分类号 TP311.5 学号 15060114

UDC 004 密级 公开

工学硕士学位论文

基于配置故障注入的软件反应能力测试与评估

硕士生姓名 郦旺

学科专业 软件工程

研究方向 软件可靠性

指导教师 廖湘科 研究员

国防科技大学研究生院

二〇一七年 十二月

Testing and Evaluating System Reactions on the Basis of Injected Misconfigurations

Candidate: **Wang Li**

Advisor: **Prof. Xiangke Liao**

A dissertation

Submitted in partial fulfillment of the requirements

for the degree of **Master of Engineering**

in **Software Engineering**

Graduate School of National University of Defense Technology

Changsha, Hunan, P. R. China

December 10, 2017

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：基于配置故障注入的软件反应能力测试与评估

学位论文作者签名：[signature] 日期：2017年11月13日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于配置故障注入的软件反应能力测试与评估

学位论文作者签名：[signature] 日期：2017年11月13日

作者指导教师签名：[signature] 日期：2017年11月13日

目　录

摘　要 i

ABSTRACT ii

第一章　绪论 1

1.1　课题研究背景 1

1.1.1　课题来源 1

1.1.2　研究背景 1

1.1.3　研究现状概述 2

1.2　课题研究内容和创新点 4

1.3　论文结构 5

第二章　相关研究 6

2.1　配置故障相关技术研究 6

2.1.1　软件开发相关技术研究 6

2.1.2　软件管理相关技术研究 8

2.2　配置故障发展趋势研究 10

2.2.1　软件系统规模不断增大对配置故障的影响 10

2.2.2　软件系统不断普及对配置故障的影响 10

第三章　ConfVD 系统框架 12

3.1　ConfVD 总体设计 12

3.2　关键技术 13

3.2.1　基于配置项分类的约束提取 13

3.2.2　配置故障自动生成与注入 14

3.2.3　软件反应能力分析 14

第四章　基于配置项分类的约束提取 16

4.1　软件配置分类 16

4.1.1　开源软件配置项调研分析 17

4.1.2　配置分类树 17

4.2　配置约束提取 18

4.2.1　基于值类型的配置约束 19

4.2.2　配置文件格式约束 22

第五章 配置故障自动生成与注入 24
5.1 配置故障生成 24
5.1.1 基于值类型的配置故障 24
5.1.2 配置文件格式故障 26
5.2 配置故障注入 27
5.2.1 配置抽象化表示 27
5.2.2 配置故障生成模板 28
第六章 软件反应能力分析 29
6.1 配置故障测试 29
6.1.1 测试流程 29
6.1.2 测试用例 30
6.1.3 测试结果记录 30
6.2 软件反应能力分类模型 31
6.3 反应能力自动分析 32
6.3.1 异常日志提取 33
6.3.2 日志诊断内容充分性分析 33
第七章 实验与评估 37
7.1 ConfVD 技术评估 37
7.1.1 配置分类覆盖率评估 37
7.1.2 配置约束一致性评估 38
7.1.3 诊断日志充分性评估 39
7.2 软件反应能力评估分析实验结果 39
7.2.1 软件反应能力分析 39
7.2.2 配置故障诊断能力分析 41
7.2.3 ConfVD 能力分析 42
7.3 结果有效性分析 43
7.4 实践经验总结 45
第八章 结束语 46
8.1 工作总结 46
8.2 研究展望 47
致谢 48
参考文献 50
作者在学期间取得的学术成果 54

表 目 录

表 2.1 配置故障相关技术研究分类 6
表 4.1 各类型配置项数量表 18
表 4.2 配置项值的语法约束 21
表 4.3 配置项值的语义约束 22
表 4.4 配置项值相关的环境属性 22
表 5.1 语法配置故障生成实例 25
表 5.2 格式配置故障生成实例 27
表 6.1 软件系统反应分类表 32
表 7.1 ConfVD 运行测试软硬件环境 37
表 7.2 实验评估软件信息 37
表 7.3 分类树对现实配置项的覆盖率 38
表 7.4 语法约束一致性比例 38
表 7.5 诊断日志充分性评估 39
表 7.6 配置故障注入信息表 40
表 7.7 软件反应能力分类结果 41
表 7.8 ConfVD 与 ConfErr 的比较实验结果 43
表 7.9 ConfVD 与变体的比较实验结果 44

图 目 录

图 1.1 Linux 代码量历年增长情况 2
图 1.2 配置故障诊断困难 3
图 3.1 ConfVD 总体结构图 12
图 4.1 软件配置类型约束实例 16
图 4.2 配置项分类树 19
图 4.3 各类型约束作用范围 20
图 4.4 Yum 配置文件内容 23
图 5.1 语义配置故障生成实例 26
图 5.2 ConfVD 配置故障注入过程 27
图 5.3 配置格式化表示 28
图 6.1 配置故障测试流程 29
图 6.2 测试结果记录 31
图 6.3 软件不良反应实例 32
图 6.4 异常日志提取实例 34
图 7.1 软件对不同类型配置故障的反应情况 42
图 7.2 不同约束产生的配置故障对软件反应的影响 44

摘　要

随着软件规模的不断扩大，软件的配置也变得更加复杂。配置故障已经成为导致软件失效的主要原因之一。然而目前研究工作主要集中在配置故障的自动诊断上，对于软件应对配置故障反应能力的研究仍很欠缺。本课题针对目前研究的不足，对大型开源软件配置进行了大规模调研分析，提出覆盖面全的软件配置项类型分类树，设计实现了基于配置故障注入的软件反应能力测试和评估框架ConfVD。ConfVD 通过推断软件配置约束生成配置故障，并注入配置故障到软件系统当中进行测试，根据软件对配置故障的测试结果分析和评估软件系统的反应能力。全文的主要工作分为以下几个方面：

1、系统分析了目前广泛应用的八款开源软件 Squid、Nginx、Redis、Nagios、Lighttpd (core)、Puppet、SeaFile 和 Vsftpd，总结出了覆盖率达到 96.5% 软件系统配置项分类。基于该配置项分类树，本课题提出了值类型软件配置约束与配置文件格式约束，经验证，本课题提出的配置约束条件符合评估使用的软件配置总数的 91%。

2、基于本文提出的配置约束，ConfVD 实现了配置故障的自动生成技术。通过使用 ABNF 范式表示配置约束，ConfVD 能够根据细粒度的约束产生全面的配置故障，从而得到更丰富的注入结果。同时对于已经生成的配置故障，通过对软件配置的抽象表示，实现了不同软件系统的自动化注入。

3、ConfVD 实现了对配置故障的自动测试框架，并根据诊断配置故障的能力将软件反应分成了六大类，同时基于自然语言处理的相关技术实现了软件反应的自动分类。通过分析软件反应能力的分布结果，本课题发现 15% 的软件配置故障引起了软件的不良反应导致配置故障的诊断困难，而其中路径类型的软件配置故障是最难被软件系统诊断的。根据实验结果，本课题还总结了软件配置设计实现过程的实践经验。

对比前人工作，由于拥有约束作为指导，ConfVD 能够发现三倍于 ConfErr 的软件配置脆弱性。同时对于前人工作中的简单约束而言，ConfVD 使用细粒度约束生成的配置故障能够比前人多发现 18.6% 的软件配置故障引起的不良反应，使得充分暴露软件应对配置故障时的脆弱性成为可能。

关键词: **软件系统**; **配置故障**; **反应能力**;

ABSTRACT

As the software systems become more and more complex, so does their configurations. Nowadays, misconfiguration has become one of the major causes of failures. But most of state-of-art researches concentrate on automatically diagnosing the misconfigurations, and there is seldom work on system reactions to misconfigurations. Targeted at the drawbacks of these researches, we conduct a empirical study on configurations from widely-used open source software systems, and we proposed a more comprehensive type classification for configuration options. Based on this classification, we design and implement our tool ConfVD, to infer the configuration constraints and generate misconfigurations. ConfVD injects these misconfigurations into software systems and analyze the system abilities according to their reactions to misconfigurations. We classify our work into the following aspects:

1. We conduct an empirical study on 8 widely-used open source software systems, including Squid, Nginx, Redis, Nagios, Lighttpd(core), Puppet, SeaFile and Vsftpd. We propose a configuration type classification which can well represent 96.5% of configuration options in realistic. Based on the classification, ConfVD can infer syntactic, semantic and format-related constraints for configuration. After verification, our inferred constraints are consistent with 91% of options in our study.

2. On the basis of the constraints ConfVD infer，ConfVD can generate syntactic, semantic and format-related misconfigurations. Then, ConfVD uses the abstract representation for configurations to automatically inject misconfigurations into different software systems. ConfVD uses ABNF to describe the constraints within configuration options, so that fine-grained constraints can guide ConfVD to generate more comprehensive misconfigurations and get more various injection reaction results.

3. We design and implement a testing framework in ConfVD aiming at testing the system reactions to misconfigurations. According to the results from the test, we classify system reactions into 6 main types to leverage the ability in diagnosing misconfigurations. Based on natural language processing techniques, classification can be done automatically by ConfTest. After analyzing the results, we found that 15% of system reactions give rise to the difficulties in diagnosing misconfigurations. Among these misconfigurations, path-related ones are most difficult to deal with. Our experiment data reveals that the

configuration correctness check in software startup could effectively reduce the risk of bad system reactions.

Through the comparison with the formers, ConfVD can find 3 times bad reactions more than ConfErr, because of the benefit from constraints we found in configurations. For coarse-grained constraints used in SPEX, our misconfigurations guided by more fine-grained constraints can find 18.6% more bad reactions, which makes comprehensively evaluating system reaction ability possible.

Key Words: Software System; Misconfiguration; Reaction Ability;

第一章　绪论

随着信息化水平的不断提高，大规模基础软件不断发展和完善，正广泛应用于政府，金融，航天，军事等关键应用中，渗透到国民经济和国防建设的各个领域，在现代化社会中扮演着不可或缺的角色。但同时由于软件系统具有规模庞大，自身逻辑复杂、可靠性要求逐渐增强、出错后难以分析和调试等特点，软件的复杂性已经成为了影响软件广泛应用的重要因素。而其中软件配置故障已成为导致软件系统失效的最主要原因。因此针对软件系统的配置故障的研究也愈加迫切。本章首先在 1.1 节陈述了课题的来源及研究背景，并且概述了与课题相关的研究现状及存在的问题，在 1.2 节总结了本文的主要研究内容和工作创新点，最后在 1.3 节简要介绍了本文的组织结构。

1.1　课题研究背景

1.1.1　课题来源

本课题为自选课题。

1.1.2　研究背景

为了满足用户以及目标平台的不同需求，可配置性已经成为目前软件的主流特征之一。可配置软件正在被广泛应用在了政府，金融，航天，军事等关键应用之中，在人们的生活中发挥着重要作用。随着计算机技术的迅猛发展，计算能力的不断提高，人们对于软件系统的需求愈加复杂。因此软件系统规模也随之不断扩大，其中最直观的表现就是代码量的爆炸式增长，目前常用的开源软件，例如 MySQL 等，其规模均达到了百万行的量级。根据 Coverity 公司的年度开源软件代码分析报告 [1] 所述，目前 Linux 内核源码在过去八年增长了接近 4 倍，如图 1.1所示，截止 2017 年已经达到了 4 千五百万行。

随着软件规模的不断扩大，软件的配置也变得更加复杂。例如目前流行的开源软件 Httpd，PostgreSQL 均有超过 200 个配置项。同时，软件配置还必须满足特定的约束条件，这些都使得配置故障的发生愈发频繁。很多研究 [2–5] 指出配置故障是现今软件可靠性下降的重大威胁之一。同时一份报告 [6] 指出，配置故障占到了某商业存储系统的失效原因的 30%。大量的商业系统，例如微软的 Azure [7]，亚马逊的网页服务 [8]，以及 Facebook [9] 在近年来都曾遭受过配置故障引起的软件失效，从而带来了大量的经济损失 [10]。

然而配置故障诊断仍然是学术界及工业界的难点问题。其难点主要反映为以

下三点：1. 配置故障的原因十分复杂。软件系统用户的失误通常是导致配置故障的最直接原因，但是不合理的软件实现也会导致用户的误解从而引发软件失效，如图 1.2(a)所示，MySQL 回滚了错误设置却并没有通知用户。这种不合理的软件实现容易误导用户，从而产生配置故障引起软件失效。2. 配置故障在触发之前会潜伏在软件系统当中，因此难以察觉。如图 1.2(b)，在 Yum 当中，当配置项“cachedir”被错误设置之后，软件仍能够正常运行。只有当 Yum 需要访问本地缓存资源时，软件才会发生失效，因此用户难以察觉软件系统当中已经存在的配置故障。3. 软件失效之后反馈信息的缺乏也阻碍了配置故障的诊断。如图 1.2(c)所示，当遇到配置故障时 MySQL 仅仅提示用户“can't start server”，并没有指明当前发生的配置故障。由于缺乏特定软件知识以及大量的时间精力，用户并不能像软件开发人员一样进行 debug。但是如果软件系统能够提供足够充分的提示信息，用户完全可以自己解决配置故障，从而节省大量用于诊断和修复配置故障的开销。

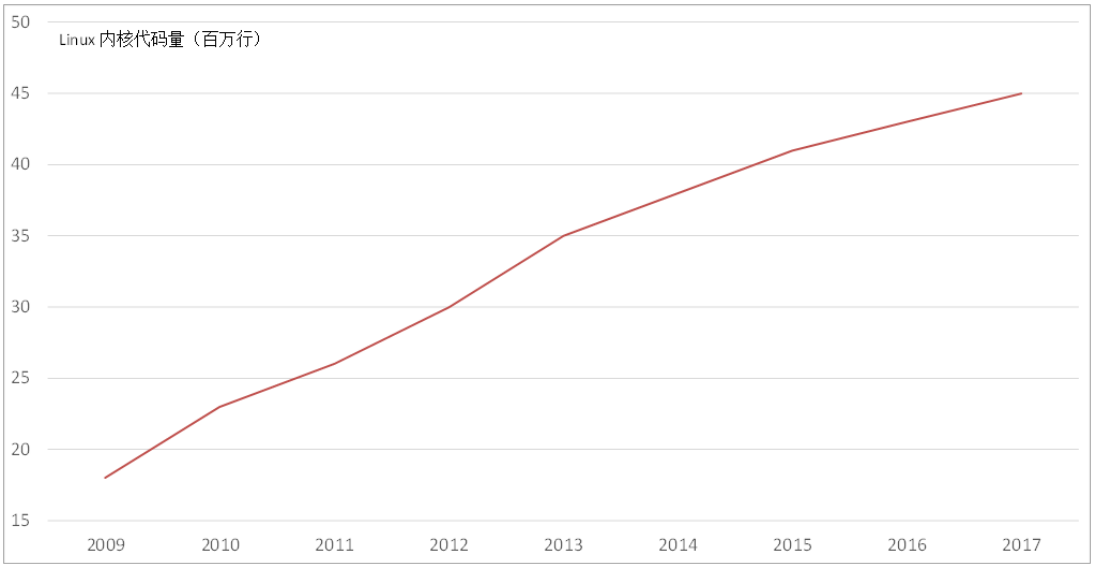

图 1.1　Linux 代码量历年增长情况

1.1.3　研究现状概述

目前针对配置故障的研究按照技术手段主要分为程序分析，统计学习，对比技术，重放技术以及配置故障测试技术五个大类。

基于程序分析的工作分为静态与动态分析，相关工作主要包括 ConfAnalyzer [11]，ConfAid [12]，X-ray [13] 等等。程序分析方法需要软件源码或者二进制码，这些技术将软件系统视为白盒，通过分析软件系统的数据流和控制流信息，确定软件系统中受到配置项影响的程序语句及执行路径，从而推断出配置故障产生的

配置故障 1：symbolic_lin=0 /* MySQL */
原因：配置项“symbolic_link”中存在错误拼写，误将字母“k”遗漏。
配置故障 2：symbolic_link=yes
原因：配置项“symbolic_link”只能识别“0”、“1”，无法识别“yes”，从而将设置回滚成默认配置。

(a) 配置故障原因复杂

/* Yum */
配置故障：
cachedir=/nonexistent/directory
软件反应：
Yum能够正常运行，但是一旦需要访问本地缓存就会发生失效。

(b) 配置故障潜伏性强

配置故障： /* MySQL */
socket=/nonexistent/filename
软件反应：
MySQL启动失败并输出日志语句“Can’t start server : Bind on unix socket: Address already in use”，缺乏帮助故障诊断的信息。

