计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(7):1−12 [doi: 10.15888/j.cnki.csa.007973]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

软件系统配置研究综述①

陈　艳[1], 叶宏杰[2,3], 陈　伟[2,3]

[1](中国电子科技集团公司第十五研究所, 北京 100083)
[2](中国科学院大学, 北京 100049)
[3](中国科学院 软件研究所, 北京 100190)
通讯作者: 叶宏杰, E-mail: yehongjie19@otcaix.iscas.ac.cn

摘　要: 随着软件系统规模和复杂度的不断提升, 软件配置已经成为软件工程领域中的一个重要话题. 大量、复杂的配置项为正确使用软件系统带来了极大的困难, 例如, 配置错误会影响系统性能, 并带来严重损失. 软件系统配置技术得到广泛关注, 并取得了众多的研究成果. 本文对软件配置领域的研究现状和主要成果进行分析和综述. 文章首先提出了基于软件生命周期和技术手段两个维度的软件配置相关工作分析框架, 然后基于该框架对当前主要研究成果进行分类总结和分析评价, 最后总结软件配置领域的工作特点, 探讨未来可能的研究热点, 对于今后该领域的深入研究具有一定借鉴意义.

关键词: 软件系统配置; 配置错误; 配置管理; 分析框架; 软件生命周期; 综述

引用格式: 陈艳,叶宏杰,陈伟.软件系统配置研究综述.计算机系统应用,2021,30(7):1–12. http://www.c-s-a.org.cn/1003-3254/7973.html

Survey on Software System Configuration

CHEN Yan[1], YE Hong-Jie[2,3], CHEN Wei[2,3]

[1](The 15th Research Institute of China Electronic Technology Group Corporation, Beijing 100083, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Amid the growing scale and complexity of software systems, their configuration has become an essential topic in the field of software engineering. Massive complicated configuration entries bring difficulties in correctly deploying and using software systems; for instance, misconfigurations will lead to performance degradation and significant losses. Researchers are devoted to handling software configuration, mainly for resolving software misconfiguration. This paper presents a systematic literature review on the work of software configuration, including the research status and main achievements. It first proposes a two-dimensional analysis framework for the research work from the perspectives of software lifecycle and techniques. Based on this framework, this paper analyzes and categorizes the state-of-the-art results. Finally, it summarizes the characteristics of the current work and envisions beneficial prospects of future work.

Key words: software system configuration; misconfiguration; configuration management; analysis framework; software lifecycle; survey

当前软件系统更加趋于具有高可配置性 (high configurability) 的特点, 目标在于提升系统的灵活性和可变性, 使开发人员和系统管理人员能够在不修改程序代码的前提下通过改变配置来控制系统的行为以及改变系统的特性. 软件配置是指以用户需求和软件的功能、结构及主要特性等为依据, 选择和确定相关硬

① 基金项目: 装备发展部“十三五”预研课题 (31505303)
Foundation item: 13th Five-Year Plan Pre-research Project of Equipment Development Department (31505303)
收稿时间: 2020-10-24; 修改时间: 2020-11-23; 采用时间: 2020-12-01; csa 在线出版时间: 2021-06-30

件、软件和固件的型号、版本及数量, 规划软件放置位置和关联关系, 设置软件系统相关参数值等; 狭义上的软件配置主要指对软件系统配置参数取值的设置[1].

随着信息技术与互联网技术的发展, 软件系统 (尤其是网络分布式系统) 的规模和复杂度不断提升, 其在功能和非功能方面特性的设置和定制化往往是通过对配置项的不同取值来实现的, 由此也使得复杂系统的配置项数量越来越多、复杂程度越来越高.

但是, 大量配置参数在提高系统灵活性的同时也为应用的配置正确性带来困难[1], 为系统后续运行的可靠性、可用性、易用性以及系统性能等服务质量带来极大挑战,具体包括:

(1) 大量配置项给用户的使用和操作带来困难和干扰. 不恰当的、数量过多的配置项设计, 以及缺少系统的、详细的配置参数使用说明, 会使用户难以理解配置项与系统特性之间的关联性, 因此也很难通过正确调节配置来使系统达到预期效果. Xu 等人实证研究发现, 大部分用户只会修改系统 6.1%~16.7% 的配置项[2]. Yin 等人也指出, 有 63.6% 以上的配置错误是在用户初次修改配置项时发生的[3].

(2) 配置错误已经成为导致软件系统错误和运行故障的重要因素之一[4]. 配置错误是指由于配置项取值设置不当而导致的软件系统错误和运行故障, 包括参数错误、兼容性错误和组件错误等[3]. 软件配置错误往往带来灾难性后果. Barroso 等人发现配置错误是 Google 一个主要服务运行错误的主要原因之一[5]. 更严重地, 在 2009 年, 配置错误导致整个“.se”域瘫痪了一个多小时, 影响了近百万台主机的正常工作.

(3) 软件配置与系统性能紧密相关, 设置不当将严重影响系统的服务质量. 软件系统的资源需求往往以配置参数的形式进行设置, 以提高系统的灵活性和可配置性, 如缓冲区大小、最长响应时间、最大连接数等. 此类配置参数如果取值不恰当, 将会严重影响系统的性能, 例如将数据库最大连接数设置过小会显著降低系统的并发能力, 而如何合理设置此类配置参数, 使得系统性能达到最优也是一个十分困难的问题.

综上, 软件配置已经成为了一个备受关注的话题, 并且已存在诸多的软件配置相关研究工作. 本文旨在对已有的研究工作进行梳理, 系统化地分析相关工作, 为今后的软件配置研究提供线索依据和发展方向.

本文的组织安排如下: 第 1 节首先提出了一个面向软件配置的分析框架, 该框架从软件生命周期和软件配置技术手段两个维度对软件配置相关工作进行划分和分析评价. 第 2 节简要介绍了本文对软件配置相关文献的检索和筛选方法与过程. 第 3 节分别从软件生命周期中的设计阶段、开发阶段和后开发阶段 (即测试阶段、部署阶段和生产阶段) 中所面对的软件配置问题以及当前代表性的相关工作进行介绍和分析, 以及其他对典型软件系统进行分析的研究工作. 第 4 节对软件系统配置研究现状进行总结, 分析各类型研究工作的特点, 并对软件配置的未来研究方向作出展望. 第 5 节总结全文.

1 软件配置分析框架

软件配置活动存在于软件生命周期的各个阶段, 本文从软件开发的生命周期, 即设计阶段、开发阶段、测试阶段、部署阶段和生产阶段着手, 对现有改善软件配置设计、减轻不良配置影响以及应对配置错误的相关工作进行分析和综述. 除此之外, 软件配置分析的技术手段包括程序分析和统计分析等, 本文也根据采用的技术手段对现有软件配置分析方法进行分类.

