基于语义传递的分层事务调度技术的研究*

曲云尧 施伯乐

(复旦大学计算计系 上海 200433)

摘要 传统的读写事务模型是面向机器的,即事务的每个操作都是对数据库的存取操作,并且事务只有两层:逻辑层和物理层;逻辑层描述对逻辑数据的操作,如对记录的修改、查询等.物理层是描述对磁盘页面的操作.因此,它不能有效地描述应用程序的语义.本文提出了一个3层事务模型,在传统事务模型的基础上增加了一个语义层,该层能有效地描述应用环境的各种语义,为事务处理提供语义信息.同时设计了基于这种事务模型的并发控制机制,该机制能利用事务语义层的语义,对低层调度进行指导,即把高层操作的语义传递到低层,从而减少在低层调度的冲突机会.本文的调度协议ML-protocol可进行3种语义的传递:可并行、可交换、冲突.ML-protocol产生的调度是可串行化的.

关键词 数据库,事务处理,分层事务,并发控制.

中图分类号 TP311

随着数据库应用的不断深入,其应用领域也在不断扩大.已从传统的商业应用领域扩展到CAD/CAM,CIMS,CASE等工程领域.这些新的应用对数据库管理系统提出了新的要求,而传统的数据库技术与理论难以满足这些要求.事务管理是数据库管理系统的引擎部件,其效率的高低直接影响整个系统的性能.在传统的事务处理过程中[1],事务被看作是一系列读写操作的执行序列,事务管理机制的任务是调度这些操作序列,使它们以一致的(consistent)方式运行.但是,这种读写操作的语义单一,难以充分描述应用程序中的语义,因此不能为调度机制提供灵活高效的调度方式.我们把这种操作及其调度机制称为面向机器的操作和调度机制,因为这些操作只反映了对存储介质的存取(读写)语义.但人们逐渐发现,在调度过程中,可利用应用程序中的语义来改善事务处理的效率[2~4],即事务处理机制处理的对象不应仅仅是语义单一的读写操作,而可以是带有丰富语义的高级操作(high-level operation).我们把能处理这种高级操作的事务处理机制称为面向应用的事务处理机制.这样,系统可充分利用应用环境的各种语义来改善系统的效率.

从提取事务语义方面的研究结果来看,这些工作大致可分为以下几类.

(1)从事务的高级操作中获取语义.把事务看作是由一系列高级操作(如delete /insert, credit/debit等),而不再是仅由Read(x)和Write(x) 2种读写操作构成.高级操作之间的可

* 本文研究得到国家863高科技项目基金资助.作者曲云尧,1961年生,博士,副教授,主要研究领域为数据库理论与应用.施伯乐,1935年生,教授,博士导师,主要研究领域为数据库理论与应用,知识库,软件工程.

本文通讯联系人:施伯乐,上海200433,复旦大学计算机系

本文1996-07-20收到修改稿

交换性是判定操作之间冲突性的条件,从而可更合理地描述操作之间的可并发性.这方面的工作见文献[4,5].

(2)从事务操作的复杂对象中获取语义.根据复杂对象的构成特点来提高事务操作的执行效率.例如在队列操作中,如2个操作分别对队列的头和尾操作,则它们可能是不冲突的.另外根据复杂对象之间的引用方式,也可在一定程度上提高操作的效率.这方面的工作见文献[3,6].

(3)从事务本身获取语义.根据事务的特点,把事务分解成一系列可并行的子事务.子事务之间可以根据指定方式并行运行,从而缩短整个事务的运行时间.这方面的工作包括见文献[2,7].

(4)从数据库的完整性约束中获取语义.根据完整性约束,把数据库分成为一些子数据库,并把完整性约束分配到每个子数据库上.再把事务分配在多个子数据库上运行,以缩短整个事务的运行时间.有关这方面的工作见文献[8,9].

上述工作的共同特点是事务的操作不再是简单的面向机器的读写操作,而是面向应用的带有丰富语义的高级操作.所以,事务处理机制应能对这些高级操作进行有效调度.