(c) 缺乏软件反馈影响诊断

图 1.2 配置故障诊断困难

原因。基于程序分析的技术精确度较高，然而由于需要软件源码或字节码作为数据输入，同时需要大量的计算开销，一定程度限制了该方法的应用场景。

基于统计学习的配置故障诊断技术通过软件系统的历史信息数据，通过概率统计或机器学习挖掘出配置使用的规则，再通过查询规则库来诊断已经发生的配置故障。目前基于统计学习的工作主要包括 CODE [14]、EnCore [15]、Snitch [16] 等。由于基于统计学习的配置故障诊断技术需要大量数据集作为学习样本，而软件配置故障数据集的缺乏一直是该方法的短板，因此在实际使用当中仍存在局限性。

基于对比技术的配置故障诊断的核心思想是通过对比目标数据与样本数据的信息特征，包括系统状态，事件信息等等，实现配置故障的诊断。目前已有的工作包括 STRIDER [17]、PeerPresure [18] 和 SigConf [19]。对比技术的局限性在于人工构建样本集的过程依赖专家知识作为支撑，因此很难做到自动化地配置故障诊断过程。

基于重放技术的配置故障诊断通过模拟软件系统运行的沙箱环境，并不断尝试和回滚操作，从而定位配置故障并进行修复。目前主要工作包括 Chronus [20] 和 Triage [21]。由于需要搭建模拟系统运行的沙箱环境，并且进行重放计算，沙箱环

境内运行时的额外计算开销是该技术的主要瓶颈。

除了上述各种技术对配置故障直接进行预防、检测、诊断的工作，另一部分工作从评估软件应对配置故障反应角度出发，测试软件对于配置故障的反应能力，从而帮助开发人员不断完善软件代码，即配置故障测试技术。目前已有工作包括 ConfErr [22]、SPEX [23] 和 PCHECK [24] 等。通过注入配置故障观察软件的反应情况，开发人员能够通过修改软件系统提升对配置故障的反应能力，从而达到提升软件可靠性的目的。

1.2 课题研究内容和创新点

良好的软件反应，意味着软件在发生配置故障之后能够进行规范有效的错误处理避免软件发生崩溃、挂起等不良反应，同时能够指出用户引入的配置故障，从而帮助用户更快地定位和解决配置故障，减少配置故障诊断过程用户的负担。通过评估分析软件对于配置故障的反应能力，开发人员能够更有效地改进他们的代码，从而实现软件对于配置故障的良好反应。然而目前能够系统化地评估软件系统对于配置故障的反应能力的工作仍然十分欠缺。

针对配置故障软件系统难以诊断，软件反应能力普遍较弱的现状，本课题通过对大量实际开源软件配置进行调研，设计并实现了基于配置故障注入的软件反应能力评估分析框架 ConfVD。本文的主要贡献包括以下几个方面：

- 调研分析了八款开源软件的 1593 个配置项，包括 Squid，Nginx，Redis，Nagios，Lighttpd (core)，Puppet，SeaFile，Vsftpd。通过对上述软件的配置项进行归纳总结，本课题将软件系统配置当中配置项分成了 21 小类，该分类经过实际软件系统配置的评估覆盖率达到了 96%。本课题同时根据其类型提取出了相对应的细粒度约束，作为区分配置正确与否的详细规则，包括配置项值的类型约束以及特定配置格式的约束等等。经过验证，本文提出的约束与现实约束的一致程度达到了 91%。

- 本课题对于前文提取到的配置约束，提出了配置故障自动生成方法。利用 ABNF 范式对软件配置约束进行细粒度的描述，ConfVD 能够产生更加丰富的配置故障，从而全面地测试和评估软件系统的反应能力。同时 ConfVD 使用了配置的抽象化表示，从而能够将配置故障注入到完全不同的软件系统当中。

- 本课题实现了对配置故障的自动测试框架，并根据诊断配置故障的能力将软件反应分成了六大类，同时基于自然语言处理技术判断软件反馈给用户的异常诊断信息是否充分，实现了软件反应的自动分类。通过分析软件反应能力

的分布结果，本课题发现 15% 的软件配置故障引起了软件的不良反应从而导致配置故障的诊断困难，而其中路径类型的软件配置故障是所有配置故障类型当中最难被软件系统诊断的。根据实验结果，本课题还总结了软件配置设计实现过程的实践经验。

1.3 论文结构

本文共分 8 章，第 1 章为绪论，主要介绍课题的研究背景，指出了该研究的实际意义。然后对配置故障诊断方面的相关工作进行了简要介绍，总结了本课题当中研究的主要内容以及创新点，最后介绍了本文的结构构成。

第 2 章是本课题研究的相关工作，本文调研了有关软件配置故障方面的大量现有研究工作，并将其分成了配置故障相关技术和配置故障发展趋势两大类别进行分别介绍。

第 3 章介绍了 ConfVD 的系统框架，主要包括 ConfVD 的总体设计以及包含的关键技术。其中关键技术分别为基于配置项分类的约束提取，配置故障自动生成与注入以及软件反应能力分析三个方面。

第 4 章主要描述了基于配置项分类的约束提取方法。首先本课题调研了大量的软件配置项，根据调研结果提出了一种覆盖面更广的配置项分类树，并基于此分类提出了细粒度的配置约束的提取方法。

第 5 章介绍了配置故障自动生成与注入的相关技术。基于第 4 章本文提出的约束，ConfVD 能够生成配置故障并自动化地注入到不同的软件系统当中。

第 6 章回答了如何分析软件反应能力的问题。首先 ConfVD 能够通过测试记录下软件的相关反应，并提出了软件反应能力的分类，通过自然语言处理技术，实现了对软件反应进行自动化分析归类的功能。

第 7 章是实验与评估，主要包含两大部分：ConfVD 使用技术的评估，以及使用 ConfVD 对软件反应能力测试与评估。

第 8 章是对全文的总结，并展望了下一步的研究方向。

第二章　相关研究

由于近些年软件配置故障对于人们生产生活造成的持续威胁，针对软件配置故障的相关研究工作目前已经成为学术界研究热点之一。为了应对配置故障带来的挑战，消除配置故障带来的不利影响，针对配置故障相关技术研究贯穿软件开发和管理过程，如表 2.1所示。同时，随着软件系统的不断发展，软件系统的配置故障也存在一定的发展趋势，这也引起了国内外研究人员的注意。本文将从配置故障相关技术和配置故障发展趋势两大方面论述近年来学术界对于软件配置故障的研究工作。

2.1　配置故障相关技术研究

2.1.1　软件开发相关技术研究

软件开发指软件系统的构建和维护过程。一个软件开发周期包括软件的设计，实现，文档，测试等等多个部分。配置作为软件系统的一部分，必然遵循这些标准的开发过程。如前文所述，配置故障每年造成 IT 行业的大量损失，因此在软件开发过程中减少或者消除配置故障造成的不良影响是大有益处的。这些技术主要分为以下三个方面。

2.1.1.1　软件系统自动配置

许多研究把注意力集中在了如何将用户的配置过程自动化上，因为这样可以帮助用户减少配置软件的负担，并且防止了用户在配置过程时引入配置故障。但是显而易见的，该技术存在许多难点。首先，并不是所有配置都能够被自动完成的，举例而言，有些配置需要软件系统甚至操作系统以外的信息，比如远程主机的 IP 地址。其次，软件自动配置很难兼顾可用性与复杂性的权衡。一方面人们都希望软件能够被简单地配置和使用，但是实际上许多软件功能依赖繁多的软件选项来进行设置。

表 2.1　配置故障相关技术研究分类

	应用场景	相关技术分类
软件开发相关研究	软件设计和实现	软件系统自动配置
		让软件系统易于配置
	质量保证	提升应对配置故障的代码质量
软件管理相关研究	设置和验证	配置正确性检查
	部署和监控	配置自动部署及程序监控
	解决故障	配置故障的诊断

尽管有相当多的挑战，该领域仍催生了许多研究在消除配置选项，自动选择推荐配置项值等工作中。Xu [25] 等人在 2015 年的一篇报告中指出，31.1%-54.1% 的配置选项很少被用户使用到，因此将这部分配置选项隐藏或者移除对现有用户的影响非常小。同时，目前已经有软件提供商开始收集用户的配置信息来指导设计用户真正需要的配置 [26]。Apache Flink 的存储管理系统 [27] 在设计过程中消除了一些可能导致故障的配置。

一部分研究尝试对软件性能与配置的关系的进行建模，提出了一些黑盒的优化算法来近似计算配置对软件性能的影响，从而帮助指导软件系统配置过程，例如 Ye 等人 [?] 提出的随机算法，Xi 等人 [28] 提出的爬山法，以及 Osogami 和 Itoko 等人提出的邻居选择算法 [29]。其他一部分研究尝试描述软件性能的特征与配置项的关系。然而这部分工作由于负载和环境的局限性，在实际应用中仍不是很常见。

2.1.1.2 让软件系统易于配置

让系统易于配置的技术相比较自动配置更为实际，因此在现实生活中更加常见。该技术的核心思想即将软件配置视为用户接口来采用用户友好的思想来进行设计和实现软件配置。Velasquez 等人 [30] 与 Barrett [31] 等人对用户在软件配置过程中遭遇的困难进行了调查和研究分析。DeTreville [32] 提出了一种层级的系统模型来描述软件系统之间的相互依赖关系，并被配置管理工具 Puppet，Chef 等采用来方便用户的配置软件的过程。另外一些工作 [33, 33, 34] 提出了配置在设计过程当中应当遵循的准则，包括直觉性、一致性、提供反馈、最小化原则以及提供帮助和文档等等。

然而目前成熟的大型软件项目并不能满足上述这些基本的设计原则。Xu 等人 [23] 在文章中指出，目前的成熟软件在这些设计原则方面都有着不同程度的欠缺。同时，他们指出软件配置的实现过程同样需要遵循类似的规则。例如不安全的 API 调用应当避免在软件配置实现过程当中使用，例如“atoi”，“sscanf”，“printf”等等，这些 API 并没有提供错误的检查机制，因此会将配置故障传递到软件系统的运行过程中。

2.1.1.3 提升应对配置故障的代码质量

另一些研究者认为，为了应对配置故障，软件应该提升自身的代码质量来避免发生不良反应。举例来说，当用户配置系统出现故障的时候，系统不应该仅仅发生失效、崩溃等现象，而应该采取一些措施例如输出出错配置的信息，或者阻止配置故障产生的不良影响使得软件能够继续正常运行。

ConfErr [22] 提出了一种测试方法来检测软件系统对配置故障的反应能力。首先 ConfErr 从人类模型出发，将人类错误分为基于技巧、基于规则和基于知识三个层面，并映射到软件配置过程，从而模拟软件可能会面临到的配置故障，包括

拼写错误（例如遗漏、冗余、大小写等等），结构错误（例如没有满足配置文件当中特定的结构要求）。ConfErr 测试了五款广泛使用的开源软件，并暴露出了其中的低质量实现的代码部分。由于作为一种黑盒测试技术，ConfErr 并不能够理解配置包含的约束信息，因此 ConfErr 无法测试约束相关的软件配置故障。

SPEX [23] 在 ConfErr 的基础上引入了约束的概念。约束定义了软件在配置过程中应当遵循的规则。通过违反这些约束信息，SPEX 能够生成配置故障（例如一个不存在的路径，而约束要求该路径是必须存在的）并运用到测试当中。SPEX 通过静态分析程序代码记录配置选项的读入和使用过程，并在此过程中记录 SPEX 预先定义好的各类约束信息。例如 SPEX 发现配置项“file_par”被函数 open() 调用，因此推断“file_par”属于文件类型。再通过违反该约束，例如生成一个目录路径赋值给“file_par”来产生一个配置故障并加以测试。SPEX 作为第一个提出约束概念的工作，其约束种类过少，分类过于宽泛，距离生成全面的软件配置故障仍存在一定的挑战。

PCHECK [24] 提出潜在配置故障（Latent Configuration Error，简称 LC error）的概念，即配置故障进入到软件系统中并不是一定会立即发生的，通常会潜伏直到触发系统才会有反应。针对这个挑战，PCHECK 尝试通过程序分析的方法，将软件配置相关的语句封装起来，并在软件运行初期就加以触发，并观察软件的反应，从而避免了潜在配置故障可能造成的不良影响。

2.1.2 软件管理相关技术研究

相对于软件开发相关而言，软件管理相关技术研究侧重于软件系统配置使用的过程，包括设置并检查配置值，监测软件运行时配置的变化以及诊断已经发生的由配置故障引起的软件失效。这些工作通常需要大量的专业知识以及人工开销，研究者们试图通过一些工具来减轻用户的负担，甚至完全替代这部分工作。这部分工作主要分为以下三个方面。

2.1.2.1 配置正确性检查

配置正确性检查即指保证软件系统配置正确的技术方法，其中主流技术如配置故障自动检测通常指在软件系统运行前检测当前配置当中是否存在故障，其技术核心在于提取出软件系统配置所满足的正确规则，并通过查看软件配置是否与该规则匹配来确定其是否存在配置故障。

CODE [14] 针对 Windows 下的注册表系统发生的事件进行机器学习，从而推断出正确的事件序列。基于学习出的模式，CODE 能够检测出配置事件当中可能隐藏的错误。Kiciman 和 Wang [35] 提出了一种无监督聚类的算法来学习 Windows 当中注册表的特定结构与语义信息。Bauer 等人 [36] 通过关联规则挖掘的方法，识别

出了和访问控制有关的配置故障。为了进一步提升这些规则的精确性，EnCore [15] 通过预先定义规则模板，通过机器学习大量软件配置文件从而推断出配置当中包含的约束信息，并根据推测出的规则对软件配置进行故障检测。

2.1.2.2 配置自动部署和监控

随之分布式软件例如云技术、数据中心等等的不断发展，软件系统架构出现了单节点到多节点的变化趋势，他们通常包含数百个甚至上千个计算节点。因此人工部署和管理这些节点上的配置变得越来越不可行，亟待一些自动化的技术来减少人工开销以及降低人工引入的配置故障的可能。

在过去二十年里，许多配置自动部署工具被投入使用，包括 LCFG [37]、CFEngine [38]、NewFig [39]、SmartFrog [40] 等等。这些工具能够帮助用户在多个节点上快速部署软件的配置。另外一部分工作主要是监控软件系统的配置变化，因为软件研究人员认为一些关键的配置变化应该得到用户的确认，从而避免软件失效的发生。LiveOps [41] 是微软公司开发的一款系统管理服务软件，当软件系统的配置发生改变的时候，它能够提示用户这些未知修改的发生。

2.1.2.3 配置故障诊断

当软件遇到配置故障并发生了配置失效时，用户就需要根据失效时候软件系统的特征来诊断配置故障。一般而言用户只能通过人工的方式进行配置故障的诊断，比如检查配置文件，分析日志信息，查看系统状态，查看领域知识的文档等等。由于正如前文讨论的，配置故障的诊断费用开销极大，因此业界急需自动化工具来实现配置故障的自动诊断工作，然而仍存在两大挑战。首先，由于用户可能无法接触到源码，或者缺乏源码相关的专家知识，因此配置故障的诊断并不能像软件 debug 一样顺利。其次，配置故障诊断技术如何将代码当中发生的错误映射到软件配置当中也是一大技术难点。

通过程序分析的方式，相关技术能够实现对配置故障的诊断。ConfAid [12] 通过程序插桩的方式修改软件的二进制码，从而能够实现软件行为路径的监控。当发生配置故障的时候 ConfAid 能够复现故障发生时的软件执行路径，通过分析软件配置与这些路径的依赖关系就能够得到最有可能的故障原因。X-ray [13] 利用了和 ConfAid 类似的思路，通过插桩记录软件系统性能的摘要，能够量化软件配置对系统性能的影响程度，从而诊断出由配置故障引起的软件性能问题。ConfAnalyzer [11] 使用静态分析的方法，通过分析源码当中配置参数与配置故障失效的位置的关联关系，一旦产生软件失效，就能通过之前的分析以及软件日志构成的序列来定位到出错配置上。

另外有部分工作通过基于特征的对比技术实现了配置故障的诊断工作。对比技术的核心思想是通过对比目标数据与样本数据的信息特征，包括系统状

态，事件信息等等，实现配置故障的诊断。目前已有的工作包括 STRIDER [17]、PeerPresure [18] 和 SigConf [19]。对比技术的局限性在于人工构建样本集的过程依赖专家知识作为支撑。

基于检查点的重放技术来诊断软件配置故障，即通过模拟软件系统运行的沙箱环境，并不断尝试和回滚操作，从而定位配置故障并进行修复。目前主要工作包括 Chronus [20] 和 Triage [21]。由于需要搭建模拟系统运行的沙箱环境，并且进行重放计算，运行时的计算开销是该技术的主要瓶颈。

