计算机科学 1997 Vol. 24 №. 2

一个现代数据库事务模型*)

胡国玲 刘云生

(华中理工大学计算机系 武汉 430074)

摘 要 For the requirements of the databases in the advanced applications, such as communication switching, stock trading, rader tracing etc., this paper presents a new extended transaction model——Advanced Transaction Model(ATM). In ATM, a transaction has the properties of structural complexity, functional alternativity, execution depedency and result compensatability. And we presents the formal specification and the execution model of ATM.

关键词 Transaction, Advanced transaction, Execution model, Transaction model.

事务是数据库系统的基本工作单元，在传统数据库系统中，被建模为一个原子、平淡的操作序列，具有原子性、隔离性、一致性和持久性(简称 ACID 特性)[1]。这四种特性使得事务能确保将数据库从一种一致状态转换到另一种一致状态。但是这种传统的事务模型对诸如各种 CAx，电话交换、电力或数据网管理、空中交通控制、雷达跟踪、指挥控制系统、证券交易等这样一些现代(非传统)数据库应用领域已不适用，因为这些应用领域中的活动(任务)往往具有各种语义结构上的彼此联系，比如，常常包含一些反复的、"长寿"的、彼此合作的、嵌套的或有内部构造的活动。

本文针对各种现代(非传统)应用领域的要求提出了一种新的事务模型——现代事务模型 ATM (Advanced Transaction Model)，它扩展了传统事务的 ACID 特性。在 ATM 中，事务不必是原子的、隔离的、短暂的。下面我们首先讨论 ATM 事务的四个重要特性，然后给出该模型的形式化描述，最后介绍其执行模型。

一、现代事务特征

现代应用领域中，"原子、平淡的数据库操作序列"的传统事务概念已不适用，事务表现出了四种重要特征：结构复杂性、功能替代性、执行依赖性和结果补偿性。我们将具有这四种特性的事务叫现代事务。

1.1 结构复杂性

现代应用领域中常常包含一些反复、"长寿"、彼此合作的活动，活动与活动之间又往往存在诸如多层、嵌套、分裂、合并等各种结构上的彼此联系，对这样一些活动传统的原子事务模型根本无法描述。在 ATM 中，我们用现代事务来建模它们，一个现代事务可以分成为若干子事务，子事务又可以是现代事务，从而形成嵌套、层次等结构，同时子事务之间又可以具有各种彼此协作关系，详见文[2]。

1.2 功能替代性

一个非传统应用常常由若干任务组成，而一个任务又可以通过不同的途径来实现。比如，从武汉旅行到上海这样一个任务至少可以有三种途径：乘轮船、乘火车、乘飞机。一个强有力的事务模型应该允许用户表述其为实现同一目标而作的各种选择。在 ATM 中一个应用被建模为一个事务，它的每一个任务被建模为一组功能等价的子事务。能完成同一任务的子事务互称是功能可替代(或等价)的，它们构成该任务的替代集。若一个任务的替代集中的子事务之一能成功执行，则说该任务可完成。若对应一个事务的所有任务可完成，则说该事务是成功(可提交)的。功能替代性导致了事务有多条执行路径，因此 ATM 中事务的执行具有不确定性，即一个事务成功的执行路径依赖于在事务执行过程中失败的发生，且即便某些子事务的执行失败了，该事务仍可能顺利提交。这也体现了 ATM 的健壮性。

1.3 执行依赖性

执行依赖性是现代事务的另一重要特性。在

*)国家自然科学基金与国防预研资助项目

ATM 中，一个事务的子事务之间以及子事务对外界可以具有三种主要类型的依赖性——成功依赖、失败依赖、外部依赖。设有一子事务的集合 $T=\{at_1, at_2, \cdots, at_n\}$，我们有：

定义 1（成功依赖） 如果 $at_i \in T$ 在 $at_j \in T$ 执行成功之后才能执行，则称 at_i 成功依赖于 at_j，记为 $at_i \propto_s at_j$。

定义 2（失败依赖） 如果 $at_i \in T$ 在 $at_j \in T$ 执行失败之后方能执行，则称 at_i 失败依赖于 at_j，记为 $at_i \propto_f at_j$。

定义 3（外部依赖） 令 x_i 为一个与子事务 $at_i \in T$ 本身无关的谓词（比如 x_i 可以是一个关于时间的谓词），我们称之为 at_i 的外部谓词。若仅当 x_i 为真时 at_i 才能执行，则说 at_i 外部依赖于 x_i，记为 $at_i \propto_e x_i$。所有这样的外部谓词 x_i 的集合称为 AT 的外部依赖集。

