一种基于多约束组合的多租户系统配置测试技术

侯可佳 白晓颖 周立柱
(清华大学计算机科学与技术系 北京 100084)
(清华大学信息科学与技术国家实验室 北京 100084)

摘 要 通过灵活配置实现在线软件定制是多租户(Multi-Tenancy Architecture,MTA)软件即服务(Software-as-a-Service,SaaS)系统的一个重要特征.由于参数和行为配置的复杂多样,配置错误是系统质量所面临的主要风险之一.对各种配置的组合测试,是 MTA SaaS 系统测试的一个重要内容.该文针对配置之间的约束依赖关系,提出建立约束依赖图,利用图搜索算法获取具有依赖关联的配置项集合,采用组合测试算法,生成不同的配置测试场景,满足多配置组合覆盖率要求.实验以多租户游戏系统为例,对比多种测试生成算法,实验显示该方法在检测由约束违反引起的系统缺陷方面有显著改善效果.

关键词 约束组合;多租户系统;配置测试;数据生成;组合测试算法
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.00237

Constraints Combinatorial for Configuration Testing of Multi-Tenancy SaaS System

HOU Ke-Jia BAI Xiao-Ying ZHOU Li-Zhu
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
(*National Laboratory for Information Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Customizability is critical to MTA (Multi-Tenancy Architecture) SaaS (Software-as-a-Service) systems so that they can support flexible runtime configuration to address diversified customers' needs. Configuration faults are risky as a system's behavior depends heavily on its configurations. However, configuration faults are hard to detect due to the large number of configurable parameters and their possible combinations, and the complexity of parameter dependencies and constraints. In counter to this challenge, the paper applied combinatorial testing techniques in test case design to achieve confident coverage of parameter constraints. It proposed a heterogeneous graph model, Constraint Dependency Graph (CDG), to abstract the dependency relationships among parameters. The search algorithm is designed to find all the possible sets of related constraints along CDG paths. Various combinatorial strategies are exercised to examine the effectiveness of test case generation with coverage objectives. Experiments are carried on a multi-tenancy game system, which show the effectiveness of the proposed approach in achieving test coverage and detecting defects caused by constraints violations.

Keywords constraints combination; multi-tenancy system; configuration testing; test generation; combinatorial testing

收稿日期:2014-06-24;在线出版日期:2015-07-23.本课题得到国家自然科学基金(61073003)、北京市自然科学基金(4132062)、国家"九七三"重点基础研究发展规划项目基金(2011CB302505)和国家"八六三"高技术研究发展计划项目基金(2013AA01A215)资助.**侯可佳**,女,1983 年生,博士研究生,主要研究方向为软件测试.E-mail:houkejia01@163.com.**白晓颖**,女,1973 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为软件测试、服务计算.**周立柱**,男,1947 年生,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为数据库技术、数字图书馆、大规模信息处理.

1 引 言

针对特定的领域应用,多租户软件即服务(Multi-Tenancy Software-as-a-Service,MTA SaaS)提供基于共享的、可重用的基础数据和基础代码,定制特定的应用系统的服务,其目标是在保证用户个性化的同时尽可能地将资源共享最大化.这种多租户资源共享、租户应用定制和隔离的模式,可以大幅度减少用户构建和维护系统的代价,成为云计算软件服务系统的一个主要发展趋势[1].例如 Salesforce.com 作为目前最具代表性的 SaaS 提供商之一,其用户关系管理(Customer Relation Management,CRM)服务在云平台上能够同时对全球超过 100 000 家企业提供业务支持.

1.1 SaaS 系统可配置性分析

设计良好的 MTA SaaS 系统可在数据、行为、用户界面等多个层面支持用户的定制需求[2].例如,Salesforce 利用元数据驱动方法,实现了共享数据库和数据模式[3];Google App Engine① 采用命名空间机制来区分租户的数据和行为.MTA 架构通常包括两个部分:(1)通用的基础代码,基础代码中包含有丰富的领域知识,包括元数据和业务规则等内容;(2)定制机制,该机制能够满足不同用户的特定需求.可配置性是 MTA 设计的基础,与大多数软件系统中简单直接的参数配置相比,MTA SaaS 的配置需要更加灵活、动态地在基础代码中注入数据、条件规则和行为,实现应用的定制,其配置测试也面临着用户种类和配置场景复杂、缺乏内置的可测试性等挑战.传统可配置软件和 MTA SaaS 的具体区别包括:

(1)传统软件的可配置项主要是简单的参数配置,并且大部分配置项均提供候选项供用户选择,可配置项的数量也非常有限.MTA SaaS 则存在大量的可配置项,这些配置项将不再局限于简单的参数值的修改,用户甚至可以根据自己的需求增加数据项,对服务提供的 API 进行重新组装以完成特定任务等.

(2)传统软件的配置参数在软件编译时即被固化在软件代码中.多租户系统中,除静态编译的参数配置外,还支持系统行为动态的配置.这些配置内容,独立于系统的基础代码,在系统运行过程中,动态载入.这样,既保证了基础代码对所有租户的一致性,又可保证租户可以配置其特定的应用行为;且租户特定的行为更改时,不会影响基础代码,保证了不同租户之间的隔离.

例如,客户关系管理系统(CRM)是 Salesforce 公司在 Force.com 平台上提供的 SaaS 服务,它为用户提供了丰富的可配置选项,用户根据各自的需求和喜好对这些可选项进行配置,即可获得满足其需求的应用实例.Force.com 的 CRM 服务配置包括:

(1)用户界面的配置.对于以表格为主体的界面,例如联系人列表界面,用户可以选择显示或隐藏其中的某些列,或者改变列的顺序,或者通过编写过滤规则,选择需要显示的记录;对于其他类型的界面,例如联系人详细信息界面,用户可以通过自己的喜好更改页面布局,例如改变页面组件的位置、颜色、形状等.

(2)用户管理.服务管理员可以根据人员组织结构为服务添加用户、为用户分组、为用户分配不同的角色,根据角色的不同,为用户赋予不同的使用权限.

(3)数据管理.Force.com 是元数据驱动体系架构,因此数据配置是 Force.com 的核心内容.用户通过增加自定义对象来存储组织用户特有的数据,自定义对象创建成功后,便能够进一步创建自定义选项卡、报表、仪表板等,使用户能够对自定义对象进行交互操作.Force.com 并没有为用户数据创建实际的数据库表格,而只是存储相关的元数据,在服务运行时,利用这些元数据动态的生成用户数据表格.

(4)应用程序管理.当 Force.com 平台所提供的服务无法满足用户需求时,用户还可以在自定义对象的基础上,利用平台提供的 API 开发所需服务,完成用户需要的特殊服务.

1.2 SaaS 系统故障模式分析

MTA 共享/隔离机制可能给系统的质量、性能和安全带来风险[2].SaaS 平台的可靠性、可用性和安全性等对大量多租户应用的可靠安全至关重要,因此,在各种租户场景下,验证和测试系统行为的正确性、鲁棒性,是 MTA SaaS 系统开发的一个重要环节.

基于对多租户体系架构的分析,MTA SaaS 的故障能够划分为如下 4 类:MTA 数据库故障、通用

① Google App Engine. http://code.google.com/appengine/

基础代码故障、配置故障以及运行故障.

(1)数据库故障.多租户数据库的设计对于确保公共共享数据和应用专用数据的完整性、正确性和安全性至关重要[4].不同租户数据的共享和隔离均可能引发不可预见的故障.

(2)通用基础代码故障.通用基础代码对于所有的租户都是一致的,其中包含了系统的规则和行为等内容.通过对基础代码的重用,能够大大降低每个应用程序的开发和维护费用.同时,基础代码的质量也成为 SaaS 系统质量的基础.

(3)配置故障.丰富多样的可配置组件是实现 MTA SaaS 的关键,同时,配置的多样性和动态性又是极易引发故障的因素.配置故障是本文的主要研究重点,且会在论文的后续部分进行详细讨论.

(4)运行故障.不同租户应用实例的执行均可能会对彼此产生影响,特别是当存在大量不同的租户配置时,运行行为更加难以预料和控制.

在以上 4 类 MTA SaaS 系统的故障类型中,本文将着重研究配置故障.分析研究表明,配置故障主要由以下几方面原因引起:(1)未能合理满足系统的约束条件.设定配置参数常常需要大量的系统领域知识,以满足特定的约束条件,部分约束条件为系统的隐含假定,或是内嵌在基础代码中,由于文档不完善、理解错误、或是系统安全保护不够等原因,常常由于约束条件不符合而造成租户应用错误.(2)未能满足多配置之间的依赖关系.多参数的配置常常相互制约相互影响,参数的配置除了满足本身的限制之外,还需保持全局的合理性;而一组相互关联的参数之间,由于配置的不匹配所造成的错误,也是系统故障的主要原因之一.这种依赖关系的定义通常分散在各个参数的定义中,或是隐含的系统假设中,容易被忽视从而导致出错.

