支持 LRT 的失败恢复算法及其事务性质分析

梅晓勇[1]，黄昌勤[1]，郑小林[2]，陈德人[2]，李师贤[1]
（1. 中山大学 信息科学与技术学院，广东 广州 510006；2. 浙江大学 计算机科学与技术学院，浙江 杭州 310027）

摘 要：研究人员已经致力于组合事务的恢复问题研究，但是大多数成果集中通过向后恢复来维持事务的一致性，补偿是向后恢复通常使用的一种手段，但是向后恢复的最大缺陷就是导致代价相当高，且向后恢复策略不能完全满足各种不同恢复需求。提出一种基于失败类型的恢复算法（包括向前、向后和替代恢复），其是一种基于扩展 Petri 网的形式化建模方法，为实现松弛 ACID 属性，引入状态托肯、数据托肯和 QoS 托肯，增加失败变迁和补偿变迁。失败发生时，动态计算终止依赖点 TDP 和补偿集，依据任务之间的控制流、数据流、时序、状态和行为依赖，获取任务的失败类型，选择合适的恢复策略，构造一个支持无缝添加/删除失败恢复的可执行模型。

关键词：组合事务；基于范围恢复；失败恢复算法；松弛 ACID
中图分类号：TP301 **文献标识码**：A **文章编号**：1000-436X(2012)04-0031-11

Failure recovery algorithm for LRT and transaction property analysis

MEI Xiao-yong[1], HUANG Chang-qin[1], ZHENG Xiao-lin[2], CHEN De-ren[2], LI Shi-xian[1]
(1. School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510006, China;
2. College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

Abstract: Researchers have worked on recovery strategies of composition transactions, most efforts focus on transaction consistency by backward recovery, compensation is commonly used for backward recovery, which have limitations since the cost of compensation tasks are rather expensive, and the compensation mechanism can not satisfy various requirements of applications. A transaction recovery algorithm was proposed including forward recovery, backward recovery and alternative recovery, which was a formal modeling method based on extended Petri nets, state token，data token and QoS token were introduced to implement relaxed ACID properties of LRT. When failure occurs, terminate dependency point (TDP) and compensation set were calculated dynamically, recovery strategies were added or deleted automatically according to the data flow dependency, control flow dependency and state dependency among tasks. Failure types are generated by log file mining, therefore, failure coordination and recovery are implemented and an executable model which support add/delete failure recovery behavior is constructed seamlessly.

Key words: composition transaction; scope-based recovery; failure recovery algorithm; relaxed-ACID

收稿日期：2011-08-03；**修回日期**：2011-11-25
基金项目：国家科技支撑计划基金资助项目(2008BAH24B03)；国家自然科学基金资助项目(60673122,60940033)；中国博士后基金资助项目(20080440121)；湖南省自然科学基金资助项目(06017089,10JJ6100,10151063101000046)；湖南省科技计划基金资助项目(2010GK3020)
Foundation Items: The National Key Technologies R&D Program of China (2008BAH24B03); The National Natural Science Foundation of China (60673122, 60940033); The Postdoctoral Science Foundation of China (20080440121); The Natural Science Foundation of Hunan Province (06017089,10JJ6100, 10151063101000046); The Science and Technology Planning Project of Hunan Province (2010GK3020)

1 引言

组合后长运行事务 LRT(long running transactions)运行的长效性(可能持续几小时、几个月甚至更长)，这为 LRT 的事务处理带来难度。传统的 ACID(atomicity, consistency, isolation, durability)机制过于严格，不再适用于 LRT，需要放宽隔离性，松弛资源持有条件，确保可靠 LRT 执行。因此，在组合事务环境下，把 Relaxed-ACID 事务属性应用到 LRT 是非常有必要的。

文献[1]和文献[2]提出了基于容忍嵌套的 LRT 失败执行框架，探讨自动处理容错和失败处理策略。文献[3]提出了组合 Web 事务的失效恢复算法。文献[4]借助自适应执行上下文，提出一种新的基于语义的动态恢复机制。文献[5]提出了一种基于语义的 Web 事务协议。文献[6]提出了一种灵活的确保可靠面向服务事务的方法，确保控制流和事务模式的高内聚。文献[7]提出一种支持业务流程的向后恢复补偿方法，其将补偿逻辑和失败依赖作为协调逻辑的一部分。虽然这种方法解决了回滚问题，但有 2 个主要缺点：第一，多数情况下只能完成半自适应；第二，这些类型的工作流需要非常严格的流程定义。

目前支持事务机制的组合 Web 服务环境只提供有限的补偿能力，引入向后恢复确保事务的一致性。文献[8]通过扩展事务协调架构和基础设施，提出一种基于向前恢复策略的高级补偿环境。文献[9]提出一种基于规则的工程补偿方法。文献[10]提出了一种支持用户指定失败处理的 Web 服务组合框架。文献[11]给出了一个基于 Java 的事务性组合服务 JTWS 的 API 框架。为了保证 LRT 执行语义的准确性，要么正常结束，包括借助替换恢复或向前恢复策略使流程继续，要么通过执行向后恢复策略撤销已经提交的执行结果。面临如下两方面问题：① LRT 执行时可能消耗大量资源，不一定对所有已提交任务逆向补偿，并最终回滚到到初始状态；② 中止整个 LRT 需要很大代价，一种折中的恢复机制是先向后退到一个点，然后用替代任务或修复失败继续正向执行。引入灵活的 LRT 的恢复策略，有如下优势：① 重试失败任务或选择等价替代任务有助于 LRT 重新获得成功；② 比尝试“修补”失败任务更有效；③ 有利于预测长事务执行状态，而不会导致长时间占用资源。综上所述，有必要进一步研究基于 LRT 灵活的事务恢复策略。

2 基于 Petri 网的任务形式化

首先给出了基于 Petri 的原子任务形式化定义。

定义 1 原子任务 I 的三元组表示为(P,T,F)。

1) $P=P^s\cup P^{io}\cup P^{qos}\cup P^c$，其中，$P^s$ 指 I 的状态库所 $\{Ready, Activated, Failed, Running, Aborted, Cancelled, Committed, Compensated, Half\_Compensated\}$；$P^{io}$ 指 I 的 I/O 库所集，通常描述 Web 服务的功能参数；P^{qos} 指描述 I 的非功能参数；P^c 描述 I 的信号库所。对于 I，分别用 $I.p^s, I.p^{io}, I.p^c$ 和 $I.p^{qos}$ 来描述 I 的状态、功能、信号和非功能参数。

2) $T=T^n\cup T^b\cup\tau$，其中，T^n 指 I 正向事件集 $\{Activate(), Run(), Fail(), Abort(), Cancel(), Commit()\}$；$T^b$ 指 I 反向事件集 $\{Compensate(), Hcompensate(), Retry()\}$；$\tau\in T$ 为空操作。为简化描述，正向事件 $Activate(), Run(), Fail(), Abort(), Commit(), Cancel()$ 分别简记为 $t^{act}, t^{run}, t^{fal}, t^{abt}, t^{cmt}, t^{cnl}$；反向事件 $Retry()$ 和 $Compensate()$ 分别简记为 t^{rty} 和 t^{cmp}。对于任务 I，用 $I.t$ 获取 I 的执行事件。

3) $F=(P\times T)\cup(T\times P)$ 分别描述 P 到 T 或 T 到 P 控制流和数据流的有向弧集合。事件/状态对 (t_1,p_1) 和 (t_2,p_2) 间有如下约束：① 偏序关系 $(t_1,p_1)\prec(t_2,p_2)$；② 排斥关系 $(t_1,p_1)\lhd\rhd(t_2,p_2)$；③ $(t_1,p_1)\approx(t_2,p_2)$ 描述动作 t_1 和 t_2 同时发生或都不发生。

每个原子服务都有独特的事务行为，依据功能语义将分为 4 类：枢轴 (pivot) 服务(记为 WS^p)、可补偿 (Compensable) 服务 (记为 WS^c)、可重试 (Retriable) 服务(记为 WS^r)和关键 (Vital) 服务(记为 WS^v)。对于任务 I，其事务类型记为 $I.TBP$，其中，$I.TBP\in\{Pivot, Vital, Retriable, Compensable\}$。

