软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(12):3614−3634 [doi: 10.13328/j.cnki.jos.005313]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

SLA 感知的事务型组合服务容错方法*

张俊娜, 王尚广, 孙其博, 杨放春

(网络与交换技术国家重点实验室(北京邮电大学),北京 100876)

通讯作者: 王尚广, E-mail: sgwang@bupt.edu.cn

摘 要: 针对组合服务容错逻辑与执行逻辑不分离,以及容错过程易出现 SLA(service level agreement)违反的现状,提出一种 SLA 感知的事务型组合服务容错方法.该方法首先采用有限状态机建模组合服务执行过程,对其状态进行监控;其次,采用监控自动机监控执行过程中的 SLA 属性,确保不出现 SLA 违反;然后,对于补偿过程,采用改进的差分进化算法快速寻找最优恢复规划;最后,该方法与组合服务执行逻辑相分离,所以易于开发、维护和更新.基于真实数据集的实验结果验证了所提方法在故障处理时间与组合最优度方面优于其他方法,并且对不同故障规模适应良好.

关键词: 组合服务;容错;服务级别协议(SLA);差分进化算法;有限状态机

中图法分类号: TP311

中文引用格式: 张俊娜,王尚广,孙其博,杨放春.SLA 感知的事务型组合服务容错方法.软件学报,2018,29(12):3614−3634. http://www.jos.org.cn/1000-9825/5313.htm

英文引用格式: Zhang JN, Wang SG, Sun QB, Yang FC. SLA-Aware fault-tolerant approach for transactional composite service. Ruan Jian Xue Bao/Journal of Software, 2018,29(12):3614−3634 (in Chinese). http://www.jos.org.cn/1000-9825/5313.htm

SLA-Aware Fault-Tolerant Approach for Transactional Composite Service

ZHANG Jun-Na, WANG Shang-Guang, SUN Qi-Bo, YANG Fang-Chun

(State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications), Beijing 100876, China)

Abstract: Addressing the status quo that fault-tolerant logic of composite service is not separated from execution logic and service level agreement (SLA) violation appears frequently, this article proposes a SLA-based fault-tolerant approach for transactional composite services. Firstly, finite-state machine is adopted to model the execution process of the composite service and monitor the execution status. Secondly, monitoring automata is employed to monitor the SLA attributes during its execution to avoid SLA violation. Thirdly, an improved differential evolution algorithm is used to quickly determine the optimal recovery plan for the compensation process. Finally, a process is given to illustrate that as the approach is isolated from the execution logic of the composite service, it is easy to develop, maintain and update. The experimental results based on the real data sets show that the proposed approach is superior to other approaches in both the fault handling time and composition optimization. Meanwhile, the approach can deal with different fault scales.

Key words: composite service; fault tolerance; service level agreement (SLA); differential evolution algorithm; finite-state machine

基于服务的架构(service-oriented architecture,简称 SOA)是通过部署在不同网络节点上的服务(使用统一的接口)来实现,其可为自包含(self-contained)、松耦合(loose coupling)的分布式系统发展提供技术指导[1].随着 SOA 技术的不断成熟,很多企业把他们的全部或部分业务交由外部服务(external service)实现,从而降低开发周

* 基金项目: 国家自然科学基金(61472047, 61571066)
Foundation item: National Natural Science Foundation of China (61472047, 61571066)
收稿时间: 2017-03-10; 修改时间: 2017-05-18; 采用时间: 2017-06-08; jos 在线出版时间: 2018-01-09
CNKI 网络优先出版: 2018-01-11 17:24:30, http://kns.cnki.net/kcms/detail/11.2560.TP.20180111.1724.001.html

期与成本.因此,越来越多的应用程序(即组合服务,composite service)由外部服务(即组件服务,component service)采用服务组合技术组合而成[2].

比如:为了便于组织或者开发人员快速而轻松地在云环境中创建、部署和管理应用程序,目前很多公司推出了平台即服务(platform as a service,简称 PaaS)产品,比如,IBM 公司的基于 Cloud Foundry 的开放云应用平台 Bluemix,http://www.ibm.com/cloud-computing/cn/zh/platform/?lnk=bucl#infrastructure.通过 PaaS,组织或开发人员不仅可以把自身的 API(application program interface)即组件服务发布到云环境中以供其他组织使用,而且可以组合平台上已有的 API,以此得到增值服务,即组合服务.因此,通过 PaaS 组合组件服务可在很大程度上缩短应用程序的开发周期.但是因为 PaaS 上已有的组件服务由平台公司或者第三方提供,且其执行环境为尽力而为的网络,即不可靠的 Internet,所以由这些组件服务组合而成的组合服务在执行过程中就有可能因为网络阻塞、组件服务所在服务器宕机等问题而出现执行故障(failure).因此,在不可靠的互联网环境中如何保证组合服务的可靠执行[3],是一个亟待解决的问题.

软件可靠性保障技术主要有 4 种:故障预防(fault prevention)[4]、故障排除(fault removal)[5]、故障预测(fault forecasting)[6]、容错(fault tolerance)[7].组成组合服务的组件服务一般属于第三方,因此很难获得其内部组成结构与代码实现,那么试图采用故障预防和故障排除技术构造无故障的组合服务是不可行的.又因为组合服务的复杂执行环境以及组件服务的彼此独立和自由演化,组合服务执行阶段的故障是很难预测的,所以通过故障预测技术保障组合服务的执行也很困难.而容错技术是能在系统或部分组件出现故障时也能保障系统继续正确运行并完成其设计功能的一种技术,同时,容错技术通常在产生故障到最终导致系统失败(failure)的时间间隔中被采用,目的是在已经产生系统故障的情况下避免系统失败的出现(https://en.wikipedia.org/wiki/Fault_tolerance).因此,容错技术成为目前保证组合服务可靠执行的最有效方法.

组合服务的容错技术主要有两类:异常处理(exception handling)和事务处理(transaction)[8].前者为前向故障恢复技术(forward fault recovery),其试图修复故障,从而使组合服务继续执行;后者为后向故障恢复技术(backward fault recovery),其在故障不可被修复的情况下,保证组合服务终止在一致状态[9],多用于处理事务型组合服务出现的故障.事务型组合服务为一类特殊类型组合服务,其本身或者部分片段具有原子特性,即:要么全部执行,要么什么也不执行.对于事务型组合服务,当具有原子特性的组件服务出现故障时,必须停止组合服务的继续执行,首先回溯到一个可进行补偿的状态,然后从此状态开始为具有原子特性的组合服务或者片段重新选择组件服务,最后恢复组合服务地执行,即,需要采用事务处理容错技术.

异常处理和事务处理已被内置到 BPEL(Web services business process execution language)中,但构建 BPEL 容错逻辑是一件费时且易出错的工作,其原因有两个:首先,BPEL 的容错逻辑实现被嵌入到业务逻辑之中(即,容错逻辑仅在低层的语法层实现),使得容错逻辑的开发、维护和更新非常困难;其次,只有正确的故障处理逻辑才可以触发补偿(compensation)的执行,换句话说,组合服务能够终止在一致状态的工作留给了组合服务开发者.因此,开发独立于业务逻辑之外的、需要补偿操作时能终止在一致状态的容错逻辑,是当前容错方法需要解决的一个挑战.

除了确保组合服务正确执行之外,容错技术也需确保不违反服务使用者和服务提供者之间的服务级别协议(service level agreement,简称 SLA).SLA 的提出,是用于确保服务质量(quality of service,简称 QoS)[10],其是服务提供者和服务使用者之间正式协商的结果,也是存在于双方之间的合约(或为合约的一部分),是针对服务响应时间、可用性、吞吐率、优先权、责任和义务等方面达成的协议[11].SLA 是一种常用的指定精确交付条件(包括功能和非功能属性)的方法,通过其可确定哪一个服务必须或应交付.然而在实际应用过程中,SLA 仅指定服务使用者与提供者之间的高层接口(top-level interface)[12],即 SLA 监控工作也留给了组合服务开发者.因此,如何在组合服务的执行过程中监控 SLA,成为当前容错方法需要解决的另外一个挑战.

为了克服上述挑战,本文提出了一种 SLA 感知的组合服务容错方法(SLA-Aware fault-tolerant approach for service composition,简称 SLAFT).SLAFT 容错逻辑与组合服务执行逻辑相互分离,只有在产生故障的时候才会被触发;SLAFT 采用有限状态机(finite-state machine,简称 FSM)建模和监控组合服务执行状态,并确保产生故障

时能停留在一个一致的状态,方便进行下一步的补偿操作;采用监控自动机(monitoring automata,简称 MA)监控组合服务执行过程中的 SLA 属性,用于确保服务提供者与服务使用者之间商定的 SLA;进行补偿操作时,采用改进的差分进化算法(differential evolution algorithm,简称 DE)快速寻找最优恢复规划.

为了验证本文所提方法 SLAFT 的有效性,基于两个真实数据集:QWS(http://www.uoguelph.ca/~qmahmoud/qws/)和 WS-DREAM 的 QoSDataset2(http://www.wsdream.net/),我们进行了仿真实验:首先,采用 SLAFT 与其他两种方法进行了实验对比,结果表明,SLAFT 方法在故障处理时间与在组合最优度上均优于对比方法;然后,对影响仿真结果的参数(故障规模)进行了分析,实验结果验证了其对参数变化适应较好.

本文第 1 节给出本文的研究场景.第 2 节首先介绍 SLAFT 框架,然后详细描述框架中包含的主要模块(即服务状态监控、SLA 监控、恢复规划计算)的实现过程.第 3 节给出恢复规划计算中采用的改进差分进化算法的具体实现过程.第 4 节给出仿真实验,包括实验建立、实验对比、参数分析.第 5 节介绍相关工作.最后一节对全文进行总结,并展望下一步研究工作.

1 研究场景

首先给出本文一个研究场景:假设用户通过 PaaS 开发基于位置的服务推荐系统(service recommendation system,简称为 SRS)时,需要采用位置定位服务定位用户的位置,进而为其推荐附近的服务(比如餐饮).由于位置定位服务是一项较难开发的服务,且有现成的服务可供调用(如百度地图、谷歌地图等),所以用户可直接通过 PaaS 集成位置定位服务,以此来缩短 SRS 的开发周期.假设位置定位服务需要组合调用地图服务 Map(用户所在位置的区域地图调取)和定位服务 Loc(用于计算用户所在位置,并将其精准标识在地图上).因为不同公司所用的地图不同,且所采用的定位算法也不同,所以在调用到一个公司的地图服务后,必须调用其定位服务.也就是说,地图服务和定位服务具有原子特性,它们要么全部执行,要么什么也不执行.假设基于组合服务的服务推荐系统的工作流如图 1 所示,本流程实现语言采用 BPEL.