1.3　主要贡献

针对上述问题,本文从以下两个方面提出了一种新的 MTA SaaS 系统配置测试方法.

(1)提出了约束依赖图(Constraints Dependency Graph,CDG),作为约束条件的分析工具.CDG 模型识别主要的配置项及其约束条件,并将其相互关联起来.每个配置项可能定义多种约束条件,每个约束条件可能包含多个配置项.这样,CDG 的每个连通子图就构成了一组相互依赖的配置项.测试设计转化为可达图搜索问题,并设计生成测试用例以满足每个连通子图上的所有约束依赖关系.

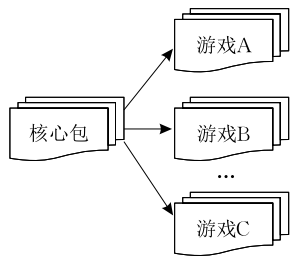
(2)提出了约束的组合测试方法.当参数和约束条件较多、依赖关系复杂时,全覆盖测试常常会引起组合爆炸问题.本文采用组合测试的思想,设计对于约束条件的组合方法,以便在较小的测试代价下达到较高的覆盖率要求,并能有效的提高系统缺陷检测能力.

本文第 2 节以一个游戏系统为例,分析了 MTA SaaS 的配置测试问题,并对问题进行形式化定义和描述;第 3 节提出约束依赖图;第 4 节讨论约束组合测试方法;第 5 节报告实验过程及其结果;第 6 节对相关工作进行总结和讨论;最后,第 7 节是对本文研究工作的总结讨论.

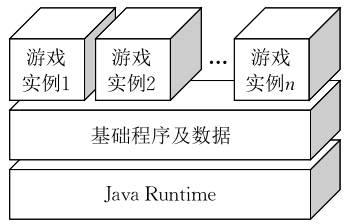
2　问题定义

2.1　案例分析

电子游戏制造器(Arcade Game Maker, AGM)是一个教学系统①,用于学生实践生产线体系架构(Product-Line Architecture, PLA).本文将其重构后,形成一个在线的 MTA SaaS 系统 MTA-AGM,其可在同一游戏平台上,支持配置静止物体、移动物体、物体碰撞规则、游戏得分规则等多种特性,动态生成多种类型的游戏应用.图 1 对比了 PLA 和 MTA 两种不同的体系架构实现方法.PLA 设计中,所有游戏产品共享同一个核心程序包,不同的游戏产品对基础程序包进行扩展,形成不同的单独编译



(a) PLA体系结构



(b) MTA体系结构

图 1　AGM 系统 PLA 与 MTA 体系结构对比

① Arcade Game Maker Pedagogical Product Line. http://www.sei.cmu.edu/productlines/ppl/

的游戏程序包. MTA 系统则只包含一个代码程序包,通过系统配置,在线生成不同的游戏实例. 游戏中物体行为的配置利用了 Java 的反射机制实现动态重新加载类和动态编译.

AGM 游戏定义了一系列静止物体和移动物体,移动物体包含可控移动物体和不可控移动物体. 游戏者通过控制可控移动物体,使其与其他物体发生碰撞从而得分. 租户可以根据各自的需求设置不同的物体行为和游戏机制来实现游戏的个性化定制. MTA-AGM 游戏系统的可配置项包括:

(1)游戏背景属性. 游戏背景限定了移动物体的可移动范围. 在游戏过程中,当移动物体超出游戏背景的范围时,游戏者失败,游戏结束. 游戏背景可以设置为不同的大小和颜色.

(2)物体属性. 静止物体和移动物体包含一组通用属性,例如物体位置、大小、颜色等. 除此之外,移动物体还有一些独特属性,例如移动速度和移动方向等.

(3)物体行为. 碰撞规则定义了每个物体在碰撞之后的行为. 对于静止物体而言,碰撞之后物体的动作包括消失和不变;而移动物体,除了消失和不变之外,还有 3 种动作行为,即垂直反弹、水平反弹、以及全反弹.

(4)游戏机制. 游戏机制包括碰撞动作的具体定义、游戏者的得分规则等等. 本研究将在实验过程中,将这些配置项设置为固定取值,主要针对上述 3 类配置项进行配置测试,考察这些配置项之间不同的取值组合对测试的影响.

2.2 问题定义

给定一个多租户系统 MTA-*Service*$=\{v_1,v_2,\cdots,v_k\}$,其中 $v_i(1\leq i\leq k)$为系统的可配置项,该系统共包含 k 个可配置项,为配置项赋值分为两种情况:(1)从系统给定的候选值中选择;(2)由用户直接赋值. 租户根据各自不同的需求对多租户系统进行配置,从而得到满足其要求的应用实例,如租户 t_i对该系统进行配置得到的应用实例表示为 $C(S,t_i)=\{a_{i1},a_{i2},\cdots,a_{ik}\}$,其中 $v_j=a_{ij}(1\leq j\leq k)$,$a_{ij}$是租户 t_i为第 j 项配置项设定的配置值. 每个配置项通常具有多个可选取值,配置项的取值个数通常有以下几种情况:

(1)配置项的取值是有限并且少量的. 例如在上述碰撞游戏中,物体状态配置项的可选取值只有两个,即移动和静止.

(2)配置项的取值是有限的,但数量非常大. 例如在 800×600 的背景中,物体起始位置横坐标的范围为[0,800),如果要求坐标为正整数,则横坐标可能的取值有 800 个.

(3)配置项的取值是无限多个. 同样以上述横坐标为例,如果其值为正实数,在 800×600 的背景中,可能的取值个数就为无限多个.

这些配置项的不同取值组合能构成不同的配置场景,产生不同的多租户应用实例,为保证多租户系统在各种配置场景下行为的正确性,就必须进行充分的配置测试. 多租户系统的各配置项之间往往存在各种约束依赖关系,对系统进行配置时,如果违反这些约束条件,则可能引发系统故障. 因此在测试过程中,配置项之间的约束条件就需要作为一项重要指标进行考察,尤其是多个约束条件的组合,更容易导致系统故障,需要进行重点测试.

假设被测系统 S 的 k 个配置项之间具有 m 条约束条件:$c_1,c_2,\cdots,c_m$,每条约束条件有多个可能的取值,不同约束条件的不同取值能产生若干不同的约束组合,此研究的目标就是为被测系统的 k 个配置项求解 N 种不同的配置情况,即

$$X=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{Nk} \end{pmatrix}.$$

其中,要求 X 能够实现对 m 条约束条件强度为 2 的组合覆盖. 该组合覆盖要求的具体定义为:m 条约束条件中的任意两条约束条件 c_i和 c_j的每一组取值组合,至少要被一组参数配置所覆盖.

3 约束依赖图

3.1 约束模型

本文基于多租户系统领域模型,提取系统约束模型,约束条件包括参数约束条件、动作约束条件以及状态-动作约束条件,即 *Constraints*$=\langle$*Data-Cons*; *Action-Cons*; *St-Act-Cons*$\rangle$.

参数约束条件 *Data-Cons* 定义了参数之间的约束限制关系,包括对单个参数的约束条件以及多个参数之间的约束条件的定义.

(1)单参数约束条件 $C(d_i)$,约束条件中仅包含一个配置参数,在参数数据类型的基础上描述参数取值方面的约束限制条件,例如对数值型数据的取

值范围的限制、对字符串型数据的长度和模式的限制等；

(2) 多参数约束条件，约束条件中包含多个配置参数，其表示这些参数之间的约束依赖关系，约束条件中可以包含数学运算符、谓词逻辑符等等：

① 简单约束，仅包含两个参数和一个二元关系符的约束条件，即 $C(d_i,d_j)=d_i\ \ rel\ \ d_j$，其中 rel 为二元关系符，如>、<、=等，能够表达两个参数之间大于、小于、等于等关系；

② 复杂约束，包含了两个或两个以上参数以及多个常量数据的约束条件，这些参数和常量之间通过多个运算符或关系符等进行连接.

多租户系统通常存在大量可供选择的动作行为，有些动作之间互不相关，而有些动作之间却存在某些约束依赖关系. 动作约束条件 $Action\text{-}Cons=\langle A_S;A_E;Sequence\text{-}Cons\rangle$ 利用集合定义系统动作的约束依赖关系，包括起始动作、终止动作定义以及动作之间的顺序约束关系等：

(1) A_S 为起始动作集. 所有动作序列均需以该集合内的动作开始，若为空，则动作序列可以系统任意动作开始；

(2) A_E 为终止动作集. 所有动作序列均需以该集合内的动作结束，若为空，则动作序列可以系统任意动作结束；

(3) 顺序约束条件 $Sequence\text{-}Cons=\langle\{After_{act_i}\};\{Parallel_{act_i}\}\rangle$ 为系统动作之间执行次序的要求. 该约束利用两类集合定义了动作顺序执行约束和并行执行约束，顺序执行约束 $After_{act_i}$ 定义了可以在动作 act_i 之后顺序执行的动作集合，并行执行约束 $Parallel_{act_i}$ 为可以与动作 act_i 并行执行的动作集合.