WS^p 指 I^p 既不可重试，也不可补偿，如图 1(a)所示。WS^c 是当 I^c 成功提交后，通过执行其补偿任务来消除产生的影响，如图 1(b)所示。WS^r 指 I^r 执行失败，有限次重试后，确保 I^r 成功提交，如图 1(c)所示。WS^v 指 I^v 具备重试和补偿的能力，如图 1(d)所示。表 1 给出了任务执行状态和变迁含义，运行时任务可能转移到就绪(p_0)、运行(p_1)、提交(p_2)、中止(p_4)、取消(p_6)、失败(p_7)和补偿(p_9)状态之一，触发 $Activate()$、$Run()$、$Fail()$、$Abort()$、$Cancel()$ 或 $Commit()$ 等内部或外部动作。

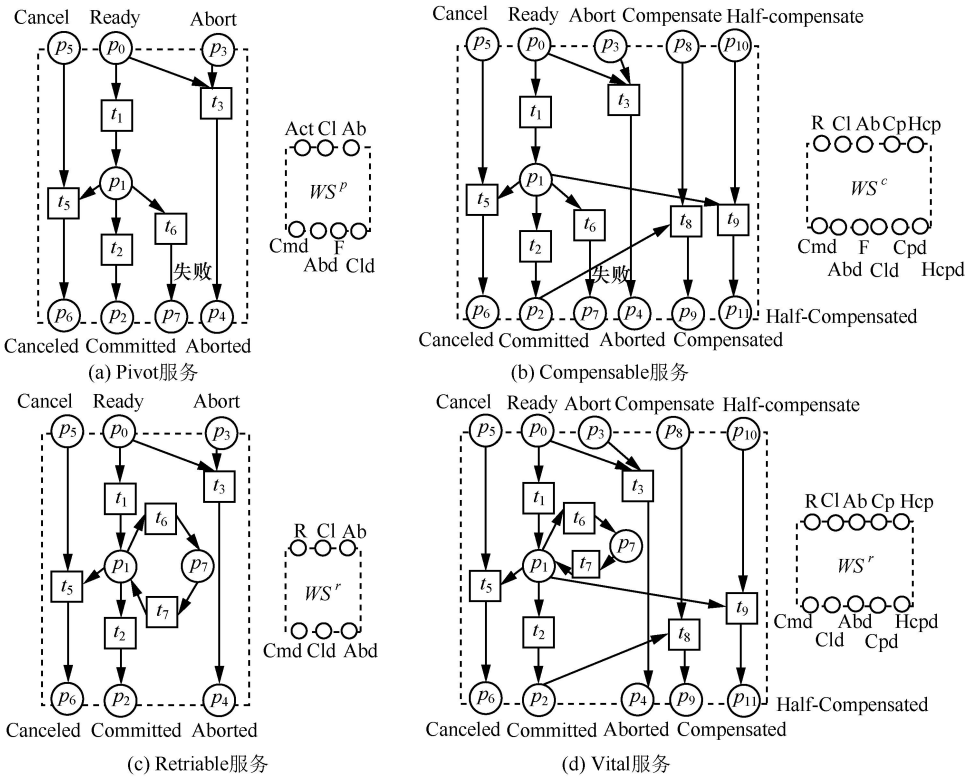


图 1　原子任务形式化

表 1　**事务属性的变迁和库所**

库所	状态	变迁	动作
p_0	就绪	t_1	激活
p_1	运行	t_2	运行
p_2	提交	t_3	中止
p_3	协调中止	t_5	取消
p_4	中止	t_6	失败
p_5	协调取消	t_7	重试
p_6	取消	t_8	补偿
p_7	失败	t_9	部分补偿
p_8	协调补偿		
p_9	补偿		
p_{10}	协调部分补偿		
p_{11}	部分补偿		

3　恢复范围及恢复契约

LRT 流程的可靠性和一致性问题变得尤为突出，有必要提出一种可靠的失败恢复算法。目前，现有规范大多涉及补偿算法，且补偿算法定义在范围(BPEL4WS)或上下文(WSCI)或编排(WS-CDL)中。本文扩展 BPEL4WS 范围，扩展后范围由静态指定改为动态计算，恢复范围相对固定，缺乏灵活性。

3.1　恢复范围

考虑不同恢复需求，为不同恢复策略灵活地定义恢复范围 Ξ。况且流程设计师为失败类型的失败恢复确定 Ξ，也是一件非常困难的事情。本文给出的解决方法是依据恢复规格说明动态计算 Ξ。

对于给定 LRT，执行迹 σ，如果 $\exists\sigma\forall I_i,I_j\in\sigma\wedge I_i.\mathrm{TBP}, I_j.\mathrm{TBP}\in\{\mathrm{Compensable},\mathrm{Vital}\}$，$I_i$ 和 I_j 之间存在 $I_j\gg^{\mathrm{exd}}I_i$、$I_j\gg^{\mathrm{inexd}}_{\mathrm{lk}}I_i$、$I_j\gg^{\mathrm{inexd}}_{\sigma'}I_i$、$I_j\gg^{\mathrm{imd}}I_i$ 和 $I_j\gg^{\mathrm{inimd}}I_i$ 依赖之一，且 I_i 和 I_j 均是可补偿的，则 σ 是可补偿的，记为σ.Compensated。若 LRT 中每个任务都是可补偿的，且与失败任务存在依赖的任务都能确保成功补偿，则该 LRT 是可补偿的。

LRT 执行失败，确定终止依赖点(TDP, terminate dependency point)非常关键，恢复处理器(RH, recovery handler)启动恢复策略(RHS, recovery handling strategy)，对 Ξ 中所有任务执行恢复，直到 TDP。对于给定 σ，如果 $\exists\sigma\forall I_i$，$I_j\in\sigma\wedge I_i.\mathrm{TBP}, I_j.\mathrm{TBP}\in\{\mathrm{Vital},\mathrm{Compensable}\}$，$I_i$ 和 I_j 之间存在数据流依赖 $I_j\gg^{\mathrm{exd}}I_i$、$I_j\gg^{\mathrm{inexd}}_{\mathrm{lk}}I_i$、$I_j\gg^{\mathrm{inexd}}_{\sigma'}I_i$、$I_j\gg^{\mathrm{imd}}I_i$ 或 $I_j\gg^{\mathrm{inimd}}I_i$，称 I_i 是 I_j 的 TDP，如图 2 所示。若 I_j 失败，从依赖点 I_i 开始对$\sigma_{\mathrm{m}}(\sigma_{\mathrm{m}}\in\sigma)$执行恢复，称$\sigma_{\mathrm{m}}$ 为 I_j 的依赖路径。

对于给定 LRT，I_j ($I_j\in\sigma$)失败，计算 I_j 的依赖

路径集$\{\sigma_{m1},\sigma_{m2},\cdots,\sigma_{mk}\}$，回滚到 I_i 并注入修正参数，正向执行σ_m直到 LRT 成功执行。考虑恢复代价，对$\{\sigma_{m1},\sigma_{m2},\cdots,\sigma_{mk}\}$的恢复范围集$\{\Xi_1,\Xi_2,\cdots,\Xi_k\}$，优先选取恢复范围小的 Ξ 执行恢复。

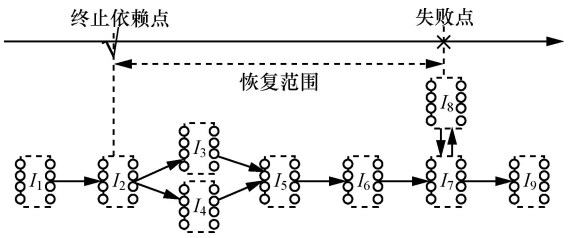


图 2 确定失败任务的恢复范围

范围嵌套类似于聚合模式嵌套，如图 3 所示，$\Xi_{[1]} \vDash \Xi_{[1][2]}$指$\Xi_{[1][2]}$是$\Xi_{[1]}$的子范围，括号中数字表明嵌套层次。$\Xi_{[1]}\vDash\cdots\vDash\Xi_{[1]\cdots[i]\cdots[m]}$表示$\Xi$包含 m 个嵌套层，其中，每个嵌套子范围可能包含零个或多个任务。RH 启动Ξ中预定义的 RHS，用二元组$(Fail_{type}, Action)^i$描述Ξ中失败任务 I 启动第 i 个事务失败处理策略 rhs_i，其中，$Fail_{type}$为不同失败触发类型，Action 为失败处理事件。$Fail_{type}$又可分为简单失败(记为 $Fail(Sname)$)和组合失败(记为 $Fail(Sname_1)\wedge Fail(Sname_2)\wedge\cdots\wedge Fail(Sname_n)$)。Action 描述触发事件定义的失败处理操作，其又可分为原子操作和组合操作，前者指单一替代或补偿操作，后者指聚合算子连接的多个原子行为。