此外还可以有“提交依赖”、“夭折依赖”等，具体见文[2]。

1.4 结果补偿性

传统事务模型的原子性和隔离性对短事务是非常适用的，但如果用于长事务便会带来诸如长时间占用大量资源从而降低事务处理并发度、回滚（roll-back）操作导致大的牺牲等一系列的问题。因此必须对其进行一定的扩展，这其中如何选择长事务的隔离粒度和原子粒度是问题的关键。

在 ATM 中，事务的子事务可分为两大类型：可补偿和不可补偿的。一个子事务是可补偿的，当且仅当存在一个语义上能“抹掉”（undo）它提交后对数据库所产生的影响的子事务或事务。如果不存在这样的一个子事务或事务，则该子事务是不可补偿的。一个事务的任一子事务要么是可补偿型的，要么是不可补偿型的，所以总是“有型的”。于是任一事务也总是“有型的”：(1)可补偿—所有子事务是可补偿的；(2)不可补偿—所有子事务不可补偿；(3)混合——一些子事务是可补偿的，另一些是不可补偿的。由于在 ATM 中事务总是“有型”的，因此我们可以针对不同的事务类型确定不同的原子和隔离粒度，从而提高事务处理的并发度和降低事务回滚操作的代价。

二、现代事务的形式模型

本节将给出 ATM 的形式化描述，并讨论在下一节执行模型中将要用到的一些概念和定理。下面在概念的叙述中，凡在前面已定义了的概念及其符号表示将直接使用，不再重复说明。在 ATM 中，我们有：

定义 4（现代事务） 一现代事务 AT 是一个五元组 $(T, \propto_s, \propto_f, X, CS)$。其中：(a)$T=\{at_1, at_2, \cdots, at_n\}$ 为有型的子事务的有限集，称之为 AT 的域。(b)$\propto_s$ 为 T 上的偏序关系。它定义了 T 中各元素之间的成功依赖。(c)$\propto_f$ 为 T 上的偏序关系。它定义了 T 中各元素之间的失败依赖。(d)$X=\{x_1, x_2, \cdots, x_n\}$ 为关于 T 的一组外部谓词。它定义了 T 的外部依赖集，即 $\forall x_i \in X$ 均 $\exists at_i \in T$ 使得 $at_i \propto_e x_i$。(e)$CS=\{cs_i \mid 1 \leqslant i \leqslant m, cs_i \subset T\}$ 为 AT 的提交集的集合，其中每个 cs_i 称为 AT 的一个提交集。

定义 5（提交集） 设 $RS=\{rs_i \mid 1 \leqslant i \leqslant k, rs_i \subset T\}$ 为 AT 的替代集的集合，若 $\forall rs_i \in RS(1 \leqslant i \leqslant k)$，均 $\exists at_j^i \in rs_i$ 能成功执行，则 AT 是可提交的。$cs_i=\{at_j^i \mid at_j^i \in rs_i, 1 \leqslant i \leqslant k\}$ 称为 AT 的一个提交集。

定义 6（现代事务的执行状态） AT 在时刻 t 的执行状态为一个 n 元组 $(e_1^t, e_2^t, \cdots, e_n^t)$，其中 e_i^t 为子事务 at_i 在时刻 t 的执行状态，它可以具有下列值：

$e_i^t =$
- N　表示在时刻 t，at_i 尚未被批准执行
- C　表示 at_i 在时刻 t 已提交
- R　表示 at_i 执行成功且在时刻 t 准备提交（但尚未提交）
- A　表示 at_i 在时刻 t 失败了（已夭折）
- E　表示 at_i 在时刻 t 正在执行

在 AT 尚未被调度执行时，其每个子事务 at_i 均处于状态 N；当 at_i 被启动之后，它处于状态 E；当执行终止，如果 at_i 是可补偿的子事务，则它到达状态 C 或 A，如果 at_i 是不可补偿子事务，则它到达状态 R 或 A。C 和 A 是最终状态，不再变化。R 是两段式提交（2PC）协议的准备提交状态。

定义 7（初始执行状态） 具有性质 $e_i^{t_0}=N(i=1, \cdots, n)$ 的执行状态 $e^{t_0}=(e_1^{t_0}, e_2^{t_0}, \cdots, e_n^{t_0})$ 称为 AT 的初始执行状态。

