软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(1):63−86 [doi: 10.13328/j.cnki.jos.006835]

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

软件运行时配置研究综述*

周书林, 李姗姗, 董 威, 王 戟, 廖湘科

(国防科技大学 计算机学院, 湖南 长沙 410073)

通信作者: 李姗姗, E-mail: shanshanli@nudt.edu.cn

摘 要: 运行时配置为用户使用软件提供了灵活性和可定制性, 但其巨大的规模和复杂的机制也带来了巨大的挑战. 大量学者和研究机构针对软件运行时配置展开了研究, 以提升软件系统在复杂外部环境中的可用性和适应性. 建立运行时配置研究分析框架, 从配置分析与理解、配置缺陷检测与故障诊断、配置应用 3 个阶段对现有研究工作进行归类和分析, 总结归纳现有研究的不足和面临的挑战, 并对未来的研究趋势进行展望, 对下一步研究具有一定的指导意义.

关键词: 运行时配置; 配置理解; 配置缺陷检测; 配置故障诊断; 配置应用

中图法分类号: TP311

中文引用格式: 周书林, 李姗姗, 董威, 王戟, 廖湘科. 软件运行时配置研究综述. 软件学报, 2024, 35(1): 63–86. http://www.jos.org.cn/1000-9825/6835.htm

英文引用格式: Zhou SL, Li SS, Dong W, Wang J, Liao XK. Survey on Software Runtime Configuration Researches. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 63–86 (in Chinese). http://www.jos.org.cn/1000-9825/6835.htm

Survey on Software Runtime Configuration Researches

ZHOU Shu-Lin, LI Shan-Shan, DONG Wei, WANG Ji, LIAO Xiang-Ke

(College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Runtime configuration brings flexibility and customizability to users in the utilization of software systems. However, its enormous scale and complex mechanisms also pose significant challenges. A large number of scholars and research institutions have probed into runtime configuration to improve the availability and adaptability of software systems in complex environments. This study develops an analytical framework of runtime configuration to provide a systematic overview of state-of-the-art research from three different stages, namely configuration analysis and comprehension, configuration defect detection and misconfiguration diagnosis, and configuration utilization. The study also summarizes the limitations and challenges faced by current research and outlines the research trend of runtime configuration, which is of guiding significance for future work.

Key words: runtime configuration; configuration comprehension; configuration defect detection; misconfiguration diagnosis; configuration utilization

随着信息化社会的不断发展和计算能力的不断提升, 软件系统在社会和国防各个领域发挥着越来越重要的作用, 呈现出网络化、智能化的趋势, 引领社会发展逐步进入“软件定义”的时代. 伴随着软件系统的复杂化和规模化, 配置作为软件系统的重要组成部分, 在完善软件功能多样性、提升系统灵活性和可定制性方面发挥着重要作用. 软件系统逐步向高可配置化方向发展, 从而适应复杂的外部环境, 提升软件服务的能力和效率.

然而, 随着软件规模的持续增长和软件间交互关系的日益复杂, 软件配置的灵活性, 特别是面向用户的运行时配置在带来便利的同时, 也容易引发各种问题, 导致严重后果的案例屡见不鲜. 一方面, 不正确的配置设置导致软

* 基金项目: 国家自然科学基金 (61872373, 62272473, 61872375, 62032019)
收稿时间: 2022-03-07; 修改时间: 2022-07-31; 采用时间: 2022-11-20; jos 在线出版时间: 2023-07-04
CNKI 网络首发时间: 2023-07-05

件出现故障, 影响软件系统的正常运行. Barroso 等人的研究[1]结果表明, 配置故障已成为导致 Google 产品服务失效的第二大原因, 占比达到 29%. 华盛顿大学的 Katz 研究组调查发现[2], 从客户案例数量及技术支持时长两方面评估, 配置故障均已是导致 Hadoop 集群失效的最主要原因. 相似的案例同样发生在 Amazon、Salesforce 等大型互联网公司, 导致严重的全球服务宕机事件, 造成了巨大的商业损失[2−6]. 另一方面, 不合适的配置设置会对软件性能等非功能目标产生巨大影响, 影响软件的服务质量, 造成大量经济损失. 例如, Yin 等人[7]发现 MySQL 中 20% 的配置故障导致了严重的性能衰减, 文献 [8] 也指出仅通过调整文件系统配置设置即可使性能表现差距达到 9 倍以上.

导致软件运行时配置相关问题日益严重的因素是多方面的. 首先, 软件的功能不断增强, 配置在带来灵活性的同时, 对应的复杂度也不断上升. 配置参数规模庞大, 类型复杂多样, 且需要满足多种约束条件, 导致软件用户难以理解配置的运行机理和相关代码模式. 其次, 导致配置故障的原因复杂, 用户主观错误和配置相关代码缺陷均可能导致软件出现故障或者功能/非功能目标失效. 最后, 文档信息的不规范以及软件失效之后反馈信息的缺乏都会导致诊断困难. 目前, 已存在大量研究工作针对配置相关问题展开研究, 涉及配置的理解与使用、配置故障诊断等各个方面. 但目前仅存在少量工作针对配置故障诊断[9,10]和配置开发工程实践[11]相关研究进行了归纳综述, 而配置相关的其他研究领域研究现状仍缺乏系统性的分析和理解.

此外, 随着软件系统的复杂化和规模化, 其软件基础设施面临环境和资源不断变迁的挑战, 软件系统也需要持续演化以提升适应能力. 运行时配置作为软件对外交互的重要接口, 可以管理软件的行为和资源的使用, 从而快速适应不同的环境和负载, 满足不同的用户需求. 然而, 如何调整配置以保证软件适应外部环境, 以及在调整过程中如何保证软件质量, 现有研究仍主要处于起步阶段.

基于上述现状, 本文拟针对软件运行时配置的研究现状和进展进行综述. 首先, 针对现有配置相关概念进行梳理和定义, 确定本文的研究目标和调研对象. 然后, 从如何保障用户正确高效使用配置出发, 提出了运行时配置研究分析框架, 从配置分析与理解、配置缺陷检测与故障诊断、配置应用 3 个阶段对现有研究工作进行系统性的梳理和综述, 着重关注不同阶段已有工作的进展与不足. 最后, 基于现有研究的局限性和面临的挑战, 对运行时配置的未来研究方向进行展望.

1 概 述

软件配置 (software configuration) 是软件系统不可或缺的组成部分, 广泛存在于软件部署、运行、升级以及迁移等应用场景中, 主要指通过特定接口或者文件对软件系统调整配置参数的取值. 软件配置又分为运行时配置 (runtime configuration) 和编译时配置 (compile-time configuration). 运行时配置是软件的重要接口, 可以定制不同的功能、管理资源的分配、适应环境的变化以及满足不同用户的需求, 主要面向软件管理员和用户, 无需重新编译部署即可实现软件调整. 而编译时配置也称软件生产线配置 (software product line configuration, SPL configuration), 主要用于软件构建和部署时期决定特定功能模块是否加入可执行程序中, 以此决定软件的最终形态, 主要面向软件开发和部署人员, 必须重新编译才能使用, 一定程度上限制了配置在实际生产环境中的适应能力. 本文主要关注软件的运行时配置.

目前, 部分研究组已针对软件配置相关研究工作进行了综述研究[9−11]. Xu 等人[9]对已有解决配置故障的研究系统方法进行了整体化和结构化的概述, 从面向软件配置故障的预防、检测与诊断出发, 总结归纳了现有研究的进展和不足. Chen 等人[10]同样针对软件配置故障诊断与修复技术进行综述, 提出了一种涵盖了方法类型、方式和适用范围的分析框架对该领域的主要研究工作进行了分类总结和分析评价. Sayagh 等人[11]尝试从工程实践的角度出发研究软件运行时配置相关的问题, 并基于人工调查问卷总结归纳的主要工程实践活动和相关挑战, 对现有研究文献进行了综述, 探索不同工程实践活动中现有研究的进展和未来的挑战. 总结来看, Xu 等人[9]和 Chen 等人[10]的研究工作主要偏向于软件配置故障的解决, 而 Sayagh 等人[11]更偏向于针对软件开发过程. 而且, 上述综述工作中最近完成的也已经是 4 年前, 缺乏对最新研究进展的梳理. 因此, 本文从用户使用软件配置的需求角度出发, 对近年来软件运行时配置不同方面的相关研究工作展开了系统性的综述和分析.

本文以“config”为搜索关键词, 在 ACM Digital Library、Springer Link Online Library、IEEE Xplore Digital Library、Elsevier ScienceDirect 及 CNKI 等在线数据库中全面搜索了近 10 年来 (2011 年 1 月至 2022 年 5 月间) 在期刊和会议中, 特别是在 CCF 推荐列表中的顶级期刊和会议发表的关于软件运行时配置的相关研究工作, 并对检索得到的文献集合的相关工作、参考文献进行进一步分析, 寻找找到与本文主题密切相关且在检索中未发现的相关成果; 然后, 两名研究人员通过人工阅读文献内容并讨论, 从而共同判断对应文献是否为软件运行时配置相关的研究工作. 最终, 基于上述筛选流程, 本文共得到 122 篇相关文献. 基于 CCF 推荐学术会议和期刊目录, 本文进一步对 122 篇文献所属的研究领域进行了归纳, 形成了如图 1 所示的统计信息. 具体来说, 现有关注软件运行时配置的研究人员重点集中于软件工程/系统软件领域 (30 个相关会议或期刊), 特别是传统的软件工程领域会议和期刊, 如 ICSE (14 篇)、ASE (16 篇)、FSE (10 篇)、TSE (6 篇) 等. 部分关注计算机体系结构/并行与分布计算/存储系统领域的研究人员同样对软件运行时配置相关问题进行了系列研究, 相关会议或期刊数量也达到了 13 个. 此外, 其他研究领域虽然涉及文献数量不多, 但相关文献均是所述领域内的顶级期刊或会议, 如 SIGMOD、VLDB 等. 上述统计信息充分说明了软件运行时配置研究的重要性.

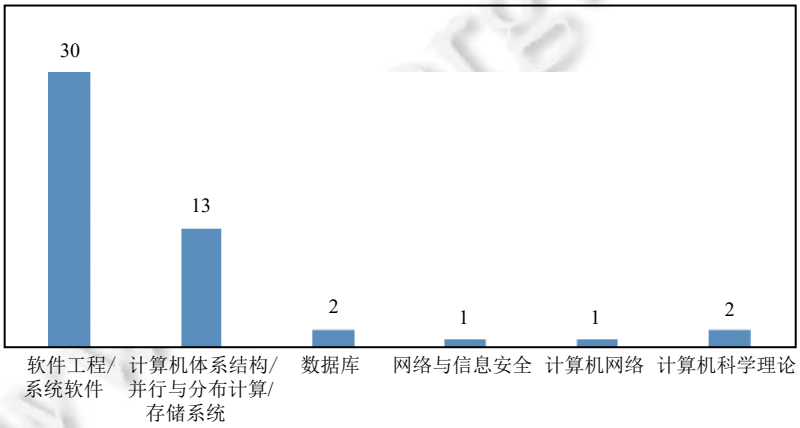

图 1 相关文献涉及研究领域内期刊/会议数量统计

2 运行时配置研究分析框架

从软件配置设计和使用的角度看, 理解配置的运行机理和配置规范是基础, 既可以帮助开发人员完善配置设计, 提升配置相关代码的可用性和可维护性, 也可以保障用户使用配置的正确性. 在此基础上, 为确保软件配置按照设计初衷正常工作, 需要检测配置相关代码缺陷以预防配置故障, 同时在配置故障发生后能快速诊断并修复错误; 在保证配置功能正确的情况下, 基于自动化的方法简化配置使用难度可以极大提升用户使用配置的效率.

正确理解配置是设计和使用配置的基础. 配置理解可以帮助软件开发人员理解配置的运行机理和相关代码模式, 从而优化软件系统的配置设计, 从根源上降低配置的使用难度, 提升配置的使用效率. 对于用户而言, 为实现软件功能的定制, 用户需要明确可用的配置参数信息, 深入理解不同类型配置参数所需满足的配置约束规范, 从而保证配置设置的正确性, 避免因相关知识缺乏导致的配置故障.

配置缺陷检测与故障诊断是保障用户正常使用配置的前提. 大规模软件系统代码中通常存在缺陷, 配置相关的代码实现同样在所难免. 因此, 即使用户正确理解了配置, 并提供了满足复杂约束的配置取值, 软件系统仍可能因配置相关代码缺陷产生故障, 影响正常运行. 因此, 通过事故前的配置缺陷检测有助于提前暴露代码缺陷, 提升代码质量, 有助于预防配置故障. 此外, 当软件因用户主观错误导致配置故障发生后, 准确高效的配置故障诊断与修复方法有助于及时恢复软件的正常运行, 减少故障造成的损失.

配置应用相关研究是用户高效使用配置的重要保障. 基于配置理解、配置缺陷检测与故障诊断的基础, 可以帮助用户深入理解配置运行机理, 并保证软件配置功能按照设计的初衷正确工作. 然而, 配置参数的庞大规模和复

杂功能同样对用户高效使用配置造成了巨大挑战. 为提升用户使用体验, 提高配置使用效率, 深入研究如何实现自动高效的配置应用方法, 满足用户功能和性能需求, 适应复杂外部环境, 具有重大意义.

基于此, 提出了如图 2 所示的运行时配置研究分析框架, 通过针对配置分析与理解、配置缺陷检测与故障诊断、配置应用 3 个阶段已有工作的综述与分析, 了解现有研究的进展与不足, 总结归纳面临的挑战与突破的机遇, 为进一步的研究方向提供指导. 本文后续将按照图 2 的研究分析框架对现有研究工作进行综述, 其中第 3 节介绍配置分析与理解研究进展, 第 4 节关注配置缺陷检测与故障诊断相关工作, 第 5 节则概括配置应用研究领域的现状.

