一种基于动态规划的云环境测试配置方法

周欢欢 姜 瑛
(云南省计算机技术应用重点实验室 昆明 650500)
(昆明理工大学信息工程与自动化学院 昆明 650500)

摘 要 为了得到测试代价最低的资源配置组合方案,简要介绍了测试配置对软件测试质量和效率的影响。通过分析当前云平台中松散耦合、动态变化可用资源的特点,提出了云环境下测试配置的过程、测试需求和测试配置的结构,并提出了基于动态规划的测试资源选择算法以满足用户的测试需求。最后在CloudSim下通过相关实验验证了方法的可行性。

关键词 云环境,测试配置,测试需求,动态规划,测试资源选择
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.040

Test Configuration Method Based on Dynamic Programming under Cloud Environment

ZHOU Huan-huan JIANG Ying
(Yunnan Key Laboratory of Computer Technology Application, Kunming 650500, China)
(Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China)

Abstract In order to obtain the lowest test cost scheme of the resource configuration combination, this paper briefly described the influence of test configuration on the test quality and efficiency. Through analyzing the characteristics of loose coupling and dynamic changing available resources of cloud platform, this paper proposed the process of test configuration, structure of test requirement and test configuration. In order to meet the users' test requirements, a method based on dynamic programming was proposed to select available test resource. Finally, the feasibility of the method was verified by the relevant experiments in CloudSim.

Keywords Cloud environment, Test configuration, Test requirement, Dynamic programming, Test resources selection

1 引言

软件测试的首要工作就是配置测试环境,测试配置是根据测试需求中提出的环境、价格、时间等来搭建满足条件的测试环境。由于测试执行前需要进行充分的准备,因此要求测试配置是科学的、规范的,有效的测试配置可以提高测试的质量和效率[1]。随着软件运行环境多样性、测试软件兼容性的出现,软件测试环境的构建变得较为复杂和频繁,环境的配置和部署所花费的时间和代价相当大,增加了测试配置的成本。云计算已经成为为应用提供灵活、按需分配、动态且可扩展的计算基础设施的解决方案。云环境是整个开放云平台的核心,它给基于开放云平台的云应用提供执行环境,并帮助开发者构建和维护云应用。云环境中整合了大量的虚拟资源,具有分布式、松散耦合、动态变化等特点,可以为软件测试提供大量可用的资源。为了在云环境中顺利完成软件测试,首先需要找到能够完成测试任务的资源,如何基于测试需求以最低的成本进行测试配置是亟需解决的问题,云计算环境的特点给软件测试配置提出了新的要求。

目前很多学者对测试资源配置开展了相关研究。郑军等人提出了具有3层软件体系结构的仿真测试环境,通过基于任务的测试仿真模型和测试配置的蓝图,解决了实时测试数据的激励、收集和动态测试配置[2]。王岩等人分析了自动测试系统中测试资源的优化设计存在的问题,给出了设计和研制自动测试系统应遵循的基本原则[3]。方永甲等人针对多UUT(Unit Under Test)并行测试任务调度与资源配置问题,提出了一种遗传蚁群融合算法,得出最短并行测试时间基础上的最少资源需求[4]。陈粤等人分析了测试资源与测试任务的约束关系,借助遗传算法的全局启发式优化能力,可以计算出给定UUT所需的测试资源配置的总体效益最优解[5]。周德新等人引入匹配系数的概念,提出了一种面向信号的测试资源动态配置方法,取得了一定的效果[6]。另外,金天等人针对如何实现准确提供动态、多样的试验环境需求问题,采用Web本体语言OWL(Web Ontology Language)描述课程实验本体和云资源领域本体,提出一种基于本体匹配技术的资源自动分配方法,该方法能够有效地实现并行测试,提高了测试的效率和系统的可靠性[7]。

Lian Yu和Wei-Tek Tsai等人针对云环境中非均匀的并发请求,指出用户的测试请求要与TaaS(Testing as a Service)平台的能力进行匹配,采用k-中心点的快速集群算法对用户的需求进行集群,并开发一种启发式的调度算法动态地把这些测

