基于历史的云平台故障注入测试

马 骅 聂长海 吴化尧

(南京大学计算机软件新技术国家重点实验室 南京 210046)

摘 要 云计算是一种能够以便利的、按需付费的方式通过网络获取计算资源并提高其可用性的模式.近年来,以云计算为基础的服务平台——云平台逐渐成为各大企业数据存储和业务部署的主要平台.由于云平台结构复杂、服务多样,发生故障在所难免.为了提高云平台的可靠性,开发人员在设计云平台时加入了容错机制,目的是在发生故障的情况下也能保证云平台的正常运行.但是容错机制并不能保证云平台完全可靠,因此我们还需要对云平台的可靠性进行检验.故障注入是检验云平台可靠性的方法之一,通过人为地将故障注入正在运行的系统中,观察系统动作并判断系统的容错机制是否正常工作.现有的故障注入方法侧重于分析待测系统特征以确定故障注入点,属于白盒或灰盒测试,对复杂的云平台来说,这一工作无疑要耗费大量的时间.因此,我们提出一种不依赖于系统信息的黑盒测试方法以提高检验效率.本文在现有工作的基础上做了以下几个方面的工作:第一,我们收集了云平台历史宕机事故报告,并分析其中故障模式出现的规律.我们发现,云平台中发生的故障类型具有重复性,在此基础上,我们提取了这些故障的特征,包括所在组件、根因、产生的影响、修复方法等;第二,通过对云平台历史宕机事故报告的分析,我们发现很多事故当中的故障并不是单一出现的,并且多个故障之间具有关联性、组合性,我们深入分析了多故障之间的关系以及故障之间的组合形式,在此基础上,为了尽可能完全地检测云平台的可靠性,我们提出在故障注入过程中需要对多故障进行组合注入;第三,在对多故障进行组合的过程中我们发现,由于云平台的复杂性,故障种类的多样性,多故障之间的组合会产生组合空间爆炸问题,针对这一问题,我们做了初步探究,并提出了几种约减策略;第四,基于上述工作,我们提出了一种基于历史的故障组合方法,并利用历史故障数据,结合基础云平台架构进行模拟实验,实验结果表明我们提出的基于历史故障进行故障组合注入方法是有效可行的.

关键词 云平台;故障模式;历史故障;故障注入

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.02281

History-Based Fault Injection Testing for Cloud Platform

MA Hua NIE Chang-Hai WU Hua-Yao

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046)

Abstract Cloud computing is a model which can access to a pool of configurable computing resources that can be shared in a convenient, on-demand way through internet. These years, based on cloud computing, cloud platform which composed of foundation, a group of infrastructure services and some dedicated application services, has become a main platform for the deployment of enterprise applications and data storage. Due to the complexity of cloud platform architecture, and the diversity of services it provided, failure is difficult to avoid. In order to improve the reliability of the cloud platform, developers have added fault-tolerance mechanism when designing the cloud platform, the purpose is to ensure that even there is a failure in the cloud platform, it also have a normal operation performance. However, this fault-tolerance mechanism does not guarantee that the cloud platform is completely reliable. Therefore, we also need to test the reliability of the cloud platform. Fault injection is one of the ways to test the reliability of a cloud platform, by artificial-

收稿日期:2018-02-02;在线出版日期:2019-03-04.本课题得到国家重点研发计划(2018YFB1003800)资助.**马 骅**,硕士,主要研究方向为故障注入测试.E-mail: jupiterma@vip.qq.com.**聂长海**(通信作者),教授,博士生导师,主要研究领域为软件测试与软件质量保证.E-mail: changhainie@nju.edu.cn.**吴化尧**,博士,主要研究方向为软件测试.

ly injecting faults into the system under test, observing the system actions and determining if the system's fault tolerance mechanism is working properly. However, most of the existing fault injection methods, which focus on analyzing the characteristics of the system under test to determine the fault injection location, belong to white or gray box testing. These methods will take a long time due to the complexity of the cloud platform. Therefore, we propose a black box testing method that does not depend on system analyzing in order to improve test efficiency. We have done the following works based on the existing works. First of all, we have collected historical cloud outage reports on cloud platforms and analyzed the characteristics of the failure modes which appears in reports. We have found that the types of failures in the cloud platform are repetitive. Based on this, we thoroughly analyzed the characteristics of these failures, including the components, root causes, impacts, and methods of repairing. Secondly, through the analysis of historical outage reports on cloud platforms, we have found that failure in many accidents does not occur alone, and that multiple failures are related and combined. We have deep analysed the relationships of multiple failures, and the combination relationships, based on this, in order to detect the reliability of the cloud platform as completely as possible, we propose that multiple faults must be combined and injected during the fault injection process. Thirdly, in the process of combining multiple faults, we discovered that due to the complexity of the cloud platform, the diversity of fault types, and the combination of multiple faults, the problem of combinatorial space explosion will arise. To address this issue, we have done preliminary exploration and proposed several reduction strategies. Fourthly, based on the above work, we propose a history-based fault combination method, using historical fault data, combined with the basic cloud platform architecture to conduct simulation experiments. The experiment results show that the proposed fault combination injection method based on historical faults is effective and feasible.

Keywords cloud platform; failure mode; historical faults; fault injection

1 引 言

云平台为企业应用及 Web 服务提供了一种具有革命性意义的新模式,该模式中云使用者节省了购买及管理基础设施的费用,而云提供者则节省了维护未充分使用的 CPU、存储和网络等所耗费的资源.在此模式下,越来越多的互联网企业将自己的产品及服务部署于云平台之上,云平台的可靠性直接影响着这些应用服务的稳定性,如 2017 年 2 月 28 日,亚马逊位于美国北弗吉尼亚的云存储服务器出现故障,导致使用该服务器的数千个网页完全无法访问,大量 app 功能失效,十多万个网页中部分链接失效且图片无法显示①.为了避免类似事故的发生,人们越来越重视云平台的可靠性.故障注入作为检验云平台可靠性的常用方法也备受人们关注[1-2].故障注入,即人为地向云平台中注入若干故障,若云平台依然能够正常工作,则可以认为云平台具有较好的容错性,其可靠性较高,若不能正常工作,则该云平台容错能力不足,可靠性欠佳.

1.1 研究动机

通过故障注入来检验云平台可靠性这一方法有两个主体,待注入的故障和待检验的云平台.通过对相关研究工作的调研我们发现,现有的故障注入方法和工具大多数是从待测云平台的角度出发,或通过对系统相关信息如系统执行路径[3],系统调用或函数调用的执行点以及上下文信息[4]等进行分析、计算,得到故障注入点;或通过执行系统所有可能的事件序列以暴露系统中难以发现的故障(如传统的模型检查方法)[5].在结构复杂的云平台环境中,这些方法无疑会耗费大量的时间.

为了进一步完善云平台可靠性检验、节省云平

① http://tech.huanqiu.com/internet/2017-03/10232719.html

台故障注入测试时间，本文从分析故障模式特征的角度出发，利用已有的故障模式库作为故障注入的原材料，提出一种不依赖于系统信息的黑盒云平台故障注入测试方法.

1.2　研究目标

本文主要的研究目标是分析并回答以下几个问题，以探究基于故障模式的故障注入方法的可行性：

(1) 历史故障数据如何使用？在检验云平台的可靠性时，有什么依据能够支持我们在已有的故障模式库中选取待注入的故障模式？

(2) 故障模式之间的组合有没有意义？在检验云平台的可靠性时，是否有必要考虑多个故障模式之间的组合注入？

(3) 如何进行故障模式之间的组合？故障模式库中的众多故障模式依据什么策略进行组合？

(4) 组合空间爆炸问题如何解决？在复杂的云平台背景下，多故障组合必然会产生组合空间爆炸问题，应该如何约减组合空间？

1.3　研究过程

本文的研究工作主要分为以下几个阶段：

(1) 故障模式收集与特征分析阶段. 我们主要通过两条途径收集故障模式，首先，我们按照云平台不同的层次划分，收集每一层相关组件的故障模式；其次，我们检索各云平台服务商历年来发布的宕机事故报告，从事故报告中提取相应的故障模式. 针对收集到的故障模式进行特征分析，为将来故障模式的选取和组合提供依据. 一般来说，宕机事故是由于某个故障导致系统性能下降，进而导致服务不可用. 例如在附录[案例 28]中，黑莓仅仅在北美遇到了间歇性的服务延迟，最终演变成了包括中东、欧洲、非洲、南美和亚洲的服务宕机事故. 因此我们在分析云平台历史宕机事故报告时，面对的不仅仅是表面的宕机事故，更多的是引起宕机事故的微小的故障. 我们抽取的特征主要包含：故障发生所在的组件、修复手段、发生原因、产生影响以及一些特殊的约束关系等. 与此同时，我们通过分析宕机事故报告，进一步探究云平台中多故障组合的一般规律；

(2) 提出基于历史的故障组合方法. 基于对故障模式的特征分析结果，我们初步提出了一个关于多故障模式之间相互组合的方法，该组合方法适用于在维护有一个历史故障模式库的前提下，在云平台的硬件层面，针对云平台中的各个组件，进行多组件之间故障模式的组合，得到相应的待注入故障序列，为故障注入以验证云平台各组件的可靠性做准备；

(3) 组合空间约减阶段. 在提出组合算法的过程中，由于云平台规模庞大且故障模式数量众多，故障模式间的组合会产生组合空间爆炸问题，因此我们将组合测试覆盖表生成相关理论应用在组合方法中以约减组合空间；

(4) 实验验证阶段. 由于条件限制，我们无法搭建完整复杂的云平台实验环境，因此我们选择进行模拟仿真实验. 我们以 Oracle 的简单云平台架构为检验对象，利用基于历史的故障组合方法得到若干待注入故障组合集合，分析该集合并得出实验结论.

