代价驱动的服务组合

朱 锐[1,2] 王怀民[1,2] 唐扬斌[1,3]
(国防科学技术大学计算机学院 长沙 410073)[1]
(国防科学技术大学并行与分布处理国防科技重点实验室 长沙 410073)[2]
(国防科学技术大学理学院 长沙 410073)[3]

摘 要 互联网环境的开放性、自治性和动态性给基于互联网的资源共享与应用集成带来了新的挑战。在组合服务运行过程中服务质量保证和可靠性等研究方面,服务冗余技术、流程重构技术是非常重要的容错手段,但是目前尚存在着容错开销过大、难以满足实时性要求以及流程重构策略缺乏可行性等问题。服务容错技术是保证组合服务QoS和带来额外开销(代价)的一个权衡。在分析冗余技术、流程重构可能带来的高额容错代价后,提出用于减少容错代价的组合服务拆分原则,遵循该原则有助于组合服务设计者选择合适的组件服务粒度并降低服务间的耦合度。模拟实验表明,提出的方法有助于提高组合服务的可靠性并降低失效容错代价。

关键词 服务组合,服务粒度,服务冗余,流程重构,可靠性,容错

Cost-drive Service Composition

ZHU Rui[1,2] WANG Huai-min[1,2] TANG Yang-bin[1,3]
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)[1]
(Key Laboratory of Science and Technolgoy for National Defence of Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China)[2]
(School of Science, Nationd University of Defence Technology, Changsha 410073, China)[3]

Abstract The characteristics of Internet such as open, autonomy and dynamic impose new challenges on share and composition over internet resources. Meanwhile, researchs on runtime QoS-guarantee of Web service composition, service redundant technology and replanning are very important fault-tolerant methods. But they are facing severe problems on high-cost, requirement of real-time and infeasibility of replanning strategy. Fault tolerance is a trade-off between QoS-guarantee and extra-overhead. In order to reduce the fault-tolerant cost, several decomposed principias were presented based on analyise of high-cost brought by fault-tolerant technology. According to these principias, it is helpful for the designer to select the proper service granularity and reduce the degree of coupling of services. Simulation results indicate that the mechanism is effective to increase the reliability of composite services and reduce the fault-tolerant cost.

Keywords Servcie composition, Service granularity, Service redundant, Replanning, Reliability, Fault tolerance

随着Web服务技术的出现和推广,如何组合分布的、自治的Web服务以构建新的企业应用是近年的研究热点。互联网环境是一个非可靠异构的、分布自治的和快速演变的网络生态系统。组合服务在运行过程中可能会受到网络通信QoS变化、拒绝服务攻击、基础设施失效以及安全性等问题的影响,使组合服务的功能与服务质量难以得到保障。同时,由于组合后服务的行为并不仅仅是组件服务行为的简单加和,而是被组件服务之间的交互行为强烈影响着,这种功能和质量保障的难度被进一步放大,因此,灵活、有效的运行时保障机制就显得至关重要。

互联网上充斥着大量功能相同或相似的服务,因此,许多研究工作将研究重点聚焦于通过配置冗余服务(Redundant Services)来提高服务的执行成功率和服务质量上[1,2]。目前服务冗余技术可以划分为时间冗余和空间冗余两种:时间冗余指当故障发生时,将计算任务恢复到前一个正确的状态,并继续执行;空间冗余则指使用多个副本执行同一任务,以容忍单个副本失效。空间冗余又可分为主动冗余(active redundant)、被动冗余(passive redundant)以及混合冗余(hybrid redundant)[2],针对不同的应用场景每种冗余技术都能发挥其关键作用。流程重构(Replanning)是指组合服务运行时,由于需求变更和环境变化引起服务质量变化或不可用,服务组合技术使得应用系统能够通过更改或调整组合服务模型而实现快速应用重构。可以说,当冗余技术不能满足因需求或环境变化所引起的较大变动时,就需要采用流程重构技术。文

