软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(5):1314−1331 [doi: 10.13328/j.cnki.jos.005955]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

Web 应用前后端融合的遗传算法并行化测试用例生成*

王微微, 李奕超, 赵瑞莲, 李 征

(北京化工大学 信息科学与技术学院,北京 100029)

通讯作者: 赵瑞莲, E-mail: rlzhao@mail.buct.edu.cn

摘 要: Web应用测试用例生成并行化是提升Web应用测试生成效率的一个有效手段.Web应用的前后端分离、事件驱动等特性,导致传统的并行化技术难以直接应用于 Web 应用的测试用例自动生成中.因此,如何针对 Web 应用进行并行化测试用例生成,是一项具有挑战性的工作.将种群并行化计算引入到基于遗传算法的 Web 应用前后端融合的测试用例生成中,通过线程池及调度逻辑设计、多浏览器进程管理及后端覆盖路径获取,实现种群个体在多浏览器上的并行化执行及基于后端路径覆盖的适应度值并行化计算,以更高效地生成 Web 应用的测试用例.实验结果表明:相对于 Web 应用的 GA 串行化测试用例生成方法,所提的并行化测试生成方法能够更充分地利用系统资源,极大地提升 Web 应用测试用例的生成效率.

关键词: Web 应用测试;测试用例生成;遗传算法;并行化;敏感路径

中图法分类号: TP311

中文引用格式: 王微微,李奕超,赵瑞莲,李征.Web 应用前后端融合的遗传算法并行化测试用例生成.软件学报,2020,31(5):1314−1331. http://www.jos.org.cn/1000-9825/5955.htm

英文引用格式: Wang WW, Li YC, Zhao RL, Li Z. Parallel test case generation based on front and back end of Web applications with genetic algorithm. Ruan Jian Xue Bao/Journal of Software, 2020,31(5):1314−1331 (in Chinese). http://www.jos.org.cn/1000-9825/5955.htm

Parallel Test Case Generation Based on Front and Back End of Web Applications with Genetic Algorithm

WANG Wei-Wei, LI Yi-Chao, ZHAO Rui-Lian, LI Zheng

(School of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: Parallelization of test case generation for Web applications is an effective way to improve the efficiency of test generation. Due to the characteristics of front-back end separation and event-driven of Web applications, the traditional parallelization technology is difficult to be applied to automatic test case generation of Web applications. Therefore, it becomes a challenging task to parallelize test case generation for Web applications. In this study, parallelized computing is introduced into the test case generation for Web applications based on GA. By means of the design of thread pool and scheduling logic, the management of the multi-browser process and the acquisition of the path coverage of back end code, the parallel execution of individuals on multiple browsers and the parallel computation of fitness values based on the back end path coverage are realized, making test case generation more efficiently. The experiment results show that compared with GA serialization test case generation, the proposed parallelization method can make full use of system resources and greatly improve the efficiency of test case generation for Web applications.

Key words: Web application testing; test case generation; genetic algorithms; parallelization; sensitive path

随着Web技术的普及与快速发展,Web应用越来越广泛.然而,Web应用的安全性不容乐观.据国家互联网应

* 基金项目: 国家自然科学基金(61672085, 61702029, 61872026)
Foundation item: National Natural Science Foundation of China (61672085, 61702029, 61872026)
本文由“系统软件构造与验证技术”专题特约编辑赵永望副教授、刘杨教授、王戟教授推荐.
收稿时间: 2019-09-01; 修改时间: 2019-10-24; 采用时间: 2019-12-24; jos 在线出版时间: 2020-04-07

急中心监测数据显示:仅 2019 年 5 月 13 日~19 日这 1 周内,国家信息安全漏洞共享平台新收录网络安全漏洞 363 个,其中应用程序漏洞占比高达 47.1%[1].Web 应用的前后端分离、异步通信及事件驱动等特点,使得传统软件测试用例生成方法不适用于 Web 应用的测试生成,给 Web 应用软件的安全测试带来了新的挑战.目前, 关于 Web 应用安全测试生成的研究主要集中在前端(客户端)模型的测试生成和后端(服务器端)代码的测试生成两方面.基于前端模型的测试生成[2−5]大多以模型自身的状态/迁移/迁移序列为目标,生成覆盖前端目标的测试用例.这种方法未考虑后端代码,因此,其测试用例难以对后端代码的安全漏洞进行有效测试.而基于后端代码的测试用例生成[6−11]大多仅以后端代码为依据,未考虑 Web 应用前端的行为,难以有效检测出比如二阶注入之类的安全漏洞.因此,实验室前期研究中[12−14]提出了一种 Web 应用前后端融合的测试用例生成方法,以 Web 应用后端的敏感路径覆盖为目标,利用遗传算法 GA 从前端行为模型 CBM(client-side behavior model)进行测试用例集自动生成.然而,将种群搜索算法应用于 Web 应用测试用例生成时,由于个体的执行需要模拟用户在浏览器上的操作,种群中个体的串行执行会频繁启动浏览器执行各个体的事件序列,且适应度值计算也较为耗时,使得测试用例生成时间开销较大.因此,如何提高 Web 应用测试用例的生成效率,是一个值得关注的问题.

种群个体并行执行[15−17]可以更好地利用系统资源,提高测试用例生成效率.事实上,遗传算法作为基于种群的搜索算法,其自身具有良好的并行性,这表现在其个体的操作是独立的,且允许个体的适应度函数并行计算.然而,很少有人尝试将种群并行化应用于Web应用测试生成中.此外,由于Web应用的前后端分离及事件驱动等特性,使得将 GA 应用于 Web 应用测试用例生成的并行化时,存在多浏览器进程管理、后端代码执行路径收集等一系列问题.因此,研究一种针对 Web 应用的高效并行化测试用例生成方法以提高其测试生成效率十分必要.

为此,本文将种群并行化计算引入到基于 GA 的 Web 应用前后端融合的测试用例生成中.在种群个体执行过程中,通过设计线程池模型及相应的调度逻辑,动态地为个体分配空闲资源,控制多浏览器进程的并行及复用,实现种群个体的并行执行及其适应度值的并行计算;此外,在多个个体并发执行的情况下,为收集不同个体的后端覆盖信息,采用反向代理及 URL 模糊匹配标识不同个体,并对其后端覆盖信息进行区分;从而实现 Web 应用的测试用例生成过程并行化,提高其测试用例生成效率.

为验证 Web 应用并行化测试用例生成的有效性,本文对不同被测程序进行了大量实验,并与现有的串行方法进行了对比分析.实验结果表明:

- 本文提出的GA并行化测试用例生成方法,能够从Web应用前端行为模型CBM上生成覆盖后端敏感路径的测试用例;
- 与现有的 Web 应用前后端融合的 GA 串行化测试用例生成方法相比,本文提出的并行化方法确实能够有效提高测试用例的生成效率;并且能够显著提升系统的资源利用率.

本文第 1 节介绍 Web 应用的前端模型、后端敏感路径等基本概念及 Web 应用前后端融合的 GA 串行化测试生成相关技术.第 2 节阐述 Web 应用前后端融合的 GA 并行化测试用例生成面临的主要问题及解决方案.第 3 节介绍 GA 并行化测试用例生成方法.第 4 节针对开源 Web 应用进行 GA 并行化测试用例生成实验,从时间开销及系统资源利用率等方面验证并行化方法的有效性.第 5 节是相关研究分析.最后总结全文,并对未来值得关注的研究方向进行初步探讨.

1 基本概念及相关技术

本节介绍 Web 应用的前端行为模型 CBM、后端敏感路径以及基于 Web 应用前后端融合的 GA 测试用例生成技术.

1.1 Web应用前端的行为模型

Web 应用程序的一般工作流程是:用户在前端浏览器的 Web 页面上点击按钮触发相应事件,将对应的请求发送到后端服务器,后端处理请求并将响应返回至前端.然后,前端根据收到的响应更新当前页面的内容,从而实现 Web 应用前后端的数据交换.在此过程中,Web 页面和事件是表征 Web 应用动态行为的两个主要组件.对于 Web 页面,传统的 Web 应用为每个页面绑定了一个唯一的 URL,因此,页面可以用 URL 表示.然而在 Web 2.0 中,

随着 JavaScript(JS)和 DOM(document object model)的广泛使用,页面不再由 URL 决定,而是表示为 DOM 树,并且 JS 代码的执行可以更改 DOM 树.也就是说,页面的改变由 DOM 树的动态改变而决定.此外,事件执行过程中,事件处理函数响应用户操作并完成前端的业务逻辑功能[18].但事件处理函数中,不同的执行条件可能导致不同的页面跳转,从而引起不同的参数或 DOM 元素的变化.因此,页面的跳转不仅与事件有关,而且还与事件的触发条件以及对参数或 DOM 元素的变化有关.然而,现有的 Web 应用行为模型不能完全表示事件触发条件和页面跳转之间的联系.因此,我们定义了一个新的 Web 应用前端行为模型 CBM,并给出了一个基于用户行为轨迹的 CBM 模型构建方法[14],模型的定义如下.