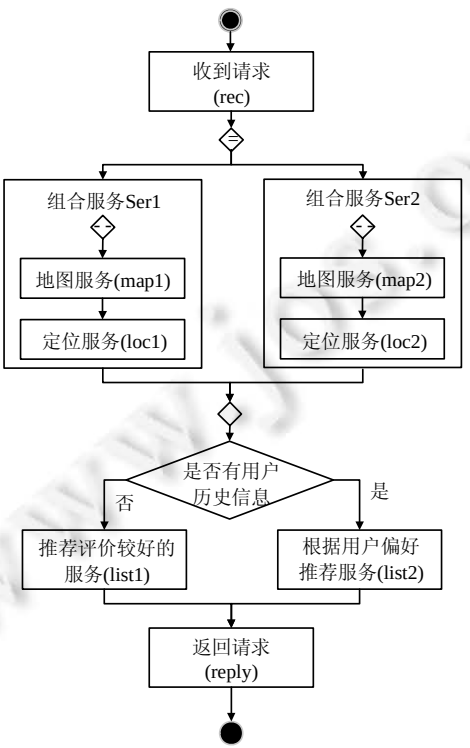


Fig.1 Workflow for service recommendation system based on service composition

图 1 基于组合服务的服务推荐系统工作流

BPEL[13]是实现组合现有 Web 服务的一个事实上的工业标准,其通过指定一个预先定义的可执行工作流来实现组合过程.通过 BPEL 的基本活动⟨receive⟩,⟨invoke⟩和⟨reply⟩分别实现接收消息、执行组件服务和返回执行结果,从一组消息中等待一个消息的出现,通过⟨pick⟩基本活动实现.组合服务的控制流通过⟨sequence⟩,⟨while⟩和⟨if⟩活动来分别实现顺序、循环和条件结构,还可以通过⟨flow⟩实现并行结构.⟨scope⟩可以包含其他活动,且能被关联到一个补偿句柄(compensation handler)中,该句柄指定了补偿操作需要执行的活动区间.

图 1 中,当收到用户的请求 Rec 时,启用一个⟨pick⟩活动(采用符号◈表示)用来等待两个服务(组合服务 Ser1,组合服务 Ser2)中的一个服务做出响应.如果 Ser1 首先做出了响应,一个⟨sequence⟩活动(采用符号◈表示)被调用,地图服务 Map1 和定位服务 Loc1 将被顺序调用.如果 Ser2 首先做出响应的情形与此类似.组合服务执行过后,一个⟨if⟩活动(采用符号◇表示)用来判断是否有用户的历史信息:如果没有,就推荐附近评价较好的服务(list1);如果有,就依据用户历史信息推荐符合其偏好的服务(list2),最后把推荐列表返回给用户(reply).

现在,让我们考虑这样的场景:当收到用户的请求时,首先得到了 Ser1 的响应,工作流就对此进行了调用(即需要首先调用组件服务 Map1,然后调用组件服务 Loc1).但在成功调用 Map1 后,发现 Loc1 不可用.经典的容错策略是点恢复策略(point recovery strategy),即:对 Loc1 进行重试,或者将其替换为实现同样功能的服务[14].但是,本文主要考虑如下场景:如果 Loc1 确实不可用(重试策略失效),且不存在替换服务时需要采用的容错策略.出现此类故障时就需要采用另外一种重要的容错策略,即流恢复策略(workflow recovery strategy).其需要指定如何对现有工作流进行补偿(compensation),并回溯到以前的状态,且能找到一个替代执行路径,确保组合服务的继续执行.一个好的容错策略还需要保证不违反服务使用者与服务提供者事先商定的 SLA.所以,针对流恢复策略需要解决如下问题:(1) 如何对组合服务的执行状态进行建模;(2) 如何在容错的过程中不违反 SLA;(3) 如何寻找最优替代执行路径.

2 SLA 感知的事务型组合服务容错方法

针对上一节的问题,我们提出了 SLA 感知的事务型组合服务容错方法 SLAFT.SLAFT 是在正常的组合服务执行逻辑之外加入监控逻辑,同时监控组合服务的执行状态和执行过程中的 SLA 属性值.当执行状态正常和不出现 SLA 违反时,监控逻辑只是起到监控作用;一旦出现错误状态和 SLA 违反,就触发相应的补偿操作.SLAFT 方法框架如图 2 所示,其主要包含 3 个模块:模块 1 为服务状态监控模块,其采用有限状态机建模组合服务的执行状态,并对执行过程中的状态进行监控,一旦进入故障状态,将触发模块 3 恢复规划计算的执行(详细分析见第 2.1 节);模块 2 为 SLA 监控,其采用监控自动机对执行过程中的 SLA 属性进行监控,如果出现 SLA 违反,也将触发模块 3 的执行(详细分析见第 2.2 节);模块 3 为恢复规划计算,其针对出现的故障状态和 SLA 属性违反,采用改进差分进化算法计算恢复规划,然后交由组合服务执行引擎执行,使得组合服务能继续执行和不出现 SLA 违反(详细分析见第 2.3 节).

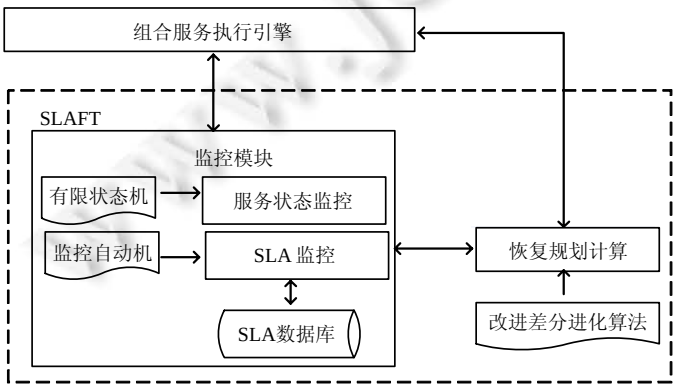


Fig.2 Framework of SLAFT

图 2 SLAFT 方法框架

2.1 服务状态监控

组合服务的正常执行是从起始状态开始,经过若干个中间状态,最终到达结束状态.但也可能因为某些原因,使得组合服务停留在某个中间状态无法继续执行.为了感知这些状态,保证组合服务的正常执行,必须对组合服务的执行状态进行监控.本文采用有限状态机建模组合服务执行状态,并对其执行过程中的状态进行监控.本节首先给出组合服务的定义,然后再给出有限状态机的定义,最后叙述如何采用有限状态机建模和监控组合服务状态.

定义 1(组合服务). 组合服务 CS 是一个三元组(Var,V_0,P_0),其中,Var 为一个有限变量集合,V_0 为集合中每个变量在其值域内的初始赋值,P_0 为初始赋值后组合服务工作流程.

定义 1 中,Var 表示组合服务包含的抽象组件服务集合,而 V_0 是对各个抽象组件服务的初始赋值,P_0 为组件服务的执行顺序(如并行、串行、选择等).

定义 2(有限状态机). FSM 是一个五元组(Σ,S,s_0,δ,f),其中,Σ是输入符号的非空有限集合;S 是非空的有限状态集合;s_0 是 S 中的一个特殊状态,起始状态;f 是 S 中另外一个特殊状态,结束状态;δ是一个从空间 $S\times\Sigma$到 S 的映射函数,即$\delta:S\times\Sigma\to S$.

本文采用 FSM 建模含有 n 个组件服务的组合服务的执行过程,每个状态 s 的形式为(V,P).$s_0=(\{Var_1\to\phi,Var_2\to\phi,\ldots\},P_0)$,$s_1=(\{Var_1\to Ser_1\phi,Var_2\to\phi,\ldots\},P_1)$,…,$f=(\{Var_1\to Ser_1,\ldots,Var_n\to Ser_n,\ldots\},P_n)$.其中,$Var_i\to\phi$代表 Var_i 没有被定义,$Ser_1,\ldots,Ser_n\in\Sigma$.状态 s 代表的实际意义是:随着组合服务的执行,对抽象组件服务的逐步实例化.P 为组合服务的执行流程,可能随着执行过程有所改变,也可能一直保持不变,这跟抽象组件服务的具体实例化过程相关.

本文中,我们假设 FSM 中存在一个错误活动(输入符号)$err(err\in\Sigma)$,其是指组合服务执行过程中遇到的一些阻碍其正常执行的问题,比如组件服务不可用、SLA 违反等.且错误活动 err 的输入能使得处在任何状态的 FSM 转换到错误状态 $s_{err}(s_{err}\in\Sigma)$.也就是说,$\forall s\in S|\{s_{err}\},(s,err,s_{err})\in\delta$.

对于一个 FSM $F=(\Sigma,S,s_0,\delta,f)$,我们采用 $s\xrightarrow{a}s'$ 的形式表示$(s,a,s')\in\delta$.给定一个状态$s\in S$,$enable(s)$代表从状态 s 经过了一步转换,即 $enable(s)=\{s'\mid s'\in S\wedge a\in\sum\wedge s\xrightarrow{a}s'\in\delta\}$ .F 的完整执行π是一个有限的状态和活动的交替序列$\langle s_0,a_1,s_1,\ldots,s_{n-1},a_n,f\rangle$,其中,$s_0,s_1,\ldots,s_{n-1},f\in S,a_1,a_2,\ldots,a_n\in\Sigma,s_0$ 和 f 分别代表 FSM 的起始状态和结束状态.对于所有的 $0\leqslant i\leqslant n-2$ 有 $s_i\xrightarrow{a_{i+1}}s_{i+1}$,且 $s_{n-1}\xrightarrow{a_n}f$.我们采用形式 $s_0\xrightarrow{a_1}s_1\xrightarrow{a_2}\ldots s_{n-1}\xrightarrow{a_n}f$ 代表完整执行π.执行π的前缀(prefix)为π的一个片段(fragment),其从开始状态 s_0 开始,在状态 $s_i(i\leqslant n-2)$结束.执行π的后缀(suffix)也为π的一个片段,但其从状态 $s_i(i\leqslant n-2)$开始,结束于结束状态 f(一个状态 $s\in S$ 是一个结束状态,当其不存在状态 $s'\in S$ 和活动 $a\in\Sigma$使得 $s\xrightarrow{a}s'$;否则,称其为非结束状态).

对于场景 SRS 的 FSM $F(SRS)$如图 3 所示(点划线代表的语义将在第 2.3 节中解释).场景 SRS 的 BPEL 活动语义是基于文献[15],比如:对于状态 s_9的条件活动,当条件为真(拥有用户历史信息)时,活动 List2 被执行;而当为假(不拥有用户历史信息)时,活动 List1 被执行.也就是说,从状态 s_9 拥有两个可能的转移状态 s_{10} 和 s_{11}.但条件语句要么为真要么为假,一旦其值确定,只能转移到一个状态.与条件活动类似,〈pick〉活动也拥有两个可能转移状态,但真正执行过程中,也只能转移到其中一个状态.图 3 中,每个状态均可以转移到状态 err,但为了增加可读性,只画出了状态 s_7 的错误转移状态.

如果一个状态具有备选的执行方案,称其为迁移状态(migration state),也就是说,其可从当前的执行迁移到另外一个执行.迁移状态包括那些可以调用〈flow〉活动、〈pick〉活动或者非幂等性(non-idempotent)服务调用(如果对于任何相同输入参数的调用均可得到相同的结果,则称这个服务调用是幂等的;反之为非幂等性服务调用)的状态.在图 3 中,有效的从状态 s_7 出发的迁移状态为 s_5,s_3 和 s_2.