到稿日期:2009-06-05 返修日期:2009-09-01 本文受国家自然科学基金杰出青年基金(60625203),国家重点基础研究发展计划(973)(2005CB321800, 2005CB321801)资助。

朱 锐 博士生,CCF会员,主要研究方向为分布计算、可信计算等,E-mail:mruzcce@gmail.com;王怀民 教授,博士生导师,主要研究方向为分布计算中间件、软件Agent、网络与信息安全;唐扬斌 博士,主要研究方向为可信计算。

献[3]指出，若一个基于 Web 服务的系统没有动态调整的机制，那么该系统是不可靠的。Canfora 等人[4]提出计算组合服务 QoS 估计值与运行实测值的偏差，并在必要时进行在线的流程重构。Zeng[5]提出了 Web 服务组合中间件平台，为适应动态的网络环境，当组件服务不可用或 QoS 发生较大的变化时，一套流程重构程序（replanning procedure）被触发以调整组合服务模型。Chan[6]认为服务组合系统应具有自愈能力（self-healing），而自愈能力又包括监控能力、分析能力、流程重构能力和执行能力。当某个组件服务发生失效，自愈系统首先检查该服务的历史信息，如果历史记录表明该服务是较稳定的，那么就重新调用服务；如果历史记录显示该服务不稳定，则重新绑定一个可信的服务；如果没有可信的服务供绑定，则流程重构程序被触发。Peer[7]认为组合服务具有可用性未知、状态未知、服务行为不确定和结果不可预测等特性，他结合偏序规划（Partial Order Planning，POP）和流程重构算法，使用反馈机制从失败的计划中获取信息，从而避免了再次规划错误。Massimiliano[8]等人的研究工作聚焦在平台和语言方面，他们提出了 SCENE 平台并扩展了 BPEL 语言以支持服务组合的动态重配置。

以上研究工作通常把组合服务 QoS 的故障分为通信故障、崩溃故障和误差故障 3 类，并根据不同的故障类型做出重试、替换、流程重构的容错处理方式。表 1 描述了 3 种 QoS 故障类型。如果发生通信故障则首先采用重试调用的方法，其所带来的开销较小，甚至可以忽略不计；当规定时间内重试调用依旧失败，则从备选服务中选取一个可用的服务替换；如果没有可用的备选服务或者 QoS 评估值与运行实测值产生较大误差，替换手段难以满足组合服务 QoS 的要求，则需要进行流程重构。当前基于流程重构的容错处理方法大都分为 3 个步骤：1）确定流程重构的触发机制，最基本的思想就是当实际运行 QoS 与预期的 QoS 偏差达到一定阈值或百分比后流程重构被触发；2）确定重构片段，较简单的策略是重构组合服务中未执行的部分；3）执行重构。

表 1　服务组合 QoS 故障类型

故障类型	故障描述
通信故障	临时性故障，通信链路故障造成的消息丢失或服务提供方暂时失效
崩溃故障	组件服务较长一段时间内不可用
误差故障	由于需求变更、执行路径和环境变化引起实际服务 QoS 与预测值的偏差

组合服务执行成功的保证主要从设计时和运行时两方面考虑，现有服务冗余、流程重构等容错方法存在以下几点问题：

- 大多数研究工作更为侧重运行时的容错机制，在设计阶段缺乏对运行时容错机制的支持。
- 流程重构方案大都基于重新绑定一组服务，并不是真正意义上的重构，此外，重构方案缺乏可行性验证，易导致重构计划失败。
- 服务冗余、流程重构常常会影响到组合服务中执行完毕的部分，从而导致容错开销的增加，这也是当前大多数研究中不曾考虑而又不可忽略的一部分。
- 服务容错技术是在保证组合服务 QoS 和带来额外开销的一个权衡（trade-off）。因此，有必要研究容错技术带来的改进和开销之间的关系，并尽可能避免需要较大开销的容错行为。