2.2 配置故障发展趋势研究

配置故障并不是近些年出现的新兴问题，自软件配置诞生之日起配置故障就一直伴随其左右。但是由于近些年配置故障愈发的普遍，研究配置故障的发展趋势也变得尤为重要。

2.2.1 软件系统规模不断增大对配置故障的影响

目前流行的软件系统已经呈现复杂度和代码量上的急剧增长。大型系统在十年前指代数十台计算机构成的集群系统，而在今天大型系统，例如云平台，数据中心等等，通常都运行在数以万计的计算节点上。这些计算节点有着截然不同的属性，例如软件组件，硬件平台等等。在这些大型软件系统当中，配置故障也会产生等比例的放大，从而造成更加严重的影响。首先，系统关键节点，或者关键路径上的配置故障可能影响整个软件系统。例如 Google 公司在 2012 年，由于错误配置了负载均衡服务器，导致同步服务器也受到牵连，发生了宕机 [42]。其次，配置故障也可能通过的实例的复制从而进行传播而加剧产生的不良影响。另一方面，目前很多软件系统组件之间存在复杂的依赖。例如 GoogleDocs 的服务就依赖于 50 余项 Google 其他的服务 [43]。由于目前软件系统高度复杂化，组件之间的相互关联也愈发常见，因此某个组件中的配置故障就可能通过传播影响其他组件，从而产生灾难性的后果。

不断扩大的软件规模也增加了软件配置过程的难度。根据目前软件实际的应用场景而言，大型软件系统通常由各个小组分别管理系统的各个组件，然而在配置过程中，这种管理模式必然导致软件管理人员对于系统全局知识的缺乏，因此不难推测这会增加配置出错的可能。而目前的技术主要关注于单一组件的自动配置以及配置管理，如何进行跨组件的配置仍存在很大的问题。

2.2.2 软件系统不断普及对配置故障的影响

除了软件系统规模的不断增大，随着云平台等技术的普及 [44]，普通用户不再需要像以往那样租借或购买软件服务器，而只需要通过购买例如 VPS 等廉价的云

服务就能达到目的。这些技术的普及一方面使得用户能够直接使用云服务提供商提供的软件系统服务，而另一方面中小用户也更容易直接管理和使用到部署在云平台的软件系统。而现在，小型公司甚至普通用户能够更加频繁地管理和使用软件系统[45]，包括服务器，设施系统等等。这些用户平时需要维护和管理自己的小型系统，然而由于缺乏专业知识，普通用户也更容易遭受软件配置故障产生的风险。更加不幸的是，这些用户相比较专业人员花费更多的时间和精力在解决配置故障上[6]，例如和专业人员的邮件沟通等等。软件系统对于普通用户的不断普及注定会带来配置故障的日益普遍，从而影响人们的生产生活。

第三章　ConfVD 系统框架

在充分调研现有工作的基础上，针对软件反应能力分析的相关工作不足的现状，本课题提出和实现了 ConfVD，一个基于配置故障注入的软件反应能力评估分析框架。ConfVD 首先推断出软件配置约束信息，基于约束生成配置故障并注入到软件系统当中。通过测试记录下软件的反应情况，最后对软件的反应能力进行评估分析。ConfVD 的关键技术包括基于配置项分类的约束提取，配置故障自动生成和软件反应能力评估分析。本章给出了 ConfVD 系统框架和相关的关键技术。

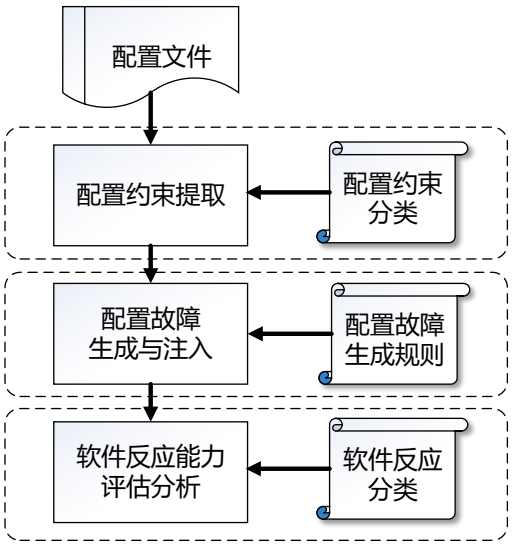

图 3.1　ConfVD 总体结构图

3.1　ConfVD 总体设计

本节将介绍 ConfVD 的总体设计框架。ConfVD（Configuration Vulnerability Detector）的主要思想是利用软件配置当中存在的约束人工生成配置故障，并通过评估分析软件系统对配置故障的反应能力，从而发现软件当中存在的对配置故障的不良反应。软件的反应能力，即软件在遭遇到配置故障的过程时发生的反应，例如软件正常运行、软件崩溃、挂起、没有提示信息的程序终止等等。良好的软件反应能力，例如程序检测到配置故障之后为了避免故障的传播，停止软件服务并在命令行或者日志系统中报告给用户发生配置故障的位置，能够大大减少配置故障发生之后诊断故障原因所花费的开销，同时代码当中对于配置故障的良好处理能够最小化地减少配置故障引起的损失。而不良的软件反应能力，即软件配置脆弱性（Vulnerability）[23]，如软件崩溃，挂起，或者包含意义不明误导用户的日志

信息，则在某种程度上加剧了诊断配置故障的难度。ConfVD 的总体结构如图 3.1。

ConfVD 的工作流程主要分为三个步骤。首先，对于得到的软件系统配置，提取出正确配置需满足的规则信息，即软件系统配置的约束。ConfVD 将软件配置的约束分为了两大类，配置项值类型约束以及配置文件格式约束。值类型约束为配置项值应该满足的约束，其又包含语法、语义两种约束。语法约束即配置项的值应当满足的文法约束。相应的，语义约束指语法约束满足的情况下，配置项值应当满足当前环境相关的约束。配置文件格式约束指用来满足配置文件特定格式需求的约束信息。

其次，在得到软件配置约束之后，ConfVD 制定了基于约束的配置故障产生规则。约束为软件配置应当满足的正确的规则，相应的，违反了配置约束即产生了配置故障。ConfVD 根据上述约束提取方法，分别提出了值类型配置故障及格式配置故障的生成方法。对于生成的配置故障，ConfVD 通过对软件配置的抽象表示，自动将其注入到不同的软件系统中。

最后测试已经注入的配置故障，ConfVD 记录并评估分析软件对于配置故障的反应能力，包括系统状态，软件日志等等。根据软件反应是否益于配置故障的诊断，ConfVD 将软件反应能力进行分类。同时，基于自然语言处理技术，实现了对软件反应行为的自动分析并归入前文提出的分类当中。

3.2 关键技术

本节将主要介绍 ConfVD 当中的关键技术，包括基于配置项分类的约束提取，配置故障自动生成与注入以及软件反应能力评估分析。ConfVD 首先提取出软件配置当中存在的正确规则，即配置约束，并通过违反这些约束生成配置故障，最后注入并测试这些配置故障，从而评估软件软件系统对于配置故障的反应能力。

3.2.1 基于配置项分类的约束提取

配置约束的提取作为配置故障生成的基础，也能被广泛运用在软件配置故障的其他相关工作，例如配置故障检测，配置故障自动诊断等等。目前针对软件配置约束的研究工作 [15, 23] 数量仍然较少，SPEX 主要依赖程序分析配置项读入之后的数据流控制流，并且仍需要人工改写软件代码，加入特定的 Annotation，无法完全自动地完成配置约束的提取工作。EnCore 对大量的配置文件进行机器学习，从而提取出配置文件当中蕴含的约束信息。然而该方案依赖大量的软件配置文件，在现实使用中存在一定的难度，同时，EnCore 依赖研究人员设置软件配置当中约束的模板。对于不存在模板的约束，EnCore 并不能实现软件配置约束的提取。

通过前期大量的调研工作，本课题发现配置约束与配置类型存在着千丝万缕的联系。一个软件配置项的类型往往能够决定这个配置项值的格式特征，甚至包

括与操作系统环境相关的语义信息。以软件 Httpd2.4 中的配置项“Port”为例，配置参数“Port”属于端口类型，这决定了“Port”必须是一个 0 到 65,535 范围内的整数，同时端口值必须是没被其他软件占用的端口号。本课题从软件配置项类型分类出发，归纳总结出了基于值类型的配置约束，即各个配置项类型所对应的语法约束，语义约束。同时，对于广泛存在于配置文件当中的配置格式要求，本文提出了基于配置文件特定格式特征的配置文件格式约束，用来描述该部分的约束特征，并用于后文的配置故障生成。

3.2.2 配置故障自动生成与注入

配置故障作为软件反应能力测试的输入，其质量直接决定了软件反应能力测试的结果。单一的，种类贫乏的配置故障使得软件反应也趋于单一，从而达不到测试的预期目的。现有技术关注配置故障自动生成仍比较稀缺，ConfErr [22] 使用简单的字符修改来模拟人类可能产生的拼写错误，例如遗漏，错拼，重复的字符等等。其生成的配置故障并没有软件配置约束作为指导，因此并不能产生包含特定配置约束的配置故障，例如当配置项值必须满足 0 到 7 的范围时，测试值越界为 8 的情况。SPEX [23] 克服了 ConfErr 的这点不足，提出了包括值类型、值范围等五大配置约束用于指导配置故障。然而由于 SPEX 的约束粒度过大，种类不丰富等问题，其产生的配置故障距离全面暴露软件不良反应能力仍有一定的距离。

基于本课题提出的值类型约束与配置格式约束，ConfVD 能够实现相应的配置故障自动生成，即基于值类型的配置故障（包括语法配置故障和语义配置故障）与配置文件格式故障等等。同时，ConfVD 使用了软件配置的抽象表示，通过树形结构对原配置文件进行抽象，并通过模板修改这些配置的抽象表示，最后将修改后的配置抽象表示重新组合成软件配置，从而实现将软件配置故障注入到不同种类的软件系统当中。

3.2.3 软件反应能力分析

目前已有的配置故障注入相关的工作 [22, 23] 目前仍然缺乏对于软件反应能力更加系统的研究，更多的是集中研究个别配置故障案例的情况。其根本原因是难以系统化地对于软件反应能力进行定义，从而区分软件反应能力的优劣程度。进一步的，对于软件应对配置故障所发生的软件能力，进行充分的评估分析，得到一个宏观的配置故障测试结果。

对于已经注入到软件系统的配置故障，ConfVD 启动软件测试来模拟软件的实际运行过程，并在该过程中，通过检测软件系统的命令行以及日志文件输出，观察并记录下软件系统对配置故障的反应情况。基于本课题观察到的软件系统对

于配置故障大量各异的反应，本课题从帮助诊断配置故障角度出发建立了软件反应能力评估模型，通过判断软件是否通过所有的测试用例，是否存在异常信息，是否能够指出配置故障位置，将软件反应能力分成 6 类。同时 ConfVD 使用自然语言处理技术，实现了对记录到的软件反应能力进行自动分类的工作。

第四章　基于配置项分类的约束提取

目前已有工作[15, 23]通过机器学习，程序分析等方法实现了软件配置约束的提取，但仍存在约束不够全面，依赖人为制定规则等弊端，距离投入实际生产与使用仍存在一定的差距。为了弥补现有工作的不足，本课题需要提出一种更加有效的约束提取方法。本课题观察到，如图 4.1所示，配置选项并不是用户随意的输入，其必须满足特定的类型要求。同时配置选项的特定类型，决定了该配置项取值的必须满足的条件。

为了更好的理解软件配置的约束信息，本章主要对 Squid，Nignx，Redis，Nagios，Lighttpd，Puppet，SeaFile 和 Vsftpd 这八款开源软件的 1593 个软件配置项进行了调研分析并进行分类。根据上述调研结果本课题总结出了覆盖率全面的软件配置类型分类树。基于提出的软件配置类型分类，本课题通过查阅标准文档，软件说明等分别制定出了基于值类型的配置约束以及配置文件格式约束。

Httpd 用户手册对配置项“ServerPath”的描述	RFC1738对URL语法的描述
… Option Name: ServerPath Syntax: ServerPath *URL-path* …	… In general, URLs are written as follows: *<scheme>:<scheme-specific-part>* … The schemes covered are: *ftp, http, gopher, mailto …* … common syntax for the scheme-specific data: *//<user>:<password>@<host>:<port>/<url-path>* …

图 4.1　软件配置类型约束实例

4.1　软件配置分类

尽管前人[46]已经有了对软件配置分类的相关调研工作，但是由于本课题所采用的配置类型主要用于产生提取软件约束并生成配置故障，已有工作的配置分类仍存在分类粒度不够细，不适用于开源 C 代码类型的软件等缺点，因此本课题需要找到一种更细粒度的软件配置分类。本节主要介绍软件配置分类的提出过程，包括调研的准备工作以及研究结果。并根于该研究结果，本课题提出了软件配置分类树。

4.1.1 开源软件配置项调研分析

本课题的调研对象为八款被广泛使用的开源软件，包括 Squid，Nignx，Redis，Nagios，Lighttpd，Puppet，SeaFile 和 Vsftpd。Squid 是目前热门的网页缓存代理软件，被广泛部署在如 Flickr 以及维基百科等等知名网站服务器之中。Nginx 是目前被数百万网站正在使用的轻量级网页服务器，包括 WordPress 和 SourceForge 等大型网站也正在使用。Redis 是一种开源的基于内存的数据服务器，其被广泛部署在 Twitter，Github，SnapChat 等知名网站之中。Nagios 是一种广泛流行的企业级系统监控软件，目前拥有超过 100 万以上的用户。Lighttpd 是一种更加高效和轻量的网页服务器软件。Puppet 被广泛部署在云服务器中，用来分发软件配置，进行节点操作等等，目前有超过 30,000 个组织机构正在使用。Seafile 是一个高可靠性与高性能的文件备份和分享系统，在世界范围内有超过 30 万用户。Vsftpd 作为 Linux 系统中默认的 FTP 服务器，在 Ubuntu，CentOS，Fedora 等操作系统中均有部署。

在本节中，本课题通过观察这些软件的配置文件，并查阅了配置相关的软件技术文档，记录下每个配置项的详细信息，包括配置项名，默认的值，相关的描述等等。由于本文中的工作基于键值对类型的配置实现的，因此对于不满足要求的配置项格式，本课题将其转化为键值对类型再进行记录。

4.1.2 配置分类树

实证研究的结果如表 4.1所示。通过该表可以观察到，绝大部分软件使用的配置项都能被十几种配置项类型所概括。本课题将这些配置项类型提炼出来并绘制成配置项分类树。如图 4.2所示，本课题将配置项类型分成字符串类型（String）和数字类型（Numberic）两大类。其中字符型又能分为统一资源标识符（URI），枚举型（Enumerative），名称型（Name），其他（Others）。而数字类型又分为带单位的数字类型（Number with units）和纯数字型（Number）。在这些类型下又进一步分为 22 个子类。例如，路径类型分为绝对文件路径，相对文件路径，绝对目录路径以及相对目录路径四个子类。同时，需要指出的是，为了保证该分类的全面性，该分类并不一定是固定不变的，其能够随着软件的扩充而进行相应的调整。关于本课题当中分类有效性的讨论将在第 7 章被讨论。

尽管本文中的配置分类被证明是非常有效的，然而在实际操作过程中仍存在一些困难。比如说，一些软件系统中配置项的类型并不唯一，以 MySQL 中的配置项“LogFile”为例，“LogFile”作用是定义日志文件的存放位置，同时根据文档所述，“LogFile”能通过 URL 设置在远程终端上。对于这种情况，本课题在统计过程中对于配置项的每种类型都统计了一次，并记录在了表 4.1中。同时，存在一些软件配置项格式比较复杂，例如 Httpd 当中存在一些配置项由两个以上的参数构

表 4.1　各类型配置项数量表

软件名	IP-address	Boolean	Mode	Path	Email	Count	Permission	Port	Domain Name	Name	Others	-
Squid-3.5	11	73	46	26	4	111	0	7	11	34	16	339
Nginx-1.10.3	7	114	124	65	1	186	5	0	7	104	24	637
Redis-3.2.8	2	13	9	6	0	33	0	1	0	0	6	70
Nagios-XI-5.6.3	0	33	10	17	2	37	0	0	0	16	8	123
Lighttpd-1.4.45	1	12	6	5	0	12	0	0	1	14	4	55
Puppet-4.10	0	45	1	77	0	19	0	1	6	57	2	208
SeaFile-6.0.4	0	15	2	1	0	13	0	1	3	1	2	38
Vsftpd-3.0.3	2	73	2	19	0	16	0	4	2	1	2	123

成的情况，无法直接转化为键值对的形式。对于这种情况，本课题在研究的过程中考虑每个参数的取值类型，并加入到统计数据中。对于没有出现在图 4.2中的配置项类型，本课题将其分类到类型“Others”当中，尽管这部分的配置项只占到了实证工作的总体配置数量的 3% 左右，然而在实际场景中，配置项的类型与软件是密切相关的，因此存在一些软件特定的配置项类型。对于这些特殊的类型，可以非常方便的扩展进该分类当中，并且相应的提取后文中的约束信息。

