Web 事务的弱原子性与正确性标准*)

郭玉彬[1] 汤德佑[1] 奚建清[1] 李西明[2]

(华南理工大学计算机科学与工程学院 广州 510640)[1] (华南农业大学现代教育中心 广州 510640)[2]

摘 要 事务处理是现代应用系统的基础支撑部件,也是Web服务能否成功支持商务应用的关键技术。但由于环境的高度分布与自治性,传统的可串行化理论已不能再作为Web事务的正确性标准。本文放松了事务的原子性规定,将事务看作由若干原子单元组成的偏序集,并将可串行性扩展为事务的弱可串行性,作为事务的正确性标准。文章最后给出了一个完全分布式的事务并发控制算法,用以实现弱串行性的判定。

关键词 Web服务,事务,弱原子性,弱可串行化,事务调度算法

Weak Atomicity and Correctness Criterion of Web Transaction

GUO Yu-Bin[1] TANG De-You[1] XI Jian-Qing[1] LI Xi-Ming[2]

(College of Computer Science and Technology, South China University of Technology, Guangzhou 510640)[1]

(Modern Education Center, South China University of Agriculture, Guangzhou 510640)[2]

Abstract Transaction is the infrastructure for concurrency and recovery. And it is one of the key technologies for using Web service in e-business. But Serializability theory is not proper to be correctness criterion of service transactions for they work in much more distributed and autonomic environment. In this paper, the atomicity of transaction is loosed in that a transaction is divided into a partial set of atomic units. Accordingly the traditional correct criterion, serializability theory is improved into weakly serializability. A fully distributed service scheduling algorithm is presented at last to realize this correct criterion.

Keywords Web service, Transaction, Weak atomicity, Weak serializability, Scheduling algorithm

1 引言

目前Web服务技术发展迅速,其在异构环境下松耦合的消息传递与互操作已形成工业标准,如SOAP,WSDL,UDDI等。但是由于它们缺乏对事务的有效支持,而无法实现存在事务约束的复杂服务。因此,Web服务的事务机制、长事务、交互事务的结构与调度算法已成为目前研究的热点。

在事务基础理论研究方面,最早的事务都具有ACID特性[1],即原子性、一致性、隔离性和持久性。随着分布式应用的发展,出现了许多高级事务模型[1,2]。其中嵌套事务模型[3]用于对嵌套调用的事务进行建模。Saga模型[4]则是为解决长事务问题设计的。它将事务看作子事务的序列,并为每个子事务提供补偿事务。它允许子事务提前提交,当事务要回滚时需运行其已提交子事务的补偿事务。近年来,G Alonso[5,6]等人对组合环境下的事务进行了研究。所谓组合环境,即由若干个组件构成的系统,其中每个组件都有自己的事务管理机制。目前大多数应用环境,如Web,P2P,网格等都是组合环境的特例。文[5,6]中定义了事务之间的弱顺序关系(weak transaction order),它允许有序事务的操作并发执行,从而提高了事务之间和事务内部的并发度。

在Web服务的事务处理方面,文[13]总结了Web服务环境下事务的主要特点。文[12]在满足传统事务ACID性质的基础之上引入协调器(coordinator)的概念,提出了适用于Web服务的协同工作框架及协议。此外,文[15]针对服务组合,基于CORBA和J2EE对象通信模型,提出了对象事务(object transaction)的解决方案。而WS-Coordination,WS-Transaction[7,8]和BPEL4WS[9]是由IBM,Microsoft和BEA公司提出的Web服务事务标准。

WS-Transaction定义了原子事务和商务活动两类事务。原子事务是具有ACID特性的事务,商务活动则是对长事务的建模。它允许事务嵌套,是Saga模型、嵌套事务模型的具体应用。BPEL4WS[9]集中考虑活动(activity)激活,活动之间数据流及对错误替代执行等问题。它也采用了Saga中补偿事务的思想来处理事务恢复问题。WS-Coordination则描述了可以容纳多种协调协议的Web服务事务处理框架,定义了协调者的组成元素和协调协议。另外,OASIS的Business Transaction Protocol(BTP)[10,11]也是一种较流行的Web事务处理机制。它定义一组在协调者与参加者之间传递的消息。