针对以上问题并结合当前的研究现状，保证容错机制的有效运作，减少组合服务失效概率以及保证容错方法的低开销已经成为保障组合服务质量的迫切需求。

本文首先提出一种描述服务组合关系的方法，并基于该方法描述一个流程重构的案例；第 2 节针对流程重构方案常常不可行的问题，探讨如何搜索可行的重构方案，并确定最优的方案；第 3 节在分析容错机制可能带来的高额代价后，提出用于减少容错代价的组合服务功能拆分原则，组合服务的设计者可以遵循该原则对组合服务的功能进行拆分，并以此为基础确定组件服务的粒度和耦合度；第 4 节结合典型案例模拟说明拆分原则的有效性，并给出测试结果；最后对本文进行小结。

1　服务组合关系描述

服务组合可以集成现有的简单服务，按照顺序、并行、选择和循环等组合模式形成复杂服务，快速而又灵活地构建功能强大的新应用。为了能够简洁清晰地描述组件服务之间的关系，我们把服务关系定义为激活依赖、取消依赖、替换依赖、合并与拆分 5 类。为此，我们定义：

1）执行依赖关系：若服务 S_j 执行依赖于服务 S_i，则 S_j 只有在 S_i 执行完毕之后才能执行，记为 $Active(S_j)=S_i$。

2）取消依赖关系：若服务 S_j 取消依赖于服务 S_i，则当服务 S_i 执行失败后，S_j 的执行也将被取消，记为 $Abort(S_j)=S_i$。

3）替换依赖关系：若服务 S_j 替换依赖服务于 S_i，则当服务 S_i 失效后，S_j 的执行被激活，记为 $Alternative(S_j)=S_i$。

4）合并关系：若服务 S_i 和一组服务 $\{S_{i1},S_{i2},\cdots,S_{in}\}$ 有合并关系，则服务 S_i 的功能可以被该组服务 $\{S_{i1},S_{i2},\cdots,S_{in}\}$ 功能的合集所满足，记为 $Fun(S_i)\subseteq Fun(S_{i1})\vee Fun(S_{i2})\vee\cdots\vee Fun(S_{in})$。

5）拆分关系：若服务 S_i 和一组服务 $\{S_{i1},S_{i2},\cdots,S_{in}\}$ 有拆分关系，则服务 S_i 的功能可以被拆分为该组服务 $\{(S_{i1},S_{i2},\cdots,S_{in}\}$ 功能的合集，记为 $Fun(S_i)\supseteq Fun(S_{i1})\vee Fun(S_{i2})\vee\cdots\vee Fun(S_{in})$。

在用户出行服务组合中包含对用户行程进行规划（CRS）、购买飞机票（FB）、预订当地宾馆（HB）和租赁汽车（CB）、进行网上支付和送票 6 个子活动，如图 1 所示。其中 FB，HB 和 CB 执行依赖于 CRS，而 OP 执行依赖于 $FB\wedge HB\wedge CB$，TDU 执行依赖于 OP；FB，HB 和 CB 之间又相互存在取消依赖关系。

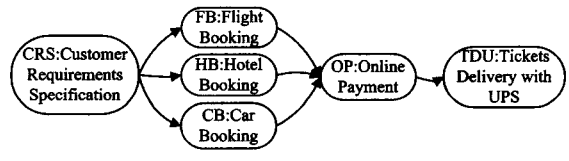

图 1　用户外出旅行服务组合

若用户难以搜索到合适的飞机票，或者搜寻到的飞机票难以与预订宾馆的地点与时间匹配，就需要选择一个备用服务重新调用。而当备选服务都不能满足用户的需求时，一套流程重构方案将被触发，如图 2 所示。这里，我们为用户搜寻到一个旅行代理服务（TA），该代理服务可以根据用户的出行