定义 1(Web 应用前端行为模型 CBM)[14]. Web 应用前端行为模型 CBM 定义为一个四元组(S,I,O,T),其中,S 为非空的状态集合,I 是非空的输入参数集合,O 是非空的输出参数集合,T 是非空的迁移集合.S 中的状态为 Web 页面的抽象表示,用〈URL,abstract DOM〉表示,其中,abstract DOM 表示 Web 页面的 DOM 结构及其属性;I 中的每个元素为事件触发的输入参数;O 中的每个元素为事件响应时的输出参数;T 中的每个元素为状态之间的一个迁移,可用一个五元组〈*Src*,*Event*,*Cond*,*Action*,*Tgt*〉表示.具体来说,对于 T 中的元素 t:*Src*(t)为迁移 t 的源状态;*Tgt*(t)为其目标状态;*Event*(t)表示用户在 *Src*(t)状态下触发的 Web 事件,用 *event*(*InputList*)表示,表明触发 *event* 事件需输入参数 *InputList*;*Cond*(t)是事件的触发条件;*Action*(t)是迁移 t 引起的后续操作,用 *action*(*OutputList*)表示,说明事件触发之后对客户端的参数及服务器端返回的参数 *OutputList* 进行相应的 *action* 操作.即表示在源状态 *Src*(t)上,若触发事件 *Event*(t)且满足迁移条件 *Cond*(t),则引发后续操作 *Action*(t),并将状态转换到 *Tgt*(t)状态.

例如,开源校园管理系统(SchoolMate)用户管理模块的部分 CBM 如图 1 所示.其中,S_0 为登录页面,若管理员的用户名或密码错误(T_0/T_2),则管理员仍停留在登录页(S_0);否则(T_1),进入系统主页面(S_1);此时,管理员可点击管理用户操作按钮(T_3),进入用户管理页(S_2);在用户管理页面,管理员可以编辑已有用户信息,添加/删除用户等;假定管理员要添加一个新用户,则他需要点击添加按钮(T_4),进入添加用户信息页(S_4);然后输入新用户的用户名、密码及用户角色,再点击确认按钮,若新用户信息正确(T_8),则添加成功,返回用户管理页(S_2);否则(T_6/T_7),进入添加用户失败页(S_5).详细迁移信息见表 1,具体的模型构建方法参见文献[14].

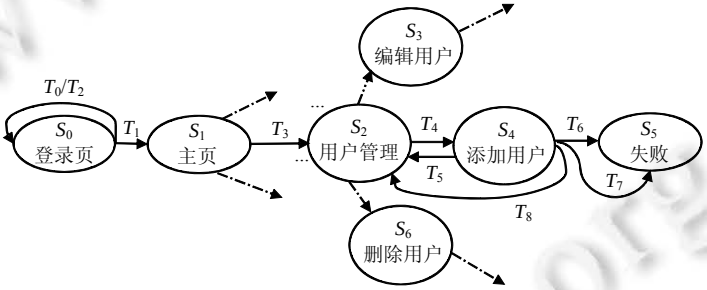


Fig.1 CBM of user management module in SchoolMate

图 1 SchoolMate 用户管理模块的行为模型 CBM

Table 1 Details of transitions on the CBM of user management module in SchoolMate

表 1 SchoolMate 用户管理模块的行为模型 CBM 迁移信息

迁移	源状态	事件	条件	后续操作	目标状态
T_0	S_0	*click*,*Xpath*://*input*[@*value*='*Login*'] (*username*,*password*)	*username.value*=='·'‖ *password.value*=='·'	*Alert* ('*invalid params*')	S_0
T_1	S_0	*click*,*Xpath*://*input*[@*value*='*Login*'] (*username*,*password*)	*username.value*!='·' && *password.value*!='·'	*document.login.value*=1	S_1
T_2	S_0	$("#btn-login").*click*[*username*,*password*]	*username.value*!='·' && *password.value*!='·'	*document.login.value*=1	S_0
T_3	S_1	*click*,*link*=*Users*	–	–	S_2
T_4	S_2	*click*,*Xpath*://*input*[@*value*='*Add*']	–	*document.users.submit*(·)	S_4

Table 1 Details of transitions on the CBM of user management module in SchoolMate (Continued)

表 1 SchoolMate 用户管理模块的行为模型 CBM 迁移信息(续)

迁移	源状态	事件	条件	后续操作	目标状态
T_5	S_4	*click*,*Xpath*://*input*[@*value*='*Cancel*']	–	*document*.*adduser*.*submit*(·)	S_2
T_6	S_4	*click*,*Xpath*:(//*input*[@*value*='*AddUser*'])(*username*,*password*,*password2*,*type*)	*password*.*value*!=*password2*.*value*||*password*.*value*=='·')	*alert*('*Passwords* do not *match*!')	S_5
T_7	S_4	*click*,*Xpath*:(//*input*[@*value*='*AddUser*'])(*username*,*password*,*password2*,*type*)	*password*.*value*==*password2*.*value* && *password*.*value*!='·'	*document*.*adduser*.*submit*(·)	S_5
T_8	S_4	*click*,*Xpath*:(//*input*[@*value*='*AddUser*'])(*username*,*password*,*password2*,*type*)	*password*.*value*==*password2*.*value* && *password*.*value*!='·'	*document*.*adduser*.*submit*(·)	S_2

1.2 后端敏感路径

在 Web 应用的后端代码中,由于对外部输入的过滤不充分,恶意数据可能会从后端代码的入口(源)语句(如 $_*GET*,$_*POST*))流到敏感的槽语句(如 *mysql_query*)[19],从而引发输入验证漏洞.这些语句在后端代码形成了一个容易受到注入攻击的执行路径.本文将这种执行路径定义为后端代码的敏感路径.

定义 2(敏感路径). 假设后端代码的语句执行序列 $Sseq=\langle st_1,st_2,\ldots,st_n\rangle$满足下面 3 个条件,则该语句序列 *Sseq* 被称为敏感路径.

(1) 语句 st_1 涉及可以接收外部输入的变量,该变量称为污染变量.

(2) 语句 st_n 含有对污染变量的操作,如访问数据库、输出到 Web 页面等.

(3) 语句 st_1 上的污染变量在到达 st_n 时,仍然处于受污染状态,即从 st_2 到 st_{n-1} 的语句序列,对 st_1 上污染变量的过滤不充分.

例如,在下面 SchoolMate 添加用户模块的后端代码 *ManageUser.php* 中,语句 2~语句 4 中的变量$*username*, $*password* 及$*type* 为污染变量,用来接收外部输入;语句 14 和语句 15 将污染变量分别存储至数据库和输出到页面中;若语句 2~语句 15 的语句执行序列中未对上述的污染变量进行过滤处理,如⟨2-3-4-5-6-8-10-11-13-14-15⟩,则该语句序列为一条敏感路径.

代码片段 1. SchoolMate 添加用户模块后端代码 *ManageUser.php*.

```
1:  〈?php
2:      $username=$_POST["username"];    //source
3:      $password=$_POST["password"];    //source
4:      $type=$_POST["type"];    //source
5:      if ($_POST["adduser"]==1){    //trigger the adduser event
6:          if (strpos($username,'<script')){    //sanitization
7:              $username=htmlspecialchars($username);
8:          if ($username in $user)
9:              echo 'username already exist';
10:         else
11:             if (strlen($password)<6)
12:                 echo 'invalid password';
13:             else
14:                 $query=mysql_query("INSERT INTO users VALUES('·','$username','.md5($password)',
                    '$type')");    //sink
15:                 echo $username.'add successful';    //sink?〉
```

1.3 Web应用前后端融合的测试用例生成

Web 应用前后端融合的测试用例生成是指将 Web 应用前端 CBM 模型与后端敏感路径的覆盖结合起来进

行测试用例生成,即以后端敏感路径覆盖为目标,利用搜索算法进行前端 CBM 模型测试用例的演化生成[12,13].例如,以 SchoolMate 添加用户模块后端代码的敏感路径⟨2-3-4-5-6-8-10-11-13-14-15⟩为目标,利用搜索算法从前端 CBM 模型(如图 1 所示)上找到能覆盖该敏感路径的测试用例,即迁移序列(如⟨$T_1T_3T_4T_8$⟩)及其输入数据(如 T_1(‘*admin*’,‘123456’),T_8(‘*teacher*1’,‘111111’,‘111111’,‘teacher’)).其测试用例生成方法如算法 1 所示.

具体而言,针对 Web 应用的后端敏感路径集 *Sspath*,在其前端 CBM 模型上,通过全局搜索 GA 生成迁移序列,使之能覆盖敏感路径或达到其源点(第 1 行~第 27 行),即从 CBM 模型上随机生成 *N* 条迁移序列作为初始个体,形成初始种群;然后,执行种群中个体,获取个体对后端敏感路径的覆盖信息.为使个体能够自动执行,需利用 Selenium(Selenium 是一个用于 Web 自动化测试的工具,具有丰富的应用程序编程接口)将个体转换为测试脚本,借助于浏览器运行测试脚本,模拟用户在浏览器上的操作.同时,根据个体对后端敏感路径的覆盖情况,构造覆盖矩阵 *M*,矩阵的列 *k* 表示个体 *ind*[*k*]对各条敏感路径的覆盖情况,行 *j* 表示敏感路径 *Sspath*[*j*]被各个个体覆盖的情况,其元素 *M*[*j*][*k*]表示第 *k* 个个体 *ind*[*k*]对第 *j* 条敏感路径 *Sspath*[*j*]的覆盖情况,即 *M*[*j*][*k*]=*approach_level*+*branch_distance*.依据覆盖矩阵 *M* 可计算个体适应度值(该个体对所有敏感路径的覆盖情况),并据此进行遗传操作及种群更新.若敏感路径全部被覆盖,则其测试用例生成结束;若仅达到敏感路径的源点,则借助于局部搜索模拟退火算法 SA 调整测试序列上的输入参数,使之能完全覆盖相应的敏感路径(第 28 行~第 31 行).

算法 1. 基于 Web 应用前后端融合的测试用例生成算法.

输入:Web 前端 CBM 行为模型,敏感路径集 *Sspath*,最大迭代次数 Max.

输出:测试用例集 *TS*.