4.2　配置约束提取

软件配置约束为区分出正确配置与配置错误之间的规则。为了能够更好地评估软件反应能力，本课题需要产生尽可能丰富的配置故障，因此要求本课题能够尽可能细粒度地提取配置约束，然而这方面的工作仍存在诸多不足。Xu 等人 [23] 将配置约束分为五大种类，并依靠程序分析软件数据、控制流的方法，推断软件约束。Zhang 等人 [15] 通过机器学习配置文件当中设置，推断出配置文件中存在的约束信息。然而上述方法在实现过程以及本课题故障注入的应用场景中仍有诸多不足：Xu 等人的工作 SPEX 对于约束的分类粒度较大，并不适合生成种类复杂的配置故障，此外 SPEX 依赖开发者手动写入 Annotation 的方式进行半自动分析，而此过程需要大量的专家知识作为支撑。EnCore 需要大量的配置样本，同时需要软件系统相应的运行环境信息，因此在实现过程中存在难度，同时其推断约束的精确性极大地依赖样本，并不适合用来生成配置故障。因此，本课题提出了一种粒度更细，分类更加全面的配置约束提取方法来帮助生成配置故障。

基于上文对软件配置类型的观察，本课题发现软件配置类型蕴含了丰富的约束信息。本课题将这些约束分成基于值类型的约束与配置文件格式约束。同时值类型的约束又能分为配置类型语法约束和配置类型语义约束。语法约束指配置项值应该满足的文法，相应的语义约束指在满足文法的基础上，与软件运行环境例如操作系统状态等等之间需要满足的条件。以端口（Port）类型的配置项为例，其值的语法约束即为必须为 0 到 65,535 之间的整数值。相应的，其语义约束为，该端口号不能被系统环境中其他进程占用。格式约束则意味着软件配置应当满足的

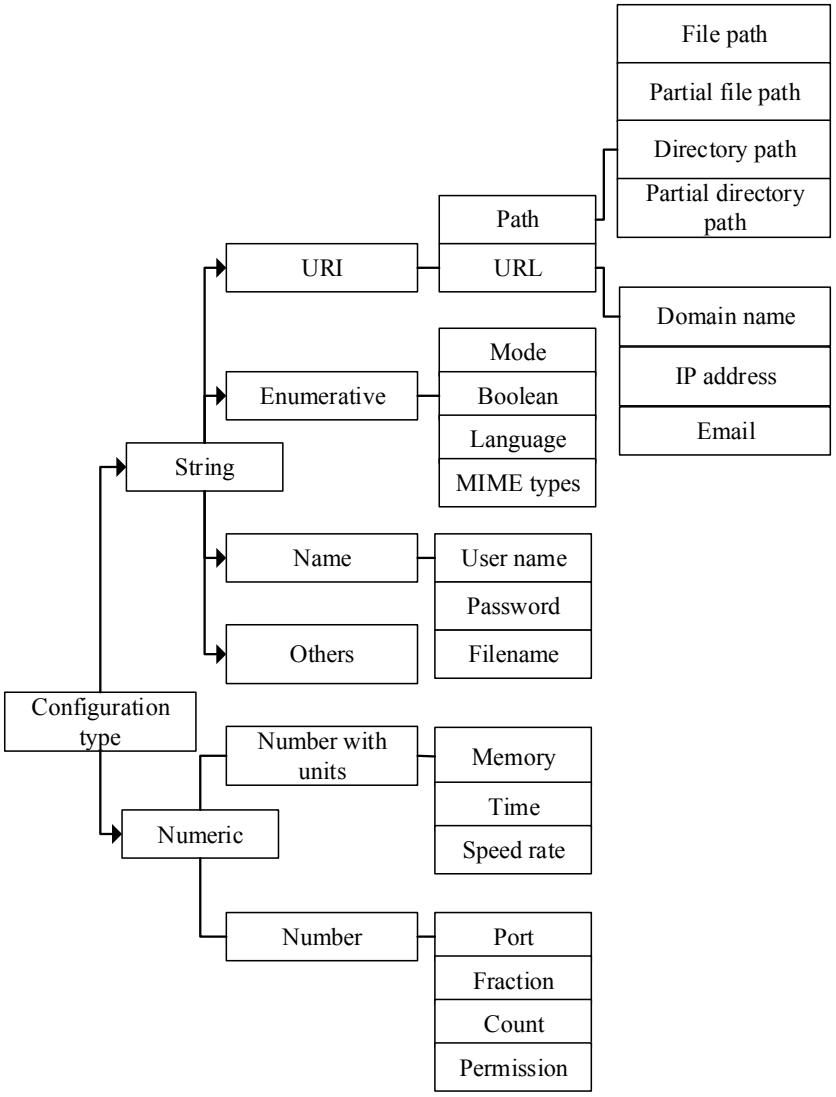

图 4.2　配置项分类树

一些格式要求。如图 4.3所示，值类型的配置约束如语法、语义约束作用在配置项值上，而配置文件格式约束则作用在整个配置文件中。本节将详细介绍这些约束的分类原则并阐述如何通过软件配置类型信息指导推断配置约束。

4.2.1　基于值类型的配置约束

4.2.1.1　配置类型语法约束提取

用户在设定配置项值的过程中，会通过默认配置、文档等观察了解该配置项的使用方法。配置项类型即为一个重要的参考，用户能够根据配置项类型对值进行设置和修改。因此本课题能够通过配置项类型的内在要求与相关领域知识（例如 RFC 文档）来进行约束的提取。

本课题采用预先定义的文法模式来表示语法约束。现有工作 EnCore [15] 通过

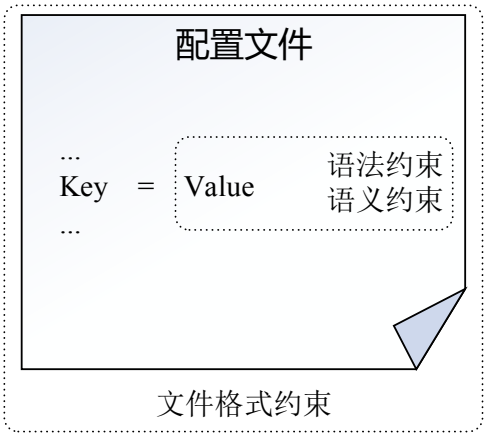

图 4.3　各类型约束作用范围

正则表达式来描述配置项值的语法约束。但是，由于 EnCore 中的正则表达式仅仅适用于推测配置项可能存在的类型，因此并不足以描述配置项值当中存在的复杂语法约束。例如，对于端口类型的配置项，EnCore 中的语法约束为正则表达式“[\d]+”，含义为 0 到任意多个整型数字，明显不包含 0 到 65,535 等范围约束。

为了更好地帮助生成配置故障，本课题采用了扩展的巴科斯范式（Augmented Backus-Naur Form，下文将采用 ABNF 进行指代）来更细粒度地描述语法约束。ABNF 是一种互联网规约语言，并定义于 RFC2234 [47]。如表 4.2所示 ABNF 将一大串字符串模式拆分为若干元素，元素可以使用正则表达式进行定义，同时本课题对于元素的定义过程引入了数值范围等概念，从而实现了语法约束的更好描述。以表 4.2当中的“File(Directory) Path”类型的配置项为例，该配置项值得 ABNF 含义为，其由 1 到任意多个“/”加上“DirString”元素构成，而字符串的最后可能存在“/”。而对于元素“DirString”的定义为，若干个文件目录系统当中允许使用的字符构成的字符串。通过这种定义元素，再将元素集成到字符串模式的方式，构成了 ABNF 对于软件配置语法约束的表达形式。同时，课题组相关工作能够帮助实现提取一些特定的约束，例如 Count 类型的范围，Mode 类型的取值集合等等。

4.2.1.2　配置类型语义约束提取

相对于明确直观的语法约束，语义约束代表着软件系统与运行环境之间复杂的关系。例如用户在配置前文中的端口（Port）类型的配置项时，还需要注意不能与软件运行环境中其他正在被使用的端口号相冲突。因此，语义约束并不具备一成不变的范式，它的提取是和软件当前环境紧密联系的。同时值得注意的是，语义约束并没有放之四海而皆准的普遍适用的规则，通常情况下，语义约束具有极强的软件特征，对于同种类型的软件配置项，不同的软件可能拥有着截然不同的

表 4.2　配置项值的语法约束

配置项类型	配置语法约束			
	ABNF 模式	元素名	元素定义	
File (Directory) Path	1* ⟨“/” DirString⟩ [“/”]	DirString	[\w.-]+	
Partial File (Directory) Path	DirString *⟨“/” DirString⟩ [“/”]	DirString	[\w.-]+	
Domain Name	[Scheme “://”] ServerName[“:”Port]	Scheme	(“telnet”/“https”/“http”/“ftp”/···)	
		ServerName	(\w)+((\.\w+)+)	
		Port	See option type “Port”	
IP Address	Int 3⟨“.” Int⟩	Int	1*3DIGIT	[0-255]
Email	Username“@”ServerName	UserName	(\w)+(\.\w+)*	
		ServerName	See element “ServerName”	
Mode	Mode	Mode	(“value1”/“value2”/“value3”/···)	
Boolean	Bool	Bool	(“on”/“off”/“yes”/“no”/“true”/“false”)	
Language	Lang	Lang	2ALPHA	
MIME-types	Type“/”SubType	Type	(“application”/“image”/“audio”/“text”/···)	
		SubType	1*(HEXDIG/“-”)	
Memory	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“K”/“M”/“G”/“T”/“KB”/“GB”/“TB”/“B”)	
Time	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“s”/“min”/“h”/“d”/“ms”)	
Speed Rate	Number Unit	Number	1*DIGIT	[min-max]
		Unit	(“bps”/“Mbps”/“Kbps”)	
Count	Int	Int	1*DIGIT	[min-max]
Fraction	Fraction	Fraction	1*DIGIT“.” 1*DIGIT	[min-max]
Port	Int	Int	1*DIGIT	[0-65535]
Permission	Octet	Octet	1*OCTET	[0-777]
Username	UserName	UserName	[a-zA-Z][a-zA-Z0-9]*	
Password	Pwd	Pwd	N/A	
Filename	FileName	FileName	[\w -]+.[\w -]+	

语义要求。就以文件类型的软件配置项为例，有些软件可能需要该文件为可读的，而另一款软件则可能要求其为可写的。然而这种语义层面的约束区别并不能从配置项字符的层面被简单区分，而需要通过比对配置项值与软件系统运行环境之间的关系才能判断得到。

本课题从实际出发，通过观察目前配置项的使用模式，总结出了一套常见的语义约束模板。本课题将语义约束分为两部分：配置项值的约束，以及值相关的环境属性约束。配置项值的约束如表 4.3所示，这部分约束描述了配置项在满足语法约束的前提下另外需要满足的和软件运行环境之间的依赖关系。举例而言对 URL 类型的配置项而言，配置项值除了满足正确的 URL 格式，该 URL 路径能否被当前环境访问到则取决于运行环境。值相关的环境属性约束如表 4.4所示，由于配置项通常包含一些属性来描述与运行环境之间的关系，而这部分属性并不会与配置项值有着直接的联系。受 EnCore [15] 启发，本课题从六种资源类型（网络，硬件，文件系统，安全属性，环境变量，运行服务）当中提取这些约束信息。

表 4.3　配置项值的语义约束

配置项类型	配置语义约束	相关资源
Path	该文件路径应该存在	文件系统
URL	该 URL 应该可达	网络
IP address	该 IP 地址应该可达	网络
Email	Email 地址需要存在	网络
Port	端口号不能被其他应用占用	运行服务
Language	系统应该支持该语言包	运行服务
MIME type	系统应该支持该 MIME 类型	运行服务
Memory	内存设置应该小于系统当前硬件支持内存大小	硬件
Speed rate	速率设置应该小于硬件带宽	硬件
Username	用户名应该来自 root 用户组	安全

表 4.4　配置项值相关的环境属性

配置项类型	运行环境相关的属性	类型	描述	相关资源
Path	Owner	String	该路径的拥有者	文件系统
	Group	String	该路径所归属的组	
	Permission	Octal	该路径的权限信息	
	Contents	String	该路径的内容	
	HasSymLink	Bool	是否存在符号链接	
	Type	Enum	该路径的类型	
URL	Forbidden	Bool	URL 是否被防火墙禁止	网络
	Type	Enum	该 URL 的类型	
IP address	Forbidden	Bool	该 IP 地址是否被防火墙禁止	网络
	Type	Enum	IP 地址的类型	
Port	User	String	Port 的用户	运行服务
	Type	Enum	端口号的类型	
Language	IsSupported	Bool	系统是否支持该语言	运行服务
MIME type	IsSupported	Bool	系统是否支持该 MIME 类型	运行服务
Memory	Allocation	Enum	内存分配策略	硬件
Username	GroupName	String	用户名所归属的组	安全

4.2.2　配置文件格式约束

软件配置通常使用特定的格式保存在软件系统当中，常见的配置形式有 Windows 下的注册表，Java 软件常用的 XML 文件，以及纯文本文件等等。为了方便用户使用，规范配置管理过程，这些软件系统配置通常都具备一些特定的格

式。例如软件 Yum3.4.3 中的配置文件 yum.conf，如 4.4，配置文件中包含了一个 section 头部“[main]”，用来定义内部配置项的作用范围。而配置项遵循“键 = 值”的格式，依次逐行排列。

因此本课题认为这样类似的规范格式即为软件的配置文件格式约束。针对不同的软件系统，配置文件格式约束存在不同，但对于相同的软件，配置文件格式是一定的，因此软件在配置过程中必须满足这些约束。对于文件格式约束，能够通过简单的字符串分析加以实现。

```
[main]
cachedir=/var/cache/yum
debuglevel=2
logfile=/var/log/yum.log
pkgpolicy=newest
distroverpkg=redhat-release
tolerant=1
exactarch=1
retries=1

[base]
name=Fedora Core $releasever - $basearch - Base
baseurl=http://download.atrpms.net/mirrors/fedoracore/$releasever/$basearch/os
http://rpmfind.net/linux/fedora/cor...er/$basearch/os
http://mirror.clarkson.edu/pub/dist...er/$basearch/os

[updates-released]
name=Fedora Core $releasever - $basearch - Released Updates
baseurl=http://download.atrpms.net/mirrors/fedoracore/updates/$releasever/$basearch
http://redhat.linux.ee/pub/fedora/l...sever/$basearch
http://fr2.rpmfind.net/linux/fedora...sever/$basearch
```

图 4.4　Yum 配置文件内容

第五章　配置故障自动生成与注入

配置故障是本课题当中软件反应能力分析评估的基础。自动化生成配置故障的主要挑战，即在于如何模拟现实中的配置故障。然而由于巨大的错误空间[25]，尝试测试所有可能的错误是几乎不可能的。因此，生成配置故障的关键在于使用尽可能少的有效的，实际的配置故障。同时生成配置故障的过程应当具备系统性的特点来帮助暴露尽可能多的软件脆弱性。

已有的生成配置故障的研究主要包括 ConfErr [22] 和 SPEX [23]。ConfErr 只是对配置项值进行简单的修改来模仿人类的拼写错误，无法生成约束相关的配置故障，从而带来了测试过程的局限性。SPEX 引入了配置约束的概念，然而 SPEX 提出的约束只包含值范围，值类型，依赖关系等五大类，过大的约束粒度影响了配置故障产生的效果。因此本课题提出了更细粒度的约束来帮助生成更加丰富的配置故障，并实现了工具 ConfVD 实现了配置故障的自动生成与注入过程。本章主要介绍了 ConfVD 基于前文约束的配置故障自动生成方法，在自动生成配置故障之后，通过修改配置文件等方式将配置故障注入到了软件系统当中。

5.1　配置故障生成

与前文的基于值类型的配置约束与配置文件格式约束相对应，本节将介绍基于值类型的配置故障与配置文件格式故障的生成方法，即通过违反相应的约束产生配置故障。同时对应上一章当中的分类规则，基于值类型的配置故障又能分为语法配置和语义配置故障两部分。

5.1.1　基于值类型的配置故障

5.1.1.1　语法配置故障

ConfVD 能够通过违反语法约束来生成配置故障。语法配置故障的生成流程如下：首先，通过前文的约束提取工作，ConfVD 能够得到软件配置项对应的语法约束，即 ABNF 范式规约，ABNF 范式中包含了各个元素。其次，ConfVD 对 ABNF 范式中的各个元素进行基于规则的变异操作。然后 ConfVD 将这些变异的元素与正常的元素再次拼接起来，组成了一个语法故障候补。最后通过检查该候补是否违反了当前的语法约束，来决定是否保留该语法配置故障。

其中，产生配置故障的关键为变异操作。变异操作又可以分为元素级别的变异操作与格式级别的变异操作。元素级别的操作作用在 ABNF 范式当中的各个元素上，例如 ConfVD 能够随机替换，增加，删除元素上的字符。而格式级别的变异操作作用于多个元素，来实现元素的删除，倒序等操作。经过变异操作，ABNF