到稿日期:2013-11-06 返修日期:2014-01-30 本文受国家自然科学基金项目(60703116,61063006)资助。
周欢欢(1989-),女,硕士,主要研究方向为软件测试、云计算,E-mail:zhh_0405@163.com;姜 瑛(1974-),女,博士,教授,主要研究方向为软件工程、软件测试、服务计算,E-mail:jy_910@163.com(通信作者)。

试任务分配到虚拟机上[8]。Olumide Akerele 等人给出了测试任务迁移到云中的方法，同时设计出一种独特的云测试环境模型，该模型可以帮助管理者在云环境中制定测试策略，提高软件质量，降低测试代价[9]。Takayuki Banzai 和 Hitoshi Koizumi 等人使用云计算技术设计出一种软件测试环境，利用 Eucalyptus 将测试者导入的测试配置文件上传到分布式云计算系统的前端，云计算系统根据导入的测试配置文件自动"搜索"可用的节点，该方法使得测试者能够很快地对分布式系统进行配置和测试，有效降低了测试代价并减少了时间[10]。

现有研究大多针对特定领域系统进行测试配置，且主要针对已知测试环境进行优化配置，普遍存在资金重复投入大、资源利用率低等问题。部分研究使用测试者定义的配置来寻找可用资源，并未考虑测试成本对测试配置的影响，也无法处理云环境中资源与测试需求动态变化的情形。因此，由于云平台中资源松散耦合以及动态变化的特点，使得软件测试配置需要以代价为核心，灵活应对测试资源多变的环境。为了选择最优的资源配置组合方案，本文提出一种基于动态规划的云环境测试配置方法。该方法首先通过分析云环境下测试配置的过程，定义测试需求以及测试配置的结构；然后根据用户提出的测试需求，利用动态规划算法生成当前可用的、满足条件的、成本最低的资源配置方案。

2 测试配置过程

在测试执行前，首先要明确测试的需求，本文的测试需求主要包括测试对象、测试时间、测试经费、测试类型以及测试环境。云环境中的可用资源是动态变化的，并且测试需求也在不断地改变，基于这种情况采用动态规划算法可以找到满足要求的测试配置。

测试配置生成的动态规划过程如图 1 所示，资源选择是根据测试需求对资源进行选择的过程。在某类资源有多个的情况下，需要通过决策对资源进行选择。一旦选定资源，当前的资源选择状态就发生转移，并产生相应的测试配置。

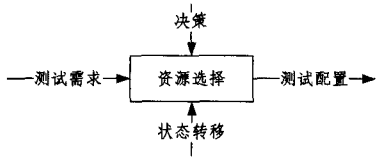

图 1 测试配置生成的动态规划过程

3 测试需求与测试配置结构

测试需求是进行软件测试的基础，既描述了测试的相关信息，又为分配资源以及确定某个阶段测试工作是否完成提供了相应的基础。XML(Extensible Markup Language)由于具有良好的数据存储格式、可扩展性、高度的结构化、精确的数据搜索等优点，能够被不同类型的程序所读取。为了适应云环境中资源动态变化的特点，本文采用 XML 来定义测试需求。

测试需求需要定义测试范围、明确测试目的和内容，因此本文定义的测试对象、测试时间、测试经费、测试类型以及测试环境如图 2 所示。测试对象是待测试软件的内容，测试时间和测试经费是用户给出的时间和经费需求，测试类型包括功能、性能、技术(黑盒、白盒和灰盒)、阶段(单元测试、集成测试和系统测试)和模式(动态测试、静态测试)等。由于云环境中资源的多样性，本文加入了测试环境的需求，包括操作系统、CPU 个数、内存、硬盘和带宽需求，以便于在复杂多变的云环境中找到最合适的测试资源。

```
〈userRequirements〉
  〈testObject〉
    〈name〉〈/name〉
    〈testTime〉〈/testTime〉
    〈testMoney〉〈/testMoney〉
    〈testType〉
      〈function〉〈/function〉
      〈performance〉〈/performance〉
      〈technology〉〈/technology〉
      〈stage〉〈/stage〉
      〈mode〉〈/mode〉
      〈/testType〉
    〈testEnvironment〉
        〈os〉〈/os〉
        〈numberOfCpu〉〈/numberOfCpu〉
        〈ram〉〈/ram〉
        〈storage〉〈/storage〉
      〈bw〉〈/bw〉
    〈/testEnvironment〉
  〈/testObject〉
〈/userRequirements〉
```