1: //基于全局搜索 GA 的测试序列生成
2: i=0, TS=∅, TS_{tem}=∅;
3: *pop*=*random_generator*(*CBM*,*N*) //生成规模为 *N* 的初始种群
4: *M*=*Executor*(*pop*,*Sspath*) //执行个体,构建覆盖矩阵 *M*
5: **for** each $ind_k \in pop$ **do**
6: *Fitness*[ind_k]=*Global_Fit_Eval*(ind_k,*M*) //计算个体 ind_k 的适应度值
7: **end for**
8: **while** *i*<Max **do**
9: set *childpop*=∅
10: **while** *size*(*childpop*)≤*N* **do**
11: ind_1,ind_2=*GeneticOp*(*pop*,*Fitness*) //生成子代
12: $childpop=childpop \cup \{ind_1,ind_2\}$
13: **end while**
14: *M*=*Extend*(*M*,*Executor*(*childpop*,*Sspath*)) //执行子代个体,并将子代的覆盖信息扩展到 *M* 中
15: **for** each $ind_k \in childpop$ **do**
16: *Fitness*[ind_k]=*Global_Fit_Eval*(ind_k,*M*) //计算个体适应度
17: **end for**
18: Sp_{cov}=*Get*(*Sspath*,*M*), Sp_{srccov}=*Get*(*Sspath*,*M*) //获取已被覆盖或源点被覆盖的敏感路径
19: $TS=TS \cup ind_{Spcov}$ //将覆盖了敏感路径的个体加入到测试用例集中
20: $TS_{tem}=TS_{tem} \cup ind_{Spsrccov}$ //将覆盖了敏感路径源点的个体加入到临时测试用例集中
21: $Sspath=Sspath-Sp_{cov}-Sp_{srccov}$ //修订要被覆盖的敏感路径集
22: **if** (|*Sspath*|≠0) **then**
23: *pop*=*Update*(*pop*,*childpop*,*Fitness*)
24: *M*=*ReviseM*(*pop*,*M*,*Sspath*) //利用原有的覆盖矩阵、新种群 *pop* 及未被覆盖的 *Sspath* 修订 *M*
25: $i=i+1$

26: **else** break

27: **end while**

28: //基于局部搜索 *SA* 的测试数据生成

29: **if** ($|Sp_{srccov}|\neq 0$) **then**

30: *SA_test_data_generator*(Sp_{srccov},TS_{tem})

31: **end if**

32: **return** *TS*

该算法以覆盖准则为媒介,将 Web 应用的前端 CBM 模型与后端敏感路径关联起来,进行前端测试用例的生成.相较于目前基于前端模型的 Web 应用测试用例生成方法,算法 1 同时考虑前端模型和后端代码,提高了从前端模型生成的测试用例对后端代码的覆盖能力.相较于面向后端代码的测试用例生成方法,算法 1 考虑了 Web 应用的前端行为,提升了测试用例对后端代码漏洞的检测能力.

在 GA 的一次迭代过程中,为计算种群个体适应度,需对种群个体的后端覆盖信息 *M* 矩阵进行构造.*M* 矩阵为 *N* 个个体对全部敏感路径的覆盖信息,本文采用分支距离来度量个体对敏感路径的覆盖,因此,*M* 矩阵构造的时间复杂度与敏感路径分支数以及种群规模成正比,为 $O(|Sspath.branch|\times N)$,其中,$|Sspath.branch|$表示所有敏感路径包含的全部分支数量,*N* 为种群规模.而在 Web 应用 GA 串行化测试用例生成中,最差情况下,GA 的迭代次数达到上限,则矩阵 *M* 构造的时间复杂度为 $O(|Sspath.branch|\times N\times \mathrm{Max})$,其中,Max 为 GA 最大迭代次数.再者,种群个体适应度值计算依赖于覆盖矩阵 *M*,其时间复杂度与矩阵大小成正比,为 $O(|Sspath|\times N)$,其中,$|Sspath|$为敏感路径条数,GA 的最大迭代次数为 Max,故整个 GA 迭代过程中,适应度值计算的时间复杂度为 $O(|Sspath|\times N\times \mathrm{Max})$.因此,基于 GA 的 Web 应用测试用例生成的整体时间复杂度为 $O(|Sspath.branch|\times N\times \mathrm{Max})+O(|Sspath|\times N\times \mathrm{Max})$.可以看出,适应度值计算在测试用例生成过程中是十分耗时的.

此外,个体的执行需频繁启动浏览器进程,模拟用户触发事件序列向后端发送请求,后端代码处理请求并将响应返回至前端的过程.其时间开销较大,导致 Web 应用测试生成效率较低.

2 Web 应用前后端融合的 GA 并行化测试用例生成关键问题及解决方案

对 GA 串行生成 Web 应用测试用例过程进行监视发现,个体串行执行及适应度值计算时,CPU 占用、磁盘读写、网络带宽等都处于间断活跃的状态,系统资源利用率较低.此外,之前的研究发现,GA 串行化测试用例生成过程中,个体执行时间占比高于 87%[12].种群个体并行执行及适应度并行计算,可以充分利用系统资源,缩短每代种群的执行时间及适应度计算时间,提高 Web 应用测试生成效率.但不同于传统程序的 GA 并行化测试用例生成,Web 应用前后端融合的并行化测试用例生成中存在诸多问题,如前端多浏览器进程的并行执行、后端覆盖信息的收集、个体执行时间及适应度值计算的时间差异等.为此,本节重点讨论 Web 应用前后端融合测试用例生成过程中 GA 并行化存在的关键问题及其解决方案.

2.1 Web应用前后端融合的GA并行化测试用例生成存在的关键问题

i. 多浏览器进程的管理

不同于传统程序,Web 应用测试用例的执行需要借助于浏览器,模拟用户在浏览器上的操作,如将请求发送到后端、接收后端处理请求后返回至前端的响应等.对于 Web 应用测试用例的 GA 串行化生成,个体为前端行为模型 CBM 的迁移序列,模拟个体执行,即为每个个体创建相应的浏览器进程,但浏览器进程的反复创建将带来极大的时间开销.

若对 Web 应用进行 GA 并行化测试用例生成,则种群个体的并行执行需同时启动多个浏览器进程.如果为每个个体创建各自的浏览器进程,多个并行浏览器进程的反复启动会带来大量的资源消耗.由于浏览器进程对系统资源的消耗较大,普通 PC 一般无法支持 10 个以上的浏览器进程.如果强行启动过多的浏览器进程,则会导致大量虚拟内存的读写,降低系统性能.因此,如何合理利用系统资源,对并发浏览器进程进行有效管理,是GA并行化 Web 应用测试用例生成的一个关键问题.

ii. 多线程下后端覆盖信息的收集

GA 串行生成 Web 应用前后端融合的测试用例过程中,个体执行对后端路径的覆盖情况可由插桩程序得到.插桩程序通过在后端敏感路径的每个分支开始处植入探针,即可获取个体执行时对后端代码的覆盖情况.由于个体串行执行时,仅有 1 个线程执行后端代码,后端的执行覆盖信息直接对应前端个体.但在 GA 并行生成 Web 应用测试用例时,多个个体并发执行,插桩程序无法区分各个个体的执行信息,也就无法向对应的个体提供正确的覆盖信息,这也阻碍了并行化测试用例的生成.

iii. 种群个体执行时间及适应度值计算时间的差异

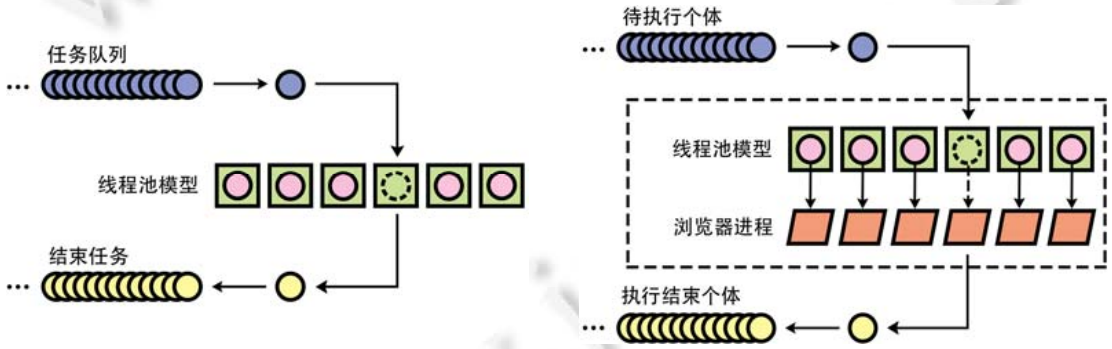
基于GA的Web应用测试用例生成中,个体长度不尽相同,其包含的状态(Web页面)渲染时间及各个迁移的执行时间也不相同,因此,种群中不同个体的执行时间差异较大.此外,不同个体对后端敏感路径的覆盖也各不相同,使得适应度值的计算时间也有所不同.这些均影响 GA 并行化 Web 应用测试用例生成的性能.因此,在 GA 并行化 Web 应用测试用例生成中,如何为个体合理分配资源,减少个体执行时间差异对并行化性能的影响,是提高 GA 并行化测试生成效率的一个关键问题.

