乐观嵌套工作流事务模型的形式化描述*)

董云卫[1] 郝克刚[2]

(西北工业大学计算机学院 西安 710072)[1] (西北大学计算机科学系 西安 710069)[2]

摘 要 本文对乐观嵌套工作流事务的基本概念及其规则进行形式化描述,并对乐观嵌套工作流事务的层次化、可串行性进行了讨论。最后得出结论:1)利用乐观嵌套模型来调度事务是可以串行调度的,它能够保证数据的一致性。2)在乐观嵌套工作流事务执行过程中采用多版本的数据管理机制来处理并发事务访问同一数据及其中间状态,是无联级回退的。

关键词 乐观嵌套工作流事务模型,工作流事务,形式化描述

A Formal Description of Optimistic Nested Workflow Transaction Model

DONG Yun-Wei[1] HAO Ke-Gang[2]

(School of Computer Science, Northwest Polytechnic University, Xi'an 710072)[1]

(Department of Computer Science, Northwest University, Xi'an 710069)[2]

Abstract The formal description of the definition and running rules in Optimistic Nested Workflow Transaction Model are introduceed in this paper. Some properties, such as Hierarchies and serializability, are discussed. Two conclusions below are also presented: 1) this model can ensure the data integrity, because transactions are serializable by a transaction scheduler; 2) this model adopts multi-data version to store intermediate state of data accessed by transaction, and therefore transaction recovery is cascadeless.

Keywords Optimistic nested workflow transaction model, Workflow transaction, Formal description

1 前言

目前的工作流事务模型几乎都是通过对传统的事务模型进行扩展,即对事务的 ACID 属性进行放松来实现工作流事务管理要求的。文[1]对事务的原子性与孤立性限制进行放松,把工作流事务分成 3 种类型:可补偿事务、支点事务和可再次执行事务。可补偿事务失败后通过执行与其操作互补的操作(称为补偿事务)对事务执行产生的影响进行恢复,保证系统的一致性;支点事务则不可补偿,严格遵循数据库事务的属性;可再次执行事务假设该类事务反复执行后一定能够成功提交,当事务执行失败后,它通过有限次的反复执行,最终成功提交来保证系统的正确性。文[2]则在文[1]的基础上进一步阐述了只有满足良构性的工作流事务才能对其原子性和孤立性放松,这样的事务才具有可补偿性或可再次执行直到功提交。文[3]则是通过建立工作流事务控制流脚本,事务脚本中定义了事务执行的所需的变量,这些变量只能由定义它们的 ConTract 事务来访问,当事务执行失败时,通过撤销事务脚本表示的控制操作来实现事务的回退,相当于执行了补偿事务。文[4]则是为每一类工作流事务建立一种事务代理机制,由事务代理为被访问的数据建立不同的版本,当建立数据的该事务执行失败时,与该数据所有版本相关的事务将被补偿。

这些事务模型应用在工作流事务管理中针对事务的不同属性进行放松,以提高工作流应用的事务处理能力,它们的共同特点是:事务执行失败后通过执行它的补偿事务来保证系统的正确性。这种基于补偿原理的事务模型有一个假设:所有补偿事务都能够正确执行并提交。实际上,补偿事务与一般事务一样也可能因执行失败而不能提交;并且,在这样的事务工作流系统中,需要用户为每个活动设计事务代理或补偿事务,这在实际应用中往往不可行,因为用户不是软件工程师,它不能较好地理解事务的语义和语法。

本文把乐观并发控制机制用在工作流事务管理中,并采用嵌套的机制来刻画工作流的控制流程管理机制,提出了一种乐观嵌套工作流事务模型,它克服了补偿事务模型的缺点。同时,在工作流事务管理中死锁的监测与工作流活动的控制交织在一起,是一件非常困难的事,采用乐观嵌套工作流事务模型,这些问题都变得简单了[5,6]。本文第 2 节介绍乐观嵌套工作流事务模型的思想,第 3 节对乐观嵌套工作流事务模型进行形式化描述,第 4 节对乐观嵌套工作流事务模型的正确性进行证明,最后对该模型的特点进行总结。

2 乐观嵌套事务模型

为克服目前工作流事务模型存在的不足,最大限度地满足工作流事务管理的需要,本文作者在文[7]中提出了乐观嵌套工作流事务模型,其主要思想主要包括以下三方面内容:

1. 在一个工作流应用中,事务(工作流的活动或任务项)从数据管理中心读取数据在各客户机上处理,处理完毕后再把结果回写到数据管理中心。因此可采用乐观并发控制方法来管理事务的执行机制,把一个事务分为三阶段执行:工作阶段、有效性认证阶段和可能的写阶段。工作阶段的任务是读

*)本技术成果受"陕西省自然科学基金"(项目编号:2005F46)和"西安科技攻关计划"(项目编号:GG05022)项目资助。董云卫 博士,副教授,研究兴趣:工作流技术、软件测试技术。