BPEL 语言自身提供补偿机制(compensation mechanism),其采用特定的编程方式来撤销已经完成活动产生的结果.但该补偿机制具有局限性,即,很难保证补偿过后系统所在的状态能够满足系统的功能性要求,之所以产生这种结果,是因为该机制不能确定补偿后软件系统所处的状态.为了克服这个局限性,我们分析了 BPEL 的每个活动可以产生的两种变化,即内部变化(internal change)和外部变化(external change):内部变化使得系统从

状态 s 改变到状态 s',而外部变化改变组件服务的状态.

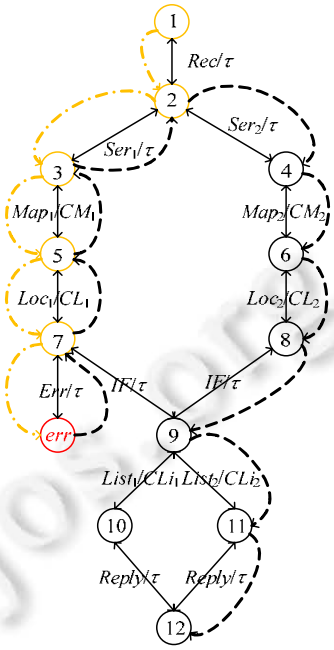


Fig.3 Finite-State machine for service recommendation system based on service composition

图 3 基于组合服务的服务推荐系统的有限状态机

为了撤销内部变化,必须把一个活动执行前的状态值存储下来,在恢复的过程中,通过把当前的状态值 s'恢复成存储的状态 s,起到了撤销内部变化的作用.外部变化仅可通过 BPEL 的通信活动(communication activity)来实现,比如⟨*receive*⟩,⟨*invoke*⟩和⟨*reply*⟩.因为通信活动是唯一能够与组件服务通信的活动,所以为了撤销外部变化,用户需要为每个通信活动 a 指定一个补偿句柄.其指定一个操作 a_{bak},通过执行 a_{bak} 可以撤销活动 a 产生的结果,回到 a 执行前的状态.例如:如图 4 所示,是为图 3 中操作 Loc_1 指定的补偿句柄,其通过调用操作 CL_1 来取消调用定位服务 Loc_1 后所产生的结果,以此来补偿因调用操作 Loc_1 所产生的外部变化.

```
⟨pick ext: isControllable=true…⟩
  …
    ⟨invoke operation="Loc1"…⟩
      ⟨compensationHander⟩
        ⟨invoke operation="CL1"…⟩
        ⟨/invoke⟩
      ⟨/compensattionHander⟩
    ⟨/invoke⟩
  …
⟨/pick⟩
```

Fig.4 Compensation for invoked location service

图 4 对定位服务调用地补偿

对于每个活动 a,都有一个相应的后向行动(backward action)a_{bak},其通过存储的状态值来逆转内部变化和基于补偿句柄来逆转外部变化,以此来返回到执行 a 之前的状态.一个 $F(SRS)$后向行动的例子为 $s_7 \xrightarrow{CL_1} s_5$,该行动使组合服务的状态由 $s_7=(P,V)$补偿为 $s_5=(P',V')$,其补偿过程为把状态值 V 改变为保存的状态值 V',同时,通过补偿句柄 CL_1 撤销对定位服务的调用.我们采用符号 τ来表示一类后向行动,其补偿过程不需要做什么实际操作.非后向的行动为前向行动(forward action),比如 $s_5 \xrightarrow{Loc_1} s_7$.给定一对前向行动 a_f 和后向行动 a_b,即 $s \xrightarrow{a_f} s'$ 和

$s' \xrightarrow{a_b} s$,我们合并为一个符号 $s \xleftrightarrow{a_f / a_b} s'$,比如 $s_5 \xleftrightarrow{Loc_1 / CL_1} s_7$.我们采用符号$\Sigma_F$和$\Sigma_B$代表所有可能的前向行动和后向行动的集合.

2.2 SLA监控

为了不违反服务提供者和服务使用者事先商定的 SLA,必须对组合服务执行过程中的 SLA 属性进行监控.我们采用确定性有限自动机(deterministic finite automata)监控 SLA 属性,本文称其为监控自动机.首先给出监控自动机的定义如下:

定义 3(监控自动机). 监控自动机 MA 为五元组(Q,Q_0,Σ,δ,F),其中,Q 为状态集合,$Q_0 \subseteq Q$ 为初始状态集合,Σ 为通用活动集合,$\delta: Q \times \Sigma \times Q$ 为转换关系,$F \subseteq Q$ 为可接受状态集合.

我们采用符号Σ^*表示一组有限活动序列.给定监控自动机 A,序列活动 $a_1,a_2,\ldots,a_n \in \Sigma^*$是被 A 接受的,如果在 A 中存在一条路径 $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots q_{n-1} \xrightarrow{a_n} q_n$,其中,$q_0 \in Q_0, q_i \in F (1 \leqslant i \leqslant n), a_i \in \Sigma$,而且对于$\forall 1 \leqslant i \leqslant n$,$(q_{i-1},a_i,q_i) \in \delta$.一个执行 $\pi = s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots s_{n-1} \xrightarrow{a_n} s_n$ 是可被 A 接受的,如果序列活动 $a_1,a_2,\ldots,a_n$ 可被 A 接受;反之是被 A 拒绝的.对于可被 A 接受的活动序列,我们采用符号 $L(A)$表示.

给定一个 SLA 属性 P_{SLA},$A(P_{SLA})$代表其的监控自动机.一个执行是被 $A(P_{SLA})$接受的,如果其违反了设定的 SLA 属性.也就是说,组合服务的执行过程中,如果出现违反了设定的 SLA 属性,就会触发其的相应监控自动机.如果不违反,就不会触发.例如:给定一个属性 P_1"在 SRS 中,组件服务的不可用绝不会发生",当出现组件服务不可用时,就会触发错误活动 err.给定一系列 SLA 属性值,我们定义一个组合服务 CS 的监控自动机为 M_{CS}=$\langle A(P_1),\ldots,A(P_N)\rangle$,其中,$P_i (1 \leqslant i \leqslant N)$为 SLA 属性.给定 $L(CS)$中的一个执行π,如果π被所有的自动机 A 拒绝,那么π满足 M_{CS};其他情况下,π违反 M_{CS}.也就是说,当执行过程不违反 SLA 时,MA 只起到监控作用;而违反 SLA 时,MA 被触发,该信息将会被组合服务执行引擎和恢复规划计算模块收到,将会触发相应的恢复规划计算,以确保不产生违反 SLA 的执行结果.

2.3 恢复规划计算

图 3 中,采用点划线(-·→)表示一次从状态 s_1 到状态 s_7 的执行.在状态 s_5,如果服务 Loc_1 能被成功调用,组合服务状态将转移到 s_7.而如果此时因为网络或服务器故障,Loc_1 不能被成功调用,组合服务将进入错误状态 err,服务状态监控模块应立刻发现此类阻碍组合服务正常执行的故障.为了从故障中恢复,需要计算恢复规划.该恢复规划是一个指导方针,其可使组合服务从当前错误状态补偿到一个迁移状态(采用后向行动),然后选择一个可以到达结束状态的替代路径(采用前向行动),以此来确保组合服务的成功执行.图 3 中,场景 SRS 的一个可能的恢复规划 r_{SRS} 使用划线(- - →)表示,其首先采用后向行动从 err 状态补偿到一个转移状态 s_2,然后采用前向行动从状态 s_2 到状态 s_{12},即,组合服务成功执行完成.

定义 4(恢复规划).恢复规划 r 是一个执行 $s_{err} \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_m} s_m \xrightarrow{a_{m+1}} s_{m+1} \xrightarrow{a_{m+2}} \ldots \xrightarrow{a_n} f$.其中,$f$ 为结束状态,s_m 为迁移状态,且满足:(1) $0 \leqslant m \leqslant n$;(2) $\forall i \leqslant m, a_i \in \Sigma_B$;(3) $\forall j > m, a_i \in \Sigma_F$.

恢复规划 r 的前缀为 r 的从状态 s_{err} 开始,到状态 $s_i (i \leqslant n)$结束的一个执行片段,其后缀为 r 的从状态 s_i 开始,至结束状态 f 结束的一个执行片段.

针对场景 SRS,在转移状态 s_2,根据恢复规划 r_{SRS},组合服务需要继续转换到状态 s_4.但是,根据⟨pick⟩活动的语义,其应该选择执行最先做出响应的服务.如果这样,就违反了按照恢复规划执行组合服务的语义.因此,根据 BPEL 扩展属性特征,为⟨pick⟩活动扩展一个 isControllable 属性[16],其允许用户通过恢复模块来指定可控制的活动.在我们的场景中,如图 3 所示,对于状态 s_2,通过设置其的 isControllable 属性值为真(如图 4 所示),使得⟨pick⟩活动选择的转换的状态可控制,因此,可按照恢复规划执行组合服务.除了⟨pick⟩活动,在某些情况下,用户还需要指定⟨flow⟩和⟨if⟩活动的可控性.如果⟨flow⟩活动被指定为可控,那么执行引擎将忽略并发语义,根据恢复规划按照顺序语义执行组合服务.如果⟨if⟩活动被指定为可控,那么执行引擎将忽略判断语句的值,直接按照恢复规划指定的分支执行组合服务.假如在状态 s_2 的⟨flow⟩活动的 isControllable 属性设置为真,而在状态 s_9 的⟨if⟩活动的 isControllable 属性设置为假,那么,恢复过程将持续到状态 s_9,因为⟨if⟩活动不可控,恢复过程结束,执行引擎将按

照正常的执行语义执行余下的组件服务,即,⟨if⟩活动将转移到状态 s_{10} 或状态 s_{11},取决于是否有用户历史信息.

我们称一个执行的最大可控部分为可控前缀,对于 SRS,可控前缀从状态 s_{err} 开始到状态 s_9 结束.对于状态 s_9 本身而言,其为不可控的状态,但这个状态结束了可控前缀.同样的,我们称一个从不可控的状态开始的执行部分为不可控后缀,尽管不可控后缀(即从状态 s_9 到状态 s_{11})不是恢复过程的一部分,但其提供了一个从不可控状态开始的执行参考.在计算恢复规划过程中,不可控后缀可以有助于找到一个以不可控状态结束的恢复规划和一个从此不可控状态开始的最好的执行后缀.因此,在恢复过程结束和正常执行逻辑开始时,组合服务具有较高的机会可以同时符合功能性和非功能性方面的需求.