定义 8（经历） AT 的一个经历（history）H_k 是 AT 的一个可执行状态的集合 $\{e^{t_0}, e^{t_1}, \cdots, e^{t_k}\}$，其中 $t_j(j=1, 2, \cdots, n)$ 是按如下方式选择的：$e^{t_j} \neq e^{t_{j-1}}$ 且不存在一个时刻 t'，$t_{j-1}<t'<t_j$，使 $e^{t'} \neq e^{t_{j-1}}$。

经历 H_k 反应了 AT 的执行状态的变迁。

定义 9（可接受状态） AT 的相应于提交集 cs_i 的可接受状态 AC_i 是一个 n 元组 $(c_{i,1}, c_{i,2}, \cdots, c_{i,n})$，这里，对 $j=1, 2, \cdots, n$，有

$c_{i,j} =$
- C　如果 $at_j \in cs_i$ 且 at_j 是可补偿的
- R　如果 $at_j \in cs_i$ 且 at_j 是不可补偿的
- A　如果 $at_j \notin cs_i$ 且 $\exists at_k \in cs_i$，$at_k \propto_f at_j (j \neq k)$
- Y　其他任何子事务的执行状态

其中，C，R，A 与定义 5 具有同样的意义。

定义 10（可执行子事务） 令 AT 在时刻 t 的执行状态为 $e^t=\{e_1^t,e_2^t,\cdots,e_n^t\}$，$AC_i=(c_{i,1},c_{i,2},\cdots,c_{i,n})$ 是 AT 的关于 $cs_i\in CS$ 的可接受状态，如果下列条件满足，则说子事务 $at_j\in T$ 在时刻 t 相应于 cs_i 是可执行的（其中，$k=1,2,\cdots,n$；$k\neq j$，$Z_k,Z_j\in\{R,C\}$）：(a)在时刻 t，对 at_j 有 $x_j\in X$ 为真。(b)如果∃ at_k，使 $at_j\infty_s at_k$ 且 $c_{i,k}=Z_k$、$c_{i,j}=Z_j$，则 $e_k^t=Z_k$。(c)如果∃ at_k，使 $at_j\infty_f at_k$ 且 $c_{i,k}=A$，则 $e_k^t=A$。

定义 11（正确经历） 设 $H_k=\{e^{t_0},e^{t_1},\cdots,e^{t_k}\}$ 为 AT 的一个经历，如果下列条件满足，则称 H_k 是 AT 相对于 $cs_i\in CS$ 的正确经历：(a)$H_{k-1}=\{e^{t_0},e^{t_1},\cdots,e^{t_{k-1}}\}$ 是 AT 相对于 cs_i 的正确经历；(b)如果 $at_j\in cs_i$ 且 $e_j^{t_k}\neq N(j=1,2,\cdots,n)$，则 at_j 相对于 cs_i 在时刻 t_k 是可执行的。

定义 12（提交集状态） 令 $e^t=(e_1^t,e_2^t,\cdots,e_n^t)$ 为 AT 在时刻 t 的执行状态，$AC_i=(c_{i,1},c_{i,2},\cdots,c_{i,n})$ 为 AT 相对于 $cs_i(i=1,2,\cdots,m)$ 的可接受状态。AT 在时刻 t 的提交集状态 d^t 是一个 m 元组 $(d_1^t,d_2^t,\cdots,d_m^t)$。其中：

$$d_i^t=\begin{cases}\text{如果}\exists\, c_{i,k}=A\text{ 且 }e_k^t=Z\text{ 或}\exists\, c_{i,k}=Z\text{ 且 }e_k^t=A(Z\in\{R,C\}) & \\ S\quad \text{如果对所有 } j,1\leqslant j\leqslant n \text{ 均有 } c_{i,j}=e_j^t\text{ 或 }c_{i,j}=Y\text{ 成立} & \\ O\quad \text{其他} & \end{cases}$$

定理 1（最后提交集状态） 设 H_k 为 AT 的一个经历，$e^{t_j}=(e_1^{t_j},e_2^{t_j},\cdots,e_n^{t_j})$ 为在 H_k 中的一个执行状态；$d^{t_j}=(d_1^{t_j},d_2^{t_j},\cdots,d_m^{t_j})$ 为在 $t_j(j=1,2,\cdots,k)$ 时刻的提交集状态。若不存在一个 j'：$j<j'\leqslant k$ 使 $e_u^{t_j}=R$ 且 $e_u^{t_{j'}}\neq R(u=1,2,\cdots,n)$，那么如果 $d_i^{t_k}=S(d_i^{t_k}=F)$，则对 $i=1,2,\cdots,m$；$a=0,1,\cdots,k$；$a\leqslant w\leqslant k$ 有 $d_i^{t_w}=S(d_i^{t_w}=F)$。