2.2 Web应用前后端融合的GA并行化测试用例生成解决方案

线程池是一种有效的并行程序设计技术,可以避免线程反复创建和销毁带来的开销.因此,本文拟利用线程池模型实现种群个体执行及适应度值计算的并行化.但不同于传统程序,Web 应用前后端融合的测试用例并行化生成存在上述诸多问题,为此,本文探讨一种新的线程池模型解决方案.

i. 新的线程池模型及调度策略设计

通用线程池模型含有多个线程,用以管理多线程的并发执行,设有主线程和若干子线程,主线程为子线程分配任务并使其并发执行,同时监听子线程的执行状态,如图 2(a)所示.Web 应用测试用例的执行必须借助于浏览器模拟用户操作.因此,在 Web 应用 GA 并行化测试用例生成中,个体的并行执行不仅需要线程池模型创建多个线程,而且需要为每个个体分配相应的浏览器.为此,本文设计的线程池模型如图 2(b)所示,设有一个主线程及若干子线程和浏览器进程,子线程负责个体的初始化及后续适应度值的计算等,浏览器进程负责执行测试脚本,模拟用户操作,二者协同,完成个体的执行及评估.主线程负责分配子线程及相应的浏览器进程.一旦浏览器进程结束,则子线程关闭浏览器进程,释放浏览器资源.当线程结束,则关闭线程,并将子线程及浏览器进程同时置为空闲状态.



(a) 通用线程池模型 (b) Web 应用测试用例并行化的线程池模型

Fig.2 Thread pool model

图 2 线程池模型

为减少个体执行及适应度值计算的时间差异对并行化测试生成性能的影响,提高系统资源利用率,该线程池模型采用动态分配方式为待执行个体分配空闲线程及浏览器进程.也就是说:若存在待执行个体,一旦主线程识别到有空闲的线程及浏览器进程,则将其分配给该个体.考虑到线程执行时间长,线程轮换次数较少,空闲线

程及浏览器进程的判定采用临界区线程同步机制,即轮询线程及进程是否为空闲状态.其次,为减少个体并行执行过程中浏览器进程反复启动带来的资源消耗,该线程池模型采用浏览器进程复用的方式,即当浏览器进程执行结束并释放其浏览器资源后,该空闲浏览器进程便可分配给下个待执行的个体,不需要为下个待执行个体创建新的浏览器进程.此外,由于个体执行过程不能被中断,因此,非抢占式的线程调度策略被采纳,以保证种群中个体执行的连贯性.

综上,线程池模型的调度策略设计如下:假设子线程及浏览器进程数为 m,主线程为 m 个个体分配子线程及浏览器进程,并使其个体并发执行.当某个个体执行结束并释放子线程及浏览器进程后,主线程判断是否存在待执行个体:如有,则为待执行个体分配空闲线程及浏览器进程;否则,执行现有线程直至所有线程执行完毕.

由此可见,该线程池模型及调度策略的设计解决了上述所涉及的两个问题.

(1) 采用浏览器进程并行化及复用,解决了并发执行过程中由于浏览器反复创建及启动带来的资源消耗问题.

(2) 采用线程及浏览器进程的动态分配,解决了因个体执行时间差异导致的个别线程积压待执行个体,而其他线程空闲的情况,提高了系统资源利用率.

ii. 反向代理及 URL 模糊匹配设计

反向代理是一种代理服务,可以介于浏览器与 Web 服务器之间,用于捕获前后端的信息传递.本文利用反向代理及 URL 模糊匹配,实现个体并行执行时,各个体后端敏感路径覆盖信息的区分及获取.即为待执行个体添加标识 *ID*,形成个体访问路径.当浏览器进程向反向代理服务器发起请求时,反向代理获取个体的访问路径,并对个体访问路径进行 URL 模糊匹配,识别出附加在 URL 上的个体标识 *ID*,将其写入个体请求的头文件 *header* 字段,同时将其作为插桩标记 *key*;然后,反向代理将带有插桩标记的个体请求转发到 Web 应用所在的服务器,执行后端代码.后端代码从个体请求的 *header* 字段中取出插桩标记 *key*,插桩程序依据插桩标记感知不同个体,记录并区分不同个体的覆盖信息.

3 Web 应用前后端融合的 GA 并行化测试用例生成方法

基于 Web 应用前后端融合的 GA 并行化测试用例生成,旨在通过线程池管理、浏览器进程控制及复用,实现多浏览器进程的并行化管理;通过反向代理及 URL 模糊匹配来标记个体的后端代码执行路径,从而获得各个体并发执行时的后端路径覆盖信息;通过设计非抢占式的线程池调度逻辑,实现个体系统资源的动态分配,减少个体执行时间差异对并行化性能的影响,以提高 Web 应用测试用例生成效率及系统资源利用率.

第 1.3 节已说明 Web 应用前后端融合的 GA 测试用例生成算法,并行化测试用例生成的本质是对个体执行及适应度值计算进行并行化,即对算法 1 的第 14 行~第 17 行进行并行处理,故 Web 应用前后端融合的 GA 并行化测试用例生成框架如图 3 所示.

具体而言,为使种群个体并行化执行以及适应度值并行化计算,先假设种群规模为 n,可启动的最大线程数及浏览器进程数设置为 m,$n \geqslant m$.初始化线程池模型,即对于 n 个待执行个体,主线程首先为前 m 个个体分配子线程 thread 及浏览器进程 driver,并启动其并发执行,执行过程如下:① 子线程加载个体对应的迁移序列、Selenium 脚本以及个体标识 *url_ind_i*,其中,url 表示个体对应的初始 URL,*i* 为对应个体 *ind_i* 的标识 *ID*;② 浏览器进程执行个体的脚本,并将个体请求发送至反向代理;③ 反向代理接收个体请求,利用 URL 模糊匹配提取个体标识 *ind_xi*,并将其添加至个体请求的 *header* 字段,形成插桩标记 *header_ind_xi*;④ 后端插桩程序依据插桩标记记录个体覆盖信息;⑤ 子线程读取其覆盖信息并进行适应度值计算.当某个个体执行结束并释放浏览器进程及相应线程时,该线程及浏览器进程置为空闲状态.若种群存在待执行个体,则主线程为其分配空闲线程及浏览器进程,并启动其执行;否则,等待线程执行结束.

因此,Web 应用前后端融合的 GA 并行化测试用例生成流程如下:首先,根据 Web 应用的前端 CBM 模型生成初始种群,并为种群中的每个个体添加标识 *ID* 形成其访问路径,为后端敏感路径覆盖信息的获取做准备;之后,根据 CBM 模型、个体迁移序列及个体标识 *ID* 初始化线程池,即,将个体迁移序列、Selenium 脚本以及个体

标识 *ID* 加载到线程池中,并且设置最大线程及浏览器进程数;然后,线程池模型为待执行个体动态分配空闲线程,同时启动浏览器进程并行化执行个体;之后,由插桩程序收集当前并发执行个体对后端敏感路径的覆盖情况,形成种群个体覆盖矩阵 *M*;在此基础上,并行计算各个个体的适应度值,并依据其个体适应度,选择当前种群中的个体进行交叉、变异等遗传操作,生成新个体;若后端敏感路径集全部被覆盖,则 Web 应用测试用例生成结束;否则种群更新,进入下一轮 GA 迭代.重复上述过程,直至生成覆盖后端敏感路径的测试用例集或达到最大迭代次数.