取要访问的数据，并建立该数据的临时版本，在该版本上进行操作，不会影响系统数据的完整性，因此，对工作阶段操作并不作任何的限制；事务的写阶段可能影响数据的完整性，需要进行限制，事务在执行写阶段操作前必须先进行有效性确认，只有通过有效性确认的事务才能进入写阶段的执行过程。

2. 为了保证事务读的一致性和写操作数据的正确性，为每一个事务建立一个时间戳，该时间戳具有两方面作用：一是授权事务在何时能够用私有版本数据更新公用版本数据；二是规定数据回写操作执行的最长允许时间。

表1 乐观嵌套工作流事务模型的操作原语表

	事务原语	原语语义
1	事务开始 Begin_Transaction	表示一个事务开始。
2	事务结束 End_Transaction	表示一个事务结束。
3	事务读 Read [a]	事务读取数据 a，并建立自己的一个私有数据版本 a′=a。
4	事务确认 Validation_ Transaction	进行事务有效性验证。
5	事务写 Write [a]	事务把其私有数据版本的操作结果回写到公共数据版本中。
6	事务发通知 Send_Signal	事务写阶段执行成功后向读取了旧数据，且未提交的事务发送一个通知。
7	请求提交 Reqest_Commit	子事务执行成功结束时调用，其目的子事务向它的父母事务请求提交。该原语之后跟随的是提交成功或失败原语。
8	事务提交 Commit	事务回写成功结束。
9	事务失败 Abort	读事务失败没有返回值或写事务回写失败，删除其私有数据版本。
10	创建并发子事务块 Par [T_i,…,T_j]	一个事务创建一个子事务块 Tset＝{T_i,…,T_j}时调用，Tset 中的事务并发执行。
11	创建串行事务块 Seq [T_i,…,T_j]	一个事务创建一个子事务块 Tset 时调用，并且这些子事务按顺序 T_i＜…＜T_j 串行执行。
12	创建选择执行事务块 Select [T_i,…,T_j]	一个事务创建一个子事务块 Tset 时调用，且这些子事务只有一个执行成功。
13	创建占优执行事务块 Dominant [T_i,…,T_j]	一个事务创建一个子事务块 Tset 时调用，并且这些子事务有序关系 T_i＜…＜T_j，只有关系前面的子事务执行失败后后面的事务才能执行，并且前面的事务执行成功提交，后面的事务将不再执行。
14	同意事务提交 Grant_commit [T_i]	一个事务同意其子事务 T_i 提交时调用。
15	强迫事务失败 Force_Abort [T_i]	一个事务强迫其子事务 Ti 失败时调用。
16	Dummy(op)	Op 是操作原语 10-13 中操作的一种或多种组合。

3. 对于事务访问的数据组织采用多版本控制协议，为每一个事务建立一个它或它的子事务需要访问的数据版本。事务的内部结构采用乐观嵌套工作流事务机制，事务所访问的数据项也嵌套处理。每一级事务在对某数据第一次读、写操作时，都根据上一级的该数据版本，建立属于本级事务的该数据版本；如果上一级的该数据版本不存在，还要根据上上一级的该数据版本，建立上一级和本级事务的该数据版本，依次类推。每一级事务的读、写访问都是在本级事务数据版本中进行。当子事务成功提交时，用该事务的数据版本更新其父事务数据版本中的相应数据；子事务失败后不进行提交，从而其父事务的数据版本不变，相当于进行了恢复。

一个乐观嵌套工作流事务被分为三个阶段执行，它由 9 个操作原语组成，如表 1 所示事务操作原语 1～9。同时还定义表示嵌套事务控制子事务执行的生成事务原语，如表 1 所示事务操作原语 10～16。

在乐观嵌套事务模型中，每一个事务必须是以原语 Begin_Transaction 开始，以原语 End_Transaction 结束，事务在执行原语 Commit 和 Abort 时以相互排斥的方式进行，如果叶子事务向父事务回写私有数据版本成功后，也要向所有读取了旧数据的其他事务发通知；没有执行写阶段操作时，叶子事务要么是以原语 Commit 结束，要么是以原语 Abort 结束，对于执行写阶段操作的叶子事务要么是以原语 Send_Signali 结束，要么是以原语 Abort 结束，一个包含了写阶段、工作阶段和有效性验证阶段的事务，其有效性验证阶段在所有工作阶段执行完成后才能执行，同时在可能的写阶段的所有操作执行前执行。

3 乐观嵌套工作流事务的形式化描述

乐观嵌套工作流事务描述了工作流事务之间的依赖关系，是对工作流执行过程中控制流的反映，它可以表示成为一棵由叶子事务和内部事务组成的事务树。内部事务又称为生成事务，可以由叶子事务生成，并管理叶子事务的提交或失败[8~10]。叶子事务的形式化定义为：

定义 1 一个事务 T_i 是由具有偏序关系$\ll_i$操作 OP_i 组成的一个集合，如果这种偏序关系$\ll_i$和操作 OP_i 满足如下条件，就称该事务 T_i 为叶子事务：

1. $OP_i \in T_i \subseteq I_i \cup \{Abort_i, Commit_i, Validationi\_ Transaction, Send\_Signal_i\ Request\_Commit_i\} \cup \{Begin\_ Transaction_i, End\_Transaction_i\}$，这里 I_i 属于集合{$Read_i$ [a]，Write $_i$[a] /a 是数据项}的幂集；.