图 2 测试需求文档 XML 结构

测试配置是对测试的软硬件环境进行部署，是测试实施的一个重要阶段，测试配置适合与否会直接影响测试结果的真实性和有效性。本文的测试配置是根据测试需求生成的。为了使定义出来的测试配置结构具有通用性，测试配置中以测试对象为单位，包含了测试对象名称、测试时间、测试经费、测试类型和云环境中的资源选择方案。云环境中的某些资源之间存在包含关系，测试配置会根据测试需求中的环境生成对应的内容，具体内容如图 3 所示。

```
〈testConfiguration〉
  〈name〉〈/name〉
  〈testTime〉〈/testTime〉
  〈testMoney〉〈/testMoney〉
  〈testType〉〈/testType〉
  〈resource〉
    〈datacenter id=" "〉
      〈os〉〈/os〉
      〈host〉
        〈vm id=" "〉
          〈mips〉〈/mips〉
          〈ram〉〈/ram〉
          〈storage〉〈/storage〉
          〈bw〉〈/bw〉
          〈numberOfCpu〉〈/numberOfCpu〉
          〈cloudlet id=" "〉
            〈fileSize〉〈/fileSize〉
          〈/cloudlet〉
        〈/vm〉
      〈/host〉
    〈/datacenter〉
  〈/resource〉
〈/testConfiguration〉
```

图 3 测试配置文档 XML 结构

4 基于动态规划的云环境测试资源选择算法

为了提高云环境中测试任务的执行效率，降低测试执行的成本，需要在云环境的测试配置中为测试任务选择满足要求的测试资源。由于云环境中存在着大量资源，按需采用“即用即付费”的方式分配计算、存储和带宽资源，因此用户可以按需向计算平台提交自己的硬件配置、软件安装、数据访问需求，平台根据用户需求、资源需求和当前系统的可用资源灵活地分配和调度任务。用户需要为自己实际使用的计算能力、存储空间和网络带宽付费，因此代价最低是云用户追求的首要目标。基于云环境下资源动态变化的特点，本文使用动态规划的方法来选择测试资源。动态规划是用来解决多阶段决策过程最优化的一种数量方法[11]。针对用户需求不断变更的现状，如果对云环境下的所有资源进行遍历，需要耗费较多的时间和效率。动态规划算法可以对当前符合条件的资源进行“剪枝”，以降低计算复杂度和资源使用代价，提高资源分配的效率。

4.1 动态规划模型的建立

云环境中有多种资源，主要的资源类别为数据中心、主机、虚拟机以及其对应的应用服务程序，每类资源的数量可能有多个。由于云环境按需进行计算、存储和带宽等各类资源的分配，再具体到虚拟机、应用服务程序及其使用数量，在对资源方案成本进行计算时，最先搜索的是底层资源，因此整个动态规划过程是逆推进行的。假设云环境中有的资源种类为 $n(n>1)$，为了与图 1 提出来的测试配置生成的动态规划过程对应，本文将测试资源选择的动态规划模型定义为：

(1)阶段变量 k：根据资源的种类将资源选择划分为 k 个阶段，$k=n,n-1,\cdots,1$。

(2)S 为状态空间，第 k 段状态集合为 S_k，其元素 s_k 表示在第 k 阶段初尚未分配的资源价格，即 $s_k \in S_k$。

(3)U 为决策空间，第 k 段允许决策集合为 $D_k(s_k)$，其元素 $u_k(s_k)$ 表示第 k 阶段从 s_k 出发所作的决策，即 $u_k(s_k) \in D_k(s_k) \subset U$。