状态-动作约束关系 $St\text{-}Act\text{-}Cons$ 定义了系统状态与相关动作之间的约束依赖关系，例如系统的某些行为动作只有在系统达到某种特定状态时才能够执行，而在其他状态下则不能执行. 因此我们将状态-动作约束关系定义为一种状态到动作集的映射关系集，即 $\{St\rightarrow\{A_i\}\}$，该映射关系表示当系统处于状态 St 时，集合 $\{A_i\}$ 中的动作均能够正常执行，而集合外的其他动作则不能在此状态下进行.

3.2　约束依赖图

本文利用参数的约束依赖图(Constraints Dependency Graph，CDG)来表示参数之间的约束依赖关系，该约束依赖图中包含两类对象 P 和 C：P 为服务参数对象 $P=\{p_1,p_2,\cdots\}$，C 为约束条件对象 $C=\{c_1,c_2,\cdots\}$，CDG 的具体定义及其性质如下：

(1) $CDG=\langle V,E\rangle$，为定义在被测系统配置参数 P 和约束条件 C 上的约束依赖图；

(2) V 为节点集，并且 $V=P\cup C$；

(3) E 为无向边集，$E=\{(p_i,c_j)\}$，其中 $p_i\in P$，$c_j\in C$，其表示约束条件 c_j 中包含参数 p_i，表达参数 p_i 受到约束条件 c_j 限制的含义，不存在连接两个参数对象或者两个约束条件对象的边，即如果边 $e'=(p_i,p_j)$ 或 $e'=(c_i,c_j)$，则 $e'\notin E$；

(4) 参数节点的度 $D(p_i)$ 为与参数 p_i 相关的约束条件的数量，并且 $D(p_i)\geq 0$，若 $D(p_i)=0$，说明参数 p_i 不受任何约束条件的限制；

(5) 约束节点的度 $D(c_i)$ 为约束条件 c_i 中包含的参数个数，并且 $D(c_i)>0$；

(6) 若参数节点 p_i 和 p_j 之间仅经过一个约束节点 c 相连，即 p_i 和 p_j 之间存在路径 $L=p_icp_j$，则说明参数 p_i 和 p_j 出现在同一条约束条件 c 中，即 p_i 和 p_j 之间存在直接约束依赖关系；若节点 p_i 和 p_j 之间经过多个约束节点相连，即 p_i 和 p_j 之间存在路径 $L=p_ic_1p_1\cdots c_{k-1}p_{k-1}c_kp_j\ (k>1)$，则说明参数 p_i 和 p_j 之间通过其他参数 $p_1,\cdots,p_{k-1}$ 等产生联系，存在间接约束依赖关系；

(7) 若参数节点 c_i 和 c_j 之间存在一条路径 $L=c_ipc_j$，说明约束条件 c_i 和 c_j 包含相同的参数 p.

构建参数约束依赖图的过程如算法 1 所示.

算法 1.　数据约束依赖图构造算法.

```
输入：系统参数和参数之间的约束条件
输出：数据约束依赖图
Let G be a graph with no vertices or edges;
FOR (each constraint c_i)
  Generate a vertice vc_i according to constraint c_i
END FOR
FOR (each parameter p_i)
  Generate a vertice vp_i according to parameter p_i
END FOR
FOR (each vertice vc_i)
  IF (constraint c_i contains parameter p_j)
    Then generate an edge e between vc_i and vp_j,
      e=(vc_i,vp_j)
  END IF
END FOR
```

图 2 为一个简单的参数约束依赖图，包含 2 个参数和 3 条约束条件. 其中约束条件 $0\leq x<800$ 和 $0<l\leq 800$ 的度均为 1，表示约束条件中仅包含 1 个参数，约束条件 $0<x+1\leq 800$ 中包含 2 个参数，且该约束节点的度为 2，表示两个参数之间的约束依

赖关系.2 个参数节点的度均为 2,表示参数 x,l 均受到两条约束条件的限制.合理的参数取值能够使与其相关的约束条件为真,如果图中包含的所有参数的取值使全部约束条件均为真,则称这组参数取值为合理参数值,能够激发被测系统的正常运行情况.

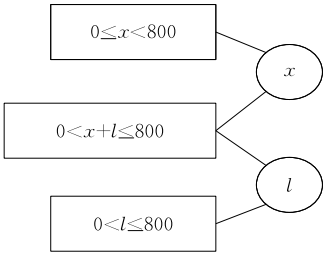


图 2 约束依赖图示例

CDG 的连通性表达了图中所包含的参数以及约束条件之间的相关性,相关性包括以下 3 种类型:

(1)参数之间的相关性.两个参数之间存在路径相连通,则称这两个参数相关,表现为这两个参数同时存在同一条约束条件,或者这两个参数通过其他参数间接相关;

(2)参数和约束条件之间的相关性.参数和约束条件之间存在路径相连通,则称该参数和约束条件相关,表现为约束条件直接包含该参数,或者约束条件中的其他参数与该参数相关;

(3)约束条件之间的相关性.两条约束条件之间存在路径相连通,则称这两条约束条件相关,表现为这两条约束条件包含相同参数,或者相关参数.

对于多租户系统中一组具有相关性的参数,生成其中一个参数的取值时,必须同时考虑其他参数的取值情况,与这些参数相关的约束条件则是判定这些参数取值合理与否的根据.从被测系统参数的 CDG 中能够很容易的判定参数以及约束条件之间的相关性,依据 CDG 的连通性将其划分为多个连通子图 G_1,G_2,…,每个子图中包含的参数以及约束条件之间均存在相关性,而存在于不同子图中的参数或者约束条件之间则不存在相关性.每个子图中包含的参数和约束条件就是每次生成参数取值时需要考虑的参数及约束条件的最小集合.CDG 连通子图的性质包括:

(1)对于任意子图 G_i 中的任意两点 v,v',存在路径 $P=ve_1v_1e_2v_2\cdots e_{n_i}v'$ 连接两点;

(2)对于任意两点,$v\in G_i$,$v'\in G_j$,$i\neq j$,不存在边 $e=(v,v')$;

(3)$\{v|v\in G_i\}\bigcap\{v'|v'\in G_j\}=\varnothing$,$i\neq j$.

4 约束组合测试方法

4.1 配置参数生成

本研究针对 CDG 的每个连通子图,考察其中包含的约束条件,生成约束条件的不同取值组合,利用组合约束为测试生成定义目标函数,并采用模拟退火搜索算法实现参数取值的生成.本研究仅考虑两种类型的约束条件,枚举类型的约束条件和布尔表达式类型的约束条件.枚举约束的取值将直接确定测试用例中相应参数的取值,约束条件的取值个数与参数可选的取值个数相同;布尔约束的取值个数为两个,即真和假.每个连通子图中参数的生成过程如算法 2 所示,每组约束组合的目标函数将可能存在多种定义形式,算法需要针对每种目标函数定义生成一组测试数据:

算法 2. 配置参数生成算法.

输入:系统约束依赖图

输出:测试数据

Let P be the subset of the configuration parameters

Let $C=\{c_1,c_2,\cdots,c_k\}$ be the set of all the constraints of P

Let V_i be the set of all the values of constraint c_i

Generate value combinations of C using combinatorial testing methods, such as Orthogonal Arrays, In Parameter Order, or one-row-at-a-time

Let $C_i=\{c_1=v_{i1},c_2=v_{i2},\cdots,c_k=v_{ik}\}$ be the ith value combination of all the constraints

FOR (each value combination C_i)

  Construct object functions $obj_1,obj_2,\cdots,obj_w$ for P according to C_i

  FOR (each object function obj_j)

    Generate values for P using Simulated Annealing method according to obj_j

  END FOR

END FOR

算法 2 中每组约束组合包含多条约束条件,因此其目标函数也由多个子目标函数组成,每个子目标函数对应约束组合中的一条约束条件,当子目标函数存在多种定义形式时,则总目标函数也会存在多种定义.枚举类型的约束条件直接确定参数取值,因此与其对应的子目标函数值为 0,在目标函数生成算法中着重考虑布尔类型的约束条件.对于每条布尔类型的约束条件 c_i,如果其取值为 0,则首先将其取反,即 $\neg c_i=1$,然后将 c_i 或 $\neg c_i$ 转换为等价的析取范式形式,即 $(\neg)c_i=p_1+p_2+\cdots+p_{i'}$,其中包含

i'个乘积项，因此与约束条件 $c_i/\neg c_i$ 对应的子目标函数将拥有 i' 种不同的定义，那么该约束组合也将至少有 i' 种不同的目标函数定义. 如果在某一组约束组合中，超过一条约束条件拥有多个乘积项，也就是说超过一个子目标函数拥有多种定义形式，则在定义总目标函数时需要再次利用组合测试的方法. 对于每一组约束组合，其目标函数的具体生成过程如算法 3 所示.