证明：由条件可知 H_k 中不会包含子事务从执行状态 R 到 A 或 C 的状态变迁，而 A 和 C 又是事务的最终执行状态，因此 H_k 中一定不会存在子事务从执行状态 $Z_a\in\{R,A,C\}$ 到 $Z_b\in\{R,A,C\}$ 的状态变迁，又由定义 12 可知定理成立。 □

定义 13（可启动子事务） 设 $d^t=\{d_1^t,d_2^t,\cdots,d_m^t\}$ 为 AT 在时刻 t 的提交集状态。子事务 $at_i(i=1,2,\cdots,n)$ 是可启动的，当且仅当对所有 $j,1\leqslant j\leqslant m$ 均有 $d_j^t\neq S$ 或 at_i 相对于 cs_j 在时刻 t 可执行。

定义 14（成功，失败） 设 H_k 为 AT 的一个经历，$e^{t_j}=\{e_1^{t_j},e_2^{t_j},\cdots,e_n^{t_j}\}\in H_k(j=0,1,2,\cdots,k)$，$Ac_i=(c_{i,1},c_{i,2},\cdots,c_{i,n})$ 为相对于 cs_i 的可接受状态。

如果下列条件满足，则说 AT 相对于被选出的提交集 $cs_i\in CS$ 执行成功且其执行时间为 $t_c=t_k-t_0$。(a)H_k 为相对于 cs_i 的正确经历。(b)对任意 $j,1\leqslant j\leqslant n$，均有 $c_{i,j}=e_j^{t_k}$ 或 $c_{i,j}=Y$。(c)∃ $j,1\leqslant j\leqslant n$ 使 $c_{i,j}\neq e_j^{t_{k-1}}$ 且 $c_{i,j}\neq Y$。

如果下列条件满足，则说 AT 在执行时间 $t_c=t_{k'}-t_1$ 内失败：(a)不存在一个经历 $H_{k''}(k<k')$ 且 AT 在执行时间 $t_{k''}-t_1$ 内执行成功$(1<k''<k')$。(b)AT 在执行时间 $t_{k-1}-t_1$ 内没有失败。

当一个现代事务 AT 执行成功时，便选择了一个提交集 cs_i。AT 提交时，所有属于 cs_i 并且处于准备提交状态的子事务必须被提交；所有不属于 cs_i 的子事务则要么处于尚未启动状态，要么就是被夭折或被补偿。如果 AT 夭折，则所有已经启动的子事务都必须被夭折或被补偿。

定义 15（完成事务） 令 $e^t=(e_1^t,e_2^t,\cdots,e_n^t)$ 为 AT 在 t 时刻的执行状态，如果下列条件之一满足，则说 AT 在时刻 t 已完成：(a)AT 相对于一个被选择的提交集 cs_i 已执行成功。(b)AT 已失败$(cs_i=\Phi)$。

如果 AT 相对于被选择的提交集 cs_i 执行成功且提交，则对 $i=1,2,\cdots,n$ 有：(1)$e_i^t=A$ 或 $e_i^t=C$ 或 $e_i^t=N$；(2)如果 $at_i\in cs_i$ 则 $e_i^t=C$；(3)如果 $at_i\in cs_i$ 且 at_i 是不可补偿的，则 $e_i^t\neq C$；(4)如果 $at_i\in cs_i$ 且 $e_i^t=C$，则与 at_i 相联系补偿活动 ca_i 在时刻 t 执行完成；(5)不存在一个时刻 $t'>t$ 使 $e_i^t\neq e_i^{t'}$；

否则 $cs_i=\Phi$。

上面，我们给出了现代事务模型的形式描述，它还不是执行模型。确立现代事务的执行模型的关键是确立提交集的选择和产生一个正确的经历之间的彼此相关性。在一个现代事务开始执行之前选择一个提交集是没有意义的，因为这将意味着失去现代事务的功能替代性。在下一节我们将给出现代事务的执行模型，它对被选出的提交集都能确保产生一个正确的经历。

三、执行模型

本节将给出 ATM 的执行模型。我们用 type(AT)来表示一个现代事务的类型，type(AT)=C 表示 AT 是可补偿的，type(AT)=NC 表示 AT 是不可补偿的，type(AT)=COM 表示 AT 是混合型的。一个现代事务的执行由算法 1 描述。