对Web服务的事务管理,目前并没有采用更好的事务模型,一般仍然以冲突可串行化作为调度正确性的标准。本文将事务看作由若干个原子单元组成的偏序集,原子单元是一组不允许被中断的操作,并允许事务在原子单元之间与其它事务交叉存取,将可串行性扩展为事务的弱可串行性,作为事务的正确性标准。文章最后给出一个分布式并发控制算法,实现弱可串行性的判定。

本文第2部分给出事务模型的基本定义及正确性定理。第3部分给出一种完全分布的服务调度算法。最后总结全文。

2 服务事务模型及正确性标准

Web服务环境是组合环境[6]的一个实例。系统中每个结点提供一种或多种服务,每个结点有自己的事务管理机制。一个应用系统是按照调用关系临时生成的,所以系统体系结构更加松散。事务管理的主要工作是协同服务调用的运行。另外由于系统对结点没有太多约束力,单个结点失效的可能

*)本文受到广东省科技攻关计划项目(004A10205003)、广东省自然科学基金项目(B6480598)、国家教育部项目(D4109029)和广州市科委重点科技攻关项目(G03B2060940)支持。**郭玉彬** 博士生,讲师;**汤德佑** 博士生,讲师;**奚建清** 博士,教授,博士生导师,主要研究领域为数据库与网络计算;**李西明** 博士生,讲师。

性大，相应的单个子事务的可靠性低，但事务协调机制可在不同结点启动功能相同的子事务。因此系统可靠性较传统分布式系统又有所提高。针对这样的特点，本节给出单层的服务事务模型及正确性标准。

2.1 基本概念和术语

事务是服务的一次执行，是被调用操作的偏序集。在此，我们采用传统方式定义事务。

定义 1(事务 Transaction) 事务 $t=\langle O_t, <_t\rangle$是一个二元组。其中 O_t 是事务 t 的操作集合，$<_t$ 是 O_t 上的偏序关系。

事务原子性是指将事务的所有操作定义为一个整体，要么全做，要么全不做。这一点在数据库操作中是必需的，对应用系统，这样的原子性过于严格。例如一个应用要求采购 A,B,C 三种商品，若系统执行时只买到 A,B 两种商品。应用一般不会要求撤销对 A,B 的购买。因此，我们依据应用语义，将事务中那些需要保证"要么全做，要么全不做"的一组操作划分为一个原子单元。事务可看作这样一些原子单元的偏序集。

定义 2(原子单元 Atomic Unit) 设 $t=\langle O_t, <_t\rangle$是一个事务，原子单元是一个二元组 $au=\langle O_{au}, <_{au}\rangle$。其中事务 $O_{au}\subseteq O_t$ 且 $<_{au}=\{(o_i, o_j)\mid (o_i<_{au} o_j)\wedge (o_i, o_j\in O_{au})\}$。

定义 2 中原子单元的划分是依据应用语义进行的。比如，购买某产品的付款和发货操作应在同一原子单元中。而购买 A 商品和 B 商品的操作则一般不会划分在同一原子单元中。

记事务 t 中所有原子单元的集合为 AU_t，将事务 t 的每一个原子单元看成一个元素，则事务 t 可简化为 AU_t 中所有原子单元的偏序集 $t=\langle AU_t, <'_t\rangle$，其中 $<'_t=<_t-\bigcup_{au\in AU_t}<_{au}$。可见，如果事务只有一个原子单元，那么它是具有 ACID 特性的传统事务，是我们所给出事务模型的一个特例。依据包含原子单元的个数，事务可划分为原子事务和弱原子事务。

定义 3(原子事务和弱原子事务) 设 $t_s=\langle O_t, <_t\rangle$是一个事务，若 $|AU_t|=1$，即只有一个原子单元，则称事务 t 为原子事务。否则，事务 t 为弱原子事务。

在事务模型中操作之间的冲突一直是非常重要的概念。在此我们采用普遍接受的冲突定义。如果对所有可能的操作序列 α，β，$\alpha o_i o_j\beta$ 和 $\alpha o_j o_i\beta$ 的返回参数是一致的，则两个操作 o_i，o_j 是可交换的(commutative)或者二者可交换(commute)。若两个操作不是可交换的，则称两操作 o_i，o_j 处于冲突关系，记作 $(o_i, o_j)\in CON$。若操作 o_i，o_j 处于冲突关系，且 $o_i\in t_i$，$o_j\in t_j$，称事务 t_i，t_j 处于冲突关系，记作 $(t_i, t_j)\in CON$。