算法 3.　目标函数定义算法.

输入：系统约束依赖图

输出：目标函数

Let v_1 to v_k be an assignment to constraints c_1 to c_k

FOR (each Boolean constraint c_i)

　IF ($c_i=0$)

　　THEN $c_i=\neg c_i$

　END IF

　Convert c_i to its equivalent disjunctive normal form

　FOR (each term p_j in c_i)

　　Construct sub-object function obj_{ij} according to p_j

　END FOR

　obj_i has different definitions $obj_{i1}, obj_{i2}, \cdots$

END FOR

FOR (each enumerative constraint c_i)

　$obj_i=0$

END FOR

Using combinatorial testing method on $obj_1, obj_2, \cdots, obj_k$ to construct object function $obj=obj_1+obj_2+\cdots+obj_k$

下面以图 2 所示的约束依赖图为例，简单介绍利用正交试验设计方法[5]生成参数取值的过程. 图 2 中包含的约束条件及其可能的取值定义如表 1 所示，其中取值为 1 表示该约束条件为真，取值为 0 表示该约束条件为假.

表 1　约束条件取值表

可能的取值	约束条件		
	$0\leq x<800$	$0<x+l\leq 800$	$0<l\leq 800$
1	1	1	1
2	0	0	0

针对表 1，利用正交试验设计方法生成的约束组合，如表 2 所示.

表 2　正交试验设计生成的约束组合

组合号	约束条件		
	$0\leq x<800$	$0<x+l\leq 800$	$0<l\leq 800$
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

表 2 中每一行就是一组约束组合，其中第一行对应的约束组合为

$$\begin{cases}(0\leq x<800)=1\\(0<x+l\leq 800)=1\\(0<l\leq 800)=1\end{cases}.$$

该约束组合中包含 3 条约束条件，因此总目标函数定义为 $obj=obj_1+obj_2+obj_3$，由于每一条约束条件均只包含一个乘积项，因此每个子目标函数均只有一种定义方式，即

$$(1)\ obj_1=\begin{cases}0, & 0\leq x<800\\ -x, & x<0\\ x-800, & x\geq 800\end{cases};$$

$$(2)\ obj_2=\begin{cases}0, & 0<x+l\leq 800\\ -(x+l), & x+l\leq 0\\ (x+l)-800, & x+l>800\end{cases};$$

$$(3)\ obj_3=\begin{cases}0, & 0<l\leq 800\\ -l, & l\leq 0\\ l-800, & l>800\end{cases}.$$

最后基于上述目标函数定义，利用模拟退火搜索算法能够生成一组测试数据.

4.2　动作序列生成

多租户系统不仅为用户提供系统参数配置，还为用户提供系统动作行为方面的配置，用户能够根据需求配置不同的动作序列，完成不同的系统功能. 因此配置测试也应当包含动作配置测试，以确保系统在不同动作配置下能够正常运行.

从系统提供的 n 个候选动作中选取其中 n' 个动作，生成目标动作序列，能够产生 $n^{n'}$ 种不同的动作序列，这显然是难以实现的，因此本文采用正交试验设计方法从候选动作中选取部分动作生成动作序列. 为此我们定义两个集合，候选动作集合和目标动作序列集合.

(1) 候选动作集合 $Action=\{act_i \mid i=1,2,\cdots,n\}$，即系统提供的所有系统动作的集合；

(2) 目标动作序列集合 $S=\{S_i \mid i=1,2,\cdots,n'\}$，即租户对系统候选动作进行选择排列得到的动作序列的集合.

第 j 个目标动作序列 S_j 包含有 n_j 个动作，$S_j=\langle s_1, s_2, \cdots, s_{nj}\rangle$，其中，$s_{i'}=act_i$，$i'\in[1,n_j]$，$i\in[1,n]$. 在正交试验设计中，将动作序列 S_j 中的各预期动作作为试验因素，因素水平根据动作约束条件定义. 如果被测系统对起始动作或终止动作有特殊要求，则动作序列 S_j 中的起始动作 s_1 或终止动作 s_{nj} 定义为两水平因素，s_1 的两水平分别为起始动作集合

A_S以及除起始动作之外的其他动作$Action-A_S$，同理s_{nj}的两水平分别为A_E和$Action-A_E$. 动作序列中的其余动作均为三水平因素，因素水平的定义依赖于前一个动作，例如对于S_j的动作$s_{i'}$，其三水平分别定义为$After_{s_{i'-1}}$，$Parellel_{s_{i'-1}}$及$Action-After_{s_{i'-1}}-Parellel_{s_{i'-1}}$. 如果被测系统对起始动作和终止动作没有特殊要求，则起始动作为一水平因素，其水平为动作集合$Action$，终止动作的因素水平与序列中其余动作的水平定义相同. 动作序列生成的具体过程如算法4所示.

算法4. 动作序列生成算法.

输入：候选动作集合，动作约束条件

输出：目标动作序列集合

Let $Action=\{act_i \mid i=1,2,\cdots,n\}$ be the set of all the system actions

Let A_S be the set of start actions, and A_E be the set of end actions

Let $\langle s_1,s_2,\cdots,s_{n'}\rangle$ be the target action sequence

IF (A_S is not null) THEN s_1 has two levels A_s and $Action-A_s$

ELSE

s_1 has one level $Action$

END IF

FOR (each $s_i(1<i<n')$)

s_i has three levels $After_{s_{i'-1}}$, $Parellel_{s_{i'-1}}$, and $Action-After_{s_{i'-1}}-Parellel_{s_{i'-1}}$

END FOR

IF (A_E is not null) THEN $s_{n'}$ has two levels A_E and $Action-A_E$

ELSE

$s_{n'}$ has three levels $After_{s_{n'-1}}$, $Parellel_{s_{n'-1}}$, and $Action-After_{s_{n'-1}}-Parellel_{s_{n'-1}}$

END IF

Using combinatorial testing method to construct action sequence on $\langle s_1,s_2,\cdots,s_{n'}\rangle$

对起始动作和终止动作均有特殊要求的被测系统，动作序列的因素水平定义如表3所示，对于具体的因素水平定义选取合适的正交表来安排动作序列.

表3 动作序列的因素水平表

水平	试验因素					
	1	2	…	i'	…	n_j
1	A_S	$After_{s_1}$	…	$After_{s_{i'-1}}$	…	A_E
2	$Action-A_S$	$Parellel_{s_1}$	…	$Parellel_{s_{i'-1}}$	…	$Action-A_E$
3		$Action-After_{s_1}-Parellel_{s_1}$	…	$Action-After_{s_{i'-1}}-Parellel_{s_{i'-1}}$	…	

5 实验与评估

5.1 实验设置

本文选取的单人游戏系统中包含大量可配置元素，实验主要关注游戏中物体特征配置和物体行为配置，并且本实验仅讨论包含两个静止物体和两个移动物体的情况，因为这种情况已经足以表达物体与边界之间、静止物体之间、移动物体之间、以及静止物体和移动物体之间的各种关系. 游戏背景通过背景的长、宽、颜色定义，为固定值，即$L=800$，$H=600$，$Color=$Blue. 静止物体和移动物体的配置参数分别如表4和表5所示.

表4 静止物体配置参数

静止物体	横坐标	纵坐标	长度	宽度	颜色
S_1	x_{s1}	y_{s1}	l_{s1}	h_{s1}	c_{s1}
S_2	x_{s2}	y_{s2}	l_{s2}	h_{s2}	c_{s2}

表5 移动物体配置参数

移动物体	横坐标	纵坐标	长度	宽度	颜色	速度	方向
M_1	x_{m1}	y_{m1}	l_{m1}	h_{m1}	c_{m1}	s_{m1}	d_{m1}
M_2	x_{m2}	y_{m2}	l_{m2}	h_{m2}	c_{m2}	s_{m2}	d_{m2}

5.1.1 不考虑约束条件的实验设计

本实验以正交试验设计方法为例，考察在不考虑约束条件的情况下，生成测试数据的质量和效率. 传统的正交试验设计直接将参数及其可能的取值作为组合测试考察的因素，参数可能的取值必须是有限个，并且数量尽可能少. 本研究的被测系统中有多个参数的取值是某一规定范围内的正整数，虽然是有限个，但数量巨大，因此无法直接将参数取值作为组合测试的因素. 对于这类参数，本研究将取值区间作为组合测试的因素，将配置参数根据静止物体和移动物体分为两类进行测试生成. 静止物体各参数的合法取值区间如表6所示，两个静止物体相关参数的可能取值如表7所示，其中非法取值以从非法区间中选取的两个具体数值作为代表.

