文章编号:1001 - 9081(2007)10 - 2433 - 04

基于冲突概率的组合服务事务混合并发控制算法

曾慧琼,李建华,许 甸,马 华
(中南大学 信息科学与工程学院,长沙 410075)
(zhq265@163.com)

摘 要:由于 Web 服务的松散耦合性及独立性的特点,组合服务中的两个服务可以并行运行。针对这一特点,结合乐观并发与悲观并发控制的优点,提出一种基于冲突发生概率的混合并发控制机制,以提高 Web 服务的并发度。由于充分考虑了不同组合服务实例的并发控制,以及同一种组合服务实例的并发控制,使得该机制能够确保在复杂的应用环境中,多个组合服务实例并发执行的正确性和一致性。

关键词:组合服务;Web 服务;并发控制;冲突
中图分类号: TP393.03;TP311 **文献标志码**:A

Mixed concurrency control algorithm of composed service transaction based on conflict probability

ZENG Hui-qiong, LI Jian-hua, XU Dian, MA Hua
(*School of Information Science and Engineering, Central South University, Changsha Hunan* 410083, *China*)

Abstract: Because Web Services are relaxed coupled and independent to each other, two single services in a composed service can execute concurrently. In allusion to the characteristic, combining the advantages of optimistic and pessimistic concurrency controls, this paper presented a mixed concurrency control mechanism on occurrence probability of conflict, which can improve the degree of concurrency. Taking full account of concurrency control of instances which are form the same composed Web Services or different composed Web Services, the mechanism can ensure correctness and consistency when multi instances of composed web service execute concurrently.

Key words: composed service; Web Service; concurrency control; conflict

0 引言

随着 Web 服务的广泛应用,Web 服务的事务处理越来越受人们的关注。Web 服务事务不同于传统事务,它放宽了事务的原子性和隔离性,执行时间较长。人们已经做了大量研究,并且提出了各种 Web 服务事务处理的模型和协议[1,2],然而这些模型都没谈及并发控制。因此,Web 服务提供者必须采取有效措施,提高服务执行的并发度,同时保证多个服务实例的正确性,降低失败补偿的成本。

在组合服务运行环境中,往往不止一个服务在运行。由于 Web 服务事务的长事务特点,在多个组合服务事务实例并发执行的环境中,保证各实例之间不受彼此的影响,正确高效的执行是非常重要的。服务组合往往是一个业务流程,其操作单元不仅仅是一个读操作或写操作,而是一个服务,这使得传统的使用可串行化准则来判断并发控制正确性的方法不适用。组合服务事务实例正确性应该以业务的正确性为准则,以提高组合服务事务并发度。同时由于服务的分布性、独立性,服务存在并行运行的机会。组合服务事务并发控制的关键问题在于保证两个或者多个服务正确的并行执行。两个彼此无关的服务完全可以并行执行。

在组合服务事务的运行环境中,并发控制包括两种情况:同一服务组合模式(即组合服务的编排)的事务实例并发和不同组合模式的事务实例并发。因此,在并发控制时,必须充分考虑各种可能的情况,同一种组合服务的事务实例之间、不同种组合服务的事务实例之间都可能发生冲突。冲突的发生将会导致事务的失败补偿,而且会大大降低组合服务实例之间的并发度,降低服务的效率,尤其在一个大的复杂的执行环境中,若不进行有效的控制,业务损失的代价无法估量。所以 Web 服务事务,尤其是组合服务事务急需一种有效的事务并发控制机制来保证多个服务事务实例并发有效的执行。

1 相关概念

1.1 组合服务事务

本文中组合服务采用嵌套事务模型。每一个组合服务代表一个业务流程,流程中每个活动代表一个完整业务功能。以图 1 为例,整个流程执行的活动序列构成一个组合服务事务 WCST,而一个活动则由服务调用等相关操作组成,如读取数据、服务调用、数据更新,这些服务操作则构成活动子事务 AT,是一个整体。活动子事务 AT 中调用的服务可以只是单个服务,也可以是组合服务,它由其服务提供者封装成一个服务子事务 ST。在不考虑参与者服务具体的事务行为的情况下,组合服务事务可以分为三层,即 WCST 层、AT 层、ST 层,