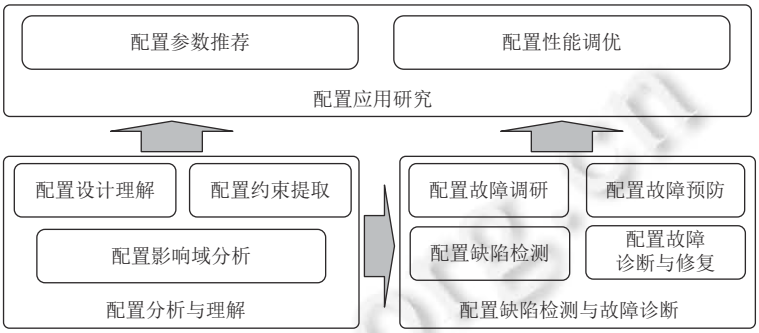

图 2 运行时配置研究分析框架

具体来说, 对于概述中收集的 122 篇相关文献而言, 其在图 2 中 3 个主要配置阶段的文献数量分布如图 3 所示. 其中“其他”类别表示本文收集到的已有综述类型文献, 其余研究方向中配置性能调优、配置故障诊断与修复, 以及配置约束提取 3 个研究方向为主要研究方向, 占比达到了 63% 左右. 具体分析来看, 配置约束提取主要面向用户使用, 是用户接触和使用软件配置的重要基础, 因此占据了较高的研究比重. 而配置故障诊断与修复则由于重点涉及影响了软件的可靠性与可用性, 受到了从业者和研究人员的广泛关注. 至于配置性能调优问题, 由于分布式、云计算等技术的不断发展, 不同配置下软件性能的巨大差异严重影响了服务提供商的经济效益, 受到各界的广泛关注, 因此涉及配置性能调优的研究占比最大.

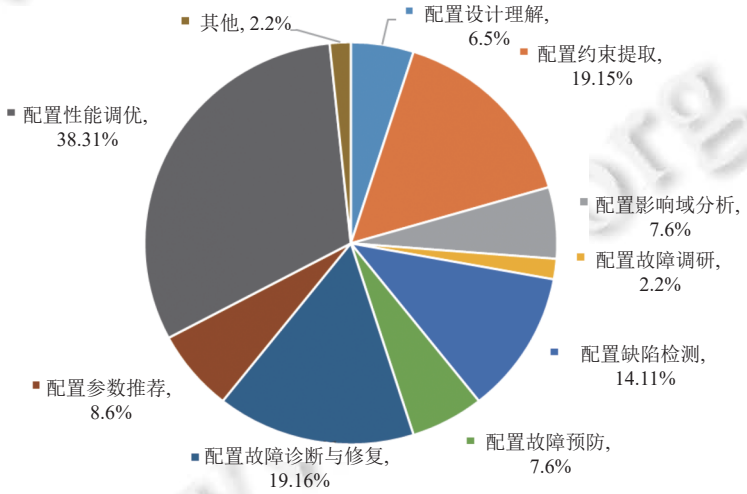

图 3 不同类别中相关文献数量及比例

3 配置分析与理解相关研究

作为用户正确使用配置的基础, 配置分析与理解可以帮助用户理解配置相关的设计原则、代码模式、所需满

足的约束条件等相关知识, 从而深入理解配置的运行机理, 明确配置参数取值的合法范围, 指导用户正确使用配置调控软件系统. 从研究目标来看, 现有研究工作主要分为 3 类, 包括配置设计理解、配置约束提取和配置影响域分析.

3.1 配置设计理解

配置设计理解相关研究工作[11−16], 主要通过调研分析挖掘软件配置在源码中的代码模式, 理解配置的运行机理, 从而提升配置的整体设计质量. Xu 等人[12]聚焦于配置参数设置相关的设计问题进行量化研究, 基于商业软件 Storage-A 和开源软件 Httpd、MySQL 中的 620 份用户配置故障报告, 分析了配置参数取值规模和取值设计对于用户的影响, 从而理解真实用户使用配置的特点, 如超过 54.1% 的配置参数并不会被用户使用, 47.4% 的数值型配置参数用户取值不超过 5 个等, 并提出了具体可行的指导建议用于简化配置设计, 通过减少不必要的配置参数和缩小配置参数取值空间, 降低用户使用的难度. 文献 [14] 关注 Java 软件源码中实现配置管理和使用的配置框架选择问题, 通过针对 GitHub 上 1 938 个 Java 项目进行调研, 总结其中 11 款常用配置框架的特性, 及对于软件配置相关开发和维护开销的影响, 为开发人员选择合适的配置框架提供了建议. 调研结果显示, 更基本的框架应用范围更广, 但仍需要功能更完备的框架进行补充; 同时, 更活跃的配置框架往往功能更完备, 文档更详细, 但需要的维护开销也更多. Zhang 等人[15]调研了 4 款常用开源云计算软件中配置相关代码的演化历史, 深入理解配置设计与实现过程中开发者相关决策的基本原理和具体行为, 特别是为应对配置故障做出的相关行为, 从不同维度细分演化类别, 总结了 16 种配置演化模式和 10 条发现, 例如超过 50% 的常量配置化 (即将程序中的常量取值修改为配置参数) 均是由于常量值在特定运行环境下造成了严重后果而驱动的, 超过 50% 的新增配置检查代码通常仅为基础性检查等. 文献 [13] 基于多种产品的真实需求, 总结归纳了 Facebook 互联网服务在配置管理方面的挑战, 并通过真实案例详细描述了 Facebook 配置管理栈中对应的解决方案. Sayagh 等人[11]通过人工调查问卷与文献综述相结合的方法, 研究软件运行时配置相关的工程实践活动需要关注的问题, 总结归纳了 9 个主要工程实践活动、22 个相关技术及管理挑战, 并提出了 24 条建议用于提升软件配置质量, 以 Configuration As Code 为主线, 涉及如何组织配置、如何新增配置、相关文档如何维护等多方面详细内容. 进一步, 为了解决上述挑战中的 13 个技术挑战, Sayagh 等人[16]经验性的总结了 4 类需求, 形成了 10 个典型配置工程实践任务, 并实现了一个配置框架原型系统 Config2Code, 用于辅助从业者完成相关配置工程任务.

3.2 配置约束提取

配置约束提取是配置分析与理解的主要研究目标, 已有大量研究工作[17−35]关注从不同信息来源运用多样技术手段提取配置相关约束信息. 部分研究工作[17−23]基于程序分析技术从软件源码中提取配置约束. Xu 等人[17]深入分析了配置参数在软件源码中的使用形式, 总结归纳了若干类配置约束的代码模式, 如配置类型约束 (type constraint)、取值范围约束 (value range)、取值关联 (value relation) 等, 最终设计实现了基于静态分析技术的自动化约束提取工具 SPEX. Liao 等人[18]针对 5 款常用 C/C++开源软件源码进行人工调研, 深入分析了配置约束在源码中的存在形式, 总结归纳了不同约束类型的特征模式和自动化提取的挑战, 并基于启发式规则针对部分配置约束实现自动化提取. Chen 等人[19]考虑到现有软件配置故障通常涉及多配置参数, 针对 Hadoop 和 OpenStack 两大软件生态中共 16 款软件系统进行人工调研. 基于软件提供的文档和配置描述信息, 人工寻找存在关联关系的配置参数, 并定位到源码对应位置, 最终总结归纳了 5 种配置依赖类型及对应源码模式, 并指导实现了配置依赖自动化分析工具 cDep. Zhang 等人[20]设计实现了工具 ConfigX, 通过分析配置参数使用时影响的代码块信息以提取配置参数之间的关联约束关系. 相较于已有配置约束提取工作只关注配置变量的直接使用代码段, ConfigX 进一步深入分析了配置变量影响的代码块信息, 从而可以基于关联的代码块信息提取语义相关的配置关联约束关系. 文献 [21,22] 均通过识别和定位软件源码中的配置读入接口使用代码段, 提取配置参数名称、对应变量和取值类型信息. 考虑到 C/C++程序不同于 Java 的语言特性和实现方式, Zhou 等人[23]针对 8 款 C/C++开源软件源码进行调研, 总结归纳了软件源码中实现配置参数到程序变量映射的代码特征, 并设计实现自动映射工具 ConfMapper, 自动定位配置参数在源码中对应的程序变量.

部分研究工作[24−32]关注从大量配置样本数据中挖掘潜在的配置约束信息. Zhang 等人[24]针对常用软件系统的配置参数进行分类挖掘, 设计了一系列描述配置参数之间关联及配置与环境关联关系的约束模板. 然后设计实现了自动化工具 EnCore, 基于约束模板从配置样本中自动挖掘配置关联约束关系. Chen 等人[25,26]和王焘等人[27]关注 Web 应用中不同组件之间的关联关系, 基于自定义的启发式约束规则模板挖掘配置参数关联约束. ConfigC[28]提出了配置概率类型 (probability type) 的概念, 将配置参数类型表示为类型及对应概率的模式. 然后 ConfigC 将正确样本配置文件解析成结构化的中间表示, 并根据配置参数值的特征推断其概率类型. 最后基于针对不同配置参数类型的预定义约束规则模板, ConfigC 从样本数据中挖掘对应的规则, 最终输出配置参数应该满足的约束规则. 相较于 ConfigC, ConfigV[29]不要求保证样本配置文件的正确性, 而是通过人工预定义约束规则支持度 (support) 和置信度 (confidence) 阈值的方式, 从样本配置文件中挖掘概率化的配置约束. Bhagwan 等人[30]利用版本控制软件中软件配置文件相关的演化历史数据, 基于字符串分析技术从历史配置参数取值中挖掘对应正则表达式, 作为配置取值的格式约束信息. Mehta 等人则设计实现了自动化工具 Rex[31], 从 Microsoft Office 365 和 Azure 等产品的版本维护历史中挖掘频繁同时修改的文件集合, 并使用差异词法分析 (differential syntactic analysis) 处理文件内容, 从而生成细粒度配置参数取值的关联修改规则. Tuncer 等人设计实现了工具 ConfEx[32], 基于已知配置文件构建配置相关的关键字字典, 并通过文本相似度匹配算法自动识别云主机、容器、镜像等实例中未知的配置文件.

少量研究工作[33−35]从语义相关的文本信息中提取配置约束. Xu 等人[33]通过人工调研多款常用开源软件, 总结归纳了配置参数类型分类树, 以及配置参数命名的通用特征, 进而设计实现了基于名称的配置参数类型推断工具 ConfTypeInferer[33], 基于配置名称信息自动推断其可能类型. 同时, 为保证类型推断的正确性, ConfTypeInferer 通过定位配置参数在源码中的对应变量类型验证推断结果. 考虑到软件用户手册通常包含了配置相关描述以及对于配置取值的推荐, Xiang 等人[34]针对 6 款大规模开源软件的用户手册展开调研, 收集并研究了其中 261 条关于配置设置的推荐内容, 发现其中 60% 的相关描述通常包含明确可检查的约束信息, 而且其中 97% 的约束并没有在源码中进行检查. 基于上述现状, Xiang 等人设计提出了自动化工具 PracExtractor, 基于自然语言处理技术从软件用户手册中自动提取配置设计推荐, 并生成对应配置约束. Zhou 等人[35]观察到软件日志中通常包含对于配置约束的描述信息, 设计实现了自动化工具 ConfInLog, 基于静态分析方法从软件源码中筛选配置相关的日志信息, 并基于预生成的配置约束描述相关自然语言模板从日志信息中自动识别配置约束信息.

综合分析来看, 文献 [17,18,20,23] 主要针对 C/C++软件源码进行分析以实现约束提取, 文献 [19,21,22] 主要针对 Java 编程语言实现的软件进行分析. 而文献 [24−32] 主要挖掘样本配置文件中的潜在约束信息实现提取. 文献 [33−35] 主要通过利用文档/配置文件/配置参数名称中的语义信息挖掘配置约束信息. 后两者均不会受限于特定程序开发语言, 但对于配置文件的数量或者文档信息的质量存在一定的要求. 在应用软件类型上, 文献 [17,18,20−23,28−35] 主要针对单个软件内部的配置约束进行分析理解, 而文献 [19,24−27] 则可关注软件不同组件, 甚至是不同软件间的配置约束信息. 此外, 由于现有成熟软件, 特别是服务器软件的代码规模通常比较庞大, 因此上述研究工作均利用了软件配置的部分特性, 可以针对大规模软件进行分析, 并提取约束信息. 最后, 在约束形式上, 文献 [21−23,30,32,33] 只能获得单个配置参数的相关约束信息, 文献 [17−20,25−29,31] 则可以获得多配置参数间的关联约束信息, 而文献 [24,34,35] 等甚至可以获得特定上下文语义环境下的配置约束信息.

3.3 配置影响域分析

配置影响域分析相关研究[36−42]的主要目的是理解到达源码中特定代码段的配置取值约束信息, 从而为提升软件测试覆盖率等目标任务提供优化的基础. Song 等人将软件配置交互 (configuration interaction) 定义为部分配置参数取值组合的约束条件, 该取值约束可以保证特定代码区域被测试覆盖, 而该组合的任何子集都不能保证对应代码区域被测试覆盖, 并设计实现了一个基于迭代搜索的自动化工具 iTree[36], 基于动态分析和决策树算法探索软件源码不同代码块的相关配置交互. Nguyen 等人针对 iTree[36]进行改进, 设计实现了 iGen[37], 基于不同配置取值组合下的代码覆盖率相关信息推断可能的配置交互. 相较于 iTree 主要针对 C 语言程序实现, 且仅支持合取 (conjunctive) 形式的配置交互命题逻辑条件, iGen 可以支持多种编程语言, 以及更复杂的命题逻辑形式. 进一步的, Nguyen 等人又提出了 GenTree[38], 通过结合代码覆盖率信息和决策树算法实现配置交互的识别. Lillack 等人[39,40]

设计实现了工具 Lotrack, 基于改进的静态污点传播分析方法, 将配置参数的取值信息加入污点分析过程中, 从而得到到达不同代码块的配置参数取值约束. 相较于程序切片技术, 特别是前向切片技术, Lotrack 只关注与配置参数直接相关的语句与分支, 显著减少了切片大小. 为分析探索所有配置组合的可能性, Meinicke 等人[41]设计实现了动态分析工具 VarexJ, 用于分析研究软件源码中配置交互的复杂程度. VarexJ 基于条件取值 (conditional value) 和可变上下文 (variability contexts) 的设计, 实现了针对 Control-flow Interaction Degree、Data Interaction Degree 和 Interaction Overhead 这 3 种配置交互复杂程度的度量, 并分析了不同软件在不同负载情况下的配置交互的复杂程度. Angerer 等人[42]提出了配置敏感的变更影响分析技术 (change impact analysis, CIA), 在传统 CIA 技术的系统依赖图 (system dependence graph, SDG) 基础上, 提出了增强的条件系统依赖图 (conditional system dependence graph, CSDG), 基于二元决策图 (binary decision diagram, BDD) 表示对应存在条件, 从而将配置导致的变化表现在现有 SDG 中, 实现了更准确的 CIA 分析.