1.4　本文贡献

本文贡献主要包括以下几点：

(1) 在已有开源故障模式数据库的基础上，通过多种途径收集整理了与云平台有关的故障模式数据库. 我们将该故障模式数据集以一个故障模式管理系统进行管理，可为后续其它工作参考使用，下载地址：http://gist.nju.edu.cn/~mahua/mh.html；

(2) 通过分析云平台历史宕机事故报告，我们发现导致云平台宕机的故障是重复出现的，也就是说，利用旧的故障来检验新产品的可靠性是合理的，但这种方法有其局限性，由于故障的分析依据来源于历史数据，因此我们只能够以这些曾经出现过的故障为原材料设计待注入故障序列. 与此同时，我们发现要验证云平台的可靠性，多故障组合进行故障注入必不可少，而在云这一特殊的环境下，故障之间的组合形式是有规律可寻的，利用好这些规律可以极大地约减组合空间(第 2 节)；

(3) 基于以上发现，本文提出了一种基于历史的故障组合与约减方法(第 3 节)，并以一个简单的云平台架构为例进行了相应的实例研究(第 4 节).

1.5　本文组织

本文第 2 节主要介绍云平台故障模式的收集途径、特征分析以及对云平台事故报告的分析；第 3 节简要介绍基于历史的故障组合方法与几种约减组合空间的策略；第 4 节通过实例研究梳理基于历史的云平台故障注入测试的具体流程；第 5 节介绍相关工作的分析与比较；第 6 节是对本文的总结以及未来工作的展望.

2　故障模式收集与特征分析

2.1　故障模式收集

我们主要通过三种方式收集故障模式，包括分层次收集故障模式、分析宕机事故报告收集故障模

式和利用开源数据库收集故障模式.

在分层次收集故障模式阶段,我们对云平台划分的 3 个层次,即基础设施层、虚拟化层和应用层,并做了进一步的划分(如图 1 所示).基础设施层和虚拟化层是各种不同类型云计算平台的基础,该层次的故障模式具有通用性,因此该层次针对基础设施层和虚拟化层中的常见组件(例如路由器、负载均衡器、虚拟存储、虚拟网络等),我们通过查找相应技术手册,收集了这些组件在使用过程中可能出现的故障.

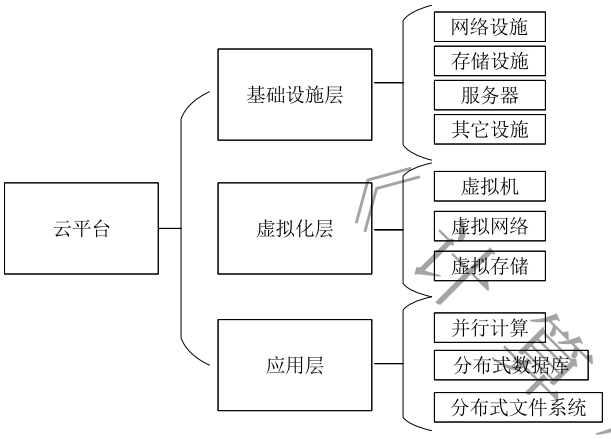

图 1 云平台层次划分

在分析宕机事故报告收集故障模式阶段,我们检索了亚马逊云平台、微软云平台、Office 365、苹果云平台、腾讯微云、阿里云平台等多家云平台服务提供商历年来发生的重大宕机事故报告和相关文章,从中抽取引起事故的故障模式.

在利用开源数据库收集故障模式阶段,我们使用了 CosDB,该数据库由 Gunawi 等人于 2016 年建立,其中记录了 32 个云计算平台的 597 条宕机事故报告[6].为了便于我们后续分析云平台中多组件多故障之间的关系,我们以涉及多个故障为准则,从上述宕机事故报告中挑选出共计 27 条多故障事故报告作为说明本文得出的结论的案例(附录).

2.2 故障模式特征分析

针对收集到的故障模式,我们给出一些用来描述故障模式的特征,依据这些特征可以帮助我们更加合理地利用历史故障数据,例如对故障模式进行归类、组合、约减.

(1) ID. 我们给每个故障模式一个身份编号,用数字来代表该故障模式;

(2) Component. 该特征用来描述故障模式是在云平台的哪个组件上发生的,可作为后续组合算法中的输入参数之一.我们参考云平台的分层结构,从硬件组件的角度,根据组件不同的功能进行划分;

(3) Fix. 该特征用来描述故障的修复方法,该特征可与 impact 特征共同应用于组合算法的结果约减.我们分析了云平台事故报告对故障的处理过程,并提取出了一些常见的修复故障的方法如下:

① ADDRESOURCES. 即引起故障的原因可能是计算资源、存储资源、网络资源短缺,因此修复方法是增加相应的资源.

② FIXHW. 即引起该故障的原因可能是硬件问题,因此修复方法是修复硬件.

③ FIXSW. 即引起该故障的原因可能是软件问题,因此修复方法是修复软件.

④ FIXCONFIG. 即引起该故障的原因可能是配置问题,因此修复方法是修复配置文件.

⑤ RESTART. 有些故障的修复可能需要重启受影响的组件.

⑥ RESTOREDATA. 通过数据转存或数据重定向等手段修复故障.

⑦ ROLLBACKSW. 通过回滚操作修复故障.

⑧ UNKNOWN. 以及一些未知的修复操作,适用于人为引起的故障.

(4) Root cause. 该特征用来描述故障的根本原因.我们将云平台事故报告中关于事故发生的原因做了归纳总结,提取出其中与云平台可靠性相关的根因,并以我们收集的事故报告为样本库,统计了每个根因发生的比例,利用该比例可以对组合算法中的待注入故障序列结果进行优先级的排序,确保经常发生的故障会在第一时间被检测出来.具体如下所示:

① Upgrade:所占比例为 16%.更新维护云平台硬件[案例 10、11、13]、升级固件[案例 6、16]、更换过期证书[案例 15]时可能会引起一些故障.

② Network:所占比例为 15%.网络设施是云平台提供服务的基础[案例 19],在设计之初就充分考虑了其容错性(如一级网络和二级网络),因此,由网络问题引起的事故常常是多个故障组合导致[案例 3].

③ Config:所占比例为 10%.错误的配置有时同样会引发云平台故障[案例 12、17、18].通常来说,在云平台的维护、升级、改造过程中,由于相关人员的疏忽导致系统配置不当,进而导致严重的后果.

④ Load:所占比例为 9%.过高的负载可能会导

致严重的事故. 云平台面向的是大规模的、高并发的服务调用，这就意味着云平台中的组件长时间工作在高功率、高负荷的状态下. 云平台中的电源设备[案例 1]、网络设备[案例 7、26、27]、服务器[案例 8、21]、内存[案例 20]、存储设备都有发生过载的风险.

⑤ Power：所占比例为 6%. 云平台中成千上万的硬件设备需要稳定可靠的、高负载量的电力系统. 电力系统的故障往往会导致云平台全面宕机的严重后果，如在案例 1 中，供电系统的故障导致了亚马逊云在美国东部的区域发生服务中断.

⑥ Storage：所占比例为 4%. 存储节点一般可以是云平台中的硬件存储组件[案例 9]、存储集群[案例 14、24]、文件系统、数据库系统[案例 25]、虚拟存储等. 存储结点的可靠性是云平台可靠性最重要的组成部分.

⑦ Server：所占比例为 3%. 服务器在云平台中扮演多种角色，不同类型的服务器负责不同的任务，共同实现云平台的功能，因此服务器发生故障会直接导致云平台服务不可用[案例 23].

⑧ Hardware：所占比例为 1%. 硬件原因引起的故障是云平台基础设施层最常见的故障原因之一. 我们将事故报告中并未明确说明故障来源而仅提到是由硬件设备引发的故障标记为 Hardware.

(5) Impact. 该特征用来描述该发生故障后对云平台产生的影响. 根据云平台事故报告中的描述，我们将故障对云平台的影响分为以下几类：

① FULLOUTAGE. 表明该故障会对云平台造成极为严重的影响，导致组件或云平台宕机[案例 1、2].