2. $Begin\_Transaction_i \in T_i$，$\forall p \in T_i$，有 $Begin\_Transaction_i \ll_i p$ 成立；

3. $End\_Transaction_i \in T_i$，$\forall p \in T_i$，有 $p \ll_i End\_Transaction_i$ 成立；

4. $Abort_i \in T_i$，当且仅当 $Commit_i \notin T_i$；

5. 如果 $Write_i[a] \in T_i$ 并且有 $p = Commit_i \in T_i$，那么，不存在 $p' \in T_i$，$p' \neq p$ 且 $p' \neq Send\_Signal_i$ 有 $p \ll p' \ll_i Send\_Signal_i$ 成立；

6. 如果 Write $_i$[a]$\in T_i$，$p \in T_i$ 且 p＝ $Abort_i$ 或 $p = Send\_Signal_i$，那么，$\forall p' \in T_i$，$p' \neq p$，一定有：$p' \neq End\_Transaction_i$ 时，$p' \ll_i p \ll_i End\_Transaction_i$ 成立；如果 Write $_i$[a]$\notin T_i$，$p \in T_i$ 且 $p = Abort_i$ 或 $p = Commit_i$，那么，$\forall p' \in T_i$，$p' \neq p$，一定有：$p' \neq End\_Transaction_i$ 时，$p' \ll_i p \ll_i End\_Transaction_i$ 成立；

7. 如果 $Request\_Commit_i \in T_i$，并且 $p \in T_i$，这里 $p = Abort_i$ 或 $p = Commit_i$，那么不存在 $\forall p' \in T_i$，使得 $Request\_Commit_i \ll_i p' \ll_i p$；

8. 对于事务 T_i 及其所访问的任何数据 $a \in D$，如果 $Read_i[a] \in T_i$，$Write_i[a] \in T_i$，$Validation\_Transaction \in T_i$，那么 $Read_i[a] \ll_i Validation\_Transaction \ll_i Write_i[a]$。

一个乐观嵌套事务事务 T 可表示为一棵事务树 Tree (T)，事务树的节点用 Nodes(T)表示，叶子用 Leaves(T)表

示，事务 T_i 在事务 T 中的父母事务用 $Parent_T(T_i)$ 表示，用 $Ancestors_T(T_i)$ 表示事务 T_i 在事务 T 中的祖先集合(包括事务 T_i 和 $Parent_T(T_i)$)。事务树的每一条边连接一个生成事务 T_i 到一个被事务 T_i 生成的事务，生成事务 T_i 生成多少个事务就会有多少条边。事务树的结构刻画了工作流事务之间控制流和数据流的关系，它们可以用来生成新的事务。设 $T=\{T_1,\cdots,T_n\}$ 表示一个事务集合，事务 $T_1,\cdots,T_n$ 通过选择或组合事务生成操作原语 $E_i(T)\subseteq\{Grant\_Commit_i[T_j]$, $Force\_Abort_i[T_j]$, $Par[T_i,\cdots,T_j]$、$Seq[T_i,\cdots,T_j]$, $Select[T_i,\cdots,T_j]$, $Dominant[T_i,\cdots,T_j]\mid T_i,T_j\in T\}$ 形成的所有事务的集合 $S\Gamma_T(T_1,\cdots,T_n)$，就是用叶子事务 $\{T_1,\cdots,T_n\}$ 生成的事务，同样，我们把生成的事务连同其子事务看作是一个叶子事务，利用事务生成操作原语 $E_i(T)$ 也可以生成更高层次的事务。其形式化描述为：

定义 2 $F=\{T_1,\cdots,T_n\}$ 是叶子事务集合，基于关系 $\ll_i$ 的偏序集，利用操作原语 $E_i(F)$ 生成事务 T_i，满足 $T_i\in J_i\cup\{Abort_i, Commit_i, Validation\_Transaction, Send\_Signal_i, Request\_Commit_i\}\cup\{Begin\_Transaction_i, End\_Transaction_i\}$，则称 $T_i(i>1)$ 为 F 上生成的 i 级生成事务，其中：

1. 如果 $i=1$，那么，有 J_i 属于集合 $E_i(F)$ 的幂集；

2. 如果 $i>1$，那么，有 J_i 属于集合 $E_i(F\cup g\Gamma_{n-1})\cup\{Abort_i, Commit_i, Validation\_Transaction, Send\_Signal_i, Request\_Commit_i\}$ 的幂集，这里，$g\Gamma_{n-1}$ 是 $i-1$ 级的生成事务集合。

定义 2 表明：所有 i 级生成事务的子事务最多有 $i-1$ 级，在生成事务执行时，其中的叶子事务将遵循定义 1 中的条件，执行生成事务与其子事务(非叶子事务)将遵循如下规则：