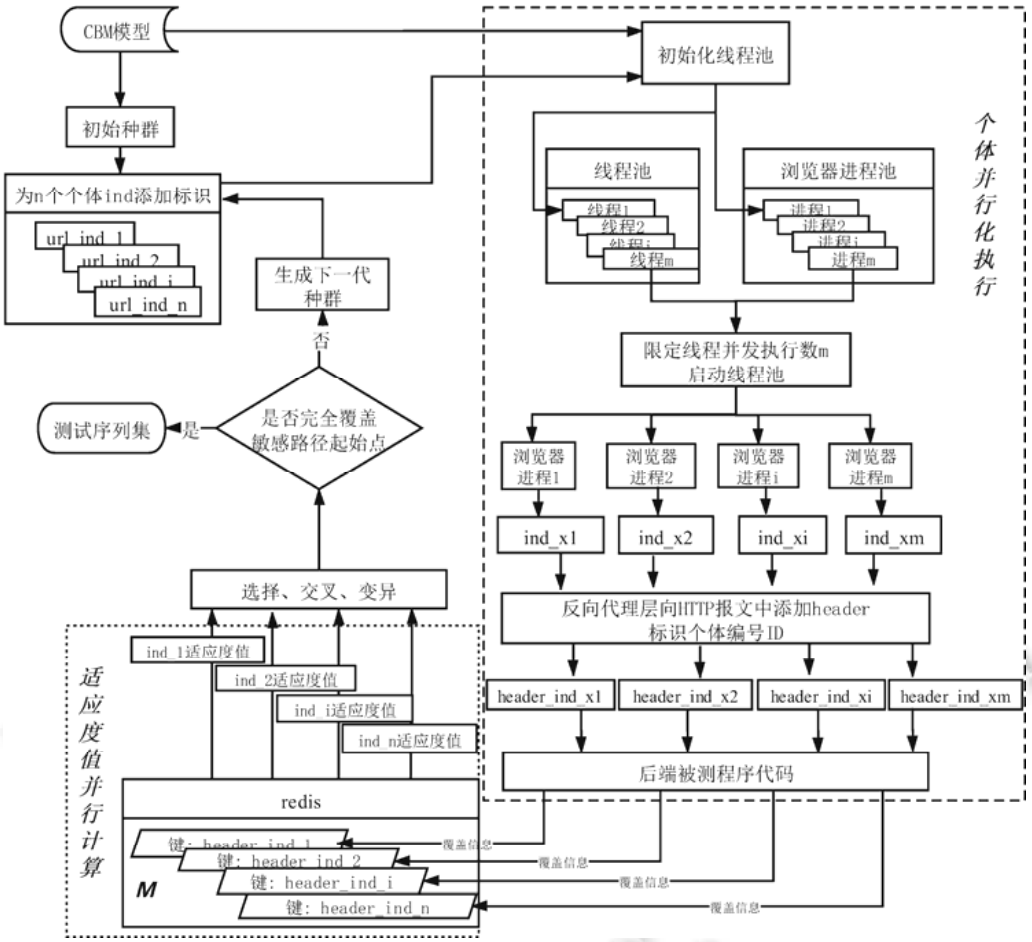


Fig.3 Framework of test case generation based on GA parallelization

图 3 基于 GA 并行化的测试用例生成框架图

对于 GA 串行化 Web 应用测试用例生成算法,前面第 1.3 节中已分析讨论其最差情况下的整体时间复杂度为 $O(|Sspath.branch|\times N\times \text{Max})+O(|Sspath|\times N\times \text{Max})$,其中,$|Sspath|$为敏感路径条数,$|Sspath.branch|$表示敏感路径包含的分支总数,$N$ 为种群规模,Max 为 GA 最大迭代次数.对于并行化 Web 应用测试用例生成,同时启动 m 个线程并行执行种群个体并计算其适应度值,则每个线程的平均时间复杂度为串行的 $1/m$,故 GA 并行化测试用例生成算法的时间复杂度仍为 $O(|Sspath.branch|\times N\times \text{Max})+O(|Sspath|\times N\times \text{Max})$.然而,由于在 GA 并行化测试用例生成过程中需对种群个体进行多线程及浏览器进程动态分配,线程调度会产生额外的时间开销,其时间复杂度与种群规模成正比,即为 $O(N)$.GA 的迭代上限为 Max,故线程调度的总体时间复杂度为 $O(N\times \text{Max})$.因此,最差情况下,整个 GA 并行化测试用例生成算法的时间复杂度为 $O(|Sspath.branch|\times N\times \text{Max})+O(|Sspath|\times N\times \text{Max})+O(N\times \text{Max})$.

4 实验结果与分析

为了验证面向前后端融合的 Web 应用 GA 并行化测试用例生成方法的有效性,本文从敏感路径覆盖、时间开销及资源利用率这 3 个维度对比分析了 GA 串行化和并行化的测试用例生成方法.在 GA 并行化方法中,主要通过种群个体并行化执行及适应度并行化计算提高测试生成效率,因此,实验中主要对种群个体执行和适应度值计算的时间及系统资源占用情况进行了分析.

4.1 实验被测程序与环境

本文实验的被测程序是 PHP5 的开源 Web 程序(https://sourceforge.net),均为基于 PHP 5.6.40 非线程安全版本解释器的 Web 应用,分别是 Faqforge、Addressbook、Webchess、SchoolMate、phpaaCMS,其详细描述见表 2.被测 Web 应用对应的 CBM 模型及敏感路径信息见表 3.其中,Addressbook 应用中的部分代码为配置文件 z-push,该文件是为 iPhone 和 Android 同步设置的配置文件,约占总代码行数的 68%,因此,Addressbook 实际执行的代码行数约为 15 200 行.

Table 2 Description of experimental Web applications

表 2 被测 Web 应用描述

Web 应用	版本	语言	程序大小(KB)	代码行数	功能描述
Faqforge	1.3.2	PHP	227	1 712	问答系统管理工具
Addressbook	8.2.5	PHP	3 799	47 481	在线地址簿系统
Webchess	1.0.0	PHP	468	4 722	在线象棋游戏
PhpaaCMS	0.0.5	PHP	1 659	5 711	在线文章管理系统
SchoolMate	1.5.4	PHP	365	8 181	学校管理系统

Table 3 Information for client-side CBM model and sensitive paths

表 3 被测 Web 应用的 CBM 模型信息及敏感路径信息

CBM 模型	状态数	迁移数	敏感路径数	敏感路径上的分支总数
Faqforge	9	23	3	3
Addressbook	31	75	5	9
Webchess	14	36	12	16
PhpaaCMS	54	92	16	31
SchoolMate	69	262	26	81

本文实验运行的机器配置为:Intel(R) Core(TM) i5-7200U CPU@2.50 GHz 2.70 Ghz;8.00GB 内存;Windows 10 64 位操作系统.实验过程中关闭无关进程,关闭 Windows Defender,关闭外网的连接.

测试用例生成程序使用 python2.7 编写.

线程池中总线程数及总进程数均为 6.

PHP 被测程序使用 php-cgi 转发到本地端口,由 Nginx 服务器程序做代理.为进行 Redis 连接,添加 php_redis-2.2.7-5.6-nts-vc11-x64 插件.

PHP 进程使用 xxfpm@0.01 调用 php-cgi.exe 启动.

KV 存储使用 Redis server v=3.2.100 malloc=jemalloc-3.6.0 bits=64.

Web 服务器使用 nginx/1.12.2 built with OpenSSL 1.0.2l(prel 正则引擎使用--with-pcre=objs.msvc8/lib/pcre-8.41)

数据库使用 mysql Ver 14.14 Distrib 5.7.21, for Win64 (x86_64)

开发语言与工具版本:Python 2.7.14;Selenium 3.141.0;geckodriver 0.24.0 WebDriver implementation for Firefox;Firefox 67.0 (64-bit).

4.2 实验设计

实验从不同的维度对比了基于GA串行化和并行化的Web应用测试用例生成方法,并提出了3个研究问题.

- 研究问题 1. Web 应用前后端融合的 GA 并行化测试用例生成方法是否能够生成覆盖后端敏感路径的

测试用例?

- 研究问题 2. 相较于 GA 串行化的测试用例生成方法,GA 并行化测试生成方法的时间开销如何?
- 研究问题 3. 相较于 GA 串行化测试用例生成方法,GA 并行化测试生成方法对系统资源的占用情况如何?

为了保证 GA 并行化测试用例生成方法与串行化方法生成过程中各种群的一致性,在并行化方法中,通过执行与 GA 串行化方法相同的种群,获取并行化前后的执行效率.在实验过程中,除了要关注测试生成及执行的时间以外,由于并行化测试生成对系统的各方面性能产生了更多的压力,因此需要观察系统各方面性能的变化,进而从更多角度来分析导致实验结果的原因.此外,由于测试过程中部分迁移会操作写数据库,导致实际的迁移操作不幂等,即其任意多次执行所产生的影响与一次执行的影响不同,因此在每次实验前,通过 SQL 脚本初始化数据库的数据,从而避免实验的先后顺序影响到实验结果的准确性和可靠性.

4.3 实验结果与分析

为了研究GA并行化测试用例生成方法是否能够生成覆盖Web应用后端敏感路径的测试用例,我们对被测 Web 应用进行了实验,记录了随着迭代次数的增加,后端敏感路径的覆盖情况.实验结果如图 4 所示.

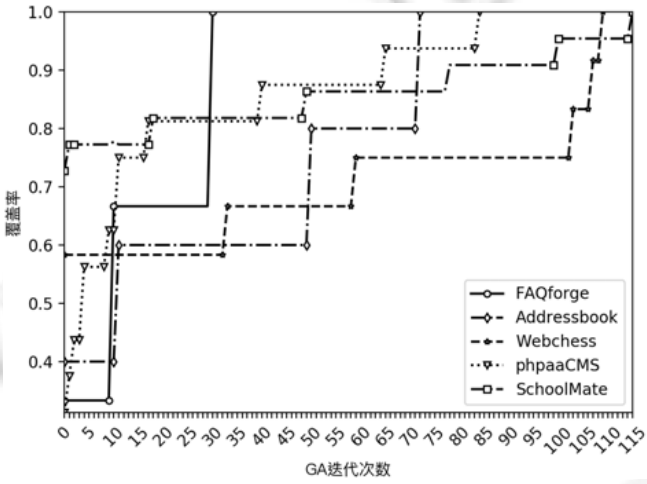


Fig.4 Sensitive path coverage with iterations in GA parallelism of different Web applications

图 4 不同 Web 应用 GA 并行中随迭代次数的敏感路径覆盖率

对于不同的被测 Web 应用,在 GA 并行化测试用例生成中,由于保留了种群中的精英个体,使得种群随着迭代次数的增加,对敏感路径的覆盖率逐渐增加,当敏感路径达到全部覆盖时,得到最终的测试用例.从图 4 可以看出,GA 并行化测试用例生成方法能够生成覆盖 Web 应用后端敏感路径的测试用例.

为了研究 GA 并行化对测试用例生成时间开销的影响,我们对五个被测 Web 应用进行了实验.以 Faqforge 为例,采用 3 种方法对其进行测试用例生成,分别是不复用浏览器的 GA 串行化测试用例生成方法(串行)、复用浏览器的 GA 串行化测试用例生成方法(串行复用浏览器)以及本文复用浏览器进程的 GA 并行化方法(并行复用浏览器).图 5 展示了种群执行时间(s)随迭代次数增长的变化.

在图 5 中,通过对比 GA 串行方法和串行复用浏览器方法可以看出,是否复用浏览器影响了种群的执行时间.分析发现,其耗时在于每次浏览器进程启动时载入内存和初始化相关页面内容上.理论上,浏览器启动时间为串行方法总时间与种群的串行执行时间(即串行复用浏览器方法中种群的执行时间)的差值,然而由于浏览器进程启动速度受系统影响较大,使整体时间呈现不稳定的趋势,但总体上,不使用浏览器复用的 GA 串行化方法时间开销远高于另外两种方法.