② OPFAIL. 表明该故障会使某次关键操作无效化[案例 8].

③ PERFORMANCE. 表明该故障会使云平台某硬件设备的性能下降[案例 4、6、10].

④ LOSS. 表明该故障会导致数据丢失，包括服务器、存储器以及网络链路中的数据[案例 5].

⑤ STALE. 表明该故障会导致数据过期、不一致等后果[案例 6].

⑥ SECURITY. 表明该故障会为平台带来安全上的问题[案例 15].

⑦ AftID. 该特征表明在故障 ID 为 AftID 的故障发生之后该故障才会发生. 常见的故障场景如满足因果关系的两个故障，存在故障发生的先后顺序.

⑧ nAftID. 该特征表明在故障 ID 为 nAftID 的故障发生之后该故障绝不会发生. 常见的故障场景如服务器发生宕机故障，那么关于该服务器的其它故障就不会发生.

2.3　云平台宕机事故报告分析

云平台当中的故障是重复出现的. 通过对云平台历史事故报告的调研分析，我们发现同样的故障可能会在不同的时间地点甚至不同的云平台上再次发生. 例如[案例 1]与[案例 2]中，两次事故都是由于公共电源断电后，不间断供电系统出现故障导致云平台宕机事故. 云平台中故障的重复性是必然的. 首先，云平台内部结构十分复杂，功能相同、互为冗余的组件很多，这些组件发生的故障往往是相同的；其次，各个云平台之间有着相同或相似的架构，提供类似的服务，这些相似性决定了不同云平台之间可能会发生相同的故障. 因此，利用历史上曾经出现过的故障作为故障注入的原材料是合理的.

云平台当中的故障大多数是组合出现的. 我们在分析事故报告时发现，由单点故障而导致的宕机事故十分罕见，绝大多数事故是由多个故障共同作用引发的，由此不难看出，检验云平台可靠性时考虑多个故障的交互是必不可少的. 我们进一步分析了涉及多故障的事故报告，试图找出多个故障之间可能存在的组合关系，并将其分类如下：

(1) 顺序发生

顺序发生是指在某次事故当中，引发此次事故的若干个故障是按一定顺序依次发生的，多个故障之间存在一定的关系，这种关系可能是：

① 多个故障之间存在因果关系，由前一个故障导致了后一个故障[案例 3].

② 在前一个故障恢复的过程中产生了新的故障[案例 23].

(2) 组合发生

组合发生是指在某次事故中，引发此次事故的若干个故障必须同时发生，可以分为以下两种情况：

① 某一组件和其容错组件同时发生故障[案例 5].

② 某两个故障 A、B 同时发生，且 A、B 之间有依赖关系(A 所在组件能否正确运行需要依赖于 B 所在组件)[案例 6].

2.4　发　现

通过对故障模式特征的分析，我们抽取了故障模式的一般特征及故障模式之间的关系特征，利用这些特征，我们可以在生成的待注入故障序列集合的基础上使用一些约减策略，并且最终我们可以根据根因当中的故障出现频率对待注入故障序列集合

中的待注入故障序列进行排序，将最有可能发生的故障尽早注入系统，以期尽早发现云平台的可靠性问题.

通过对云平台事故报告的分析，我们发现了云平台当中的故障(尤其是硬件层面的故障)是重复发生的，经常重复发生故障的组件多数是一些负责网络流量(如路由器)、供电系统、提供特殊功能(如服务器)的组件，在我们收集的宕机事故报告中，这三种类型的故障占比达到了 24%. 因此，利用历史故障数据来指导我们检验新云化产品的可靠性是可行的；除此之外，我们发现由于现代云平台的容错措施越来越完善，严重的宕机事故往往是由多个故障组合发生导致的，因此，我们有必要探究故障模式之间的组合关系，目的是指导我们做多故障的故障注入工作.

3 基于历史的故障组合与约减方法

我们在分析已有故障模式的基础上，提出了基于历史的故障组合方法，该组合方法适用于在维护有一个历史故障模式库的前提下，在云平台的硬件层面，针对云平台中的各个组件，进行组件之间故障模式的组合，得到相应的待注入故障序列，为故障注入以验证云平台各组件的可靠性做准备. 该组合方法的目标是基于已有的故障模式库：(1) 针对待测组件找出合适的待注入故障模式组合；(2) 尽可能减少待注入故障模式的组合空间；(3) 保证尽可能高的故障模式覆盖率.

该组合方法一共分为 3 个阶段，分别是系统待测组件选取阶段、故障模式选取阶段、组合算法应用阶段. 以下详细介绍每个阶段的主要内容.

3.1 系统待测组件选取阶段

在待注入故障组件的选取阶段，我们的目的是利用云平台拓扑结构图，通过某种选择策略，选出最大注入故障数 $maxfault$(简记为 mf)个目标组件对集合 $A=\{(A_{11},A_{12},\cdots,A_{1mf}),\cdots,(A_{i1},A_{i2},\cdots,A_{imf})\}$，其中 A 代表一个组件，$(A_{i1},A_{i2},\cdots,A_{imf})$ 表示第 i 个组件对，其中共有 mf 个组件，在本文中我们规定每个组件每次注入工作只注入一个故障.

其中，选择策略可以是：

(1) 对拓扑图中的所有节点进行全组合，假设 $mf=2$，即系统中所有组件两两组合都将作为一组待注入故障的组件对集合.

优点：既考虑了组件之间的关系，向有数据通信的组件中组合注入故障，也考虑了没有直接数据通信的组件之间的相互作用，更容易暴露系统中可能存在的问题.

缺点：由于云平台的复杂性，组件之间的全组合很多，增加测试的复杂性.

适用场景：云平台中需要重点测试的少部分重要组件.

(2) 通过广度优先遍历拓扑图中的所有节点.

优点：与全组合相比，测试规模较小，并且能够尽可能快地检查系统当中某一部分的组件.

缺点：仅考虑存在数据通信的组件之间的组合，忽略了虽然组件之间没有直接数据通信但依然可能相互作用对系统整体产生影响的情况.

(3) 通过深度优先遍历拓扑图中的所有节点.

优点：深度优先遍历的优点在于能够尽可能快地对一条数据通路或一个完整的功能进行遍历.

缺点：与广度优先相似，忽略了虽然组件之间没有直接数据通信但依然可能相互作用对系统整体产生影响的情况.

3.2 故障模式选取阶段

在故障模式选取阶段，我们的目的是为每一个组件选取出可以注入的故障模式集合. 假设待测组件为 A_i，从故障模式库中选取所有满足条件“Component=A_i”的故障模式加入该组件相应的故障模式集合 $B_i=\{b_{i1},b_{i2},\cdots,b_{in}\}$，其中 B_i 表示第 i 个故障模式集合，b_{ij} 表示第 i 个故障模式集合中的第 j 个故障模式.

3.3 组合方法应用阶段

在组合方法应用阶段，我们根据待测组件集合 A 和相应的故障模式集合 B，利用组合测试当中的覆盖表生成算法，生成 mf 维覆盖表，作为待注入的故障序列. 具体来说，我们将待测组件集合记作 $A=\{A_1,A_2,\cdots,A_n(n\leq N)\}$，每个组件相应的故障模式集合 $B=\{B_1,B_2,\cdots,B_n\}$ 作为输入，集合 A 可以看作是待测软件的参数，而集合 B 可以看作是每个参数对应的取值，应用组合测试覆盖表生成方法最终得到关于云平台组件集合与待注入故障模式集合的覆盖表，即待注入故障序列集合.

覆盖表 $CA(M;t,k,v)$ 用以描述测试用例集，是一个值域大小为 v 的 $M\times k$ 矩阵，任意的 $M\times t$ 子矩阵包含了在 v 值域上所有大小为 t 的排列. 这里 t 被称为强度，k 被称为阶数，v 称为序. 在本文中，在

该待注入故障模式序列所构成的覆盖表中，M 表示所有的待注入故障序列，t 与 k 相等，表示所选组件的个数，即 mf，v 表示每个组件对应的故障模式集合中故障的个数.

得到该覆盖表之后，我们可以根据根因的统计信息对每条待注入故障序列进行排序. 即提取出每条待注入故障序列中每个故障模式的根因，将这些根因发生的概率相加得到该条待注入故障序列发生的概率，概率越高意味着该待注入故障序列中的故障越容易发生，可以优先进行注入以减少测试成本.

由于我们将待测组件看作参数，将组件上可能发生的故障看作参数的取值，在这个模型下使用覆盖表生成算法会产生一个问题：每条测试用例中每个参数只能有一个取值，即每次注入任务中每个组件只能注入一个故障. 但是由于同类型的组件在系统中是重复出现的，因此在一次注入任务中同一个故障可能会重复注入到不同位置的同类型组件中.