尽管事务处理机制可对高级操作进行有效调度,但最终这些操作还是要转换成面向机器的一系列读写操作,即高级操作需由低级操作去完成.因此,系统仍需对这些读写操作进行管理与调度,这样就产生了事务(操作)的层次问题,即事务的调度应是分层的.[10,11]在分层事务管理中,事务的上层操作由下层的一系列子操作完成.例如,设一个高级操作O,由一组下层操作$o_1 \ldots o_n$完成.我们可把O看成一个子事务,显然O的执行成功当且仅当$o_1 \ldots o_n$全部执行成功,否则O就不是原子的(这一点和嵌套事务不同.在嵌套事务中,主事务允许某些子事务失败(夭折),但主事务仍可成功(提交)).

关于分层事务的调度技术,已有许多研究,并且取得了一些成果.但是这些结果基本上是基于事务本身的组织结构.例如,层次间的可串行化、一致性等,并没有考虑如何有效利用高层操作的语义进行调度.本文将对如何在层次之间进行语义传递(即用高层操作的语义指导低层调度)这方面的问题做一探讨.

1 相关的工作

关于多层事务管理的研究,已出现很多.[10~18]文献[10]给出多层事务调度的准则,讨论了多层事务及并发控制和恢复技术,并涉及了性能方面的考虑.文献[12]给出关于嵌套事务的形式化正确性模型,并证明,为保证顶层事务的可串行化,其充分条件是各层的执行都是可串行化的且每个操作在各层的执行顺序是一致的.其整个过程是采用自底向上的归约方式,没有考虑上层操作语义对下层的影响.

文献[11]给出了一个简洁的n层串行化定理,基于这个定理设计了n层调度机制.该文较全面地介绍了分层事务的恢复技术,并给出了性能模拟.在此,也提出了用高层操作的语义来获得更高的并发度,但并没有提到如何把高层语义用于调度机制中.

捕捉高层操作的语义已是基于语义并发控制的核心,这方面的工作主要围绕着操作的可交换性进行.文献[4]提出了可恢复性(recoverable),这种语义比可交换性弱.文献[13]首次把语义信息引入分层调度.尽管基于语义的调度研究已较深入,但如何把这些语义用于

分层事务调度中还不多见.[14]

2 可交换性与可并行性

并发控制机制的效率在很大程度上取决于能否最大限度地利用操作之间的可并行性和可交换性.如果2个操作能同时执行则称它们是可并行的.例如,2个读操作是可并行的.如果2个操作的执行效果与它们之间的串行执行顺序无关,则称它们是可交换的.例如,对1个集合的2个插入操作是可交换的;对某一个数据项进行的加或减的2个操作也是可交换的.

数据库是一个对象集合.在任一时刻,每一对象都有一个状态,所有对象的状态集合构成了数据库状态.设 S 为一个数据库状态,$State(S,O)$ 表示在 S 上执行操作 O 后的数据库状态.$Return(S,O)$ 表示在状态 S 上执行 O 操作后所得到的返回值(设任一操作都有一返回值,如,读操作的返回值为读到的结果,而写操作的返回值可为执行的成功与否等).例如,设数据库状态 $S=\{x=1,y=2\}$,则 $Return(S,Read(x))$ 的值为1,$State(S,x=x+1)=\{x=2,y=2\}$.其中 O 也可为一个操作执行序列,此时数据库的状态是执行完 O 序列后产生的,而返回值是 O 中所有操作的返回值集合:$\{(O_i,Value)\}$,O_i 为 O 操作序列中的一个操作,$Value$ 为 O_i 的返回值.

设 O_i 由一组子操作 $oi_1\ oi_2\cdots\ oi_n$ 去完成.$O_1//O_2$ 表示 O_1 和 O_2 并行执行,即它们的子操作之间可以任意的方式执行.O_1O_2 表示 O_1 和 O_2 串行执行,即 O_1 的所有子操作执行完后再执行 O_2.

定义. 如果 $State(S,O_1//O_2)=State(S,O_1O_2)$,且 $State(S,O_1//O_2)=State(S,O_2O_1)$,且 $Return(S,O_1//O_2)=Return(S,O_1O_2)$,且 $Return(S,O_1//O_2)=Return(S,O_2O_1)$,则称 O_1 和 O_2 是可并行的.

定义. 如果 $State(S,O_1O_2)=State(S,O_2O_1)$,且 $Return(S,O_1O_2)=Return(S,O_2O_1)$,则称 O_1 和 O_2 是可交换的.