收稿日期:2007 - 04 - 23;**修回日期**:2007 - 06 - 23。

作者简介:曾慧琼(1981 -),女,湖南娄底人,硕士研究生,主要研究方向:Web 服务的事务性; 李建华(1963 -),男,湖南湘乡人,教授,博士研究生,主要研究方向:分布式计算与多媒体技术、软件工程; 许甸(1981 -),男,广东丰顺人,硕士,主要研究方向:工作流管理系统; 马华(1979 -),男,湖南株洲人,硕士,主要研究方向:工作流技术、Web 服务、企业应用集成。

其中 AT 可以看作是 WCST 层和 ST 层之间的过渡层,ST 层的事务行为由 AT 报告给 WCST。

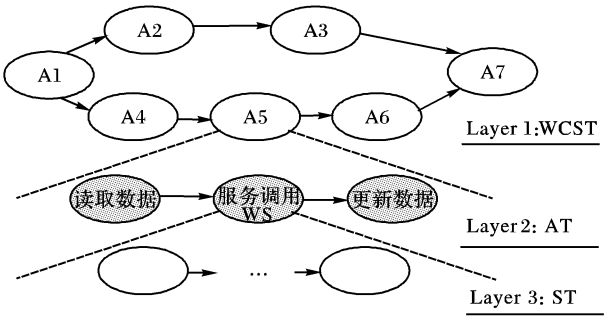

图 1 组合服务事务实例

本文不考虑服务参与者服务的事务处理。它的行为会影响到过渡层 AT 层的操作。如果服务失败中止,AT 层中的数据更新操作就不会再执行。同时不考虑单个组合服务事务实例内部各活动子事务实例间的冲突,即假设单个组合服务事务实例在无并发控制的情况下能正确执行。

显然,在组合服务的运行环境中,需要处理的主要是 WCST 层和 AT 层的事务。而并发则不仅在于多个服务实例的并发,还可以细化到 AT 活动子事务实例的并发。在不发生冲突时,AT 中的服务调用操作会使得多个服务能并行执行,从而提高并发度以及组合服务效率。

组合服务事务的一次执行叫做组合服务的一个事务实例。用 CT_i 表示一个组合服务事务,用 ct_{ij}(或 ct_i)表示该组合服务事务的一个运行实例,设组合服务事务集 $CT^* = \{CT_1, CT_2, \cdots, CT_i, \cdots\}$,则在组合服务的运行环境中的所有事务实例集 $\Omega^* = \{ct_{11}, ct_{12}, \cdots, ct_{1n}, ct_{21}, ct_{22}, \cdots, ct_{2n}, \cdots, ct_{i1}, ct_{i2}, \cdots, ct_{in}, \cdots\}$。

定义 1 对于来自不同的组合服务事务实例的两个活动子事务实例 at_i、at_j,若其事务序列$\langle \cdots at_i \cdots at_j \cdots \rangle$ 的执行效果与$\langle \cdots at_j \cdots at_i \cdots \rangle$ 的执行效果不相同,则称 at_i 和 at_j 是相互冲突的,记做 $at_i \ conf \ at_j$。at_i 和 at_j 相互冲突是因为要访问同一资源,设该资源为 r,则称 $at_i \ conf_r \ at_j$,即 at_i 和 at_j 关于 r 相互冲突。

活动子事务 at_i、at_j 的操作可以并发调度,当且仅当 $at_i \ unconf \ at_j$,即 at_i、at_j 不相互冲突。当前运行环境中的活动子事务实例之间的冲突越少,则并发度越大,服务效率也越高。