多故障组合方法会带来一个问题：组合空间爆炸. 假设现在有一个大规模云平台，假设有一测试任务，需要测试该云系统的某一小部分共计 100 个组件（包括服务器、磁盘、路由器、负载均衡器等），每个组件有 10 种故障场景（crash、degree、data corruption 等），若每次最多只向其中任意 3 个组件中各注入 1 个故障、不考虑各个故障发生的时间顺序，则该组合空间为 $C_{100}^{3}\times 10^{3}=1.617\times 10^{8}$，即共需 1.6 亿次故障注入. 可以看出，在待测目标组件和相应故障场景规模都比较大的情况下，就会产生组合空间爆炸问题，因此我们需要找到一个行之有效的方法对该组合空间进行约减.

3.4　约减故障模式组合空间

通过对云平台事故报告及故障模式特征的分析，我们总结出了以下几种策略以约减组合空间：

（1）根据故障模式标签中的“AftID”可确定约束类型 1.

假设组件 A_1 中的故障模式 b_1，拥有标签“AftID”，且其值为 b_2，那么：

① 若此时存在组件 A_2，其恰好有故障模式 b_2，那么需要保证 b_2 先被注入，再注入 b_1，不符合这种顺序的待注入故障序列可以被约减.

② 若所有的组件均没有故障模式 b_2，则组件 A_1 中含有 b_1 的待注入故障模式序列可以被约减.

该策略伪代码如下：

```
Begin
输入：待注入故障序列集合 C={c1,c2,…,cp}，其中 p
      表示该集合中故障序列的条数，ci 表示一条待注
      入故障序列，由 mf 个故障组成
For 任意故障 b1∈ci
  If b1.AftID==b2 and b2∈ci and b2 位于 b1 之前
    保留 ci
  Else
    删除 ci
End
```

适用场景：该约束类型大多发生在故障及其容错路径之间，能够约减一部分容错路径（包括故障监控/检测系统、故障恢复机制）上的故障与其它不相关故障之间的组合.

例如：公共电源故障和不间断供电系统之间就存在这种关系. 不间断供电系统是公共电源的容错系统，当公共电源系统正常时，向不间断供电系统中注入故障是无意义的，因此一些不相关的故障（比如路由器的故障、磁盘的故障等）与不间断供电系统故障相组合就可以被约减.

（2）根据故障模式标签中的“nAftID”可确定约束类型 2.

假设组件 A_1 中的故障模式 b_1，拥有标签“nAftID”，且其值为 b_2，那么：

若此时存在组件 A_2，其恰好有故障模式 b_2，组件 A_1 的故障模式 b_1 不能在 b_2 之后被注入；

该策略伪代码如下：

```
Begin
输入：待注入故障序列集合 C=(c1,c2,…,cp)，其中 p
      表示该集合中故障序列的条数，ci 表示一条待注
      入故障序列，由 mf 个故障组成
For 任意故障 b1∈ci
  If b1.nAftID==b2
    If (b2∈ci and b1 位于 b2 之前) or b2∉ci
      保留 ci
    Else
      删除 ci
End
```

适用场景：该约束类型大多发生在一些特殊场景下.

场景 1. 某些故障（如断电、过载、驱动损坏、升级故障等等）会导致路由器 crash（即该故障的标签 Impact==FULLOUTAGE），那么此时再注入该路由器本身的故障（即 Component==该路由器 id）、其它组件与该路由器在数据交互时的故障（如某服务器给该路由器发送的数据包丢失等故障），是无意义的.

场景 2. 某些故障会导致组件的某些关键操作(如登录、支付、搜索)失效(即该故障的标签 Impact==OPFAIL),那么此时再注入与该操作有关的故障(如登录之后修改信息时的故障)是无意义的.

(3)在同一个组件中,若两个故障产生的影响相同、修复方法也相同,那么可以把这两个故障归为一类.即针对故障模式集合 B_i 中的任意两个故障模式 b_j.Fix==b_k.Fix&&b_j.Impact==b_k.Impact,则 b_j 与 b_k 可归为同一类故障.

该策略伪代码如下:

```
Begin
输入:故障模式集合 B={B1,B2,…,Bn}
For 任意故障 b1, b2 in Bi
  If b1.Fix==b2.Fix and b1.Impact==b2.Impact
    b1,b2合并为一个故障 b
    Delete b1,b2 from Bi
    Insert b into Bi
  Else
    保留 b1,b2
End
```

4 实例研究

我们通过一个实例研究来验证本文的工作.在该实例研究中,我们使用本文提出的基于历史的故障组合方法,针对某一云平台的基础设施组件,得到若干相应的待注入故障模式序列,并将这些故障序列与我们收集的历史数据进行比较,判断故障序列中是否包含了曾导致云平台宕机事故的故障,以此来判断此次故障注入是否能够发现云平台的缺陷.

4.1 研究对象

根据对云平台宕机事故报告的分析统调研我们发现,与存储、网络相关的组件比较容易发生故障,基于此我们选择以一个简单的 Oracle 云平台典型物理架构(图 2)作为模拟实验的验证对象.该 Oracle 云平台典型物理架构主要包括以下 9 种基础设施:光纤收发器、防火墙、VPN 服务器、云平台管理服务器、云软件管理服务器、云计算物理服务器、千兆以太网交换机、光纤交换机、存储设备.在本实例研究中我们故障注入的对象即为这 9 种基础设施组件.

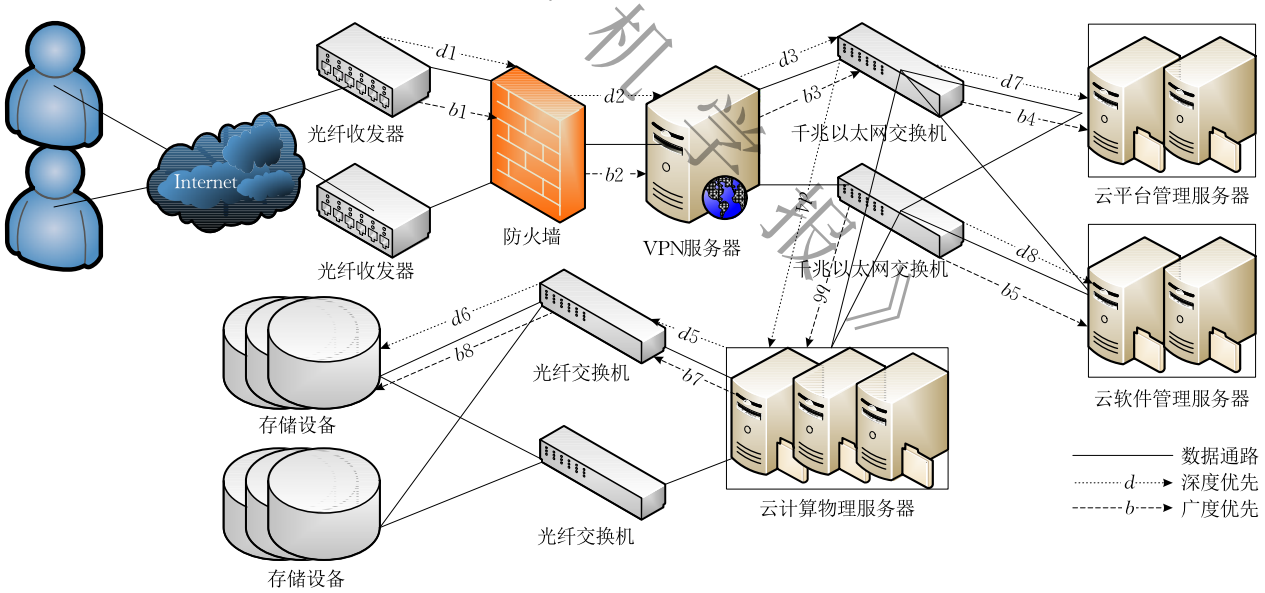

图 2 Oracle 云平台拓扑图

4.2 故障模式集合

我们从所收集的故障模式库中分别为上述 9 种基础设施组件选择若干故障模式.根据 Gunawi 等人对云平台中不同类型故障出现的频率的统计结果,我们优先选择经常出现的故障类型加入本实例研究的故障模式集合,其中包括与网络流量相关的故障、与组件自身配置相关的故障、与组件更新升级相关的故障及与负载相关的故障,如表 1 所示,表中前两列分别表示故障模式所属的组件与 ID,第 3 列表示该故障模式的名称,后 3 列分别表示该故障模式的修复方法、根因和对系统产生的影响.