显然,如果2个操作是可并行的,则一定是可交换的,反之不然.如果2个操作 O_1 和 O_2 需满足一定条件才是可交换的,则称它们为条件可交换的.例如,设 $S=\{x=1,\cdots\}$,在数据库上有完整性约束:$x>0$,2个操作 O_1:$x=x+c$(把 x 加 c),O_2:$x=x-d$(把 x 减 d).O_1 和 O_2 可交换的条件为 $1-d>0$,否则 O_1 和 O_2 是不可交换的.条件可交换性是利用语义信息进行调度的有效方法.

如果2个操作是不可交换的,则是冲突的.冲突操作之间的执行顺序决定了调度的执行结果.

3 分层调度

在传统的数据库管理系统中,事务管理机制所控制的数据单位往往仅为页面(页面是数据库和操作系统的数据交换单位).但页面控制往往可使同一页面中的其它数据在整个程序运行过程中不能被其它程序使用,从而降低了系统的效率.因此为了改善系统的效率并保证事务的原子性,就引进了记录锁和页面锁.记录锁只封锁事务要使用的数据记录,页面锁封

锁这些记录所在的页面. 数据一旦使用完毕,页面锁便可立即释放,但记录锁需在事务结束后释放. 这就出现了事务层次的概念,即记录层和页面层. 系统需对这2层进行控制. 例1给出了一个2层事务的调度.

例1:A,B 为2个数据项,有2个事务

T_1:$Debit_1(A)$ $Credit_1(B)$

T_2:$Debit_2(B)$ $Credit_2(A)$

设每个 $Debit(X)$和$Credit(X)$都由底层操作 $R(X)W(X)$实现,h 为它们的一个调度.

h:$Debit_1(A)Debit_2(B)$ $Credit_1(B)$ $Credit_2(A)$

h' 为 h 相应的底层调度

h':$R_1(A)W_1(A)R_2(B)W_2(B)R_1(B)W_1(B)R_2(A)W_2(A)$

如果从低层开始调度,即只考虑页面上的冲突,显然 $R_2(B)$和$W_1(B)$,$W_2(B)$和$R_1(B)$等都是冲突的,因此 h'调度是不能成功的. 而利用 $Debit(X)$和$Credit(X)$的可交换性知,h调度和串行调度 T_1T_2 等价,因此 h 是可串行化的,即 h'也是可串行化的.

因此为了充分地利用应用程序中高级操作的语义信息进行调度,根据事务操作的语义,把事务分层是自然的.

事务分层是利用高级操作的语义来提高并发度的有效方法. 这样,事务的每个层次可充分表达其操作语义,由于每层都有不同的语义,因此可根据层次选择不同的并发控制技术.

一般来讲,在一个分层调度系统中,层次数的选择是任意的,但是层次过多,系统可能很复杂,效率也可能随之降低,而层次太少又不能有效捕捉操作的语义. 因此,如何合理地分层,仍是一个值得探讨的问题. 在图1中,我们用分层调度图表示例1中的调度 h.

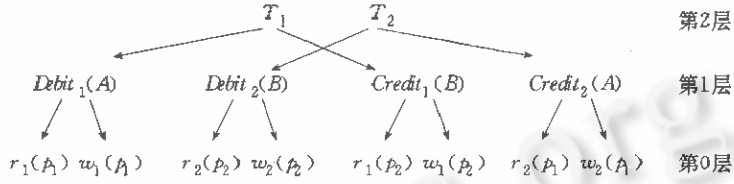

图1 T_1和T_2的分层调度

我们把对页面的操作层称为第0层,再上一层称为第1层,依次类推. 所以,图1是一个3层调度、在调度中,我们设 A,B 记录分别存放在页面 p_1 和 p_2 中,容易看到 T_1 和 T_2 的相对于页面操作的串行图(Serialization Graph)是有环的. 但是此调度是正确的.

在图1中,每个 $Credit$ 和 $Debit$ 操作由低层操作 $r(p)w(p)$去执行,表示为对页面读和更新.

定义. 如果操作 a 和 b 是可并行或可交换的,则称 a 和 b 是相容的.