表 5.1　语法配置故障生成实例

变异操作	配置项样例: MemSize=64MB					
	变异前		变异后			结果
	Number	Unit	Number	Unit		
替换字符	64	MB	64	Ma		64Ma
增加字符	64	MB	64	Mba		64Mba
转换大小写	64	MB	64	mb		64mb
删除字符	64	MB	64	M		64M
范围越界	64	MB	129	MB		129MB
改变数值类型	64	MB	64.5	MB		64.5MB
打乱元素	64	MB	Unit	Number		MB64
删除元素	64	MB	Number			64
重复元素	64	MB	Number	Unit	Unit	64MBMB

即产生了变异了的 ABNF 元素，再将这些变异后的元素与未经变异的元素进行拼接，即得到了配置故障的候补。

表 5.1给出了 ConfVD 生成语法故障的一个实例。对于配置项“MemSize”首先 ConfVD 得到它的类型是属于图 4.2中的“Memory”类型，对于该类型它的 ABNF 范式包含两部分元素：“Number”和“Unit”。表 5.1展示了元素级的变异操作，例如替换字符，增加字符，转换大小写，删除字符等等。同时对于“Number”元素，生成了范围越界的数值“129”，以及改变了其数值类型，将其从整型修改为了浮点型。而对于格式级别的操作，表 5.1中列出了倒序，删除某个元素，重复某个元素等操作。以上这些操作都能够通过变异前和变异后详细查看到各个元素的变化，同时产生的配置故障如结果一列所示。表 5.1所展示的操作内容只是一个简化的过程，实际操作中 ConfVD 会对每个元素都进行完整的变异操作以确保产生覆盖面更加广泛的配置故障。

为了更好的解释语法配置故障的生成过程，本文使用操作“替换字符”来进行说明。以配置项“MemSize”为例，配置项“MemSize”的初始值为“64MB”，通过 ABNF 得知该“Memory”类型的配置项包含两部分元素构成：“Number”和“Unit”，通过元素的正则匹配本课题能够得知“64”属于“Number”元素，而“MB”属于“Unit”元素。接下来对“Unit”元素进行替换字符操作，将字符“B”随机修改为“a”，从而得到了“MB”的变异元素“Ma”。接下来将变异元素“Unit”与之前未改变元素“Number”拼接起来从而得到了一个候选的配置故障“64Ma”。接下来 ConfVD 检查“64Ma”该值是否满足语法约束，即 ABNF 范式，结果是“64Ma”不满足该范式要求，因此生成了配置项“MemSize”的一个语法故障。通

过对 ABNF 范式定义的语法约束的违反，ConfVD 能够更加系统更加全面的地生成语法配置故障，从而更好地测试软件反应能力。

5.1.1.2 语义配置故障

与语法配置故障生成相类似，ConfVD 也能通过语义约束生成语义配置故障。通过语义约束的描述，本课题定义了一系列规则模板来表示违反这些语义约束的方法。如图 5.1所示，ConfVD 使用规则来通过违反配置项值与运行环境之间的关系生成配置故障。另一方面，ConfVD 也能改变软件运行环境来实现这一目的。

同时由于语义约束灵活性及软件特定性，ConfVD 能够提供定制化的规则模板接口使用户能够扩展和修改语义配置故障生成的规则。ConfVD 本身也提供了大量的规则供用户直接使用。

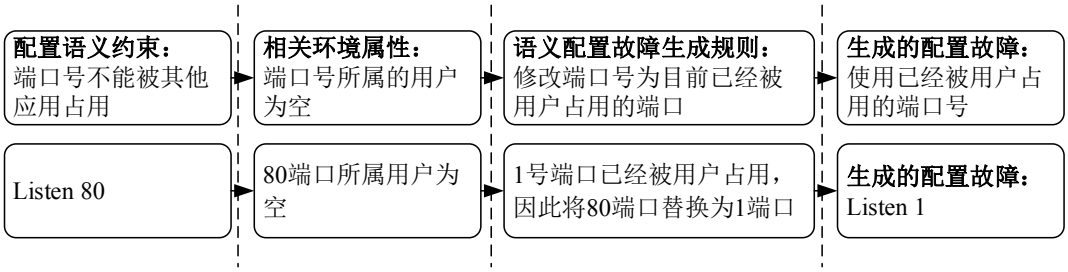

图 5.1 语义配置故障生成实例

5.1.2 配置文件格式故障

用户在配置软件过程中，由于复制粘贴或者其他误操作很容易引起软件配置格式的错误。因此，ConfVD 模拟用户的误操作，用来测试软件是否能够检测出并很好地处理格式相关的配置故障。

格式配置故障的生成步骤包括获取当前配置的格式约束和违反约束生成配置故障两大步骤。ConfVD 运行时会首先读取目标软件系统的配置文件，根据当前配置的特征定义格式约束。然后通过违反该格式生成配置故障来对软件系统进行测试。

表 5.2给出了格式配置故障生成的例子。对于配置项“EnableLog = On”，分别产生了如下的格式配置故障：遗漏了配置项名“EnableLog”；错误拼写产生了“EnableLogs”；删除了配置项值“On”；改变了配置项名的大小写生成了“enablelog”；使用了错误的操作符“:”；删除了操作符“=”。另外对于一些包含 section 头部的配置文件，还定义了一部分规则用于改变配置文件中的结构。

表 5.2　格式配置故障生成实例

配置故障生成规则	配置项样例: EnableLog = On	
	修改前	修改后
遗漏配置项名	EnableLog = On	= On
错误拼写	EnableLog = On	EnableLogs = On
删除配置项值	EnableLog = On	EnableLog =
改变配置项名大小写	EnableLog = On	enablelog = On
使用错误的操作符	EnableLog = On	EnableLog : On
删除操作符	EnableLog = On	EnableLog On
使用错误的 Section 结构	[mysqld]	[mysqld
使用错误的 Section 名	[mysqld]	[mysqlf]

5.2　配置故障注入

ConfVD 的配置故障注入即将生成好的配置故障注入到待测试的软件系统当中，其步骤如图 5.2。ConfVD 读入软件配置，并将其用格式化，使用统一方式进行表示，并套用配置故障生成模板产生配置故障，并对格式化表示的配置进行相应的修改，最后将格式化的软件配置重新生成软件能够读写的配置文件供测试使用。

本节主要介绍了 ConfVD 如何将配置故障注入到软件系统，以及其中包含的所有技术细节，主要包括软件配置格式化表示方法，配置故障模板。

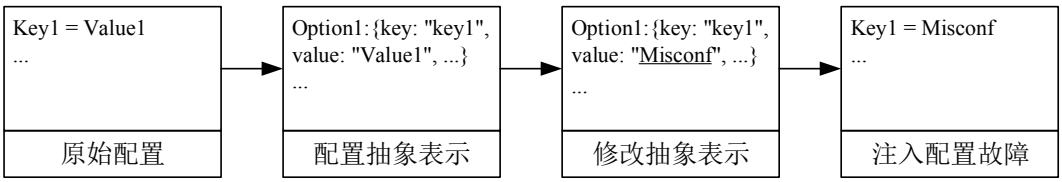

图 5.2　ConfVD 配置故障注入过程

5.2.1　配置抽象化表示

如前文所述配置在不同的软件系统中有着不同的格式，尽管有过统一软件配置表示方面的努力和研究 [25]，但是实际生产过程中，配置在软件设计的过程中还是存在着鲜明的软件特征。因此，ConfVD 如何对不同种类的配置文件实现故障注入，是本节主要讨论的技术方案。

Augeas [48] 是一种方便用户对软件配置进行管理的软件，能够解析目前市面上主流软件的配置文件并生成树结构的形式，例如 XML 文件来保存用户的配置。

然而 Augeas 对配置文件的解析主要应用于用户增添、删除和修改配置，对于故障注入仍存在不适用的方面。例如无法通过对 XML 的修改来生成结构相关的配置故障，等等。

因此 ConfVD 对配置文件的抽象化表示使用了更加底层的方式。首先 ConfVD 去除了软件配置当中的注释信息，对于配置信息使用了如图 5.3中树的形式对配置的结构进行保存，生成 XML 格式文件。ConfVD 逐行将配置文件中的信息进行拆分，将每行配置项拆分为键、值、操作符等部分，作为该行的若干个子节点。当需要修改配置信息的时候，只需要对该树中的某个节点进行操作，例如当需要删除某个配置项值时，只需要删除该配置项下的子节点，再重新将修改过的树生成为包含故障的配置文件。

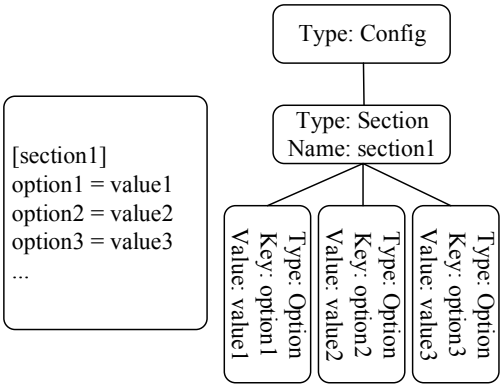

图 5.3　配置格式化表示

5.2.2　配置故障生成模板

ConfVD 使用配置故障生成模板对配置树进行转换。5.2.1 中配置被抽象化为若干个树节点的形式，通过规则模板 ConfVD 能对树进行相应的操作。例如增加删除树节点，修改某个节点中的内容，改变同节点下的子节点顺序等等。ConfVD 提供了一系列的故障生成模板来帮助前文的故障生成技术改变已有的软件系统配置。同时 ConfVD 也是高度可定制的，用户能够定义自己的故障生成模板，来实现定制化的功能。

第六章　软件反应能力分析

当软件系统遭受配置故障影响时，可能会出现不良反应，甚至导致软件失效，系统崩溃等，对于部署在军事、金融、医疗的软件系统，不良影响巨大。因此对软件反应能力的评估分析变得尤为重要。然而目前少有工作对软件反应能力评估分析提出系统的解决方案。

ConfVD 基于配置故障测试，使用业界标准的软件测试方法测试软件对于配置故障的反应能力，并根据软件反应对于配置故障诊断的帮助情况进行评估分析。

6.1　配置故障测试

6.1.1　测试流程

ConfVD 在读取配置文件之后生成若干包含配置故障的配置文件，并观察被测系统（system-under-test, 即 SUT）[22] 的反应能力。其过程如图 6.1所示。对于配置故障集合 S，ConfVD 遍历当中的每个配置故障 s，对于配置故障 s，ConfVD 能够产生有且仅包含配置故障 s 的配置文件 f。由这样一组配置文件 f 构成了错误配置文件集合 F。测试流程即为测试 SUT 对于 F 当中所有错误配置的反应能力。

ConfVD 使用测试用例来对软件进行测试。每个测试用例包含了启动、关闭在内的若干个功能性测试。为了保证软件测试用例是能可靠运行的，在 SUT 接受配置故障测试之前，ConfVD 会先在没有配置故障的情况下运行一遍测试用例，当软件能够成功完成所有测试用例时才会开始对配置故障进行测试。ConfVD 将测试过程中的软件反应信息记录到反应能力报告中，包括软件测试结果以及系统日志信息。通过对于这部分内容的分析评估，即能得到软件对于配置故障的反应能力情况。本文将在 6.2 节中介绍软件反应能力的分析评估工作。

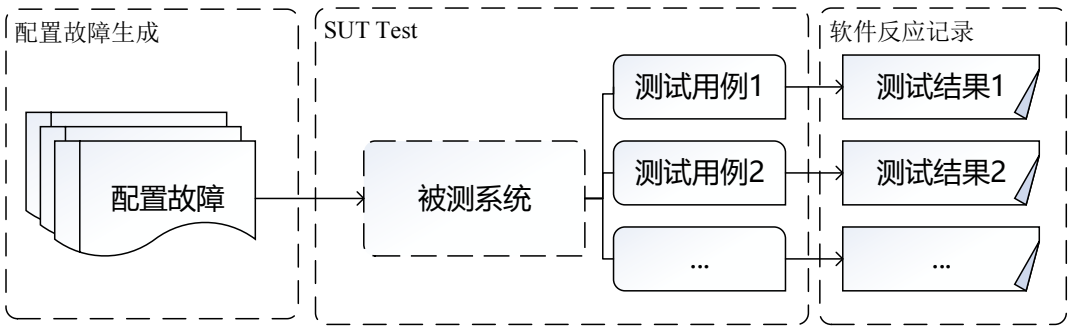

图 6.1　配置故障测试流程

6.1.2 测试用例

在实际操作过程中，通过对 ConfVD 结果的观察发现测试结果的有效性依赖于软件的测试过程是否充分。因此选取可靠的，专业的测试是必要的。然而软件测试的编写需要大量的领域专家知识，同时大规模软件的空间爆炸问题也阻碍了该目标的实现。

ConfVD 选取和使用了软件官方提供的测试用例，例如在第 7 章实验与评估中，MySQL 使用的测试用例来自 MySQL Test Framework[49]。这些测试用例提供了一套标准的结果评估方式。同时，ConfVD 支持用户使用定制化的软件测试脚本来满足一些特定的功能测试需求。

当软件测试用例无法满足需求时，即配置故障注入到软件系统之后变成了潜在的配置错误（Latent Configuration Error）[24] 而无法被触发，ConfVD 能够借助 Xu 等人提出的配置故障提前检测技术[24]，将软件配置相关的代码封装并在软件启动时进行触发，从而达到充分测试软件反应能力的目的。

6.1.3 测试结果记录

在软件遭遇配置故障时，用户可以根据软件的反应来尝试对配置故障进行修复。本小节主要介绍 ConfVD 如何记录下软件用户对配置故障进行诊断的“证据”。ConfVD 通过运行脚本等方式监控软件系统在测试运行过程中的各种输出信息，包括测试用例的输出结果，测试阶段软件日志的输出，系统日志等等。ConfVD 通过记录这些信息来模拟软件用户诊断配置故障搜集信息的过程。

基于上述的软件测试流程，ConfVD 对 SUT 得到的测试结果能分为以下三种：

- 1）SUT 启动失败。通常情况为配置故障引起软件系统错误阻止了软件启动。
- 2）SUT 启动成功但是没能通过全部的功能测试。一般为配置故障未被程序检测到，从而进入了软件运行过程引发了功能测试的失败或者组件的失效。
- 3）SUT 通过了所有的测试。软件可能阻止了故障的传播，但也有可能是由于测试自身限制，导致故障并未被触发。

由于软件在正常配置的条件下能够完成所有测试用例，对于结果 1）和 2）引起的软件系统失败本课题认为是由于注入的配置故障导致的。因此需要记录这两种情况下的软件反应信息，包括运行过程中的软件日志，失败的测试用例，注入的配置故障，默认的配置信息。而对于结果 3），也需要记录下当前运行的日志信息以及配置故障的相关信息。

图 6.2展示了一个软件反应记录的实例。对于 Httpd 当中的配置项“Listen”，首先由约束推断得到其满足端口类型（Port）的配置约束，因此注入了配置故障模拟了该端口号被占用的情况。在使用测试用例“systemctl start httpd.service”对 httpd 进行启动的过程发生了失败，记录下的系统日志如图所示，其中包含了“Permission denied”等字眼的异常信息。

配置项名称： Listen	**类型：** Port	**注入的配置故障：** Listen 1 //使用已占用的端口
测试用例： systemctl start httpd.service		
记录到的系统日志： ... Jan 15 18:58:32 bogon systemd[1]: Starting The Apache HTTP Server... Jan 15 18:58:32 bogon httpd[26606]: (13)Permission denied: AH00072: make_sock: could not bind to address [::]:1 Jan 15 18:58:32 bogon httpd[26606]: (13)Permission denied: AH00072: make_sock: could not bind to address 0.0.0.0:1 Jan 15 18:58:32 bogon httpd[26606]: no listening sockets available, shutting down Jan 15 18:58:32 bogon httpd[26606]: AH00015: Unable to open logs ...		