定义 2 组合服务事务实例运行上下文 $ctx = (CT^*_{ctx}, ct^*_{ctx}, \sigma, at^*_{ctx}, conf)$,其中 CT^*_{ctx} 为上下文中所涉及到的组合服务事务集,$CT^*_{ctx} \subseteq CT^*$;$ct^*_{ctx}$ 为与当前组合服务实例并发的所有事务实例集合;σ 为 ct^*_{ctx} 到 CT^*_{ctx} 的一个映射,可以表示为 $\sigma: ct^*_{ctx} \rightarrow CT^*_{ctx}$;$at^*_{ctx}$ 表示 ct^*_{ctx} 中所有事务实例的所有活动子事务实例集合;$conf$ 表示 at^*_{ctx} 中所有活动子事务实例之间的冲突关系,$conf \subseteq at^*_{ctx} \times at^*_{ctx}$,设 $pconf$ 为所有来自于同种或不同种组合服务事务实例的活动子事务实例间可能发生的冲突关系,则 $conf \subseteq pconf$。可以用一个图来描述上下文,见图 3。

定义 3 组合服务事务的一次执行可以用一个三元组来表示,$ct = (ctx, at_{ct}, \rightarrow)$,$ctx$ 表示其运行上下文,$at_{ct} \rightarrow$ 表示组成 ct 的所有活动子事务实例集,$\rightarrow$ 是 at_{ct} 中子事务实例的偏序关系,$\rightarrow \subseteq at_{ct} \times at_{ct}$。

定义 4 冲突类:冲突关系具有可传递性,即如果 $at_i \ conf_r \ at_j$,$at_j \ conf_r \ at_k$ 则 $at_i \ conf_r \ at_k$。利用冲突关系的可传递性,将所有相互冲突的 at 集合在一起形成一个冲突类,如 $\{at_i, at_j, at_k\}$ 就可以构成一个冲突类,冲突类中任何两个元素关于 r 相互冲突。

1.2 组合服务事务冲突模型

组合服务事务冲突模型是用于描述整个服务环境所提供的服务事务之间的冲突关系,是一个抽象的关系模型,文中用 WCM 表示。由此模型结合具体的实例运行场景,可以推导出实例运行上下文,从而得出所有 at 实例之间的冲突关系。

定义 5 WCM 为一个三元组,$WCM = \{CT^*, AT^*, \theta\}$,其中 CT^* 为服务环境所提供的所有组合服务事务集,AT^* 表示 CT^* 中所有服务事务的所有活动集,也即活动子事务集,$\theta \subseteq AT^* \times AT^*$,表示活动子事务之间的冲突关系。

下面以两个组合服务事务 $CT1$, $CT2$ 为例,用一个图形来描述 WCM,如图 2。其中单箭头表示同种活动子事务的实例间相互冲突,双箭头表示不同种活动子事务的实例间相互冲突。从图中可以看出 $AT^* = \{A11, A12, A21, A22, A23\}$,且 $A11, A12 \in CT1$, $A21, A22, A23 \in CT2$,而活动子事务之间的冲突关系 $\theta = \{\langle A11, A11\rangle, \langle A12, A12\rangle, \langle A21, A21\rangle, \langle A22, A22\rangle, \langle A23, A23\rangle, \langle A11, A22\rangle, \langle A12, A23\rangle, \langle A21, A23\rangle, \langle A23, A21\rangle\}$。从此关系中可以看出同一种服务组合模式的活动子事务实例冲突比较大,因为同一个活动子事务的实例一定是相互冲突的。

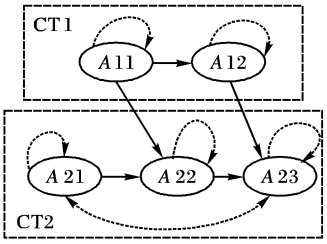

图 2 冲突模型

WCM 是事务实例运行上下文 ctx 的基础。其相互关系可以归纳为以下三点:1)$CT^*_{ctx} \subseteq CT^*$,$at^*_{ctx} \in AT^*$ 中所有活动子事务的所有实例集,θ 表示 AT 间的冲突关系,$conf$ 则表示 AT 实例 at 间的冲突关系。2)ctx 是相对于某个具体组合服务事务实例而言的,只有在运行时才能确定;而 WCM 则是基本服务环境,是静态的。3)根据组合服务冲突模型 WCM 以及组合服务事务实例运行时的 CT^*_{ctx}、ct^*_{ctx}、σ、at^*_{ctx},可以推导出该实例的上下文冲突关系 $conf$,具体推导在概率计算时会涉及。