总的来说,恢复规划的计算可分为 3 个部分:(1) 首先需要从状态 s_{err} 开始,补偿到一个迁移状态;(2) 从此迁移状态开始,计算一段到不可控状态的最优路径(不包括出故障的路径);(3) 计算最优不可控后缀.第 1 部分和第 2 部分得到恢复规划的可控前缀,第 3 部分得到不可控后缀.第 1 部分计算比较简单,只需要恢复到迁移状态.恢复过程需要撤销从状态 s_{err} 到迁移状态产生的内部变化和外部变化.内部变化的撤销只需要把现有状态恢复为保存的状态.对于 SRS,如图 3 所示,就是把当前状态 s_{err} 恢复到状态 s_2.外部变化需要通过执行补偿句柄指定的操作 a_{bak},以此撤销已执行活动产生的结果.对于 SRS,需要执行操作 CL_1,CM_1 和 τ来撤销已执行活动 Loc_1,Map_1 和 Ser_1 产生的结果.第 2 部分和第 3 部分计算过程较类似,均为组合服务片段的抽象组件服务重新寻找最优组件服务的过程.第 2 部分与第 3 部分的唯一区别就是第 2 部分的计算过程中需要去除出故障的路径分支.

为组合服务片段重新选择最优组件服务与对整个组合服务选择最优组件服务的过程类似,均需要对所有的候选组件服务根据其 SLA 属性进行组合,在满足 SLA 约束的前提下,寻找最优组合结果.比如:对于顺序结构的组合请求,如果其包含 m 个抽象组件、每个抽象组件有 l 个候选组件服务,那么将会有 l^m 种组合方法,随着抽象组件个数的增加,组合方法的数目会呈现指数级别增长.对仅存在一条单一路径结构尚且如此,对存在多条路径的选择结构,其组合方法会更多.所以,随着抽象组件数目的增加,采用穷举搜索算法将不可行.文献[17]分析了对组合服务寻找最优组件服务的过程可以建模为多维多选择背包问题(multichoice multidimensional knapsack problem,简称 MMKP),且文献[18]已证明 MMKP 为 NP-hard 问题,所以只能采用一些方法寻找近似最优解.

针对为组合服务寻找最优组件服务的问题,已有许多学者采用不同的算法予以求解.有的学者采用穷举搜索算法,但其仅适用抽象组件数目较少的组合服务.因为随着抽象组件数目的增加,其组合数目呈现指数级别增长,采用穷举搜索算法求解的时间将很长,或者根本无法求解.有的学者采用粒子群算法[19]或者遗传算法[16],但这两类算法的参数较多,且不同的参数设置将产生不同的求解结果,即结果的优劣在很大程度上依赖于参数设置.近年来,差分进化算法因其具有参数较少、对求解结果影响较小的优势,在多个领域得到成功应用[20−22].所以,我们就采用改进的差分进化算法用于求解组合服务寻找最优组件服务的问题,具体实现过程见第 3 节,第 4 节的实验结果也验证了改进的差分进化算法优于对比算法.

3 基于改进差分进化算法的最优恢复规划

我们的主要工作是基于改进的差分进化算法寻找最优恢复规划,所以本节首先介绍经典差分进化算法,然后再描述改进的地方,最后说明适应度函数构造及编码方法.

3.1 差分进化算法

1997 年,Storn 和 Price 在其他进化算法(evolution algorithm)的基础上提出了差分进化算法 DE[23].DE 是一种简单但强大的随机搜索技术,可以对非线性、不可微、连续空间函数进行最小化,其具有易用性、稳健性和强大的全局寻优能力.传统差分进化算法实现过程如下:

1) 种群初始化

在解空间中,随机、均匀地产生 NP 个个体(种群规模),每个个体由 D(解的维数)个染色体组成,作为第 0 代种群,标记为

$$X(0)=\{X_1(0),X_2(0),\ldots,X_{NP}(0)\} \tag{1}$$

其中,第 i 个染色体 $X_i(0)(i=1,2,\ldots,NP)$的第 j 维取值方式为

$$x_{i,j}(0)=L_j+rand(0,1)(U_j-L_j),i=1,2,\ldots,NP;j=1,2,\ldots,D \tag{2}$$

其中,$[L_j,U_j]$为第 j 维的取值范围,$rand(0,1)$为介于 0 和 1 之间的均匀分布随机数.

2) 变异

经过 $g-1$ 次迭代后,可得到第 $g-1$ 代种群为 $X(g-1)=\{X_1(g-1),X_2(g-1),\ldots,X_{NP}(g-1)\}$.根据第 $g-1$ 代种群的个体可以得到第 g 代种群的个体.首先进行如下变异操作得到中间体 $H_i(g)$:

$$H_i(g)=X_{p_1}(g-1)+F(X_{p_2}(g-1)-X_{p_3}(g-1)),1\leqslant i\leqslant NP \tag{3}$$

其中, $X_{p_1}(g-1),X_{p_2}(g-1),X_{p_3}(g-1)$ 为从第 $g-1$ 代种群中随机选择的 3 个个体,且 $p_1\neq p_2\neq p_3\neq i$; $\Delta_{p_2,p_3}(g-1)=X_{p_2}(g-1)-X_{p_3}(g-1)$ 为差分向量;F 为缩放因子,取值范围为[0,1],用于控制差分向量的影响力.

如果变异过后的向量个体不可行(也就是说落到了搜索空间以外),可采用文献[24]中提出的修补算子进行修补计算.

3) 交叉

为了提高种群的多样性,DE 算法引入了交叉操作.与其他进化算法中基于多个来自父代中的基准个体交换染色体的交叉操作不同,DE 中的交叉操作采用基准个体与变异个体进行交叉操作,具体如下:

$$v_{i,j}(g)=\begin{cases}h_{i,j}(g), & rand(0,1)\leqslant CR\\ x_{i,j}(g-1), & \text{其他}\end{cases},i=1,2,...,NP;\ j=1,2,...,NP \tag{4}$$

其中,$CR\in[0,1]$为交叉概率,$rand(0,1)$为介于 0 和 1 之间的均匀分布随机数.

4) 选择

根据适应度函数 f 选择 $V_i(g)$或 $X_i(g-1)$作为 $X_i(g)$:

$$X_i(g)=\begin{cases}V_i(g), & \text{if } f(V_i(g))<f(X_i(g-1))\\ X_i(g-1), & \text{其他}\end{cases},i=1,2,...,NP \tag{5}$$

DE 的选择操作使得子代的适应度值总是好于父代的适应度值,从而使种群始终向最优解的位置进化,并逐步聚焦到最优解位置.

3.2 改进差分进化算法

DE 需要设置的参数有种群规模 NP、缩放因子 F 和交叉概率 CR.而本质上,F 的取值决定差分向量的影响力,即确定搜索步长;交叉概率 CR 确定多样性,避免出现早熟现象.所以,F 和 CR 的设置对算法最终性能影响较大[25].因此,本文将通过对 F 和 CR 的设置来改进传统的 DE 算法.关于对 F 和 CR 的设置方法,可以分为 3 个类别:常数、随机、适应(包括自适应).本文是采用 DE 寻找最优恢复规划,而其会因组合服务的具体执行情况而有所不同,因此把 F 和 CR 设置为常数或者随机均不能得到较优结果,所以本文采用自适应的方法设置 F 和 CR 的值.改进过后的差分进化算法的具体实现过程见算法 1.

算法 1. 改进差分进化算法 IDE.

输入: F_l, F_u, CR_l, CR_u, G_m, NP, D;

输出: 最优解 *best_vector*.

1. X=在解空间中随机生成一个 NP 行 D 列的矩阵; //初始化种群
2. $H=zeros(NP,D)$; //用于放置变异过后的中间值
3. $V=zeros(NP,D)$; //用于放置交叉过后的中间值
4. $X\_next=zeros(NP,D)$; //用于放置进化过后的下一代值
5. G=1; //初始化进化代数,用于对进化次数计数
6. **while** ($G\leqslant G_m$) **or** (可接受的满意解) //G_m 为设置的最大循环次数

   ------变异操作------
7. $H=Mutation(X)$;

   ------交叉操作------

```
8.    V=Crossover(X,H);
9.    ------选择操作------
10.   for i=1 to NP
11.     if Fitness(V(i))<Fitness(X(i))
12.       X_next=V(i);
13.     else
14.       X_next=X(i);
15.     endif
16.   endfor
      ------找出第 G 代中适应度值最小的个体------
17.   for i=1 to NP
18.     fit_value(i)=Fitness(X_next(i));
19.   endfor
20.   (value_min,pos_min)=min(fit_value);
21.   Gmin(G)=value_min;                    //得到第 G 代的最小适应度值
22.   best_X(G)=X_next(pos_min);            //保存第 G 代中得到最小适应度值的个体
23.   X=X_next;                             //得到进化过后的下一代种群
24.   G=G+1;
25. Endwhile
26. (value_min,pos_min)=min(Gmin);
27. best_value=value_min;                   //进化的所有代中的最小适应度值
28. best_vector=X_next(pos_min);            //取得最小适应度值的个体,即最优值
```

算法 1 中,F_l,F_u 是为缩放因子设置的下限和上限,CR_l,CR_u 是为交叉概率设置下限和上限,G_m 是设置的最大循环次数.算法第 1 行~第 5 行为算法的初始化操作,包括种群初始化、进化过程中的中间值和进化代数初始值设置.算法第 6 行~第 25 行为差分进化算法的循环迭代过程,每次循环均经历变异操作(算法第 7 行)、交叉操作(算法第 8 行)、选择操作(算法第 10 行~第 16 行)以及保留此次循环中适应度值最小的个体(算法第 17 行~第 24 行).而结束循环的条件为超过最大迭代次数或者得到用户可接受满意解.算法中,函数 *Fitness* 用于计算个体的适应度值,其构造过程和具体实现过程见第 3.3 节.函数 *Mutation* 为变异操作,函数 *Crossover* 为交叉操作,这两个函数的具体实现过程如下.

1) 自适应的变异操作

对第 g 代种群进行变异操作时,随机选择的 3 个个体按照其适应度值进行从优到劣地排序,假设为 $X_b(g-1)$,$X_m(g-1)$,$X_w(g-1)$,其对应的适应度值 $f(X_b(g-1))\leqslant f(X_m(g-1))\leqslant f(X_w(g-1))$(仅考虑函数的最小化问题,对于最大化问题,可进行乘以–1 操作后转化为最小化问题),变异操作修改如下:

$$H_i(g)=X_b(g-1)+F_i(X_m(g-1)-X_w(g-1)) \tag{6}$$

通过采用公式(6),各代的每个个体均在随机选出的、上一代的 3 个个体中适应度最好的个体附近进行变异,可加快收敛速度.又因为每次随机选出的适应度最好的个体不是全部相同,所以还可以增加种群多样性,避免陷入局部最优.

同时,为了平衡全局搜索与局部搜索之间的矛盾,F 的取值根据生成差分向量的两个个体进行自适应变化,其取值如下:

$$F_i = F_l + (F_u - F_l)\frac{f(X_m(g-1)) - f(X_b(g-1))}{f(X_w(g-1)) - f(X_b(g-1))} \tag{7}$$

其中,F_l 和 F_u 为 F 设定的上限和下限,比如可取 F_l=0.1,F_u=0.9.采用公式(7)自适应改变 F_i 值的原因为:当随机选

取3个个体的适应度值差别较大(一般为迭代初期)时,采用较大的缩放算子用于提高全局搜索能力;适应度值差别较小(一般为迭代末期)时,采用较小的缩放算子,防止错过最优解,即用于提高局部搜索能力.自适应的变异操作具体实现过程见算法2.