本文首先建立基于软件生命周期的分类方法. 软件配置是软件系统中的重要组成部分, 软件配置的设计、开发、测试和维护包含于软件系统的设计、开发、测试和维护过程中. 因此, 从软件生命周期着手, 在各个阶段中寻求软件配置面临的问题以及相应的解决方法, 可以有效提升软件配置质量. 软件生命周期可以划分为设计阶段、开发阶段和后开发阶段, 其中后开发阶段即测试阶段、部署阶段和生产阶段. 在上述各个阶段中, 软件配置面临的主要问题如下:

(1) 设计阶段中的软件配置问题主要是如何进行配置项的选择和设计, 即如何在提高软件的配置性以及减少配置参数数量上进行权衡, 从而通过清晰、有限的配置参数来实现软件的高可配置. 配置项的选定即明确系统的可配置点, 指出哪些功能需要以可配置的形式展现给软件系统的使用者. 配置项的设计即确定配置项在系统中的表示形式, 包括配置项的名称、类型、存储形式甚至使用的语言等[6].

(2) 开发阶段中的软件配置问题主要是如何进行配置项的实现和管理. 配置项实现需要关注配置项的约束和用户配置手册的编写. 配置项的管理即维护已经实现的配置项, 并有效应对其变更.

(3) 后开发阶段中的软件配置问题主要是配置错误处理和配置性能分析. 配置错误处理即应对软件配置带来的系统错误, 包括配置错误的检测、诊断和修复等. 配置性能分析即分析软件配置与系统性能间的关系, 包括配置性能间关系分析以及性能优化等.

其次, 本文以软件配置研究工作中所采取的主要技术手段为依据, 对现有的研究工作进行分类分析, 主要包括白盒方法 (white-box) 和黑盒方法 (black-box) 两种, 以及两种方法的综合使用:

(1) 白盒方法的适用前提是软件系统的源代码 (或字节码) 可获取, 从而能够将目标系统作为白盒处理. 采用静态程序分析或动态程序分析技术, 分析代码中配置项的读入点、控制流和数据流等, 找到软件配置与软件系统本身的关系, 进而解决具体问题.

(2) 黑盒方法适用于无法获得软件系统源码的应用场景, 该方法主要基于统计方法、对比方法、机器学习等技术, 通过运行一定数量的测试用例, 观察目标系统在不同条件下的执行结果, 总结和发现软件配置与软件系统的内在关系, 进而解决具体问题.

基于上述观点和视角, 本文以软件生命周期作为文献分类的主方法, 简述各项软件配置相关的研究工作; 当工作涉及到软件配置与软件系统关系时, 本文将指出其基于的主要技术手段.

2 文献检索与筛选

为了保证综述的全面性和时效性, 本文采用的检索策略如下:

(1) 将研究工作的检索范围设定在近 7 年, 即 2014 年~2020 年.

(2) 以软件配置 (software configuration) 为第一关键词, 软件配置相关的活动, 如: 设计 (design)、管理 (management)、性能 (performance), 为第二关键词, 组合成若干关键词词组.

(3) 使用 (2) 中的词组, 分别使用谷歌学术和中国知网搜索引擎进行搜索, 并按照相关性排序、选取相关性最高的前 20 条结果. 再以人工的方式进行去重、筛去不符合需求的文章.

(4) 使用 (2) 中的词组, 在多个文献数据库中, 以与软件配置相关的期刊和会议为范围进行检索, 再以人工的方式进行去重, 筛去不符合需求的文章.

(5) 分析 (2)、(3) 步获得文章的参考文献, 选取在时间范围内且与主题密切相关的文章, 并重复该步骤.

(6) 综合以上 (3)~(5) 步结果, 构成本文检索范围.

最终, 本文选取了 32 篇文章作为本综述的范围. 文章的发表时间分布如图 1 所示. 从图中可以看出, 文章的发表时间分布较为均匀, 表明软件配置问题是软件工程领域一个持续得到关注和研究的主题. 文章的类型分布如图 2 所示. 大量研究工作集中在配置错误的处理上, 关注配置项选定和设计的工作较少. 本文认为, 这是因为与此相关的工作被包含于软件系统设计的工作当中, 而软件系统设计相关的研究已经较为成熟.

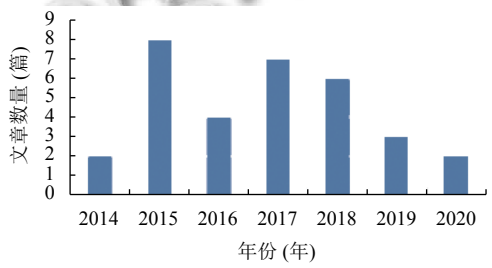

图 1 文章发表时间分布

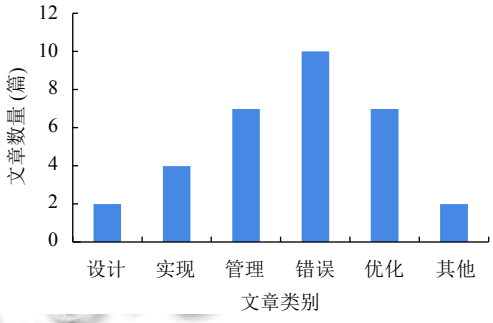

图 2 文章类别分布

3 软件配置研究概述

3.1 软件设计阶段的相关工作

软件配置项的选择和设计是软件设计阶段需要解决的问题. 需要把需求中的部分可变点抽象出来作为配置项, 以提高系统灵活性. 配置项的选定应该准确且适量, 这是因为: (1) 不准确的配置项难以表达系统的可变点, 增加用户的理解难度和系统的维护成本; (2) 过少的配置项不能满足系统灵活性的需求, 降低用户满意度并可能在特定场景下给开发人员带来困难; (3) 过多的配置项不利于维护, 也可能使用户很难确定需要改动的配置. 在完成基于可变点的配置项选定后, 需要对配置项进行设计, 完善其语法和语义信息, 使其易于

理解和使用. 配置项的设计包括配置项的名称、类型、存储形式、与变动点的映射关系等.