表6 静止物体参数合法取值区间定义

	横坐标x	纵坐标y	长度l	宽度h
合法取值区间	[0,800)	[0,600)	(0,800]	(0,600]

利用正交表$L_{12}(2^8)$安排表7中的参数及其取值，共生成12组区间组合. 在生成参数具体取值时，

表 7　两个静止物体相关参数可能的取值

	x_{s1}	y_{s1}	l_{s1}	h_{s1}	x_{s2}	y_{s2}	l_{s2}	h_{s2}
1	[0,800)	[0,600)	(0,800]	(0,600]	[0,800)	[0,600)	(0,800]	(0,600]
2	−10/810	−10/610	−10/810	−10/610	−10/810	−10/610	−10/810	−10/610

如果遇到参数取值为区间时，则从该区间随机选取一个值作为测试数据. 分别为每组区间组合生成 8 组测试数据，共为静止物体生成 96 组参数取值.

移动物体各参数的合法取值区间如表 8 所示，两个移动物体相关参数的可能取值如表 9 所示.

表 8　移动物体参数合法取值区间定义

	横坐标 x	纵坐标 y	长度 l	宽度 h	方向 d
合法取值区间	[0,800)	[0,600)	(0,800]	(0,600]	垂直、水平、其他

表 9　两个移动物体相关参数可能的取值

	x_{m1}	y_{m1}	l_{m1}	h_{m1}	d_{m1}	x_{m2}	y_{m2}	l_{m2}	h_{m2}	d_{m2}
1	[0,800)	[0,600)	(0,800]	(0,600]	垂直	[0,800)	[0,600)	(0,800]	(0,600]	垂直
2	−10/810	−10/610	−10/810	−10/610	水平	−10/810	−10/610	−10/810	−10/610	水平
3					其他					其他

利用正交表 $L_{36}(2^8\times3^2)$ 安排表 9 中的各参数及其取值，共生成 36 组区间组合，为每组区间组合生成 8 组测试数据，共为移动物体生成 288 组参数取值.

5.1.2　基于约束组合的实验设计

MTA-AGM 游戏系统对租户的配置行为有以下的约束条件限制：

(1) S_1 在边界内：$(0\le x_{s1}<800)\wedge(0\le y_{s1}<600)\wedge(0<l_{s1}\le800)\wedge(0<h_{s1}\le600)\wedge(0<x_{s1}+l_{s1}\le800)\wedge(0<y_{s1}+h_{s1}\le600)$.

(2) S_2 在边界内：$(0\le x_{s2}<800)\wedge(0\le y_{s2}<600)\wedge(0<l_{s2}\le800)\wedge(0<h_{s2}\le600)\wedge(0<x_{s2}+l_{s2}\le800)\wedge(0<y_{s2}+h_{s2}\le600)$.

(3) S_1 与 S_2 不重叠：$\forall a,b,\ 0\le a\le l_{s1},0\le b\le h_{s1},((x_{s1}+a<x_{s2})\vee(x_{s1}+a>x_{s2}+l_{s2}))\vee((y_{s1}+b<y_{s2})\vee(y_{s1}+b>y_{s2}+h_{s2}))$.

(4) M_1 在边界内：$(0\le x_{m1}<800)\wedge(0\le y_{m1}<600)\wedge(0<l_{m1}\le800)\wedge(0<h_{m1}\le600)\wedge(0<x_{m1}+l_{m1}\le800)\wedge(0<y_{m1}+h_{m1}\le600)$.

(5) M_2 在边界内：$(0\le x_{m2}<800)\wedge(0\le y_{m2}<600)\wedge(0<l_{m2}\le800)\wedge(0<h_{m2}\le600)\wedge(0<x_{m2}+l_{m2}\le800)\wedge(0<y_{m2}+h_{m2}\le600)$.

(6) M_1 与 M_2 不重叠：$\forall a,b,\ 0\le a\le l_{m1},0\le b\le h_{m1},((x_{m1}+a<x_{m2})\vee(x_{m1}+a>x_{m2}+l_{m2}))\vee((y_{m1}+b<y_{m2})\vee(y_{m1}+b>y_{m2}+h_{m2}))$.

(7) M_1 的运动方向有 3 种：水平方向、垂直方向和其他方向.

(8) M_2 的运动方向有 3 种：水平方向、垂直方向和其他方向.

(9) 由于物体颜色对本实验结果的影响非常有限，因此将所有物体的颜色均设置为白色.

约束条件 1～6 均为布尔类型约束条件，约束条件 7 和 8 为枚举类型的约束条件，分别包含 3 个可选取值，约束条件 9 为固定值约束，在测试生成时将该约束条件中定义的配置参数设定为固定值即可. 上述约束条件中包含多个配置参数，直接为这些参数生成约束依赖图将会大大增加运算复杂度，因此我们提高了约束依赖图的抽象层次，以物体为单位构造约束依赖图，如图 3 所示. 根据图的连通性将该图划分为两个子图 G_1 和 G_2，G_1 是关于静止物体的约束条件，G_2 是关于移动物体的约束条件.

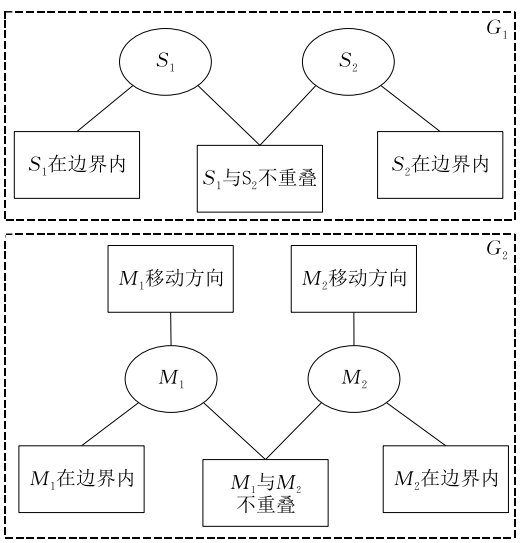


图 3　物体间的约束依赖关系图

确定约束条件以及约束条件可能的取值后即可利用组合测试方法生成约束组合，并为每组约束组合生成配置参数的具体取值.

本实验除了生成配置参数还要为物体设置

碰撞后的动作，移动物体动作集合为 $M\text{-}Action=\{verticalRevese, horizontalReverse, allReverse, M\text{-}appear, M\text{-}disappear\}$，静止物体动作集合为 $S\text{-}Action=\{S\text{-}appear, S\text{-}disappear\}$. 碰撞过程由两个物体参与完成，包括移动物体与静止物体的碰撞以及移动物体与移动物体的碰撞. 物体碰撞后的动作是同时发生的，因此这些动作均为并行执行，动作配置约束条件仅针对移动物体与移动物体的碰撞，包括：

(1) $Parellel_{Vertical}=\{M\text{-}appear, verticalReverse, allReverse, M\text{-}disappear, S\text{-}appear, S\text{-}disappear\}$.

(2) $Parellel_{Horizontal}=\{M\text{-}appear, horizontalReverse, allReverse, M\text{-}disappear, S\text{-}appear, S\text{-}disappear\}$.

由于被测系统动作数量有限，并且动作之间的约束条件简单，因此可以直接为两个移动物体构造如表 10 所示的 4 组动作组合，移动物体与静止物体碰撞后的动作进行随机配置.

表 10 移动物体动作组合

M_1	M_2
$verticalReverse$	$horizontalReverse$
$verticalReverse$	$Parallel_{Vertical}$
$horizontalReverse$	$Parallel_{Horizontal}$
其他动作	其他动作

本实验将现有的组合测试算法应用于 MTA SaaS 服务的配置测试中，直接将服务的约束条件作为算法的组合元素，为 MTA SaaS 服务生成若干配置场景，与现有方法相比，本方法主要有以下几点创新：

(1) 传统组合测试通常应用于参数的生成，而本实验则扩展了组合测试算法的应用范围，不仅利用组合测试算法生成参数，还生成了服务的动作序列；

(2) 在处理服务约束条件方面也有所不同，现有算法主要分为两种：一种是在生成参数组合前考察约束条件，避免在生成过程中产生非法的参数组合；另一种是在生成参数组合后考察约束条件，用以剔除非法参数组合. 而本实验则直接将约束条件作为组合测试算法的组合元素，参数的具体取值则是在约束组合的基础上利用模拟退火算法生成；

(3) 在生成参数取值时，本方法面临多条约束条件对参数的共同约束，因此将传统模拟退火算法的目标函数定义为多个子目标函数之和，每个子目标函数对应一条约束条件，总目标函数则对应于一组约束组合.

5.2 实验结果

5.2.1 对约束组合的覆盖比较

游戏系统中静止物体和移动物体的参数约束条件组合均通过正交试验设计、IPO 算法[6]以及 one-row-at-a-time[7]算法生成，由于静止物体配置参数数量少、约束条件简单，因此 3 种算法生成的测试数据在数量、质量上均无显著差异. 本文将着重分析上述 4 种算法为移动物体生成的配置参数对约束条件的覆盖情况. 本节将系统的约束条件概括为以下 4 类：

(1) 物体是否出界；

(2) 两移动物体是否重叠；

(3) 两移动物体的运动方向组合约束；

(4) 两移动物体的碰撞动作组合约束.