表 1　故障模式

Component	ID	Name	Fix	Root cause	Impact
光纤收发器	1	光纤收发器线路断路	FIXHW	Network	FULLOUTAGE
	2	光纤收发器丢包	RESTART	Network	LOSS
	3	光纤收发器两端不能通信	RESTART	Network	FULLOUTAGE
	4	光纤收发器宕机	RESTART	Hardware	FULLOUTAGE
防火墙	5	防火墙配置故障	FIXCONFIG	Config	LOSS
	6	防火墙流量过载	RESTART	Load	PERFORMANCE
	7	防火墙软件更新故障	ROLLBACKSW	Upgrade	OPFAIL
VPN 服务器	8	服务器过载	RESTOREDATA	Load	PERFORMANCE
	9	服务器网络端口故障	RESTART	Network	LOSS
	10	服务器配置故障	FIXCONFIG	Config	OPFAIL
云平台管理服务器	11	服务器硬件故障	FIXHW	Server	FULLOUTAGE
	12	服务器配置故障	FIXCONFIG	Config	OPFAIL
	13	服务器过载	RESTOREDATA	Load	PERFORMANCE
	14	云平台管理系统故障	FIXSW	Server	STALE
云软件管理服务器	15	服务器软件更新故障	FIXSW	Upgrade	OPFAIL
	16	服务器过载	RESTOREDATA	Load	PERFORMANCE
	17	软件管理系统故障	FIXSW	Server	OPFAIL
云计算物理服务器	18	服务器硬件故障	FIXHW	Server	FULLOUTAGE
	19	服务器固件升级故障	ROLLBACKSW	Upgrade	OPFAIL
	20	服务器过载	RESTOREDATA	Load	PERFORMANCE
千兆以太网交换机	21	交换机配置故障	FIXCONFIG	Config	STALE
	22	交换机过载	RESTOREDATA	Load	PERFORMANCE
光纤交换机	23	光纤交换机端口故障	FIXHW	Network	FULLOUTAGE
	24	光纤交换机电源故障	FIXHW	Power	FULLOUTAGE
存储设备	25	存储设备硬件故障	FIXHW	Storage	FULLOUTAGE
	26	存储设备控制系统更新故障	FIXSW	Upgrade	OPFAIL
	27	存储设备读写内存块延迟	RESTART	Storage	PERFORMANCE
	28	存储设备磁道伺服信息故障	FIXSW	Storage	LOSS

4.3　研究过程

我们取组件两两之间进行组合，每次分别向相应的两个组件中各注入一个故障，即最大注入故障数 $mf=2$，分别使用深度优先（图 2 中短虚线路径）、广度优先（图 2 中长虚线路径）、全组合选择策略，得到相应的组件集合，并将组件看作参数，对应组件的故障模式看作该参数的取值，应用基于历史的故障组合方法，分别得到每个组件集合相应的 2 维待注入故障序列集合. 我们将 Oracle 云平台中 9 种基础设施两两分组，深度和广度策略中可以得到 8 组组件集合，全组合策略中可以得到 C_9^2 即 36 组组件集合，并进一步分别将每一组组件集合中的两组故障进行两两组合，得到待注入故障序列. 例如将组件光纤收发器与防火墙相组合，且光纤收发器中包含 4 个故障，防火墙中包含 3 个故障，将这些故障两两组合，共可得到 $4\times3=12$ 条待注入故障序列. 以深度优先策略为例，组件及故障的组合结果可参见附表 1 至附表 8，表中的第 1 行代表此次故障注入的目标组件，从第 2 行开始每行表示一条待注入的故障模式序列.

得到待注入故障序列集合之后，我们依据“在同一个组件中，若两个故障产生的影响相同、修复方法也相同，那么可以把这两个故障归为一类”这一策略逐条判断待注入故障序列是否能够被约减. 例如在以光纤收发器与防火墙为目标组件的故障序列集合中，由于光纤收发器中的故障“光纤收发器两端不能通信”和“光纤收发器宕机”两者对系统造成的影响相同（光纤收发器完全无法工作）、修复方法相同（重启光纤收发器），因此可以将这两个故障归为同一类，在与防火墙中的故障相组合时可以约减其中的 3 条故障序列. 以深度优先策略为例，我们在上一步得到的故障模式序列中找出所有可以被该策略约减的故障序列，共计 12 条故障序列（附表中加粗字体表示的故障序列）.

三种选择策略相应的组件选取结果、约减前故障序列数量、被约减故障序列数量及约减百分比结果见表 2，其中“—”代表该项不存在. 显然，全组合方法相比深度及广度优先方法需要更多的待注入故障序列数量，而深度及广度优先方法结果相等. 但是，三种方法各有其着重点：全组合方法对组件的组合情况覆盖全面，但极耗费测试资源，因此适用于检测云平台中小规模的、核心的组件群的可靠性；深度

表 2 实例研究结果比较

选择策略	组件集合($mf=2$)	待注入故障序列数量	约减故障序列数量	约减百分比/%
深度优先	d1={光纤收发器、防火墙}	12	3	25
	$d2$={防火墙、VPN 服务器}	9	—	—
	$d3$={VPN 服务器、千兆以太网交换机}	6	—	—
	$d4$={千兆以太网交换机、云计算物理服务器}	6	—	—
	$d5$={云计算物理服务器、光纤交换机}	6	3	50
	$d6$={光纤交换机、存储设备}	8	4	50
	$d7$={千兆以太网交换机、云平台管理服务器}	8	—	—
	$d8$={千兆以太网交换机、云软件管理服务器}	6	2	33
总计	12+9+6+6+6+8+8+6=61	61	12	19.67
广度优先	$b1$={光纤收发器、防火墙}	12	3	25
	$b2$={防火墙、VPN 服务器}	9	—	—
	$b3$={VPN 服务器、千兆以太网交换机}	6	—	—
	$b4$={千兆以太网交换机、云平台管理服务器}	8	—	—
	$b5$={千兆以太网交换机、云软件管理服务器}	6	2	33
	$b6$={千兆以太网交换机、云计算物理服务器}	6	—	—
	$b7$={云计算物理服务器、光纤交换机}	6	3	50
	$b8$={光纤交换机、存储设备}	8	4	50
总计	12+9+6+6+6+8+8+6=61	61	12	19.67
全组合	9 种组件两两组合即 $C_9^2=36$ 种组合情况 12+12+8+16+12+12+8+16+9+6+12+9+9+6+12+6+12+9+9+6+12+8+6+6+4+8+12+12+8+16+9+6+12+6+12+8=346	346	73	21.10

优先方法能够优先覆盖一条完整的数据流路径,适用于检测云平台中某一完整功能的可靠性;广度优先方法能够优先覆盖某一组件及其相邻的组件群,适用于检测云平台组件与其周边组件的组合.

4.4 研究结论

通过对这些待注入故障模式分析我们发现,在这些待注入故障序列中包含了曾经导致云平台事故的故障组合[案例 27],当我们向实验云平台中注入该集合中的故障序列时,能够保证类似案例 27 中的云平台可靠性缺陷被检测出来,从而达到利用故障注入检验云平台可靠性的目的.

5 相关工作分析与比较

在云平台可靠性研究领域中,故障的预测与发现、硬件设施的容错以及可靠性检测是其重要的组成部分.针对目标系统故障日志数据的分析、集群工作记录数据的分析工作能够为云平台故障分析、预测、发现提供数据支撑;针对云平台基础设施特征的分析工作能够指导云平台硬件容错设施的设计;故障注入测试作为验证云平台可靠性的方法之一,大量的研究工作从不同角度提出了相应的检测工具及框架.

大多数云平台故障模式分析的工作是基于系统日志或故障报告的.Jia 等人[7]基于开源云平台日志数据,分析其中由 Bug 引发的故障特征,提出了一个自动化的方法,能够从日志数据中精确地找出由 Bug 引发的故障.Garraghan 等人[8]通过分析谷歌 Cloud trace 工具的日志,分别研究了工作量及服务器中故障及修复时间的相关特征.Chen 等人[9]通过分析谷歌云集群工作记录和其中故障的特征,分别提出了工作和任务故障的统计特征,并将它们与关键调度约束、节点操作、人为因素联系起来.除了分析系统日志或故障报告外,开源社区的问题讨论区也值得人们关注.Gunawi 等人分析了 6 款云系统在开源社区中从 2011 年到 2014 年共计 21399 条问题报告,并建立了云 Bug 研究数据库(CbsDB)[10].此外,他们还收集了从 2009 年到 2015 年共计 1247 条新闻报道中的 597 条云平台中断事故,分析中断的持续时间、根因、影响、修复步骤等,并建立了云中断研究数据库(CosDB)[6].关于云平台故障分析的类型还有很多,如通过分析导致 API 故障的原因以提高云平台可靠性[11];通过对 Hadoop 中的性能相关 Bug 的分析研究,寻找出现在云平台当中的性能 Bug 的根因、检测方法及防止性能 Bug 的新方向[12];云平台的恢复机制是应对云平台故障的重要手段,但是恢复机制本身也可能出现故障,进而降低云平台的可靠性[13].

云平台基础设施的可靠性是云平台整体可靠性的基石,是影响云服务可靠性的重要因素.Khalil 等人[14]提出了一个基于 CPU 和内存使用情况,用来识别由于故障或维护使得从系统中被移除的机器的

分类方法. Vishwanath 等人[15]研究了数据中心的服务器故障及硬件修复特征，并对故障预测做了初步分析. Do 等人[16]通过研究 5 款云系统之后发现，降级的硬件设备能够对系统造成严重的影响，并基于此提出了 limplock 的概念：由于降级的硬件设备的存在导致系统运行缓慢，并且无法利用健康的设备进行失效备援. 此外，对硬件故障的分析还包括机器、硬盘、内存、网络设备故障等[17-19].