在可变点分析与配置项选择方面, Muñante 等人参考基于众包的需求获取方法, 提出基于众包的配置项选择方法[7]. 方法将众包分为领域专家和潜在用户两类, 其中领域专家负责确定系统的可变点, 并根据可变点初步选定配置项, 之后专家对潜在用户的用户画像进行分析, 并综合用户画像和系统配置问题设计问卷. 潜在用户负责填写问卷, 展现自己的用户画像并表达对系统的反馈. 最后, 专家和开发人员对问卷进行统计和分析, 确定配置项. 该方法的优势在于: (1) 使用户尽早接触到系统配置, 提高配置的可用性和易用性; (2) 不仅在配置项的选定过程中有所帮助, 在系统版本更迭、配置项变更时, 也可以采用类似的方法进行系统配置的更新. 其不足之处在于: (1) 难以对方法的代价和收益进行定量评估, 导致方法代价过大; (2) 潜在用户选择困难: 一方面, 在项目开发初期, 难以明确哪些群体是潜在用户; 另一方面, 调研的潜在用户数量不好确定, 太少可能导致调研结果出现偏差, 太多则会带来额外时间和金钱的开销.

在配置项设计原则方面, Xu 等人通过调研指出现有的系统配置项存在以下问题[2]: (1) 大部分用户只会修改小部分的配置项; (2) 相比数值型配置项, 枚举型配置项更容易被用户所使用. 同时, 配置项的值域越小, 越不容易出现配置错误. 由此, 作者给出了 2 个配置项设计原则: (1) 减少或者隐藏不必要和使用频率低的配置项, 对使用频率不高但确实需要的配置项, 可以通过“高级配置”的方式与普通配置项分隔开; (2) 缩小无意义的配置项值域, 多使用枚举型的配置项, 而枚举数量以 2~3 个为佳. 这些原则能够降低配置项设计的冗余、提高配置项的质量, 使用户更易于配置系统. 但是这些设计原则覆盖面有限, 不够全面和完备, 没有考虑到配置项的命名等问题. 例如, 有研究显示, 配置项语义含糊是导致配置错误的重要原因之一[8].

3.2 软件开发阶段的相关工作

开发阶段中的软件配置相关工作主要包括配置项实现和配置项管理.

3.2.1 配置项实现

配置项实现以设计为依据, 主要任务包括: 完善配置项的约束, 将配置项写入配置文件或配置库, 编码实现配置项访问接口, 以及形成配置说明文档. 当前已有工作主要关注于配置项约束的完善和配置文档的编写两方面.

在配置项的约束方面, Li 等人研究了 MySQL、Redis、ApacheHttpd 等常见大型开源系统的软件配置项, 对配置项按照类型进行了分类, 并指出了每类配置项的约束[9]. 作者首先将配置项划分为字符串型和数值型两大类, 再将每个大类划分若干小类, 继续细分直到语义明确为止. 对于每类配置项, 作者根据其类型和运行时需求定义了语法约束和语义约束. 如端口号应是 0–65535 中的一个整数 (语法约束) 且不能使用已经被占用的端口号 (语义约束). 该文章使用树状结构对配置项进行分类, 对研究对象系统配置项的覆盖率达到 95% 以上, 并且具有很好的可扩展性. 但基于类型的配置项约束完善方法主要考虑的是单个配置项的约束. 现实中的配置项约束不局限于单个配置项内, 还存在于多个配置项之间. 对于配置项之间的约束, 需要通过其他方法进行完善.

在配置项文档方面, Zhang 等人实现了配置文件自动注释工具 ConfigFile++[10]. 作者认为, 当前许多系统存配置文件注释不充分、不规范的问题, 导致了许多的用户配置错误. ConfigFile++从配置项的类型、约束、作用等方面对配置项进行概括, 生成统一格式的配置项注释, 插入配置文件中. ConfigFile++沿用 Xu 等人的工作[2] 和 Li 等人的工作 [9], 分析配置项的类型和约束; 对于配置项的作用, ConfigFile++对用户手册进行搜索, 找到与配置项最相关的段落, 使用 TextRank 模型[11] 提取关键词, 概括配置项作用. 最后, ConfigFile++生成统一格式的配置项注释, 插入配置文件中. 该方法相当于生成了一份精简版的用户手册, 用户在修改 ConfigFile++注释后的配置文件时, 能够对配置项有全面的了解, 减少配置错误的发生. 但是, ConfigFile++生成的配置项注释段落长、信息量大, 用户很难快速找到关键信息, 也会成倍地增加配置文件的大小. 同时, ConfigFile++生成优质配置注释依赖于成熟的用户手册. 极端情况下, 如果系统用户手册没有对配置进行说明, ConfigFile++只能对配置项本身进行分析, 能够提取的信息比较有限.

此外, Nadi 对配置项约束进行了分类[12]. 作者认为约束按照作用划分, 主要包括以下 4 类: (1) 底层配置项依赖关系, 配置项间的取值关联隐含在程序代码中, 配置项设置时必须满足此类依赖; (2) 根据运行环境配

置系统, 保证系统在不同环境下的正确运行; (3) 指导用户进行配置, 减少无效配置、提高用户配置体验; (4) 避免极端情况, 防止系统在配置项取某些值时不能正确运行. 王焘等人关注配置项间的关联, 定义了一致性关联和类型性关联[13]. 一致性关联指两个配置项具有相同的值, 或一个值是另一个值的子串; 类型性关联指两个配置项之间存在类型上的语义关联, 若一个配置项的值发生变化, 另一个配置项取值需相应进行改变. 这两类关联是配置项之间的约束, 要求修改存在关联的配置项时, 考虑其他配置项的变化.

文献 [10,12,13] 的工作都对配置项约束进行研究, 但分类方法和覆盖面有所不同, 总结如表 1 所示.

表 1 配置项约束研究工作对比总结

参考文献	约束提取方法	约束类型	约束信息
文献[10]	基于配置项类型提取	单个配置项约束	语法信息、语义信息
文献[12]	基于约束作用提取	单个配置项约束、配置项间约束	语义信息
文献[13]	基于配置项关联提取	配置项间约束	语义信息

3.2.2 配置项管理

软件开发过程中, 由于需求的变化或者新功能的增加, 配置项可能发生变化. 面对变化, 如果缺乏配置项的管理手段, 可能会出现配置项更新困难、配置不一致和配置项丢失等问题. 因此, 配置项的管理是软件开发过程中配置项有效性、一致性的重要保证.