定义. 设操作 a 为第 i 层的的操作,其相应第 $i-1$ 层的操作由 $a_1,a_2\ldots a_n$ 组成,且任一操作的第0层操作 $r(p)$和 $w(p)$的执行都是原子的(不可再分的). S_i 为第 i 层操作的调度. 如果第 $i-1$ 层调度满足下列条件之一,则称 a 的执行为原子的.

(1) $S_{i-1}=\ldots a_1a_2\ldots a_n\ldots$,即在 a_i 和 $a_{i+1}(1\leqslant i\leqslant n-1)$之间无其它操作执行. 其中每个 a_i 的执行是原子的.

(2) $S_{i-1}=\ldots a_1a_2\ldots a_i\ b\ a_{i+1},\ldots a_n\ldots$,在 a_i 和 a_{i+1}之间有其它操作 b 执行,但 b 和

$a_1,\ldots a_i$ 或 $a_{i+1},\ldots a_n$ 是相容的.其中每个 a_i 的执行是原子的.

在图 1 中,$Credit(x)$和 $Debit(x)$的执行都是原子的.又因为 $Credit(x)$和 $Debit(x)$都是可交换的,所以第 2 层操作(事务层)的执行也是原子的.

定理 1. 设操作 O 为分层调度中第 i 层的一个操作,如果 O 的执行不是原子的,则此分层调度是不可串行化的.

证明:设 O 为事务 T_1 的操作,且其第 $i-1$ 层操作由 $o_1,\ldots o_n$ 组成,由于 O 的执行不是原子的,则在 $o_1,\ldots o_n$ 之间有其它事务的操作 b 执行.设其执行过程为:$o_1\ldots o_i\ b\ o_{i+1}\ldots o_n$.由于 b 和 $o_1,\ldots o_i$ 中的某些操作不相容,且和 $o_{i+1},\ldots o_n$ 中某些的操作也不相容,因此在 T_1 和 T_2 的调度的串行图中,有环产生,所以调度是不可串行化的.

定理 2. 设操作 a,b 为第 i 层操作,其 $i-1$ 层相应的操作分别为 $a_1,\ldots a_n$ 和 $b_1,\ldots b_m$,如果对任意 i,j,a_i 和 b_j 都是相容的,则 a 和 b 是相容的.反之不然.

证明:如果对任何 a_i 和 b_j,a_i 和 b_j 是可并行的,则由定义知,a 和 b 也是可并行的.如对任何 a_i 和 b_j,a_i 和 b_j 是可交换的,我们证明 a 和 b 也可是交换的.对任一状态 S,

$$Return(S,ab)=Return(S,a_1\ldots a_nb_1\ldots b_m),$$

由于 b_1 和 $a_1,\ldots a_n$ 都可交换,所以

$$Return(S,a_1\ldots a_nb_1\ldots b_m)=Return(S,b_1a_1\ldots a_nb_2\ldots b_m),$$

依次类推可得,

$$\begin{aligned}&Return(S,a_1\ldots a_nb_1\ldots b_m)\\&=Return(S,b_1b_2\ldots b_ma_1\ldots a_n)\\&=Return(S,ba)\end{aligned}$$

同样可得 $State(S,ab)=State(S,ba)$,所以 a 和 b 也是可交换的.

在例 1 中,我们看到,操作 $Credit(A)$和 $Debit(A)$是相容的.但是其相应的低层操作之间是不相容的.证毕.

定理 2 说明在一般的分层调度过程中,下层的相容性语义会直接反应到上层,但上层的语义并不一定能被下层准确反应.因此如何把上层的语义用来指导低层调度是一个很有意义的问题.

4　分层调度过程中的语义传递

本节以一个 3 层事务模型来讨论分层调度中的语义传递问题.我们把事务分成 3 层:语义层、逻辑层和物理层.即在传统两层事务模型上加一个语义层.

语义层:由各种可充分描述事务语义的高级操作构成,如删除、插入、修改一个对象等.这一层的目的是为并发控制机制提供调度语义.

逻辑层:由对记录的读写操作构成,其操作的数据项为记录.语义层上的一个操作可能需一组逻辑层上的操作去完成.

物理层:由对页面的读写操作构成.

图 2 描述了事务的描述环境和事务操作的数据项的层次,以及事务的层次之间的对应.在例 1 中,事务 T_1 的 3 个层次如图 3.设 A 放在页面 p_1 中,B 放在页面 p_2 中.