通常，Ξ 与 *RHS* 集$(rhs_1,rhs_2,\cdots,rhs_n)(rhs_i\neq rhs_j,i\neq j)$相关联。例如，TRP 中 Ξ 有 2 个恢复策略 $rhs_1=(Fail(ReserveAirline_Tickets), Compensate(UndoAirline Tickets))$和 $rhs_2=(Fail(CarRenting), Alter(BusRenting))$，其中，当 *ReserveAirline Tickets* 失败，执行补偿，撤销预订机票；而 *CarRenting* 失败，调用替代恢复 *Bus Renting* 执行巴士租赁服务。

3.2 恢复契约

局部依赖伴随于任务间交互，全局依赖往往存在时序和条件约束。LRT 中任务间前后执行时序采用事件约束描述：$\sigma_1\triangleright e\triangleright\sigma_2,\sigma_1\triangleright e$ 和 $e\triangleright\sigma_2$，其中，事件 $e\in\{Activate(),Run(),Fail(),Abort(),Cancel(),Retry(),Compensate,Hcompensate(),Commit()\}$，$\sigma_1\triangleright e\triangleright\sigma_2$ 使得 e 一定发生，记为∇e；而$\sigma_1\triangleright\neg e\triangleright\sigma_2$ 则禁止 e 发生，记为$\neg\nabla e$。扩展后的串行约束$\sigma_1\triangleright(e_1,\cdots,e_n)\triangleright\sigma_2$ 记为 $\nabla(e_1,\cdots,e_n)$，等价于$\nabla e_1\oplus\cdots\oplus\nabla e_n$，表明事件 $e_1,\cdots,e_n$ 均发生。

为了更好地采用 Petri 网描述恢复契约，引入扩展谓词逻辑 CTR，除了$\vee,\wedge,\neg,\forall$和$\exists$等算子外，还包括连接算子($\oplus$和|)和模态算子$\odot$。$\nabla e_1\oplus\nabla e_2$ 指先后执行 e_1 和 e_2；$\nabla e_1\|\nabla e_2$ 指 e_1 和 e_2 以交织方式并发执行；$\nabla e_1\wedge\nabla e_2$ 指 e_1 和 e_2 同时执行；$\nabla e_1\vee\nabla e_2$ 仅 e_1 或 e_2 执行。$\oplus$,$\|$,$\wedge$和$\vee$分别用于 *sequence*, *And-split*, *Or-split* 控制流交互，并伴有数据流。此外，还存在如下复杂约束：① $\nabla^{Ii}_{Run()}\wedge\nabla^{Ii}_{Commit()}$ 指 I_i 执行了 *Run*()和 *Commit*()；② $\neg\nabla^{Ii}_{Fail()}\wedge\neg\nabla^{Ii}_{Commit()}$指 I_i 不可能执行了 *Fail*()和 *Commit*()；③ $\nabla^{Ii}_{Compensate()}\rightarrow\nabla^{Ii}_{Commit()}$指 *Commit*()需要在 *Compensate*()之前执行；④ $\nabla^{Ii}_{Retry()}\rightarrow(\nabla^{Ii}_{Fail()}\oplus\nabla^{Ii}_{Retry()})$指 *Retry*()需要在 *Fail*()之后执行；⑤ $\nabla^{Ii}_{Commit()}\wedge\nabla^{Ii}_{Compensate()}\rightarrow(\nabla^{Ii}_{Commit()}\oplus\nabla^{Ii}_{Compensate()})$指 *Commit*()和 *Compensate*()都发生，且 *Commit*()在 *Compensate*()之前执行。

任何复杂约束都可以转换为$\vee_i(\wedge_j Constr_{i,j})$，其中，$Constr_{i,j}$为约束表达式。同样可以讨论 I_i 和 I_j 之间的复杂约束，例如$\nabla^{Ii}_{Activate()}\rightarrow\nabla^{Ij}_{Commit()}$是指如果 I_i执行 *Activate*()，则 I_j必须在其之前执行 *Commit*()；$\nabla^{Ii}_{Fail()}\rightarrow\nabla^{Ii}_{Compensate()}$是指 I_i执行失败将导致 I_j补偿。

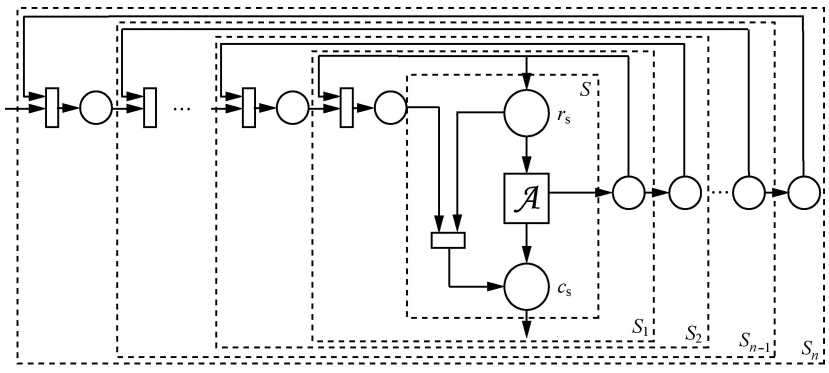


图 3 嵌套范围

LRT执行过程中，可能出现 3 种类型失败：① 调用Web服务执行失败，失败信息在调用服务描述文档中给出；② 调用或执行服务超时；③ 执行引擎失败。本文讨论的失败恢复主要针对前 2 种情形，因此在发布服务时，需要给出服务契约，任务执行时，给出调用契约：① 若I_i和I_j存在满足契约$CnlContract(I_j)$，当且仅当I_i失败且$I_i.Failed \in CnlContract(I_j)$，则触发依赖$I_j \xrightarrow{Cnl} I_i$；② 若$I_j$满足中止契约$AbtContract(I_j)$，$I_i.failed \in AbtContract(I_j) \vee I_i.Aborted \in AbtContract(I_j) \vee I_i.Cancelled \in AbtContract(I_j)$，中止$I_i$则触发中止依赖$I_j \xrightarrow{Abt} I_i$；③ 若$I_i$和$I_j$满足补偿契约 $CptContract(I_j)$，$I_i.failed \in CptContract(I_j) \vee I_i.Compensated \in CptContract(I_j)$，$I_i$失败则触发依赖$I_j \xrightarrow{Cpt} I_i$。为了更准确地选择RHS提供失败恢复，RH用恢复契约来配置RHS，优点是随时增加或减少契约条目。

4　失败恢复模型及算法

目前业界所提供的失败处理规范主要是向后恢复，恢复代价较高开销大，其没有考虑失败修复后 LRT 继续执行。基于此，本文提出一种支持组合事务的综合恢复策略，除了保留向后恢复，还扩展了向前和替代恢复。

4.1　向后恢复模型及算法

LRT 执行失败，I_j激活 *fail*()，中止正向流，触发 RH，RH 捕获 I_j事件 *fail*()，计算Ξ，抽取执行日志，计算恢复输入$I_i'.\text{in}$ (从$I_i.In$和$I_i.Out$获取)和$I_i'.Cp$ (由$I_i.TBP$ 和 $I_i.State$ 产生)，启动向后恢复策略。为便于问题描述，将逆向恢复流从正向事务流中分离出来，Ξ中的任务满足 $I_i.TBP \in \{Compensable, Vital\}$，RH 激活 I_i'，I_i逐个映射 I_i'。下面给出向后恢复配对策略 BRPS 的形式化定义。