Tang 等人介绍了 Facebook 配置管理套件[14]. 套件由 Configerator、Gatekeeper、PackageVessel、Sitevars 以及 MobileConfig 组成. Configerator 提供最底层功能, 提出配置即代码 (configuration as code) 的思想. 配置通过平台无关语言 Thrift 编写, 输出 JSON 格式的配置文件. Sitevars 提供编写和修改配置的图形化接口. PackageVessel 解决大型配置文件频繁更新的问题, 将配置文件和其元数据分开更新. 元数据上传时立即更新; 对于配置文件本身, PackageVessel 从元数据中找到文件位置, 使用混合订阅 P2P 模型传输. Gatekeeper 负责新特性发布, 控制特性的部分开放、对部分用户开放或禁用等, 控制产品逐步推出. MobileConfig 负责解决移动端 APP 的配置问题, 包括平台多样性、移动网络速度和新旧版本兼容性等问题. Facebook 的配置管理工具套件的优点在于: (1) 提出“配置即代码”的概念, 可以通过类似管理代码的方法管理配置项, 提高配置的可维护性; (2) 从多角度考虑配置管理过程中可能遇到的困难, 并设计了相应的解决方案. 该套件对于大型项目的配置管理具有很高的参考价值. 但另一方面, 该套件并不适用于中小型规模的软件系统, 此类软件系统的配置相对简单, 使用上述工具反而可能带来不必要的开销和效率的降低.

配置的管理可以依赖第三方工具完成, 许多研究人员着手于开发配置项收集分析工具, 辅助进行配置项管理. Tuncer 等人开发的云平台上的配置项自动发现和提取工具 ConfEx[15] 是该类工作的代表之一. 对待检测系统, ConfEx 首先根据词汇表和文件名划分出各个配置文件所属的软件; 其次 ConfEx 对配置文件进行文本分析, 以键值对的形式提取出文件中的配置项及其取值; 然后消除歧义, 保证每个键只对应唯一的值; 最后将解析得到的配置文件和配置项列表展现给用户. 对于 httpd、MySQL 和 Ngnix 等大型应用, 该工具可以找到 99.7% 的配置文件及包含的配置项, 而之前的方法 Augeas[16] 只能找到 35.8%. 但 ConfEx 的核心工作在于配置文件的发现上, 对于配置项的提取, ConfEx 仅是沿用了 Augeas 的方法. ConfEx 仅提供了配置项提取方法, 并不涉及配置项的语义和约束等信息.

此外, Bartusevics 等人设计了一套模型驱动的软件配置管理方式[17]. 该方式定义了配置管理过程中的 3 种模型: 环境模型、行为模型和分支模型, 以及 2 条规则: 环境转换规则和分支合并规则. 应用该方法, 在配置变化时, 首先明确变化在模型变更中的体现, 然后基于规则进行模型的转化, 完成配置项的管理. Perera 提出了一种自动化配置管理方法[18]. 该方法通过程序分析, 自动发现系统中的配置项及其约束, 存入数据库中进行统一管理. 相比参照程序规格说明书进行管理, 该方法更具有可靠性、且能够发现说明书与实际程序不符之处. Jin 等人开发了配置项查找工具 PreFinder[19]. PreFinder 工作流程分为 4 步: (1) 通过程序分析和 API 调用提取配置项; (2) 根据分隔符和贪心算法对配置项名称分词; (3) 对自然语言查询输入进行缩略语转化、停止词消除等预处理; (4) 使用 TF-IDF 算法计算查询与各个配置项的关联性, 推荐配置项. PreFinder 是最早将自然语言处理技术应用于配置领域的工作之一. 曾广福等人提出了枚举类型配置项配置空间的提取方法[20]. 该方法使用程序分析技术, 找到枚举类型配置项集中定义的代码段, 提取配置项的枚举值, 确定配置

空间. 作者使用 Redis、MySQL 等 6 款常用 C/C++开源软件进行测试, 结果显示该方法的提取准确率高于 80%. 复杂软件系统通常使用配置框架, 如 Java Properties、Spring 等, 统一管理配置. Sayagh 等人研究了配置框架对系统开发的影响[21]. 通过对 2000 余个 Java 项目和 11 种 Java 配置框架的分析, 作者指出基础、成熟、文档完备的框架经常在开发中被使用且稳定性较好.

本文认为已有工作主要从以下两方面提高软件开发过程中配置项管理的能力: (1) 配置演化变更管理, 建立和完善配置项变更应对机制, 以成熟的理论和工具变更配置项 (包括参考文献 [14,17]); (2) 配置理解发现, 发现和理解系统配置项, 对配置项进行集中管理 (包括参考文献 [15,18–20]). 两类工作相辅相成. 在实际配置管理过程中, 可以首先通过 (2) 中工作的方法整理配置项, 建立配置管理中心, 然后通过 (1) 中工作的方法变更配置.

3.3 软件后开发阶段的相关工作

软件后开发阶段主要包括测试、部署和生产. 在这 3 个阶段中, 软件配置所面临的主要问题既有相似性, 侧重点又有所不同. 3 个阶段都以配置错误处理和优化为主. 测试阶段侧重于配置错误的处理; 部署阶段侧重于配置项调优; 运维阶段则同时关注配置错误处理和性能优化两方面.

3.3.1 配置错误处理

软件配置错误常见且负面影响极大, 因此, 尽早地发现软件配置错误并进行修复极为重要. 软件配置错误处理主要分为检测、诊断和修复 3 个阶段[1]. 错误检测主要判断系统是否存在配置错误, 或者当前系统错误是否是软件配置导致的; 错误诊断主要分析配置错误具体是由哪一个或者哪一组配置项导致的, 以及为什么会导致配置错误; 错误修复为错误的配置项重新设置正确的值, 使系统可以正确运行. 此外, 还有一些工作研究配置项在程序中的表现形式 (如程序中的配置项读入点、配置项在程序中的类型等), 辅助进行配置错误的处理.

Xu 等为尽早发现配置错误, 将软件测试的思想引入软件配置中, 提出了软件配置测试方法[22]. 配置测试分为 3 步: (1) 将硬编码在源代码中的配置项参数化; (2) 给配置项赋予实际运行环境中使用的值; (3) 运行测试用例. 作者还提出了以下几点看法: (1) 配置测试应该保证测试的充分性和路径覆盖率; (2) 可以沿用软件测试中的测试用例, 因为这些测试用例已经包含了大多数配置项的执行路径; (3) 当软件测试的测试用例不足以覆盖执行路径时, 需要配置测试专用的测试用例; (4) 当配置出现变更时, 需要对变更的配置项本身和依赖该配置项的所有配置项进行配置测试. 该方法指出可以像测试代码一样充分测试配置, 避免配置错误的发生. 但是, 大型软件系统配置项数量多、结构复杂, 如何设计测试用例, 通过少量用例覆盖大部分配置, 作者在文中并没有进一步讨论.