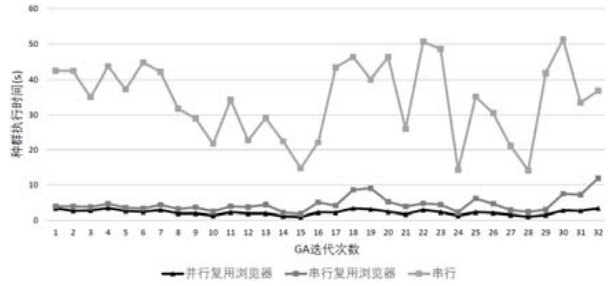


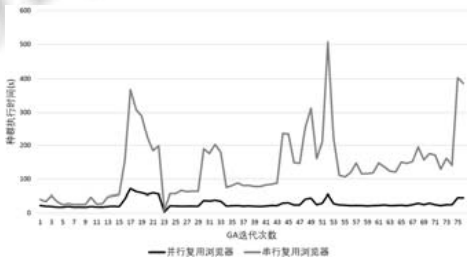
Fig.5 Comparison of execution time for Faqforge by different algorithms

图 5 Faqforge 不同方法的执行时间对比

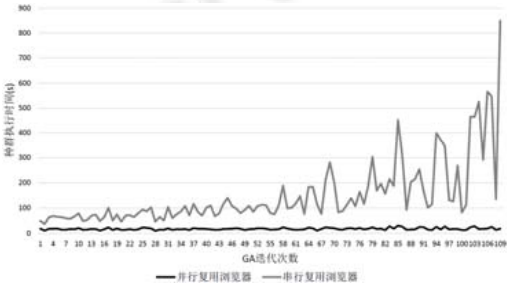
由于复用浏览器进程的 GA 串行化测试生成方法中去除了浏览器的反复启动时间的干扰,种群的执行时间更趋于理论值,因此,后续的实验中不再采用不复用浏览器的 GA 串行化生成方法.下文中的串行化方法指的是串行 GA&复用浏览器进程,并行化方法指的是并行 GA&复用浏览器进程.

针对 5 个被测 Web 应用,在串行化方法与并行化方法的测试用例生成中,种群执行时间随着迭代次数的变化如图 6 所示,横坐标为 GA 迭代次数,纵坐标为种群执行时间(s).

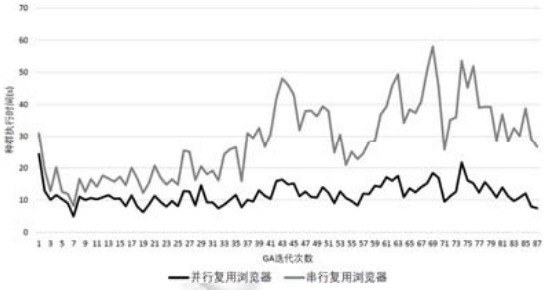
(a) Faqforge



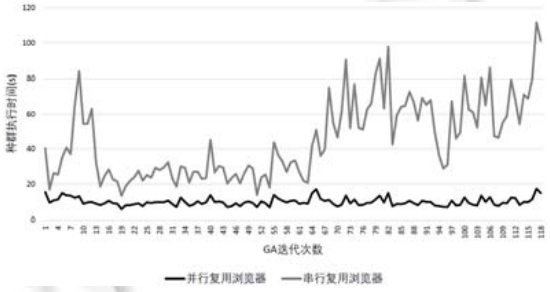
(b) Addressbook



(c) Webchess



(d) PhpaaCMS



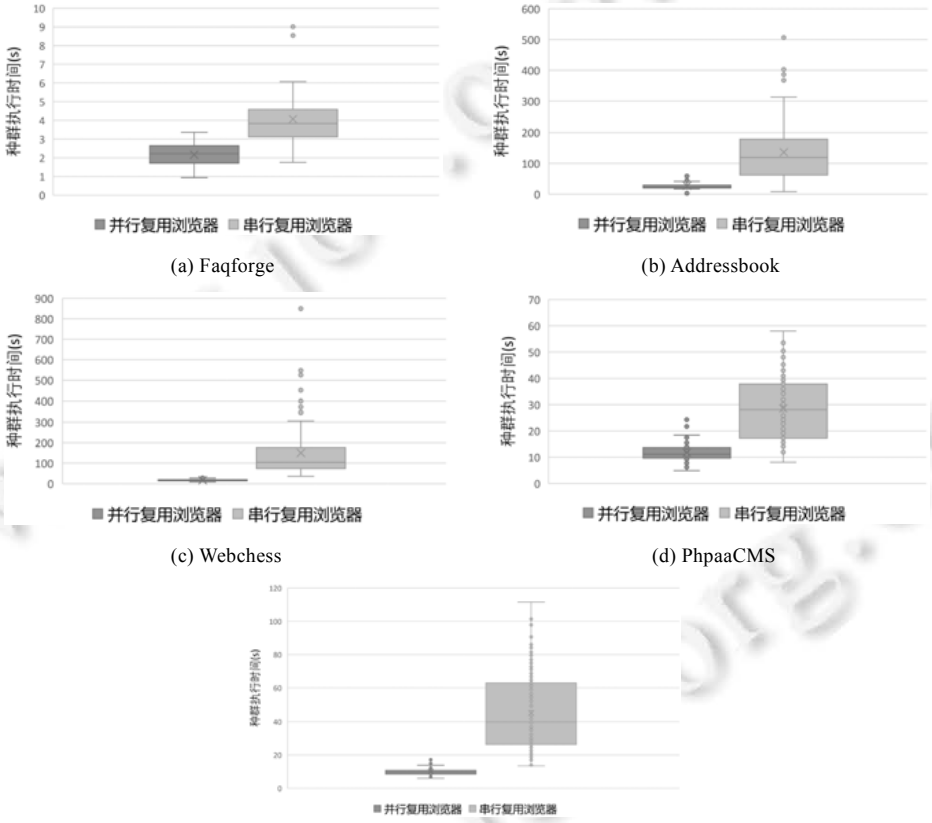
(e) SchoolMate

Fig.6 Comparison of execution time between serialization and parallelization of GA population for Web applications

图 6 Web 应用 GA 种群的串行化与并行化执行时间对比

可以看出,Web 应用测试用例生成过程中,随着迭代次数的增加,整体的执行时间呈现上升趋势,尤其是串行化方法,其上升趋势更为明显.串行化方法中,种群执行时间为所有个体包含的迁移的执行时间之和,执行时间上升可以从两个角度解释:一是个体中迁移的平均执行时间上升,迁移的平均执行时间增加受 Web 应用前端页面的渲染时间以及后端接口性能的影响,由于个体执行导致后台存储了越来越多的脏数据,增加了数据库压力,并且导致浏览器渲染时间变长,迁移的执行时间增加;二是个体包含的迁移个数变多了,迁移在种群中的总数量上升主要是由于经过遗传算法的迭代过程,种群中能有效覆盖后端敏感路径的个体逐渐留存在种群中,而初始种群中较短的个体由于其无法覆盖后端敏感路径,会被遗传操作逐渐淘汰出种群,从而使种群中个体的平均长度有明显的增加趋势.

由图 6 可知,总体上,GA 串行化测试用例生成方法中种群执行所用的时间远高于 GA 并行化方法;且随着迭代次数的增加,GA 并行化方法的优势越来越明显.为了比较不同方法(串行/并行)中种群执行时间的总体表现,利用箱型图对 5 组被测程序的种群执行时间进行了分析,结果如图 7 所示.从图中可以看出,对于所有被测程序,并行化方法的种群平均执行时间均远小于串行化方法.



(a) Faqforge (b) Addressbook

(c) Webchess (d) PhpaaCMS

(e) SchoolMate

Fig.7 Box diagram of execution time of population serialization and parallelization for five different Web applications

图 7 5 组 Web 应用的种群串行化与并行化执行时间箱型图

表 4 为 Web 应用测试用例并行和串行生成方法的种群平均执行时间及其比值,从表中也可以看出,并行化方法中种群平均执行时间仅占串行化方法中的 11.6%~49.3%左右,平均值为 18.9%,即串行化方法的种群平均执行时间是并行化方法 5.29 倍.因此,从图 5~图 7 及表 4 可以得出以下结论:GA 并行化测试用例生成方法可以

有效提高测试生成效率,减少测试生成的时间开销.

Table 4 Average execution time of population serialization and parallelization and its ratio for Web application

表 4 Web 应用串行/并行种群平均执行时间与比例

被测程序	平均执行时间(s)		平均执行时间比
	串行	并行	
Faqforge	4.478	2.207	0.493
Addressbook	136.118	27.227	0.200
Webchess	149.884	17.418	0.116
phpaaCMS	28.583	11.761	0.411
Schoolmate	45.222	10.058	0.222
平均值	72.857	13.734	0.189

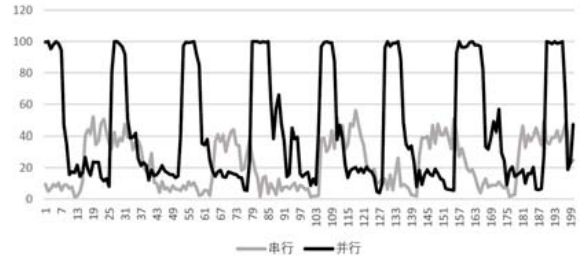
从以上的结果可以看出,虽然并行化都相对有效地减少了种群的执行时间,但是不同的被测程序,其并行优化效果仍不同.进一步分析被测 Web 应用的特点及敏感路径集,并结合实际测试中的系统资源占用情况,我们给出优化效果不同的解释如下.