1. 生成事务失败前，强迫它的所有子事务失败；

2. 每一个生成事务对于那些不能自动失败的子事务要么迫使它失败，要么同意它提交；

3. 当生成事务提出提交请求时，它结束前对子事务唯一能做的操作是同意它们提交或迫使它们失败。

在给出了叶子事务和生成事务的定义后，我们就可以给出嵌套乐观事务的形式化描述：

定义 3 设 F 是叶子事务集合，g 是在叶子事务集合 F 上的生成事务集合，使得 $\forall T_i\in g$，$\forall T_j\in children(T_i)$，都有 $T_j\in g\cup F$。在集合 g 和 F 上的事务 T 是满足关系 $\ll$ 的偏序集合称 T 为乐观嵌套工作流事务，即有 $T=\{op\mid\exists T'\in g\cup F$，使得 $op\in T'\}$ 和 $\forall T_i\in g\cup F$，满足 $\ll_i\subseteq\ll$。

根据定义 3 我们可以把乐观嵌套工作流事务的执行规则表示为：

定义 4 设 F 是叶子事务集合，g 是在叶子事务集合 F 上的生成事务集合，T 是在集合 g 和 F 上的乐观嵌套工作流事务，如果 g 中的每一个生成事务执行都满足下面条件，则称乐观嵌套工作流事务 T 是良构的。

1. 有且只有一个事务 $T_i\in g\cup F$，使得 $\forall T_k\in g\cup F$，$T_i\notin children(T_k)$ 并且 $Request\_commit_i\notin(T_i)$；

2. $\forall T_j\in g\cup F$ 使得 $\exists T_k\in g\cup F$ 和 $\exists p\in T_j$，这里 $p\in\{Par_j[Tset], Seq_j[Tset], Select_j[Tset], Dominan_j[Tset]\}$，$T_j\notin Tset=\{T_1,\cdots,T_k,\cdots,T_n\}$。下面条件满足：

1) $p<Begin\_Transactionk$；

2) $\forall T_k\notin Tset$，$End\_Transaction_h<Begin\_Transaction_j$；

3)如果 $Request\_commit_i\in T_i$，$T_i\in g$，那么，$\forall T_k(children(T_i)$ 并且 $\exists p\in T_k$，使得 $End\_Transaction_k<Request\_Commit_i$ 成立；

4)如果 $Abort_i\in T_i$，$T_i\in g$，那么 $\forall T_k(children(T_i)$，$Abort_k\in T_k$，并且 $Abort_k<Abort_i$；

5. 如果 $Commit_i\in T_i$，$T_i\in g$，那么，$T_k\in children(T_i)$，下面条件之一成立：

1) $Force\_Abort_i(T_k)\in T_i$，$Abort_k\in T_k$，并且 $Force\_Abort_i(T_k)<Abort_k<Commit_i$；

2)$Grant\_Commit_i(T_k)\in T_i$，$Request\_Commit_k(T_k)$，并且 $Request\_Commit_k<Grant\_Commit_i(T_k)<Commit_k<Commit_i$；

3) $Force\_Abort_i(T_k)\notin T_i$，$Grant\_Commit_i(T_k)\notin T_i$，$Abort_k(T_k)$，并且 $Abort_k<Request\_Commit_i$。

4 乐观嵌套工作流事务的串行调度

用乐观嵌套事务原语定义或生成的嵌套事务都是良构的，我们可以按一定的规则对乐观嵌套事务进行层次化，通过嵌套事务的层次特性实现事务操作的串行化，进而保证了事务的一致性。下面就这些属性进行讨论。

有序层次公理 设 T 是一个乐观嵌套工作流事务，它的一个有序层次就是一个二元组 $(\Gamma,\rightarrow)$，事务树节点集合 $\Gamma=Nodes(T)$，关系 $\rightarrow$ 是在 $Nodes(T)$ 上的一个非自反和非对称的关系，并满足如下三条公理：

1)有序性：对任何一个事务和它的父母事务都有父子秩序，即 $\forall T_i\in\Gamma$，$T_i\rightarrow Parent_T(T_i)$ 成立；

2)传递性：$\forall T_i,T_j,T_k,\in\Gamma$ 有 $T_i\rightarrow T_j$，$T_j\rightarrow T_k$ 成立，必定有 $T_i\rightarrow T_k$ 成立；

3)授权性：如果有 $T_i\rightarrow T_j$，那么下列结论成立：

a)如果 $Parent_T(T_i)\notin Ancestors_T(T_j)$，那么一定有 $Parent_T(T_i)\rightarrow T_j$；

b) 如果 $Parent_T(T_j)\notin Ancestors_T(T_i)$，那么一定有 $T_i\rightarrow Parent_T(T_j)$。

上面公理中，公理 1)强调乐观嵌套工作流事务的执行顺序是以父-子关系进行的，也就是说，子事务要求在父事务结束之前结束，公理 2)则是表明这种执行结束关系是能够传递的，公理 3)则是把两个节点的执行结束关系的影响在层次中传播。