图 6.2　测试结果记录

6.2　软件反应能力分类模型

软件配置故障和软件 Bug 最大的不同在于，软件配置故障能够通过软件良好的反应，例如清晰的日志输出，帮助用户自身完成故障的诊断工作。尽管软件反应能够帮助用户诊断配置故障发生的原因，参差不齐的软件代码质量直接影响了用户诊断配置故障成功与否。不良的软件反应，如崩溃，挂起，模糊地日志输出等等，反而会阻碍整个配置故障诊断过程。然而面对日益严峻的配置故障挑战，软件系统对于配置故障的反应能力仍不容乐观。例如图 6.3所示，软件 OpenLDAP 中配置项“Listener-threads”被设置超过 16 的时候，服务器就会发生崩溃，并报出故障信息“Segment Fault”，然而，面对用户对这种模糊地日志输出的反馈，开发者认为这只是用户设置错误了而已，而不给予重视，并选择关闭了该用户报告来回避该问题[23]。开发者对待配置故障如此的态度比比皆是，Xu 等人[23] 则认为用户并不应该为开发者在配置设计实现当中的不合理因素买单，因此非常有必要对软件系统的反应能力进行评估分析，来找出软件系统中对配置故障的脆弱性并加以改进。

本课题基于对于软件配置故障反应情况的调查，将软件反应能力分为六大类。

配置项名称： Listener-threads	**类型：** Count	**注入的配置故障：** Listen-threads 32 //超过数量限制16
记录到的系统日志： ... Feb 18 10:20:09 oradb slapd[20909]: segment fault ...		
开发人员反馈： 拒绝修改源码与用户手册，并认为这只是用户的错误设置		

图 6.3　软件不良反应实例

表 6.1　软件系统反应分类表

是否通过了所有测试用例？	是否存在异常日志信息？	是否能够诊断出配置故障？	简称
是	是	是	Type 1
是	是	否	Type 2
是	否	-	Type 3
否	是	是	Type 4
否	是	否	Type 5
否	否	-	Type 6

如表 6.1所示，根据软件反应结果，ConfVD 首先判断软件系统是否通过了所有的测试用例。在此结果基础上，进一步观察软件在测试过程中是否产生了除正常日志以外的异常日志。最后分析产生的异常日志，是否明确地指出或者能够帮助诊断 ConfVD 注入的软件配置故障。本课题当中使用表 6.1中的软件反应能力 Type 1 到 Type 6 来分别指代这些软件反应。例如 Type 4 类型的软件反应，其含义为你软件系统没能通过所有测试用例，并在此同时产生了异常日志信息，通过分析这些异常日志信息，ConfVD 发现这部分信息能够定位到配置故障。

6.3　反应能力自动分析

基于上文中的软件反应能力分类模型，ConfVD 能够对软件的反应能力进行系统化地分类评估。然而由于软件反应能力的分析过程基于人工并且高度依赖专家知识，自动化实现该过程将极大地提高反应能力分析评估过程的效率，同时避免了繁重的人力劳动，避免了人为引入的错误。因此 ConfVD 记录下软件反应过程中的命令行、日志等输出信息，并对这些信息进行基于自然语言处理的分析，检查这些信息是否足够充分能够帮助诊断配置故障，并将其划分到节 6.2 中的分类当中。自动分析的主要步骤为：通过软件测试，能够根据测试结果判断得到软件是否通过了所有测试，即为反应分类的第一步。通过异常日志提取，能够得出

软件是否存在异常日志，即完成了分类的第二个步骤。最后分析这些异常日志是否能够帮助诊断出配置故障，即对异常日志进行诊断内容的充分性分析，即为分类的第三步。由于软件开发人员提供的测试用例都具有很好的使用接口，因此反应分类的第一步即判断测试用例是否全部通过能够直接实现，本文主要讨论自动分析的第二、三步。

6.3.1 异常日志提取

由于日志信息数量庞大，直接对整个测试过程的日志输出进行分析并不现实。由于配置故障测试过程的特殊性，测试用例导致软件系统日志输出遵循特定的序列特征，因此异常日志提取只需要比对现有日志序列与特定日志序列，若存在序列的差异，则认为存在异常日志。然而在实际中日志当中包含时间，数值等变量信息，两次相同的日志序列其中的日志也并不是完全相同的。因此需要对日志信息进行格式化操作，去除当中诸如时间“2017-10-25 15:59:17”等变量信息。

具体步骤为首先对正常日志序列集合进行格式化操作得到序列 S，以一组新的格式化的日志序列 S’作为输入，遍历其中的每条日志，从正常日志当中查找相似日志并进行标记，若不存在相似日志则标记为异常日志。遍历结束后，若该日志序列满足正常日志序列，则该日志表示软件正常运行，否则存在异常反应，且异常日志即为 S’与 S 的差异部分。

如图 6.4所示，本文给出了 MySQL 正常日志序列与异常日志序列的一个对比参照，并对异常日志提取步骤加以说明。对于正常日志序列，图中节选了 3 条日志进行说明，我们将其标记为 1、2、3 号日志。通过格式化操作，ConfVD 将去除日志当中的时间戳等信息。对于新的日志 ConfVD 采取同样的操作得到了日志序列 S’。通过对于可以发现，S’ 当中也出现了 1、2、3 号日志，然而 1 和 2 号日志当中出现了新的日志，即图中虚线框的部分，作为正常日志当中不会出现部分，这些日志即被标记为异常日志，记做 δ(S)。

通过上述步骤，ConfVD 能够在注入软件配置故障之后，分析并提取出软件执行后出现的异常日志，并为日志诊断内容充分性分析做了铺垫。

6.3.2 日志诊断内容充分性分析

良好的日志信息能帮助用户找到软件崩溃的原因，为了评估异常日志当中诊断内容的充分性，将分析结果分为以下三种情况：

- 如果异常日志 δ(S) 中包含错误配置项的名称或配置项的值，或异常日志明确指出了配置文件中某一行出错导致系统故障，则认为日志的诊断内容是充分的。

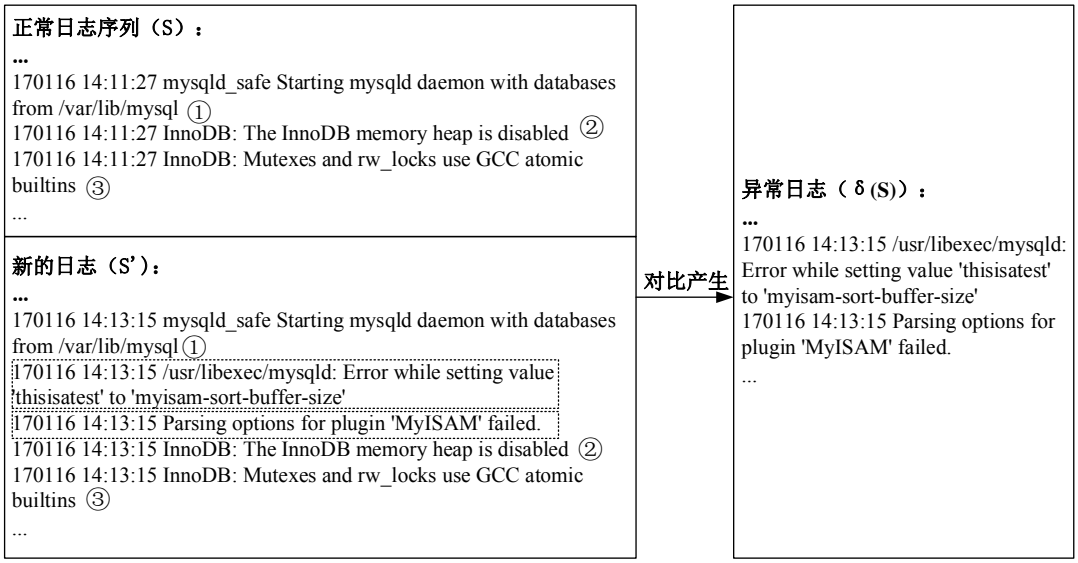

图 6.4　异常日志提取实例

- 如果异常日志 $\delta(S)$ 与用户手册对于该配置的描述存在一定的语义相似性，则认为日志的诊断内容是充分的。因为用户可以根据日志的内容找到相关的配置项。

- 对于不满足上述两种的情况，认为异常日志 $\delta(S)$ 的诊断内容是不充分的。

对于第一种情况，只需分析异常日志是否包含相关配置项的名称或配置项的值，或分析异常日志是否明确指出了配置文件中某一行出错导致系统故障，因此利用简单的字符串匹配功能就能进行实现。而对于第二种情况，需要计算出异常日志与用户手册相关内容的语义相似性，本课题使用自然语言处理的方法计算异常日志与用户手册相关内容的相似性。如果异常日志与配置项手册的语义相似性较高，可认为日志是充分的。

对于两个给定的文本，为了获取两者在语义层面的相似度分数，不能仅考虑词汇间的匹配。对于文本语义相似度的度量应该综合考虑文本的结构和单词的语义信息，但是根据前人在基于语料库的文本相似度研究 [50]，通过将单词相似度和单词特异性的结合考虑也可以较好的表示两个待比较文本的语义相似性。因此，本课题采用一种基于语料库和知识库的计算文本语义相似度的方法，来判断软件系统的异常日志信息是否和文档中的配置文件描述语义相关，进而判断软件系统的异常日志信息的诊断内容是否充分。

在考虑文本相似度时，除了单词的相似性之外，还应该考虑到单词的特殊性，以便可以对两个特定词（例如，fail 和 go wrong）之间识别的语义匹配给予更高的权重，并且对计算过程中不重要的词（例如 get 和 become）给予相对较低的权重。

本文使用由 Sparck-Jones 在 1972 年引入的逆文档频率（idf）来确定单词的特异性。单词的逆文档频率的定义为语料库中的文档总数除以包括该单词的文档总数。前人的工作 [50] 已经从理论上证明了逆文档频率在计算文本语义工作时的有效性。假设有一篇长文档，需要在不加以人工干预的前提下用计算机提取其关键词，很容易想到一篇文档中出现次数最多的词，往往能够反映文档的主题。但是单靠词频是不能完成这项的工作的，这需要一个重要的调整性系数来衡量一个词是不常见词：如果某个词在其他文档中很少见，但在本文档中出现的次数越多，这个词就越能反映这篇文档的主题。通常逆文档频率作为该词的调整性系数。

ConfVD 采用对 idf 取对数函数的方式，使得计算逆文档频率的结果尽量平滑。这是为了在语料库中，即使一个单词出现了 10 次，也不应该在计算特征权值时，比出现 1 次的情况的权值大 10 倍。所以加入了对数机制来抑制这种过大的差异。综上所述，公式 6.1给出了计算单词逆文档频率的方法。其中 num_of_corpus 代表语料库中的文档总数，num_of_wordfile 代表包括该单词的文档总数。

$$idf(word) = log(num\_of\_corpus/num\_of\_wordfile) \tag{6.1}$$

给定两个句子，首先通过句子分块，将一个句子划分成逻辑上紧密相关的单词或短语的组合。如短语“go wrong”就是一个基本逻辑单元。其次可以计算出句子 T_1 中每一个单词（或短语）w_1 与句子 T_2 中每一个单词（或短语）w_2 的相似度，记做 word_sim(w_1,w_2)。词汇相似度的计算方式有多种，如基于语料统计的词汇相似度计算和基于语义词典的词汇相似度计算。本课题使用 WordNet [51] 语义关系计算两个单词的相似度。然后考虑单词的特异性而引起的单词权重的不同，这里使用每个单词相对语料库的逆文档频率代表其相对权重。在实际操作中，一个文档代表待测试配置项的用户手册，语料库为该目标软件的所有配置用户手册的集合。

在已有单词间相似度计算方法 word_sim(w_1,w_2) 和单词权重计算方法逆文档频率 idf 的基础上，首先对于句子 T_1 中的每一个单词 w_1，计算出在句子 T_2 中与其相似度最高的单词的相似指数。然后每个单词的最高相似指数乘以其逆文档频率后求和，累加值标准化成值域为 [0,1] 的小数。句子 T_2 做同样的计算，最后将句子 T_1 和 T_2 的结果求平均值即为句子 T_1 和 T_2 的语义相似度指数。

以下是计算两个文本 T_1 和 T_2 的相似度公式：

$$sim(T_1, T_2) = \frac{(sim'(T_1, T_2) + sim'(T_2, T_1))}{2} \tag{6.2}$$

$$sim'(T_1, T_2) = \frac{\sum_{w_1 \in T_1} maxSim(w_1, T_2) * idf(w_1)}{\sum_{w_1 \in T_1} idf(w_1)} \tag{6.3}$$

$$maxSim(w_1, T_2) = max_{w_2 \in T_2}(word\_sim(w_1, w_2)) \quad (6.4)$$

上述公式中 maxSim 计算出单词 w_1 与句子 T_2 的最大语义相似程度，即单词 w_1 相对句子 T_2 中相似度最高的单词的相似度。sim' 通过使用逆文档频率 idf 和标准化每个句子中长度，合并了一个句子相对一个句子的单个单词相似度。sim 将两个 sim' 函数取了平均值。因此 sim 可以表示每个文本的每个单词与另一个文本的每一个单词的相似度比较。

ConfVD 使用 Python 语言实现上述算法。ConfVD 使用“The Stanford Parser”工具进行句子分块。对于计算两个单词相似度，本课题使用 NLTK 工具包。Natural Language Toolkit，中文全称为自然语言处理工具包，是自然语言处理领域最常使用的一个 Python 库，其中的 nltk.wordnet 子模块提供 WordNet 的接口 wordnet.synsets(word) 用于计算 word 的近义词。在计算 word1 和 word2 是否为近义词时，只需判断 word2 是否在 word1 的近义词集合中即可。

在实现过程中，当单词 w_1 和单词 w_2 在 WordNet 中语义相似时，设定 word_sim(w_1,w_2)=1；否则设定 word_sim(w_1,w_2)=0。

句子 T_1 和句子 T_2 的相似度 sim(T_1,T_2) 的值是 [0,1] 间的浮点数。两个完全相同（或语义完全相同）的句子相似度为 1。完全不相关的两个句子相似度为 0。判断两个句子是否语义相似，如果 sim(T_1,T_2)$\geq \delta$，则认为这两个句子语义相似。(⊠是设定的相似阈值，在本文中 δ 取 0.3)

在计算 maxSim 的值时，只考虑具有比较意义的单词。如冠词“the”、“a”对于句子的语义没有任何影响意义，不做考虑。对于文本中的数字和专有名词，如诊断日志“database system was shut down at 2017-01-16 11:57:17 GMT”中的“2017-01-16”时间等也不予考虑。

若 ConfVD 判断出异常日志信息与用户手册中该配置描述相似，则认为该异常日志信息的诊断内容是充分的，能够帮助定位配置故障，从而完成了软件反应的分类的最后一步工作。

第七章　实验与评估

本章主要内容对 ConfVD 进行的实验设计以及对结果的评估工作。表 7.1展示了 ConfVD 运行和测试使用的软硬件环境。表 7.2为实验评估使用的四款软件及相关信息：MySQL，Httpd，Yum 和 PostgreSQL。

表 7.1　ConfVD 运行测试软硬件环境

硬件环境	软件环境
戴尔 Inspiron 3647 台式机	Linux CentOS 7 操作系统
Intel I5 4 核处理器 2.9GHz	LLVM-3.4 开发框架
4G DDR3 内存	KDevelop 集成开发环境

表 7.2　实验评估软件信息

软件名	版本号	代码行数	配置项数
Httpd	2.4	148K	564
MySQL	5.7.0	1.2M	671
PostgreSQL	9.2	757K	273
Yum	3.4.3	38K	74

7.1　ConfVD 技术评估

第 3 章中本课题首先提出了配置项分类树，并基于该分类树提取出了配置项对应的约束约束信息。第 5 章中本课题提出了基于逆文本排序对软件日志信息分析并判断能否诊断日志故障。本节对这三部分工作设计相关实验进行评估。

7.1.1　配置分类覆盖率评估

首先，对于表 7.2中的四款软件，本课题通过查看用户手册，开发者文档等方式，记录下了所有配置项及其相关描述信息。对于这些配置项，逐个比对查看他们是否能够被分入图 4.2的配置分类当中（对于分类为“Others”的配置项，本课题认为该配置项不能被配置分类表示）。结果记录在表 7.3中。通过实验评估，本课题发现该配置项分类能够覆盖所有 1582 个配置项当中的 96.5%，对每个软件至少达到了 95.8%。

表 7.3　分类树对现实配置项的覆盖率

软件名称	配置项数量	覆盖率
Httpd	564	543(96.2%)
MySQL	671	643(95.8%)
PostgreSQL	273	270(98.9%)
Yum	74	71(95.9%)
总计	1,582	1,527(96.5%)

7.1.2　配置约束一致性评估

对基于分类推断出的配置约束，本课题也通过类似上述的方法进行评估。对于每个已经得知类型的配置项，本课题通过查看对该配置项描述的方式确定其约束。一部分软件手册拥有详尽的使用说明，例如 httpd 对配置选项定义过程就使用了统一的语法形式。如果本课题观察到的配置项约束形式能够被 ConfVD 推断得到，就认为该配置项约束是符合本文约束推断方法的。在表 7.4中本文展示了符合本文提出的语法约束的配置项比例，可以看到 91% 的配置项的语法约束能够被 ConfVD 的语法约束所表示。