- Faqforge 的整个服务规模小,后端敏感路径少,前端 CBM 模型也相较于其他被测程序更简单,数据库结构也十分简单只有两张表.因此,初始生成的种群中大部分都是执行时间非常短的迁移,从而使线程池大部分时间均处于不满的状态,因此,初始优化效果相当低.迭代多次后,由于 Faqforge 对数据库的读写量仍十分低,后端处理的效率也很快,主要瓶颈出现在后台 CGI 接口线程和浏览器渲染线程上,优化效果接近系统的实际并行计算的能力.
- Addressbook 具有复杂的表结构,其中,addressbook 表的新建项和修改项对应的前端界面更是拥有十分复杂的输入结构,除了包含了各类输入校验外,还有下拉选择、文件选择等高级页面操作.由于部分下拉选择字段存在字段联动(主要是日期字段),执行 Selenium 脚本时需要设置延时保证二级下拉字段加载完成.同时,Addressbook 本身还会在写库之后进入一个中间界面等待若干秒.这两个特点导致即使是复杂的个体,也会在执行过程中释放一定的系统资源,从而使并发效果更加接近理论优化值.所以当种群迭代次数达到一定程度时,优化比例会高于系统实际并发能力的 1/4,但低于线程池提供的并发量 1/6.从优化效果趋势图中可以看出,从 35 代之后,优化比例就维持在该范围内.
- Webchess 的优化效果不同于前两个被测程序,其并行优化效果随代数的变化更加平稳.分析发现,Webchess 的写库操作和 Faqforge 一样相对较少,但其后端敏感路径的个数远多于 Faqforge 及 Addressbook.因此,Webchess 在并发场景下是 CPU 密集型的 Web 应用,且 Webchess 中的种群中的个体数较多,减少了线程池中线程轮空的时间以及线程池中执行线程等待的时间,从而可以更好地利用多线程的性能.
- Schoolmate 的执行时间则在中间有明显的陡升,通过查看覆盖情况可以发现,在第 67 代时,种群覆盖了新的敏感路径,该覆盖敏感路径的个体由变异操作引入,涉及了更多的数据库读写操作.该个体会被精英策略保留下来,并与其他个体交叉,从而显著拉高了整个种群的串行执行时间.
- phpaaCMS 整体功能分布接近 Addressbook 与 Schoolmate,但其中部分敏感路径需要在后端执行一个.sql 脚本,该操作通过调用系统控制台执行 *mysql* 指令实现,从而导致进程空闲,但仍然保持较高的硬盘读写压力和一定的 CPU 压力.因此可以看到,在中间部分的优化效率有明显的阶梯感.对应地查看实际执行的个体可以发现,大部分陡升陡降的位置都出现了涉及.sql 脚本调用的迁移数量的变化.

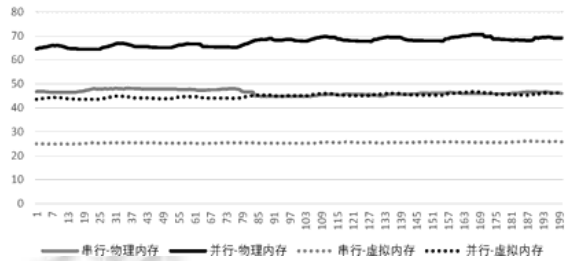
为了研究 GA 串行化方法和并行化方法对系统资源的占用情况,我们对两种方法在测试用例生成过程中对内存及 CPU 等的占用情况进行了对比分析.GA 并行化测试用例生成的本质是更好地利用系统资源进行测试用例并行化执行与适应度值的并行化计算,从而提高生成效率.因此,我们对并行化测试用例生成的系统资源利用情况进行计算与评估.图 7 从 CPU 占用率、内存(物理内存、虚拟内存)、硬盘读写量(单位:字节)这 3 个方面,分别针对串行和并行两种方法 200s 内对系统资源的利用情况进行了统计.

由图 8 可见,并行化确实更好地利用了系统资源,使 CPU、内存、硬盘这 3 个系统主要资源的利用率都有

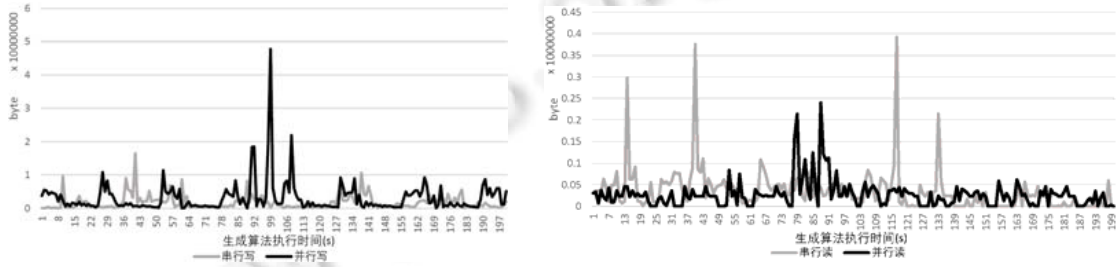
了十分明显的提高.其中,CPU 占用率图中,无论种群串行还是并行,都可以看到明显周期性出现的峰值与低谷,其中,峰值部分对应种群并行化的操作,其他部分对应遗传算法的交叉变异等操作以及数据恢复操作.可以看出,在种群并行化执行过程中,CPU 占用率基本稳定在 99.5%以上,比串行时提高了 1 倍.就存储而言,并行化方法的内存占用率约为 65%~70%,而串行化方法的内存利用率仅在 45%~48%之间;且硬盘的写入量在并行时有较明显的增加,读取量变化不大.可见,硬盘的性能在并行时仍是有空闲的,并不会成为并行化方法的性能瓶颈.因此,相较于 GA 串行化方法,并行化方法可以有效地提高系统资源的利用率.



(a) CPU 占用率



(b) 内存(及虚拟内存)使用率



(c) 硬盘读写量

Fig.8 System resource consumption comparison chart

图 8 系统资源消耗对比图

5 相关工作

5.1 Web应用的测试用例生成方法研究

基于模型的 Web 应用建模和测试用例生成,目前国内外已有一些相关研究进展[8,20].Marchetto 等人[21]给出了一种基于网页 DOM 结构变化的 Web 应用软件建模和测试用例生成方法.Ajax 操纵的网页 DOM 被抽象成一个状态模型,Ajax 请求回调函数的执行被关联到状态的迁移,由状态模型的语义交互事件产生测试用例.Alimadadi 等人[22,23]从理解 JavaScript 代码角度建立了客户端的用户行为模型,针对前端事件之间的时序和因果关系建立模型,以客户端事件为状态,以事件的执行顺序如时间顺序或因果序为迁移建立模型,并基于该模型生成事件序列作为测试用例.Xue 等人[24]提出利用确定性有限自动机(DFA)模型,对同一攻击类型的恶意 JavaScript 的常见行为进行抽象和总结,构建了一个自动行为学习框架,从 JS 恶意软件的动态执行轨迹中学习出 DFA,并结合数据依赖性分析、防御规则和轨迹重放机制,实现了在线恶意行为检测.

但这些 Web 应用模型大多只关注网页的更改及事件,忽略了事件的触发条件以及对参数或 DOM 元素的后续操作.因此,现有的 Web 应用行为模型不能全面表示 Web 应用的动态行为.此外,基于前端模型的测试研究以模型自身的状态/迁移/迁移序列覆盖为目标,不能对后端代码进行有效测试.

Web 应用测试用例生成是保障 Web 应用可靠性与安全性的有效手段,其中,基于搜索的测试用例生成方法被广泛应用于 Web 应用程序.基于搜索的 Web 应用测试用例生成将测试用例生成问题表述为一个优化问题[25].

通常将服务器端代码语句覆盖率、分支覆盖率或安全漏洞覆盖作为覆盖准则,基于覆盖准则设计其适应度.例如,Alshahwan 等人[9]首次将爬山算法应用于 Web 应用软件测试中,以未被覆盖的分支为目标,利用爬山算法,结合 Korel 的交替变量选择法,实现测试数据的自动生成.Elyasov 等人[26]提出了一个以 DOM 作为输入的 JavaScript 进化测试框架.以 JS 目标分支为覆盖指标,结合全局 DOM 状态,利用遗传搜索实现测试数据的自动生成.Mirshokraie 等人[27]为 JavaScript 代码单元测试提供了一个测试框架,可以在事件级和 JavaScript 函数级自动生成 JavaScript 应用的测试用例.Thome 等人[28]针对 SQL 注入漏洞,通过爬虫工具 Crawljax 获得 Web 应用软件的导航页面,跟踪入口 URL 到其他 Web 页面的输入和事件;使用 HTTP 代理来收集客户端和服务器端的交互参数,构建一个有限状态机模型.在 FSM 模型上,利用遗传算法实现测试用例的自动生成.

然而,这些方法没有考虑 Web 应用后端业务逻辑模块间的关系,只能对各个功能模块进行单元测试,难以有效检测二阶注入漏洞.因此,实验室前期研究中[12]提出了基于搜索的 Web 应用前后端融合的测试用例生成方法.然而,当 Web 应用的规模较大时,Web 应用模型中包含的信息量也会很大,给基于模型的测试序列生成性能提升带来了极大的挑战.因此,如何优化基于 Web 应用模型的测试序列生成、提升测试效率,是目前的研究热点.并行化测试用例生成方法是一种有效的提高测试生成效率的手段,但由于 Web 应用测试并行化面对的种种问题,这方面的研究成果还鲜有问世.本文方法将并行化引入到基于 GA 的 Web 应用测试用例生成中,通过设计一种新的并发策略,对个体适应度并行化计算,可以更有效地生成覆盖后端敏感路径的测试用例,提高 Web 应用测试用例生成效率.

5.2 GA并行化测试用例生成研究

目前,基于搜索的技术已经成功应用于软件测试领域,其中,遗传算法 GA 作为全局搜索算法,解决了一系列的测试用例生成问题.然而,由于其计算开销较大,在工业上的应用有限.GA 并行化是改善其性能,提高求解质量的有效途径,已被应用于各种测试用例生成问题中.例如,Geronimo 等人[17]提出了一种基于 Hadoop MapReduce 的 GA 并行化测试用例集自动生成方法,该方法以待测软件和随机解的初始种群作为输入,依据分支覆盖准则并行演化出能覆盖 MapReduce 编程模型分支的测试用例.在每次迭代中,并行计算个体适应度的值.Nugroho 等人[29]采用并行遗传演算法寻找人工神经网络的最佳参数,该方法将并行化应用于种群的适应度评估、选择、交叉和变异等进化过程中.Qi 等人[30]采用一种通用的并行计算平台 SPARK 对遗传算法进行并行处理,给出了一个适应度评价并行化和遗传操作并行化的成对测试用例集生成方法.Gong 等人[31]利用多种群并行遗传算法生成覆盖多路径的测试数据,该方法在对目标路径进行分组的基础上,采用多种群并行遗传算法生成测试数据.