定理 1 在乐观嵌套工作流事务树或事务森林中，不存在由事务组成的圈。

证明：设事务 $T_{k1},\cdots,T_{kn}$ 是事务树 $Tree(T_0)$ 中的一个圈，即 $T_{k1\rightarrow k2},\cdots,T_{k-1}\rightarrow T_{kn}$，$T_{kn}\rightarrow T_{k0}$，从有序层次公理可知，除根事务 T_0 外，每一个事务 T_{ki} 都有：$T_{ki}\rightarrow Parent_T(0T_i)$。反复利用有序层次公理中的公理 2 和公理 3，则在圈中的任何两个事务 T_{ki} 和 T_{kj}，有关系 $T_{ki}\rightarrow T_{kj}$ 和关系 $T_{kj}\rightarrow T_{ki}$，即事务 T_{ki} 既是事务 T_{kj} 的祖先，又是事务 T_{kj} 的孩子，这与公理一矛盾。于是定理得证。

推论 1 乐观嵌套工作流事务不能对应循环活动。

$H=(\Gamma,\rightarrow)$ 是一个乐观嵌套工作流事务 T 上的一个有序层次。设 $T_i\in Nodes(T)$，$Tree^r(T_i)$ 是以事务 T_i 为树根的事务树 $Tree(T_i)$ 的一棵子树，并且 $Leaves(Tree^r(T_i))\subseteq Leaves(Tree(T_i))$。对应子事务树 $S=Tree^r(T_i)$ 的层次关系 $H^r=(S,\rightarrow_i)$ 就是有序层次 H 上的一个约束，相应地，关系 $\rightarrow_i$ 是 $\rightarrow$ 在 S 上的一个约束。如果 S 等于组成事务 T_i 的所有事务的集合 Γ_i，那么，约束 $(\Gamma_i,\rightarrow_i)$ 就是有序层次 $H=(\Gamma,$

→)关于事务 T_i 的一个子层次。根据文[9],我们可以得到如下性质:

定理 2 给定一个乐观嵌套工作流事务 T 上的有序层次 $H=(\Gamma,\rightarrow)$,它的一个约束$(\Gamma_i,\rightarrow_i)$也是一个有序层次。

孤立子层次公理 设 $H_i=(\Gamma_i,\rightarrow_i)$是有序层次 $H=(\Gamma,\rightarrow)$关于事务 T_i 的一个子层次,($T_{i1},T_{i2},\in\Gamma_i$,并且 $\forall T_{l1},T_{l2},(\Gamma\backslash\Gamma_i\backslash \text{Ancestors}_T(T_i)$,至少下列两个关系有一个不成立:1)$T_{i1}\rightarrow T_{l1}$;

2)$T_{l2}\rightarrow T_{i2}$。

这里"\"表示集合的差运算。有序层次 H 的子层次 $H_i=(\Gamma_i,\rightarrow i)$是孤立层次,在乐观嵌套事务树中,任何一个子层次都是孤立的。

一个孤立的、自反的和全序的有序层次 $H=(\Gamma,\rightarrow)$称为串行层次。由于串行层次的每一个子层次都是孤立的并且是全序的,因此乐观嵌套事务树的所有节点都是有序的,并且每一个子事务都满足孤立性条件。

定义 5(层次可串行化) 对于有序层次 $H=(\Gamma,\rightarrow)$,如果存在一个串行层次 $H^+=(\Gamma,\rightarrow^+)$,使得$\rightarrow\subseteq\rightarrow^+$成立,我们称该有序层次 $H=(\Gamma,\rightarrow)$是可串行化。

定理 3 一个有序层次 $H=(\Gamma,\rightarrow)$是可串行化,当且仅当该层次中的关系→是全序关系。

证明:(1)充分性:有序层次 $H=(\Gamma,\rightarrow)$的关系→是全序关系,则由全序关系的定义知有序层次 $H=(\Gamma,\rightarrow)$是自反;并且每一个子层次 $H_i(\Gamma_i,\rightarrow_i)$都是有序层次(这里$\rightarrow_i$ 表示关系→在子层次上的一个约束),也是自反的。由于→是偏序关系,$\rightarrow_i$ 也是偏序关系,根据定义孤立层次公理,$H_i=(\Gamma_i,\rightarrow_i)$是孤立的。

对于每一个孤立子层次,我们根据有序层次公理,对其所有事务全序排队$\rightarrow_i^+$,并且$\rightarrow_i\subseteq\rightarrow_i^+$,于是我们就可以得到一个串行层次 $H^+=(\Gamma,\rightarrow^+)$,其中 $\Gamma=\cup\Gamma_i$,$\rightarrow^+=\cup\rightarrow_i^+$,$\rightarrow=\cup\rightarrow_i$,$\rightarrow\subseteq\rightarrow_i$,于是充分性得证。