定义 6 内在冲突:用于描述一个组合服务事务模型的内部冲突关系 $inner$,$inner \subseteq \theta$。在冲突模型中,当 CT^* 中只有一个组合服务事务时,$\theta = inner$。在图 2 中,$CT2$ 的内在冲突关系:$inner = \{\langle A21, A21\rangle, \langle A22, A22\rangle, \langle A23, A23\rangle, \langle A21, A23\rangle, \langle A23, A21\rangle\}$,$n(inner) = 5$。

定义 7 交叉冲突:用于描述任意两种组合服务事务模型 CT_i,CT_j 之间的冲突关系 $cross$,$cross \subseteq \theta$。在图 2 中,$CT1$, $CT2$ 之间的交叉冲突关系 $cross = \{\langle A11, A22\rangle, \langle A12, A23\rangle\}$,$n(cross) = 2$。

2 基于冲突概率的服务事务并发控制算法

2.1 冲突概率的计算

设实例 ct 运行时上下文中所有活动子事务实例间的实际冲突对数用 c 表示,则 c 为 $conf$ 中的元素个数,$c = n(conf)$。另假设在该上下文中所有可能的冲突对数用 C 表示,C 为 $pconf$ 的元素个数,$C = n(pconf)$,则有实例之间冲突概率为:

$$p = \lambda \cdot \frac{c}{C}。$$

其中 λ 为冲突系数,它与该时刻组合服务事务实例的个数 n 有关,n 越大,则 λ 越大,且 $0 \leqslant \lambda < 1$,当 $n = 1$ 时,有 $\lambda = 0$。

λ 是一个动态因素,根据具体场合而定,其值与实例个数直接相关。在不考虑服务的组合模式情况下,组合服务事务实例数越多,活动子事务实例间发生冲突的概率越大,因此引入冲突系数 λ 表示。

设当前有 n 个组合服务事务实例在运行,每个实例所具有的活动子事务实例数分别为 $m_1, m_2, \cdots, m_n$,可以推导出可能发生的冲突对数:

$$C = \frac{m_1\left(\sum_{i=1}^{n} m_i - m_1\right) + m_2\left(\sum_{i=1}^{n} m_i - m_2\right) + \cdots + m_n\left(\sum_{i=1}^{n} m_i - m_n\right)}{2} \Rightarrow$$

$$C = \frac{\left(\sum_{i=1}^{n} m_i\right)^2 - \sum_{i=1}^{n} m_i^2}{2}$$

实际冲突对数 c 的值和冲突模型密切相关。$c = c_{inner} + c_{cross}$,其中 c_{inner} 为同一种模式的组合服务事务实例间冲突对数,c_{cross} 为不同种模式的组合服务事务实例间对数。设当前运行环境中存在 $a = n(CT^*_{ctx})$ 种组合服务事务,每一种组合服务所存在是事务实例数分别为 $b_1, b_2, \cdots, b_a$ ($b_1 + b_2 + \cdots + b_a = n$),每种组合服务事务的内在冲突对数分别为 $c_1, c_2, \cdots, c_a$,每两种组合服务事务之间的冲突对数分别为 $c_{12}, c_{13}, \cdots, c_{1a}, c_{23}, \cdots, c_{2a}, c_{34}, \cdots, c_{(a-1)a}$,则这些事务实例产生的所有内在冲突对数:

$$c_{inner} = \frac{b_1(b_1 - 1) \times c_1 + b_2(b_2 - 1) \times c_2 + \cdots + b_a(b_a - 1) \times c_a}{2} \Rightarrow$$

$$c_{inner} = \frac{1}{2}\sum_{i=1}^{a} b_i(b_i - 1)c_i$$

这些事务实例产生的所有交叉冲突对数:

$$c_{cross} = c_{12}b_1b_2 + c_{13}b_1b_3 + \cdots + c_{1a}b_1b_a + c_{23}b_2b_3 + \cdots + c_{2a}b_2b_a + c_{34}b_3b_4 + \cdots + c_{(a-1)a}b_{(a-1)}b_a \Rightarrow c_{cross} = \sum_{j=1}^{a-1} b_j\left(\sum_{i=j+1}^{a} c_{ji}b_i\right)$$