假定$I=(P,T,F)$和$I'=(P',T',F')$分别表示LRT的正向和逆向流，$\exists \Xi \forall I \in \Xi | I \div I' \in T \div T'$，$t_i^c$映射 I_i到 I_{i+1}的补偿，t_i^f映射 I_i到 I_i'的失败转移，$t_i'^c$描述 I_i'到 I_{i+1}'的逆向转移。BRPS 的三元组表示为($\overline{P}$, $\overline{T}$, $\overline{F}$)：① $\overline{P}=P \cup P'$；② $\overline{T}=T \cup T' \cup T^f \cup T^c \cup T'^c$；③ $\overline{F}=F \cup F' \cup F^c \cup F^f \cup F'^c$，其中，$F^c=\{(p_i,t_i^c) \cup (t_i^c,p_{i+1}) | p_i, p_{i+1} \in P, t_i^c \in T^c\}$，$F^f=\{(p_i,t_i^f) \cup (t_i^f,p_i') | p_i \in P, p_i' \in P', t_i^f \in T^f\}$，$F'^c=\{(p'_{i+1},t_i'^{fc}) \cup (t_i'^c,p_i') | p_i', p'_{i+1} \in P, t_i'^c \in T'^c\}$。下面给出 BRPS 的逆向构建算法。

算法 1　构造恢复流算法

输入：LRT 中任务 I 的集合，失败任务 I_ℓ和补偿依赖集 CP

输出: 恢复流 Rflow

taskstate=*getTaskState*(I_ℓ,*RecordLog*);

if *taskstate*=*Failed* then

Ξ_1=*caculateCompensationScope*(I_i);/* 计算 I_ℓ 的失败补偿范围Ξ_1*/

else if *taskstate*=*Aborted* then

Ξ_2=*caculateCompensationScope*(I_i);/* 计算 I_ℓ 的中止补偿范围Ξ_2*/

else if *taskstate*=*Cancelled* then

Ξ_3=*caculateCompensationScope*(I_i);/* 计算 I_ℓ 的取消补偿范围Ξ_3*/

for each $I_i \in \Xi_1$ or $I_i \in \Xi_2$ or $I_i \in \Xi_3$

if *binding*(I_i,*ws*)and *getTaskState*(I_ℓ, *RecordLog*)= *Committed* then

对 LRT 中已提交的任务 $I_i \rightarrow I_i'$,$(P,T,F) \rightarrow (P', T',F')$;

for each *cpair*(I_i,I_i')and($I_i \in \Xi_1$ or $I_i \in \Xi_2$ or $I_i \in \Xi_3$)

if *dep*(I_i,I_i')=*true* then

构造 I_i,I_i'及相邻 I_i,I'_{i+1}之间的边(p_i,t_i^c),(p_i,t_i^f), (t_i^f,p_i'), $(p'_{i+1},t_i'^c)$, $(t_i'^c,p_i')$,建立恢复流 Rflow;

return Rflow

该算法的时间复杂度与 LRT 中任务数目及Ξ中补偿任务数有关，若 $\max(|LRT|,|\Xi|)=n$，则该算法的时间复杂度为 $O(mn)$。下面给出向后恢复算法。

算法 2　RH 恢复执行算法

输入: LRT 中任务 I 的集合，恢复流 *Rflow*

输出: 补偿服务执行信息

for *each LogItem*=*seek*(*RecordLog*,*I*) *of task I in the committed process*

if $I_i.state$=*Completed* then

for each *binging*(I_i,*ws*)

$I_i' \leftarrow$*ActID*,*Desc*,*Qos*,*TBP*,*Behavior*,*State*,Γ,*In*,*Out log information of ws*;

if $I_i.Failed$=*true* then

trigger *Compensating Handler CH*;

if $I_i.TBP \in \{Compensable, Vital\}$ *and* $I_i.state$= *Completed and* I_i *is atomic task* then

for *each compensation task* I_i'

CH excute I_i' and record excution information of I_i'

else if I_i is composition task then

{*divide task* I_i' *into subtask* I_{i1}', I_{i2}',…,I_{ik}';
for each subtask I_ℓ'(ℓ *from* i_1 *to* i_k)
excute(I_ℓ');}
else if *compensate*(I_i)=*false* and I_i is atomic task then
Forward I_i' ,compensation continue;if *notExit* (*CBP*) then
Compensation flow LRT is not exist;
for each task I_i
{if $I_j \gg^{Abt} I_i$ and I_i.*state=running* then
{*CH send abort*() *to* ws_i;
set task I_i'*s* state as "*Aborted*";}
if $I_j \gg^{Cnl} I_i$ and I_i.*state=active* then
{*CH send cancel*() *to* ws_i;
set task I_i'*s* state as "*cancelled*";}
if $I_j \gg^{Cpt} I_i$ then
Compensate task I_i', set I_i'*s* state as "*cancelled*";
}
if allcompensation(*CBP*)=true then
return success
else
return failure;
}

若|Rfolw|=n 和 Edge(I_ℓ)=m 分别为 Rfolw 中任务数和边数，该算法的时间复杂度为 $O(mn)$。

4.2 向前恢复模型及算法

与 BRPS 模型构造类似，不同的是向前恢复模型回滚到 TDP 后，RH 注入正确输入，重启正向流执行。假定 I=(P,T,F)是 LRT 正向流，若 I_j 执行失败，计算其 I_i.*TDP* 和Ξ，向前恢复策略 FRPS 的三元组表示为 $(\vec{P},\vec{T},\vec{F})$，其中：① $\vec{P}=P\cup\bar{P}$；② $\vec{T}=T\cup\bar{T}\cup\{t^r\}$；③ $\vec{F}=F\cup\bar{F}\cup\{(p_i',t^r),(t^r,p_i)\}$。下面给出 FRPS 恢复算法。

算法 3 向前恢复 FRPS 算法

输入：失败任务 I_ℓ，执行日志 *Log*，业务流程 LRT

输出：恢复成功或失败信息

{*taskstate=getTaskState*(I_ℓ,*RecordLog*);
if *taskstate=Failed* then
Ξ_1=*caculateCompensationScope*(I_i);/* 计算 I_ℓ 的失败补偿范围Ξ_1*/
if *taskstate=Aborted* then
for *each task* I_k.*state=Running*
if $I_\ell \gg^{Abt} I_k$ and $I_k\in\Xi_1$ then
Interrupte each task I_k *in runningflow*;
for *each* $I_\ell\in\Xi_1$
{if *getTaskState*(I_ℓ,*RecordLog*)=*committed* then
{*LogItem=seek*(*RecordLog*, I_ℓ)
input the value of the LogItem to task I_l and *compensate* I_l;
}}
for *each* $I_\ell\in\Xi_1$
Re-excute I_ℓ and *change the value of the LogItem of task* I_l;
if I_ℓ.*state=Failed* or I_ℓ.*state=Cancelled* or I_ℓ.*state=Aborted* then
Delete log *record in FailedLog*,*AbortedLog*, *CancelledLog*;
}

逆向执行恢复流的时间复杂度为 $O(n)$，而从 TDP 重新执行 LRT 的时间复杂度为 $O(n)$，因此，FRPS 算法的时间复杂度为 $O(n)$。

4.3 替代恢复模型

LRT 失败，失败任务 I_i.*TDP*={I_j}且 I_i.Ξ=I_i，若{I_i}={I_j}，则 I_i 的失败发生不依赖其他任务，选择 FRPS 或 BRPS 恢复代价都过高，仅需要选择 I_i 的替代服务，替代恢复 ARS 的形式化描述如下。

假定 I_i=(P_i,T_i,F_i)是 LRT 正向流，I_j 执行失败，计算 I_i.*TDP* 和 I_i.Ξ，若 $\exists\Xi\forall I_j\in\Xi|I_i=I_j$，引入替代流 I_i''=(P_i'',T_i'',F_i'') ARS $(\bar{P},\bar{T},\bar{F})$，其中：① $\bar{P}_i=P\cup P''$ ② $\bar{T}=T\cup T''\ T^f\cup T^r\cup T^a\cup\tau$；③ $\bar{F}=F\cup F''\ \{(I_i.p,t_i^f),(t_i^f,I_i''.p),(t_1^r,I_i''.p),(I_i''.p,t_2^r)\}$。LRT 执行失败，依据 I_i 失败类型选取 RHS，BRPS 恢复代价最高，ARS 最低，FRPS 介于两者之间。