故障注入测试是验证云平台可靠性的重要手段之一，关于故障注入测试框架及工具的研究也有很多. 例如：由于线下测试有许多故障场景难以被覆盖，故而产生了故障演习：与其等待故障发生，不如演习故障发生的场景[20]. 在此基础上，Gunawi 等人[21]提出了 FaaS，一种提供在线演习的新型云服务. 在可靠性测试中，随机测试通常是一种行之有效的测试方法，Netflix 工程师基于 Chaos Engineering 提出了 Chaos Monkey[22]，模拟类似猴子随意点击的思路，是一种基于随机思想的云平台故障注入测试，除此之外，Joshi 等人提出了 SETSUDO[23]，一种基于扰乱系统思路的大规模分布式系统测试框架，通过扰乱待测系统（如向系统中注入故障）并分析系统内部状态，暴露系统中的缺陷；UC Berkeley 和 Netflix 合作提出了 LDFI[3]，一种基于谱系驱动的故障注入器，通过从正确的系统输出向后推理，以确定某些故障是否能够影响系统的正确结果；Gunawi 等人提出了 FATE and DESTINI[24]，一种用以测试云平台恢复机制的测试框架，并在此基础上提出了 PREFAIL[4]，一种可编程的故障注入工具，支持开发人员或经验丰富的测试人员自己编写剪枝策略，对多故障组合空间进行剪枝，并进行故障注入. 此外，分布式系统模型检查（DMCK）为云平台故障注入提供了新思路，然而由于多故障状态空间爆炸问题，DMCK 难以直接应用于有复杂故障场景的云平台中，Leesatapornwongsa 等人对 DMCK 进行扩展，提出 SAMC[5]，一种利用目标云平台语义信息得到状态空间约减策略的白盒的模型检查器，同时提出了 4 种约减策略：本地信息独立性（LMI）、宕机信息独立性（CMI）、宕机恢复对称性（CRS）和重启同步对称性（RSS）.

目前关于云平台故障模式分析以及云平台故障注入测试的研究工作十分丰富，然而却鲜有工作将两者结合起来，即通过分析研究故障模式来指导故障注入测试. 本文的工作通过分析历史宕机事故报告中的故障模式、提出基于历史的故障组合方法，回答了故障注入测试中注入什么故障、向什么组件注入故障两个问题.

6　结语和未来工作

检验云平台可靠性的方法有很多，传统检验云平台可靠性的方法很大程度上依赖于具体的云系统，当待测云系统发生改变时，对应的检验方法也必须随之改变. 本文试图从一个全新的角度思考该问题，通过对云平台历史事故的调研及相关故障模式特征的分析，本文证明了利用历史故障数据指导故障注入工作是合理的、可行的；本文分析并提取了故障模式的若干特征，依据这些特征，可以支持对故障模式进行选取、排序、约减；多个故障之间的交互作用在高度复杂的云平台中不可避免，本文从历史事故报告中抽象出了两类多故障组合关系；最后本文提出了一种基于历史的故障组合方法，并针对组合空间爆炸问题提出了一些约减策略，由于该故障组合方法并不依赖于具体的云系统，因此具有一定的通用性.

本文分析的故障模式大多来源与历史事故报告，多数集中于基础设施层和虚拟化层，较少涉及应用服务层的故障，因此故障的特征分析具有一定的局限性，我们希望在未来的工作中着重收集分析云平台系统、应用服务层面的故障模式，研究云系统、服务中故障的一般特征，为验证云系统服务的可靠性寻找依据. 并且，我们希望通过广泛的数据收集与分析统计，从统计学的角度证明云平台故障出现的重复性. 除此之外，我们将进一步完善基于历史的故障组合方法，并在实际云平台上验证该方法的有效性，与此同时，在基于历史的组合方法之上，我们将进一步研究通过故障注入的方法自动学习多故障组合后可能产生的新故障场景.

参　考　文　献

[1] Arlat J, Aguera M, Amat L, et al. Fault injection for dependability validation: A methodology and some applications. IEEE Transactions on Software Engineering, 1990, 16(2): 166-182

[2] Herscheid L, Richter D, Polze A. Experimental assessment of cloud software dependability using fault injection// Proceedings of the 6th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS). Costa de Caparica, Portugal, 2015: 121-128

[3] Alvaro P, Rosen J, Hellerstein J M. Lineage-driven fault injection//Proceedings of the ACM SIGMOD International Conference. Melbourne, Australia, 2015: 331-346

[4] Joshi P, Gunawi H S, Sen K. PREFAIL: A programmable tool for multiple-failure injection//Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications. Portland, USA, 2011: 171-188

[5] Leesatapornwongsa T, Hao M, Lukman J F, et al. SAMC: Semantic-aware model checking for fast discovery of deep bugs in cloud systems//Proceedings of the USENIX Conference on Operating Systems Design and Implementation. USENIX Association, Broomfield, USA, 2014: 399-414

[6] Gunawi H S, Hao M, Eliazar K J, et al. Why does the cloud stop computing?: Lessons from hundreds of service outages //Proceedings of the ACM Symposium on Cloud Computing. Santa Clara, USA, 2016: 1-16

[7] Jia T, Li Y, Tang H, et al. An approach to pinpointing bug-induced failure in logs of open cloud platforms//Proceedings of the IEEE International Conference on Cloud Computing. San Francisco, USA, 2016: 294-302

[8] Garraghan P, Townend P, Xu J. An empirical failure-analysis of a large-scale cloud computing environment//Proceedings of the IEEE 15th International Conference Symposium on High-Assurance Systems Engineering. Miami, USA, 2014: 113-120

[9] Chen X, Lu C D, Pattabiraman K. Failure analysis of jobs in compute clouds: A Google cluster case study//Proceedings of the International Symposium on Software Reliability Engineering. Naples, Italy, 2014: 341-346

[10] Gunawi H S, Martin V, Satria A D, et al. What bugs live in the cloud?//Proceedings of the ACM Symposium. Seattle, USA, 2014: 1-14

[11] Musavi P, Adams B, Khomh F. Experience report: An empirical study of API failures in OpenStack cloud environments// Proceedings of the IEEE, International Symposium on Software Reliability Engineering. Ottawa, Canada, 2016: 424-434

[12] Suminto R O, Laksono A, Satria A D, Do T, et al. Towards pre-deployment detection of performance failures in cloud distributed systems//Proceedings of the USENiX Conference on Hot Topics in Cloud Computing. Santa Clara, USA, 2015: 8-8

[13] Guo Z, Mcdirmid S, Yang M, et al. Failure recovery: when the cure is worse than the disease//Proceedings of the USENIX Conference on Hot Topics in Operating Systems. Santa Ana Pueblo, USA, 2013: 8-8

[14] Khalil M H, Sheta W M, Elmaghraby A S. Categorizing hardware failure in large scale cloud computing environment //Proceedings of the IEEE International Symposium on Signal Processing and Information Technology. Limassol, Cyprus, 2016: 327-332

[15] Vishwanath K V, Nagappan N. Characterizing cloud computing hardware reliability//Proceedings of the ACM Symposium on Cloud Computing. Indianapolis, USA, 2010: 193-204

[16] Do T, Hao M, Leesatapornwongsa T, et al. Limplock: Understanding the impact of limpware on scale-out cloud systems//Proceedings of the Symposium on Cloud Computing. Santa Clara, USA, 2013: 14

[17] Bairavasundaram L N, Goodson G R, Pasupathy S, et al. An analysis of latent sector errors in disk drives//Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS'07). San Diego, USA, 2007: 289-300

[18] Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems//Proceedings of the USENIX Conference on Operating Systems Design & Implementation. Vancouver, Canada, 2010: 61-74

[19] Gill P, Jain N, Nagappan N. Understanding network failures in data centers: measurement, analysis, and implications// Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Toronto, Canada, 2011, 41(4): 350-361

[20] Leesatapornwongsa T, Gunawi H S. The case for drill-ready cloud computing//Proceedings of the ACM Symposium. Seattle, USA, 2014: 1-8

[21] Gunawi H S, Do T, Hellerstein J M, et al. Failure as a Service (FaaS): A Cloud Service for Large-Scale, Online Failure Drills[Ph. D. dissertation]. University of California, Berkeley, USA, 2011

[22] Basiri A, Behnam N, Rooij R D, et al. Chaos Engineering. IEEE Software, 2016, 33(3): 35-41

[23] Joshi P, Ganai M, Balakrishnan G, et al. SETSUDO perturbation-based testing framework for scalable distributed systems//Proceedings of the ACM Sigops Conference on Timely Results in Operating Systems. Farmington, USA, 2013: 1-14

[24] Gunawi H S, Do T, Joshi P, et al. FATE and DESTINI: A framework for cloud recovery testing//Proceedings of the USENIX Conference on Networked Systems Design and Implementation. Boston, USA, 2011: 238-252

附 录.