(4)状态转移方程 $s_k = s_{k-1} - u_k(s_k)$。

(5)$d_k(s_k,u_k)$ 为第 k 段处于状态 s_k 采取决策 $u_k(s_k)$ 时所消耗的代价，即代价函数。

由此可得到总代价的递推公式为：

$$f_k(s_k) = \min_{s_k \in D_k(s_k)} \{d_k(s_k,u_k) + f_{k+1}(s_{k+1})\} \quad (1)$$

$$f_1(s_1) = \min_{s_1} d_1(s_1,u_1) \quad (2)$$

对应于 $f_k(s_k)$ 的决策即为阶段 k 的最优决策，其中，$k=n,n-1,\cdots,1$。

4.2 动态规划算法

在云环境中，可用的资源数量很庞大。由于遍历每一种资源方案将耗费大量的时间和代价，本文采用 *resourceUsed*[k][u_k]对资源的使用情况进行标识，即第 k 段采用决策 u_k，为 0 表示该资源未使用，反之则已被占用。

为了降低计算复杂度，减少方案搜索时间，在使用动态规划算法进行测试资源搜索的过程中，如果某类或者几类资源的价格高于现阶段代价最低的资源组合方案，则直接舍弃该方案，进行下一方案的搜索，直至搜索到一种可用的、代价最低的资源分配组合方案，这样可以使整个测试资源的搜索过程效率更高。此外，云环境下可能会有多个测试对象，为了体现公平，在大于两个的测试对象要求的测试配置相同时，本文采用先到先服务的原则，让第一个测试对象获得最优的测试配置。

基于以上分析，本文提出了云环境下的测试资源选择算法，如图 4 所示。

```
Input:测试需求 XML 文档
Output:min_Cost and resultSource[i][k]
{
  i=1;j=1;k=n;//初始化
  min_Cost=0;//测试对象成本
  min_csdx_Cost=∞;//当前测试对象的最小测试成本
  resourceUsed[k][uk]=0;//资源使用标识
  resultSource[i][k]=0;//资源分配方案
  For each xi(i=1,2,…){//计算每个测试对象的测试资源分配方案成本
    While (uk≠0){//资源组合方案数不为 0
      If (resourceUsed[k][uk]==0){
        fk(sk)=fk+1(sk+1)+dk(sk,uk);
        If (fk(sk)≥min_csdx_Cost){
        uk=uk+1;
        }
      }
    }
    resultSource[i][k]=k;
    resourceUsed[k][uk]=1;
    k=k+1;
    If (fk(sk)<min_csdx_Cost){
      min_csdx_Cost=fk(sk);
    }
    min_Cost=min_Cost+min_csdx_Cost;
    }
}
```

图 4 基于动态规划的云环境测试资源选择算法

5 实验

为了验证本文提出的基于动态规划云环境资源配置方法的可行性，本文选择 CloudSim 作为云环境实验平台。CloudSim[12-14] 中提供了大量的可用资源，主要有数据中心、主机、虚拟机以及 Cloudlet，其中 Cloudlet 类似于应用服务程序。根据用户提供的测试需求文档，在 CloudSim 中依据已有的资源使用第 4 节中的动态规划算法，从中选出一种能够满足用户需求、测试代价最低的资源组合方案，从而生成最终的测试配置文档。

本文通过仿真平台进行了全遍历方式和动态规划方式下的云环境测试资源选择实验。根据测试需求在当前云环境中寻找符合要求的资源，输出符合要求的资源分配方案及该方案的最低费用。本文实验共创建了 6 个数据中心、3 台主机、10 个虚拟机、16 个应用服务程序(Cloudlet)，即对应动态规划算法中的决策集分别为 $D_4=\{u_1,u_2,\cdots,u_{16}\}$、$D_3=\{u_1,u_2,\cdots,u_{10}\}$、$D_2=\{u_1,u_2,u_3\}$、$D_1=\{u_1,u_2,\cdots,u_6\}$，具体的资源情况如表 1 所列。