Wang 等人对比异常日志与正常日志的差异, 开发了基于日志的配置错误诊断工具 MisconfigDoctor[23]. MisconfigDoctor 分为学习和诊断两个阶段. 在学习阶段, MisconfigDoctor 首先违背配置项约束, 产生若干错误的配置, 再分别在正确和错误的配置上运行测试用例, 得到日志输出, 并对日志文件进行移除时间戳、统一单词形式等预处理. MisconfigDoctor 比较正常日志和异常日志, 记录每个配置错误对应的“+语句”(正常日志不出现而异常日志出现的句子)、“–语句”(正常日志出现而异常日志不出现的句子), 以此作为配置错误的特征存入库. 在诊断阶段, 对于新的异常日志, MisconfigDoctor 对异常日志进行预处理, 与特征库中的各个配置错误进行比对, 得到最有可能的数个配置错误. 实验显示, 该方法检测出了 31 个真实存在的配置错误. 该方法充分利用了程序日志, 以白盒的方法进行配置错误的诊断. 但是, 该方法也存在一些不足: (1) 依赖于软件系统的日志信息. 如果系统日志区分度不足, MisconfigDoctor 很难进行正确的诊断; (2) 该方法只能诊断出学习阶段中预定义的配置错误, 难以发现新类型的配置错误.

Zhang 等人研究软件更新过程中产生的配置错误, 开发了工具 ConfSuggester, 推断导致配置错误的配置项[24]. ConfSuggester 运行过程分为 3 个阶段: 运行程序、比较结果和推断配置项. 运行程序阶段, ConfSuggester 使用相同配置文件和输入运行更新前后两个版本的程序, 检测每个程序断言执行的次数和判断为真的频率. 比较结果阶段, ConfSuggester 首先将新旧两个版本对应的程序断言进行匹配, 然后计算每个断言的执行次数和判断为真频率的调和平均数, 当平均数偏差大于阈值时, 该断言存在问题. 推断配置项阶段, ConfSuggester 进行程序切片, 找到引起问题断言表现不同的配置项并计算权重, 输出结果. ConfSuggester

可以在 1.8 的平均推荐顺位上找到配置错误, 而之前的工具[25] 需要在 15.3 的平均推荐顺位才能找到. ConfSuggester 在发现更新引起的配置错误上表现较好, 但扩展性有限. 对于长期存在的配置错误和无法提供历史版本的系统, ConfSuggester 作用有限. 另一方面, 当程序规模较大时, 简单的输入很难覆盖到所有程序断言, 需要高质量或者多数量的输入为 ConfSuggester 提供支持.

配置项取值不同会影响程序的控制流. 发现异常控制流是解决配置错误的重要方法之一. Nguyen 等人开发了程序交互发现工具 iGen[26]. 程序交互, 指配置项值与程序代码覆盖间的关联, 即配置项值对程序控制流的影响. iGen 生成多份被测程序的配置文件并运行程序, 划分出每份配置文件覆盖和未覆盖的代码.当覆盖条件由多个配置项取值的合取组成时, 若多份配置文件覆盖到同一段代码, iGen 对这些配置文件的配置项进行交运算并继续生成多组配置文件, 重新运行和计算直到结果不再变化. 结果集合表示对应代码被覆盖到时各个配置项的取值范围. 类似地, 当覆盖条件由多个配置项取值的析取组成, 或由多个配置项的析取和合取共同构成时, iGen 也设计了相应检测算法. 在 29 个程序片段上的实验显示, iGen 计算出各个程序交互的平均 $F1$-score 达到了 99.7%.该方法的局限性在于: (1) 检测的配置项类型以离散的数值型为主, 当配置项是字符串型时, 该方法不适用. 当配置项是连续的数值型时, 该方法很难得到准确的结果; (2) 当配置项数量多、值域大时, 检测的准确性会有所降低.

除了上述工作之外, Sayagh 等人将研究重点放在跨软件栈的配置错误上, 指出跨栈配置错误由于配置项数量多、层间配置项关系复杂等因素, 处理较为困难. 在此之上, 作者提出了基于程序分析的模块化方法, 诊断出可能出错的配置项[27]. 该方法已经应用到实践中, 发现并修复了 1082 个配置错误. Santolucito 等人使用机器学习的关联规则挖掘算法研究配置项是否有误[28]. 作者从配置文件中提取配置项并进行训练, 使用基于关联规则的学习算法得到一系列配置项的规则, 以此判断一个配置项是否存在问题. 经过实验, 该方法可以检测出多种类型的配置项错误, 并保持着 11%~18% 的低误报率. Xu 等人开发了工具 PCHECK 用于检测延迟性配置错误[29]. 延迟性配置错误是指在系统启动时不会触发, 但随着系统运行、执行到特定语句时触发的配置错误. 这类配置错误通常难以发现. PCHECK 通过程序分析的手段, 生成能够检测配置项是否存在错误的代码并插入程序中. 实验表明该方法可以有效检测出延迟性配置错误, 并对其他类型的配置错误也具有一定的检测能力. Potharaju 等人开发了工具 ConfSeer, 用于检测配置错误, 并给出修复方案[30]. ConfSeer 结合了自然语言处理、信息检索和交互式学习等多种技术, 建立解决配置错误的知识库, 检测和修复配置错误. 该工具的准确率达到了 80%~97.5%, 并且有运行开销低的优点. Xu 等人开发了自动推测配置项类型的工具 ConfTypeInfer[31]. ConfTypeInfer 使用基于配置项名称的方法和抽象语法树分析方法推测配置项的类型. 基于配置项名称的方法通过配置项词典分析配置项名称的语义信息, 确定其类型; 抽象语法树分析方法通过分析程序的抽象语法树, 确定配置项是否为枚举类型, 是基于配置项名称方法的补足. 两种方法综合使用, ConfTypeInfer 的准确率达到了 90% 以上. Dong 等人开发了识别程序中配置项读入点的工具 ORPLocator[32]. ORPLocator 工作流程如下: (1) 以程序代码和配置父类作为输入; (2) 根据配置父类识别出配置子类; (3) 提取子类中从配置文件中读取配置项值的方法; (4) 找到调用这些方法的代码, 得到配置项对应变量. 实验表明, 该方法可以在大型程序中找到 95% 的配置项读入点.

本文对上述工作关注点和配置分析采用的技术手段的总结如表 2 所示.