如果两个操作冲突，则在调度中要依据某顺序执行。而如果两个操作不冲突，执行时没有顺序要求。依此，给出操作之间和事务之间关系定义。

定义 4(操作之间的关系：并发||，顺序→) 两个操作 o_i，o_j，定义 $o_i\rightarrow o_j$ 为操作 o_i 先于操作 o_j 执行。称 o_i，o_j 处于顺序关系。

定义 $o_i||o_j$ 为操作 o_i 和操作 o_j 可以任意顺序执行，称 o_i，o_j 处于并发关系。

定义 4 中，当两个操作并发时，任意顺序执行，是指其执行顺序可以是 $o_i\rightarrow o_j$ 或 $o_j\rightarrow o_i$，或 o_i，o_j 执行在时间上重叠。以任何一种顺序对其进行执行都是正确的。

性质 1 操作之间的顺序关系→是非自反、非对称、传递的。操作之间的并发关系|| 是非自反、对称、非传递的。

由于顺序关系是一种偏序关系，性质 1 很容易证明。记 $o_i\overset{*}{\rightarrow}o_j$ 为其顺序关系的传递闭包，若 $o_i\overset{*}{\rightarrow}o_j$，同样称 o_i，o_j 处于顺序关系。

对事务之间的关系，我们给出类似的定义：

定义 5(事务之间的关系：并发||，顺序→，可串行化↦)

设事务 t_1，$t_2\in\xi$，定义 $t_1\rightarrow t_2$ 为事务 t_1 先于事务 t_2 执行。称事务 t_1，t_2 处于顺序关系。

定义 $t_1||t_2$ 为事务 t_1 与事务 t_2 可以任意顺序执行，称事务 t_1，t_2 处于并发关系。

定义 $t_1\rightarrow t_2$ 为 $t_1||t_2$ 且其执行结果等价于 $t_1\rightarrow t_2$，称事务 t_1，t_2 处于可串行化关系。

由定义 4、5 可知，设事务 $t_1=\langle O_{t1}, <_{t1}\rangle$，$t_2=\langle O_{t2}, <_{t2}\rangle$，若 t_1，t_2 处于顺序关系 $t_1\rightarrow t_2$，则 $\forall o_i\in O_{t1}\ \forall o_j\in O_{t2}$：$o_i\rightarrow o_j$。若 t_1，t_2 处于并发关系 $t_1||t_2$，则对其中冲突操作，需要按某顺序执行，即 $\forall o_i\in O_{t1}\ \forall o_j\in O_{t2}$：$(o_i, o_j)\in CON\Rightarrow(o_i\rightarrow o_j)\vee(o_j\rightarrow o_i)$。对其中没有冲突的操作，若二者之间没有隐含的顺序关系，即 $o_1\in O_{t1}$，$o_2\in O_{t2}$，$(o_1, o_2)\notin CON$，且不存在 $o_i\in O_{t_1}\cup O_{t_2}$，满足 $o_1\rightarrow o_i\rightarrow o_2$ 或 $o_2\rightarrow o_i\rightarrow o_1$，则 $o_1||o_2$。若 t_1，t_2 处于可串行化关系 $t_1\mapsto t_2$，则其运行结果等价于 $t_1\rightarrow t_2$ 的运行结果。所以，对其中所有冲突操作，按相同顺序执行，即 $(o_{1i}, o_{2j})\in CON\wedge(o_{1i}\rightarrow o_{2j})\in CON\wedge(O_{1i}\rightarrow O_{2j})\Rightarrow\forall(o_{1k}, o_{2l})\in CON$：$(o_{1k}\rightarrow o_{2l})$ 其中 o_{1i}，$o_{1k}\in O_{t1}$，o_{2l}，$o_{2j}\in O_{t2}$。