算法 2. 变异操作 Mutation.

输入: 种群 X;

输出: 变异中间体 H.

1. **for** i=1 to NP
2. (num_1,num_2,num_3)=在区间[1,NP],产生3个不同且不等于 i 的数;
3. ($best$,$middle$,$worst$)= $sort(Fitness(X_{num_1}),Fitness(X_{num_2}),Fitness(X_{num_3}))$;
   //按照3个个体的适应度值进行升序排序
4. $F_i=F_l+(F_u-F_l)\times(Fitness(X_{middle})-Fitness(X_{best}))/(Fitness(X_{worst})-Fitness(X_{best}))$;
   //得出第 i 个个体的缩放算子 F_i
5. $TemVector=X_{best}+F_i\times(X_{middle}-X_{worst})$;
6. **for** j=1 to D
7. **if** $TemVector(1,j)$落在解空间中
8. $h(i,j)=TemVector(1,j)$;
9. **else**
10. $h(i,j)$=采用修补算子进行修补计算;
11. **endif**
12. **endfor**
13. **endfor**
14. **return** H

算法2是对第 g–1代所有个体进行变异操作,其中,第2行~第4行得到第 i 个个体的缩放算子 F_i,根据该缩放算子,对第 i 个个体进行变异操作(算法第4行).而得到的变异个体的染色体有可能落在解空间之外,所以需要采用修补算子进行修补计算(算法第6行~第11行).最终返回均落在解空间中的变异种群.

2) 自适应的交叉操作

对适应度好的个体,取较大的 CR,可使该个体进入下一代的几率增大;对适应度差的个体,取较小的 CR,加快改变其的结构,使该个体能尽快被淘汰掉.所以,对于每代的进化,按照公式(8)对 CR 进行取值:

$$CR_i=\begin{cases}CR_l+(CR_u-CR_l)\dfrac{f_i-f_{\min}}{f_{\max}-f_{\min}}, & \text{if } f_i<\overline{f}\\ CR_l, & \text{if } f_i\geqslant\overline{f}\end{cases} \tag{8}$$

其中,f_i 是个体 X_i 的适应度,$f_{\min}$ 和 $f_{\max}$ 分别为当前种群中最优和最差个体的适应度, $\overline{f}$ 是当前种群中适应度平均值,CR_l 和 CR_u 分别是 CR 的下限和上限(比如 CR_l=0.1,CR_u=0.6).自适应的交叉操作具体实现过程见算法3.

算法 3. 交叉操作 Crossover.

输入:种群 X,变异中间体 H;

输出:交叉中间体 V.

1. $f_{\min}=\min(Fitness(X))$; //当前种群中最小的适应度值,即当前最优解的适应度值
2. $f_{\max}=\max(Fitness(X))$; //当前种群中最大的适应度值,即当前最差解的适应度值
3. $f_{aver}=aver(Fitness(X))$; //当前种群的平均适应度值
4. **for** i=1 to NP
5. **if** $Fitness(X\_next(i))<f_{aver}$
6. $CR_i=CR_l+(CR_u-CR_i)\times((Fitness(X\_next(i))-f_{\min})/(f_{\max}-f_{\min}))$;

```
7.        else
8.           CR_i=CR_l;
9.        endif
10.       for j=1 to D
11.          if rand(0,1)>CR_i
12.             v(i,j)=x(i,j);
13.          else
14.             v(i,j)=h(i,j);
15.          endif
16.       endfor
17.    endfor
18. return V
```

算法 3 是对 g–1 代种群变异过后进行的交叉操作.算法第 1 行~第 3 行得到当前种群中适应度最小、最大和平均值.如果个体的适应度值小于当前种群平均适应度值,交叉概率按照公式(8)计算得出;否则,交叉概率为 CR_l(算法 3 第 5 行~第 9 行).然后,按照交叉概率对种群进行交叉操作(算法 3 第 10 行~第 15 行).最终返回交叉后的种群.

3.3 适应度函数构造及编码

与遗传算法和粒子群算法类似,差分进化算法也采用适应度概念,用来度量种群中的各个个体在优化计算中有可能达到或接近于或有助于找到最优解的优良程度,适应度值较低的个体进入到下一代的概率较大,而适应度值较高的个体进入到下一代的概率就相对小一些(仅针对最小化优化问题,最大化优化问题则反之).度量个体适应度值的函数称为适应度函数,其一般由目标函数变换而来.

针对本文问题,我们的最终目标是找到最优恢复规划,其优化程度由服务的 SLA 属性确定.而一个恢复规划通常需要考虑多个 SLA 属性,所以需要对这些 SLA 属性进行聚合.本文沿袭我们的前期工作[19,26–30]对 QoS 的聚合方法对 SLA 聚合,只考虑消极指标的最小化,对于积极指标采用乘上–1(负值计算)就可转化为消极指标[17],并采用文献[31,32]中对所有 SLA 属性进行归一化处理,使得不同量纲的 SLA 属性值可以进行比较,并采用简单加权方法构建聚合函数.

假如恢复规划中含有 m 个任务(对应差分进化算法中解的维数 D),每个任务需要绑定的服务需要考虑 l 个 SLA 属性,S_{ij} 表示第 j 个任务需要绑定服务的第 i 个 SLA 归一化后的属性值,w_i 表示对第 i 个 SLA 属性选择的权重(其值可根据用户偏好设置),且 $\sum_{i=1}^{l} w_i = 1$,那么聚合函数为

$$U = \sum_{j=1}^{m}\sum_{i=1}^{l} S_{ij} w_i \tag{9}$$

那么,最优恢复规划为使得聚合函数 U 取得最小值时的对 m 个任务的一组服务绑定,即:

$$r=\arg\min U \tag{10}$$

也就是说,公式(9)为本文目标函数.所以,我们直接采用其作为适应度函数,即构造差分进化算法的适应度函数如公式(11):

$$f = \sum_{j=1}^{m}\sum_{i=1}^{l} S_{ij} w_i \tag{11}$$

构造出适应度函数后,为了能采用差分进化算法求出最优恢复规划,还需要对算法的个体进行编码.针对本文问题,个体采用整数数组表示,数组包含元素的个数为恢复规划中抽象组件的个数,数组中的每个整数元素为抽象组件可绑定的组件服务的索引,其最小值为 1,最大值为该组件可绑定组件服务的数目.通过循环迭代,改变数组元素的值(经过修补和取整操作得到落到解空间的整数)等同于改变抽象组件的组件服务绑定.而差分进化

算法的最终目标是根据适应度值,计算使得适应度值取得最小值的一组组件服务索引.

算法 4 为适应度函数的具体实现过程,其根据整数数组的值,取出对应每个任务的组件服务绑定,根据每一个组件服务经过归一化后的 SLA 属性值,计算恢复规划的聚合函数值,即适应度值.

算法 4. 适应度值计算 *Fitness*.

输入:个体 x,权重 w;

输出:适应度值 f.

1. 根据个体 x 的取值,取出对应每个任务的组件服务绑定以及每个组件服务的、经过归一化后的 SLA 属性值 S;
2. f=0;
3. **for** j=1 to m
4. **for** i=1 to l
5. $f=f+S_{ij}\times w_i$;
6. **endfor**
7. **endfor**
8. **return** f

4 仿真实验

为了验证 SLAFT 方法的有效性,我们进行了大量的仿真实验:首先,将我们提出的方法与其他两种方法在故障处理时间和组合最优度两个性能指标进行了实验对比;然后,对 FRFTA 方法中的重要参数进行了分析.

4.1 实验建立

我们的前期工作[30,33,34]也是研究组合服务容错方法,但其是更侧重于非事务型组合服务的容错方法.事务型组合服务的容错方法与非事务型容错方法有很大不同,所以本文延续我们的前期工作,针对事务型组合服务的容错方法进行了研究.我们在前期实验环境的基础上进行了改进,以更适合本文场景,然后基于改进后的实验环境进行了本文的实验.

为了进行仿真实验,需要有关于服务 SLA 的数据集,但我们没有找到足够进行实验的相关数据.而服务的 QoS 也可以看做服务使用者与服务提供者事先商定的,关于服务提供者对于其提供服务的 QoS 方面达成的 SLA,即 QoS 为 SLA 的一部分.所以不失一般性,我们还继续采用两个真实的关于服务的 QoS 的数据集 QWS 和 WS-DREAM 的 QoSDataset2 进行我们的实验.

QWS[35]包含 2 507 个真实 Web 服务,每个服务包含 9 个 QoS 属性.WS-DREAM 的 QoSDataset2[36,37],包含 339 个用户使用 5 825 个 Web 服务的响应时间(response time)和吞吐率(throughput).实验环境为:Intel(R) Core(TM) i5-4210U 2.39GHz,8.0GB of RAM,Windows 8.1 专业版.

本文实验中,组合服务包含的任务数量设置为 5~60,候选服务数量设置为 60.组合服务的最初的、最优任务绑定可通过已有文献[38,39]的方法求得.组合服务执行过程为:按照组合服务的结构,执行其的每一个组件服务.当遇到故障时,采用容错方法排除出现的故障,然后继续执行组合服务,直至其全部服务均执行完成,也就是说,组合服务成功执行.软件可靠性(software reliability)是软件产品在规定的条件下和规定的时间区间完成规定功能的能力,其值为软件不引起系统失效的概率,即不出现故障的概率.所以,组件服务的可靠性值决定了组合服务执行过程中可能出现的故障数量.仿真实验中,我们通过设定组件服务的不同可靠性值来仿真组合服务执行过程中出现的不同故障规模.为了便于与其他方法进行比较以及说明本文所提方法的有效性,在仿真实验中,采用故障处理时间和组合最优度两个性能指标.具体定义如下.

定义 5(故障处理时间). 在组合服务的执行过程中,排除出现的故障所花费的时间总和 F_t,即:

$$F_t=\sum_{i=1}^{m} f_t^i \tag{12}$$

其中,m 代表执行过程中出现的故障的总次数, f_t^i 代表处理第 i 次故障所耗费的时间(从故障产生至排除,恢复组合服务地执行所耗费的总时间,单位为 ms).F_t 值的大小反映了处理故障的效率,其值越小,则算法效率越高;其值越大,则算法效率越低.