表 2 配置错误处理研究工作总结和对比

研究问题	参考文献	技术手段	适用场景
错误发现	文献[22]	白盒	软件测试阶段
	文献[28]	黑盒	可以获得同一系统的大量配置文件
	文献[29]	白盒	配置约束已在程序中定义
错误诊断	文献[23]	黑盒	可以获取软件日志
	文献[24]	白盒	诊断软件更新引起的配置错误
	文献[26]	黑盒	大部分场景都适用; 同时可以进行配置错误修复
	文献[27]	白盒	诊断跨软件栈配置错误
	文献[30]	黑盒、白盒	存在已知配置错误解决方案; 同时可以进行配置错误修复
辅助配置错误处理	文献[31]	白盒	检测配置项类型
	文献[32]	白盒	检测配置项读入点

由表 2 可看出:

(1) 配置错误处理方面的工作采用的技术手段以白盒分析方法为主 (7 项工作使用了白盒分析方法,

4 项工作使用了黑盒分析方法). 本文认为, 其原因在于: 1) 配置错误的处理通常由软件开发方进行, 其有能力获取软件代码, 可以实行白盒分析; 2) 白盒分析方法可以覆盖到更多的配置错误, 相比黑盒方法更容易找到和处理不常出现的错误, 且具有更高的准确率.

(2) 配置错误修复方面的研究工作较少 (仅文献 [26,30] 的工作可以辅助进行配置错误修复). 本文认为, 其原因在于: 许多配置错误的修复需要运用领域知识进行人工修复, 自动化程度较低, 难以形成系统化的工作.

3.3.2 配置参数调优

软件配置与系统性能密切相关. 通过调整配置以改善系统性能, 首先要建立模型, 表示配置项与系统性能的关系; 然后根据实际需求和约束, 为配置项设定恰当的值, 通过配置参数调优达到优化系统性能的最终目标.

Sarka 等人研究了建立配置项与系统性能关系模型时对配置项进行抽样的方法[33]. 方法在模型准确率与样本采样代价之间进行权衡, 提出了基于频率的投影抽样来实现模型准确率和获取样本代价间的平衡. 投影抽样是在选定初始样本后, 每次迭代中新增一定数量的样本产生 (样本容量, 准确率) 点对, 并使用常见的曲线进行拟合, 找到一条能够较好表达当前模型样本容量和准确率曲线的方法. 方法基于频率选取初始样本, 使初始样本中每个配置项都至少需要被启用和禁用若干次, 这样生成的样本才具有代表性, 能够快速找到拟合度高的曲线. 实验显示, 投影抽样方法优于渐进抽样, 而基于频率的初始样本选取方法也在大多数情况下比传统的随机选取和多路选取方法好.

Wang 等人开发了用于系统性能调优的配置框架 SmartConf[34]. 该框架根据用户期望的系统性能, 由系统自动调整系统配置以满足需求. 使用 SmartConf 框架时, 开发者说明哪些配置项与什么性能相关, 并设定配置项的初始值; 用户说明期望的系统性能, 系统通过 SmartConf 控制器, 自动调整配置项的值. SmartConf 控制器通过控制理论实现, 系统下一时刻的性能由当前时刻配置项的值所决定, 而当前时刻配置项的值又由上一时刻配置项的值、以及当前时刻系统性能与用户期望的误差所共同决定. 实验结果显示, 相比静态取值, SmartConf 动态修改配置项的值可以提高系统 1.2~1.5 倍的性能. SmartConf 的不足在于需要开发人员指出配置项和性能指标之间的关联, 需要丰富的领域知识和前提条件. 如果能够实现配置项-性能指标关联自动发现功能, SmartConf 的动态调整效果可以进一步提升. 另一方面, 由于 SmartConf 在系统运行时动态修改配置项, 因此不适用于运行时可以动态更新配置并使之生效的系统和场景.

除了上述工作之外, Zhang 等人提出基于傅里叶变换的系统性能配置模型以及模型学习方法[35]. 将该方法在多个系统上的准确率达到 92.5% 以上, 而模型的构建仅使用 2.2% 的样本. 系统性能不佳经常是由于系统资源不足、程序竞争资源产生的, 为此 Feng 等人开发了检测系统资源竞争并通过调整配置项解决竞争的工具 Relax[36]. Relax 通过静态程序分析的方法找到和资源相关的软件配置项, 调整竞争软件对资源的占有率以解决冲突. 引入 Relax 后, 程序的运行时间能够缩短 15% 以上. 关于选取模型的学习样本, Nair 等人认为建立精确的配置性能模型是代价巨大且不必要的, 无需精确的性能值即可对配置方案进行排名[37]. 作者提出了基于排名的方法, 显著地降低了建立模型的成本, 并在多个系统中取得了明显的成果. Kaltenecker 等人提出了基于距离的采样方法[38]. 该采样策略基于距离度量和概率分布, 以给定的概率分布在配置项空间中采集配置项样本. 经过实验, 该方法在配置项空间较大时仍然可以建立更准确的性能模型. Han 等人使用神经网络估计系统性能, 分析输入程序、服务器配置配置、云服务器间干扰程度与程序执行时间之间的关系[39]. 在用户选择购买时, 使用该模型评估当前服务器配置的性能, 购买到恰当配置的服务器资源.

本文对上述工作关注点和配置分析采用的技术手段的总结如表 3 所示.

表 3 配置参数调优工作总结和对比

研究问题	参考文献	技术手段	适用场景
配置性能模型建立	文献[33]	—	数据集建立阶段
	文献[37]	—	数据集建立阶段
	文献[38]	—	数据集建立阶段
	文献[35]	—	模型构建阶段
配置参数选定	文献[34]	黑盒	配置项与性能关联已知; 系统配置可以在运行时调整
	文献[36]	白盒	系统配置可以在运行时调整
	文献[39]	黑盒	云服务器配置

由表 3 可看出: 在配置参数调优领域, 许多工作的焦点集中在快速建立高质量的配置数据集上. 本文认

为, 这是因为: 1) 软件系统配置项多、配置空间巨大, 采集典型的配置样本集合比较困难. 2) 对每一个配置样本, 系统都需要以该配置运行系统才能采集评估指标, 获取评估指标代价较大. 3) 数据集建立后, 构建配置性能模型的方法有一定的数学理论作为支撑, 相对容易.

3.4 其他工作

除了上述工作之外, 还有一些工作的重点在于对已有系统配置的分析和实证研究方面.