例 1 事务 $T_1=\langle O_1, <_1\rangle$，包含七个操作 $O_1=\{o_{11}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}, o_{17}\}$，$<_1$ 如图 1。设原子单元为 $[o_{11}, o_{12}]$，$[o_{13}, o_{14}]$，$[o_{15}, o_{16}]$，$[o_{17}]$。操作 o_{11}，o_{12} 处于顺序关系，所有正确调度中都需先执行 o_{11}，完成后再执行 o_{12}。操作 o_{13}，o_{16} 处于并发关系，可以任意序执行。

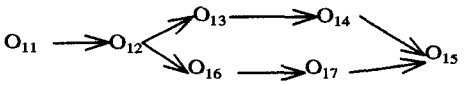

图 1 事务 T_1

2.2 事务调度的正确性标准及判定方法

事务调度的目标首先是满足每个事务内部操作之间的偏序关系，然后依据输入要求，给出并发事务中冲突操作的执行顺序 $\forall(o_1\in O_{t_1}, o_2\in O_{t_2})$：$(o_1, o_2)\in CON\Rightarrow(o_1\rightarrow o_2)\cup(o_2\rightarrow o_1)$ 并保证每个事务都能得到正确执行结果。

定义 6 调度是一个四元组 $s=(T_s, \ll_s, <_s, CON_s)$，其中 T_s 是调度中所有事务的集合，事务的所有操作形成集合 $O_s=\bigcup_{t\in T}O_t$。$CON_s\subseteq O_s\times O_s$ 是冲突谓词。$\ll_s$ 是事务输入顺序关系，是 T_s 上的偏序关系；$<_s$ 是事务输出顺序关系，满足：

1) $\forall t\in T, \forall o, o'\in O_t$：$(o<_t o')\Rightarrow(o<_s o')$

2) $\forall t, t'\in T_s, t\neq t', \forall o\in t, \forall o'\in t'$：$(o, o')\in CON_s$，则 $(t\ll_s t')\Rightarrow(o<_s o')$

3) $\forall o, o'\in O_s$：$(o, o')\notin CON_s$，则 $(o<_s o')\vee(o'<_s o)$。

冲突可串行化理论中，冲突等价一直是一个很重要的概念。简单讲，两个调度冲突等价是指它们对相同的冲突操作，给予相同的执行顺序。即，对调度 $s_1=(T_{s_1}, \ll_{s_1}, <_{s_1}, CON_{s_1})$，$s_2=(T_{s_2}, \ll_{s_2}, <_{s_2}, CON_{s_2})$，如果，$T_{s_1}=T_{s_2}$，$\ll_{s_1}=\ll_{s_2}$，$CON_{s_1}=CON_{s_2}$，且对任意两个冲突操作 $(o_i, o_j)\in CON$，$o_i<_{s_1} o_j\Leftrightarrow o_i<_{s_2} o_j$，则两个调度 s_1，s_2 冲突等价。在冲突等价下，我们给出串行调度、弱串行调度及弱可串行化调度的定义。

定义 7(串行调度 Serial Schedule 和弱串行调度 Weakly Serial Schedule) 调度 $s=(T_s, \ll_s, <_s, CON_s)$，$T_s=\{t_1, t_2, \cdots, t_n\}$，如果输出顺序 $<_s$ 满足下式，

[1] 原子单元早分依据应用语义进行，此处仅给出一种示例。另外无特别说明，[]中操作处于同一原子单元中。

$\forall(t_i, t_j \in T_s(i\neq j))$：$((\forall(o_1 \in O_{t_i}, o_2 \in O_{t_j}) \Rightarrow o_1 <_s o_2) \vee (\forall(o_1 \in O_{t_i}, o_2 \in O_{t_j}) \Rightarrow o_2 <_s o_1)$ (1)

则 s 为串行调度

调度 s 如果和一个串行调度冲突等价，则称 s 是可串行化调度。

若 s 满足：$\forall t_i, t_j \in T_s(i\neq j), o_{ik} \in O_{t_i}, o_{jl} \in O_{t_j}$，且 $o_{ik} \in au_{im}, o_{jl} \in au_{jm}$ 其中 $au_{im} \in AU_{t_i}, au_{jm} \in AU_{t_j}$。如果输出顺序 $<_s$ 满足下式，则 s 是一个弱串行调度。

$(o_{ik} <_s o_{jl}) \Rightarrow \forall(o'_{ik} \in au_{il}, o'_{jl} \in au_{jm}) : (o'_{ik} <_s o'_{jl})$ (2)