计划合理地为用户预订机票、宾馆和出租车，即 $Fun(TA) \supseteq Fun(FB) \vee Fun(HB) \vee Fun(CB)$。当然，这是一种简单的重构方案。用户也可以选择火车出行，但这将对整个流程产生较大的影响，需要重新预订宾馆和出租车，以及调整出行计划。

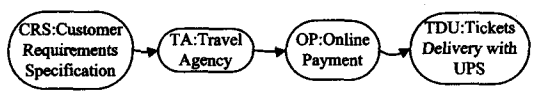

图 2　重构后的用户外出旅行服务组合

如前所述，目前的重构方案大都基于重新绑定一组组件服务改善全局 QoS，并非真正意义上的重构，如果没有备选的组件服务作为冗余，则重构失败；替换和重构都可能导致组合服务中已执行完成的部分的变更，带来额外的执行开销；此外，当前重构方案缺乏可行性验证，也容易导致重构计划的失败。举例来说，若不存在旅游代理服务(TA)，那么图 2 所示的流程重构方案也将失败。

2　流程重构方案

由于组合服务模型中没有指定组合服务功能的拆分方案，也未指定组件服务的具体提供者，因此在组合服务执行期间，运行环境将根据组合服务模型及用户需求来确定拆分方案并绑定具体的组件服务，也可以在运行过程中动态地调整拆分方案和重绑定组件服务。

流程重构只有在用户的功能需求有多种拆分方案时才能够实现。组合服务执行前，首先应根据候选服务的数量和质量，确定最优的拆分方案；次优的拆分方案将作为流程重构方案制定的参考依据，这在一定程度上保证了重构方案的可行性。下面详细阐述拆分方案的搜索方式并给出确定最优拆分方案的方法。

2.1　拆分方案的搜索

一个组合服务由基本服务和组合服务组成以完成某种新的功能，其中组合服务还可以再次进行分解，同样多个基本服务也可以合并成一个组合服务。可以说，组合服务的分解策略不同于以往的功能分解，没有必要把流程分解细化为更小更具体的子流程以便处理，而是应该把流程分解为有更多服务可以提供功能和非功能支撑的片段。此外，在拆分的同时也要考虑服务的执行开销、拆分后组合服务的结构等。为此，可以把组合服务表示为一棵功能树，树的根节点表示用户的功能需求，中间节点表示子功能，而叶节点表明该节点的功能如果再进行拆分则没有功能上可以匹配的服务。

假设组合服务的功能 Fun 可以被拆分为 n 个子功能，记为 $\{fun_1, fun_2, \cdots, fun_n\}$。设 $|S_i|=m$ 表示功能 fun_i 对应的候选服务集合的数量，记为 $S_i=\{s_{i1}, s_{i2}, \cdots, s_{im}\}(i=1,2,\cdots,n)$。若该功能拆分方案中，对 $\forall fun_i$，$|S_i| \neq \emptyset$，则称该功能拆分方案为可行的。如图 3 所示，用户的出行需求按照功能被拆分为 CRS, TA, OP 和 TDU 4 个子功能，而 TA 还可以被拆分为 FB, HB 和 CB 3 个子功能。拆分的原则是保证树中所有节点的功能都有可以匹配的服务，这就可以避免拆分出一些不可能或者不够实际来实现的组件服务。本案例中至少存在两种功能拆分方案，分别是 $\{CRS, TA, OP, TDU\}$ 和 $\{CRS, FB, HB, CB, OP, TDU\}$。接下来，就要确定最优的拆分方案。