下面以图2中的冲突模型为例来计算 C、c 的值,假设当前运行上下文中 $CT^*_{ctx} = CT^* = \{CT1, CT2\}$,$ct^*_{ctx} = \{ct1, ct2, ct3\}$,其中 $ct1 \in CT1, ct2, ct3 \in CT2$,$at^*_{ctx} = \{a11, a12, a21, a22, a23, a31, a32, a33\}$,则可以画出当前上下文的冲突关系图,如图3。

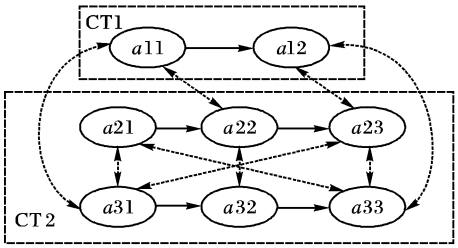

图3　组合服务事务实例冲突关系

从图3中可以数出 $c_{inner} = 5, c_{cross} = 4, c = 9, C = 21$。利用公式,因为 $a = 2, b_1 = 1, b_2 = 2, c_1 = 2, c_2 = 5, c_{12} = 2$,可以算出 $c_{inner} = \frac{1 \times (1-1) \times 2 + 2 \times (2-1) \times 5}{2} = 5, c_{cross} = 2 \times 1 \times 2 = 4, c = 4 + 5 = 9, C = \frac{(2+3+3)^2 - (2^2 + 3^2 + 3^2)}{2} = 21$,和图中数出的结果一致。

此时,由 $p = \lambda \cdot \frac{c_{inner} + c_{cross}}{C}$ 可以得出冲突概率。

本文从集合的角度考虑概率,由概率论的相关知识可知,事件 ω 的概率是子集 A 的元素个数记做 $n(A)$ 与全集 I 的元素个数 $n(I)$ 的比值,即 $P(\omega) = \frac{n(A)}{n(I)}$。本文中 $pconf$ 为所有活动子事务实例间可能发生冲突,构成了冲突事件 ω 的全集,$conf$ 则是实际发生的冲突集,$conf \subseteq pconf$,因此有 $P(\omega) = \frac{n(conf)}{n(pconf)} = \frac{c}{C}$。同时考虑到实例数对冲突概率的影响,引入一个冲突系数 λ,得 $p = \lambda \cdot \frac{c}{C}$。

2.2　组合服务事务的混合并发控制的算法

2.2.1　乐观并发控制

乐观并发控制应用于冲突概率很小,甚至不存在冲突的情况。它主要使活动子事务并发执行而很少发生阻塞。在传统的并发控制中主要有两种乐观的并发控制方法[5],即后向乐观并发控制(BOCC)和前向并发控制(FOCC)。本算法涉及到的乐观并发控制部分主要采取 BOCC。在 BOCC 有效确认下,组合服务事务实例 ct_i 确实是有效的(或"可接受的"),如果对于每一个已经提交的组合服务实例 ct_j,满足:

1) ct_j 在 ct_i 开始之前已经提交完成的;

2) 或者,若 ct_j 是在 ct_i 开始之后才提交完成的,则在 ct_i 的上下文 ctx 中,不存在$\langle at_i, at_j\rangle \in conf$,且 $at_i \in ct_i, at_j \in ct_j$。

当组合服务事务实例 ct_i 提交时,若系统检测到它不满足上述条件,则简单的放弃该事务并重新启动,否则正确提交。显然,在冲突发生概率很小或者根本不会发生冲突时,这种方法能大幅度提高服务的并发度,保证服务的效率。但是一旦系统检测到冲突,则要放弃组合服务事务实例,并对已提交的活动子事务实例从业务上提供补偿,这是需要一定代价的,尤其是商业性质的服务。