调度 s 如果和一个弱串行调度冲突等价，则称 s 是弱可串行化调度。

可串行性一直是事务调度正确性的标准。一般认为一个并发调度只有冲突等价于某个串行调度，即冲突可串行化时，它才是正确调度。弱串行调度以原子单元为单位，强调不破坏原子单元的原子性。事实上，若一个调度的所有事务都只包含一个原子单元，则弱串行调度与串行调度等价。相应的弱可串行化调度与串行化调度等价。因此，串行调度、可串行化调度都是弱可串行化调度的特例。

例 2 设事务 $t_1=[o_{11}, o_{12}][o_{13}, o_{14}, o_{15}], t_2=[o_{21}, o_{22}, o_{23}]$，其中 $CON=\{(o_{11}, o_{21}), (o_{15}, o_{23})\}$。调度 $s_1 = o_{11}, o_{12}, o_{13}, o_{14}, o_{15}, o_{21}, o_{22}, o_{23}$。$s_2 = o_{11}, o_{12}, o_{21}, o_{22}, o_{23}, o_{13}, o_{14}, o_{15}$。$s_3 = o_{11}, o_{21}, o_{12}, o_{22}, o_{13}, o_{14}, o_{23}, o_{15}$

可见，s_1 是串行调度，s_2 不破坏每个原子单元的原子性，是弱串行调度。对调度 s_3 进行冲突等价交换，可知 s_3 冲突等价于 s_2，所以它是弱可串行化调度。

在串行化理论中，对调度的正确性判定可通过串行化图进行。同样，对弱可串行性的判定，可构造弱可串行化图。为描述方便，先给出偏序集上线形链及线形链上第一元素、最后一个元素的定义。设 A 是偏序集，$X\subseteq A$。如果任给 $x, y\in X$，都有 $x\leqslant y$ 或 $y\leqslant x$，即 $\leqslant | X$ 是全序，则称 X 是 A 的线形链。在线形链 X 中，若 $\forall y\in X: x\leqslant y$，则称 x 为 X 中的首元。若 $\forall y\in X: y\leqslant x$，则称 x 为 X 中的尾元。

定义 8(首元集和尾元集) 设事务 $t=\langle O_t, <_t\rangle, au_k \in AU_t$，对于操作 $o_j \in au_k$，$H(o_j, t)$ 是 au_{ik} 中所有包含 o_j 的线形链上首元的集合，$T(o_j, t)$ 是 au_{ik} 中所有包含 o_j 的线形链上尾元的集合。

定义 9(弱串行化图，Weakly Serialization Graph) 设 $s=(T_s, \ll_s, <_s, CON_s)$ 是一个调度，其弱可串行化图 $WSG(s)=(V,E)$ 是一个二元组，其中 $V=O_s$ 是 s 中所有操作的集合，$E\subseteq V\times V$，E 中包含以下四类边

1)事务内部的偏序关系边，称为内边(I-arcs)。即，

$\forall t_i \in T_s, o_j, o_k \in O_{t_i} \langle o_i, o_j\rangle \in <_{t_i} \Rightarrow \langle o_i, o_j\rangle \in E$

2)冲突操作边，称冲突边(C-arcs)。即，

$\forall(o_i \in O_{ik})(o_j \in O_{il}): ((o_i, o_j) \in CON) \wedge (o_i <_s o_j) \Rightarrow \langle o_i, o_j\rangle \in E$

3)首元边(*H-arcs*)，即，对任一冲突边

$\langle o_i, o_j\rangle \in E \Rightarrow \forall x \in H(o_i, t_k): \langle o_i, x\rangle \in E$

4)尾元边(*T-arcs*)，即，对任意冲突边

$\langle o_i, o_j\rangle \in E \Rightarrow \forall x \in T(o_j, t_l): \langle x, o_j\rangle \in E$