对于配置的语义约束而言，这些约束在推断过程中需要大量的专家知识或者软件源码内容。因此 ConfVD 在语义约束提取的过程中主要依赖用户或者软件开发人员的指导信息，ConfVD 本身提供了一系列的语义约束模板供用户使用，用户也能够根据自己的需要修改或者添加自定义的模板帮助定义配置项的语义约束。因此这部分约束内容无法在本工作中被评估。对于配置格式约束，本文主要基于对于配置文件以及用户手册的观察制定，因此认为格式约束的一致性比例达到了 100%。

表 7.4　语法约束一致性比例

软件名称	配置项数量	语法约束一致性比例
Httpd	564	466(82.6%)
MySQL	671	641(95.5%)
PostgreSQL	273	265(97.0%)
Yum	74	68(91.9%)
总计	1,582	1,440(91.0%)

7.1.3 诊断日志充分性评估

本节选取 Httpd，PostgreSQL，Yum 三款软件对分析诊断日志充分性的工具进行评估，对注入故障日志测试后的共 50 个诊断日志进行评估。如表 7.5所示，本文的诊断日志分析工具检测出三款软件中的诊断日志充分性结果。对于这 50 个诊断日志，共检测出了 14 个不充分的诊断日志和 36 个充分性良好的诊断日志。通过人工验证，在这 50 个日志当中共检查出 2 个误报，因此 ConfVD 的正确率达到了 96%，其中误报率达到了 4%，所有的被工具识别为充分性差的诊断日志也都被人工确认是充分性差的，漏报率为 0%。

表 7.5 诊断日志充分性评估

软件	充分	不充分	误报	漏报
Httpd	18	8	2	0
PostgreSQL	10	3	0	0
Yum	8	3	0	0
总计	36	14	2	0

7.2 软件反应能力评估分析实验结果

本节主要对基于 ConfVD 对目前广泛使用的大型开源软件应对配置故障时的反应能力进行测试和评估得到的实验结果进行分析。实验分析的主要流程为，首先通过 ConfVD 对目标软件的配置推断得到约束信息，再通过违反约束生成配置故障，然后通过将配置故障注入到软件系统当中，并测试得到软件的反应结果，最后对反应进行分析并分类，得到软件反应能力评估的实验结果。实验分析主要包括软件反应能力分析，故障诊断能力分析，ConfVD 能力分析三部分组成。

其中配置故障注入的相关信息如表 7.6所示。由于软件配置空间爆炸，大数量的配置项会使得测试变得不可行，因此本课题通过分层采样的方式，近似模拟配置文件当中各配置项类型所占的比例的方式随机抽取相应类型的配置项，从而使结果更加接近实际水平。本课题通过分层采样抽取了 Httpd 等四款大型开源软件 115 个软件配置项作为样本，共计生成了 1069 个配置故障。对于软件反应，本课题使用表 6.1中使用的分类方法，将软件反应分成六类。整个实验过程基于该分类对软件反应进行分析。

7.2.1 软件反应能力分析

整个软件反应能力的结果如表 7.7所示。下文将分别讨论每种反应类型发生的原因。Type 1（1.03%）和 Type 2（0.37%）这两种反应很少发生，在整个软件测试

表 7.6　配置故障注入信息表

软件名称	配置项数量	生成配置故障数
Httpd	30	236
MySQL	26	255
PostgreSQL	33	334
Yum	26	244

的结果当中只在 MySQL 和 Yum 两款软件中被观察到。Type 1 反应发生的原因是由于软件在对变量进行赋值的过程检测到了配置故障，从而忽略了非法值改为使用默认值。例如在 MySQL 当中如果配置项“table_open_cache”被用户错误设置为一个非法数字，MySQL 会将该配置变量重新赋值，并在软件日志当中输出该信息。

Type 2（0.37%）反应中包含了软件对于配置故障的异常反馈信息，但是软件并未发生失效而且通过了功能测试。例如 Yum 在运行过程中输出了包含不能打开特定文件内容的日志，却没有在任何测试中发生失败。

Type 3（42.28%）反应相对而言比较常见，主要表现为通过了所有测试，没有任何异常迹象。本课题认为该反应的大量发生主要包含两大原因：第一种情况，软件在设计实现过程中加入了鲁棒性设计，因此能够在故障传播前解决问题。举例来说，PostgreSQL 在设计过程中加入了配置文件格式的鲁棒性设计，因此能够容许“key value”和“key = value”两种格式的配置作为输入，避免了配置故障的发生。第二种情况，配置故障作为潜在的配置错误（latent configuration errors [24]）存在于软件系统当中，这些配置项在软件读取配置的过程中未被检查，又或是由于特定的运行逻辑关系根本没有被软件读取。因此本课题采用尽可能丰富的软件测试用例来避免该种情况的发生。

Type 4（41.25%）反应能够清晰地指出配置故障所发生的位置，包括配置文件信息，相应的行数，甚至出错的配置项名。大多数 Type 4 反应发生在软件启动中的配置检查过程。例如 Httpd 在遇到配置故障时，会输出“Syntax error on line 100 of /etc/httpd/conf/httpd.conf”来提醒用户发生配置故障的位置。

Type 5（6.64%）反应发生时触发了软件系统的异常信息，但是这些异常信息并不能帮助定位配置故障。主要是由于配置项并没有被检查，或者是检查过程没有捕捉到这些错误，因此这些配置故障的诊断信息比较模糊，甚至可能会误导用户。举例来说，用户改写 Httpd 配置时误将配置项“Listen 80”重复写了两遍后，软件运行失败会打印日志信息“Could not bind to address”，而这些信息极有可能误导用户的诊断过程。

Type 6（8.42%）反应更为严重地影响了配置故障的诊断。这些反应（例如崩

溃，挂起，等等）是由于不恰当地异常处理代码导致的，同时配置检查的缺乏也会引起这些反应。

表 7.7　软件反应能力分类结果

反应类型缩写.	Yum	Httpd	PostgreSQL	MySQL	总计	占比
Type 1	0	0	0	11	11	1.03%
Type 2	4	0	0	0	4	0.37%
Type 3	127	65	155	105	452	42.28%
Type 4	82	113	176	70	441	41.25%
Type 5	3	8	3	57	71	6.64%
Type 6	28	50	0	12	90	8.42%
总计	244	236	334	255	1069	100%

7.2.2　配置故障诊断能力分析

本文在本节中评估了软件系统应对不同类型的配置故障时产生的反应。如图 7.1所示，本课题分析了广泛使用的三种类型配置项，如路径型（Path），布尔型（Bool）以及计数型（Count）导致的配置故障所引起的软件反应。本文通过定义配置故障诊断率来衡量软件对于这些不同种类配置故障反应能力的优劣。配置故障诊断率即在配置故障引起软件失效之后，软件能够诊断出配置故障原因的比例。具体而言即为 Type 4 类型反应，占总体软件失败的反应（Type 4，Type 5，Type 6 反应）数量之和。

根据结果本课题发现，MySQL 和 Yum 都无法诊断出路径相关的配置故障。相较于路径相关的配置故障而言，布尔类型的配置故障更加容易定位，Yum，Httpd，PostgreSQL 三款软件的诊断率都达到了 70% 以上，而 MySQL 却只有 55.91%。经过观察实验结果，本课题认为发生这种现象的原因是由于 MySQL 仅仅输出了配置故障相关的异常信息，而没有在日志中明确指出配置故障的具体位置。计数类型的配置故障的诊断率比布尔型更高，分别为 87.49%（Httpd），80%（Yum），100%（PostgreSQL），而由于相似的原因，MySQL 只有较低的 15.63%。

对于这三种类型的配置故障而言，路径类型的配置故障是最难被软件系统成功诊断的，即便是最有经验的软件开发者也有可能无法很好地处理这些配置故障。但是，尽管需要开发人员大量的精力，这并非是一件不可能的事。例如 PostgreSQL 在启动过程中就检查了每个配置的语法是否符合要求。有着简单约束的配置项（例如布尔型，计数型等等）的配置故障诊断率比较高，因为验证这些配置项的约束是否规范更加容易一些。因此，本课题强烈建议开发者尽可能地使用简单约束

的配置项，来减少配置故障发生的可能。同时，用户需要更多的提示信息，而不仅仅是软件运行过程当中发生的异常。因此我们建议开发者能够指出发生故障的原因，而不仅仅只是记录下了软件系统发生的行为。

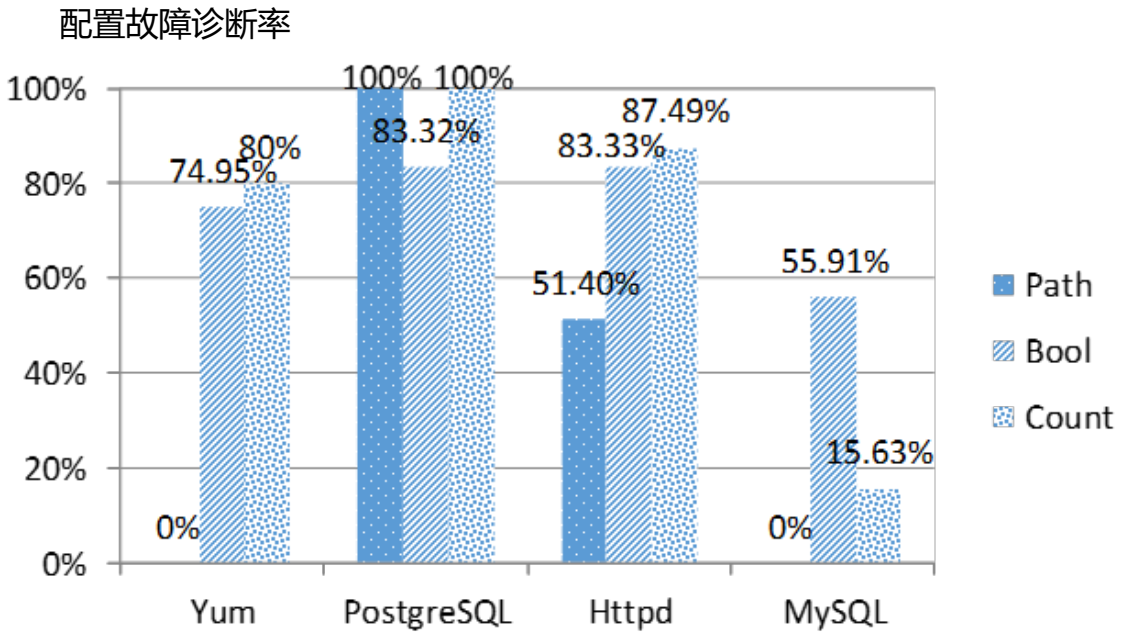

图 7.1　软件对不同类型配置故障的反应情况

7.2.3　ConfVD 能力分析

为了验证本文提出的三大类型的细粒度约束对于配置故障注入以及软件反应能力分析评估帮助的有效性。本节主要从两个方面对 ConfVD 的能力进行对比：基于简单变换生成配置故障的工作，以及基于简单约束生成配置故障的工作。

7.2.3.1　基于简单变换的配置故障产生方法

ConfErr [22] 使用了一种比较直接的方法，通过对正确的配置项进行一些简单的变换，例如字符的替换，遗漏，大小写修改等等来产生简单的配置故障。本课题采用无约束指导的 ConfErr 进行对比，验证约束对配置故障生成的重要性。

这本节中，本课题通过不良反应检出率来评估 ConfVD 和 ConfErr 的能力。不良反应检出率，即不良反应（Type 5 和 Type 6 反应）占注入配置故障的百分比，不良反应检出率越高，说明该工具越容易从注入的配置故障中找到软件的配置脆弱性。表 7.8展示了每款工具发现的软件不良反应占总反应数目的比例。结果显示 ConfVD 的不良反应检出率全面大于 ConfErr，总的不良检出率为 25.05% 接近 ConfErr（8.74%）的三倍。同时结果显示，ConfErr 发现的软件不良反应同时也能

够被 ConfVD 所发现。对该结果进行分析，本课题认为导致 ConfErr 低效的原因，是目前成熟的软件系统能够轻易的检测到软件配置格式的违反情况，但是诊断约束相关的配置故障不仅仅需要领域知识，同时也需要软件系统运行的环境信息，因此 ConfVD 能够发现更多潜在的软件不良反应。

表 7.8　ConfVD 与 ConfErr 的比较实验结果

	不良反应检出率（未能诊断出的配置故障数量 / 注入的配置故障数量）				
	Httpd	MySQL	PostgreSQL	Yum	总计
ConfVD	38/106 (35.85%)	60/169 (35.5%)	3/119 (2.52%)	23/101 (22.78%)	124/495 (25.05%)
ConfErr	30/183 (16.39%)	7/170 (4.12%)	2/105 (1.90%)	18/194 (9.28%)	57/652 (8.74%)

7.2.3.2　对比基于简单约束的配置故障产生方法

而目前最先进的工具 SPEX [23]，使用简单的约束来帮助生成配置故障。但是这些粗粒度的约束，例如数值类型，值范围等等并不能指导生成全面而丰富的配置故障。为了证明细粒度约束的必要性，本课题移除了 ConfVD 中由细粒度约束生成的配置故障创造了一个 ConfVD 的变体，来模拟 SPEX 的工作过程。本课题不使用 SPEX 的原因，一方面由于 SPEX 没有公开源码，另一方面它假设需要用户提供软件源码同时需要加入特定领域知识的注释来帮助运行。对比试验的结果在表 7.9中，结果显示变体只发现了 131 个软件不良反应，比起 ConfVD 的 161 个减少了 18.6%。图 7.2给出了一个例子，Httpd 很轻易地检测到了简单约束生成的配置故障，而对于细粒度产生的配置故障却无能为力。因此本课题研究表明，一个全面的测试对于提升软件质量至关重要，相应的 ConfVD 使用了细粒度约束来生成和注入这些全面的配置故障来测试和评估软件的反应能力。ConfVD 细粒度约束被证明对于 Httpd，MySQL，Yum 都是及其有效的，但是在 PostgreSQL 上的提升并不明显。尽管 81.3% 的软件不良反应只用通过简单约束就能发现，但是粗粒度约束并不总是足够发现所有的软件不良反应，因此本课题认为使用细粒度约束生成配置故障，能够更加全面更加有效地发现影响用户的软件不良反应。

7.3　结果有效性分析

影响本文研究结果有效性的威胁主要分为四种。

- 尽管本文研究的软件都是大型并且成熟的，但是基于这些软件本课题提出的配置项分类可能并不具备代表性。在本课题工作中，为了验证配置项分类的普遍性，避免过拟合情况的发生，用来分类的配置项来自八款开源软件，同时验证采用的软件则是另外四款开源软件。

表 7.9　ConfVD 与变体的比较实验结果

	ConfVD				使用简单约束的 ConfVD 变体			
	Type4	Type5	Type6	未能诊断出的配置故障数量	Type4	Type5	Type6	未能诊断出的配置故障数量
Httpd	113	8	50	58	69	6	44	50
MySQL	70	57	12	69	42	49	8	57
PostgreSQL	176	3	0	3	129	3	0	3
Yum	82	3	28	31	57	3	18	21
总计	441	71	90	161	297	61	70	131

简单约束产生的配置故障（变体）：
DocumentRoot = /path/to/file （正确的配置为目录路径）
日志信息：
Jan 15 19:00:55 bogon httpd[26875]: AH00526: Syntax error on line 24 of /etc/httpd/conf/httpd.conf:
Jan 15 19:00:55 bogon httpd[26875]: DocumentRoot must be a directory

细粒度约束产生的配置故障（ConfVD）：
DocumentRoot = \/var/www/html （不正确的路径格式）
日志信息：
Jan 15 19:00:23 bogon systemd[26875]: Failed to start The Apache HTTP Server.
Jan 15 19:00:23 bogon systemd[26875]: Unit httpd.service entered failed state.