3.4 分析与小结

总体而言, 目前已存在部分工作[11,15,17]针对配置相关代码模式和运行机理进行了探索, 但仍缺乏相关的方法指导, 未来需要更加规范的配置设计指导, 提升配置相关的代码可用性和可维护性. 此外, 配置约束提取和配置影响域分析的相关工作大多基于程序分析、机器学习等技术手段实现, 功能实现上存在诸多限制. 首先, 现有配置约束提取方法大多均需要人工预先定义配置约束的模式规则, 对研究人员的领域相关知识要求较高; 其次, 现有配置参数取值类型多样, 现有工作仅能针对部分简单取值类型的配置参数提取约束, 而在处理具有复杂取值类型的配置参数时, 如结构化取值配置参数、多参数取值配置参数, 存在较大局限性; 而且, 现代软件系统规模庞大, 且大多由多编程语言开发实现, 基于程序分析的研究方法均面临分析规模受限、语言特性不同等问题; 最后, 配置参数的合理取值通常还与软件系统的运行环境及工作负载相关, 满足静态配置约束的参数取值在特定环境或负载下可能也会导致配置故障. 现有静态分析方法难以获得变化环境/负载下的准确配置约束, 对于环境/负载相关配置故障的预防能力不足.

4 配置缺陷检测与故障诊断

现有配置缺陷检测与故障诊断研究工作从研究目标上看主要分为两类, 即事故前的配置相关缺陷检测与故障预防, 以及事故后的配置故障诊断与修复. 具体研究工作可按图 4 流程进行归纳. 在配置故障发生前, 通过检测配置相关的代码缺陷并进行故障预防, 可以最大程度地避免生产环境中配置故障的发生. 当配置故障发生后, 快速准确的配置故障诊断与修复有助于减少损失、及时恢复软件的正常运行. 本节主要从配置故障现状调研、配置缺陷检测、配置故障预防和配置故障诊断与修复 4 个方面对现有研究工作进行归纳分析.

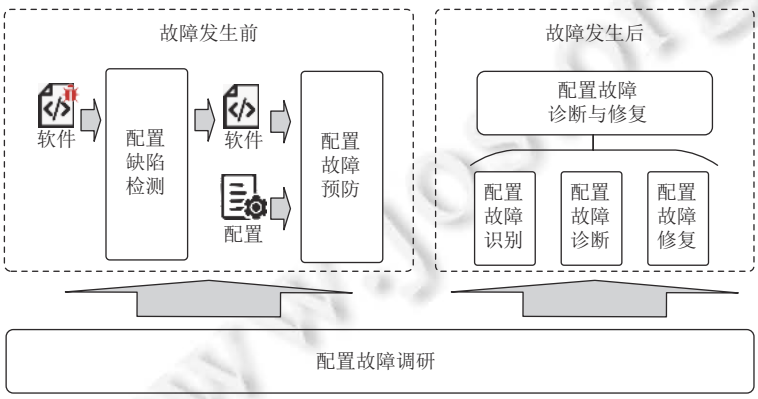

图 4 配置缺陷检测与故障诊断相关工作流程

4.1 配置故障调研

部分研究工作[7,43]针对真实配置故障的存在现状进行研究, 深入理解其特征.

Yin 等人[7]首次针对软件配置故障展开实证调研, 通过对 1 款商业存储软件和 4 款开源软件中共 546 个配置进行特征分析, 深入研究配置故障类型、故障模式、故障原因、系统反应以及故障的影响, 总结归纳了相关发现, 如 70.0%–85.5% 的配置故障由配置参数错误导致, 其余配置故障则由兼容性错误和组件错误导致; 38.1%–53.7% 的配置参数错误明显违反了格式或规则要求, 为实现配置故障自动检查提供了机会, 而其中 12.2%–29.7% 的配置参数错误则是由于不同参数间的不一致取值导致等. 上述实证调研的结论为后续配置故障预防和故障诊断工作提供了实践经验和数据支撑.

Han 等人[43]针对配置相关的性能故障展开调研, 尝试通过理解配置如何影响软件性能, 指导配置性能故障诊断相关技术的发展, 以及配置性能调优. 调研主要关注配置故障导致的软件性能故障, 以及特定配置下触发的软件性能故障. 通过人工分析 3 款开源软件中的 193 个性能故障, 发现 59% 的性能故障均由配置问题导致, 其中 78%–92% 的配置性能故障与配置参数设置相关, 且通常由配置取值空间的一小部分子集导致. 此外, 通过研究配置性能故障的原因、修复措施的特点等为软件开发人员和研究人员理解和解决软件性能故障总结了经验.

现有配置故障调研工作通过对真实案例的调研和分析, 深入梳理了配置故障的相关特征和模式, 为后续故障诊断和预防工作提供了良好的基础.

4.2 配置缺陷检测

良好的配置缺陷检测技术有助于辅助开发人员优化配置相关代码实现, 提升软件代码质量, 从而避免配置故障的发生. 现有配置缺陷检测工作主要分为配置功能代码缺陷检测、配置故障反应能力缺陷检测、配置相关测试优化 3 类.

4.2.1 配置功能代码缺陷检测研究

配置功能代码缺陷检测相关研究[44−48]主要关注检测软件代码中存在的配置相关功能缺陷或者性能缺陷. Behrang 等人[44]提出了自动化工具 SCIC (software configuration inconsistency checker), 通过分析软件配置的用户接口层与功能实现代码层间涉及的配置参数不一致, 实现用户接口层中无效 (无对应功能实现) 配置参数的检测. Toman 等人[45]设计实现了工具 Staccato, 基于动态污点技术软件运行后的 trace, 并匹配预定义的配置变量值使用规则, 从而检测配置动态更新相关功能存在的缺陷. 考虑到动态分析方法受限于测试用例质量的问题, Toman 等人进一步提出了工具 Legato[46], 通过静态分析方法, 严格证明配置动态更新过程中是否存在访问值不一致. 由于在静态分析过程中假设最极端的条件, 因此保证了 Legato 静态分析的可靠性 (soundness). 针对软件性能故障检测缺乏有效测试语言的问题, 即软件运行低效时, 难以判别是由于性能缺陷导致或由负载较大导致, He 等人[47]针对配置相关性能缺陷展开调研, 发现 86.7% 的配置相关的性能缺陷都具备“反直觉”特征 (如增大资源分配, 预期性能提升, 但实际性能下降), 并总结归纳了 7 类“反直觉”模式和软件测试加速启发式规则. 基于上述模式设计实现了配置相关性能缺陷的自动化检测工具 CP-Detector, 利用配置文档预测配置调节直觉模式, 并基于启发式方法高效暴露配置相关性能缺陷. Velez 等人[48]针对开发人员诊断软件配置相关性能故障的需求进行分析调研, 总结归纳了 3 类相关信息有助于开发人员进行诊断, 分别是相关配置参数 (influencing options)、相关配置参数的相关函数调用 (option hotspots), 以及相关配置参数的影响链 (cause-effect chain). 然后, 基于上述发现, Velez 等人[48]设计实现了自动化原型工具 GLIMPS, 用于为开发人员诊断提供相关的信息, 并通过用户调研的形式评估了 GLIMPS 工具对于帮助诊断配置性能问题的有效性.

4.2.2 配置故障反应能力缺陷检测研究

配置故障反应能力缺陷检测相关研究[17,49−53]则主要通过配置故障注入的方式, 观察评估软件系统对故障的反应能力, 从而指导开发者提升相关反应代码质量, 提高配置故障诊断的效率. SPEX[17]基于软件源代码中的配置相关使用模式提取对应约束信息, 并设计实现了基于约束的自动化配置故障注入工具 SPEX-INJ, 评估软件的反应能力. Arshad 等人[49]通过调研 Java EE 应用服务器软件 GlassFish、JBoss 中的配置故障发现, 超过一半都是由于配置参数的不当设置导致. Arshad 等人进而设计实现了面向 XML 型配置文件的自动化配置故障注入工具 ConfInject. ConfInject 主要使用两种启发式规则生成配置故障, 即对配置参数值中的“/”进行增减, 以及直接将配置参数值替

换为空. Zhang 等人设计实现了自动化工具 ConfDiagDetector[50], 主要基于预定义的 5 种变异规则生成配置故障, 包括删除配置值、随机选择同类型预定义其他取值、随机生成其他类型配置值、随机注入拼写错误、修改大小写. 此外, ConfDiagDetector 还实现了基于自然语言处理技术的日志信息处理功能, 通过使用 IDF (inverse document frequency) 算法计算日志中诊断信息与故障配置参数描述的文本相似度, 从而评估诊断信息是否充分. Li 等人[51,52]基于人工调研总结归纳了配置参数类型分类树, 并提出了增强 BNF 范式 (augmented backus-naur form) 归纳描述每种配置类型的语义和语法约束, 最后通过违反配置参数类型的语义与语法约束生成配置故障. 进一步的, Li 等人[53]针对现有的配置故障注入测试技术 (configuration error injection testing, CEIT) 进行研究, 理解和分析现有 CEIT 技术的机遇与挑战. 为实现上述目标, Li 等人设计实现了模块化的配置注入测试框架 CeitInspector, 实现了 3 种配置故障生成技术, 并针对 6 款大型开源服务器软件中近两千个配置参数进行测试和实证研究. 实证研究结果表明, 目前的配置故障注入测试, 仍然受限于软件自身机制, 配置故障生成方法以及软件测试等环节, 并分别提出了可行的解决方案, 如自动化日志插入、配置故障变异规则优化等, 帮助提升配置缺陷检测能力.

4.2.3 配置相关测试优化研究

配置参数取值通常会影响程序的执行路径, 从而影响相关代码的缺陷检测. 考虑到配置参数规模庞大、配置取值组合空间爆炸等问题, 部分研究工作[54−57]关注配置相关测试优化问题, 主要研究如何筛选配置相关测试用例和待测配置参数以提升软件测试效率. 文献 [54] 针对工业软件 ABB 和两款开源软件 Firefox、LibreOffice 中展开调研, 量化分析了配置对于软件测试和故障诊断的影响与带来的挑战. Qu 等人[55]基于静态程序切片技术分析待测配置参数改变时影响的代码段, 从而筛选出执行路径覆盖对应代码段的测试用例. 进一步的, Qu 等人针对软件回归测试提出了待测配置筛选方法[56], 给定目标软件的新旧版本源码, 通过基于切片的代码影响域分析技术 (slice-based change impact analysis) 确定两个版本之间代码改变的切片, 并在单个版本内对配置参数进行切片, 若两个切片之间存在语句重叠, 则将该配置作为待测配置. Marijan 等人[57]设计实现了工具 Titan, 通过构建对应二分图 (bipartite graph), 并基于 Ford-Fulkerson 算法求解最大流, 从而确定可以覆盖所有目标待测配置的最小测试用例集合; 然后, Titan 通过计算新增代码覆盖率对测试用例进行排序, 从而提升测试的效率.

4.2.4 分析与小结

综合来看, 配置缺陷检测相关研究的方法主要分为基于测试和基于程序分析的方法. 基于测试的方法主要包括文献 [17,47,49−53], 它们主要依赖软件系统已有测试, 包括测试预言、测试用例等, 存在测试用例不足、测试不充分等问题. 而且, 软件已有测试用例通常针对软件功能进行设计, 缺乏面向配置的测试用例设计, 进一步加剧了上述问题的严重性. 而基于程序分析的方法又分为基于静态程序分析的方法 (文献 [44,46,55−57]) 和基于动态程序分析的方法 (文献 [45,48]). 然而, 上述基于静态程序分析方法的应用通常会受限于目标软件的代码规模, 难以适用于大规模软件系统, 而动态程序分析同样受限于程序输入的生成等问题. 此外, 现有研究工作仅将配置作为普通的输入, 并未考虑配置的特点和其在源码中的存在及使用特征, 同样导致现有程序分析方法受限.

4.3 配置故障预防

配置故障预防相关研究的目的是在软件运行前或初始启动时, 提前暴露可能的配置故障, 从而避免软件运行过程中出现如软件失效、服务不可用等严重配置故障, 减少不必要的数据和经济损失. 目前, 配置故障预防的相关工作主要包括基于知识的配置故障预防, 以及基于模拟的配置故障预防两类.

4.3.1 基于知识的配置故障预防研究

基于知识的配置故障预防相关工作[58−61]主要利用已有知识检测当前配置设置中是否存在错误的配置设置. Eshete 等人[58]针对 Web 应用中的配置安全相关故障进行预防, 设计实现了工具 Confeagle. 给定一个 Web 应用和对应的部署环境, Confeagle 提取其中配置键值对信息, 并与已知的配置安全设置标准进行比较并打分, 最终将配置故障隐患输出给用户, 并基于已知的设置标准为用户修复提供参考. Otsuka 等人[59]则通过分析软件故障历史记录中配置故障恢复前后的配置参数取值差异, 定位导致故障的配置参数及对应的值特征, 从而构建配置故障与配置参数取值差异的关联知识库. 最后, 对于给定的配置设置输入, 基于配置参数取值的统计信息实现潜在配置故障

的检测与预防. Huang 等人设计实现了配置验证框架 ConfValley[60], 开发者基于声明式语言 CPL 描述配置所需满足的规约, ConfValley 则基于推断引擎将声明式语言进行转换, 并通过检查器判断输入的配置设置是否满足规约. 相似的, Baset 等人[61]设计实现了声明式语言 CVL (configuration validation language) 用于声明配置规约, 以及对应的验证工具 ConfigValidator 用于预防配置故障.