如例 3 中，图 2 是调度 s_2 的弱可串行化图。

引理 1 如果两个调度是冲突等价的，则它们的弱串行化图是相等的。

证明：设 $s_1=(T_{s_1}, \ll_{s_1}, <_{s_1}, CON_{s_1}), s_2=(T_{s_2}, \ll_{s_2}, <_{s_2}, CON_{s_2}), T_{s_1}=T_{s_2}, \ll_{s_1}=\ll_{s_2}, CON_{s_1}=CON_{s_2}$，若 s_1, s_2 冲突等价，则 $\forall(o_i, o_j) \in CON_{s_1}: o_i <_{s_1} o_j \Leftrightarrow o_i <_{s_2} o_j$。

1)由 $T_{s_1}=T_{s_2}$ 知 $WSG(s_1)$ 与 $WSG(s_2)$ 中结点集相等且内边都相同，且所包含事务的原子单元划分也是相同的。

2)再由 $CON_{s_1}=CON_{s_2}$，$\forall(o_i, o_j) \in CON: o_i <_{s_1} o_j \Leftrightarrow o_i <_{s_2} o_j$ 知，$WSG(s_1)$ 与 $WSG(s_2)$ 中冲突边也完全相同。

3)而 H-arcs 和 T-arcs 都是由 C-arcs 和原子单元得到的，所以 $WSG(s_1)$ 与 $WSG(s_2)$ 中 H-arcs、T-arcs 对应相等。

由以上三点知，$WSG(s_1)=WSG(s_2)$ □

定理 1 一个调度 s 是弱可串行化的，当且仅当 $WSG(s)$ 是无环的。

证明：设 $s=(T_s, \ll_s, <_s, CON_s)$ 是一个弱可串行化调度，其弱可串行化图为 $WSG(s)$。

先证必要性。若 s 是弱可串行化调度，由引理 1，必存在一个与之冲突等价的弱串行调度 s' 满足 $WSG(s)=WSG(s')$。只要证明 $WSG(s')$ 中无回路即可。

我们首先说明在 $WSG(s')$ 中 $(o_i, o_j) \in E \Leftrightarrow o_i <_s o_j$。

这个结论对 I-arcs 和 C-arcs 显然成立。对 H-arcs 的情况：设 $o_i \in O_{t_i}, x \in O_{t_j}$ $(o_i, x) \in E$ 是一条 H-arcs，则必存在 $o_j \in O_{t_j}$，(o_i, o_j) 是一条 C-边，o_j，x 属于同一原子单元 au_{ik}，x 是 au_{ik} 中 o_j 所在的某一线形链上的首元。可知 $x <_{t_j} o_j$，所以 $x <_s o_j$ 成立。再由调度 s' 的弱可串行性知，在 s 中操作 o_i 出现在原子单元 au_{ik} 所有操作之前，即 $o_i <_s x$ 成立。

对 T-arcs，结论同样成立。因此，$WSG(s')$ 中无回路。所以，$WSG(s)$ 中也没有回路。

再证充分性：设 $WSG(s)$ 中无回路，取 s' 为对 $WSG(s)$ 中结点进行拓扑排序得到的调度，则 s，s' 冲突等价，由此可知 $WSG(s)=WSG(s')$。

设调度 s' 不是弱可串行化调度，则存在 $t_i, t_j \in T_{s'}, o_i, o_k \in O_{t_i}, o_j \in O_{t_j}$ 满足 $o_i <_s o_j <_s o_k$ 且 o_i, o_k 属于同一原子单元 au_{ik}，即 o_j 与 au_{ik} 交叉存取。由此可知，存在操作 $x \in O_{t_i}: (x, o_j) \in CON_{s'}$。对此分 $x <_s o_j, o_j <_s x$ 两种情况讨论。

1)由 $x <_s o_j$ 可知，存在 T-arcs $\langle y, o_j\rangle \in E$ 其中 $y \in T(x, t_i)$。由原子单元定义知，存在尾元 y' 与操作 o_k 在同一条线上且 $o_k <_s y'$。所以 $o_k <_s y' <_s o_j <_s o_k$，与 $WSG(s)$ 中无回路矛盾。

2)由 $o_j <_s x$ 可知存在 H-arcs $\langle o_j, z\rangle \in E$ 其中 $z \in H(x, t_i)$。由原子单元定义知，存在首元 z' 与操作 o_i 在同一条线上且 $o_i <_s y'$。所以 $o_i <_s y' <_s o_j <_s o_i$，与 $WSG(s)$ 中无回路矛盾。