事务的描述环境	事务操作的数据项的层次	事务的层次
现实世界	对象层	语义层
机器逻辑世界	记录层	逻辑层
机器物理世界	页面层	物理层

图 2 事务的层次之间的对应与划分

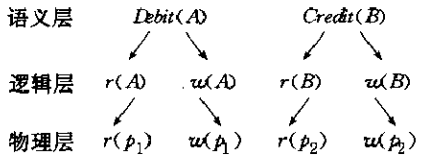

图3 事务T_1的3个层次

我们以 $2PL$ 调度机制为基础讨论事务层次之间的语义传递问题.

设语义层上的 2 个操作分别为 a 和 b,它们逻辑层上相应的操作分别为 $a_1 \ldots a_n$ 和 $b_1 \ldots b_m$. 为完成 a,操作序列 $a_1 \ldots b_n$ 是按此顺序串行执行的.

我们给出语义传递原则:

(1)如果 a 和 b 是可并行的,则 a_i 和 b_j 之间可以任一次序执行.

(2)如果 a 和 b 不可并行但可交换,则 $a_i(b_j)$ 可执行,当且仅当 $b_m(a_n)$ 已执行完.

一般来说,2 个高层操作可交换,不一定意味着其低层操作中某 2 个子操作之间也可交换. 如在例 1 中,$Debit_1(A)$ 和 $Credit_2(A)$ 是可交换的,但其记录层的操作 $r_1(A)$ 和 $w_2(A)$ 之间是不可交换的.

如果 2 个操作之间是可并行的或可交换的,则称它们互不阻塞的. 尽管 2 个操作互不阻塞,但它们之间的并行度是有差别的. 图 4 描述了可并行,可交换和冲突操作之间的执行情况.

O_i 和 O_j	执行情况
可并行	O_i 和 O_j 的低层操作可以任意方式执行
可交换	$O_i(O_j)$需等 $O_j(O_i)$执行完后,再执行,但不需等整个事务执行完
冲突	$O_i(O_j)$需等 $O_j(O_i)$所在事务执行完(提交),再执行

图 4 可并行、可交换和冲突操作之间的执行情况

根据上述讨论,我们给出分层调度协议 *ML-protocol*.

ML-protcol:

(1)加锁协议

设 O_1 已持有要操作的对象的锁,O_2 申请对此对象加锁.

语义层:O_1 和 O_2 对同一对象操作

1)如果 O_1 和 O_2 可并行,则 O_2 可得锁,执行

2)如果 O_1 和 O_2 可交换且 O_1 已执行完,则 O_2 可得锁,执行,否则,如 O_1 未执行完,则让 O_2 等待

3)如果 O_1 和 O_2 冲突,则 O_2 等待,直到 O_1 放锁.

逻辑层:O_1 和 O_2 对同一记录操作

1)如果 O_1 和 O_2 的语义层操作可并行,则 O_2 可得锁,执行

2)如果 O_1 和 O_2 的语义层操作可交换,且 O_1 的语义层操作已执行完,则 O_2 可得锁,执行,否则,如语义层操作未执行完,则让 O_2 等待

3)如果 O_1 和 O_2 的语义层操作冲突,则 O_2 等待,直到 O_1 放锁.

物理层：O_1 和 O_2 对同一页面操作

1）如果 O_1 和 O_2 都为读操作，则 O_2 可得锁，执行

2）如果 O_1 和 O_2 有一个为写操作，则 O_2 等待，直到 O_1 放锁.

(2)放锁协议，

设 O 为语义层的一个操作.

语义层：操作 O 一旦放锁，则 O 所属事务不再申请锁.

逻辑层：O' 为 O 的一个逻辑层操作，O' 一旦放锁，O 不再申请锁

物理层：O'' 为 O' 的一个页面操作，O'' 一旦执行完，立即放锁

定理 3. 按 *ML-protocol* 调度协议调度的事务是可串行化的.

证明：分 2 步进行.

(1)每层的操作是原子的

物理层：根据假设知页面上的读写操作是原子的.

逻辑层：虽然物理层上的锁用完即放，但逻辑层的操作不会在物理层被干扰. 这是因为逻辑层上的操作放锁之前，其它的对同一记录的冲突操作不会被执行.