4.3.2 基于模拟的配置故障预防研究

而基于模拟的配置故障预防相关工作[62−64]则是通过在模拟环境中提前运行软件, 从而尽早暴露可能存在的配置故障. Xu 等人[62]提出了工具 PCheck, 通过分析配置参数在软件源码中的使用代码段, 提取运行所需上下文信息, 自动生成对应的检查器, 并在软件启动阶段进行检查, 提前暴露潜在配置缺陷, 避免软件运行中由于配置缺乏检查导致的配置故障. Sun 等人提出了 Ctest[63]用于检测可能导致配置故障的配置修改, 以避免生产环境中的系统失效. Ctest 的核心思想在于利用软件已有的测试用例检测配置设置, 既可以避免一般的配置故障, 也可以避免出现配置参数合法取值导致的生产环境下的配置故障. Ctest 首先通过动态程序分析确定单元测试覆盖的配置参数, 并将其参数化 (parameterization); 然后将测试用例需要满足的配置取值固定, 并基于启发式规则自动生成其他可变配置参数的取值, 最后通过单元测试是否通过检测配置故障. 为减少测试开销, Cheng 等人[64]进一步基于传统测试用例排序技术 (test-case prioritization, TCP), 对 Ctest 进行排序以加速配置故障的检测.

4.3.3 分析与小结

现有基于知识的配置故障预防相关工作依赖于提前建立知识库[58,59], 或者人工编写配置规约[60,61], 对相关知识库的日常更新与维护和研究人员的领域知识存在较高要求; 而基于模拟的配置故障预防工作则仅能模拟可以静态确定的简单执行上下文环境[62]或者测试用例的相关[63,64]上下文, 难以模拟生产环境中多样的负载和复杂的上下文环境, 导致针对负载敏感的配置故障预防能力不足.

4.4 配置故障诊断与修复

当软件发生故障时, 快速识别是否是由配置导致的故障, 定位导致故障的配置参数, 并最终修复配置故障, 有助于提升软件系统的可用性. 目前文献 [9,10] 已针对配置故障诊断与修复相关工作展开了综述, 本文在上述工作的基础上进行了补充和完善, 按照人工诊断与修复工作流程, 将配置故障诊断与修复相关研究分为配置故障识别、配置故障诊断和配置故障修复 3 步.

4.4.1 配置故障识别研究

配置故障识别相关工作[65−68]主要研究如何识别软件故障是否由配置导致. 给定一个软件故障报告, Xia 等人[65]基于已标注的软件历史故障报告数据训练统计模型, 从而预测其是否为配置故障报告. 具体来说, Xia 等人首先基于特征选择技术处理故障报告中的文本信息, 生成对应的特征; 然后构建统计模型用于故障报告的预测分类. 进一步的, Xu 等人在文献 [65] 的基础上, 提出了基于组合式特征选择的配置故障报告预测工具 EFSPredictor[66], 综合考虑多种特征选择技术的结果, 选择排名靠前的特征构建模型, 以提升预测模型的准确性. Wen 等人[67]设计实现了工具 CoLUA, 使用卡方检验算法从故障报告文本中提取特征, 并基于朴素贝叶斯、决策树和逻辑回归算法构建分类器模型, 预测故障报告是否与配置相关. 当预测结果为真时, CoLUA 进一步基于故障报告与配置参数名称的文本相似度筛选可能相关的配置参数信息, 从而指导配置故障诊断. 此外, Yuan 等人设计实现了自动化工具 CODE[68], 用于在线检测 Windows 系统注册表相关的配置故障. 在 Windows 系统中, 软件运行时会频繁访问注册表获得配置信息, 因此注册表访问序列可以反映软件的控制流信息, 描述运行上下文状态. CODE 在线监控软件正常运行时的注册表访问序列, 并基于 Sequitur 算法[69]挖掘访问序列模式. 当注册表访问序列违反了已知的访问序列模式时, CODE 提示用户可能出现配置故障.

4.4.2 配置故障诊断研究

配置故障诊断是现有工作的主要研究方向, 加州大学伯克利分校的 Katz 研究组[70,71]、海德堡大学的 Andrzejak 研究组[72,73]等均对配置功能故障和性能故障诊断展开深入研究[74−81]. 配置故障诊断的目的是通过程序分析、机器学习等手段定位导致故障的可疑配置参数, 从而辅助系统维护人员快速修复故障.

动静态程序分析技术被广泛应用于配置故障诊断相关研究中[70−78]. Rabkin 等人设计实现了工具 Conf-Analyzer[70,71], 基于文献 [21] 提出的方法定位软件源码中的配置参数读入位置, 并使用静态分析技术构建配置参数与代码中潜在异常点的映射关系. 当软件出现配置故障后, 通过检索相应映射关系即可快速定位故障配置参数. Dong 等人同样使用静态分析技术定位导致故障的配置参数, 设计实现了分析工具 ConfDoctor[72,73]. ConfDoctor 分别分析配置参数影响的程序语句, 和影响出错栈的程序语句, 并通过判断两者之间是否存在相同程序语句, 检测故障是否与当前配置参数相关. Zhang 等人设计实现了自动化工具 ConfDiagnoser[74], ConfDiagnoser 首先基于瘦切片技术 (thin slicing) 确定每个配置参数影响的控制流信息, 然后针对相关分支语句进行插桩, 并记录程序正常运行情况下的执行路径信息构建数据库; 当程序运行中产生配置故障时, ConfDiagnoser 寻找数据库中与出错执行路径最相似的路径, 并通过分析两路径之间信息差别最大的分支语句程序点, 定位导致故障的配置参数. 进一步的, Zhang 等人将 ConfDiagnoser[74]的思想应用于软件演化过程中, 提出了自动化工具 ConfSuggester[75], 通过对新旧版本的软件执行相似的步骤, 从而定位导致版本间软件行为差异的配置参数. Sayagh 等人[76,77]针对跨栈软件配置故障 (cross-stack configuration errors, CsCE) 进行研究, 设计提出了模块化的跨栈配置故障诊断方法, 基于已有程序切片技术, 跟踪软件中不同栈层软件之间的调用关系, 从而构建跨栈切片依赖图 (cross-stack slice dependency graph). 当配置故障发生时, 首先定位出错日志的输出语句位置, 然后基于跨栈切片依赖图定位可能导致故障的相关配置参数. Attariyan 等人[78]针对软件中的配置相关性能故障进行诊断, 设计实现了自动化工具 X-ray. X-ray 首先针对二进制程序进行插桩, 并记录程序基本块在运行时的执行开销; 然后基于动态信息流追踪技术, X-ray 估算每个潜在故障点 (如配置读入点) 触发当前基本块执行的概率, 并将基本块开销与概率的乘积累加作为潜在故障点的总开销, 从而将潜在故障点对应的配置参数排序输出给用户.

少量研究工作[79−81]则使用其他多种方法诊断配置故障. Kannan 等人[79]基于云环境下不同组件之间的共享资源信息诊断配置故障, 首先基于 TADDM 工具[82]提取不同组件内的配置参数信息, 及配置相关的资源信息, 然后针对不同组件之间共用同种资源的配置参数取值进行差异分析, 从而定位可能出现故障的组件和配置参数. Wang 等人针对 6 款开源软件中配置故障的相关日志展开调研, 发现异常日志序列通常包含特定特征, 并设计实现了工具 MisconfDoctor[80], 基于 ConfTest[52]主动生成配置故障以获得对应日志序列, 并基于序列对齐算法 (sequence alignment algorithm) 利用正常日志序列和异常日志序列对比从而构建特征数据库. 当新的配置故障发生后, MisconfDoctor 通过对比日志序列相似度寻找最相似的已有配置故障, 从而指导故障诊断. Han 等人[81]针对配置相关的软件性能故障进行研究, 并设计实现了工具 PerfLearner, 利用自然语言处理、数据挖掘等技术, 从软件历史性能故障报告中提取易错配置参数, 并指导性能测试用例的生成, 从而检测配置相关的性能故障.

4.4.3 配置故障修复研究

部分工作[83−85]依托于故障修复历史信息研究配置故障修复问题. Huang 等人[83]提出了自动化配置故障修复工具 Ocasta. Ocasta 首先基于层次凝聚聚类 (hierarchical agglomerative clustering) 算法, 按照历史修改信息对配置参数进行聚类; 然后, 基于用户提供的配置故障复现操作和故障发生时间, Ocasta 使用重放技术搜索聚类配置的取值并检查故障是否修复. Potharaju 等人基于自然语言处理, 信息检索和交互式学习相结合的方法设计实现了 ConfSeer[84], 并依托于 Microsoft 服务维护的相关知识库实现配置故障诊断. ConfSeer 首先获得用户的配置文件并解析为键值对形式, 然后基于 TF-IDF (term frequency-inverse document frequency) 文本相似度算法与配置故障知识库文档 (knowledge base, KB) 进行匹配; 当用户配置与 KB 匹配成功后, ConfSeer 将对应 KB 发送给用户指导配置故障的诊断与修复. Talwadker 等人[85]主要基于软件配置故障产生的日志信息进行配置故障诊断和修复, 并设计实现了自动化工具 Dexter. Dexter 解析软件的日志输出, 并基于启发式指标进行排序, 从而筛选出故障相关的日志信息. 当配置故障修复后, Dexter 将修复操作与故障日志信息关联, 并形成数据库. 一旦发生新配置故障, Dexter 可以通过匹配日志信息从而快速推荐故障修复方案.

4.4.4 分析与小结

总结来看, 配置故障诊断相关研究仍面临巨大挑战. 首先, 部分配置故障与一般软件故障之间无明显区别, 且

缺乏有效提示信息, 用户难以在上报故障时提供相关的配置参数信息, 导致现有配置故障识别方法难以正确识别. 其次, 随着软件系统规模日益增大, 跨组件的配置故障已占据相当大的比例[7]. 目前已有部分研究工作[19,77]针对跨组件的配置故障诊断问题进行了探索, 但主要关注配置参数之间的显式关联, 缺乏对于配置参数间隐式关联影响的考虑, 例如由于计算资源、网络资源等资源共享竞争导致的性能故障[86]. 最后, 现有常用开源软件大多仍缺乏对历史配置故障相关知识的维护和使用, 导致现有配置故障修复工作使用受限.

5 配置应用研究

配置应用相关工作主要研究如何帮助用户快速筛选满足特定功能的配置参数列表, 以及如何寻找满足特定目标的配置参数取值, 从而保证软件系统按照用户预期的意图高效运行, 并保证软件适应外部的复杂多变环境. 现有配置应用研究工作主要包括配置参数推荐、配置性能调优两个方面.

5.1 配置参数推荐

配置作为软件与外部环境接口, 可以根据用户的意图控制功能生效及管理资源的分配和使用. 然而, 由于软件功能的不断增强, 配置参数数量日益增多, 用户难以从庞大的配置空间中快速定位满足功能性或者性能需求的目标配置参数. 基于上述问题, 部分研究工作针对配置参数推荐问题展开研究, 主要分为功能相关的配置参数推荐和性能相关的配置参数推荐两类.

5.1.1 功能相关配置参数推荐

为推荐功能相关配置参数[87−90], Jin 等人设计实现了自动化工具 PrefFinder[87], 首先解析配置参数名称和用户查询语句文本, 并计算配置参数与目标查询语句的文本相似度, 最终基于相似度排序实现相关配置参数推荐. Sayagh 等人设计实现了自动化工具 ConfigMiner[88], 通过比较配置相关问题描述与历史问题的文本相似度, 筛选相关的历史问题, 并将对应答案中的配置参数推荐给用户. Hamidi 等人[89]考虑从群体智慧中学习知识, 首先从历史配置数据中挖掘同时出现的配置参数作为关联规则, 并使用关联规则加速马尔可夫决策过程 (Markov decision processes, MDP) 模型的构建, 最终基于 MDP 模型实现 Facebook 中隐私配置相关设置的推荐引导. Zhou 等人[90]则基于配置参数文档信息自动识别配置的意图. 基于面向 19 款开源软件中 5 470 个配置参数的人工调研, Zhou 等人将现有配置参数的意图归纳为 6 类, 然后基于序列规则挖掘技术筛选配置文档中意图描述相关的文本信息, 并使用 CNN 训练分类器, 最终实现配置参数意图的自动识别.

5.1.2 性能相关配置参数推荐

性能相关的配置参数筛选[91−94]有助于减少配置性能调优方法的配置搜索空间, 或是方便用户自主调优软件. 为实现上述目标, Cao 等人设计实现了 Carver[91], 使用性能值方差衡量不同配置参数对应存储系统性能的影响程度, 并使用拉丁超立方采样对配置空间进行均匀采样, 最终基于贪心算法从少量的初始配置样本中选择对性能影响程度最大的配置参数. 相较于 Carver[91]主要针对存储系统, Kanellis 等人[92]则针对数据库软件进行分析. 通过针对 Cassandra、PostgreSQL 软件在 YCSB-A 和 YCSB-B 两种负载下进行不同配置下的性能测试, 并使用 CART 算法量化配置参数对于性能的影响程度, 从而确定对性能影响更大的配置参数. Li 等人关注理解配置参数如何通过相关的时间/空间密集型 (time/space-intensive) 的程序操作影响软件性能, 即性能相关操作 (performance operations, PerfOps), 并设计实现了 LearnConf[93], 基于程序静态分析技术自动识别 PerfOps 相关的配置参数, 从而确定配置参数对软件性能的影响属性. 考虑到其他方法仅关注软件性能而忽略其他意图, He 等人[94]通过对数十个大型开源软件的近 8 千个配置参数进行调研, 总结了 6 种性能相关配置对其他用户意图的影响模式, 并设计了自动化工具 SafeTune, 利用自然语言处理技术从文档中自动识别性能相关配置并预测其对其他意图的影响.

5.1.3 分析与小结

总结而言, 现有的配置参数推荐部分相关研究依赖于自然语言中的文本信息, 通过利用文本相似度及其包含的语义信息推荐相关的配置参数[87,88,90,94], 但是现有软件系统的配置参数并没有统一的命名规范, 配置相关文档同样存在信息不完整、更新不及时等问题, 影响了上述方法的准确性和适用性. 而基于动态测试的相关方法[91,92]则

与当前测试的工作负载和环境强相关, 且对应性能测试开销很大, 一旦环境或负载变化可能导致方法准确性大幅下降, 同样影响了方法的适用性.