综上所述，s 是弱可串行化调度 □

例 3 给定事务 $t_1=[o_{11}][o_{12}], t_2=[o_{21}\ o_{22}], t_3=[o_{31}][o_{32}], CON=\{(o_{11}\ o_{21}), (o_{22}\ o_{32}), (o_{12}\ o_{31})\}$。

调度 $s_1=(\{t_1, t_2, t_3\}, \varnothing, <_{s_1}, CON)$，$<_{s_1} = o_{11}\ o_{21}\ o_{22}\ o_{31}\ o_{32}\ o_{12}$。调度 $s_2=(\{t_1, t_2, t_3\}, \varnothing, <_{s_2}, CON)$，$<_{s_2} = o_{11}\ o_{21}\ o_{31}\ o_{22}\ o_{32}\ o_{12}$。

其中，s_1 是弱串行调度。图 2 是调度 s_2 的弱可串行化图，没有回路，所以 s_2 是弱可串行化调度。另外，对 s_2 进行冲突等价变换可知，它与 s_1 冲突等价。

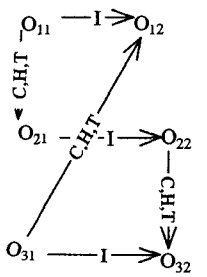

图 2 调度 s_2 的弱可串行化图

3 事务的并发控制算法

当前 Web 环境中多个服务同时执行是更加普遍的现象，此时每个服务只了解自己的组成服务之间关系等局部信息，对其它服务的情况毫不知情。因此，本文给出一个基于弱可串行化图的并发控制算法。算法分两部分，一个是结点上进行冲突检测的算法，并将检测到的冲突信息传递给服务调用者。另一个是事务正确性判定算法，它启动一个事务，然后接收从事务执行结点发来的冲突消息，构造弱可串行化图。若图中有回路，则认为调度错误，申请撤销其中已调用事务的某个单元。若图中没有回路但图中有变化，则将变化相关部分与自己的原子单元序列一起发送给与之冲突的前序操作所在事务。算法的正确性由以上定理 1 保证。

算法 1：local transaction scheduler

```
输入：消息(CALLTRANSACTION, WITHDRAW, COMMIT 三种类型)
输出：冲突消息或结果
步骤：
{//等待消息
    Waiting for message
    Switch type(message)
      Case CALLTRANSACTION { //申请调用服务消息
      Start according transaction si;
      Calculate conflict with si;//本地冲突计算
      If have some
      send conflict message to the caller;
    }
      Case WITHDRAW { //事务撤销的消息
      doing withdraw;
      collect conflict transactions with si;//撤销冲突关系
      if have some
        send message(type dependence) to the caller;
    }
    Case COMMIT { //事务提交的消息
        collect conflict transactions with si;//撤销冲突关系;
        send message(type dependence) to each conflicting caller;
        send result(success or failed) to the caller;
        }
} //算法结束
```

算法 2：global transaction scheduler

```
{
while true do {
    execute local operations
    if service call {//事务调用
        call transaction at proper site; //向站点发调用命令
        waiting for messages
        }
    receive message(from subtransaction and other global transactions) //接收消息
    switch type(message)
    caser RESULT (from subtransaction) //执行结果
        continue executing
    //继续执行其它操作，若事务完成，跳出循环
    case CONFLICT(from subtransaction or other global transaction) { //冲突关系
        UpdateWDG (message); //更新序列化图
        if HasCycle(WDG) {//出现回路
          calculate withdrawer;
          if withdrawer=itself
          //若撤销自己，执行撤销操作
          exit
          }
      else
          if Updated(DG) send message(type dependence) to preordered transactions;
      }
    case dependence (from subtransactions or other global transactions) { //撤销依赖关系
        UpdateWDG (message);
    }//end while
if finished
    return(result); //返回执行结果给调用者
    send message(dependence) to preorder;
} //算法结束
Void UpdateWDG (message)
{//依据消息更新弱可串行化图 }
Boolean Updated (WDG)
{//判定 WDG 是否被上次 update 操作改变，若有改变返回 true，否则返回 false}
Boolean HasCycle (DG)
{//判定 WDG 中是否有回路，若有返回 true，否则返回 false}
```