5 LRT 的事务性质扩展

LRT往往涉及复杂恢复行为，并以不同聚合模式构造。LRT的事务性质需借助基本聚合模式的事务性质，并扩展LRT的事务性质。假定有 3 个聚合服务cs_1、cs_2 和cs_3 对应的聚合片段分别为$I_i\odot\cdots\odot I_j$、$I_m\odot\cdots\odot I_n$和$I_p\odot\cdots\odot I_q$，其中，$\odot\in\{\oplus,\|,\otimes,\Theta\}$，借助$\oplus$,$\|$,$\otimes$,$\Theta$和$\lfloor$聚合$cs_1$、$cs_2$和$cs_3$，形成一组新的聚合服务集合 $cs_1\oplus cs_2$、$cs_1\oplus(cs_2\|cs_3)$、$(cs_1\|cs_2)\oplus cs_3$、$cs_1\oplus(cs_2\otimes cs_3)$、$(cs_1\otimes cs_2)\ cs_3$、$cs_1\otimes(cs_2\Theta cs_3)$、$(cs_1\Theta cs_2)\oplus cs_3$、$\lfloor\underline{cs_1}\oplus cs_2$ 和

$cs_1 \oplus |\underline{cs_2}$。

聚合事务除了满足⊕,||,⊗,Θ和⌊的聚合事务性质外，还需要检查 cs_1、cs_2 和 cs_3 在⊕,||,⊗,Θ和⌊上的连接点，是否满足聚合需求。

1) 对于顺序聚合 $cs_1 \oplus cs_2$，$cs_1.TBP \otimes cs_2.TBP$ 的聚合事务性质需检查 cs_1 中 $I_i \odot \cdots \odot I_j$ 的 $I_j.TBP$，cs_2 中 $I_m \odot \cdots \odot I_n$ 的 $I_m.TBP$，检查连接点 $I_j.TBP \oplus I_m.TBP$ 的事务性质，特别地，$I_j.Retriable \oplus I_m.Compensable$ 不能保证 LRT 的原子性。

2) 并行组合 $cs_1 \oplus (cs_2 \| cs_3)$ 需要检查连接点 $I_j.TBP \oplus I_m.TBP$ 和 $I_j.TBP \oplus I_p.TBP$ 的事务性质；而 $(cs_1 \| cs_2) \oplus cs_3$ 则需要检查连接点 $I_j.TBP \oplus I_p.TBP$ 和 $I_n.TBP \oplus I_p.TBP$ 的事务性质。

3) 选择聚合 $cs_1 \oplus (cs_2 \otimes cs_3)$ 和鉴别器 $cs_1 \oplus (cs_2 \Theta cs_3)$ 类似于 $cs_1 \oplus (cs_2 \otimes cs_3)$ 连接点分析，检查 $I_j.TBP \oplus I_m.TBP$ 和 $I_j.TBP \oplus I_p.TBP$ 的事务性质；分析 $(cs_1 \otimes cs_2) \oplus cs_3$ 和 $(cs_1 \Theta cs_2) \oplus cs_3$ 类似于 $(cs_1 \| cs_2) \oplus cs_3$。

6　失败恢复网的性质分析

依据 RHS 定义，仅有一个初始托肯处于源库所，如果执行 LRT 没有点火任何失败变迁，托肯将随正向传递，直到成功到达槽库所。但是，LRT 的执行可能导致失败发生，设失败恢复网系统 $\sum=(FRS, M_0)$，依据失败恢复类型不同，选取 $(\bar{P},\bar{T},\bar{F})$ 或 $(\vec{P},\vec{T},\vec{F})$ 或 (P_i'',T_i'',F_i'')，那么按流方向 $\sum$ 被划分为正向网系统 $\sum_e=(N_e, M_e)$ 和恢复网系统 $\sum_r=(N_r^{-1}, M_r)$，若 $\forall\sigma\in T^* \wedge \exists t^f\in\sigma \wedge t^f\in T^f$，满足：① $\exists M_1, M_2\in[M_e> \wedge M_1'\in[M_r>$，存在恢复路径 $\sigma=\sigma_1 t^f\sigma_2 \wedge \sigma_1\in T^*$，使得 $M_1[\sigma_1>M_2[t^f>M_3$。即细化 $M_{rf1}=[\sum_{e1}, M_{e0}]\in RS(\sum_r, M_{r0}) \wedge M_{rf2}=[M_{e2}, M_r]\in RS(\sum_r, M_{r1})$，使得 $M_{rf1}[\sigma_1>M_{rf2}[t^f> M_{rf3} \wedge M_{rf3}=[M_{e3}, M_{r1}]\in RS(\sum_r, M_{r2})$。$t^f$ 点火 $\sum_r$ 中恢复步序列 $\sigma_2\in T'^*(\|\sigma_2\|\neq 0)$，使得 $M_{e3}[\sigma_2>$，称 σ_2 是 σ_1 的恢复路径。② 若 $\exists t^f\in\sigma \wedge t^f\in T^f \wedge \sigma=\sigma_1 t^f\sigma_2$，令 $\Gamma_{t_e}^{\sigma_1}$ 和 $\Gamma_{t_r}^{\sigma_2}$ 分别为 I_e 及其恢复任务 I_r 在 σ_1 和 σ_2 上的投影序列，若 $I_e\in\Gamma_{t_e}^{\sigma_1} \wedge I_r\in\Gamma_{t_r}^{\sigma_2}$，则有恢复映射 $f: \sigma_1\rightarrow\sigma_2$；③ σ_2 是 σ_1 的恢复路径且存在映射 $f:\sigma_1\rightarrow\sigma_2$，若 $\|\sigma_1\|\neq 0$，则 $\|\sigma_2\|\neq 0$，称 $\sum$ 是可正确恢复的。

定理 1　设失败恢复网系统 $\sum=(FRS, M_0)$，其正向网和恢复网系统分别为 $\sum_e=(N_e, M_e)$ 和 $\sum_r=(N_r^{-1}, M_r)$，若 $\sum_e$ 执行失败，则 $\sum$ 是恢复可达的。

证明　$\sum_e$ 和 $\sum_r$ 中正向流和恢复流分别为 $(P\times T)\cup(T\times P)$ 和 $(P'\times T')\cup(T'\times P')$，由 $\sum_e$ 执行失败，依据 $(\bar{P},\bar{T},\bar{F})$，$(\vec{P},\vec{T},\vec{F})$ 和 (P_i'', T_i'', F_i'') 定义，构造 $(P\times T^f)\cup(T^f\times P')\cup(T\times P^f)\cup(P^f\times T')$ 失败转移，从而转向恢复流 $\sigma(\sigma\in TP^fT'$ 或 $\sigma\in(TP)^*T^f(P'T')^*)$，$P^f$ 或 T^f 使得 $\sum$ 恢复可达。设 $M_1, M_2\in M_0[> \wedge M_1\in RS(\sum_e)$，存在恢复步序列 σ，使得 $M_1[\sigma>M_2 \wedge M_2\notin RS(\sum_e)$，若失败发生 $t^f(t^f\in T^f)$，$\sigma\in TP^fT'$ 或 $(TP)^*T^f(P'T')^*$（记为 $\sigma_1t^f\sigma_2$），使得 $M_1[\sigma_1>M_1' \wedge M_1'\in RS(\sum_e)$，$M_1'[t^f>M_1'' \wedge M_1''\notin RS(\sum_e))$，即 t^f 点火 $\sum$ 中的步序列 σ_2，使得 $M_1''[\sigma_2>$，从而 $\sum$ 恢复可达。

定理 2　设失败恢复网系统 $\sum=(FRS, M_0)$，其中 RHS 可以是 $(\bar{P},\bar{T},\bar{F})$ 或 $(\vec{P},\vec{T},\vec{F})$ 或 (P_i'', T_i'', F_i'')，$\sum$ 保证正确恢复。