5.2 配置性能调优

软件性能作为软件的常见非功能指标, 广泛影响着软件系统的服务能力和生产经济效益. 而配置作为用户调控软件的主要抓手, 配置性能调优领域受到了广泛的关注, 大量工作针对配置性能调优展开研究. 配置性能调优相关研究的目的是从软件配置空间中筛选出性能表现更好的配置参数取值组合, 从而最大化硬件利用率, 提升软件的服务能力. 从研究方法上看, 配置性能调优相关工作可以分为 3 类, 分别是基于搜索的方法、基于模型的方法和其他方法. 基于搜索的方法主要通过迭代采样配置取值, 检查当前性能是否最优, 并根据相关信息和优化算法决定下一步的采样方向; 而基于模型的方法则是通过配置采样及对应性能表现信息构建模型, 并基于模型实现对于特定配置取值的预估; 其他方法相关工作则暂时不能归类于上述两类方法, 且不存在统一的特点.

5.2.1 基于搜索的配置性能调优方法

基于搜索的方法主要目标是通过有限次的探索配置取值空间, 从而寻找最优配置取值. 为实现上述目标, 基于搜索的方法通常利用已搜索配置空间的相关信息指导下一次搜索的方向, 从而减少需要的搜索开销. 现有相关研究工作主要关注如何提升优化算法的搜索效率, 避免陷入局部最优解.

部分研究工作[95−98]关注通用的搜索算法优化. Zhu 等人考虑在有限采样数量条件下的配置性能调优问题, 提出了工具 BestConfig[95]. 在给定 k 次采样限制下, BestConfig 采用 divide-and-diverge 的采样算法, 通过将配置取值空间分为 k 段, 实现配置空间的覆盖, 搜索最优配置时, BestConfig 使用递归的 bound-and-search 算法, 通过已有搜索历史中的较优配置信息, 限制下一次搜索的范围, 从而提高搜索的效率. Wang 等人设计实现了基于控制论的配置性能调优工具 SmartConf[96], 对于复杂动态变化的负载, 通过深度定制一个动态调节配置参数取值的外部库, 并由开发者参与指定性能相关配置参数及对应属性, 从而实现了动态自适应调整性能相关配置参数的框架, 可以实现针对不同工作负载和预期性能目标的自动搜索更优配置, 以达到预期性能指标. Bao 等人[97]假设性能表现较好的配置通常共享某些隐藏结构, 并使用生成式对抗网络 (generative adversarial network, GAN) 自动挖掘上述结构特征, 从而指导下一轮迭代搜索, 以获得性能更好的配置. Chen 等人[98]提出了多目标优化模型 (multi-objectivization model, MMO) 方法, 通过在建模过程中设置多个优化目标, 通过利用辅助性能目标状态信息, 避免搜索陷入主性能目标的局部最优.

部分研究工作[99−106]则针对特定类型的目标软件系统进行研究, 充分利用目标软件系统的特征实现搜索算法的优化. Lengauer 等人[99]针对 Java 垃圾回收机制 (garbage collection, GC) 的性能调优展开研究, 给定一个 Java 目标软件, 使用迭代的局部搜索算法寻找对应最优配置. Li 等人提出了 Mronline[100], 针对 MapReduce 应用实现动态在线性能调优, 支持单个 MapReduce 任务 (Task) 的配置调整. Mronline 通过使用一种基于灰盒的智能爬山算法 (hill climbing algorithm), 并且针对 MapReduce 定制了启发式调优规则以提升搜索质量和收敛速度, 通过从不同的点重新爬山, 以增加从局部最优到全局最优的机会. Ding 等人提出了一个在线性能调优框架 JellyFish[101], 用于提升 MapReduce 的性能以及 Yarn 资源利用率. JellyFish 设计实现了一种可以动态扩张或缩小的弹性容器 (elastic container), 以及相应的一套调度策略 (rescheduling strategy). 在线调优时, JellyFish 基于分治 (divide-and-conquer) 方法减小配置搜索空间维度, 并使用基于模型的爬山算法 (model-based hill climbing algorithm) 进行搜索提升最优配置搜索效率. Singh 等人[102]基于多目标遗传算法 (multi-objective genetic algorithm) 自动寻找 ORM (object-relational mapping) 软件的最优配置. Jamshidi 等人[103]基于高斯过程 (Gaussian processes) 方法迭代获得配置空间的后验分布 (posterior distributions), 并使用贝叶斯优化 (Bayesian optimization) 算法在有限开销下寻找流处理系统的性能最优配置. Bilal 等人[104]将流处理系统的性能问题视为一个多目标优化问题, 即考虑在满足特定吞吐量目标的情况下尽可能减小延迟, 并基于多种优化算法实现对应调整框架. 此外, 考虑到 SPS 软件的特点, 提出了一种基于灰盒的优化算法, 基于 SPS 软件中的先验知识, 启发式的加快调优过程. Mahgoub 等人提出了 SOPHIA[105], 利用 NoSQL 软件的历史运行 Trace 训练了一个负载预测器, 基于当前负载特征预测未来负载的变化模式; 然后, SOPHIA

基于成本效益分析 (cost benefit analysis, CBA) 预估不同配置调整方案过程中每个步骤的收益, 从而决策当前步骤下配置调整的操作, 保证大规模分布式 NoSQL 集群的性能. Krishna 等人[106]针对以 Hadoop 和 Spark 为代表的大数据处理软件性能调优问题提出了基于进化马尔可夫链与蒙特卡罗 (evolutionary Markov chain Monte Carlo, EMCMC) 的采样策略, 即将经典的 MCMC 算法与遗传算法思路进行结合, 一方面可以通过 MCMC 算法估计出配置空间的分布, 提升采样效率; 另一方面也可以通过遗传算法的加入加快 MCMC 算法的收敛速度. 在获得预期样本后, Krishna 等人进一步发现, 轻负载下的较优配置可以应用于对应的重负载或者具有相似运行时动态相似度 (dynamic similarity) 的其他类型负载中, 从而设计实现了工具 ConfEx, 可以基于 EMCMC 采样方法快速发现较小负载下的较优配置, 进而在生产环境中快速应用于真实负载.

综合来看, 基于搜索的配置性能调优研究工作主要发展历程如下: 首先, 文献 [95,99] 等通过基础的搜索方法寻找最优配置配置, 但往往会存在陷入局部最优的情况出现. 针对上述问题, 文献 [100−102] 等提出了基于多种智能算法的搜索方法, 尝试将智能算法的优化思想应用到配置搜索上, 减少局部最优的情况出现; 文献 [98,104] 则关注应用多目标优化算法进行配置搜索, 通过多个优化目标的设置避免陷入局部最优. 为了提升对性能最优配置的搜索效率, 文献 [103,106] 还针对配置空间分布问题进行了研究, 尝试理解配置空间的分布特性从而加快搜索收敛速度. 此外, 其他研究工作还分别尝试使用控制论[96]、生成式对抗网络 (generative adversarial network, GAN)[97]、成本-收益分析 (cost-benefit analysis)[105]、负载扩张[106]等思想进行配置性能调优.

5.2.2 基于模型的配置性能调优方法

基于模型的方法相关工作主要关注如何构建描述配置参数取值与软件系统性能表现的模型, 并通过构建的模型对于给定配置输出给出预测的软件性能值. 基于模型的方法通常首先针对配置空间采样, 然后通过测试方法获得样本配置取值的对应软件性能值, 最终通过机器学习或者统计方法构建对应模型. 特别的, 考虑到配置空间的庞大规模以及采样测试在建模过程中的重要性, 部分工作着重关注如何实现高效、高覆盖的配置空间采样算法.

部分研究工作[107−119]研究普适性的性能模型构建方法. Siegmund 等人[107]首次使用黑盒方法针对多布尔类型配置之间的交互情况进行分析, 提出了一种自动检测性能特征交互的方法, 以提高性能预测精度; 同时, 基于 3 种启发式方法, 可以有效减少检测配置交互所需的测试数量. 进一步的, Siegmund 等人[108]通过利用 Plackett-Burman 设计针对数值类型配置参数进行采样将配置空间采样扩展到布尔类型和数值类型; 在构建模型时, 首先针对单个配置参数建立对应的模型性能线性方程, 并最终以线性的形式将不同的方程合成一个总的性能模型方程. Guo 等人[109]首次将配置性能建模问题抽象为非线性回归问题, 基于随机采样算法采样布尔类型配置组合, 并使用分类与回归树 (classification and regression trees, CART) 算法构建性能模型. 然后, Guo 等人[110]提出了重采样策略, 并结合自动化的 CART 模型超参调节构建配置性能模型. Zhang 等人[111]基于傅里叶变换相关理论, 将配置性能模型抽象为构建傅里叶稀疏函数 (Fourier sparse function), 将建模过程抽象为计算函数对应的傅里叶系数 (Fourier coefficients), 实现配置性能模型的构建. Nair 等人[112]通过调研与实验发现, 构建配置性能模型通常不需要性能的准确描述, 只需要区分不同配置参数对性能影响的相对排名, 即可最终确定最优配置. 因此, Nair 等人提出了基于排序的性能模型构建方法, 通过较少测试开销构建粗糙模型对配置性能表现进行排序, 实现较优配置的筛选, 避免构建精确模型的巨大采样开销. 进一步的, Nair 等人提出了方法 FLASH[113]. 相较于文献 [112], FLASH 实现了基于可排序模型的优化方法 (sequential model-based optimization) 推测配置空间的可能分布信息, 从而基于已有的样本运行信息在当前条件下选择下一步的最优配置, 提高寻找最优配置的效率. Ha 等人提出了 DeepPerf[114]工具, 利用深度前馈神经网络 (deep feedforward neural network , Deep FNN) 结合稀疏正则 (sparsity regularization) 采样的方法, 实现面向布尔类型和数值类型配置参数的采样和性能模型构建. 同时, 为解决稀疏 FNN (sparse FNN) 中超参数 (hyper-parameter) 数量多的问题, DeepPerf 设计实现了自动化搜索策略, 用于调优神经网络的超参数设置, 提升模型训练的效率和准确率. Velez 等人设计实现了工具 ConfigCrusher[115], 基于静态程序分析方法确定配置参数影响的代码段, 并通过插桩相关代码段确定配置参数的性能影响, 从而为构建配置性能模型提供详细的局部信息. 为解决 ConfigCrusher 静态分析技术受限于软件规模的问题, Velez 等人进一步提出了 Comprex[116], 通过使用动态污点分析和集成局部性能模型的方法, 首先分别建立局部代码的配置线性模型, 然后将局部模型组合成整个软件的

性能模型. Dorn 等人[117]考虑到现有配置性能建模研究通常将配置对性能的影响表示为确定的标量 (scalar), 忽视了建模过程中由于测试、数据表达、模型选择和训练集数据量等多方面引入的不确定性 (uncertainty), 因此提出了基于贝叶斯推断 (Bayesian inference) 的概率编程 (probabilistic programming) 方法, 通过将概率分布纳入考虑, 从而构建对应的概率分布性能模型, 用于解释导致部分情况下性能模型预测不准确的原因等关键问题. 考虑到已有基于机器学习的配置性能模型构建方法不能重复将模型使用到其他场景, 且模型本身不具备可解释性等局限性, Ding 等人[118]提出了工具 GIL 和 GIL+, GIL 通过建立配置参数与性能之间的线性模型, 使模型具有更强的可重复利用性和可解释性; GIL+则首先构建 context switch rate、cache misses rate 等 5 个底层性能指标与软件性能之间的映射模型, 然后分别构建配置参数与 5 个底层指标的性能模型, 从而根据需求输出性能最优模型. Krishna 等人[119]提出了领头羊环境 (bellwether environment) 的概念, 即对于每个可配置软件而言, 均至少存在一个特定的环境 (由负载、硬件、软件版本三者共同组成) 可以用于构建高质量的迁移学习模型, 也就是指在这种环境下训练构建的配置性能模型可以通过迁移学习较为准确的迁移到其他环境下. 通过实验, Krishna 等人发现发掘领头羊环境最多仅需要评估软件中 10% 的配置空间取值. 基于上述发现, Krishna 等人提出了自动化工具 BEETLE, 第 1 个实现了将迁移学习应用于配置性能模型构建.

部分研究工作[120−123]则充分考虑特定目标软件系统的特征从而构建更加准确的性能模型. Bei 等人[120]针对 MapReduce 进行深入分析, 收集不同处理阶段 (Stage, 包括 Map、Reduce 等)、不同配置取值在不同工作时期 (Phase, 如 Read、Collect 等操作) 的性能表现, 并基于随机森林算法分别建立每个 Phase 对应的回归模型, 然后综合不同 Phase 模型以构建 Stage 性能模型, 最后基于 Stage 性能模型的预测值应用遗传算法找到最优性能表现时的配置取值. Mahgoub 等人[121]主要针对生物信息学领域对于 NoSQL 数据库的高性能需求, 提出了自动化工具 Rafiki. Rafiki 首先基于已有 Benchmark 进行测试, 并使用方差分析 (analysis of variance, ANOVA) 确定性能敏感的配置参数; 然后, Rafiki 对配置空间和负载 (使用 read ratio 和 key reuse distance 两类特征进行描述) 进行采样, 训练基于深度神经网络 (deep neural network, DNN) 的“< 配置, 负载>→性能”三元模型, 并基于遗传算法寻找当前负载下的最优配置. van Aken 等人提出了 OtterTune[122], 基于历史调优数据训练机器学习模型以实现 DBMS 性能调优. OtterTune 基于聚类算法寻找历史调试过程中与当前负载相似的负载, 并进一步使用历史调试过程的配置取值数据训练高斯过程模型, 最终基于模型寻找最优配置. Bao 等人针对分布式消息系统, 提出了一种基于比较的性能模型构建方法 (comparison-based model) 及相应自动化工具 AutoConfig[123]. AutoConfig 通过将不同配置取值下软件性能表现的排序与模型预测的性能排序之间的相对误差的平均值 (mean magnitude of relative error, MMRE), 作为模型训练过程中衡量模型误差的指标, 从而构建更加鲁棒的模型; 此外, AutoConfig 使用加权拉丁超立方采样 (weighted Latin hypercube sampling) 方法以实现对高维配置空间的覆盖率, 并通过多轮 bound-and-search 以选择更优的配置.