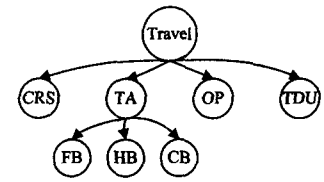

图 3　组合服务功能拆分树

2.2　最优拆分方案

在确定最优的功能拆分方案时我们考虑 4 方面因素：一是拆分后组合服务的结构，二是子功能所对应的候选服务的数量，三是候选服务的可靠性，最后是服务的执行代价。若一个子功能 fun_i 对应 m 个候选服务，且候选服务的可靠性分别为 $\{r_1, r_2, \cdots, r_m\}$，则该子功能能够通过重试与替换方法可靠完成的概率为 $R(fun_i)=1-(1-r_1)(1-r_2)\cdots(1-r_m)$。基于这样的前提，表 2 给出了可靠性关于顺序、并行、选择和循环 4 种组合模式下的聚合规则，其中，$p(fun_i)$ 表示在选择模式中子功能 fun_i 的相对执行率，$n(fun_i)$ 表示在循环模式中子功能 fun_i 循环执行的次数。

表 2　可靠性聚合规则

组合模式	聚合规则
顺序	$R(Fun)(\prod_1^n R(fun_i))$
并行	$R(Fun)(\prod_1^n R(fun_i))$
选择	$R(Fun)=\sum_1^n (p(fun_i)\times R(fun_i))$
循环	$R(Fun)=R(fun_i)^{n(funi)}, i\in\{1,2,\cdots,n\}$

从模式集成的角度来看，组合服务的功能是多个子功能通过不同的组合模式按照顺序结构相连接而形成的。因此，不妨将组合服务表示为 $Fun=\langle Fun_1, Fun_2, \cdots, Fun_k\rangle$ 为一顺序执行序列，其中 k 是组合服务中组合模式的个数，Fun_1，$Fun_2, \cdots, Fun_k$ 是顺询、并行、选择和循环 4 种组合模式中的一种。因此组合服务通过重试与替换方法能够获得的可靠性可由下式得到：

$$R(Fun)=\prod_{i=1}^{k} R(Fun_i) \tag{1}$$

通过该计算公式，就能够判断出哪种功能拆分方案的执行成功率最高，即该拆分方案可以获得最高的可靠性。那么，$(1-R(Fun))$ 即是该拆分方案可能发生流程重构的概率，如果不能进行重构，则组合服务执行失败。

如前所述，在确定最优拆分方案时，必须考虑组合服务的执行代价。使用 $C(Fun)$ 表示组合服务的执行代价。若组合服务按照功能被拆分为 $\{fun_1, fun_2, \cdots, fun_n\}$，则：

$$C(Fun)=\sum_{i=1}^{n} C(fun_i) \tag{2}$$

式中，$C(fun_i)=\sum_{j=1}^{m}(\prod_{k=1}^{j-1}(1-r_k))C(s_{ij})$。服务执行代价主要包括执行费用和执行时间，这已在文献[9]中做过详细的阐述，本文不做更多讨论。基于此，如果存在多种组合服务拆分方案，就可以通过下式取最小值(min)：

$$\min\left(\frac{C(Fun)}{R(Fun)}\right) \tag{3}$$

从而判定哪个拆分方案为最优，这样既保证了较高的可靠性，同时也使得执行开销较低。

3　服务组合功能拆分

3.1　服务容错代价

服务冗余、流程重构技术可以显著提高 QoS 质量并有效地压制组合服务的失效率，但是服务容错的代价常常超过其

能获得的 QoS 质量提升，其中原因包括：①流程重构计划确定的执行开销；②重新绑定、调用服务的执行开销和时间开销；③备选服务调用再次失败；④补偿操作[9]带来的开销；⑤替换、流程重构潜在的影响范围可能较大，某些已经执行完成的组件服务可能需要重新执行；⑥对实时性有要求的应用来说，服务替换、流程重构可能会带来较大的延时，重构的发生即意味着用户需求的破坏，从而导致业务的失败。文献[10，11]指出重构是一个权衡，一方面是在执行新计划时所产生的开销以满足非功能的需求，另一方面是容错所带来的改善。Berbner[10]在其重构策略上展开的实验表明：在不违反用户需求的条件下，流程重构降低了 25％的执行开销；Canfora[11]等人在其流程重构策略上的实验表明，流程重构导致组合服务的响应时间增加了 44％。