证明　设 $\sum$ 的正向网系统和恢复网系统分别是 $\sum_e=(N_e, M_e)$ 和 $\sum_r=(N_r^{-1}, M_r)$，t^f 是 $\sum$ 的失败发生，那么 $\bullet t^f\subseteq\sum_e \wedge t^f\bullet\subseteq\sum_r$，$\sum$ 中任意包含 t^f 的执行路径 σ（其中 σ 可表示为 $\sigma_1t^f\sigma_2$），若 $[M_e, M_{r1}]\in RS(\sum, M_0) \wedge M_{e1}\in RS(\sum_e, M_{e0}) \wedge M_{r1}\in RS(\sum_r, M_{r0})$，存在正向执行路径 σ_1，使得 $M_0[\sigma_1>M_{e1}[t^f>M_{r1}$，依据 t^f 失败类型，FRS 触发失败恢复策略 $(\bar{P},\bar{T},\bar{F})$ 或 $(\vec{P},\vec{T},\vec{F})$ 或 (P_i'',T_i'',F_i'')，则 t^f 点火 $\sum_r$ 恢复步 $\sigma_2\backslash t^f$，并执行 $\sigma_2\backslash t^f$，使得 $M_{r1}[(\sigma_2\backslash t^f)>$，满足 $f: \sigma_1\rightarrow\sigma_2$，$\sum$ 是可正确恢复的。

对于 (FRS, M_0) 中的 σ，若 $FRS=(\bar{P},\bar{T},\bar{F})$，步序列 $\sigma_1t^f\sigma_2$，由 BRPS 有 $\left\|\Gamma_{I_e}^{\sigma_{12}}\right\|=\left\|\Gamma_{I_r}^{\sigma_{21}}\right\|$ 且 $f: I\rightarrow I'$（其中，$I\in\sum_e$，$I'\in\sum_r$），即 t^f 点火 $\sum_r$，I' 消除 I 的影响；若 $FRS=(\vec{P},\vec{T},\vec{F})$ 或 (P_i'',T_i'',F_i'')，$M_{rf1}=[M_{e1}, M_{r1}]\in RS(\sum, M_0)$，其中，$M_{e1}\in RS(\sum_e, M_{e0})$，$M_{r1}\in RS(\sum_r, M_{r0})$，使得 $M_0[\sigma_1>M_{e1}[t^f> M_{r1}$，$t^f$ 点火 $\sum_r$ 恢复步 $\sigma_2\backslash t^f$ 并执行，使得 $M_{r1}[(\sigma_2\backslash t^f)>M_{e2} \wedge M_{e2}\in RS(\sum_e, M_{e0})$，$\sum$ 恢复后执行继续，称 $\sum$ 是可恢复的。

定理 3　设失败恢复网系统 $\sum=(FRS, M_0)$，RHS 是 $(\bar{P},\bar{T},\bar{F})$ 或 $(\vec{P},\vec{T},\vec{F})$ 或 (P_i'',T_i'',F_i'')，$\sum_e=(N_e, M_e)$ 和 $\sum_r=(N_r^{-1}, M_r)$ 分别是 $\sum$ 的正向网和恢复网系统，$\sum$ 失败恢复是安全的。

证明　$\forall M_{rf1}\in(\sum, M_0)\exists t_r\in T^*$，依据 t^f 失败类型，触发 $(\bar{P},\bar{T},\bar{F})$ 或 $(\vec{P},\vec{T},\vec{F})$ 或 (P_i'',T_i'',F_i'')，分 3 种情形证明 $\sum$ 的安全性：① 若 $t_r\in\bar{T}$，由于 $\sum_e=(N_e, M_e)$ 是安全的，$\forall M_{e0}\in RS(\sum_e, M_0)\exists M_{e1}\in RS(\sum_e, M_0)$，存在执行路径 $\sigma_0\in\bar{T}^*$，使得 $M_{e0}[\sigma_0>M_{e1}[t_r>$，$t_r$ 仅影响 $\sum_e$ 中 M_{e1}，而 $\sum_r$ 中标识不变，依据 $\sum_r$ 定

义，存在标识转移 $M_{e1}[t_r>M_{r1}[\sigma_1>$(其中，$M_{r1}\in RS(\sum_r, M_{r0})$)，$(\vec{P},\vec{T},\vec{F})$ 确保 $\sum$ 安全恢复。② 若 $t_r\in\vec{T}$，$\exists M_e\in RS(\sum_e, M_0)\wedge M_r\in RS(\sum_r, M_{r0})\wedge M_{e1}\in RS(\sum_e, M_1)\wedge M_{c1}\in RS(\sum_r, M_{r1})\wedge\sigma_0\in\overline{T}^*$，使得 $M_{r0}[\sigma_0>M_{r1}[t_r>$，$t_r$ 点火 $\sum_r$ 中 M_{r1}，表明 $\sum_r$ 是安全的。由于 $\sum_e$ 是安全的，有 $M_{r1}[t_r>M_1[\sigma_1>$，$\sum_r$ 成功恢复，恢复策略 $(\vec{P},\vec{T},\vec{F})$ 确保 $\sum$ 安全恢复。③ 若 $t_r\in T''\wedge\exists M_e\in RS(\sum_e, M_0)\wedge M_{e1}\in$RS$(\sum_e, M_1)\wedge t''\in T''$，使得 $M_e[\sigma_0>M_e[t_r>M_{e1}[\ t''>M_{e2}[t>$(其中 $M_{e2}\in RS(\sum_e, M_2)$)，$t_r$ 点火 I'' 并执行它，转向 $\sum_e$ 执行继续，$(\vec{P},\vec{T},\vec{F})$ 确保 $\sum$ 安全恢复。

7 基于组合事务恢复的应用

组合事务应用实例用经典的旅行预订流程 TRP，如图 4 所示，旅行者 TE 向旅行社 TA 提交旅行需求说明 CRS；由 CRS 分别预订机票 FB(或火车票 TR)和宾馆 HB，租赁小车 CR(或大巴 BR)，TE 确认预订并在线支付 OP，快递 EMS(或 UPS)投递给 TE，TE 签收确认 TC。聚合嵌套后的事务流程为 $CRS\oplus((FB\otimes TR)\|HB\|(CR\|BR))\oplus OP\oplus(EMS\otimes UPS)\oplus TC$。实例中旅行社在广州，而航空公司、酒店和汽车租赁公司分布于全国 20 多个景点，表 2 给出了 TRP 中可能组合的服务及其事务性质。若 TRP 聚合了 WS^p，其既不可补偿也不可重试。要获得 LRT 事务原子语义，WS^p 必须保证在执行向后恢复之前提交或确保不发生失败。

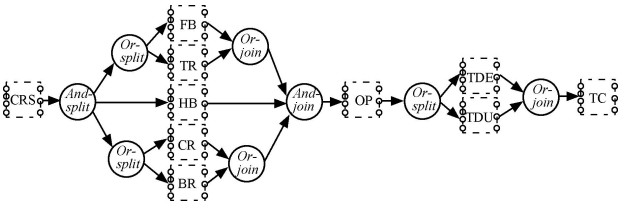


图 4　旅行预订流程

试验环境配置：执行引擎选用 ActiveBPEL，2 台服务器 IBM x3650 (4 核 Intel 2930MHz CPU、1GB RAM 和 SuSE 10.2 Linux OS)和 12 台网络工作站 (Intel E5200 2.5 GHz CPU, 2GB RAM 和 Windows XP SP2)。TE 自主选择旅行路线并提交给 TA，TA 生成路线图(协商最短路线)，TE 确认路线，旅行社预订机票、火车票、酒店以及出行交通工具，向 TE 提交预算，一旦预订失败，协商预订子流程：① 变更无法预订的服务，如将交通工具从火车改飞机、将酒店单人间改双人间。② 变更不能成功预订的景点，系统推荐景点。模拟事务回滚机制，自动退订已预订的机票和酒店等。③ 删除不能预订的景点。预订成功，将预订信息返回给 TE。组合事务的执行语义保证组合事务正确执行，其不仅可以检测 TRP 执行的逻辑错误，还能检测执行失败恢复的