算法 1

输入：一个现代事务 AT，其域 $T=\{at_1,at_2,\cdots,at_n\}$，其提交集的集合 $CS=\{cs_1,cs_2,\cdots,cs_m\}$。

输出：一个被选出的提交集 cs_x。

变量：AT 的当前执行状态 $e=(e_1,e_2,\cdots,e_n)$，提交集 cs_x，索引集 f 及一个布尔变量 b，AT 的当前提交集状态 $d=(d_1,d_2,\cdots,d_m)$。

1. 置 $e=(N,N,\cdots,N)$，$f=\Phi$，b=FALSE。

2. 执行步(a)到步(e)，当 b=TRUE 时停止执行步 2。

(a)对 $i=1,2,\cdots,n$ 执行步 I 和 I：(I)at_i 可启动之前等待，当 at_i 可启动时置 $e_i=E$。(I)如果 at_i 执行成功且 $type(at_i)=C$ 则置 $e_i=C$；如果 at_i 执行成功且 $type(t_i)=NC$，则置 $e_i=R$；否则(at_i 执行失败)置 $e_i=A$。

(b)如果 type(AT)=C 执行此步，等待直到在 d 中存在某个 $d_j=S$，然后置 $cs_x=cs_j$，b=TRUE。

(c)如果 type(AT)≠C 则执行步 I 至步 I：(I)等待直到在 d 中存在某个 $d_j=S$ 且 $j\notin f$，然后置 $cs_x=cs_j$；(I)等待用户输入 in，如果 in=TRUE，则置 b=TRUE；否则置 $f=f\cup\{j\}$，然后转(I)执行。/*由用户确定是否要提交 AT*/

(d)等待直到 $d_j\neq O(j=1,2,\cdots,m)$且如果 $d_j=S$ 则 $j\in f$ 成立；然后置 $cs_x=\Phi$，b=TRUE。

(e)等待直到 AT 在执行过程中夭折，则置 $cs_x=\Phi$，b=TRUE。

3. 对 $i=1,2,\cdots,n$ 执行步

(a)到步(d)：(a)如果 $at_i\in cs_x$ 且 $e_i=R$，则使 at_i 提交。

(b)如果 $e_i=E$ 或 $e_i=R$ 且 $at_i\notin cs_x$，则使 at_i 夭折。

(c)如果 $at_i\notin cs_x$ 且 $e_i=C$，则执行 at_i 的补偿活动 ca_i，等待直到 ca_i 执行完成提交后再继续往下执行。

(d)如果 $e_i\neq N$，则等待直到 at_i 执行完。

在上述算法中，我们假设仅仅显式等待某个事件发生的语句需要花费时间。则根据该算法我们可以有下述定理：

定理 2 如果给算法 1 输入一个现代事务 AT，且经过运行时间 t_r 算法终止并产生一个输出 $cs_x\neq\Phi$，则 AT 一定执行成功，且选出一个提交集 cs_x；如果算法产生输出 $cs_x=\Phi$ 且 AT 未被导致夭折，则在时刻 $t_c<t_r$ 算法执行失败。

本定理的证明过程较长，限于篇幅我们将在另文给出。□

定理 3 如果给算法 1 输入一个现代事务 AT，且在时刻 t 执行终止，则 AT 在时刻 t 执行完成。

证明：由算法 1 的步(3)和定义 15 可知定理成立。□

定理 4 如果给算法 1 输入现代事务 AT 且下列条件满足则算法 1 一定会终止。

(a)算法 1 的步 2 一定会产生一个输出。

(b)如果 type(AT)≠C，则对算法 1 的任一个输出最后都要决定是否提交 AT。

(c)如果 type(AT)=C，则对$\forall\ at_i\in T$，当其正被算法 1 执行时，与 at_i 相联系的补偿活动 ca_i 必须停止执行。

(d)对$\forall\ at_i\in T$，如果 at_i 处于准备提交状态，则不会被夭折。

证明：由于上述条件可以保证算法 1 的步 2(b)和步 3 不会永远等待，因此定理成立。□

结束语 我们针对各种现代应用领域的要求提出了一个扩展的事务模型——现代事务模型 ATM，从而弥补了传统事务模型的局限与无能，本文的主要工作在于：①提出了一个新的事务模型。②给出了该模型的形式化描述。③给出了它的执行模型。

参考文献

[1]P. K, Chyrsanthis et al., ACTA: A frame work for specifying and reasoning about transaction structure and behavior

[2]刘云生，关于实时数据库事务，软件学报，1995. 9