文献 [124−128] 单独针对配置空间的采样算法展开研究. Sarkar 等人[124]针对常用的渐进式采样 (progressive sampling) 和投影采样 (projective sampling) 算法进行比较实验, 并基于已有的投影采样方法, 提出了两种启发式的改进方法: 基于 T-wise 采样和基于配置取值频率的启发式方法, 通过使用渐进式的投影采样策略, 通过逼近预测模型准确度的学习曲线来寻找最佳样本量. Jamshidi 等人[125]针对 4 款开源软件在不同配置、不同环境 (硬件、负载) 下的性能表现进行实证调研, 研究迁移学习对于配置性能建模的有效性及适用性. 基于上述实证调研结果, Jamshidi 等人[126]进一步提出了面向知识的跨环境迁移学习策略, 通过学习原有环境中的相关知识, 如性能敏感配置参数、性能分布 (performance distribution) 等, 实现配置空间样本的筛选, 提高模型训练的有效性和准确性, 同时也可以避免直接挪用传统迁移学习的昂贵开销. Nair 等人[127]提出了基于频谱学习的方法来减少配置空间维度, 即通过配置空间中不同配置间的距离矩阵的特征值 (文中用频谱指代, spectrum) 对配置进行聚类, 从而对每个类别中的配置即可以使用一个配置代表整个类别, 从而有效探索配置空间, 以找到影响关键性能特征的配置组合. Kaltenecker 等人[128]针对布尔类型配置参数提出了基于距离度量和概率分布的配置空间采样方法, 基于配置间距离度量的统计概率分布信息尽可能地覆盖不同距离度量下的配置组合, 从而实现在整个配置空间中尽可能均匀的采样配置组合, 并覆盖不同的配置交互, 提升采样的有效性.

总结来看, 目前基于模型的方法根据其主要特点可按照表 1 所示进行归类. 通常来说往往认为线性模型具有更好的可解释性[108], 而非线性模型描述多配置参数间的复杂交互关系能力更强[109]. 而在针对的配置参数类型来说, 部分研究工作由于方法复杂性的限制, 仅能针对布尔类型的配置参数进行建模. 而较先进的方法则针对数值类型配置参数进行了采样等预处理操作, 并在方法上进行改进, 使其处理和适应具有布尔和数值类型配置参数的软件系统. 最后, 在模型训练的数据来源上来说, 大部分研究工作主要通过黑盒性能测试的方法获得不同配置下的软件性能指标, 然后再通过算法训练模型, 少部分研究工作[115,116]结合程序分析技术以白盒方法构建配置与性能间的模型, 其他剩余研究工作[118,119,121,122]则尝试引入诸如底层性能指标[118,122]、环境及负载信息[119,121,122]等额外信息来源构建性能模型.

表 1　基于模型的配置性能调优方法特征分类

分类维度	主要特征	相关文献
模型形式	线性模型	[107,108,115−118]
	非线性模型	[109−114,119−123]
配置参数类型	布尔类型	[107,109−111,116]
	布尔类型+数值类型	[108,112−115,117−123]
模型数据来源	黑盒性能测试	[107−114,117,120,123]
	性能测试+白盒程序分析	[115,116]
	性能测试+灰盒监控分析	[118,119,121,122]

5.2.3　其他配置性能调优方法

除了基于搜索的方法和基于模型的方法外, 仍存在部分研究工作[86,129−131]使用其他方法实现配置性能调优问题. Feng 等人从配置角度出发研究资源竞争导致的软件性能问题, 设计实现了自动化调优工具 Relax[86], 通过监控软件性能的突变点以及系统资源使用率自动识别资源竞争导致的软件性能问题, 并基于程序静态分析和动态测试确定影响资源使用的配置参数列表, 最终基于网络拥塞控制的 SIMD 算法自动调整配置以解决资源竞争, 避免出现性能衰减. Moreno 等人针对全文检索 (text retrieval) 引擎, 如 Lucene, 中诸如迭代次数等配置调优问题进行研究, 提出了基于有监督学习的工具 QUEST (query-based configuration for software retrieval)[129], 通过分析已有检索请求的特征, 然后在有标记的训练集上运行不同配置, 获得每类检索请求的最优配置. 然后, QUEST 基于上述样本训练模型. 当用户输入新检索请求时, QUEST 基于检索请求的特征选择对应的最优配置. Kolesnikov 等人[130]针对两款软件系统进行定性分析, 研究配置参数在代码内部的交互关系 (internal interaction) 是否与软件性能模型中的配置参数之间关联关系 (external interaction) 存在影响关系, 即 internal interaction 是否会导致 external interaction. 本文基于静态分析技术确定代码中配置参数间的控制流交互关系 (control-flow interaction), 并基于已有机器学习算法建立性能模型. 调研结果显示, 基于交互关系预测关联关系可以有效减少配置搜索空间, 但是准确率和查全率仍然较低, 需要综合考虑其他相关因素. Hu 等人设计实现了 Violet[131], 基于静态程序分析和动态符号执行技术, 搜索可能导致程序性能衰减的程序路径及对应的配置取值, 从而避免用户因配置不当而导致的软件性能问题. 具体来说, Violet 首先通过静态分析, 寻找到存在 control dependency 的配置参数集合; 然后, Violet 通过针对存在 control dependency 的配置参数集合进行符号执行, 记录函数调用执行时间来衡量执行开销. 为了探索软件负载对于性能的影响, Violet 总结归纳相关的负载模板以描述不同类型的负载从而方便符号化. 最终通过对比不同配置取值下特定路径的软件执行时间检测会导致性能衰减的配置参数取值, 避免性能问题.

5.2.4　分析与小结

现有配置性能调优相关工作面临的主要挑战仍然是庞大的配置规模和配置取值空间, 通过有效的方法覆盖探索配置空间影响着调优方法的有效性和效率. 此外, 现有工作利用配置空间采样尝试构建准确的模型, 或者使用不同搜索算法探索配置空间, 均依赖于重量级测试的方式, 所需测试样本数量多、性能测试开销大, 且绝大多数无法应对变化的外部环境和工作负载等问题, 特别是在分布式环境下基于微服务架构和容器化环境部署的软件系统.

6 挑战与展望

考虑到现有工作的研究进展与面临的挑战, 本节对未来的研究方向进行展望.

(1) 配置分析与理解方面

针对现有研究缺乏方法指导、领域知识要求高、程序分析受限、环境影响复杂等方面的局限性, 下一步研究工作可以从多个方面展开. 首先, 应考虑提升对于软件配置的重视和地位, 充分发挥配置元级控制和定制软件的作用, 研究系统的配置设计和使用方法, 从而以更好的规范和设施支持软件的演进. 其次, 可以考虑构建广泛适用的常用软件配置文件数据集[12]及配置文件演化历史信息库, 并通过深入挖掘相应特征规律, 从而实现复杂类型配置参数的约束提取. 另一方面, 基于良好的公共数据集, 可以综合评估现有方法的有效性和局限性, 从而推进配置约束提取与挖掘的进一步研究. 对于复杂环境/负载下的配置参数约束提取问题, 建议考虑深入研究配置与负载之间的交互方式和关联关系, 分析配置相关处理代码的特征, 并结合运行时监控、软件运行 trace 信息挖掘等技术手段, 实现复杂动态环境下的配置约束信息提取.

(2) 配置缺陷检测与故障诊断方面

考虑到现有工作测试相关方法存在测试用例不足、测试不充分等问题, 下一步研究工作可以考虑面向配置的测试用例生成研究, 深入研究代码中配置使用的特征模式, 结合程序分析、机器学习、fuzzing 等技术手段生成配置相关的测试用例, 从而提升现有测试对于软件配置的覆盖率, 提高软件配置缺陷的检测能力. 而对于配置故障难以识别和诊断的问题, 建议下一步研究工作考虑从配置处理的根源入手, 研究软件源码中配置出错处理相关代码的自动增强技术, 一方面通过良好的出错处理避免软件因配置问题出现崩溃等严重故障, 另一方面则通过增强相关日志输出, 既提升了配置故障识别的效率, 也有助于指导配置故障诊断和修复. 对于多组件配置关联导致的配置故障和隐式关联导致的配置故障, 可以考虑深入分析配置、资源、环境之间的关联关系, 从资源入手简历配置间的关联关系, 从而实现复杂配置故障的诊断. 此外, 考虑到深度学习技术和智能软件的快速发展与广泛应用, 智能系统中的配置缺陷与故障是否存在新的表现形式和特征模式, 同样值得开展深入研究.

(3) 配置应用方面

考虑到庞大的配置规模和配置取值空间仍是良好高效的配置应用的最大挑战, 下一步研究工作可以考虑进一步重点关注配置自动筛选技术, 结合程序分析、机器学习等手段挖掘配置参数的相关特征, 从而有效筛选功能目标相关的配置参数规模, 减少使用过程中配置取值空间, 有效提升配置应用的效率. 此外, 针对配置性能调优问题, 可以考虑通过程序分析等手段分析性能测试的相关特征模式, 从而充分利用已有测试信息减少不必要的测试过程, 提升相关配置取值下的测试效率, 最终减少测试开销. 进一步的, 建议考虑深入研究如何分析构建配置-负载-资源的关联依赖关系, 从而突破现有工作的局限性, 实现环境感知的配置性能调优方法. 考虑到当前容器化和微服务架构的广泛应用, 建议考虑深入分析微服务架构下容器化软件的相关部署和运行特征, 研究部署环境与软件配置的关联关系, 从而实现集群服务整体的性能调优. 此外, 深度学习的广泛应用促进了智能化时代的发展, 但其庞大的计算开销同样影响着进一步的发展, 因此建议关注智能软件相关的配置性能调优研究, 如超参数的选择与快速调优, 从而有效提升模型训练效率和智能软件的进一步发展.

7 总 结

随着软件系统的复杂化和规模化, 软件配置有效提升了软件功能多样性、系统灵活性和可定制性. 然而, 配置的灵活性也同样为用户提出了巨大的挑战, 因配置引发的各种软件问题逐渐引起人们的关注, 如何帮助用户正确高效使用配置定制软件成为研究的重点. 本文系统地综述了软件运行时配置的研究进展, 从配置分析与理解、配置缺陷检测与故障诊断、配置应用 3 个方面介绍了已有工作如何帮助用户正确高效使用配置. 基于现有工作的研究进展和面临的挑战, 本文总结并展望了未来的研究方向. 本文认为, 通过进一步提升配置理解的有效性和效率, 并深入理解复杂环境下, 特别是云计算分布式环境下的配置相关问题特征, 发掘配置应用的策略和方法, 有助于充分提升软件系统的可靠性、可用性和适应性.

References:

[1] Barroso LA, Hölzle U, Ranganathan P. The Datacenter as a Computer: Designing Warehouse-scale Machines. 3rd ed., Cham: Springer, 2019. [doi: 10.1007/978-3-031-01761-2]

[2] Rabkin A, Katz RH. How Hadoop clusters break. IEEE Software, 2013, 30(4): 88–94. [doi: 10.1109/MS.2012.73]

[3] Amazon Web Services Team. Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region. 2011. http://aws.amazon.com/message/65648

[4] Speed R. That salesforce outage: Global dns downfall started by one engineer trying a quick fix. 2021. https://www.theregister.com/2021/05/19/salesforce_root_cause/

[5] Sloss BT. An update on Sunday's service disruption. 2019. https://cloud.google.com/blog/topics/inside-google-cloud/an-update-on-sundays-service-disruption

[6] Wakefield J. One fastly customer triggered one fastly cusinternet meltdown. 2021. https://www.bbc.com/news/technology-57413224

[7] Yin ZN, Ma X, Zheng J, Zhou YY, Bairavasundaram LN, Pasupathy S. An empirical study on configuration errors in commercial and open source systems. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles. Cascais: ACM, 2011. 159–172. [doi: 10.1145/2043556.2043572]

[8] Sehgal P, Tarasov V, Zadok E. Evaluating performance and energy in file system server workloads. In: Proc. of the 8th USENIX Conf. on File and Storage Technologies. San Jose: USENIX Association, 2010. 253–266.

[9] Xu TY, Zhou YY. Systems approaches to tackling configuration errors: A survey. ACM Computing Surveys, 2015, 47(4): 70. [doi: 10.1145/2791577]

[10] Chen W, Huang X, Qiao XQ, Wei J, Zhong H. Research on software misconfiguration troubleshooting. Ruan Jian Xue Bao/Journal of Software, 2015, 26(6): 1285−1305 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4823.htm [doi: 10.13328/j.cnki.jos.004823]

[11] Sayagh M, Kerzazi N, Adams B, Petrillo F. Software configuration engineering in practice interviews, survey, and systematic literature review. IEEE Trans. on Software Engineering, 2020, 46(6): 646–673. [doi: 10.1109/TSE.2018.2867847]

[12] Xu TY, Jin L, Fan XP, Zhou YY, Pasupathy S, Talwadker R. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 307–319. [doi: 10.1145/2786805.2786852]

[13] Tang CQ, Kooburat T, Venkatachalam P, Chander A, Wen Z, Narayanan A, Dowell P, Karl R. Holistic configuration management at facebook. In: Proc. of the 25th Symp. on Operating Systems Principles. Monterey: ACM, 2015. 328–343. [doi: 10.1145/2815400.2815401]

[14] Sayagh M, Dong Z, Andrzejak A, Adams B. Does the choice of configuration framework matter for developers? Empirical study on 11 Java configuration frameworks. In: Proc. of the 17th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation. Shanghai: IEEE, 2017. 41–50. [doi: 10.1109/SCAM.2017.25]

[15] Zhang YL, He HC, Legunsen O, Li SS, Dong W, Xu TY. An evolutionary study of configuration design and implementation in cloud systems. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 175–176. [doi: 10.1109/ICSE-Companion52605.2021.00075]

[16] Sayagh M, Kerzazi N, Petrillo F, Bennani K, Adams B. What should your run-time configuration framework do to help developers? Empirical Software Engineering, 2020, 25(2): 1259–1293. [doi: 10.1007/s10664-019-09790-x]

[17] Xu TY, Zhang JQ, Huang P, Zheng J, Sheng TW, Yuan D, Zhou YY, Pasupathy S. Do not blame users for misconfigurations. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. Farminton: ACM, 2013. 244–259. [doi: 10.1145/2517349.2522727]

[18] Liao XK, Zhou SL, Li SS, Jia ZY, Liu XD, He HC. Do you really know how to configure your software? Configuration constraints in source code may help. IEEE Trans. on Reliability, 2018, 67(3): 832–846. [doi: 10.1109/TR.2018.2834419]