对比基于约束组合的 3 种组合测试方法和不考虑约束条件的正交试验设计方法生成的测试数据，能够得出如下结论：

(1) 传统的正交试验设计方法生成的测试数据数量最多，基于约束组合的正交试验方法次之. 在基于约束组合的 3 种测试方法中，正交试验设计方法生成的测试数据数量远远大于其他两种组合测试方法，大约是其他两种方法测试数据数量的 2.4 倍，因为正交试验设计除了要满足强度为 2 的组合覆盖要求，还要满足正交性等特有的要求. 传统正交试验设计方法生成的测试数据数量最多，但其中有效数据的数量非常有限，仅占全部测试数据的 13%.

(2) 对单一约束条件的覆盖能力基本一致. 基于约束组合的测试方法均能覆盖上述 4 种约束条件的各种取值情况，约束条件覆盖率为 100%；传统正交试验设计方法能够覆盖运动方向组合约束中的 5 种情况以及其余约束条件的所有取值情况，约束条件覆盖率约为 93%. 这 4 种方法生成的测试数据与单一约束条件的覆盖能力对比如表 11 所示.

(3) 对约束组合的覆盖能力相差较大. 基于约束组合的 3 种测试方法对约束组合的覆盖能力明显优于传统正交试验设计方法，传统正交试验设计方法仅能覆盖 50%的约束组合；基于约束组合的 3 种测试方法中，正交试验设计和 one-row-at-a-time 算法对约束组合的覆盖能力优于 IPO 算法，前两种方法均能覆盖 100%的约束组合，而 IPO 算法仅能覆盖 75%的约束组合. 约束组合的覆盖能力对比如表 12 所示，各约束组合的含义为：

表 11　对单一约束的覆盖

约束条件	测试方法	基于约束组合的测试						不考虑约束条件的组合测试	
		正交试验设计		IPO		one-row-at-a-time			
	测试数据	131		54		54		288	
	约束覆盖	数量	比率	数量	比率	数量	比率	数量	比率
1	两物体均在界内	23	17.56	8	14.81	14	25.93	2	0.69
	只有一个物体出界	72	54.96	30	55.56	24	44.44	17	5.90
	两物体均出界	36	27.48	16	29.63	16	29.63	19	6.60
2	两物体重叠	57	43.51	22	40.74	30	55.56	16	5.56
	两物体不重叠	74	56.49	32	59.26	24	44.44	22	7.64
3	(*horizontal*,*horizontal*)	11	8.40	10	18.52	4	7.41	14	4.86
	(*vertical*,*vertical*)	11	8.40	6	11.11	6	11.11	9	3.13
	(*other*,*other*)	11	8.40	12	22.22	4	7.41	10	3.47
	(*horizontal*,*vertical*)	30	22.90	22	40.74	11	20.37	0	0
	(*horizontal*,*other*)	34	25.95	2	3.70	22	40.74	3	1.04
	(*vertical*,*other*)	34	25.95	2	3.70	7	12.96	2	0.69
4	(*verticalReverse*,*horizontalReverse*)	26	19.85	14	25.93	13	24.07	2	0.69
	(*verticalReverse*,$Parallel_{Vertical}$)	23	17.56	10	18.52	11	20.37	12	4.17
	(*horizontalReverse*,$Parallel_{Horizontal}$)	23	17.56	10	18.52	11	20.37	12	4.17
	(*other*,*other*)	59	45.03	20	37.04	19	35.19	12	4.17

表 12　对约束组合的覆盖

约束组合	基于约束组合的组合测试			不考虑约束条件的组合测试
	正交试验设计	IPO	one-row-at-a-time	
1	4	0	2	0
2	5	1	3	1
3	3	4	2	0
4	5	1	3	1

① 组合 1：两物体均在界内，不重叠，运动方向组合为(*other*，*other*)、(*horizontal*，*other*)或者(*vertical*，*other*)，碰撞动作组合为(*verticalReverse*，*allReverse*)或(*horizontalReverse*，*allReverse*)；

② 组合 2：两物体均在界内，不重叠，运动方向组合为任意组合，碰撞动作组合为包含动作 *appear* 的任意动作组合；

③ 组合 3：两物体均在界内，重叠；

④ 组合 4：两物体均在界内，不重叠，运动方向组合为任意组合，碰撞动作组合为(*verticalReverse*，*horizontalReverse*)．

5.2.2　对错误检测能力的比较

针对被测系统的参数配置和行为配置，我们将配置故障进一步划分为以下 7 类，表 13 列出了 MTA-AGM 系统可能的配置故障及其在被测系统中的具体表现．

表 13　被测系统配置故障类型

故障类型	子故障类型	故障描述	被测系统具体故障描述
(1) 参数配置故障	(1.1) PF_1	违反单个配置参数的取值约束	(1.1.1) 物体横坐标超出了约定的取值范围
			(1.1.2) 物体纵坐标超出了约定的取值范围
			(1.1.3) 物体长度超出了约定的取值范围
			(1.1.4) 物体宽度超出了约定的取值范围
	(1.2) PF_2	违反多个配置参数之间的约束条件	(1.2.1) 两物体的起始位置以及长和宽的设置之间存在冲突，造成两物体重叠
			(1.2.2) 物体横坐标及长度均在合理取值范围，但其和超出边界范围
			(1.2.3) 物体纵坐标及宽度均在合理取值范围，但其和超出边界范围
	(1.3) PF_3	违反系统约束	(1.3.1) 游戏中不包含可控的移动物体
(2) 行为配置故障	(2.1) BF_1	系统将动作绑定到了错误的物体上	(2.1.1) 移动物体 *A* 的动作设置为 *verticalRerverse*，移动物体 *B* 的动作设置为 *allReverse*，两物体碰撞后，*A* 执行的动作为 *allReverse*，而 *B* 执行的动作为 *verticalReverse*
			(2.1.2) 移动物体 *A* 的动作设置为 *horizontalRerverse*，移动物体 *B* 的动作设置为 *allReverse*，两物体碰撞后，*A* 执行的动作为 *allReverse*，而 *B* 执行的动作为 *horizontalReverse*
	(2.2) BF_2	冲突的两个动作同时配置于两个发生碰撞的物体	(2.2.1) 动作 *verticalRerverse* 和 *horizontalReverse* 不能同时作为一个游戏中两个移动物体的碰撞行为
	(2.3) BF_3	违反物体行为约束	(2.3.1) 当物体碰撞面为物体的左/右侧面，但碰撞后的动作设置为 *verticalRerverse* 时，就会发生故障
			(2.3.2) 当物体碰撞面为物体的上/下侧面，但碰撞后的动作设置为 *horizontalRerverse* 时，就会发生故障
			(2.3.3) 不可控移动物体的碰撞动作设置为 *appear*，物体碰撞后运动发生故障
	(2.4) BF_4	违反系统约束	(2.4.1) 将可控移动物体的碰撞动作设置为 *disappear*，发生碰撞后游戏结束

(1)参数配置故障类型.定义如下 3 种故障类型,$\{PF_i|i=1,2,3\}$:

① PF_1违反对单个配置参数的取值约束.例如物体的坐标超出了系统对其约定的取值范围;

② PF_2违反多个配置参数之间的约束条件.例如判定物体是否出界的约束条件中就包含了多个配置参数,有物体的横坐标、纵坐标、物体长度以及物体宽度等;

③ PF_3违反系统约束.例如,要求每个游戏实例必须包含一个可控移动物体,如果游戏中不包含可控移动物体,则该游戏无法进行.

(2)行为配置故障类型.定义如下 4 种故障类型,$\{BF_i|i=1,2,3,4\}$.

① BF_1动作绑定故障.系统将动作绑定到错误的物体,例如,对于两个发生碰撞的物体 A 和 B,将 A 碰撞后的动作设置为 *verticalReverse*,B 碰撞后的动作设置为 *allReverse*,但由于配置系统故障,将动作 *verticalReverse* 错误的绑定到物体 B,而且将动作 *allReverse* 绑定到物体 A;

② BF_2两个发生碰撞的移动物体的动作配置冲突.例如,对于两个发生碰撞的移动物体 A 和 B,如果 A 的碰撞动作设置为 *verticalReverse*,则 B 的碰撞动作就不能设置为 *horizontalReverse*;

③ BF_3动作配置违反物体行为约束.例如,物体碰撞动作受到碰撞面的约束,当物体的碰撞面为左/右侧面时,动作不能设置为 *verticalReverse*,否则游戏将出现运行故障;

④ BF_4动作配置违反系统约束.例如,可控移动物体在碰撞之后的动作不允许设置为 *disappear*,否则游戏将无法继续.