不同于上述方法,本文的测试用例生成方法针对 Web 应用程序,在 Web 应用前端模型上,利用 GA 算法探讨后端敏感路径进行测试用例自动生成.Web 应用前后端融合的测试用例生成方法不同于传统程序.再者,个体的执行需要借助于浏览器模拟用户从前端发送请求至后端,后端代码处理该请求并将响应返回前端.因此在 GA 并行化过程中,需要解决如多浏览器进程管理、后端代码执行路径收集等 Web 应用测试生成中特有的问题.这是传统 GA 并行化技术未涉及的问题.此外,就 GA 并行化技术来说,传统 GA 并行化的线程只涉及适应度计算或个体交叉等操作,线程池模型设计只关注线程的实际执行过程,不涉及线程与进程之间的交互.而本文的 Web 应用 GA 并行化测试用例生成方法中,主线程在为个体分配线程的同时,也分配了相应浏览器进程,使得线程池的管理及调度更加复杂,加大了 Web 应用并行化测试生成的难度.

6 结 论

本文通过线程池管理、多浏览器进程复用及并行化、后端覆盖信息区分、调度逻辑设计等实现了 Web 应用前后端融合的 GA 并行化测试用例生成,解决了现有 Web 应用串行测试用例生成中系统资源不能充分利用、测试生成效率低等问题,减少了 Web 应用测试用例生成的时间开销,使并行化成功应用于 Web 应用测试用例生成.实验结果表明:本文提出的 GA 并行化方法能够生成覆盖后端敏感路径的测试用例,且与现有的 Web 应用前后端融合的 GA 串行化测试用例生成方法相比,并行化方法的测试用例生成的种群平均执行时间减少了 81.1%,平均 CPU 利用率提高了 1 倍.下一步将尝试对 Web 应用 GA 测试用例生成过程中的种群遗传操作及更新操作

进行并行化,从而进一步优化生成效率.

References:

[1] CNNVD. Information Security Vulnerability Report. Vol.473, Beijing: China Information Technology Security Evaluation Center, 2019 (in Chinese with English abstract).

[2] Doğan S, Betin-Can A, Garousi V. Web application testing: A systematic literature review. Journal of Systems and Software, 2014, 91:174−201.

[3] Lee TK, Wei KT, Ghani AAA. Systematic literature review on effort estimation for open sources (OSS) Web application development. In: Proc. of the 2016 Future Technologies Conf. (FTC). San Francisco: IEEE, 2016. 1158−1167.

[4] Schur M, Roth A, Zeller A. Mining workflow models from Web applications. IEEE Trans. on Software Engineering, 2015,41(12): 1−1.

[5] Alimadadi S, Sequeira S, Mesbah A, Pattabiraman K. Understanding JavaScript event-based interactions. In: Proc. of the Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 367−377.

[6] Avancini A, Ceccato M. Security testing of Web applications: A search-based approach for cross-site scripting vulnerabilities. In: Proc. of the 2011 IEEE 11th Int'l Working Conf. on Source Code Analysis and Manipulation. Williamsburg: IEEE, 2011. 85−94.

[7] Zhao RL, Lyu MR, Min YH. Automatic string test data generation for detecting domain errors. Software Testing, Verification and Reliability, 2010,20(3):209−236.

[8] Li YF, Das PK, Dowe DL. Two decades of Web application testing—A survey of recent advances. Information Systems, 2014,43: 20−54.

[9] Alshahwan N, Harman M. Automated Web application testing using search based software engineering. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lawrence: IEEE, 2011. 3−12.

[10] Ahmed MA, Ali F. Multiple-path testing for cross site scripting using genetic algorithms. Journal of Systems Architecture, 2016,64: 50−62.

[11] Marashdih AW, Zaaba ZF, Omer HK. Web security: Detection of cross site scripting in PHP Web application using genetic algorithm. Int'l Journal of Advanced Computer Science and Applications (IJACSA), 2017,8(5):64−75.

[12] Wang WW, Guo XH, Li Z, Zhao RL. Test case generation based on client-server of Web applications by memetic algorithm. In: Proc. of the 2019 IEEE 29th Int'l Symp. on Software Reliability Engineering (ISSRE). Berlin: IEEE, 2019. 206−217.

[13] Guo XH. Sensitive paths oriented automatic test cases generation for Web applications based on client EFSM [MS. Thesis]. Beijing: Beijing University of Chemical Technology, 2018 (in Chinese with English abstract).

[14] Wang WW, Guo JX, Li Z, Zhao RL. EFSM-oriented minimal traces set generation approach for Web applications. In: Proc. of the 2018 IEEE 42nd Annual Computer Software and Applications Conf. (COMPSAC). Tokyo: IEEE, 2018. 12−21.

[15] Ahmad SG, Liew CS, Munir EU, Ang TF, Khan SU. A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. Journal of Parallel and Distributed Computing, 2016,87:80−90.

[16] Roberge V, Tarbouchi M, Labonté G. Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. IEEE Trans. on Industrial Informatics, 2012,9(1):132−141.

[17] Di Geronimo L, Ferrucci F, Murolo A, Sarro F. A parallel genetic algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites. In: Proc. of the 5th IEEE Int'l Conf. on Software Testing, Verification and Validation. Montreal: IEEE, 2012. 785−793.

[18] Chen JC, Xue YZ, Zhao C. Approach for GUI testing based on event handler function. Ruan Jian Xue Bao/Journal of Software, 2013,24(12):2830−2842 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4399.htm [doi: 10.3724/SP.J.1001.2013.04399]

[19] Medeiros I, Neves N, Correia M. Detecting and removing Web application vulnerabilities with static analysis and data mining. IEEE Trans. on Reliability, 2015,65(1):1−16.

[20] Lebeau F, Legeard B, Peureux F, Vernotte A. Model-based vulnerability testing for Web applications. In: Proc. of the 6th Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). Luxembourg: IEEE, 2013. 446−452.

[21] Marchetto A, Tonella P, Ricca F. State-based testing of Ajax Web applications. In: Proc. of the 2008 1st Int'l Conf. on Software Testing, Verification, and Validation. Lillehammer: IEEE, 2008. 121−130.

[22] Alimadadi S, Sequeira S, Mesbah A, *et al*. Understanding JavaScript event-based interactions with clematis. ACM Trans. on Software Engineering and Methodology, 2016,25(2):12.1−12.38.

[23] Alimadadi S. Understanding behavioural patterns in JavaScript. In: Proc. of the ACM Sigsoft Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 1076−1078.

[24] Xue YX, Wang JJ, Liu Y, Xiao H, Sun J, Chandramohan M. Detection and classification of malicious JavaScript via attack behavior modelling. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. Baltimore: ACM, 2015. 48−59.

[25] Jan S, Nguyen CD, Arcuri A, Briand L. A search-based testing approach for XML injection vulnerabilities in Web applications. In: Proc. of the 2017 IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Tokyo: IEEE, 2017. 356−366.

[26] Elyasov A, Prasetya I, Hage J. Search-based test data generation for JavaScript functions that interact with the DOM. In: Proc. of the 2018 IEEE 29th Int'l Symp. on Software Reliability Engineering (ISSRE). Memphis: IEEE, 2018. 88−99.

[27] Mirshokraie S, Mesbah A, Pattabiraman K. JSEFT: Automated Javascript unit test generation. In: Proc. of the IEEE Int'l Conf. on Software Testing. Graz: IEEE, 2015. 1−10.

[28] Thomé J, Gorla A, Zeller A. Search-based security testing of Web applications. In: Proc. of the 7th Int'l Workshop on Search-based Software Testing. Hyderabad: ACM, 2014. 5−14.

[29] Nugroho ED, Wibowo ME, Pulungan R. Parallel implementation of genetic algorithm for searching optimal parameters of artificial neural networks. In: Proc. of the Int'l Conf. on Science & Technology-Computer. Yogyakarta: IEEE, 2017. 136−141.

[30] Qi RZ, Wang ZJ, Li SY. A parallel genetic algorithm based on spark for pairwise test suite generation. Journal of Computer Science and Technology, 2016,31(2):417−427.

[31] Gong DW, Tian T, Yao XJ. Grouping target paths for evolutionary generation of test data in parallel. Journal of Systems and Software, 2012,85(11):2531−2540.

附中文参考文献:

[1] 国家信息安全漏洞库.信息安全漏洞周报.Vol.473,北京:中国信息安全测评中心,2019.

[13] 郭小红.面向 Web 服务器端敏感路径的客户端 EFSM 测试生成[硕士学位论文].北京:北京化工大学,2018.

[18] 陈军成,薛云志,赵琛.一种基于事件处理函数的 GUI 测试方法.软件学报,2013,24(12):2830−2842. http://www.jos.org.cn/1000-9825/4399.htm [doi: 10.3724/SP.J.1001.2013.04399]

王微微(1990－),女,河北沧州人,博士生,CCF 学生会员,主要研究领域为软件测试,Web 应用测试,基于搜索的测试用例生成.

赵瑞莲(1964－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件可靠性.

李奕超(1997－),男,学士,主要研究领域为软件测试.

李征(1974－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为基于搜索的软件工程,软件测试,源代码分析.