定义 6(组合最优度). 为组合服务的 SLA 聚合值,其采用文献[31,32]中的简单加权方法构建,即

$$U=\sum_{i=1}^{N}(0.5r_i+0.5th_i) \tag{13}$$

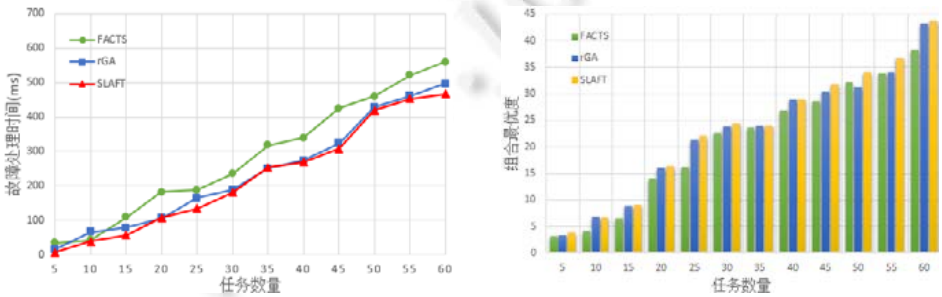
因为 WS-DREAM 的数据集 QoSDataset2 只包含服务的响应时间和吞吐率,我们只考虑这两个 SLA 属性,两者的权重均设定为 0.5(也可以根据用户偏好设定).公式(13)中:N 为组合服务包含的任务个数;r_i 和 th_i 分别为采用文献[31,32,40]中方法归一化后的服务 i 的响应时间和吞吐率;U 代表组合最优度,其值的大小代表组合服务 SLA 属性的优劣,值越高,代表 SLA 属性越优;值越低,则相反.

4.2 实验对比

首先进行了实验对比,以此来验证我们所提方法 SLAFT 的有效性.对比方法有两种,分别为 FACTS[9] (提出了一个容错框架 FACTS,其为事务性型组合服务的容错提供了一个包括规范、验证和执行的集成环境)和 rGA[16](基于改进的遗传算法对组合服务进行补偿).不同类型的组合服务包含具有原子特性的任务数占总任务数的比率各不相同,不失一般性,我们仅针对比率为 10%,50%和 100%的 3 种类型的实验(分别采用 A 类实验、B 类实验和 C 类实验表示),分别采用我们所提方法 SLAFT 和两种对比方法进行容错实验.WS-DREAM 的 QoSDataset2 数据集中没有组件服务的可靠性,所以对组件服务的可靠性均设定为 0.9.对于 3 类实验,分别按照任务数量从 5 开始,并按照以 5 依次增加的规律,采用 3 类方法分别进行实验.为了保证实验的公正性,结果均取运行 100 次实验后的平均值.

4.2.1 A 类实验结果

图 5 为我们所提方法 SLAFT 与两种对比方法 FACTS 和 rGA 对于 A 类实验,不同总任务数量情况下的实验结果对比,图 5(a)为所需故障处理时间结果,图 5(b)为组合最优度实验结果对比.从图 5(a)和图 5(b)可以看出:采用 SLAFT 方法所需故障处理时间略少于 rGA 方法,较少于 FACTS 方法;而组合最优度略优于 rGA 方法,而较优于 FACTS 方法.但总的来说,3 类方法差别不是特别大.分析出现这种结果的原因为:在 A 类实验中,具有原子特性的任务个数占总任务个数的比率仅为 10%,即在出现故障后,仅需要回溯较短的步长就可开始寻找最优替代路径.也就是说,恢复规划中所含任务数量较少,这使得整个状态空间不是很大,所以故障处理时间均较短,且组合最优度均较高.FACTS 因没有采用启发式算法,导致总的故障处理时间偏多和组合最优度偏低,而同采用启发式算法的 rGA 和 SLAFT 方法,故障处理时间较短,组合最优度较高.且还可以看出,SLAFT 略优于 rGA.对比实验结果说明:虽同为启发式算法,针对本文所提问题,差分进化算法要优于遗传算法.



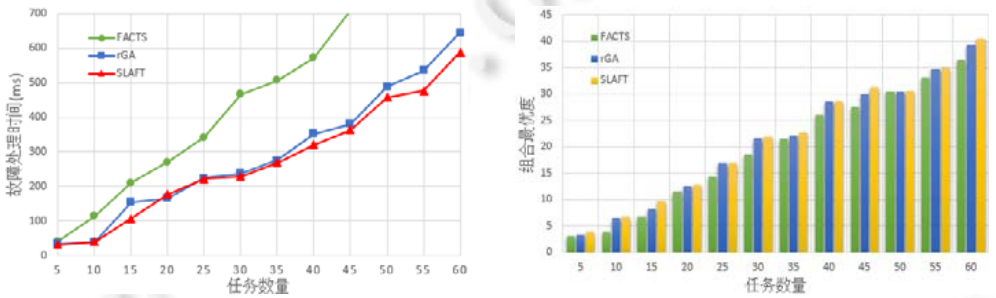
(a) 故障处理时间 (b) 组合最优度

Fig.5 Comparison results for experiment A

图 5 A 类实验的结果对比

4.2.2 B 类实验结果

图 6 为 B 类实验结果,B 类实验的组合服务中,具有原子特性的任务数占总任务数的比率为 50%,与 A 类实验相比,这类组合服务一旦出现故障,其需要回溯的步数会变多,恢复规划中包含的任务数也会增加.而随着任务数的增加,其寻优过程中的状态空间会出现指数级别增长.这种特性导致不采用启发式算法的 FACTS 在任务数比较少时,故障处理时间和 rGA 和 SLAFT 还没有太大差别(见图 6(a));但随着任务数的增多,其故障处理时间增加很多,这将使得在对时延敏感的场景下不可采用 FACTS 方法.rGA 和 SLAFT 方法在任务数较少的情况下,故障处理时间线性增加,在任务数大于 40 个后,故障处理时间增加明显.但任务数大于 40 的组合服务一般为较长事务,人们对此类事务的运行时间预期也会增加.从图 6(a)和图 6(b)可知:SLAFT 方法在故障处理时间略低于 rGA,在组合最优度上略高于 rGA,这说明 SLAFT 方法更接近最优解.与图 5(b)相比,图 6(b)的整体结果均有所降低,这是因为与 A 类实验相比,B 类实验中的恢复规划任务数增多,也就是说,有更多的任务因故障解除了与最初的、最优组件服务的绑定,重新绑定了次优组件服务,这将导致组合最优度的降低.



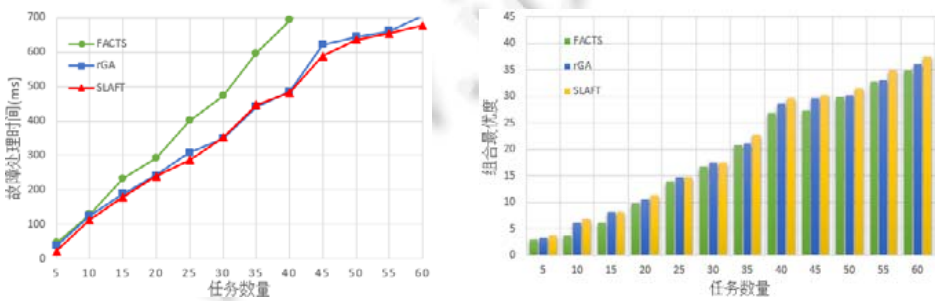
(a) 故障处理时间 (b) 组合最优度

Fig.6 Comparison results for experiment B

图 6 B 类实验的结果对比

4.2.3 C 类实验结果

C 类实验的组合服务中具有原子特性的任务数占总任务数的比率为 100%,也就是说,此类组合服务在执行过程中一旦出现故障,必须回溯到组合服务的开始,重新绑定组件服务,重新执行组合服务,即恢复规划中包含任务数量会更多,所以与 A 类实验和 B 类实验相比,其故障处理时间增加较多(如图 7(a)所示).又因为有更多的任务因故障解除了与最初的、最优组件服务的绑定,重新绑定了次优组件服务,这将导致组合最优度的进一步降低(如图 7(b)所示).但总体来说,SLAFT 方法更趋近最优解.



(a) 故障处理时间 (b) 组合最优度

Fig.7 Comparison results for experiment C

图 7 C 类实验的结果对比

从上述 3 类实验结果来看,我们所提方法 SLAFT 与 rGA 比较,还是有一定的优势.本文研究问题为 NP-hard,

在问题规模较大的情况下,只能寻找近似最优解,为了描述方便,我们一般称为最优解.SLAFT 和 rGA 均为启发式算法,采用这两种算法求得的最终结果均不是数学意义上的最优解,而是接近于最优解的近似解.与对比算法 rGA 相比,我们所提方法 SLAFT 提高幅度不是特别明显,甚至在曲线和柱状图上有交叉和各有上下的情况,但总体上其结果略优.为了方便展示略优结果,我们对实验结果作了进一步统计,统计结果见表 1.

Table 1 Improvement rate of SLAFT

表 1 SLAFT 的改进率

实验类型	A 类实验		B 类实验		C 类实验	
	与 FACTS (%)	与 rGA (%)	与 FACTS (%)	与 rGA (%)	与 FACTS (%)	与 rGA (%)
故障处理时间	21.04	5.55	33.61	7.71	29.45	2.77
组合最优度	12.57	3.46	11.52	2.66	10.35	2.34

表 1 根据上述 3 类实验的结果,依据公式(14)计算得到 SLAFT 方法与对比方法的改进率:

$$I=\frac{|r_c-r_{SLAFT}|}{r_c}\times 100\% \tag{14}$$

其中,I 表示方法 SLAFT 与对比方法相比较的改进率,r_c 和 r_{SLAFT} 表示对比方法和 SLAFT 实验结果(同一类型实验结果求和).从表 1 可以看出,SLAFT 方法的实验结果整体上略优于 rGA.也就是说,我们所提 SLAFT 方法的求解结果更接近于最优值.

4.3 参数分析

作为一种容错方法,故障规模将会直接影响实验结果.为了进一步验证我们所提方法 SLAFT 的性能,需要改变故障规模大小,以此来观察其对实验结果的影响.而故障的规模由组件服务的可靠性值确定,所以我们改变组件服务的可靠性值,继续进行实验.在现实生活中,可靠性低于 0.5 的组件服务基本上已不可用,所以不对其进行实验.因此,针对参数分析实验,设定组合服务均包含任务数为 20,每个组件服务的可靠性从 0.5 开始,按照依次增加 0.05 的规律,直到 0.95.对包含具有原子特性的任务数占总任务数的比率从 10%开始,依次增加 10%,直至 100%的组合服务分别进行实验.每类实验运行 100 次,实验结果取其平均值.

图 8 为不同可靠性取值的、采用 SLAFT 方法的故障处理时间实验结果.

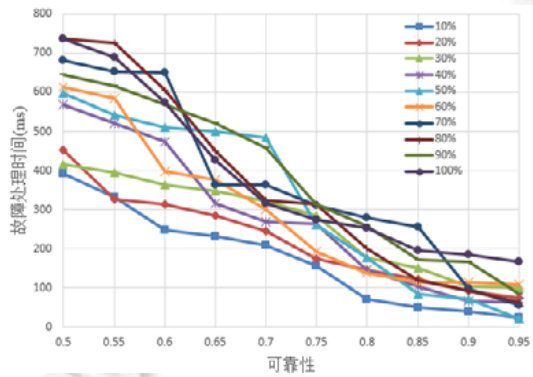


Fig.8 Experimental results of fault handling time for different value of reliability

图 8 不同可靠性取值的故障处理时间实验结果

从图中可以看出:随着组件服务可靠性的减小,故障处理时间渐渐增多;随着具有原子特性的任务数占总任务的比率的增加,故障处理时间也随之增加.这是因为随着可靠性的降低,出现故障的概率就会增加,故障处理时间也随之增加.而随着原子特性任务数占总任务数比率的增加,恢复规划中包含任务数将会增多,寻找最佳替代路径的时间随之增加,导致总的故障处理时间增加.而组件服务可靠性的降低将会增加故障规模,从图 8 可以看出,SLAFT 方法对故障规模较大的情形也能计算出故障处理时间.而原子特性任务数占总任务数的不同比率

代表了不同类型的组合服务,图 8 显示 SLAFT 方法针对不同类型的组合服务均适应良好.

