

(2)当 P_2 的第 i 个变元为一常量时,P_1 的第 i 个变元或为此常量或为一变量,当 P_2 的第 i 个变元为一变量时,P_1 的第 i 个变元也为一变量,且

(3)如果 P_1 两个变元为同一变量,则 P_2 相应位置的两个变元为同一常量或变量.

容易证明:(1)P_1 和 P_2 相关当且仅当 P_2 和 P_1 相关

(2)P_1 覆盖 P_2 且 P_2 覆盖 P_3,则 P_1 覆盖 P_3

(3)P_1 覆盖 P_2 或 P_2 覆盖 P_1,则 P_1 和 P_2 相关

我们约定谓词 P 和规则 r 相关(覆盖)是指 P 和 r 的头谓词相关(覆盖).

相关算法:

输入:谓词 P_1,P_2

输出:如果 P_1 和 P_2 相关,则返回"YES",否则返回"NO"

方法:IF P_1 和 P_2 名字不同

　　THEN 返回 "NO"

```
ELSE    I=1
    WHILE I<=n DO /*n为P1的目数*/
    BEGIN
      IF P1,P2的第I个变元分别为一常量c1,c2 AND c1≠c2
      THEN 返回"NO"
      ENDIF
      IF P1(P2)的第I个变元和第J(J∈{1…I-1})个变元相同 AND P2(P1)的第I个变元和第J个变元
        为不同常量
      THEN 返回"NO"
      ENDIF
      I=I+1
    END
ENDIF
返回"YES"
```

覆盖算法:

输入:谓词 P_1,P_2

输出:如果 P_1 覆盖 P_2,则返回"YES",否则返回"NO"

方法:

```
IF P1和P2名字不同
THEN 返回"NO"
ELSE    I=1
    WHILE I<=n DO /*n为P1的目数*/
    BEGIN
      IF P1的第I个变元为一常量c1
      THEN IF P2第I个变元为一常量c2 AND c1≠c2 OR P2第I个变元为一变量
          THEN 返回"NO"
          ENDIF
      ENDIF
      IF P1的第I个变元和第J(J∈{1…I-1})个变元为同一变量 AND P2第I个变元和第J个变元为不
        同变量或常量.
      THEN 返回"NO"
      ENDIF
      I=I+1
    END
ENDIF
返回"YES"
```

4 QUERY-FACT-RULE 锁方法

在QUERY-FACT-RULE锁方法中,每一个事务 T_i 有一个局部查询集 Q_i,一个局部事实集 F_i,和一个局部规则集 R_i. 另外,QUERY-FACT-RULE还保持一个全局查询集 Q_L,一个全局事实集 F_L,和一个全局规则集 R_L. 当事务要查询时,它把查询操作 q_i 先放在 Q_i 中,然后由QUERY-FACT-RULE检查 q_i 是否和其它事务发生冲突,如果无冲突,则再把它放在 Q_L 中并执行,否则挂起等待.

1. 局部查询集 Q_i:事务 T_i 的查询集合,其中每一个查询可为以下几种情况之一:(a)已执行过,(b)正在执行,(c)正在被检查或挂起等待.

2. 局部事实集 F_i:事务 T_i 要插入或删除的事实集合,其中每一个事实可为以下几种情况之一:(a)已插入或删除过,(b)正在插入或删除,(c)正在被检查或挂起等待.

3. 局部规则集 R_i:事务 T_i 要插入或删除的规则集合,其中每一个规则可为以下几种情况之一:(a)已插入或删除过,(b)正在插入或删除,(c)正在被检查或挂起等待.

4. 全局查询集 Q_L:是(q_j,T_i)的集合,(q_j,T_i)表示 T_i 的查询 q_j 已被允许,并且还没有释放其所持有的锁.

5. 全局事实集 F_L:是(f_j,T_i)的集合,(f_j,T_i)表示 T_i 所插入或删除的事实 f_j 已被允许,并且还没有释放其所持有的锁.

6. 全局规则集 R_L:是(r_j,T_i)的集合,(r_j,T_i)表示 T_i 所插入或删除的规则 r_j 已被允许,并且还没有释放其所持有的锁.

QUERY－FACT－RULE 算法:

输入:(o_j,T_i),o_j 是事务 T_i 的第 j 个操作.