下面，依旧以图 1 用户旅游服务组合为例，讨论流程重构可能引发的高额代价问题。不妨假设机票预订服务不能满足需求，那么用户放弃飞机选择火车出行，为此，需要对组合服务流程进行重构，如图 4 所示。

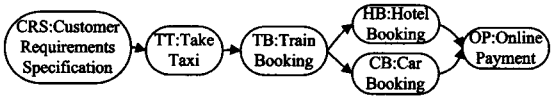

图 4　重构后的用户外出旅游服务组合

用户出行改乘火车，因此需要对整个活动进行调整：由于搭乘火车，那么已经规划好的出行计划(CRS)将被重新更改；火车票暂不能通过网上支付，用户需要搭乘出租车(TT)到达指定的购票地点，并购买火车票(TB)；搭乘火车到达目的地的时间常常会晚于搭乘飞机，因此需要更改已经预订的宾馆和汽车租赁的时间以减少不必要的开销，但也会承担更改操作所导致的补偿开销；最后，原流程中送票快递服务(TDU)被取消。用户整个出行活动的开销可能会降低，但是由于改乘火车，导致用户游玩时间减少，并且几乎影响了组合服务中所有的组件服务。

3.2　组合服务功能拆分原则

通过分析不难看出，若组件服务间存在过多的依赖、约束关系(紧耦合关系)，则会在服务替换或者流程重构期间影响其他组件服务，产生大量的重新执行、补偿等开销。从某种程度上讲，组合服务功能的拆分模式将对组合服务的性能、灵活性、重用性和组件服务的粒度产生较大影响，从而直接决定组合服务的质量。

服务容错属于一种“事后”补偿，如果在组合服务设计阶段，遵循一定的功能拆分原则，“事前”就在一定程度上降低拆分后组合服务失效的概率且降低组件服务间的耦合程度不失为一种可行、经济的方法。在对组合服务功能拆分时首先要把握好服务的粒度(service granularity)并尽可能满足组件服务间的松耦合。服务粒度是一种对交互和业务适合程度的一种度量。拆分过程中，组件服务粒度的大小会对组合服务的质量产生较大影响：如果粒度过小，则组件服务的重用性越高且增加了其灵活性，但过细的粒度会导致大量的细粒度服务，带来昂贵的开发和维护成本；粒度过大也可能带来几方面的问题，大粒度组件服务封装了复杂的功能，的确可以减少必须的服务请求数量和交互次数，但也会导致响应时间的增加且一次返回过多的数据，同时过大的粒度使得组件服务很难再被修改以满足不断变化的商业需求。因此，粒度的选择是一个平衡，也是面向服务架构的一部分。

当前，无论在工业界还是在学术界并不存在一个标准的组合服务功能拆分标准，这是因为拆分方案往往需要根据具体的需求来确定。但是，依旧可以从与之相关的多方面利弊权衡的设计实践中，总结出一些能够帮助正确选择组合服务拆分方案的经验法则。下面给出服务连续性(Service Continuity)和服务保护性(Service Protection)的定义，并以此指导设计功能拆分原则。

- 服务连续性(Service Continuity)：如果有一种拆分方案，在由它得到的组合服务结构中，某个组件服务的微小变动只会引起一个(或少量)组件的变化，那么该拆分方案就满足服务连续性。
- 服务保护性(Service Protection)：如果有一种拆分方案，在由它得到的组合服务结构中，某个组件服务在运行时出现异常不会影响到该组件服务之外(或最多只蔓延到少数周边服务)，那么该拆分方案就满足模块保护性。