图 9 为不同可靠性取值下,采用 SLAFT 方法的组合最优度实验结果.随着可靠性取值的降低或者原子特性任务数占总任务数比率的增加,组合最优度会降低.这是因为可靠性较低时,组件服务在执行过程中出故障的概率将增加;而随着原子特性任务数占总任务数比率的增加,恢复规划中包含任务数将会增多.而这两种情况都会导致组合最优度的降低.但从图 9 可以看出,SLAFT 方法均可计算出最终组合最优度值.

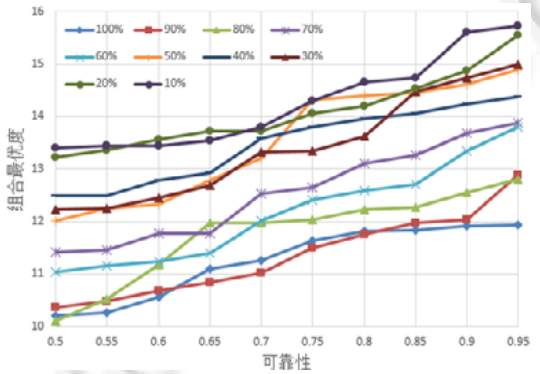


Fig.9 Experimental results of composition optimization for different value of reliability

图 9 不同可靠性取值的组合最优度实验结果

通过实验对比和参数分析可以看出:SLAFT 方法与对比算法相比,拥有较低故障处理时间的同时,还具有较高的组合最优度,且对不同故障规模适应良好.

5 相关工作

针对事务性组合服务的事务特性与容错方法,许多学者提出了研究思路或解决方法,并取得了较好的成果.限于篇幅原因,本文只对一些与本文类似的文献予以分析.

传统的服务选择方法根据组合服务的功能性需求,满足用户要求的 QoS 约束,而不考虑组合过程中的事务性约束(transactional constraints).而文献[41]认为,事务属性是确保组合服务可靠执行的必备条件之一,所以必须在服务选择过程中兼顾事务性属性.并提出了一种用于组合服务设计阶段的自动服务选择方法,其在选择过程中整合(integrate)QoS 和事务性需求.通过该方法为组合服务选择的每个组件服务,在确保满足全局事务约束的前提下,是满足局部 QoS 约束中最好的.文献[42]采用用户查询(user query)方法形式化定义组合服务问题,采用扩展的 CPN(colored petri-nets)技术来实现在服务选择过程中兼顾事务性属性.但上述两个文献仅在组合服务执行前的服务选择阶段考虑事务性,其只能在组合服务能完全正确执行时才能确保事务特性,而一旦遇到执行故障,事务特性将无法保障.

文献[43]针对事务性组合服务提出了一种容错方法,该方法可支持向后和向前错误恢复,而向后错误恢复采用松弛的原子执行(relaxed atomic exectution)策略实现,向前错误恢复采用异常处理实现.为了实现原子执行,还提出了一种可扩展的交付协议 SCP(scalable commit protlcol),其允许一个组合服务包含异质的事务性 Web 服务.同时还提出了一个恢复算法,其可以确保出现故障的组合服务的继续执行.但文献[43]提出的 SCP 比较低效,可能出现资源被未知过程占用一段时间的现象.

文献[44,45]采用 CPN 技术处理事务性组合服务执行过程中出现的故障.文献[44]提出了一个高效、可容错和保证事务性组合服务正确执行的框架,该框架通过组件服务替代和补偿协议支持向前或向后的恢复,具体实现过程采用 CPN.文献[45]也提出一个框架,名称为 FaCETa(fault tolerant cws execution based on transactional properties),其基于 CPN 的展开过程来处理故障.虽然 CPN 技术可以用来处理事务性组合执行过程中出现的故障,但 Petri-Nets 构建出来的模型一般比较庞大,且不支持构造大规模模型,所以文献[44,45]所提框架只适合应

用于组件服务比较少的事务性组合服务的故障处理过程.

文献[9]为事务型组合服务提出了一种容错框架 FACTS,其为容错提供了一个包括规范、验证和执行的集成环境.且该框架与组合服务执行逻辑分离,使得故障处理逻辑易于开发、维护和更新.其次,采用 ECA(event-condition-action)规则来描述何时和如何触发故障处理逻辑.另外,提出 EXTRA(exception handing+transaction)机制,其融合了异常处理和事务技术(transaction techniques)两种容错机制,用于增强组合服务的可靠性.EXTRA 采用八类高层异常处理机制来修复组合服务执行过程中的故障,给出了一种新的 Web 服务事务型分类,分析了事务型组合服务的关键特性,即服务转移(service transfer)和活力程度(vitality degree).同时也定义了一个协议 STTP(service-transfer-based termination protocol),其可使得组合服务产生不可修复的故障时终止在一个一致的状态.但该文献存在一些局限性:1) 故障处理过程只能在组合服务的全状态空间中寻找恢复规划,势必会导致所提方法的扩展性较差;2) 故障处理过程中没有考虑 SLA 属性,而具有较低 SLA 属性值的恢复规划可能导致组合服务的 SLA 违反.

文献[16]对 BPEL 进行了扩充,使之更适合应用于事务性组合服务执行过程中的补偿操作和恢复规划的执行.并对传统遗传算法进行了改进,用于恢复规划的制定,具体改进为:1) 通过采用动态长度染色体(dynamic-length chromosomes)来保存恢复规划,其在递归过程中可根据实际需求申请存储空间,可降低算法的空间复杂度;2) 制定的恢复规划除了满足组合服务的 QoS 约束外,且是在组合服务的部分状态空间中制定得到,又进一步减少了算法执行时间,使之可更容易扩展到组件服务较多的组合服务的故障处理之中或者应用到在线实时故障处理;3) 收集递归过程中的中间解信息,结合组合服务的结构,共同作用于后面的递归过程,使得得到最优解的几率加大.虽然文献[16]对遗传算法进行了改进,但遗传算法对参数取值比较敏感,不同参数设置会得出不同结果,且其实现较复杂的特性还是没有根本改变.所以,与本文所提 SLAFT 方法相比,文献[16]故障恢复时间较长,且组合最优度偏低(详见第 4.2 节).也就是说,虽同为启发式算法,针对本文所提场景,差分进化算法要优于遗传算法.

6 结论与未来工作展望

针对事务性组合服务执行期间出现的故障,本文提出了一种容错方法 SLAFT.首先,SLAFT 方法独立于组合服务的执行逻辑,只有在出现故障的时候才会被触发,因此其易于开发、维护和更新;其次,为了确保组合服务出现故障时能停留在一个一致的状态,SLAFT 采用有限状态机建模其的执行状态,并对其进行状态监控;再次,为了不违反服务使用者和服务提供者事先商定的 SLA 属性,采用监控自动机进行执行监督;最后,进行补偿操作时,SLAFT 采用改进的差分进化算法,其可以快速且精确的找到最优恢复规划.基于两个真实数据集的实验,验证了本文所提 SLAFT 方法在故障处理时间和组合最优度上均优于对比方法,且对不同故障规模适应良好.

虽然所提 SLAFT 方法与其他方法相比有一定的优势,但其也存在不足:(1) 不是通过严格的数学证明 SLAFT 方法优于其他对比方法,只是从实验结果上给于了验证;(2) 差分进化算法的初始种群是随机生成的,如果能找到对初始种群地生成优化算法,有可能进一步减少算法执行时间,进而提高容错的实时性;(3) 仅在模拟环境下实现 SLAFT 方法,其应如何在实际的网络环境中进行部署、其有效性如何以及是否优于其他对比算法均没有得以验证.我们未来的工作将针对存在的这些不足,进一步提高 SLAFT 方法的有效性.

致谢 在此,对提供 QWS 数据集的加拿大圭尔夫大学的 Mahmoud 博士和 Al-Masri 博士,以及提供 WS-DREAM 数据集的中山大学的郑子彬副教授表示衷心感谢.

References:

[1] Delac G, Silic M, Srbljic S. A reliability improvement method for SOA-based applications. IEEE Trans. on Dependable and Secure Computing, 2015,12(2):136−149. [doi: 10.1109/TDSC.2014.2327971]

[2] Li G, Zhao ZF, Han YB, Liang Y. CAFISE Framework based development for service oriented applications with high adaptability. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1372−1380 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1372.htm [doi: 10.1360/jos171372]

[3] Zhuang Y, Zhang PC, Li WQ, Feng J, Zhu YL. Web service QoS monitoring approach sensing to environmental factors. Ruan Jian Xue Bao/Journal of Software, 2016,27(8):1978−1992 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4850.htm [doi: 10.13328/j.cnki.jos.004850]

[4] Yu WD. A software fault prevention approach in coding and root cause analysis. Bell Labs Technical Journal, 1998,3(2):3−21. [doi: 10.1002/bltj.2101]

[5] Littlewood B. Stochastic reliability-growth: a model for fault-removal in computer-programs and hardware-designs. IEEE Trans. on Reliability, 1981,30(4):313−320. [doi: 10.1109/TR.1981.5221099]

[6] Vergura S, Acciani G, Amoruso V. Inferential statistics for monitoring and fault forecasting of PV plants. In: Proc. of the Int'l Symp. on Industrial Electronics. Cambridge: IEEE, 2008. 2414−2419. [doi: 10.1109/ISIE.2008.4677264]

[7] Randell B. System structure for software fault tolerance. IEEE Trans. on Software Engineering, 1975,SE-1(2):220−232. [doi: 10.1109/TSE.1975.6312842]

[8] Issarny V, Tartanoglu F, Romanovsky A, Levy N. Coordinated forward error recovery for composite Web services. In: Proc. of the Int'l Symp. on Reliable Distributed Systems. Florence: IEEE, 2003. 167−176. [doi: 10.1109/RELDIS.2003.1238066]

[9] Liu A, Li Q, Huang L, Xiao M. Facts: A Framework for fault-tolerant composition of transactional Web services. IEEE Trans. on Services Computing, 2010,3(1):46−59. [doi: 10.1109/TSC.2009.28]

[10] Wu GQ, Wei J, Huang T. A dynamic QoS assessment approach for internetware with uncertainty reasoning. Ruan Jian Xue Bao/Journal of Software, 2008,19(5):1173−1185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1173.htm [doi: 10.3724/SP.J.1001.2008.01173]

[11] Ludwig H, Keller A, Dan A, King RP, Franck R. Web Service Level Agreement (WSLA) Language Specification. Watson: IBM Corporation, 2003. 815−824.

[12] Wieder P, Butler JM, Theilmann W, Yahyapour R. Service Level Agreements for Cloud Computing. New York: Springer-Verlag, 2011. 13−20.