表 1　CloudSim 中生成的可用资源

序号	资源状态	数据中心编号	主机编号	虚拟机编号	Cloudlet 编号
1	SUCCESS	2	1	1	1
2	SUCCESS	2	1	1	2
3	SUCCESS	2	2	2	4
4	SUCCESS	5	3	10	16
5	SUCCESS	2	3	2	5
6	SUCCESS	2	1	1	3
7	SUCCESS	2	3	2	6
8	SUCCESS	3	2	8	14
9	SUCCESS	5	2	6	13
10	SUCCESS	4	3	9	15
11	SUCCESS	3	3	4	9
12	SUCCESS	3	1	4	10
13	SUCCESS	4	2	5	11
14	SUCCESS	4	2	5	12
15	SUCCESS	2	3	3	7
16	SUCCESS	2	3	3	8

本实验中，假设有 4 个测试对象：Object1、Object2、Object3 和 Object4，Object1 的环境需求为 Windows xp 操作系统、1 个 CPU、512M 内存、1G 硬盘以及 100MB 带宽；Object2 的环境需求与 Object1 相同。Object3 的需求为 Windows xp 操作系统、2 个 CPU、256M 内存、2G 硬盘以及 100MB 带宽。Object4 的需求为 Linux 操作系统、2 个 CPU、1024M 内存、1.2G硬盘以及 100MB 带宽。在云环境下对以上 4 个测试对象搜索其可用的资源方案，得到测试对象的可用资源组合方案，分别如表 2－表 5 所列。

表 2　测试对象 Object1 可用的资源组合方案

序号	数据中心编号	主机编号	虚拟机编号	Cloudlet 编号	费用
1	2	1	1	1	1182.4
2	2	1	1	2	1262.4
3	2	1	1	3	1342.4
4	2	3	2	4	1622.4
5	2	3	2	5	1702.4
6	2	3	2	6	1782.4
7	2	3	3	7	2764.8
8	2	3	3	8	2844.8
9	5	2	6	13	3844.8

表 3　测试对象 Object2 可用的资源组合方案

序号	数据中心编号	主机编号	虚拟机编号	Cloudlet 编号	费用
1	2	1	1	2	1262.4
2	2	1	1	3	1342.4
3	2	3	2	4	1622.4
4	2	3	2	5	1702.4
5	2	3	2	6	1782.4
6	2	3	3	7	2764.8
7	2	3	3	8	2844.8
8	5	2	6	13	3844.8

表 4　测试对象 Object3 可用的资源组合方案

序号	数据中心编号	主机编号	虚拟机编号	Cloudlet 编号	费用
1	2	3	3	7	2764.8
2	2	3	3	8	2844.8
3	5	2	6	13	3844.8

表 5　测试对象 Object4 可用的资源组合方案

序号	数据中心编号	主机编号	虚拟机编号	Cloudlet 编号	费用
1	2	3	4	9	3329.6
2	2	1	4	10	3409.6
3	5	2	8	14	3358.8

按照代价最低原则，测试对象 Object1、Object2、Object3、Objcet4 选取方案均为各自可用资源组合方案里的第 1 个。由表 2 和表 3 可以看出，当两个测试对象的环境需求相同时，根据先来先服务的原则，Object1 会搜索当前环境下的最优测试配置方案，Object2 会搜索次优的测试配置方案。

此外，本文计算了使用全遍历方式和动态规划方式选择测试对象资源方案的时间，4 个测试对象所用的时间如表 6 所列。

表 6　测试对象资源方案选择的时间

	Object1(ms)	Object2(ms)	Object3(ms)	Object4(ms)
全遍历	20	19	15	15
动态规划	15	16	12	11

根据表 6 的结果，针对 4 个测试对象的资源方案选择，本文提出的动态规划方式所耗时间比全遍历方式分别减少了 25%、15.8%、20%和 26.7%。可见，基于云环境的动态规划测试资源选择算法可以生成满足条件的、代价最低的资源配置组合方案。