Jha 等人以 908 个 Android 应用程序的配置文件 (即 manifest.xml 文件) 为研究对象, 对这些文件的版本变更过程进行分析和总结, 找到了一系列 Android 应用程序配置变更的规律[40]. 实证研究发现: (1) 配置变更的主要原因是新增组件 (占全部变更的 44.0%) 和申请新的权限 (占全部变更的 14.6%). 进一步地, 作者总结认为 Android 应用程序发生配置变更的首要原因是应用程序新增功能. (2) Android 应用程序配置变更的频繁程度和其功能点数量并没有明显的关系. (3) 除了新增功能之外, 很大一部分配置变更是由于改进用户界面. 改进用户界面的手段包括更新主题、图表、导航栏等 (占全部变更的 21.6%). (4) Android 应用程序常在新增功能点方面做无用功, 即新增了一个功能点、但过一段时间后又由于某些原因删除了这个功能点. 这些发现能够对 Android 应用程序的更新提供指导, 并对软件配置变更的研究有一定的意义.

Sayagh 等人以 WordProcess (WP) 为例, 研究了跨软件栈系统中低层软件配置对高层应用的影响[41]. WP 按调用关系从上至下可以分为 3 层: WP 插件、WP 本身和 PHP. 研究分析发现: (1) WP 插件为提高可用性, 通常将配置项存储在数据库中; 而 WP 本身也是如此. 因此, 数据库中的配置项可能属于软件中的任意一层. (2) WP 插件跨层使用配置项的情况较为常见, 平均每个插件使用 1.4 个 PHP 层配置项、并修改 0.4 个 PHP 层配置项的值. (3) 单个配置项被多个插件同时使用的情况十分常见, 78.88% 的可配置常量和 85.16% 的数据库配置项会同时被至少 2 个插件使用, 而单个配置项同时被最多 458 个插件使用. (4) 配置项间接使用的情况比直接使用的情况更多. 这些发现说明了跨软件栈系统中可能潜藏着许多配置问题. 进一步地, 这些发现能够为跨软件栈配置错误处理提供理论依据.

4 分析和展望

通过对综述范围内 (除参考文献 [1,3–6,8,11,16,25,42–44] 外，均为综述范围内的文献) 的研究成果分类和分析, 本文对当前软件配置领域的相关研究成果所具有的特点总结如下:

(1) 在软件生命周期中, 软件配置的研究成果主要集中于开发阶段和后开发阶段, 统计结果如图 3 所示. 其中, 配置错误处理、配置项管理和配置参数调优是研究的热点. 本文认为, 这是因为: 1) 配置项数量和复杂度的增加, 简单维护配置文件或数据库已经很难满足需求、难以应对复杂的情况, 因此需要新的技术和工具辅助进行配置项管理, 提高效率; 2) 配置错误和系统性能不佳是软件配置缺陷的直接表现, 配置错误处理和配置参数调优是解决问题的直接手段. 由于软件系统的配置项的增加, 配置缺陷逐渐暴露, 首先要解决的就是直接体现的配置错误和系统性能问题.

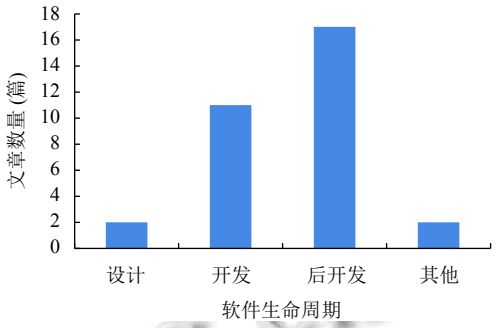

图 3 不同阶段软件配置领域文章数量统计

(2) 黑盒方法和白盒方法都广泛应用于软件配置相关的研究中, 统计结果如图 4 所示. 总体而言, 两种方法的使用频率相近, 但在具体领域, 黑盒方法和白盒方法各有优劣, 故使用率存在较大差异.

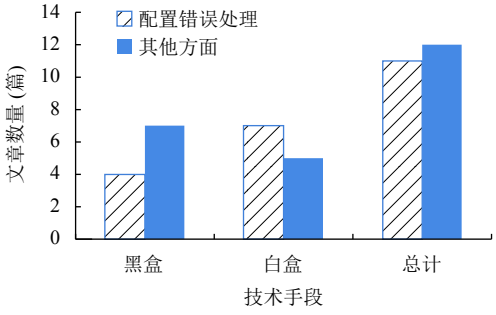

图 4 各领域工作采用的技术手段分布

(3) 对于配置错误处理领域的工作, 本文已在 3.3.1 节末尾作出分析. 在其他方面的工作, 采用的技术手段

以黑盒分析方法为主. 本文认为, 其原因在于: 1) 其他方面的工作通常有软件的使用者参与, 不宜将代码实现暴露给用户; 2) 配置参数调优关注软件配置与系统性能间的关系, 其中涉及到语言优化问题和操作系统层面问题等, 基于白盒方法进行分析并不适用.

(4) 使用白盒方法对软件系统配置进行分析的一般流程如下: 1) 获取系统配置项; 2) 找到配置项在程序中的读入点; 3) 切片分析, 找到配置项影响的控制流和数据流; 4) 定位所分析问题对应的程序片段; 5) 分析配置项控制流、数据流和对应程序片段的关系; 6) 评价各个配置项与所分析问题的关联度, 排序并给出结论.

(5) 使用黑盒方法对软件系统配置进行分析, 一般从以下几方面考虑: 1) 分析配置项名称和值中的语义信息; 2) 收集多份配置文件, 通过数据挖掘和机器学习等方法发现配置项取不同值时和所分析问题的关联; 3) 收集系统用户手册和论坛相关问答, 通过自然语言处理等方法提取有用知识; 4) 收集日志信息, 挖掘日志输出和配置的关联.

综合上述几点, 本文认为, 在选择使用黑盒方法还是白盒方法进行软件系统配置分析时, 可以从以下几方面进行考虑:

(1) 问题领域. 当进行配置错误处理时, 白盒方法更为常用; 当进行配置性能调优, 尤其涉及到多软件系统、操作系统层面的问题时, 通常考虑黑盒方法.

(2) 程序源码是否可获得. 如果可获得, 可以考虑使用白盒方法进行分析, 否则只能使用黑盒方法.

(3) 是否存在额外资源 (如用户手册、日志输出等). 如果存在, 有助于提升黑盒方法分析的准确度; 如果只有单一配置文件, 使用白盒方法分析准确度更高.

通过对当前软件配置错误检测、诊断与修复相关的研究热点和主要工作进行分析, 本文认为该领域的未来研究趋势包括以下方面:

(1) 跨软件栈配置错误的处理. 对于单个软件堆栈上的配置错误处理的研究, 目前发展较为充分, 但对于跨软件栈的配置错误处理, 研究成果仍然较少. 而在实际应用中, 跨软件栈的大型复杂分布式系统已成为常态. 调查显示, 相比单软件配置错误, 跨软件栈的配置错误更容易导致系统崩溃, 且修复难度不低于单软件配置错误[27]. 因此, 对于跨软件栈配置错误问题的研究将是未来的研究发展趋势之一.