基于以上的分析和介绍，包括第 2 小节中涉及到的一些拆分原则，我们主要从以下几个方面权衡考量：

拆分原则 1：组合服务功能拆分必须保证拆分出的所有子功能都有可匹配的服务。该原则是组合服务拆分最基本、必须遵守的原则。

拆分原则 2：拆分出的子功能必须有具体的意义。

拆分原则 3：组合服务功能拆分后的结构应该简单，尽量避免流程中出现选择和循环结构，从而减少流程中出现可能在运行阶段才能确定的执行路径。

拆分原则 4：组合服务功能拆分方案应该满足子功能所对应备选服务数量尽可能多的要求。也即功能拆分时尽量保证子功能不具有特定的业务意义和领域相关性，从而更具一般性。原则上讲，备选服务的数量越多，越能提高服务的可靠性。

拆分原则 5：如式(3)所示，选择组合服务功能拆分方案时需要考虑服务可靠性和执行开销两方面要求。

拆分原则 6：组合服务流程可以划分为核心流程和支撑流程两部分。前者执行相对稳定，较少有变更需求，需要较高的可靠性，因此，可以使用较大粒度的功能模块，从而减少组件服务的数量并降低服务间的交互和约束关系；后者是组合服务中独有的、体现差异化的部分，使用较细粒度的服务能够灵活地适应因环境和用户需求的变更带来的变化。

拆分原则 7：如果组件服务之间存在严格的调用延迟要求或者两阶段提交要求，那么就应该将它们合并为更大粒度的服务，并在服务内部实现调用和两阶段提交。

拆分原则 8：粗粒度服务的执行时间长，但是会减少服务交互次数；细粒度服务的执行时间短，但是会导致交互次数的增加，从而增加网络传输的时间。从中可以看出，服务粒度的大小将影响服务的响应时间，响应时间不宜过长也不宜过短，这里我们不给出服务响应时间大小的标准，但应根据需求确定合理的服务响应时间。

拆分原则 9：应该尽早执行可能相互产生影响的组件服务，把改变控制在某一区域内，避免循环依赖。

以上的拆分原则是我们对于当前服务组合研究工作的总结。可以看出，这些原则在研究和实践中会不断发展变化，有的在具体应用中可以发挥良好的效用，有的则会根据应用背

景的不同而变化，同时新的拆分原则也会不断出现。

4 模拟实验

本节通过模拟实验验证部分拆分原则的有效性。必须指出，本文所提出的拆分原则大多基于一些经验和法则，其带来的改善有的显而易见，有的则隐含了对组合服务可靠执行、降低容错代价的积极支持，很难通过实验的手段直接验证。模拟实验依旧基于用户旅游服务组合，如图 1 所示(PLAN1)。由于预订机票、宾馆和出租车之间存在严格的输入、输出关系，因此我们将之合并，用粒度更大旅游代理服务取代，如图 2 所示(PLAN2)。为简单起见，我们假设 CRS，OP 和 TDU 都能成功执行，其余服务的执行时间、执行费用以及可靠性分布在[1，100]，[1，100]和[0.1，1.0]内。我们认为 TA 的执行开销要略高于 FB，HB，CB 的执行开销总和，但 TA 成功执行的概率要比 FB，HB 和 CB 都执行成功的概率略高，这符合现实中服务使用的实际特征。

模拟试验环境为 PC 机，硬件为 Intel Core(TM)2 Duo CPU 2.66GH、4GB 内存；软件环境为 Windows XP Professional Service Pack 2，模拟程序利用 Python 语言编写。模拟实验采用服务容错代价(FTC)、服务容错代价(FTC)和执行代价(EC)比作为评价指标。