[19] Chen QR, Wang T, Legunsen O, Li SS, Xu TY. Understanding and discovering software configuration dependencies in cloud and datacenter systems. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. ACM, 2020. 362–374. [doi: 10.1145/3368089.3409727]

[20] Zhang JL, Piskac R, Zhai EN, Xu TY. Static detection of silent misconfigurations with deep interaction analysis. Proc. of the ACM on Programming Languages, 2021, 5: 140. [doi: 10.1145/3485517]

[21] Rabkin A, Katz R. Static extraction of program configuration options. In: Proc. of the 33rd Int'l Conf. on Software Engineering. Honolulu: IEEE, 2011. 131–140. [doi: 10.1145/1985793.1985812]

[22] Dong Z, Andrzejak A, Lo D, Costa D. ORPLocator: Identifying read points of configuration options via static analysis. In: Proc. of the 27th IEEE Int'l Symp. on Software Reliability Engineering. Ottawa: IEEE, 2016. 185–195. [doi: 10.1109/ISSRE.2016.37]

[23] Zhou SL, Liu XD, Li SS, Dong W, Liao XK, Xiong Y. ConfMapper: Automated variable finding for configuration items in source code. In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security Companion. Vienna: IEEE, 2016. 228–235. [doi: 10.1109/QRS-C.2016.35]

[24] Zhang JQ, Renganarayana L, Zhang XL, Ge NY, Bala V, Xu TY, Zhou YY. EnCore: Exploiting system environment and correlation information for misconfiguration detection. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Salt Lake City: ACM, 2014. 687–700. [doi: 10.1145/2541940.2541983]

[25] Chen W, Qiao XQ, Wei J, Zhong H, Huang X. Detecting inter-component configuration errors in proactive: A relation-aware method. In: Proc. of the 14th Int'l Conf. on Quality Software. Allen: IEEE, 2014. 184–189. [doi: 10.1109/QSIC.2014.37]

[26] Chen W, Wu H, Wei J, Zhong H, Huang T. Determine configuration entry correlations for Web application systems. In: Proc. of the 40th IEEE Annual Computer Software and Applications Conf. Atlanta: IEEE, 2016. 42–52. [doi: 10.1109/COMPSAC.2016.95]

[27] Wang T, Chen W, Li J, Liu S, Su L, Zhang W. Association mining based consistent service configuration. Journal of Computer Research and Development, 2020, 57(1): 188–201 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2020.20190079]

[28] Santolucito M, Zhai EN, Piskac R. Probabilistic automated language learning for configuration files. In: Proc. of the 28th Int'l Conf. on Computer Aided Verification. Toronto: Springer, 2016. 80–87. [doi: 10.1007/978-3-319-41540-6_5]

[29] Santolucito M, Zhai EN, Dhodapkar R, Shim A, Piskac R. Synthesizing configuration file specifications with association rule learning. Proc. of the ACM on Programming Languages, 2017, 1: 64. [doi: 10.1145/3133888]

[30] Bhagwan R, Mehta S, Radhakrishna A, Garg S. Learning patterns in configuration. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 817–828. [doi: 10.1109/ASE51524.2021.9678525]

[31] Mehta S, Bhagwan R, Kumar R, Bansal C, Maddila C, Ashok B, Asthana S, Bird C, Kumar A. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 435–448.

[32] Tuncer O, Bila N, Duri S, Isci C, Coskun AK. ConfEx: Towards automating software configuration analytics in the cloud. In: Proc. of the 48th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks Workshops. Luxembourg: IEEE, 2018. 30–33. [doi: 10.1109/DSN-W.2018.00019]

[33] Xu XY, Li SS, Guo Y, Dong W, Li W, Liao XK. Automatic type inference for proactive misconfiguration prevention. In: Proc. of the 29th Int'l Conf. on Software Engineering and Knowledge Engineering. Pittsburgh: KSI Research Inc., 2017. 295–300. [doi: 10.18293/SEKE2017-072]

[34] Xiang CC, Huang HC, Yoo A, Zhou YY, Pasupathy S. PracExtractor: Extracting configuration good practices from manuals to detect server misconfigurations. In: Proc. of the 2020 USENIX Annual Technical Conf. USENIX Association, 2020. 18.

[35] Zhou SL, Liu XD, Li SS, Jia ZY, Zhang YL, Wang T, Li W, Liao XK. ConfInLog: Leveraging software logs to infer configuration constraints. In: Proc. of the 29th IEEE/ACM Int'l Conf. on Program Comprehension. Madrid: IEEE, 2021. 94–105. [doi: 10.1109/ICPC52881.2021.00018]

[36] Song C, Porter A, Foster JS. iTree: Efficiently discovering high-coverage configurations using interaction trees. IEEE Trans. on Software Engineering, 2014, 40(3): 251–265. [doi: 10.1109/TSE.2013.55]

[37] Nguyen T, Koc U, Cheng J, Foster JS, Porter AA. iGen: Dynamic interaction inference for configurable software. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 655–665. [doi: 10.1145/2950290.2950311]

[38] Nguyen K, Nguyen T. GenTree: Using decision trees to learn interactions for configurable software. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1598–1609. [doi: 10.1109/ICSE43902.2021.00142]

[39] Lillack M, Kästner C, Bodden E. Tracking load-time configuration options. In: Proc. of the 29th IEEE/ACM Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 445–456. [doi: 10.1145/2642937.2643001]

[40] Lillack M, Kästner C, Bodden E. Tracking load-time configuration options. IEEE Trans. on Software Engineering, 2018, 44(12): 1269–1291. [doi: 10.1109/TSE.2017.2756048]

[41] Meinicke J, Wong CP, Kästner C, Thüm T, Saake G. On essential configuration complexity: Measuring interactions in highly-configurable systems. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 483–494. [doi: 10.1145/2970276.2970322]

[42] Angerer F, Grimmer A, Prähofer H, Grünbacher P. Configuration-aware change impact analysis. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 385–395. [doi: 10.1109/ASE.2015.58]

[43] Han X, Yu TT. An empirical study on performance bugs for highly configurable software systems. In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Ciudad Real: ACM, 2016. 23. [doi: 10.1145/2961111.2962602]

[44] Behrang F, Cohen MB, Orso A. Users beware: Preference inconsistencies ahead. In: Proc. of the 10th Joint Meeting on Foundations of

Software Engineering. Bergamo: ACM, 2015. 295–306. [doi: 10.1145/2786805.2786869]

[45] Toman J, Grossman D. Staccato: A bug finder for dynamic configuration updates. In: Proc. of the 30th European Conf. on Object-oriented Programming. Rome, 2016. 24:1–24:25. [doi: 10.4230/LIPIcs.ECOOP.2016.24]

[46] Toman J, Grossman D. Legato: An at-most-once analysis with applications to dynamic configuration updates. In: Proc. of the 32nd European Conf. on Object-oriented Programming. Amsterdam, 2018. 24:1–24:32. [doi: 10.4230/LIPIcs.ECOOP.2018.24]

[47] He HC, Jia ZY, Li SS, Xu EC, Yu TT, Yu Y, Wang J. Liao XK. CP-Detector: Using configuration-related performance properties to expose performance bugs. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 623–634.

[48] Velez M, Jamshidi P, Siegmund N, Apel S, Kästner C. On debugging the performance of configurable software systems: Developer needs and tailored tool support. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1571–1583. [doi: 10.1145/3510003.3510043]

[49] Arshad FA, Krause RJ, Bagchi S. Characterizing configuration problems in Java EE application servers: An empirical study with GlassFish and JBoss. In: Proc. of the 24th IEEE Int'l Symp. on Software Reliability Engineering. Pasadena: IEEE, 2013. 198–207. [doi: 10.1109/ISSRE.2013.6698919]

[50] Zhang S, Ernst MD. Proactive detection of inadequate diagnostic messages for software configuration errors. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. Baltimore: ACM, 2015. 12–23. [doi: 10.1145/2771783.2771817]

[51] Li SS, Li W, Liao XK, Peng SL, Zhou SL, Jia ZY, Wang T. ConfVD: System reactions analysis and evaluation through misconfiguration injection. IEEE Trans. on Reliability, 2018, 67(4): 1393–1405. [doi: 10.1109/TR.2018.2865962]

[52] Li W, Li SS, Liao XK, Xu XY, Zhou SL, Jia ZY. ConfTest: Generating comprehensive misconfiguration for system reaction ability evaluation. In: Proc. of the 21st Int'l Conf. on Evaluation and Assessment in Software Engineering. Karlskrona: ACM, 2017. 88–97. [doi: 10.1145/3084226.3084244]

[53] Li W, Jia ZY, Li SS, Zhang YL, Wang T, Xu EC, Wang J, Liao XK. Challenges and opportunities: An in-depth empirical study on configuration error injection testing. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 478–490. [doi: 10.1145/3460319.3464799]

[54] Jin DP, Qu X, Cohen MB, Robinson B. Configurations everywhere: Implications for testing and debugging in practice. In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 215–224. [doi: 10.1145/2591062.2591191]

[55] Qu X, Acharya M, Robinson B. Impact analysis of configuration changes for test case selection. In: Proc. of the 22nd IEEE Int'l Symp. on Software Reliability Engineering. Hiroshima: IEEE, 2011. 140–149. [doi: 10.1109/ISSRE.2011.9]

[56] Qu X, Acharya M, Robinson B. Configuration selection using code change impact analysis for regression testing. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance. Trento: IEEE, 2012. 129–138. [doi: 10.1109/ICSM.2012.6405263]

[57] Marijan D, Liaaen M, Gotlieb A, Sen S, Ieva C. TITAN: Test suite optimization for highly configurable software. In: Proc. of the 2017 IEEE Int'l Conf. on Software Testing, Verification and Validation. Tokyo: IEEE, 2017. 524–531. [doi: 10.1109/ICST.2017.60]

[58] Eshete B, Villafiorita A, Weldemariam K, Zulkernine M. Confeagle: Automated analysis of configuration vulnerabilities in Web applications. In: Proc. of the 7th IEEE Int'l Conf. on Software Security and Reliability. Gaithersburg: IEEE, 2013. 188–197. [doi: 10.1109/SERE.2013.30]

[59] Otsuka H, Watanabe Y, Matsumoto Y. Learning from before and after recovery to detect latent misconfiguration. In: Proc. of the 39th IEEE Annual Computer Software and Applications Conf. Taichung: IEEE, 2015. 141–148. [doi: 10.1109/COMPSAC.2015.222]

[60] Huang P, Bolosky WJ, Singh A, Zhou YY. ConfValley: A systematic configuration validation framework for cloud services. In: Proc. of the 10th European Conf. on Computer Systems. Bordeaux: ACM, 2015. 19. [doi: 10.1145/2741948.2741963]

[61] Baset S, Suneja S, Bila N, Tuncer O, Isci C. Usable declarative configuration specification and validation for applications, systems, and cloud. In: Proc. of the 18th ACM/IFIP/USENIX Middleware Conf.: Industrial Track. Las Vegas: ACM, 2017. 29–35. [doi: 10.1145/3154448.3154453]

[62] Xu TY, Jin XX, Huang P, Zhou YY, Lu S, Jin L, Pasupathy S. Early detection of configuration errors to reduce failure damage. In: Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation. Savannah: USENIX Association, 2016. 619–634.

[63] Sun XD, Cheng RX, Chen JY, Ang E, Legunsen O, Xu TY. Testing configuration changes in context to prevent production failures. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX Association, 2020. 42.

[64] Cheng RX, Zhang LM, Marinov D, Xu TY. Test-case prioritization for configuration testing. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 452–465. [doi: 10.1145/3460319.3464810]

[65] Xia X, Lo D, Qiu WW, Wang XG, Zhou B. Automated configuration bug report prediction using text mining. In: Proc. of the 38th IEEE Annual Computer Software and Applications Conf. Vasteras: IEEE, 2014. 107–116. [doi: 10.1109/COMPSAC.2014.17]

[66] Xu BW, Lo D, Xia X, Sureka A, Li SP. EFSPredictor: Predicting configuration bugs with ensemble feature selection. In: Proc. of the 2015 Asia-Pacific Software Engineering Conf. New Delhi: IEEE, 2015. 206–213. [doi: 10.1109/APSEC.2015.38]

[67] Wen W, Yu TT, Hayes JH. CoLUA: Automatically predicting configuration bug reports and extracting configuration options. In: Proc. of the 27th Int'l Symp. on Software Reliability Engineering. Ottawa: IEEE, 2016. 150–161. [doi: 10.1109/ISSRE.2016.29]

[68] Yuan D, Xie YL, Panigrahy R, Yang JF, Verbowski C, Kumar A. Context-based online configuration-error detection. In: Proc. of the 2011 USENIX Annual Technical Conf. Portland: USENIX Association, 2011. 28.

[69] Nevill-Manning CG, Witten IH. Identifying hierarchical structure in sequences: A linear-time algorithm. Journal of Artificial Intelligence Research, 1997, 7: 67–82. [doi: 10.1613/jair.374]

[70] Rabkin A, Katz R. Precomputing possible configuration error diagnoses. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lawrence: IEEE, 2011. 193–202. [doi: 10.1109/ASE.2011.6100053]

[71] Rabkin A, Katz R. Using static analysis to diagnose configuration errors. In: Proc. of the 11th Int'l Symp. on Software Testing and Analysis. Toronto: ACM, 2011.

[72] Dong Z, Andrzejak A, Shao K. Practical and accurate pinpointing of configuration errors using static analysis. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 171–180. [doi: 10.1109/ICSM.2015.7332463]

[73] Dong Z, Ghanavati M, Andrzejak A. Automated diagnosis of software misconfigurations based on static analysis. In: Proc. of the 2013 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Pasadena: IEEE, 2013. 162–168. [doi: 10.1109/ISSREW.2013.6688897]

[74] Zhang S, Ernst MD. Automated diagnosis of software configuration errors. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 312–321. [doi: 10.1109/ICSE.2013.6606577]

[75] Zhang S, Ernst MD. Which configuration option should I change? In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 152–163. [doi: 10.1145/2568225.2568251]

[76] Sayagh M, Adams B. Multi-layer software configuration: Empirical study on wordpress. In: Proc. of the 15th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation. Bremen: IEEE, 2015. 31–40. [doi: 10.1109/SCAM.2015.7335399]

[77] Sayagh M, Kerzazi N, Adams B. On cross-stack configuration errors. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 255–265. [doi: 10.1109/ICSE.2017.31]

[78] Attariyan M, Chow M, Flinn J. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In: Proc. of the 10th USENIX Symp. on Operating Systems Design and Implementation. Hollywood: USENIX Association, 2012. 307–320.