仅覆盖单一约束条件的测试数据能够发现的故障类型比较简单,而复杂的故障则要利用能够覆盖约束组合的测试数据来发现.例如表 13 中的故障(2.1.1)、(2.1.2)要利用能够覆盖约束组合 1 的测试数据才能发现,故障(2.3.3)由能够覆盖约束组合 2 的测试数据发现等等.测试数据故障检测能力的差别主要由对约束组合的覆盖能力引起,基于约束组合的正交试验设计和 one-row-at-a-time(ORAT)算法均能发现上述 15 种系统故障,IPO 算法能发现 11 种系统故障,传统正交试验设计只能发现 9 种系统故障,4 种方法的故障检测能力对比如图 4 所示,分别为 100%、73%、100%以及 60%.

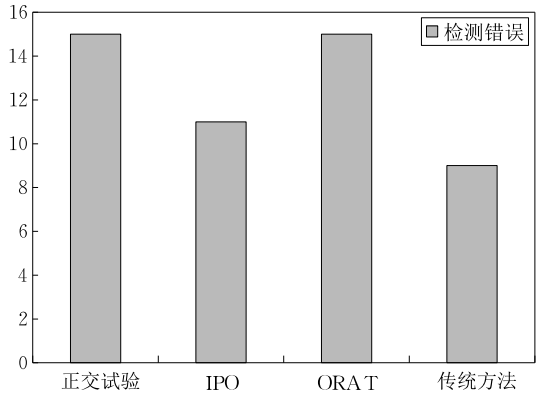


图 4 3 种组合测试检测错误能力对比

5.2.3 算法复杂度分析

被测服务约束模型中包含 k 个参数 $p_1,p_2,\cdots,p_k$以及 m 条约束条件 $c_1,c_2,\cdots,c_m$,约束条件 c_i中包含的参数个数用$|c_i|$表示.数据约束依赖图构造算法首先为每个参数和每条约束条件生成相应的节点,共生成 $m+k$ 个节点.然后为每个约束节点添加边,每个约束节点的边的数量为该约束条件中包含参数的个数,共添加$\sum_{i=1}^{m}|c_i|$条边.一条约束条件中最多包含 k 个参数,因此该算法的时间复杂度最多为 $O(m\times k)$.

正交试验设计方法的优势是正交表均已计算完成,可以直接使用,目前已有工具可以自动将确定好的实验因素、水平映射到正交表中,列出实验方案,但实验因素和水平的确定,以及正交表的选择仍需人工手动完成,难以实现自动化.正交试验设计对实验因素及其水平个数的要求也比较严格,并不是所有测试均能利用该方法.IPO 算法和 one-row-at-a-time 算法则能够实现自动化计算,对于 m 条约束条件,假设这些约束条件均为布尔类型,则每条约束条件均有 2 个可能的取值,则生成约束组合时,IPO 算法的时间复杂度为 $O(2^3m^2\log(m))$,one-row-at-a-time 算法的时间复杂度为 $O(2^4m^2\log(m))$.

对于已生成的约束组合,假设共有 M 组约束组合,每组约束组合中包含 m 条约束条件,布尔类型的约束条件 c_i中包含 λ_i个项,则每组约束组合的目标函数中将包含 m 个子目标函数,对应于约束条件 c_i的子目标函数共有 λ_i种定义方法,因此目标函数定义算法的时间复杂度最多为 $O(M\times\sum_{i=1}^{m}\lambda_i)$.

5.3 局限性分析

(1)约束的正确性、完整性和一致性

本方法的一个前提条件是能够获取相应的领域

知识，以建立被测系统正确、完整、一致的约束模型，领域知识的完整性和正确性是保证约束模型质量的基础.完整准确的约束模型能够提高测试生成的质量和效率，从而更全面的发现系统的隐藏错误.然而，由于领域知识目前存在多种不同的描述形式，并且其内容的完整性和准确性也难以保证，因此领域知识的获取和被测系统建模仍然是一个需要大量人力投入的过程，难以实现自动化，被测系统的模型质量在很大程度上依赖于测试人员的经验和技术水平，尤其对于大规模系统，此问题更加突出.

(2)生成算法的复杂度

通常情况下，系统的某个特性由多个参数共同决定，这些参数之间通常存在多种约束依赖关系，而描述不同特性的参数之间一般不存在约束依赖关系.因此，对于有大量配置参数的系统，根据参数之间是否存在约束依赖关系能够将参数分为多个参数组，属于同一组的参数之间存在约束依赖关系，而不同组的参数之间不存在约束依赖关系.这样，在生成参数取值时能够更有针对性的考察约束条件的满足情况，并且分组后，每个参数组中包含的参数数量少，约束条件相对简单，所以，能够保证参数取值的生成效率和质量.如果被测系统包含大量参数，并且这些参数之间存在错综复杂的约束依赖关系，难以分组，测试生成时需要考察的参数及约束条件的数量就会大大增加，从而提高算法的复杂度，严重影响测试生成的效率和质量.

6　相关工作

6.1　SaaS 测试技术

多租户体系架构使得 SaaS 软件的用户量有了前所未有的增长，基于庞大的用户群，SaaS 软件也具有了相当可观的规模和复杂度，同时也为 SaaS 软件测试带来了新的挑战[8].SaaS 应用软件的测试技术不同于传统的软件测试，云计算减少了软、硬件的投入，能够降低 SaaS 测试的成本；SaaS 软件测试需要实现自动化持续在线测试，并且具有一定的可扩展性，能够满足多租户需求[9].

文献[8]提出在传统软件测试领域，软件测试活动通常在所有开发活动全部结束后才开始.但是，这样的开发-测试模式显然无法满足快速发展的 SaaS 以及 MTA 的需求.因此，该文献提出一种双层数据库分区技术来支持 SaaS 定制需求，并利用元数据的约束作为测试用例来实现 SaaS 软件的持续测试.文献[9]则介绍了几种现有的 SaaS 可扩展性测试的评价标准，并提出了一种新的评价标准，结合该标准和数据挖掘技术实现了 SaaS 的可扩展性测试.

Salesforce.com 作为软件即服务(SaaS)的先驱，首先为用户提供了按需 CRM 服务.在提供按需开发服务的同时，为确保服务质量，Salesforce.com 也需要为开发者或用户提供按需服务平台的测试方法.按需服务平台的测试需要解决以下 3 个主要问题：(1)用户定制的正确性；(2)应用程序的正确性；(3)应用程序新版本的正确性.Force.com 提供了一种 Apex 测试框架，这种测试框架能够实现自动化单元测试以及自动化功能测试[10].

6.2　组合测试技术

在组合测试中，采用覆盖率标准来评价测试的完备性.对于有 n 个参数的被测服务，强度为 $t(1\leq t\leq n)$ 的覆盖要求生成的测试用例能够覆盖任意 t 个参数的所有取值组合.在定义覆盖率标准时也可以加入参数语义信息，并且在组合测试时对合法取值和非法取值区别对待[11]，根据测试需求为合法取值和非法取值分别定义覆盖率标准.

范畴(category)划分测试方法[12]针对被测系统每个能够单独测试的功能单元，将参数和环境变量的范畴划分成多个选择(choice)，通过组合这些选择为被测单元生成测试用例.分类树方法[13]是针对黑盒测试提出的一种测试方法，该方法借鉴了范畴划分方法的思想，并对其进行了改进，将测试相关信息组织成树形结构.以分类树为表头建立组合表，选择不同的叶节点组成测试用例.Cohen 等人[11]将所有参数随机排序，按照参数顺序依次为参数赋值，要求所选取值能够覆盖最多未被覆盖的组合，从而生成满足某种覆盖率要求的测试数据.IPO 算法[6]为被测服务的前两个参数生成所有取值组合，并在此基础上分别进行横向和纵向扩展生成满足两两覆盖的测试用例集.IPOG 算法[14]是对 IPO 算法的改进，能够为被测服务生成满足覆盖强度为 $t(t>2)$ 的测试用例集.

Mandl[5]首次将正交拉丁方引入软件测试领域，应用正交拉丁方为测试 Ada 编译器生成测试数据，并达到两两覆盖标准.Williams 等人[15]介绍了正交拉丁方的构造方法，并提出了正交拉丁方适用

范围的要求:(1)所有参数的取值数量相同;(2)参数之间不存在依赖关系;(3)对于相应的参数数量和取值数量存在足够的正交拉丁方用以构造测试.但在实际应用中,以上要求均难以满足,因此Williams等人通过为较少取值的参数增加重复取值以及对参数取值进行组合来扩展正交拉丁方在软件测试领域的适用范围.在正交数组的基础上,Williams[16]提出了覆盖数组的构造方法,通过对小型的正交数组进行相应的运算能够获得基础数组以及简化数组,对原始正交数组、基础数组以及简化数组进行适当的组装得到覆盖数组,覆盖数组能容纳几倍于正交数组的参数,并能够达到两两覆盖的要求.