结束语　为了在云环境中选出代价最低的资源组合方案，本文采用动态规划的思想，根据测试需求对云环境中现有的资源进行动态规划，生成最终的测试配置。本文方法能够用较少的时间得到代价最低的测试资源组合方案。下一步我们将针对云环境下的实际情况，扩展测试需求和测试配置的内容，进一步提高云测试配置的合理性。

参 考 文 献

[1] 邵维忠，杨芙清. 面向对象的系统分析(第 2 版)[M]. 北京：清华大学出版社，2006

[2] 郑军，刘畅，任占勇. 综合模块化航空电子软件测试环境[J]. 计算机工程与设计，2011，32(8)：2737-2741

[3] 王岩，应朝龙，刘军. ATS 测试资源优化配置设计[J]. 微计算机信息，2009，25(8)：228-230

[4] 方甲永，肖明清，谢娟. 基于遗传蚁群算法的并行测试任务调度与资源配置[J]. 测试技术学报，2009，23(4)：333-339

[5] 陈粤，边泽强，孟晓风. 基于信号参数集最小距离的并行测试任务调度算法[J]. 系统仿真学报，2006，18(9)：2409-2411

[6] 周德新，祁鹏，刘涛. 机载 REU 并行测试系统资源配置优化算法研究[C]//中国智能自动化会议论文集. 2009：252-257

[7] 金天，李昕. 一种基于最优匹配的测试资源动态配置方法[J]. 信息工程大学学报，2010，11(3)：322-326

[8] Yu Lian，Tsai W-T，et al. Testing as a Service over Cloud [C]// IEEE International Symposium on Service Oriented System Engineering. 2010：181-188

[9] Akerele O，Ramachandran M，Dixon M. Testing in the Cloud：Strategies，Risks and Benefits [M] // Software Engineering Frameworks for the Cloud Computing Paradigm. 2013：167

[10] Banzai T，Koizumi H，Kanbayashi R. Design of a Software Testing Environment for Reliable Distributed System Using Cloud Computing Technology[C]// International Conference on Cluster，Cloud and Grid Computing. 2010：631-636

[11] 胡运权，郭耀煌. 运筹学教程(第三版)[M]. 北京：清华大学出版社，2007

[12] Greenberg A，Hanilton J R，Jain N. VL2：A Scalable and Flexible Data Center Network [C]// Proceedings of the ACM SIG-

COMM 2009 conference on Data communication. 2009:51-62
[13] Garg S K, Ya R B. Network CloudSim: Modelling Parallel Application in Cloud Simulations [C]// Fourth IEEE International Conference on Utility and Cloud Computing. 2011:105-113
[14] Buyya R, Ranjan R, Calheiros R N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities[C]// International Conference on High Performance Computing & Simulation. 2009:1-11

(上接第 209 页)

正确系统的实验结果。

表 1 系统含有逻辑错误的实验结果

属性	步数	结果	时间(秒)
reachability1	22	unsat	2.23
reachability1	23	sat	2.31
reachability2	20	unsat	2.12
reachability2	21	sat	2.21
static1	24	unsat	2.45
static1	25	sat	2.48
static2	25	unsat	2.66
static2	26	sat	2.74
dynamic	23	unsat	2.34
dynamic	24	sat	2.47

表 2 系统错误改正实验结果

属性	步数	结果	时间(秒)
reachability1	50	unsat	4.45
reachability2	50	unsat	4.52
static1	50	unsat	4.68
static2	50	unsat	4.49
dynamic	50	unsat	4.78