(2)必要性:有序层次 $H=(\Gamma,\rightarrow)$是可串行化的,一定存在一个串行层次 $H^+=(\Gamma,\rightarrow^+)$,使得$\rightarrow\subseteq\rightarrow^+$成立,并且,$H^+=(\Gamma,\rightarrow^+)$表示的事务树的所有节点都是有序的,并且每一个子事务都满足孤立性条件。于是我们就可以以其每一个字数根节点建立关系→一个约束 $H_i=(\Gamma_i,\rightarrow_i)$与孤立子层 $H_i=(\Gamma_i,\rightarrow_i^+)$对应,使得 $\Gamma=\cup\Gamma_i$,$\rightarrow=\cup\rightarrow_i$,$\rightarrow^+=\cup\rightarrow_i^+$,假设→不是偏序关系,$\rightarrow_i$ 也不一定是偏序关系,而$\rightarrow_i\subseteq\rightarrow_i^+$,于是有$\rightarrow\subseteq\rightarrow^+$,这与$\rightarrow^+$是全序关系矛盾,于是必要性得证。

综合(1)及(2),定理得证。

在事务树中不同节点的两个事务,它们包含的操作对同一个数据项访问,其中有一个事务执行写阶段,称这两个事务是冲突的。显然,在乐观嵌套工作流事务中,只有由执行原语 Par_j 生成的子事务才可能是冲突的。

定义 6(执行顺序) 设 F 是叶子事务集合,g 是在叶子事务集合 F 上的生成事务集合,T 是在集合 g 和 F 上的乐观嵌套工作流事务,称$\rightarrow=(\rightarrow_{op}\cup\rightarrow_{pol})$是事务 T 上的一个执行顺序,其中,$\rightarrow_{op}$是由冲突操作执行顺序定义的顺序关系,$\rightarrow_{pol}=\rightarrow_{child}\cup\rightarrow_{seq}\cup\rightarrow_{Validation(ST)}$,关系$\rightarrow_{op}$、$\rightarrow_{child}$、$\rightarrow_{seq}$、$\rightarrow_{Validation(ST)}$分别定义为:

1) 叶子事务 $T_i,T_j\in g\cup F$,如果 $\text{Validation}(T_i)<\text{Validation}(T_j)$,则有关系 $T_i\rightarrow_{op}T_j$ 成立;

2)$\forall T_i\in g\cup F,T_k\in\text{children}(T_i)$,关系 $T_k\rightarrow_{child}T_i$ 成立;

3)$\forall T_i,T_j,T_k,\in g\cup F$,使得$\{\text{Seq}_k[T_i,\cdots,T_j],\text{Select}_k[T_i,\cdots,T_j],\text{Dominant}[T_i,\cdots,T_j]\}\cap T_k\neq\phi$,$\forall T_m\in\text{tree}(T_i)$,$\forall T_n\in\text{tree}(T_j)$有关系 $T_m\rightarrow_{seq}T_n$ 成立;

4)设 $T_i\rightarrow_{op}T_j$,如果被事务 T_i 和事务 T_j 执行的操作对于乐观并发控制协议是冲突的,那么,$\forall T_k\in\text{ancestors}_T(T_i)\backslash\text{ancestors}_T(T_j)$,并且,$\forall T_l\in\text{ancestors}_T(T_j)\backslash\text{ancestors}_T(T_i)$,关系 $T_k\rightarrow_{\text{Validation(ST)}}T_l$ 成立,即子事务的时间戳优先顺序父母事务能够保留。

关系$\rightarrow_{child}$是指父母事务要结束时必须等待子事务结束后才能结束,$\rightarrow_{seq}$是指两个顺序执行事务的顺序,而顺序$\rightarrow_{Select}$和$\rightarrow_{Dominant}$可以看作是顺序$\rightarrow_{seq}$的特例,$\rightarrow_{Validation(ST)}$是指事务满足乐观并发控制协议的顺序。

乐观嵌套工作流事务 T 上的一个有序层次 $H=(\Gamma,\rightarrow)$称作一个执行历史,其中→是事务 T 上的一个执行顺序。有序层次 H=(Γ,→)是事务 T 上的一个执行历史,事务 T 的串行图是以 *Nodes*(T)为节点,以二元组(T_i,T_j)组成边 $T_i\rightarrow T_j$ 的有向图。

定理 4 设 F 是叶子事务集合,g 是在叶子事务集合 F 上的生成事务集合,T 是在集合 g 和 F 上的乐观嵌套工作流事务,如果 $H=(\Gamma,\rightarrow)$是事务 T 上的一个执行历史,那么,H 是串行的。

证明:假设 H 不是串行的,则存在事务一个有限序列$\{T_i,T_{i+1},\cdots,T_k,T_i\}$,其中每一个 $T_i\in g\cup F$,使得 $T_i\rightarrow T_{i+1}\rightarrow T_{i+2}\rightarrow\cdots,T_{i+2}\rightarrow T_i$,因此,由事务序列组成一个 H 串行图中存在一个圆,利用事务的传递性公理可知关系 $T_i\rightarrow T_i$ 成立。

由于乐观嵌套工作流事务的串行图对应一个等待图,而图中存在一个圆,这意味着事务 T_i 不会结束,事务 T_i 也就不可能提交,与 H 是一个执行历史相矛盾。定理得证。