图 7.2　不同约束产生的配置故障对软件反应的影响

- 在生成配置故障分析软件反应的过程中，由于计算能力及时间开销的限制，本课题只考虑了一部分软件配置项。因此可能存在一些配置项导致该实验结果产生偏差。但是所有本课题选取的配置项都是被用户经常使用的，因此本课题认为这些配置项在大多数情况下是具有代表性的。

- 所有本课题分析的结果都来自记录的软件系统的日志信息，在对这些信息进行研究的时候人工检查可能会引入错误，因此本课题多次检查避免人为错误引入的可能。

- 本课题可能没能列举出所有的配置故障。在未来的工作中，本课题打算进一步分析更多的来自开源和商用软件的用户错误报告进一步提出更多更丰富的配置故障生成方法。

7.4 实践经验总结

在本节中，本课题主要讨论在评估和实验分析过程中观察到的经验总结和配置实现的相关建议。

- 避免不一致性：在实验分析的过程中，本课题观察到了一些软件开发者在处理软件配置时的良好行为，例如 PostgreSQL 的配置当中，数字通常情况下会伴随着单位（例如：max_stack_depth = 100kB），因此用户在改写软件配置的过程当中就不会产生单位问题的困惑。与之相反的，如果配置文件当中缺少了单位后缀或者相应的解释，用户就可能因为单位混乱的问题而产生配置故障。

- 让配置简单易懂：用户友好的配置设计同样能够大大降低配置故障产生的可能。在本文的研究当中，本课题发现了许许多多软件文档当中模糊难懂的配置项的相关描述。Xu 等人 [52] 发现和指出了关于简化配置设计的指导意见。他们同时也研究了一种名为“Configuration Navigation”的技术来帮助用户来理解配置。

- 提前检查配置故障：在检查实验评估使用的软件系统源码之后，本课题发现 PostgreSQL 使用一致的接口来对配置文件当中的内容进行处理。在这个处理过程中，每个配置项值都会进行检查，同时如果发现存在异常，PostgreSQL 便会打印相应的异常信息。提前检查能够帮助用户在用户启动过程中提前高效地检查出存在的配置故障。

- 友好的注释及文档：在软件配置当中，如 PostgreSQL 使用注释的方式明确地告诉了用户每个配置项的使用方法，例如“#1s-600s”，“#defaults to 'localhost'；use '*' for all”。这样的注释或者用户手册当中对于配置的解释能够帮助用户更好地正确配置所需要的配置选项。

第八章　结束语

本章主要回顾了本课题的主要工作，总结了基于配置故障注入的软件反应能力测试和评估研究，分析了目前工作的成果及存在的不足。最后针对研究工作的不足并结合目前配置故障研究的发展趋势，对课题下一步工作进行了展望。

8.1　工作总结

随着软件规模的不断扩大，软件的配置也变得更加复杂。配置故障已经成为导致软件失效的主要原因之一。然而目前研究工作主要集中在配置故障的自动诊断上，对于软件应对配置故障反应能力的研究仍很欠缺。本课题针对目前研究的不足，对大型开源软件配置进行了大规模调研分析，提出覆盖面全的软件配置项类型分类方法，设计实现了基于配置故障注入的软件反应能力的测试和评估框架ConfVD。ConfVD 通过推断软件配置约束生成配置故障，并注入配置故障到软件系统当中观察分析软件反应能力。全文的主要工作分为以下几个方面：

- 系统分析了目前广泛应用的八款开源软件 Squid、Nginx、Redis、Nagios、Lighttpd (core)、Puppet、SeaFile 和 Vsftpd，总结出了覆盖率达到 96.5% 软件系统配置项分类。基于该配置项分类树，提出了配置值类型约束和文件格式约束，经验证，本课题提出的配置约束条件符合评估使用的软件配置总数的 91%。

- 基于本文提出的配置约束，ConfVD 实现了配置故障的自动生成。同时，通过使用 ABNF 范式表示配置约束，ConfVD 能够根据细粒度的约束产生全面的配置故障，从而得到更丰富的注入结果。对于已经生成的配置故障，通过对软件配置的抽象表示，实现了配置故障对不同软件系统的自动化注入。

- ConfVD 实现了对配置故障的自动测试框架，并根据诊断配置故障的能力将软件反应分成了六大类，同时基于自然语言处理的相关技术实现了软件反应的自动分类。通过分析软件反应能力的分布结果，本课题发现 15% 的软件配置故障引起了软件的不良反应导致配置故障的诊断困难，而其中路径类型的软件配置故障是最难被软件系统诊断的。根据实验结果，本课题还总结了软件配置设计实现过程的实践经验。

对比前人工作，由于拥有约束作为指导，ConfVD 能够发现三倍于 ConfErr 的软件配置脆弱性。同时对于粗粒度约束而言，ConfVD 使用细粒度约束生成的配置故障能够发现 18.6% 更多的软件配置故障引起的不良反应，使得充分暴露软件应对配置故障时的脆弱性成为可能。

8.2 研究展望

针对本研究目前还存在的不足以及结合目前相关研究的发展趋势，本课题打算在如下方面开展进一步的研究：

- 针对软件配置项分类，接下来会加大调研样本的数量，从而更加丰富目前的分类树。同时对于配置约束的描述，也有进一步改善的空间。
- 在软件配置故障生成方面，由于配置故障样本的缺乏，对配置故障的模拟工作一直是个难点。在接下来的工作中，可以广泛调研目前存在的常见的配置故障，从而进一步丰富配置故障种类。
- 在软件反应自动分析方面，软件自带的测试用例并不能很好的暴露软件应对配置故障时的脆弱性。因此接下来将评估软件测试用例能否真正帮助配置故障测试，从而达到更好的测试效果。
- 在 ConfVD 对于软件反应能力评估结果的基础上，本课题能够发现软件在应对配置故障过程中存在的不足。因此在接下来的工作中，本课题希望能够分析和提炼出目前软件对于配置故障的行为模式，通过对于这些行为加以学习，实现软件反应能力的自动增强，从而更好地提升软件应对配置故障的可靠性问题。

致　谢

在此硕士学位论文即将完成之际，回顾过去两年半时间的硕士阶段的学习和生活，我自身感慨良多。有付出，有收获，有成功的喜悦，也有失败时的懊悔。最重要的是，我在大家的陪伴下一路克服艰难走来，取得的所有成绩离不开身边的人们。感谢你们!

衷心感谢我的导师廖湘科老师。廖老师学术目光长远敏锐、治学态度严谨。身为多个领域的专家学者和学科带头人，他用丰富的知识与经验指引着我走进学术殿堂门，培养了我独立思考、自信严谨的学术态度。廖老师准确而敏锐地把握了软件可靠性的最新技术及发展趋势，高屋建瓴，为我们指明了研究方向。在百忙之中还会抽时间关心课题组的学术研究进展，指导我们做研究方法。非常感谢廖老师一直以来的关心和指导。

真诚感谢李姗姗老师，在学习和生活中带给了我很多帮助，让我更快地转变到硕士阶段中来。李姗姗老师具体地指导我做课题研究，从明确研究方向，如何进行调研工作，到复现已有工作查找问题提出自主解决方案，最终到硕士课题的完成。一步一个脚印，让我感受到了科研真正的魅力所在。中间也并不总是一帆风顺，有过观点不一致激烈的讨论，也有为赶进度加班加点通宵写论文，一次次挫折和磨炼都在让我不断进步。感谢李老师给我的关心和帮助。

感谢刘晓东师兄，贾周阳师兄，周书林师兄，这三位师兄在系统、编译领域都有着非常丰富的经验，分析和解决实际问题思路清晰，认识深刻。在课题组每次的讨论都能引发我新的想法，新的思路，我的成绩也归功于他们。感谢同级徐向阳同学，我们合作克服了许多困难，完成了很多科研攻关工作，一路上相互扶持不断努力进步。感谢小组内的师弟师妹们，和你们在一起学习研究我收获很多。

感谢师门师兄弟姐妹们，大家一起构建了良好的科研和生活环境，让我无时不刻不能感受到师门大家庭的温暖。

感谢一连 11 级和五队 15 级的同学们，是你们陪伴我度过了 7 年科大的生活，陪我在这里成长，留下了无数美好的回忆。

感谢学员队领导胡浩政委，蔡巍政委，宋浩队长。整个学员队在你们的领导下纪律严明，气氛和谐，为大家创造了良好的学习和生活环境。

感谢我的父亲郦学良和我的母亲王芬芳，你们一直是我成长道路上动力的源泉。每当我失去信心，是你们一次次让我鼓起勇气，勇敢面对，克服挫折。

感谢我的女朋友郭玓，在我研究生最难熬的时候陪伴着我，给我指明了前进的方向。

感谢所有在我成长记进步道路上提供帮助和支持我的人，谢谢你们!

参考文献

[1] The Coverity Report. http://www.coverity.com. 2017.

[2] Gray J. Why do computers stop and what can be done about them [J]. Technical Report TR-85.7. 1985, 30 (4): 88–94.

[3] Nagaraja K, Oliveira F, Bianchini R, et al. Understanding and dealing with operator mistakes in internet services [C]. In Conference on Symposium on Opearting Systems Design and Implementation. 2004: 61–76.

[4] Oppenheimer D, Ganapathi A, Patterson D A. Why Do Internet Services Fail, and What Can Be Done About It? [C]. In Conference on Usenix Symposium on Internet Technologies and Systems. 2003: 1–1.

[5] Barroso L, Clidaras J, Hoelzle U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines [J]. 2009, 8 (3): 154.

[6] Yin Z, Ma X, Zheng J, et al. An empirical study on configuration errors in commercial and open source systems [C]. In ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October. 2011: 159–172.

[7] Sverdlik Y. Microsoft: misconfigured network device led to azure outage. Retrieved January 29 from http://www.datacenterdynamics.com/focus/archive/2012/07/microsoft-misconfigured-network-device-led-azure-outage. 2012.

[8] Team A. Summary of the amazon ec2 and amazon rds service disruption in the us east region. Retrieved January 29 from http://aws. amazon. com/message/65648. 2011.

[9] Robert J. More details on today's outage. www.facebook.com. Retrieved January 29, 2017, from https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919. 2010.

[10] Kapoor A. Web-to-host: Reducing total cost of ownership [R]. 2000.

[11] Rabkin A, Katz R. Precomputing possible configuration error diagnoses [C]. In Ieee/acm International Conference on Automated Software Engineering. 2011: 193–202.

[12] Attariyan M, Flinn J. Automating configuration troubleshooting with dynamic information flow analysis [C]. In Usenix Conference on Operating Systems Design and Implementation. 2010: 1–11.

[13] Attariyan M, Chow M, Flinn J. X-ray: automating root-cause diagnosis of performance anomalies in production software [C]. In Usenix Conference on Operating Systems Design and Implementation. 2012: 307–320.

[14] Yuan D, Xie Y, Panigrahy R, et al. Context-based online configuration-error detection [C]. In Usenix Conference on Usenix Technical Conference. 2011: 28–28.

[15] Zhang J, Renganarayana L, Zhang X, et al. EnCore: exploiting system environment and correlation information for misconfiguration detection [C]. In International Conference on Architectural Support for Programming Languages and Operating Systems. 2014: 687–700.

[16] Mickens J, Szummer M, Narayanan D. Snitch: interactive decision trees for troubleshooting misconfigurations [C]. In Usenix Workshop on Tackling Computer Systems Problems with Machine Learning Techniques. 2007: 8.

[17] Wang Y M, Verbowskia C, Dunagana J, et al. STRIDER: A Blackbox, State-based Approach to Change and Configuration Management and Support [C]. In Usenix Conference on System Administration. 2003: 159–172.

[18] Wang H J, Platt J C, Chen Y, et al. Automatic misconfiguration troubleshooting with peerpressure [C]. In Conference on Symposium on Opearting Systems Design and Implementation. 2004: 17–17.

[19] Attariyan M, Flinn J. Using Causality to Diagnose Configuration Bugs. [C]. In Usenix Technical Conference, Boston, Ma, Usa, June 22-27, 2008. Proceedings. 2008: 281–286.

[20] Whitaker A, Cox R S, Gribble S D. Configuration Debugging as Search: Finding the Needle in the Haystack [C]. In Conference on Symposium on Operating Systems Design and Implementation. 2004: 77–90.

[21] Tucek J, Lu S, Huang C, et al. Triage: diagnosing production run failures at the user's site [C]. In ACM Sigops Symposium on Operating Systems Principles. 2007: 131–144.

[22] Keller L, Upadhyaya P, Candea G. ConfErr: A tool for assessing resilience to human configuration errors [C]. In IEEE International Conference on Dependable Systems and Networks with Ftcs and DCC. 2008: 157–166.

[23] Xu T, Zhang J, Huang P, et al. Do not blame users for misconfigurations [C]. In Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013: 244–259.

[24] Xu T, Jin X, Huang P, et al. Early Detection of Configuration Errors to Reduce Failure Damage [J]. 2016.

[25] Xu T, Zhou Y. Systems Approaches to Tackling Configuration Errors: A Survey [J]. Acm Computing Surveys. 2015, 47 (4): 1–41.

[26] Jiang W, Li Z, Li Z, et al. Understanding customer problem troubleshooting from storage system logs [C]. In Proccedings of the Conference on File and Storage Technologies. 2009: 43–56.

[27] Alexandrov A, Bergmann R, Ewen S, et al. The Stratosphere platform for big data analytics [J]. Vldb Journal. 2014, 23 (6): 939–964.

[28] Xi B, Liu Z, Raghavachari M, et al. A smart hill-climbing algorithm for application server configuration [C]. In Int. Conf. on WWW. 2004: 287–296.

[29] Osogami T, Itoko T. Finding probably better system configurations quickly [C]. In Joint International Conference on Measurement and Modeling of Computer Systems. 2006: 264–275.

[30] Velasquez N F, Weisband S, Durcikova A. Designing Tools for System Administrators: An Empirical Test of the Integrated User Satisfaction Model [C]. In Large Installation System Administration Conference, LISA 2008, November 9-14, 2008, San Diego, Ca, Usa. 2008: 1–8.

[31] Barrett R, Kandogan E, Maglio P P, et al. Field studies of computer system administrators:analysis of system management tools and practices [C]. In CSCW ’ 04: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work. 2004: 388–395.

[32] Detreville J. Making system configuration more declarative [C]. In Conference on Hot Topics in Operating Systems. 2005: 11–11.

[33] Norman D A. Design principles for human-computer interfaces [C]. In Sigchi Conference on Human Factors in Computing Systems. 1983: 1–10.

[34] Molich R, Nielsen J. Improving a human-computer dialogue [J]. Communications of the Acm. 1990, 33 (33): 338–348.

[35] Kc E, Wang Y M. Discovering Correctness Constraints for Self-Management of System Configuration [C]. In International Conference on Autonomic Computing. 2004: 28–35.

[36] Bauer L, Garriss S, Reiter M K. Detecting and resolving policy misconfigurations in access-control systems [J]. Acm Transactions on Information and System Security. 2011, 14 (1): 1–28.

[37] Anderson P. Towards a High-Level Machine Configuration System [C]. In Usenix Conference on System Administration. 1994: 19–26.

[38] Burgess M. A Site Configuration Engine [J]. Computing Systems. 1995, 8 (3): 309–337.

[39] Lefebvre W, Snyder D. Auto-configuration by File Construction: Configuration Management with newfig [C]. In Usenix Conference on System Administration. 2004: 93–104.

[40] Goldsack P, Guijarro J, Loughran S, et al. The SmartFrog configuration management framework [J]. Acm Sigops Operating Systems Review. 2009, 43 (1): 16–25.

[41] Verbowski C, Lee J, Liu X, et al. LiveOps: systems management as a service [C]. In Conference on Large Installation System Administration. 2006: 15–15.

[42] Google. http://arstechnica.com/information-technology/2012/12/why-gmail-went-down-google-misconfigured-ch-romes-sync-server/. 2017.

[43] GoogleCluster Management. https://www.youtube.com/watch?feature=player_detailpage&v=0ZFMlO98Jkct=1674s. 2017.

[44] Armbrust, Michael, Fox, et al. Above the Clouds: A Berkeley View of Cloud Computing [J]. Eecs Department University of California Berkeley. 2009, 53 (4): 50–58.

[45] Andreessen M. Why Software Is Eating The World [J]. Wall Street Journal, http://online.wsj.com/article/ SB10001424053111903480904576512250915629460.html. 2011.

[46] Rabkin A, Katz R. Static extraction of program configuration options [C]. In International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , Hi, Usa, May. 2011: 131–140.

[47] RFC2234. http://www.ietf.org/rfc/rfc2234.txt. 2017.

[48] Augeas. http://augeas.net. 2017.

[49] The MySQL Test Framework. https://dev.mysql.com/doc/mysqltest/2.0/en/. 2017.

[50] Zhang S, Ernst M D. Proactive detection of inadequate diagnostic messages for software configuration errors [J]. 2015.

[51] WordNet. http://wordnet.princeton.edu. 2017.

[52] Xu T, Jin L, Fan X, et al. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software [C]. In Joint Meeting on Foundations of Software Engineering. 2015: 307–319.

作者在学期间取得的学术成果

发表的学术论文

[1] Li W, Li S, Liao X, Xu X, Zhou S, Jia Z. ConfTest: Generating Comprehensive Misconfiguration for System Reaction Ability Evaluation[C]. The Evaluation and Assessment in Software Engineering Conference. 2017:88-97.(DOI: 10.1145/3084226.3084244)

[2] Xu X, Li S, Guo Y, Dong W, Li W, Liao X. Automatic Type Inference for Proactive Misconfiguration Prevention[C]. The, International Conference on Software Engineering and Knowledge Engineering. 2017:295-300.(DOI:10.18293/SEKE2017-072)

其他成果

[1] 廖湘科，李姗姗，贾周阳，刘晓东，董威，林彬，周书林，徐向阳，郦旺. 一种基于程序分析的软件日志行为自动识别方法. (受理号：201611016393.)