[79] Kannan K, Bhamidipaty A. Nail-it-down: Nailing and fixing configuration faults in cloud environments. In: Proc. of the 2013 ACM Int'l Conf. on Computing Frontiers. Ischia: ACM, 2013. 22. [doi: 10.1145/2482767.2482796]

[80] Wang T, Liu XD, Li SS, Liao XK, Li W, Liao Q. MisconfDoctor: Diagnosing misconfiguration via log-based configuration testing. In: Proc. of the 2018 IEEE Int'l Conf. on Software Quality, Reliability and Security. Lisbon: IEEE, 2018. 1–12. [doi: 10.1109/QRS.2018.00014]

[81] Han X, Yu TT, Lo D. PerfLearner: Learning from bug reports to understand and generate performance test frames. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 17–28. [doi: 10.1145/3238147.3238204]

[82] Rich CR. Taddm's flexible approach to discovery. Technical Report, IBM Corporation, 2003.

[83] Huang Z, Lie D. Ocasta: Clustering configuration settings for error recovery. In: Proc. of the 44th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Atlanta: IEEE, 2014. 479–490. [doi: 10.1109/DSN.2014.51]

[84] Potharaju R, Chan J, Hu LH, Nita-Rotaru C, Wang MS, Zhang LY, Jain N. ConfSeer: Leveraging customer support knowledge bases for automated misconfiguration detection. Proc. of the VLDB Endowment, 2015, 8(12): 1828–1839. [doi: 10.14778/2824032.2824079]

[85] Talwadker R. Dexter: Faster troubleshooting of misconfiguration cases using system logs. In: Proc. of the 10th ACM Int'l Systems and Storage Conf. Haifa: ACM, 2017. 7. [doi: 10.1145/3078468.3078484]

[86] Feng ZM, Li SS, Liao XK, Liu XD, Li YF, Zhou SL. Relax: Automatic contention detection and resolution for configuration related performance tuning. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 239–248. [doi: 10.1109/APSEC.2018.00038]

[87] Jin DP, Cohen MB, Qu X, Robinson B. PrefFinder: Getting the right preference in configurable software systems. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated software engineering. Vasteras: ACM, 2014. 151–162. [doi: 10.1145/2642937.2643009]

[88] Sayagh M, Hassan AE. ConfigMiner: Identifying the appropriate configuration options for config-related user questions by mining online forums. IEEE Trans. on Software Engineering, 2021, 47(12): 2907–2918. [doi: 10.1109/TSE.2020.2973997]

[89] Hamidi S, Andritsos P, Liaskos S. Constructing adaptive configuration dialogs using crowd data. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 485–490. [doi: 10.1145/2642937.2642960]

[90] Zhou CL, Liu HR, Zhang YL, Xue ZP, Liao Q, Zhao JJ, Wang J. Deep understanding of runtime configuration intention. Int'l Journal of Software Engineering and Knowledge Engineering, 2021, 31(6): 775–802. [doi: 10.1142/S0218194021500236]

[91] Cao Z, Kuenning G, Zadok E. Carver: Finding important parameters for storage system tuning. In: Proc. of the 18th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2020. 43–58.

[92] Kanellis K, Alagappan R, Venkataraman S. Too many knobs to tune? Towards faster database tuning by pre-selecting important knobs. In: Proc. of the 12th USENIX Workshop on Hot Topics in Storage and File Systems. USENIX Association, 2020. 16.

[93] Li C, Wang S, Hoffmann H, Lu S. Statically inferring performance properties of software configurations. In: Proc. of the 15th European Conf. on Computer Systems. Heraklion: ACM, 2020. 10. [doi: 10.1145/3342195.3387520]

[94] He HC, Jia ZY, Li SS, Yu Y, Zhou CL, Liao Q, Wang J, Liao XK. Multi-intention-aware configuration selection for performance tuning. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1431–1442. [doi: 10.1145/3510003.3510094]

[95] Zhu YQ, Liu JX, Guo MY, Bao YG, Ma WL, Liu ZY, Song KP, Yang YC. BestConfig: Tapping the performance potential of systems via automatic configuration tuning. In: Proc. of the 2017 Symp. on Cloud Computing. Santa Clara: ACM, 2017. 338–350. [doi: 10.1145/3127479.3128605]

[96] Wang S, Li C, Hoffmann H, Lu S, Sentosa W, Kistijantoro AI. Understanding and auto-adjusting performance-sensitive configurations. ACM SIGPLAN Notices, 2018, 53(2): 154–168. [doi: 10.1145/3296957.3173206]

[97] Bao L, Liu X, Wang FZ, Fang BY. ACTGAN: Automatic configuration tuning for software systems with generative adversarial networks. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 465–476. [doi: 10.1109/ASE.2019.00051]

[98] Chen T, Li MQ. Multi-objectivizing software configuration tuning. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 453–465. [doi: 10.1145/3468264.3468555]

[99] Lengauer P, Mössenböck H. The taming of the shrew: Increasing performance by automatic parameter tuning for Java garbage collectors. In: Proc. of the 5th ACM/SPEC Int'l Conf. on Performance Engineering. Dublin: ACM, 2014. 111–122. [doi: 10.1145/2568088.2568091]

[100] Li M, Zeng LZ, Meng SC, Tan J, Zhang L, Butt AR, Fuller N. MRONLINE: MapReduce online performance tuning. In: Proc. of the 23rd Int'l Symp. on High-performance Parallel and Distributed Computing. Vancouver: ACM, 2014. 165–176. [doi: 10.1145/2600212.2600229]

[101] Ding XA, Liu Y, Qian DP. JellyFish: Online performance tuning with adaptive configuration and elastic container in Hadoop yarn. In: Proc. of the 21st IEEE Int'l Conf. on Parallel and Distributed Systems. Melbourne: IEEE, 2015. 831–836. [doi: 10.1109/ICPADS.2015.112]

[102] Singh R, Bezemer CP, Shang W, Hassan AE. Optimizing the performance-related configurations of object-relational mapping frameworks using a multi-objective genetic algorithm. In: Proc. of the 7th ACM/SPEC on Int'l Conf. on Performance Engineering. Delft: ACM, 2016. 309–320. [doi: 10.1145/2851553.2851576]

[103] Jamshidi P, Casale G. An uncertainty-aware approach to optimal configuration of stream processing systems. In: Proc. of the 24th IEEE Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. London: IEEE, 2016. 39–48. [doi: 10.1109/MASCOTS.2016.17]

[104] Bilal M, Canini M. Towards automatic parameter tuning of stream processing systems. In: Proc. of the 2017 Symp. on Cloud Computing. Santa Clara: ACM, 2017. 189–200. [doi: 10.1145/3127479.3127492]

[105] Mahgoub A, Wood P, Medoff A, Mitra S, Meyer F, Chaterji S, Bagchi S. SOPHIA: Online reconfiguration of clustered NoSQL databases for time-varying workloads. In: Proc. of the 2019 USENIX Annual Technical Conf. Renton: USENIX Association, 2019. 223–239.

[106] Krishna R, Tang C, Sullivan K, Ray B. ConEx: Efficient exploration of big-data system configurations for better performance. IEEE Trans. on Software Engineering, 2022, 48(3): 893–909. [doi: 10.1109/TSE.2020.3007560]

[107] Siegmund N, Kolesnikov SS, Kästner C, Apel S, Batory D, Rosenmüller M, Saake G. Predicting performance via automated feature-interaction detection. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE, 2012. 167–177. [doi: 10.1109/ICSE.2012.6227196]

[108] Siegmund N, Grebhahn A, Apel S, Kästner C. Performance-influence models for highly configurable systems. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 284–294. [doi: 10.1145/2786805.2786845]

[109] Guo JM, Czarnecki K, Apel S, Siegmund N, Wąsowski A. Variability-aware performance prediction: A statistical learning approach. In:

Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering. Silicon Valley: IEEE, 2013. 301–311. [doi: 10.1109/ASE.2013.6693089]

[110] Guo JM, Yang DY, Siegmund N, Apel S, Sarkar A, Valov P, Czarnecki K, Wasowski A, Yu HQ. Data-efficient performance learning for configurable systems. Empirical Software Engineering, 2018, 23(3): 1826–1867. [doi: 10.1007/s10664-017-9573-6]

[111] Zhang Y, Guo JM, Blais E, Czarnecki K. Performance prediction of configurable software systems by fourier learning (T). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 365–373. [doi: 10.1109/ASE.2015.15]

[112] Nair V, Menzies T, Siegmund N, Apel S. Using bad learners to find good configurations. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 257–267. [doi: 10.1145/3106237.3106238]

[113] Nair V, Yu Z, Menzies T, Siegmund N, Apel S. Finding faster configurations using FLASH. IEEE Trans. on Software Engineering, 2020, 46(7): 794–811. [doi: 10.1109/TSE.2018.2870895]

[114] Ha H, Zhang HY. DeepPerf: Performance prediction for configurable software with deep sparse neural network. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1095–1106. [doi: 10.1109/ICSE.2019.00113]

[115] Velez M, Jamshidi P, Sattler F, Siegmund N, Apel S, Kästner C. ConfigCrusher: Towards white-box performance analysis for configurable systems. Automated Software Engineering, 2020, 27(3): 265–300. [doi: 10.1007/s10515-020-00273-8]

[116] Velez M, Jamshidi P, Siegmund N, Apel S, Kästner C. White-box analysis over machine learning: Modeling performance of configurable systems. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1072–1084. [doi: 10.1109/ICSE43902.2021.00100]

[117] Dorn J, Apel S, Siegmund N. Mastering uncertainty in performance estimations of configurable software systems. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 684–696.

[118] Ding Y, Pervaiz A, Carbin M, Hoffmann H. Generalizable and interpretable learning for configuration extrapolation. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 728–740. [doi: 10.1145/3468264.3468603]

[119] Krishna R, Nair V, Jamshidi P, Menzies T. Whence to learn? Transferring knowledge in configurable systems using BEETLE. IEEE Trans.on Software Engineering, 2021, 47(12): 2956–2972. [doi: 10.1109/TSE.2020.2983927]

[120] Bei ZD, Yu ZB, Zhang HL, Xiong W, Xu CZ, Eeckhout L, Feng SZ. RFHOC: A random-forest approach to auto-tuning Hadoop's configuration. IEEE Trans. on Parallel and Distributed Systems, 2016, 27(5): 1470–1483. [doi: 10.1109/TPDS.2015.2449299]

[121] Mahgoub A, Wood P, Ganesh S, Mitra S, Gerlach W, Harrison T, Meyer F, Grama A, Bagchi S, Chaterji S. Rafiki: A middleware for parameter tuning of NoSQL datastores for dynamic metagenomics workloads. In: Proc. of the 18th ACM/IFIP/USENIX Middleware Conf. Las Vegas: ACM, 2017. 28–40. [doi: 10.1145/3135974.3135991]

[122] van Aken D, Pavlo A, Gordon GJ, Zhang BH. Automatic database management system tuning through large-scale machine learning. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1009–1024. [doi: 10.1145/3035918.3064029]

[123] Bao L, Liu X, Xu ZH, Fang BY. AutoConfig: Automatic configuration tuning for distributed message systems. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 29–40. [doi: 10.1145/3238147.3238175]

[124] Sarkar A, Guo JM, Siegmund N, Apel S, Czarnecki K. Cost-efficient sampling for performance prediction of configurable systems (T). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 342–352. [doi: 10.1109/ASE.2015.45]

[125] Jamshidi P, Siegmund N, Velez M, Kästner C, Patel A, Agarwal Y. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 497–508. [doi: 10.1109/ASE.2017.8115661]

[126] Jamshidi P, Velez M, Kästner C, Siegmund N. Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 71–82. [doi: 10.1145/3236024.3236074]

[127] Nair V, Menzies T, Siegmund N, Apel S. Faster discovery of faster system configurations with spectral learning. Automated Software Engineering, 2018, 25(2): 247–277. [doi: 10.1007/s10515-017-0225-2]

[128] Kaltenecker C, Grebhahn A, Siegmund N, Guo JM, Apel S. Distance-based sampling of software configuration spaces. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1084–1094. [doi: 10.1109/ICSE.2019.00112]

[129] Moreno L, Bavota G, Haiduc S, Di Penta M, Oliveto R, Russo B, Marcus A. Query-based configuration of text retrieval solutions for software engineering tasks. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 567–578. [doi: 10.1145/2786805.2786859]

[130] Kolesnikov S, Siegmund N, Kästner C, Apel S. On the relation of control-flow and performance feature interactions: A case study.

Empirical Software Engineering, 2019, 24(4): 2410–2437. [doi: 10.1007/s10664-019-09705-w]

[131] Hu YG, Huang GQ, Huang P. Automated reasoning and detection of specious configuration in large systems with symbolic execution. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX Association, 2020. 41.

附中文参考文献:

[10] 陈伟, 黄翔, 乔晓强, 魏峻, 钟华. 软件配置错误诊断与修复技术研究. 软件学报, 2015, 26(6): 1285–1305. http://www.jos.org.cn/1000-9825/4823.htm [doi: 10.13328/j.cnki.jos.004823]

[27] 王焘, 陈伟, 李娟, 刘绍华, 苏林刚, 张文博. 一种基于关联挖掘的服务一致化配置方法. 计算机研究与发展, 2020, 57(1): 188–201. [doi: 10.7544/issn1000-1239.2020.20190079]

周书林(1992－), 男, 博士, 助理研究员, 主要研究领域为软件可靠性.

王戟(1969－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为软件方法学, 软件分析与验证, 并行与分布计算.

李姗姗(1980－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件可信演化, 智能化软件开发.

廖湘科(1963－), 男, 博士, 研究员, 博士生导师, CCF 会士, 主要研究领域为操作系统, 高性能计算.

董威(1976－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析与验证, 软件度量与评估.