语义层：设 O 为本层上的操作. 可和 O 同时执行的操作有 2 类：可并行的和可交换的. 由定义知，可并行的和可交换的操作的执行等价于串行执行，因此 O 的执行是原子的.

(2)语义层上的操作是可串行化的

由于语义层的操作为原子的并且一旦事务放锁，它不再申请锁. 所以事务是一个两阶段执行过程，因此是可串行化的.

5　小　结

利用事务的高级操作和对象的丰富语义来提高并发度是近几年的一个重要研究方向. 特别是在非商业领域，如 CAD，CAM，CASE 等，传统的低级磁盘读写事务模型，显得很不适应. 因为在这些环境中，长事务、协作事务、嵌套事务等非常需要能从高层得到语义来指导低层调度. 这就引出了多层事务及其调度技术的研究. 本文讨论了如何把高层语义传给低层调度，并给出了一种调度策略. 我们相信这个工作在未来基于语义的并发控制机制中将起积极的作用.

参考文献

1　Bernstein P A, Hazilacos V, Goodman N. Concurrency control and recovery in database system. Addison-Wesley, Reading, MA, 1987.

2　Garcia Molina H. Using a semantic knowledge for transaction processing in distributed database. ACM TODS, June 1983.

3　Panos R. Chrycanthis, extracting concurrency from objects: a methodology. Proc. ACM SIGMOD, ACM, New York, 1991.

4　Badrinath B and Ramamritham K. Semantics-based concurrency control: beyond commutativity. ACM Transaction on Database System, 1992, **17**(1).

5　Weihl W. Commutativity-based concurrency control for abstract data types. IEEE transactions on Computers, Dec. 1988.

6 Cart M, Feeri J. Integrating concurrency control. In: Bancilhon F, Delobel Claude, Kanellakis Paris eds., Building an Object-Oriented Database/the Story of O2, Morgan Kaufmann Publishers, San Mateo, California, 1992.

7 Farrag A A, Ozsu M T. Using semantic knowledge of transactions to increase concurrency. ACM TODS, 1989, **14** (**4**).

8 Rastogi R *et al*. On correctness of non-serilizable executions. Proc. ACM PODS, 1993.

9 Korth H F, Speegle G. Formal model of correctness without serilizability. Proc. ACM SIGMOD, 1988.

10 Beeri C, Schek H J, Weikum G. Multi-level transaction management. Theoretical Art or Practical Need, EDBT, 1988.

11 Weikum G. Principles and realization strategies of multilevel transaction management. ACM TODS, 1991, **16**(**1**).

12 Beeri C, Bernstein P A, Goodman N. A model for concurrency control in nested transactions systems. J. ACM, 1989. 230~269.

13 Badrinath B, Ramamritham K. Performance evaluation of semantics-based multilevel concurrency control protocols. Proceeding of the ACM SIGMOD International Conference on Management of Data, ACM, New York, 1990.

14 曲云尧. 新一代数据库系统中事务处理技术的研究[博士论文]. 复旦大学, 1996.

USING SEMANTICS PASSING TECHNIQUES TO PROCESS MULTILEVEL TRANSACTIONS

QU Yunyao SHI Baile

(*Department of Computer Science Fudan University Shanghai* 200433)

Abstract Traditional read/write transaction model is machine—oriented, i. e., every operation of a transaction is described as a low level access to database, and the transaction has two layer: logical and physical layer. The logical layer is used to express operations to logical database, i. e., update or query to records; the physical layer is used to describe operations to pages of disks. Therefore, it can not capture semantics of applications effectively. The authors present a three layer transaction model, which is constructed by adding a Semantic layer on traditional transaction model. The layer can express various semantic information about application for transaction processing. They also design a concurrency control mechanism ML-protocol based on the transaction model. The mechanism can exploit the semantics from semantic layer for scheduling subtransaction at lower level, that is, the semantics can be passed from high-level to low-level to reduce the ratio of conflicts. ML-protocol can pass three kinds of semantics: parallelity, commutativity, and conflict. The schedules produced by ML-protocol are serializable.

Key words Database, transaction processing, multilevel transaction, concurrency control.

Class Number TP311