从定理 4 可以看到:利用乐观嵌套模型来调度事务是可串行调度的,因此它能够保证数据的一致性。

定义 7 设 F 是叶子事务集合,g 在叶子事务集合 F 上的生成事务集合,T 是在集合 g 和 F 上的乐观嵌套工作流事务,$\forall T_i,T_j\in g\cup F$,如果 $\text{Validation}(T_i)<\text{Validation}(T_j)$,有 $\text{commit}[T_i(a)]>\text{Read}[T_j(a)]$,则称事务 T 的执行是无级联调度的。

定理 5 乐观嵌套工作流事务的执行是无级联调度的(Cascadeless)。

证明:设执行顺序 $H=(\Gamma,\rightarrow)$是事务 $\Gamma=\{T_1,\cdots,T_i,\cdots,T_n\}$的一个执行顺序,$H$ 中的任何一个事务 T_i 包含的有子事务$(T_{i1},\cdots,T_{ik})$或操作 $OP_i=\{op_i^1,\cdots,op_i^k\}$,由定理 4 知有全序关系:$\text{Validation}(T_1)<\cdots<\text{Validation}(T_i)<\cdots<\text{Validation}(T_n)$。

任意两个子事务 $T_i^1,T_k^2\in g\cup F$,设它们不属于同一个事务,则存在事务 T_{i1} 和 T_{i2},$T_i^1\in T_{i1}$,$T_k^2\in T_{i2}$,有 $\text{Validation}(T_{i1})<\text{Validation}(T_{i2})$,显然有:$\text{Validation}(T_i^1)<\text{Validation}(T_k^2)$。由乐观并发控制协议知道是事务 T_i^1 的所有写阶段操作不影响事务 T_k^2 的所有工阶段作操作。

T_i^1,T_k^2 是同一个事务的两个子事务,它们满足有序层次公理和孤立子层次公理,必有关系 $T_i^1\rightarrow T_k^2$(或 $T_k^2\rightarrow T_i^1$)。因此,有关系成立:$\text{Validation}(T_i^1)<\text{Validation}(T_k^2)$(或 $\text{Validation}(T_k^2)<\text{Validation}(T_i^1)$),由乐观并发控制协议知道,事

(下转第 195 页)

挖掘效率。

(2)在各站点上,使用基于约束的 Eclat 算法来挖掘满足约束的局部频繁项集。采用了垂直数据库结构,利用基于前缀的等价关系将概念格划分为较小的子概念格,在自底向上的频繁集搜索过程中,同时进行约束条件的检验。算法无需复杂的 hash 数据结构;在生成候选项集后,不需再进行剪枝;可以在内存中独立处理每一个子网格,挖掘效率较高,占用的内存比较小。

(3)DMCASE 算法在各站点均采用前缀树和位矩阵作为存储结构,使算法真正做到了只需 1 次扫描数据库。

(4)各站点独立挖掘满足约束条件的局部频繁项集,各站点之间不需要同步。

(5)采用 4.2 中的完备性验证方法,能够使用抽样算法挖掘出完备的满足约束条件的全局频繁项集。

(6)采用归纳学习的方法归并满足约束条件的局部频繁项集。各站点不需同步传输挖掘出的信息,归纳学习元通过不断地接收信息和学习发现满足约束的全局频繁项集,大大降低了网络的通信代价。

结束语 本文将抽样与基于约束的 Eclat 类算法相结合,提出了一种分布式约束性关联规则的挖掘算法——DMCASE 算法。与其它同类算法相比,DMCASE 算法具有挖掘效率高、只需 1 次扫描数据库等优越性能。本文仅以反单调约束为例,描述了 DMCASE 算法的实现。DMCASE 算法同样适用于基于单调性约束、简洁性约束和可转变约束的分布式环境下关联规则的挖掘问题。如对于单调性约束,只需在 DMCASE 算法中将负边界设定为非频繁项集即可。进一步的工作是如何将 DMCASE 算法在多种约束条件下的实现方法更加完善和优化,并且降低网络通信代价、提高归纳学习的效率以及将挖掘结果做到最优。

参 考 文 献

1 Agrawal R, Shafer J C. Parallel mining of Association rules: Design, Implementation, and Experience [J]. IEEE Transactions on Knowledge and Data Engineering, 1996, 8(6): 962～969

2 Agrawal R, Srikant R. Fast algorithms for mining association rules [A]. In: Proceedings of the 20th International Conference on Very Large Databases [C], Santiago, Chile, 1994. 487～499

3 Cheung D W, Han J, Ng V T, et al. A Fast Distributed algorithm for Mining Association Rules [A]. In: Proceedings of 1996 International Conference on Parallel and Distributed Information Systems [C]. Miami Beach, 1996. 31～44

4 Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation [A]. In: Proceedings of the ACM-SIGMOD Conference on Management of Data [C], Dallas, ACM Press, 2000. 1～12