表 2　　**可用服务及其事务性质**

Task	WS	TP	Task	WS	TP	Task	WS	TP	Task	WS	TP
CRS	$WS_{0,1}$	p	FB	$WS_{1,1}$	cp	TR	$WS_{2,1}$	cp	HB	$WS_{3,1}$	R
	$WS_{0,2}$	r		$WS_{1,2}$	r		$WS_{2,2}$	p		$WS_{3,2}$	r,cp
	$WS_{0,3}$	r		$WS_{1,3}$	r,cp		$WS_{2,3}$	cp		$WS_{3,3}$	p
	$WS_{0,4}$	p		$WS_{1,4}$	p		$WS_{2,4}$	r		$WS_{3,4}$	cp
	$WS_{0,5}$	r		$WS_{1,5}$	p		$WS_{2,5}$	r,cp		$WS_{3,5}$	p
	$WS_{0,6}$	p		$WS_{1,6}$	r,cp		$WS_{2,6}$	p		$WS_{3,6}$	cp
	$WS_{0,7}$	p		$WS_{1,7}$	r		$WS_{2,7}$	r,cp		$WS_{3,7}$	cp
BR	$WS_{5,1}$	P	OP	$WS_{6,1}$	p	TDE	$WS_{7,1}$	r	TDU	$WS_{8,1}$	P
	$WS_{5,2}$	r,cp		$WS_{6,2}$	r		$WS_{7,2}$	p		$WS_{8,2}$	Cp
	$WS_{5,3}$	cp		$WS_{6,3}$	r		$WS_{7,3}$	p		$WS_{8,3}$	cp
	$WS_{5,4}$	cp		$WS_{6,4}$	p		$WS_{7,4}$	r		$WS_{8,4}$	cp
	$WS_{5,5}$	r,cp								$WS_{8,5}$	p
	$WS_{5,6}$	p								$WS_{8,6}$	p
CR	$WS_{4,1}$	r,cp	CR	$WS_{4,4}$	cp	TC	$WS_{9,1}$	r	TC	$WS_{9,4}$	r
	$WS_{4,2}$	p		$WS_{4,5}$	p		$WS_{9,2}$	p			
	$WS_{4,3}$	r,cp		$WS_{4,6}$	cp		$WS_{9,3}$	p			

正确性（如图 5 所示）。如图 6 所示，若预订 type 为 hotel，进入酒店预订子流程，当 star 为三星级进入 orderwanjia，Towanjia 预订万家酒店，Invokewanjia 调用 WanJia HotelServcie。退订 type 为 hotel 时，进入退订酒店子流程，当退订酒店为 huatian，将 HuaTianReturnReq 传递到 tohuatian 退订酒店，invokehuatian 调用 HuaTian HotelService，完成相应的恢复。

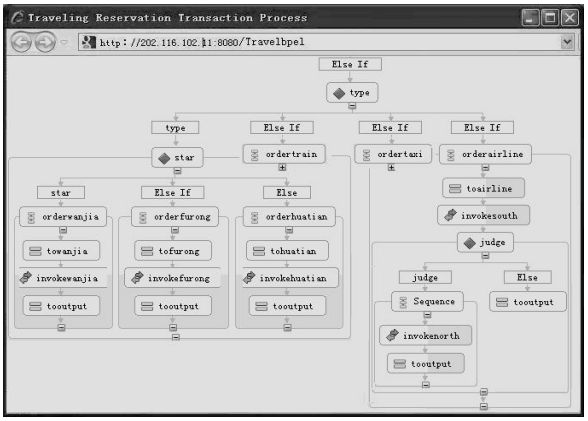


图 5　TRP 组合业务流及其恢复流

图 6　TRP 执行预订及其变更

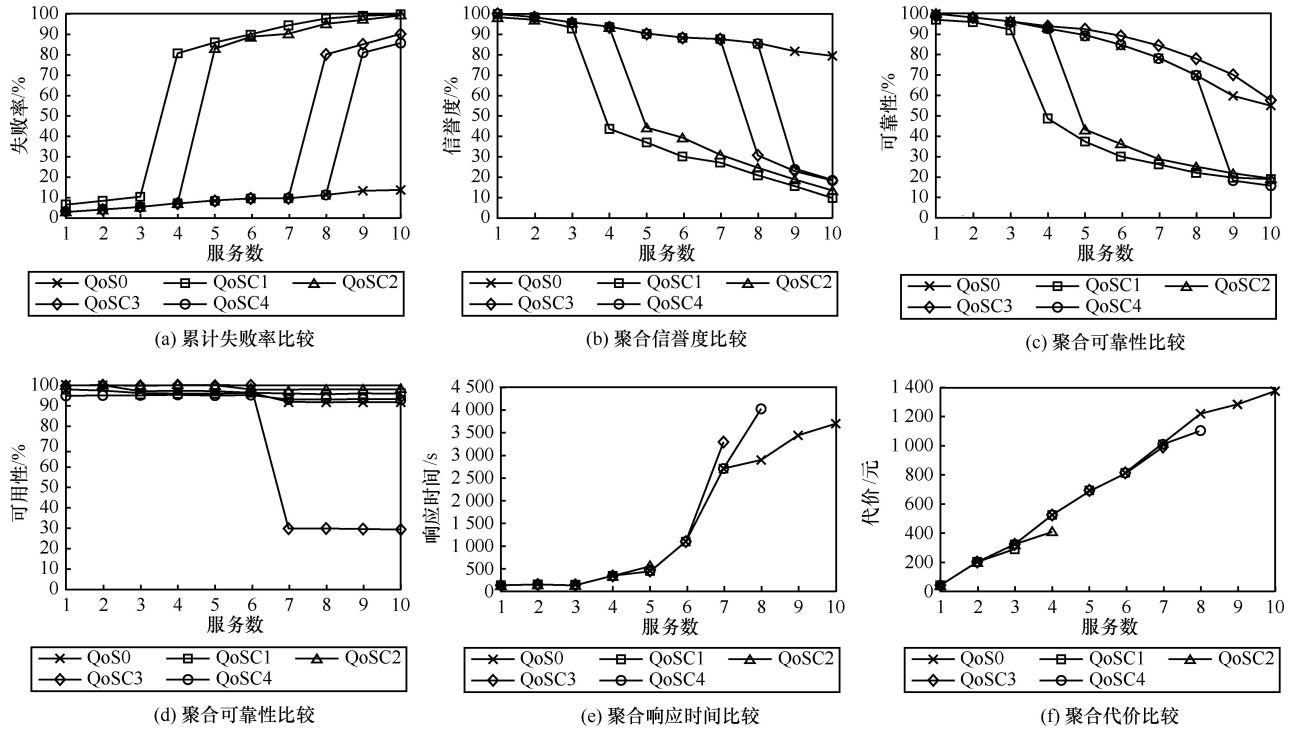
TRP 预订流程中，旅行社负责预订航班、酒店和租赁汽车。如果预订机票失败，需要取消酒店预订和汽车租赁，最后与旅行者协商改火车出行，一旦成功预订火车票将不能再次变更。表 3 描述 TRP 中多伙伴协作操作对，失败恢复时，首先检查发生失败任务是否存在恢复任务，如果存在，执行恢复；否则，执行空操作。恢复对用 *I.action*÷*I'.action* 表示，