2.2.2　悲观的并发控制

在冲突概率较大的情况下,乐观并发控制会造成大量组合服务事务实例终止,引起对已提交活动子事务实例的大量补偿,代价不可估量,此时乐观并发控制不可行。为了避免过高的补偿代价,必须采取"保守的"悲观并发控制。它在识别到冲突时,使组合服务事务实例阻塞,当冲突不再存在时,再调度执行。这就无需补偿,但却牺牲了系统的效率。在极端的情况下,有可能只有一个组合服务事务实例在运行,其他所有的实例均被阻塞。

本算法采用冲突类锁的机制进行调度控制。假设每一个冲突类都有一把锁,即冲突类锁 cl,对于冲突类中的活动子事务实例,必须获取该 cl,才会有机会执行。悲观并发控制过程可以描述如下:

对于当前要调度执行的活动子事务 at_i,先检查它是否处于某一个冲突类 $confC$ 中(假设它至多属于一个冲突类),

1) 如果没有,则调度执行 at_i。

2) 如果有,则检查该冲突类是否已经被冲突类中其他活动子事务实例加锁:

a) 如果没有被加锁,则对该冲突类加锁,$cl(confC)$,并调度执行。

b) 如果已被加锁,则阻塞,直至现有的冲突类锁被释放。

对于活动子事务实例同时属于多个冲突类的情况,则只有该实例获取到所有的冲突类锁时,才能调度执行。为了防止死锁,任何活动子事务实例都不能在阻塞的状态持有冲突类锁,即它要么拥有全部冲突类锁,要么一把都不拥有。

2.2.3　调度算法

此调度算法有两个前提条件:1) 系统必须存在一个冲突

模型 WCM,这是静态给出的。2) 预先给定一个基准概率 P,这个基准概率应该是组合服务提供商所能承受的概率,即他能接受在实际概率 p 小于该基准概率 P 时,利用乐观并发控制调度所造成的补偿代价。

每当实例化一个组合服务事务时,初始化运行上下文 ctx。之后每当其他组合服务事务实例开始时,更新 ctx,直至该实例结束前,可以得出完整的 ctx。结合 WCM 和一个服务事务实例的初始 ctx,由公式 $p = \lambda \cdot \frac{c}{C}$ 可以计算出当前服务事务的实例化可能导致的冲突概率 p。然后比较 p 和 P 的大小,决定采用乐观的还是悲观的并发控制。基于冲突概率的并发调度算法详细描述如下:

1) 实例化组合服务事务得到其实例 ct;

2) 初始化 ct 的运行上下文 ctx;

3) 计算 p 值,并比较 p 和 P 的大小;

4) 若 $p < P$,采取乐观并发控制:

(1) 依次调度执行 ct 中所有活动子事务实例;

(2) 当 ct 执行完提交时;

a) 找出在 ct 开始之后才提交完成的服务事务实例集 δ;

b) 检测初始上下文 ctx 中的冲突关系,是否存在冲突对 $\langle at, at_i \rangle \in conf$,且 at 来自于 ct,at_i 来自于 ct_i,$ct_i \in \delta$;

(a) 若存在则放弃 ct,并补偿已提交的活动子事务,同时重新启动 ct。

(b) 若不存在则提交完成。

5) 若 $p \geqslant P$,采取悲观并发控制:

(1) 依次对于 ct 中每一个活动子事务实例 at,在调度运行前;

(2) 找出其所属的所有冲突类集 ω;

(3) 若 ω 为空,则调度 at 执行;

(4) 若 ω 不为空,则对于 ω 中每一个冲突类 $confC$,检测其是否已被加锁如果没有被加锁,则对该冲突类加锁,$cl(confC)$;

(5) 若没有得到 ω 所有的冲突类锁,则:

a) at 阻塞,并释放所有已经获取的冲突类锁;

b) 等待其他活动子事务释放锁的通知,若收到通知,则跳转到(4) 重新执行。

(6) 若 at 获取到了 ω 中所有冲突类的锁,则调度执行完,并释放锁,同时通知冲突类中正在阻塞的其他活动子事务实例;