方法:/ * LOCK PHASE1 * /

CASE1 o_j 为查询操作

IF 存在$(q_m,T_i)\in Q_L$ AND q_m覆盖 o_j

THEN 返回并执行

ENDIF

IF 存在$(f_k,T_m)\in F_L$ AND o_j 和 f_k 相关 OR 存在$(r_k,T_m)\in R_L$ AND

o_j和 r_k相关$(m\neq i)$ / * Q－F 或 Q－R 冲突 * /

THEN 挂起 T_i直到 f_k 或 r_k 被释放

ENDIF

$Q_L=Q_L+(o_j,T_i)$

返回并执行

CASE2 o_j 为插入或删除一个事实 f_j

IF 存在$(f_k,T_m)\in F_L$ AND f_j 和 f_k 相关 OR 存在$(q_k,T_m)\in Q_L$ AND

f_j和 q_k 相关$(m\neq i)$ / * F－F 或 Q－F 冲突 * /

THEN 挂起 T_i直到 f_k 或 q_k 被释放

ENDIF

$F_L=F_L+(f_j,T_i)$

返回并执行

CASE3 o_j 为插入或删除一个规则 r_j

IF 存在$(q_k,T_m)\in Q_L$ AND r_j 和 q_k 相关 OR 存在$(r_k,T_m)\in R_L$ AND

r_j和 r_k 相关$(m\neq i)$ / * R－R 或 Q－R 冲突 * /

THEN 挂起 T_i直到 q_k 或 r_k 被释放

ENDIF

$R_L=R_L+(r_j,T_i)$

返回并执行

/ * LOCK PHASE2 * /

CASE4 o_j 为确认语句 commit

$R_L=R_L-R_i$/ * 释放规则锁 * /

$F_L=F_L-F_i$/*释放事实锁*/

$Q_L=Q_L-Q_i$/*释放查询锁*/

在CASE1中判断q_m是否覆盖o_i对系统的效率是重要的,因为在逻辑程序设计中,一个程序的查询语句往往会引起一系列查询操作(递归调用等),这些操作都被某一个操作覆盖,这样可省去许多不必要的加锁放锁操作.

5 算法分析与正确性证明

QUERY−FACT−RULE的CASE1,CASE2,CASE3有相同的复杂性.所以,只要分析一种情形就可以了.在CASE1中,执行第一个IF条件语句最坏情况下的比较次数为$k_1.n.(n-1)/2$,k_1为Q_L的元素个数,n为o_i的目数,执行第二个IF条件语句最坏情况下的比较次数为$k_2.n.(n-1)/2+k_3.n.(n-1)/2$,$k_2$和$k_3$分别为$F_L$和$R_L$的元素个数.所以,总的比较次数为$(k_1+k_2+k_3).n.(n-1)/2$.

在算法中我们使用二种锁模型:读锁和写锁,当查询时使用读锁,读锁是可以共享的,即两个事务可同时访问一个事实或一条规则.当插入或删除时使用写锁,写锁是互斥的.

为了说明QUERY−FACT−RULE算法是正确的,我们只要证明QUERY−FACT−RULE是二段锁方法即可,因此需完成以下命题:

1.当事务查询时,对事实或规则申请读锁.

2.当事务插入或删除时,对事实或规则申请写锁.

3.事务一旦释放锁,不再申请锁.

引理. 设O_1,O_2分别为T_1,T_2的两个操作(读或写),其操作的事实或规则头谓词分别为P_1,P_2.如果O_1,O_2发生冲突,则P_1,P_2相关.

下面证明QUERY−FACT−RULE是两段锁方法.证明分三步进行.

1.当事务查询时,对访问的事实或规则申请读锁

设o_i为查询操作,由引理知,只要对和o_i相关的事实或规则加锁即可.再由算法CASE1知,判断o_i是否和其它事务冲突时不检查Q_L,所以申请读锁.

2.当事务插入或删除时,对事实或规则申请写锁

(a) o_i为插入或删除一个事实.由引理知,只要对和o_i相关的查询或事实加锁即可.再由算法CASE2知,判断o_i是否和其它事务冲突时,检查Q_L和F_L,所以申请写锁.

(b) o_i为插入或删除一条规则.由引理知,算法只要对所有相关的查询和规则加锁就足够了,算法正是这样.

3.事务一旦释放锁,不再申请锁

由算法的LOCK PHASE2知,只有当事务T_i确认后,才分别从Q_L,F_L,R_L中去掉谓词或规则,所以T_i一旦释放锁,不再申请锁.

结 论: 我们已给出了一个知识库并发控制算法,算法使用了相关方法减少了被锁对象的数目用覆盖方法提高了系统的效率.由于基于PROLOG环境的知识库系统引起了许多人的重视[5],所以我们相信QUERY−FACT−RULE是很有吸引力的.另外,适合知识库系统的其它并发控制方法以及各种方法的优缺点分析都有待于进一步的研究.

参考文献

1 Brodie N L, Mylopulos J. On knowledgebase management system: integrating artificial intelligence and database technologies. Springer—Verlag, 1985.

2 Carey M J, Dewitt D J, Graefe G. Mechanisms for concurrency control and recovery in prolog—a proposal. Proceeding of first international workshop on expert database system, 1986: 271—291.

3 Chen Y J, Yang W P. A mechanism for concurrency control in a coupled knowledge base management system. Journal of Information Processing.

4 Eswaran K P *et al*. The notions of consistency and predicate lock in data base system. COMM. ACM19, 1976(**11**).

5 Itoh H. Research and development on knowledgebase system at ICOT. Proceeding of the twelfth international conference on very large database, 1986: 437—455.

A MECHANISM FOR CONCURRENCY CONTROL IN KNOWLEDGEBASE MANAGEMENT SYSTEM

Qu Yunyao

(*Department of Computer Science, Shandong Institute of Mining and Technology, Tai'an* 271019)

Shi Baile

(*Department of Computer Science, Fudan University, Shanghai* 210003)

Abstract　This paper presents a concurrency control mechanism in knowledgebase system. The mechanism is based on database concurrency control 2PL (TWO PHASE LOCKING), it uses relate—cover' concept to reduce the number of objects to be locked. This method has been proved to guarantee serializability and correctness.

Key words　Knowledgebased, concurrency control, 2PL.