服务容错代价用于反映高额的容错代价，其中包括重新执行代价和补偿代价等。实验中，服务的可靠性初始为 0，并以步长 0.1 递增到 1。实验结果如图 5 所示。

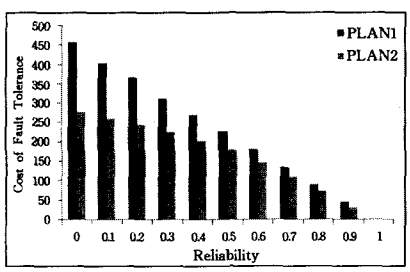

图 5 服务容错代价比较

服务容错代价(FTC)和执行代价(EC)比反映本文所提原则的有效性，若比值越低则表示拆分方案越优。实验结果如图 6 所示。

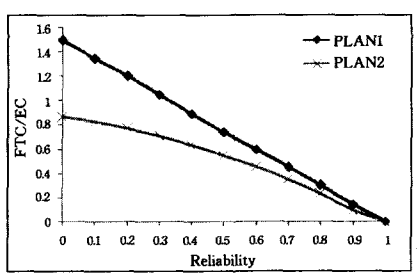

图 6 服务容错代价与执行代价比的比较

从实验数据中可以看出，若服务的可靠性较差则会带来较高额度的容错代价。PLAN2 的容错代价明显比 PLAN1 低，代价降低幅度最大可达 39%，且其服务容错和执行代价比也较 PLAN1 低，说明在该应用场景中使用粗粒度服务要好于细粒度服务。

结束语 服务冗余技术以及流程重构技术是服务组合应用中非常重要且有效的容错手段，本文针对流程重构策略缺乏可行性、容错方法开销过大以及难以满足实时性要求等问题提出一种有效的组合服务功能拆分方案的搜索方法以保证流程重构方案的可行性，并探讨总结若干提高组合服务执行成功率、降低失效容错代价的组合服务功能拆分原则。

下一步工作将主要从以下方面展开：(1)扩展并在实际应用中检验功能拆分原则的有效性；(2)以服务连续性和服务保护性为出发点，研究服务组合演化过程机理，其中包括演化需求发现机制、演化过程框架和演化过程系统状态一致性等问题。

参 考 文 献

[1] Ingham D B, Panzieri F, Shrivastava S K. Constructing Dependable Web Services[C]// Proceeding of Distributed Systems, LNCS 1752. 2000: 277-294

[2] Guo Huipeng, Huai Jinpeng, Li Huan, et al. ANGEL: Optimal Configuration for High Available Service Composition[C]// Proceeding of IEEE International Conference on Web Services (ICWS 2007). 2007: 280-287

[3] Tsai W T, Song W W, Chen Y N, et al. Dynamic system reconfiguration via service composition for dependable computing[C]// Proceeding of 12th Monterey Workshop. Laguna Beach, CA, Berlin: Springer-Verlag, 2005: 203-224

[4] Canfora G, Penta M D, Esposito R, et al. A framework for QoS-aware binding and re-binding of composite web services[J]. Journal of Systems and Software, 2008, 81(10): 1754-1769

[5] Zeng Liangzhao, Benatallah B, Ngu A H H, et al. QoS-Aware Middleware for Web services Composition[J]. Software Engineering, 2004, 30(5): 311-327

[6] May Chan K S, Bishop J. A Self-healing Composition Cycle for Web Services[C]// Proceeding of www2008. Beijing, China, 2008

[7] Peer, Joachim. A POP-Based Replanning Agent for Automatic Web Service Composition[C]// Proceeding of ESWC 2005. 2005: 47-61

[8] Colombo M, Nitto E D, Mauri M. SCENE: A service composition execution environment supporting dynamic changes disciplined through rules[C]// Proceeding of 4th International Conference on Service-Oriented Computing. Chicago, IL, 2006: 191-202

[9] 朱锐，郭长国，王怀民. 一种基于补偿代价的长事务调度算法. 软件学报，2009，20(3)：744-753

[10] Berbner R, Spahn M, Repp N, et al. Dynamic replanning of Web Service workflows[C]// Proceeding of IEEE International Conference on Digital Ecosystems and Technologies. Cairns, AUSTRALIA, 2007: 414-419

[11] Canfora G, Penta M D, Esposito R, et al. QoS-aware replanning of composite web services[C]// Proceeding of IEEE International Conference on Services Computing. Orlando, FL, 2005: 121-129