5 Ng R T, Lakshmanan L V S, Han J, et al. Exploratory mining and pruning optimizations of constrained association rules [A]. In: Proc. 1998 ACM-SIGMOD Int Conf Management of Data [C]. Seattle, Washington, 1998. 13～24

6 Pei J, Han J. Can we push more constrained into frequent pattern mining? [A]. In: Proc. 2000 Int Conf Knowledge Discovery and Data Mining [C]. Boston, MA, 2000. 350～354

7 Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases [A]. In: Proc. of the VLDB Conference [C]. Zurich, 1995. 432～444

8 Schuster A, Wollff R. Communication-efficient Distributed Mining of Association Rules. In: Proc. of the 2001 ACM-SIGMOD Int Conf Management of Data [C]. Santa Barbara, 2001. 473～484

9 Schuster A, Wollff R, Trock D. A High-Performance Distributed Algorithm for Mining Association Rules [A]. In: Proc. of the 2001 ACM-SIGMOD Int Conf Management of Data [C]. Santa Barbara, 2003. 473～484

10 Toivonen H. Sampling large databases for association rules [A]. In: Proc. of the 22nd Int'l Conf on Very Large DataBases [C]. Mumbai, 1996. 134～145

11 Zaki M J. Scalable algorithms for association mining [J]. IEEE Transactions Knowledge and Data Engineering, 2000, 12(3): 372～390

12 王春花,黄厚宽. 利用抽样技术分布式开采可变精度的关联规则[J]. 计算机研究与发展,2000,37(9):1101～1106

13 杜剑峰,李宏,陈松乔,陈建二. 单调和反单调约束条件下关联规则的挖掘算法分析[J]. 计算机科学,2005,32(6):142～144

14 蔡智兴,徐光佑. 人工智能及其应用. 第二版[M]. 北京:清华大学出版社,1995

(上接第 110 页)

务 T_i^1 的所有写操作不影响事务 T_k^2 的所有读操作(或事务 T_k^2 的所有写操作不影响事务 T_i^1 的所有读操作)。

事务 T_i 的操作 op_i^t 可以看作是 T_i 的一个子事务处理。因此,执行顺序 $H=(\Gamma,\rightarrow)$ 中的任何两个事务 T_i, T_j,如果 Validation(T_i)< Validation(T_j),必有 commit[$T_i(a)$] > Read[$T_j(a)$],说明乐观嵌套工作流事务的执行是无级联调度的。

结束语 乐观嵌套事务模型在协同工作流管理系统 Synchroflow 中实现事务管理。协同工作流管理系统采用信牌驱动工作流模型来实现对工作流程的建模。我们在工作流建模模型的操作原语与乐观嵌套事务模型的操作原语间建立了二者之间的对应关系,通过对应关系能够把工作流过程定义转化成为乐观嵌套事务树或事务森林。这种转换使得事务工作流执行过程中,其操作原语和乐观事务模型的操作原语是一致的,工作流活动的转移控制与乐观嵌套事务模型的子嵌套事务的生成过程及其表示方式也是一致的。

协同工作流管理系统中采取流程运行时分别对流程与活动进行实例化的策略,而工作流活动或活动组对应了工作流事务或嵌套事务,因此,工作流事务的时间戳可在工作流活动实例化时指定,并且根据工作流活动的类型差异,对工作流事务的时间戳的持久性进行分类管理,来保证长时间执行事务的正确执行。乐观嵌套事务模型在事务工作流应用过程中处理能力强,并且在定义嵌套事务时简单方便,易于实现。

参 考 文 献

1 Schuldt H, Alonso G, Beeri C, Schek H J. Atomicity and Isolation for Transaction Processes. ACM Transaction on Database Systems, 2002, 27(1): 63～116

2 丁柯,金蓓弘,冯玉琳. 事务工作流的建模和分析. 计算机学报,2003,26(10)

3 Wachter H, Reuter A. The ConTract Model, Database Transaction Models for Advanced Applications. In: A. H. Elmagarmid, ed. Morgan Kaufmann, San Jose, eds. 1992. 123～158

4 沈备军,陈诚,居德华. 基于规则的软件过程事务模型工作. 软件学报,2002,13(1)

5 Thomasian A. Distributed Optimistic Concurrency Control Methods for High- Performance Transaction Processing. IEEE Transaction on Knowledge and Data Engineering, 1998, 10(1)

6 Kung H T, Robinson J T. On Optimistic Methods for Concurrency Control. ACM Transaction on Database Systems, 1981, 6(2): 213～226

7 董云卫,郝克刚. 一种乐观嵌套工作流事务模型. 计算机科学,2005,32(8)

8 Eliot J, Moss B. Nested Transaction: an Approach to Reliable Distributed Computing: [MIT/LCS/RT-260]. 1981

9 Bertino E, Catania B, Ferrari E. A Nested Transaction Model for Multilevel Secure Database Management System. ACM Transaction on Information and System Security, 2001, 4(4): 321～370

10. Weikum G, Deacom A, Schaad W, Schek H. Open Nested Transaction in Federated Database System. Data Engineering IEEE Computer Society, 1992, 16(2)