案例 1

2012 年 6 月 29 日晚,亚马逊云位于美国东部的区域发生服务中断,经检查发现事故原因是由于当晚的风暴,两个区域数据中心的电源开关设备经历了电压高峰,两个数据中心的所有公共电源需要转换到发电机电源(不间断供电系统).其中一个数据中心的转换正常,而另一个数据中心的转

换出现了故障，不能提供稳定的电压，结果，发电机未能承担起负载，服务器操作发生了中断. 之后，公共电源恢复供电，数据中心供电恢复至公共电源，然而此区域中的公共电源再次发生故障，再次地，该数据中心的所有设备未能成功地转换至发电机电源.

案例 2

亚马逊云位于欧洲西部区域的一条 110kV 10MW 公共电源线路发生故障，导致一部分区域公共电源断电. 通常来说，当公共电源断电之后，不间断供电系统中控制部件会保证各个发电机同步供电，然而本次事故中，其中一个控制部件由于接地故障没能完成部分发电机的连接导致几乎所有的 EC2 实例和 58%的 EBS 卷、与外网连接的 EC2 网络装置断电.

案例 3

亚马逊云位于美国东部区域的网络了出现变化，为了升级原始网络的容量而更改了配置. 在配置更改中，有一步常规做法是转移某一个原始 EBS 网络的流量到冗余路由器，为升级做准备. 然而转移流量这一步出现错误，并未将其转移到原始网络的另一个路由器上，而是转移到了一个低容量的冗余 EBS 网络中，这个二级网络无法处理这个级别的流量，导致这部分受到影响的 EBS 节点完全与集群中的其它 EBS 节点隔离开来. 与普通的网络中断不同，这次配置更改使得原始和二级网络同时断开，使得受影响的节点被隔离.

当这个网络问题发生后，EBS 节点和集群的备份丢失. 当不正确的流量转移操作被回滚、网络问题被修复后，这些节点立刻搜索 EBS 集群以进行重镜像数据. 由于受到影响的节点很多，EBS 集群的可用空间立刻被消耗殆尽，导致很多节点被“卡死”，并连续不断地搜索集群以获得可用空间. 这种行为很快导致了“re-mirroring storm”，即当节点为它们的新副本寻找空间时，大量的卷被“卡死”.

案例 4

谷歌的一个应用引擎数据中心路由器负载增加，超出了分页阈值. 为了处理该数据中心的负载而重启了路由器，出乎意料地，重启操作加额外的负载影响了更多正常路由器，使正常路由器的数量减少，少于能够提供可靠性操作所需路由器数量的最小值. 因此导致其余正常路由器过载，并蔓延到了所有应用引擎的数据中心上，导致应用的延迟和错误率升高.

案例 5

谷歌数据中心存储层发生故障，工程师将通信重定向到另一个数据中心，导致此数据中心产生了一个短暂的负载高峰，此外，由于缓存被冲洗，需要重新将数据的查询载入内存，此项工作又引起了网络流量的高峰，导致了数据包丢失，而数据包丢失又导致数据操作变慢、URL 获取故障和延迟升高.

案例 6

亚马逊云的每个 EBS 存储服务器都有一个代理进行与数据收集服务器之间的通信并且报告维护信息等工作. 2012 年 10 月 22 日，其中一个数据收集服务器发生了硬件错误并被替换，DNS 也更新了相关替换信息，然而，该更新并未被传送到所有 DNS 服务器上，结果有一部分存储服务器没能获得新的服务器地址，依然试图连接被移除的那个数据收集服务器. 由于数据收集服务的设计具有容错性，该错误并没有立即引起问题或触发警报. 但是，这个未能连接数据收集服务器的故障在存储服务器的报告代理中引起了潜在的内存泄漏 bug. 当监控 EBS 服务器的内存消耗时，也未能对这个内存泄漏告警，直到内存泄漏率已经很高，导致受影响的存储服务器性能下降，跟不上处理普通请求的速度. 许多 EBS 服务器逐渐失去了处理用户请求的能力，被卡死的卷的数量在逐渐上升.

案例 7

谷歌的共享 HTTP 负载均衡构件中的某个组件的通信量增加，超出了它预分配的容量，此通信量增加是非恶意的，并未受到攻击，但是随后 DoS 保护措施自动启动，分流了部分通信量至 CAPTCHA. 这个出乎意料的应对导致一些客户端自动地重试，使情况恶化，最终导致一些谷歌 API 的错误率上升.

案例 8

谷歌工程师试图将谷账户系统迁移到一个新的存储后端，包括复制 API 认证服务的证书数据和重定向 API 调用到新的后端. 为了完成此次迁移，证书按计划会从旧的存储后端中删除. 由于一个软件 bug，API 认证服务仍然在旧的存储后端中查找证书（包括 Google App Engine 服务账户所使用的证书），然而这些证书已经被删除了，因此相应的服务账户未能得到认证. 随着被删除的证书越来越多，问题变得愈发严重，一些 Google App Engine 应用不断发出重试请求. 重试请求的量超出了 API 认证服务器的局部容量，导致 1.3%的 API 调用失败或超时，最终 API 认证服务超出了其全局容量，导致 12%的 API 调用失败.

案例 9

此次事故是于谷歌数据中心存储层的底层的一个故障引起的. 这个故障导致在用 Files API 打开 Blobstore 的错误增多、系统有轻微延迟. App Engine 的数据在多个数据中心中都有副本，正常情况下，在处理某个数据中心的故障前，需要将通信重定向到其它的数据中心，然而，此次 App Engine Admin Console 仍然指向了这个故障的数据中心，导致系统错误率和延迟升高.

案例 10

在谷歌负载均衡路由器升级过程中，由于一个 bug 导致部分谷歌的数据中心被错误地认为无法使用. 负载均衡的容错机制能够防止这样的故障（数据中心无法使用）导致全部的谷歌服务降级，因此它们继续路由用户通信. 这样做的结果就是部分谷歌服务正常，而与这些数据中心相关的服务停止提供服务.

案例 11

谷歌工程师对一组后端服务器进行维护，将它们正在处理的负载重定向到另一组后端服务器上. 由于内存使用计算错误，这组新的后端服务器没有足够的空间来处理这些被重定向的数据，不能处理用户请求并返回错误信息，导致某些谷歌服务重复地发送请求，最终导致后端服务器过载，用户

无法使用相关的服务.而通信过载又影响了信息路由器的功能.

案例 12

由于谷歌用户认证系统的配置错误,导致一部分的登录请求被集中到了相对而言数量较少的服务器上.当该配置错误发生后,监控系统检测到了负载上升并警告了谷歌工程师,然而,由于在目前的负载情况下认证系统工作正常,警报被清除.随后,登录通信量增加,配置错误的服务器无法处理这些负载,导致用户登录错误.而用户不断重复的登录请求又加重了请求负载.

案例 13

在一次路由器升级和服务更新的过程中,正常的步骤是将一部分的 Gmail 的网络容量关闭,并将 Gmail 的用户请求转移到另外的服务器中.然而,谷歌低估了部分需要升级的路由器上的负载,导致一些路由器过载并拒绝了用户请求,这些被拒绝的用户请求被转移到了其它路由器上,导致系统中所有路由器过载,最终造成 Gmail 无法使用.

案例 14

微软云在配置过程中,由于人为因素,一些存储节点在修复后重新投入使用时未能启动节点保护.而微软云关于这方面的监控系统存在缺陷,未能对这个故障进行报警,直至 Fabric Controller 试图将这些存储节点转换到一个新的节点,在对新节点进行配置时,Fabric Controller 需要向其中加载旧节点的状态信息,而因此触发了一个 bug:准备工作和未受保护的节点之间的 bug.准备工作的任务是让节点做好被使用的准备,这里的"使用"可能包括对节点进行格式化等操作.而节点的保护正是为了保证节点不会被 Fabric Controller 格式化.最终,有 10%的节点由于配置故障,没有进行保护,而被格式化.

案例 15

针对几种主要的存储类型,微软云存储使用 SSL 认证来保护用户数据通信.内部或外部的服务利用这些证书来解密或加密存储系统中的数据,这些证书来源于 Secret Store,并被存放在每个微软云存储节点中.在本次事故中,Secret Store 服务通知微软云存储团队这些 SSL 证书将要过期,存储团队对 Secret Store 内的证书进行了更新,并将证书包含在下一次的服务发布中,但是,存储团队没有标记这个发布是带有证书更新的发布,导致此次发布的优先级很低,没有能在证书到期之前被部署,并且,由于这些证书在 Secret Store 中是已经更新了的,因此监控系统也未能报警,这是监控系统的故障.