[13] Jordan D, Evdemon J. Web Services Business Process Execution Language Version 2.0. Vol.2. OASIS Standard, 2007.

[14] Baresi L, Guinea S. Self-Supervising bpel processes. IEEE Trans. on Software Engineering, 2011,37(2):247−263. [doi: 10.1109/TSE.2010.37]

[15] Foster H. A rigorous approach to engineering Web service compositions [Ph.D. Thesis]. London: Imperial College of London, 2006.

[16] Tan TH, Chen M, André É, Sun J, Liu Y, Dong JS. Automated runtime recovery for QoS-based service composition. In: Proc. of the Int'l Conf. on World Wide Web. New York: ACM Press, 2014. 563−574. [doi: 10.1145/2566486.2568048]

[17] Alrifai M, Risse T. Combining global optimization with local selection for efficient QoS-aware service composition. In: Proc. of the Int'l Conf. on World Wide Web. New York: ACM Press, 2009. 881−890. [doi: 10.1145/1526709.1526828]

[18] Parra-Hernandez R, Dimopoulos NJ. A new heuristic for solving the multichoice multidimensional knapsack problem. IEEE Trans. on Systems, Man, and Cybernetics—Part A: Systems and Humans, 2005,35(5):708−717. [doi: 10.1109/TSMCA.2005.851140]

[19] Wang SG, Sun QB, Yang FC. Web service dynamic selection by the decomposition of global QoS constraints. Ruan Jian Xue Bao/Journal of Software, 2011,22(7):1426−1439 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3842.htm [doi: 10.3724/SP.J.1001.2011.03842]

[20] Qin AK, Huang VL, Suganthan PN. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. on Evolutionary Computation, 2009,13(2):398−417. [doi: 10.1109/TEVC.2008.927706]

[21] Mallipeddi R, Suganthan PN, Pan QK, Tasgetiren MF. Differential evolution algorithm with ensemble of parameters and mutation strategies. Applied Soft Computing, 2011,11(2):1679−1696. [doi: 10.1016/j.asoc.2010.04.024]

[22] Das S, Abraham A, Konar A. Automatic clustering using an improved differential evolution algorithm. IEEE Trans. on Systems, Man, and Cybernetics—Part A: Systems and Humans, 2008,38(1):218−237. [doi: 10.1109/TSMCA.2007.909595]

[23] Storn R, Price K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997,11(4):341−359. [doi: 10.1023/A:1008202821328]

[24] Wang Y, Cai Z, Zhang Q. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans. on Evolutionary Computation, 2011,15(1):55−66. [doi: 10.1109/TEVC.2010.2087271]

[25] Tang L, Dong Y, Liu J. Differential evolution with an individual-dependent mechanism. IEEE Trans. on Evolutionary Computation, 2015,19(4):560−574. [doi: 10.1109/TEVC.2014.2360890]

[26] Wang SG, Sun QB, Zhang GW, Yang FC. Uncertain QoS-aware Skyline service selection based on cloud model. Ruan Jian Xue Bao/Journal of Software, 2012,23(6):1397−1421 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4084.htm [doi: 10.3724/SP.J.1001.2012.04084]

[27] Wang SG, Sun QB, Yang FC. Reputation evaluation approach in Web service selection. Ruan Jian Xue Bao/Journal of Software, 2012,23(6):1350−1367 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4051.htm [doi: 10.3724/SP.J.1001.2012.04051]

[28] Ma Y, Wang SG, Sun QB, Yang FC. Web service quality metric algorithm employing objective and subjective weight. Ruan Jian Xue Bao/Journal of Software, 2014,25(11):2473−2485 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4508.htm [doi: 10.13328/j.cnki.jos.004508]

[29] Wang SG, Zhou A, Yang FC, Chang RN. Towards network-aware service composition in the cloud. IEEE Trans. on Cloud Computing, 2016. [doi: 10.1109/TCC.2016.2603504]

[30] Zhang JN, Wang SG, Sun QB, Yang FC. Fast and reliable fault-tolerance approach for service composition in integration networks. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):940−958 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5051.htm [doi: 10.13328/j.cnki.jos.005051]

[31] Zeng LZ, Benatallah B, Ngu AH, Dumas M, Kalagnanam J, Chang H. QoS-Aware middleware for web services composition. IEEE Trans. on Software Engineering, 2004,30(5):311−327. [doi: 10.1109/TSE.2004.11]

[32] Zeng LZ, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ. Quality driven Web services composition. In: Proc. of the Int'l Conf. on World Wide Web. New York: ACM Press, 2003. 411−421. [doi: 10.1145/775152.775211]

[33] Wang SG, Ma Y, Cheng B, Yang FC, Chang R. Multi-Dimensional QoS prediction for service recommendations. IEEE Trans. on Services Computing, 2016. 1−12. [doi: 10.1109/TSC.2016.2584058]

[34] Zhou A, Wang SG, Cheng B, Zheng ZB, Yang FC, Chang R, Michael L, Buyya R. Cloud service reliability enhancement via virtual machine placement optimization. IEEE Trans. on Services Computing, 2016. 1−13. [doi: 10.1109/TSC.2016.2519898]

[35] Al-Masri E, Mahmoud QH. Investigating Web services on the World Wide Web. In: Proc. of the Int'l Conf. on World Wide Web. New York: ACM Press, 2008. 795−804. [doi: 10.1145/1367497.1367605]

[36] Zheng ZB, Zhang Y, Lyu MR. Distributed QoS evaluation for real-world Web services. In: Proc. of Int'l Conf. on Web Services. Miami: IEEE, 2010. 83−90. [doi: 10.1109/ICWS.2010.10]

[37] Zhang YL, Zheng ZB, Lyu MR. Exploring latent features for memory-based QoS prediction in cloud computing. In: Proc. of the IEEE Symp. on Reliable Distributed Systems. Madrid: IEEE, 2011. 1−10. [doi: 10.1109/SRDS.2011.10]

[38] Wang SG, Hsu CH, Liang ZJ, Sun QB, Yang FC. Multi-User Web service selection based on multi-QoS prediction. Information Systems Frontiers, 2014,16(1):143−152. [doi: 10.1007/s10796-013-9455-4]

[39] Deng SG, Wu J, Li Y, Wu ZH. Automatic Web service composition based on backward tree. Ruan Jian Xue Bao/Journal of Software, 2007,18(8):1896−1910 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1896.htm [doi: 10.1360/jos181896]

[40] Liu XZ, Huang G, Mei H. Consumer-Centric service aggregation: Method and its supporting framework. Ruan Jian Xue Bao/Journal of Software, 2007,18(8):1883−1895 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1883.htm [doi: 10.1360/jos181883]

[41] El Hadad J, Manouvrier M, Rukoz M. TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition. IEEE Tran. on Services Computing, 2010,3(1):73−85. [doi: 10.1109/TSC.2010.5]

[42] Cardinale Y, El Haddad J, Manouvrier M, Rukoz M. Web service selection for transactional composition. Procedia Computer Science, 2010,1(1):2689−2698. [doi: 10.1016/j.procs.2010.04.302]

[43] Liu A, Huang LS, Li Q, Xiao MJ. Fault-Tolerant orchestration of transactional Web services. In: Proc. of the Int'l Conf. on Web Information Systems Engineering. Evanston: Springer-Verlag, 2006. 90−101. [doi: 10.1007/11912873_12]

[44] Cardinale Y, Rukoz M. A Framework for reliable execution of transactional composite Web services. In: Proc. of the Int'l Conf. on Management of Emergent Digital EcoSystems. San Francisco: ACM Press, 2011. 129−136. [doi: 10.1145/2077489.2077513]

[45] Angarita R, Cardinale Y, Rukoz M. Faceta: Backward and forward recovery for execution of transactional composite ws. In: Proc. of the Extended Semantic Web Conf. Montpellier: Springer-Verlag, 2012. 343−357. [doi: 10.1007/978-3-662-46641-4_26]

附中文参考文献:

[2] 李刚,赵卓峰,韩燕波,梁英.基于CAFISE Framework的高适应性面向服务软件开发.软件学报,2006,17(6):1372−1380. http://www.jos.org.cn/1000-9825/17/1372.htm [doi: 10.1360/jos171372]

[3] 庄媛,张鹏程,李雯睿,冯钧,朱跃龙.一种环境因素敏感的 Web Service QoS 监控方法.软件学报,2016,27(8):1978−1992. http://www.jos.org.cn/1000-9825/4850.htm [doi: 10.13328/j.cnki.jos.004850]

[10] 吴国全,魏峻,黄涛.基于非确定性推理的网构软件服务质量动态评估方法.软件学报,2008,19(5):1173−1185. http://www.jos.org.cn/1000-9825/19/1173.htm [doi: 10.3724/SP.J.1001.2008.01173]

[19] 王尚广,孙其博,杨放春.基于全局 QoS 约束分解的 Web 服务动态选择.软件学报,2011,22(7):1426−1439. http://www.jos.org.cn/1000-9825/3842.htm [doi: 10.3724/SP.J.1001.2011.03842]

[26] 王尚广,孙其博,张光卫,杨放春.基于云模型的不确定性 QoS 感知的 Skyline 服务选择.软件学报,2012,23(6):1397−1412. http://www.jos.org.cn/1000-9825/4084.htm [doi: 10.3724/SP.J.1001.2012.04084]

[27] 王尚广,孙其博,杨放春.Web 服务选择中信誉度评估方法.软件学报,2012,23(6):1350−1367. http://www.jos.org.cn/1000-9825/4051.htm [doi: 10.3724/SP.J.1001.2012.04051]

[28] 马友,王尚广,孙其博,杨放春.一种综合考虑主客观权重的 Web 服务 QoS 度量算法.软件学报,2014,25(11):2473−2485. http://www.jos.org.cn/1000-9825/4508.htm [doi: 10.13328/j.cnki.jos.004508]

[30] 张俊娜,王尚广,孙其博,杨放春.融合网络环境下快速可靠的服务组合容错方法.软件学报,2017,28(4):940−958. http://www.jos.org.cn/1000-9825/5051.htm [doi: 10.13328/j.cnki.jos.005051]

[39] 邓水光,吴健,李莹,吴朝晖.基于回溯树的 Web 服务自动组合.软件学报,2007,18(8):1896−1910. http://www.jos.org.cn/1000-9825/18/1896.htm [doi: 10.1360/jos181896]

[40] 刘譞哲,黄罡,梅宏.用户驱动的服务聚合方法及其支撑框架.软件学报,2007,18(8):1883−1895. http://www.jos.org.cn/1000-9825/18/1883.htm [doi: 10.1360/jos181883]

张俊娜(1979－),女,河南周口人,副教授,主要研究领域为服务计算.

孙其博(1975－),男,博士,副教授,CCF 专业会员,主要研究领域为网络服务与网络智能化,物联网应用技术.

王尚广(1982－),男,博士,副教授,博士生导师,CCF 高级会员,主要研究领域为服务计算,移动云计算,车联网及网络安全.

杨放春(1957－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为通信软件,网络安全,网络智能化.