如果实验结果输出为“unsat”，则增加有界模型检测的步数 k，直到输出结果为“sat”或达到预先设定的检测步数最大值 K。如果满足，说明在 k 步之内找到了一个反例，该公式模型(包括对变量的解释)便检测出系统设计的漏洞或错误。模型检测工具可以返回一个反例，通过对反例的解读可以得到性质不成立的原因，为系统的修正提供重要线索。反之，如果 SMT 实例不可满足，则表明受验证的系统或模型运行到 k 阶段时是安全的、没有错误的。

结束语 本文提出了一种基于时间 Petri 网的嵌入式系统中断行为建模方法，该方法能够对中断嵌套和并发行为进行有效描述。此外，为了通过模型检测技术对中断模型进行验证，提出了一种从时间 Petri 网模型到时间自动机模型的转化方法。在模型检测过程中，提出了一种把时间自动机转化为一阶逻辑公式的符号编码方法，最后通过 SMT 模型检测方法对其进行有界模型检测。根据不同的建模粒度，验证的属性是不同的。

本文的研究仍然存在一些细节工作有待提高，比如在模型编码和验证的过程中，我们转化的 BMC 公式并不是最优的，同时，我们也没有为时间 Petri 网模型到时间自动机模型的转化提供严格的数学证明，在未来的工作中我们会进行更加深入的研究。

参 考 文 献

[1] Rammig F, Rust C. Modeling of dynamically modifiable embedded real-time systems [C] // The Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2003(WORDS 2003 Fall). IEEE, 2003:28-28
[2] Gu Z, Shin K G. An integrated approach to modeling and analysis of embedded real-time systems based on timed petri nets[C]// Proceedings 23rd International Conference on Distributed Computing Systems, 2003. IEEE, 2003:350-359
[3] Costa A, Gomes L. Petri net splitting operation within embedded systems co-design[C]// 2007 5th IEEE International Conference on Industrial Informatics. IEEE, 2007, 1:503-508
[4] Zhang H, Ai Y. Schedule modeling based on Petri nets for distributedreal-time embedded systems [J]. Jisuanji Gongcheng/Computer Engineering, 2006, 32(18):6-8
[5] Merlin P M. A study of the recoverability of computing systems [D]. University of California, Irvine, 1974
[6] Cassez F, Roux O H. Structural translation from time Petri nets to timed automata[J]. Journal of Systems and Software, 2006, 79(10):1456-1468
[7] Lime D, Roux O H. Model checking of time Petri nets using the state class timed automaton[J]. Discrete Event Dynamic Systems, 2006, 16(2):179-205
[8] Guoyin Z, Ming L, Aihong Y, et al. Methodology of modeling and formal verification based on extended Petri net[J]. Applicaiton Research of Computers, 2010, 27
[9] Alur R. Timed automata [C] // Computer Aided Verification. Springer Berlin Heidelberg, 1999:8-22
[10] Barrett C, Sebastiani R, Seshia S, et al. Handbook of Satisfiability [M]. Fairfax: IOS Press, 2009
[11] Biere A, Cimatti A, Clarke E, et al. Symbolic model checking without BDDs[M]. Springer Berlin Heidelberg, 1999
[12] Veanes M, Bjørner N, Raschke A. An SMT approach to bounded reachability analysis of model programs [M] // Formal Techniques for Networked and Distributed Systems-FORTE 2008. Springer Berlin Heidelberg, 2008:53-68
[13] Wang Xiao-liang. Bounded model checking of timed automata based on Yices[J]. Computer Engineering and Design, 2010, 31(1):126-129
[14] Weiqiang K, Shiraishi T, Katahira N, et al. An SMT-Based Approach to Bounded Model Checking of Designs in State Transition Matrix[J]. IEICE transactions on information and systems, 2011, 94(5):946-957
[15] Barrett C, De Moura L, Stump A. Design and results of the first satisfiability modulo theories competition (SMT-COMP 2005) [J]. Journal of Automated Reasoning, 2005, 35(4):373-390
[16] Le Berre D, Simon L. The essentials of the SAT 2003 competition[C] // Theory and Applications of Satisfiability Testing. Springer Berlin Heidelberg, 2004:452-467