案例 16

Office 365 在对数据中心中的设备的核心部件进行固件升级时发生故障.导致数据中心温度迅速上升,高温导致安保措施启动,而安保措施禁止了这些服务器上的邮件服务,也禁止了任何其它基础设施的容错机制.导致 Hotmail.com,Outlook.com,和 SkyDrive 的使用受到了影响.

案例 17

"安全阀"用来在出现通信高峰时对通信进行节流.不久前,为了应对不断增长的需求,微软云增加了欧洲西部子区域的通信容量,然而,相应的设备未能正确配置.该集群快速增长的集群使用率超出了阈值,产生了大量的网络管理信息,而不断增长的网络管理信息,又触发了集群硬件设备中的 bug,导致它们达到 100%的 CPU 使用率,影响了数据通信.

案例 18

微软发现多个云服务上的 DNS 故障导致了服务降级.负责平衡网络通信的工具更新出现故障,导致配置错误,最终导致服务中断.

案例 19

为了对数据中心进行维护,Office 365 团队试图将数据中心上的一部分负载转移出去.在这过程中,有一部分网络设施发生了故障且并未发出故障警报,并且,用户登录负载也在持续增加,这三方面相组合导致了 email 服务降级.

案例 20

Salesforce 由于 NA8 的应用层一个未知的故障导致写日志数据的线程被阻塞,进而导致线程积压,而耗尽了可用内存.当内存资源被消耗后,应用层无法对用户请求做出应答.

案例 21

Skype 的一些处理实时通信和短信的服务器过载,使得相关服务对客户端请求的响应开始变慢.而 Windows 客户端请求响应变慢又产生了一个 bug,阻止客户端处理这些响应,导致客户端宕机.更重要的问题是受影响的许多客户端都是超级节点,因此导致许多超级节点从网络中消失.而突然减少的超级节点又导致剩余的超级节点过载,开始宕机.该故障迅速席卷了整个网络,最终导致 Skype 中断.

案例 22

通常 Twitter 的数据中心的设计是有冗余的,一个数据中心故障,则会有一个相同的数据中心替代,而这次故障是由于两个相同的系统巧合地同时发生故障.

案例 23

微软云的一台虚拟机初始化 bug 被错误地认为是由于主机服务器引起,一小部分的服务器被迫关闭,当错误恢复机制将被关闭的服务器上的虚拟机重新应用于健康的服务器上之后,bug 也在不知不觉中传播到了整个集群中.

案例 24

亚马逊的 EC2 在一次网络中断之后,数据存储节点试图请求存储设备,利用数据存储节点的 seeming death 的副本创建新的副本以进行故障恢复.但是,这些 still-alive 的副本占用数据,因此导致存储设备在完成新副本的创建之前就耗尽了存储资源,该故障导致了 Amazon's EC2 服务中断.

案例 25

Facebook 的数据库中无效的配置数据导致客户端需要不断地重复发送请求,大量的请求迅速淹没了数据库,使得数据库的查询操作故障.这些故障的数据库查询又导致在数据库修复之后恢复正常的配置数据无效化.这个恶性循环只有全部重启后才能被修复.

案例 26

谷歌的几个 Gmail 服务器由于维护而关闭,导致 Gmail

服务经历大范围的中断.更新之后的负载被低估导致相关的路由器过载.故障恢复机制将通信转移到其它路由器上，又导致了这些路由器过载.几分钟之内所有的路由器都过载.

案例 27

微软的 Lync Online 服务出现无法登陆的故障，当 Microsoft 修复了该问题之后，随之而来的通信高峰导致几个网络设备过载，导致一些用户无法正常使用 Lync Online 服务.

案例 28

2011 年 10 月 12 日，黑莓用户经历了历时 3 天的网络中断，该事故于周一在中东、欧中和非洲开始，并在周二蔓延到了南美和亚洲，并于周三清晨影响到了美国和加拿大.该事故最影响到了用户手机的短信和互联网服务.

附表 1

故障序列	组件名	
	光纤收发器	防火墙
01	光纤收发器线路断路	防火墙配置故障
02	光纤收发器线路断路	防火墙流量过载
03	光纤收发器线路断路	防火墙软件更新故障
04	光纤收发器丢包	防火墙配置故障
05	光纤收发器丢包	防火墙流量过载
06	光纤收发器丢包	防火墙软件更新故障
07	**光纤收发器两端不能通信**	**防火墙配置故障**
08	**光纤收发器两端不能通信**	**防火墙流量过载**
09	**光纤收发器两端不能通信**	**防火墙软件更新故障**
10	光纤收发器宕机	防火墙配置故障
11	光纤收发器宕机	防火墙流量过载
12	光纤收发器宕机	防火墙软件更新故障

附表 2

故障序列	组件名	
	防火墙	VPN 服务器
01	防火墙配置故障	服务器过载
02	防火墙配置故障	服务器网络端口故障
03	防火墙配置故障	服务器配置故障
04	防火墙流量过载	服务器过载
05	防火墙流量过载	服务器网络端口故障
06	防火墙流量过载	服务器配置故障
07	防火墙软件更新故障	服务器过载
08	防火墙软件更新故障	服务器网络端口故障
09	防火墙软件更新故障	服务器配置故障

附表 3

故障序列	组件名	
	VPN 服务器	千兆以太网交换机
01	服务器过载	交换机配置故障
02	服务器过载	交换机过载
03	服务器网络端口故障	交换机配置故障
04	服务器网络端口故障	交换机过载
05	服务器配置故障	交换机配置故障
06	服务器配置故障	交换机过载

附表 4

故障序列	组件名	
	千兆以太网交换机	云计算物理服务器
01	交换机配置故障	服务器硬件故障
02	交换机配置故障	服务器固件升级故障
03	交换机配置故障	服务器过载
04	交换机过载	服务器硬件故障
05	交换机过载	服务器固件升级故障
06	交换机过载	服务器过载

附表 5

故障序列	组件名	
	云计算物理服务器	光纤交换机
01	服务器硬件故障	光纤交换机端口故障
02	**服务器硬件故障**	**光纤交换机电源故障**
03	服务器固件升级故障	光纤交换机端口故障
04	**服务器固件升级故障**	**光纤交换机电源故障**
05	服务器过载	光纤交换机端口故障
06	**服务器过载**	**光纤交换机电源故障**

附表 6

故障序列	组件名	
	光纤交换机	存储设备
01	光纤交换机端口故障	存储设备硬件故障
02	光纤交换机端口故障	存储设备控制系统更新故障
03	光纤交换机端口故障	存储设备读写内存块延迟
04	光纤交换机端口故障	存储设备磁道伺服信息故障
05	**光纤交换机电源故障**	**存储设备硬件故障**
06	**光纤交换机电源故障**	**存储设备控制系统更新故障**
07	**光纤交换机电源故障**	**存储设备读写内存块延迟**
08	**光纤交换机电源故障**	**存储设备磁道伺服信息故障**

附表 7

故障序列	组件名	
	千兆以太网交换机	云软件管理服务器
01	交换机配置故障	服务器软件更新故障
02	交换机配置故障	服务器过载
03	**交换机配置故障**	**软件管理系统故障**
04	交换机过载	服务器软件更新故障
05	交换机过载	服务器过载
06	**交换机过载**	**软件管理系统故障**

附表 8

故障序列	组件名	
	千兆以太网交换机	云平台管理服务器
01	交换机配置故障	服务器硬件故障
02	交换机配置故障	服务器配置故障
03	交换机配置故障	服务器过载
04	交换机配置故障	云平台管理系统故障
05	交换机过载	服务器硬件故障
06	交换机过载	服务器配置故障
07	交换机过载	服务器过载
08	交换机过载	云平台管理系统故障

MA Hua, M. S. candidate. His research interest is fault injection testing.

NIE Chang-Hai, professor, Ph. D. supervisor. His research interests include software testing techniques, and software quality assurance.

WU Hua-Yao, Ph. D. His research interest is software testing.

Background

Software testing aims to reveal the latent bugs of software, which is a important stage of assuring the software quality. It is difficult for test case generation and priorization to obtain optimal solution, search based software testing provides a novel way. This project research on solving the problems of test suite generation, ordering and reduction with genetic algorithm, particle swarm algorithm, ant colomy algorithm and adaptive random methods, using combinatorial testing as case study, we achieve the solutions those cannot be done by the traditional methods like greedy algorithms and mathematical methods. Also we get the following findings: (1) when solving problems with various search techniques, the configuration parameters of the evolutionary algorithms should be tuned. (2) The presentation of solution structure can impact the effect of the methods. (3) Parallel computation can be effectively used to overcome the time cost shortage of evolutionary computation. (4) Search based software testing, just like combinatorial testing, should have adaptive mechanism to the concrete scenario. We build a special website http://gist.nju.edu.cn, where we provide a repository, an online tool and a course video. Based on the existing research, through the tight combination of theory, practice and innovation, we work on the foundation of theory, methods, empirical study and tools for the application of search based software testing.