表 3　　TRP 中恢复对描述

恢复对	含义
TA.ReceiveRequest()÷*TA.undoReceiveRequest*()	前者指 TA 接收请求，后者为 TA 撤消预订
TA.PutRouteLeg()÷*TA.UndoRoute*()	前者指 TA 提供路线，后者为 TA 撤消路线
TA.ReceiveAcknowledgment()÷Ø	前者指旅行者接收路线确认
TA.SendReserveAirlineTickets()÷*TA.UndoReserveAirlineTickets*()	前者指 TA 预订机票，后者为 TA 撤消机票预订
TA.SendReserveTrainlineTickets()÷*TA.UndoReserveTrainlineTickets*()	前者指 TA 预订火车票，后者为 TA 撤消火车票预订
TA.SendRoomsBooking()÷*TA.UndoRoomsBooking*()	前者指 TA 预订酒店，后者为 TA 撤消酒店预订
TA.SendCarRenting()÷*TA.UndoCarRenting*()	前者指 TA 租赁小汽车，后者为 TA 撤消小汽车租赁
TA.SendBusRenting()÷*TA.UndoBusRenting*()	前者指 TA 租赁巴士，后者为 TA 撤消巴士租赁
TA.ReceiveAirlineTickets()÷Ø	前者指 TA 接收机票及保险
TA.ReceiveTrainTickets()÷Ø	前者指 TA 接收火车票
TA.ReceiveHotelTickets()÷Ø	前者指 TA 接收酒店预订
TA.ReceiveCarRenting()÷Ø	前者指 TA 接收小汽车预订
TA.ReceiveBusRenting()÷Ø	前者指 TA 接收巴士预订
TA.SendReservePayment()÷*TA.UndoReservePayment*()	前者指 TA 在线支付旅行付款，后者为 TA 撤消旅行支付
TA.PaymentAcknowledgment÷Ø	前者指 TA 发送付款确认
TA.SendEMSDeliveryRequest()÷*TA.UndoEMSDeliveryRequest*()	前者指 TA 请求 EMS 递送，后者为 TA 撤消 EMS 递送
TA.SendUPSDeliveryRequest()÷*TA.UndoUPSDeliveryRequest*()	前者指 TA 请求 UPS 递送，后者为 TA 撤消 UPS 递送
TA.SendAcknowledgment()÷Ø	前者指 TA 确认旅行预订

其中，*I.action* 表示 *I* 正向执行操作，*I'.action* 表示

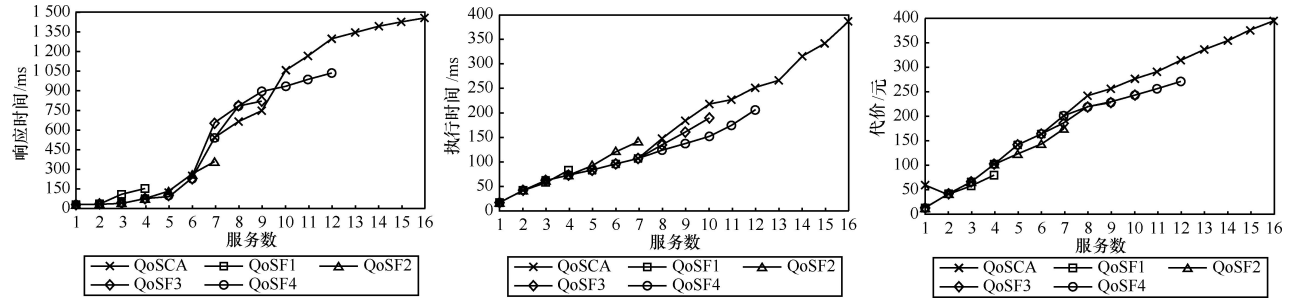
*I'*所执行恢复，特别地，*Ø*是空操作。



(a) 累计失败率比较　(b) 聚合信誉度比较　(c) 聚合可靠性比较

(d) 聚合可靠性比较　(e) 聚合响应时间比较　(f) 聚合代价比较

图 7　失败聚合性能比较



(a) 响应时间　(b) 执行时间　(c) 代价

图 8　失败恢复后性能比较

本文将 TRP 组合服务分为 4 组，前 3 组分别在不同执行任务点失败，第 4 组则执行成功，比较 4 组恢复策略下的失败率比较。图 7 给出了 TRP 执行失败的聚合性能曲线，这里的聚合曲线是累计聚合值。应用第 4 节给出的组合事务恢复算法讨论 TRP 失败恢复，更好地指导事务恢复。分 5 组聚合服务来执行 TRP 流程，结果显示，仅 QoS0 曲线成功执行，其他 4 条曲线 QoSC1、QoSC2、QoSC3 和 QoSC4 分别执行到第 3、4、7 和 8 个服务时失败，从图 7 可以看出，失败率与选取不同性质的原子服务有关，失败率与服务信誉度基本一致，与响应时间相比仅在第 7 和第 8 个失败处显著增加，与可靠性相比仅在第 3、4 和 8 个失败处一致，与可用性相比仅在第 7 个失败处一致，但执行代价和执行时间在执行恢复策略前后几乎无变化。

图 7 中曲线分别对应图 8 中 QoS0、QoSC1、QoSC2、QoSC3 和 QoSC4 恢复曲线，在第 3、4、7 和 8 个失败处启动恢复策略，恢复执行后，对响应时间产生了影响，但执行时间和代价变化不大。综上分析，可以得出以下结论：① 采用合适的恢复策略，可有效减少失败恢复代价；② 尽管失败不可避免，但最终可筛选出失败率较低的 LRT；③ 受失败率影响，聚合曲线可协调 LRT 中参与组合的服务，消除不利因素影响。

8　结束语

鉴于 LRT 事务的复杂性，LRT 中被组合任务

可能具有不同的事务性质，确保其在 SOC 环境中可靠执行，需要构建有效的失败恢复策略。为此，本文提出一种支持组合事务的失败恢复算法，为不同的失败发生选取合适的恢复策略。旅行预订实例分析表明该算法可确保 LRT 可靠执行，最后通过对 TRP 失败恢复分析，表明选择合适的恢复算法对减少事务失败率和消除不利因素影响是有利的。

未来的工作主要包括如下 4 个方面：① 引入事务流日志挖掘算法；② 建立组合事务恢复的依赖规则；③ 深入研究事务恢复的局部和全局约束；④ 违反组合事务性质的检测技术。

参考文献：

[1] LAKHAL N B, KOBAYASHI T, YOKOTA H. FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis[J]. The International Journal on Very Large Data Bases, 2009, 18(1):1-56.

[2] LIU A, LI Q, HUANG L S, *et al*. FACTS: a framework for fault-tolerant composition of transactional Web services[J]. IEEE Transactions on Services Computing, 2010, 3(1):46-59.

[3] HEWETT R, KIJSANAYOTHIN P. Privacy and recovery in composite Web service transactions[J]. International Journal for Infonomics, 2010, 3(2): 240-248.

[4] KOLLINGBAUM M, SYCARA K, VACULIN R, *et al*. Recovery mechanisms for semantic Web services[J]. Distributed Applications and Interoperable Systems, 2008, 17(3): 100-105.

[5] MINH T, LE N, CAO J L. Flexible and semantics-based support for Web services transaction protocols[A]. Grid and Pervasive Computing[C]. 2008. 492-503.

[6] BHIRI S, GAALOUL W, GODART C, *et al*. Ensuring customised transactional reliability of composite services[J]. Journal of Database Management, 2011, 22(2): 64-92.

[7] YANG Z H, LIU C F. Implementing a flexible compensation mechanism for business processes in Web service environment[A]. International Conference on Web Services[C]. Salt Lake City, USA, 2006.753-760.

[8] SCHAFER M, DOLOG P, NEJDL W. An environment for flexible advanced compensations of Web service transactions[J]. ACM Transactions on the Web, 2008, 2(2):1-36.

[9] SCHAFER M, DOLOG P, NEJDL W. Engineering compensations in Web service environment[A]. International Conference on Web Engineering[C]. Como, Italy, 2007. 32-46.

[10] KIM Y, KIM J. Allowing user-specified failure handling in Web services composition[A]. Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication[C]. Kuala Lumpur, Malaysia, 2008. 452-458.

[11] BRUNI R, FERRARI G, MELGRATTI H, *et al*. From theory to practice in transactional composition of Web services[A]. International Workshop on Web Services and Formal Methods[C]. Versailles, France, 2005. 272-286.

作者简介：

梅晓勇（1974-），男，湖南常德人，博士，中山大学副教授，主要研究方向为面向服务计算、Petri 网技术和可信计算等。

黄昌勤（1972-），男，湖南桃源人，博士，中山大学教授、博士生导师，主要研究方向为信息技术、服务计算和移动计算等。

郑小林(1977-)，男，浙江江山人，博士，浙江大学副教授，主要研究方向为电子商务、服务计算、移动计算等。

陈德人(1951-)，男，浙江杭州人，浙江大学教授、博士生导师，主要研究方向为电子商务与电子服务技术、信息系统集成。

李师贤(1944-)，男，江西于都人，中山大学教授、博士生导师，主要研究方向为软件工程和形式语义学。