(7) 当所有活动子事务实例执行完时,组合服务事务实例提交完成。

关于并发控制方法转换的问题,悲观并发控制和乐观并发控制的相互转换可能会导致因冲突没检测到而出错,尤其是在组合服务运行时间相当长时。为了防止错误,利用乐观并发控制提交时才进行冲突检测的特点,给出下面的约定:1) 从乐观到悲观的并发控制转换时,对于已在使用乐观并发控制的服务事务实例继续使用乐观控制方法执行完。2) 从悲观到乐观的并发控制转换时,对于已在使用悲观并发控制的事务实例转换为使用乐观并发控制的方法来调度。但一般不会出现这两种方法频繁转换的情况。

3 相关工作及讨论

传统的短事务并发控制常采用两段提交协议,然而这对长事务并不适用。一些高级事务模型中提出一种 check out/check in 的并发控制机制[3]。这种机制需要所有的事务都具有同样的事务行为。文献[5]中提出一种名为 Jenova 并发控制机制,该机制建立于嵌套事务和允许控制的基础上。其核心是允许控制 admission control,即一个服务在调度执行前,先检查资源是否足够,只有在资源足够的情况下,才能调度执行,同时更新资源。对于长服务,该机制采取资源锁的策略。但是该文是针对具体资源的,如订房间的服务,检查的是空房的间数,而对于抽象的数据,并不能很好地控制。文献[6]则结合了当前的 Web 服务事务协议和规范,基于服务提供方提供的依赖关系图,利用事务依赖管理器,对 Web 服务事务进行并发控制。文中扩展了 WS-Transaction,使其具备并发控制的功能。但是在依赖关系比较复杂的情况,尤其是碰到运行时间长的事务时,阻塞的事务可能将会是无限期的等待,处理也不够灵活,而且通过协调器来传递并发信息,反而会增加并发控制的复杂性。

本文中提出的并发控制算法可以很好应用于各种复杂的服务环境中,不管是服务环境中只有同种模式的组合服务事务实例,还是不同种模式的服务事务实例并存。同时它具有一定的灵活性,服务提供方可以根据情况更改基准概率。本算法充分地发挥了悲观并发控制和乐观并发控制的优点,以提高事务的并发度和正确度。

4 结语

本文首先提出了一个冲突概率模型,并在此基础上提出一个基于冲突概率进行并发控制的算法。服务提供方可以根据其补偿代价的承受能力以及对服务效率的需求选择合适的基准概率。此算法充分结合了乐观并发控制与悲观并发控制的优点,当冲突概率较小时,以提高事务的并发度、提高服务的效率以及系统的吞吐率为主,采取乐观的并发控制机制;在冲突概率较大时,以保证服务事务的正确性为主,采取悲观的锁调度机制。

参考文献:

[1] BEA, IBM, MICROSOFT. Web services atomic transaction (WS-AtomicTransaction). specificationv1.0[EB/OL]. [2005-08-15]. http://www.ibm.com/developerworks/webservices/library/specification/ws-tx/?S_TACT=105AGX04&S_CMP=LP.

[2] BEA, IBM, MICROSOFT. Web services business activity framework (WS-BusinessActivity). specificationv1.0[EB/OL]. [2005-08-15]. http://www.ibm.com/developerworks/webservices/library/specification/ws-tx/?S_TACT=105AGX04&S_CMP=LP.

[3] KIM W, LORIE R, MCNABB D, *et al*. A transaction mechanism for engineering design databases[C]// Proceedings of the 10th International Conference on Very Large Databases. San Francisco: Morgan Kaufmann Publishers, 1984: 355-362.

[4] WEIKUM G, VOSSEN G. 事务信息系统并发控制与恢复的理论、算法与实践[M]. 陈立军, 邱海艳, 赵加奎, 等译. 北京: 机械工业出版社, 2006: 71-98.

[5] GUAN H Q. Jenova: new approach on concurrency control in web service transaction management[C]// Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06). Washington DC: IEEE Computer Society, 2006: 134-142.

[6] ALRIFAI M, DOLOG P. Transactions concurrency control in web service environment[C]// Proceedings of the European Conference on Web Services (ECOWS'06). Washington DC: IEEE Computer Society, 2006: 109-118.