传统组合测试的目标是尽可能多的发现由参数交互作用引起的服务失效[7],但在参数组合时却恰恰缺少对参数之间交互作用的考察和应用,因此难以保证测试生成的质量.本文没有直接利用传统的组合测试方法来生成测试数据,而是提取系统参数的约束条件,生成约束组合,并在约束组合的基础上生成参数取值.该方法在生成测试数据时目标明确,因此提高了测试生成的效率和质量.在实验设计中,采用了正交试验设计方法、IPO算法以及one-row-at-a-time算法等.

由于参数之间的约束依赖关系是测试数据生成的一个重要因素,如果在组合测试中忽略这些约束条件,则会生成大量无意义的测试数据.文献[17]利用断言(generic predicate)定义两个参数的取值组合,利用约束条件检验断言,找出不合理的取值组合.文献[18]在输入参数模型的基础上介绍了几种处理参数约束的方法,抽象参数法和子模型法都是在构建输入参数模型时去除所有可能的冲突.对于逐个选取参数取值的组合方法,文献[19]将非法的取值组合标记为已覆盖组合,从而在构建测试用例的过程中避免了对非法取值组合的选取.也可以在测试用例生成完成后,将不合法的测试用例用合法的测试用例取代.文献[20]将参数约束条件分为硬约束和软约束,该方法根据约束条件为测试用例定义优先级,优先级高的测试用例会得到优先测试的机会.现有对约束条件的处理技术均是在参数取值离散并且取值个数较少的情况下,采取一定的措施避免选择到不合理的参数取值组合,对于参数取值连续、取值数量巨大的情况,缺少相关讨论和研究.在对被测服务或软件进行鲁棒性测试时,非法的参数取值和取值组合同样会起到非常重要的作用,因此在测试数据中包含一定数量的非法数据是非常必要的.

本文利用参数之间的约束依赖关系来指导参数取值的生成,布尔表达式是这些约束条件的主要形式之一.目前在布尔表达式测试领域已有许多成熟的策略,例如多重唯一真点策略(Multiple Unique True Point,MUTP),唯一真点和邻近假点对策略(Corresponding Unique True Point and Near False Point Pair,CUTPNFT)[21],多重邻近假点策略(Multiple Near False Point,MNFP)[22]以及MUMCUT[22]策略等.在本文的实验设计中,当遇到布尔表达式类型的约束条件时,将采用MUTP策略生成相应的测试数据.

7 总 结

配置测试是验证多租户系统能否正确应对配置过程中各种突发情况的有效手段,本文通过对多租户系统的分析、研究,总结了多租户系统的故障模型,并提出了一种多约束组合的配置测试方法,对约束条件进行组合测试,并在此基础上生成满足约束条件的测试数据.实验结果表明,该方法能够有效发现参数配置以及系统行为配置方面的多种配置故障,且可从约束条件覆盖率和故障检测率两方面提高测试效率.

参 考 文 献

[1] Chong F, Carraro G. Architecture strategies for catching the long tail. MSDN Library, 2006: 9-10

[2] Tsai W T, Bai X Y, Huang Y. Software-as-a-Service (SaaS): Perspectives and challenges. Science China Information Sciences, 2014, 57(5): 1-15

[3] Wong T, Kao L, Kaufman M. Salesforce.com for Dummies. New York, USA: John Wiley & Sons, 2010

[4] Aulbach S, Grust T, Jacobs D, et al. Multi-tenant databases for software as a service: Schema-mapping techniques// Proceedings of the 2008 ACM SIGMOD International Conference on Management of data. Vancouver, Canada, 2008: 1195-1206

[5] Mandl R. Orthogonal Latin squares: An application of experiment design to compiler testing. Communications of the ACM, 1985, 28(10): 1054-1058

[6] Lei Y, Tai K C. In-parameter-order: A test generation strategy for pairwise testing//Proceedings of the 3rd IEEE International Conference on High-Assurance Systems Engineering Symposium. Maryland, USA, 1998: 254-261

[7] Nie C, Leung H. A survey of combinatorial testing. ACM Computing Surveys, 2011, 43(2): 11

[8] Tsai W T, Shao Q, Huang Y, et al. Towards a scalable and robust multi-tenancy SaaS//Proceedings of the 2nd Asia-Pacific Symposium on Internetware. Suzhou, China, 2010: 8

[9] Tsai W T, Huang Y, Shao Q. Testing the scalability of SaaS applications//Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA). California, USA, 2011: 1-4

[10] Mathew R, Spraetz R. Test automation on a SaaS platform //Proceedings of the Software Testing Verification and Validation(ICST'09). Denver, USA, 2009: 317-325

[11] Cohen D M, Dalal S R, Fredman M L, et al. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 1997, 23(7): 437-444

[12] Ostrand T J, Balcer M J. The category-partition method for specifying and generating functional tests. Communications of the ACM, 1988, 31(6): 676-686

[13] Grochtmann M. Test case design using classification trees// Proceedings of the STAR. Washington D. C., USA, 1994, 94: 93-117

[14] Lei Y, Kacker R, Kuhn D R, et al. IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing. Software Testing, Verification and Reliability, 2008, 18(3): 125-148

[15] Williams A W, Probert R L. A practical strategy for testing pair-wise coverage of network interfaces//Proceedings of the Seventh International Symposium on Software Reliability Engineering. New York, USA, 1996: 246-254

[16] Williams A W. Determination of test configurations for pair-wise interaction coverage//Wu Weili, Daescu O eds. Testing of Communicating Systems. Springer USA, 2000: 59-74

[17] Calvagna A, Gargantini A. A logic-based approach to combinatorial testing with constraints//Beckert B, Hahnle R eds. Tests and proofs. Springer Berlin Heidelberg, 2008: 66-83

[18] Grindal M, Offutt J, Melin J. Handling constraints in the input space when using combination strategies for software testing. University of Skovde, Stockholm, Sweden: Technical Report HS-IKI-TR-06-001, 2006

[19] Cohen M B, Dwyer M B, Shi J. Interaction testing of highly-configurable systems in the presence of constraints//Proceedings of the 2007 International Symposium on Software Testing and Analysis. London, UK, 2007: 129-139

[20] Bryce R C, Colbourn C J. Prioritized interaction testing for pair-wise coverage with seeding and constraints. Information and Software Technology, 2006, 48(10): 960-970

[21] Chen T Y, Lau M F. Two test data selection strategies towards testing of Boolean specifications//Proceedings of the Computer Software and Applications Conference. Washington DC, USA, 1997: 608-611

[22] Chen T Y, Lau M F. Test case selection strategies based on Boolean specifications. Software Testing, Verification and Reliability, 2001, 11(3): 165-180

[23] Chen T Y, Lau M F, Yu Y T. MUMCUT: A fault-based strategy for testing Boolean specifications//Proceedings of the 6th Asia-Pacific Software Engineering Conference. Kawasaki, Japan, 1999: 606-613

HOU Ke-Jia, born in 1983, Ph. D. candidate. Her research interest is software testing.

BAI Xiao-Ying, born in 1973, Ph. D., associate professor. Her research interests include software testing and service computing.

ZHOU Li-Zhu, born in 1947, professor, Ph. D. supervisor. His research interests include database system, digital library and massive information processing.

Background

Software-as-a-Service (SaaS) promotes a new architecture style for building domain-specific applications in a tenant-based approach. Multiple tenants can share the same database and code base which are deployed on a cloud infrastructure and maintained by service providers. Tenant applications are isolated and customized to their individual requirements. Hence, even though they share the hardware and software resources, each customer feels like he owns a dedicated system. Tenants are charged by real usage of resources and services following a pay-per-use approach.

SaaS is a promising technique to reduce both time and cost for developing and maintaining tenants applications.

The sharing/isolation mechanisms may introduce risks to system quality, performance, and security. It thus needs additional efforts and techniques to validate system behavior under various tenants' scenarios.

Massive configurability is a key feature of SaaS applications. Hence, SaaS configuration testing faces many difficulties: the number of possible scenarios is huge, and it is still lack of built-in testability for system configuration testing.

The design of test scenarios is hard to cover different combinations of configurable artifacts with constrains considered. Many techniques have been proposed to address similar issues in unit testing, such as various algorithms for combinatorial testing. However, practical application of the algorithms at the level of system testing requires a re-formulation of the problem in proper scale.

In response to these problems, the paper proposes a new approach for SaaS configuration testing which derives test scenarios based on the combination of constraints of configurable parameters. The paper defines constraints dependency graph (CDG) as a tool for analyzing constraints and develops the algorithms to search for test scenarios based on CDG.

This work is supported in part by a grant from the National Natural Science Foundation of China (61073003), the Beijing Natural Science Foundation (4132062), the National Grand Fundamental Research 973 Program of China (2011CB302505), and the National High-Tech Research and Development Plan of China (863 Program) (2013AA01A215).