结论 将事务划分成若干个片段，允许事务在不同片段之间交互。这一思想很早之前在 Saga、相关可串行化理论中就已经出现过。尤其在文[9]中，Agrawal 等人利用事务的语义信息，向不同事务提供不同的原子性视图，并允许事务在不违背相关原子性的条件下进行交叉存取。文章还给出了基于相关可串行化图的并发控制算法。但其研究工作是在数据库系统中进行的。而当前环境中，事务只有与其它事务发生冲突时才可知道另一事务的存在，对不同事务维持不同原子视图是不可能的。同时，全局控制中心的概念已弱化，集中式的相关串行性判定也很难实现。因此，本文定义每一个事务只有一种原子单元划分，并给出事务弱串行调度，弱可串行化调度和定义及判定条件。弱可串行化是相关串行化的扩展，它更适于目前松耦合的分布式环境，也更容易推广到树型事务。另外，笔者对冲突边重新做了定义，使弱串行化图生成与判定比相关串行化图的生成与判定效率有所提高。设两事务原子单元平均包含操作个数分别是 m,n，共有 p 对直接冲突操作。相关可串行化图在两事务间平均增加 $O(p(m+n))$ 条边，判定回路的效率也是 $O(p(m+n))$ 级。而弱可串行化图在两事务间增加 $3p$ 条边，判定回路效率为 $O(p)$。

本文给出了服务的弱原子性及弱可串行性定义。弱原子性是对传统事务原子性的扩展，它更适合 Web 服务的环境。弱可串行性是服务事务的正确性标准。文章最后还给出了一个分布式并发控制算法。嵌套调用事务的弱原子性及并发控制方法我们在另一篇文章中已给出。笔者下一步的工作是研究高效的并发控制算法并建立应用原型系统。

参 考 文 献

1 Gray J. 事务处理：概念与技术. 孟小峰译. 电子工业出版社，2004
2 Weikum G, Vossen G. Transactional Information Systems Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Elsevier Science USA ISBN, 1-55860-508-8, 2002
3 Garcia-Molina H, Salem K Sagas. In: Proceeding of ACM International Conference on management of Data (SIGMOD), 249～259
4 Moss J E B. Nested transactions: An introduction. In : BBhargava, ed. Concurrency Control and Reliability in Distributed Systems. New York : Van Nostrand Reinhold , 1987. 395～425
5 Alonso G, Blott S, Fessler A, et al. Correctness and Parallelism in Composite Systems. In: Proceedings of the 16th ACM Symposium on Principles of Database Systems. (PODS'97). Tucson, Arizona, USA: ACM Press, 197～208
6 Alonso G, Fessler A, Pardon G, et al. Correctness in General Configurations of Transactional Components. In: Proceedings of the 18th ACM Symposium on Principles of Database Systems. (PODS'99). Philadelphia, Pennsylvania, USA: ACM Press, 289～293
7 Orchard D. Web Services Coordination[EB]. Http://dev. bea. com/technologies/webservices/standards. jsp. 2002-08
8 Cox W. Web Services Transaction[EB]. Http://dev. bea. com/technologies/webservices/standards. jsp. 2002-08
9 BPEL4WS v1. 1 [EB /OL]. http://www. ibm. com /developerworks/library/ws2bpel/, 2005
10 OASIS Business Transaction Protocol, Committee Specification 1. 0. 2002. http://www. oasis-open. org/business-transaction/
11 Furniss P. Business Transaction Protocol Version1 1. 0 [EB]. Http://www. oasis-open. org/committees/download. php/4343/WS2CAF%2Primer. pdf , 04-11-24
12 Cabrera F, Copeland G, Cox B, et al. Web services transaction. http://www. infosys. tuwien. ac. at/Teaching/Courses/IntAppl/Papers/ws-transpec. pdf
13 岳昆，王晓玲，周傲英. Web 服务核心支撑技术：研究综述. 软件学报，2004,15(3):428～442
14 Agrawal D, Bruno J L, El Abbadi A, et al. Relative Serializability: An Approach for Relaxing Atomicity of Transactions
15 Mikalsen T, Rouvellou I, Tai S. Reliability of composed Web services from object transactions to Web transactions. In: Proc. of the OOPSLA 2001 Workshop on Object-Oriented Web Services, 2001