(2) 基于深度学习的配置优化. 随着配置项数量的增长和软件性能指标的增加, 传统的机器学习模型面临着建模难度大和需求训练集样本容量大的问题. 在其他领域 (如声纹识别领域[42,43]) 的成果表明, 面对复杂的建模问题, 深度学习方法和传统机器学习方法相比模型表示更简单、准确率更高. 因此, 本文认为, 在未来几年的研究中, 深度学习方法会逐步应用于以提高系统性能为目标的配置优化中.

(3) 软件配置演化维护. 持续演化已成为软件系统发展演进的重要特征, 其中的软件配置也随之一同演化和发展, 并贯穿于软件生命周期的各个阶段. 因此, 对于软件配置问题的理解和认识更加需要站在全生命周期的视角去理解和看待, 将各个阶段融会贯通, 通过持续软件工程的方法提高软件配置在系统持续演进过程中的维护和管理能力. 例如, 通过不断的迭代演进将后开发阶段发现的配置问题反馈到新一轮迭代中的配置设计和实现中, 不断提升软件配置的质量、合理性和可理解性等.

(4) 软件配置领域知识的汇聚与运用. 开源软件和技术社区的发展积累了大量的软件数据, 其中也包括与软件配置相关的数据与知识, 散落和隐藏在多源异构的软件仓库与社区中. 例如, 从 Docker Hub、GitHub 和 StackOverflow 中都能够获取与软件系统配置相关的数据和知识[44]. 因此, 从上述数据来源中分析抽取软件配置知识, 并进行有效的组织和管理 (如知识图谱), 也是后续研究中提高配置领域知识应用的一个有效方法和途径.

(5) 新软件形态和技术中的配置问题. 随着云计算、人工智能和大数据等技术的成熟运用, 基于这些技术的软件系统和平台不断涌现. 新场景下是否衍生出新的软件配置需求和问题, 已有软件配置方法和成果是否能够应对, 还有待进一步研究.

5 结束语

随着软件系统规模的不断扩大, 以及用户对系统灵活可定制性要求的逐渐提高, 软件配置已经成为软件工程领域的一个重要话题. 国内外众多学者和研究人员从不同视角出发, 对软件配置的诸多问题展开研究, 取得了众多成果. 本文对软件配置领域的研究现状和主要成果进行分析综述, 首先分析了软件配置面临的问题, 然后提出了基于软件生命周期和技术手段的软件配置相关研究工作的分析框架, 然后基于该框架

对当前主流的研究成果和研究现状进行分类总结和分析评价, 最后总结当前软件配置领域研究工作的特点, 并对未来研究方向提出了展望和建议, 对于今后该领域的继续和深入研究具有一定的借鉴意义.

参考文献

1 陈伟, 黄翔, 乔晓强, 等. 软件配置错误诊断与修复技术研究. 软件学报, 2015, 26(6): 1285–1305. [doi: 10.13328/j.cnki.jos.004823]

2 Xu TY, Jin L, Fan XP, *et al*. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo, Italy. 2015. 307–319. [doi: 10.1145/2786805.2786852]

3 Yin ZN, Ma X, Zheng J, *et al*. An empirical study on configuration errors in commercial and open source systems. Proceedings of the 23rd ACM Symposium on Operating Systems Principles. Cascais, Portugal. 2011. 159–172. [doi: 10.1145/2043556.2043572]

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

5 Barroso LA, Clidaras J, Hoelzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Morgan & Claypool, 2013. [doi: 10.2200/S00516ED2V01Y201306CAC024]

6 Sayagh M, Kerzazi N, Adams B, *et al*. Software configuration engineering in practice interviews, survey, and systematic literature review. IEEE Transactions on Software Engineering, 2020, 46(6): 646–673. [doi: 10.1109/tse.2018.2867847]

7 Muñante D, Siena A, Kifetew FM, *et al*. Gathering requirements for software configuration from the crowd. Proceedings of the IEEE 25th International Requirements Engineering Conference Workshops (REW). Lisbon, Portugal. 2017. 176–181.

8 Gunawi HS, Hao MZ, Leesatapornwongsa T, *et al*. What bugs live in the cloud? A study of 3000+ issues in cloud systems. Proceedings of the ACM Symposium on Cloud Computing. Seattle, WA, USA. 2014. 1–14. [doi: 10.1145/2670979.2670986]

9 Li SS, Li W, Liao XK, *et al*. ConfVD: System reactions analysis and evaluation through misconfiguration injection. IEEE Transactions on Reliability, 2018, 67(4): 1393–1405. [doi: 10.1109/TR.2018.2865962]

10 Zhang YL, Li SS, Xu XY, *et al*. ConfigFile++: Automatic comment enhancement for misconfiguration prevention. Proceedings of 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE). Campobasso, Italy. 2018. 37–42.

11 Mihalcea R, Tarau P. Textrank: Bringing order into text. Proceedings of 2004 Conference on Empirical Methods in Natural Language Processing, EMNLP 2004, A Meeting of SIGDAT, a Special Interest Group of the ACL, Held in Conjunction with ACL 2004. Barcelona, Spain. 2004. 404–411.

12 Nadi S, Berger T, Kästner C, *et al*. Where do configuration constraints stem from? An extraction approach and an empirical study. IEEE Transactions on Software Engineering, 2015, 41(8): 820–841. [doi: 10.1109/tse.2015.2415793]

13 王焘, 陈伟, 李娟, 等. 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]

14 Tang CQ, Kooburat T, Venkatachalam P, *et al*. Holistic configuration management at Facebook. Proceedings of the 25th Symposium on Operating Systems Principles. Monterey, CA, USA. 2015. 328–343.

15 Tuncer O, Bila N, Duri S, *et al*. ConfEx: Towards automating software configuration analytics in the cloud. Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). Luxembourg. 2018. 30–33.

16 Lutterkort D. Augeas —A configuration API. Linux Symposium, Ottawa: Red Hat, Inc., 2008. 47–56.

17 Bartusevics A, Novickis L. Models for implementation of software configuration management. Procedia Computer Science, 2015, 43: 3–10. [doi: 10.1016/j.procs.2014.12.002]

18 Perera N. Automatic configuration management: Autodiscovery of configuration items and automatic configuration verification. SpaceOps 2016 Conference. Daejeon, Republic of Korea. 2016. 1–13.

19 Jin DP, Cohen MB, Qu X, *et al*. PrefFinder: Getting the right preference in configurable software systems. Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. Vasteras, Sweden. 2014. 151–162. [doi: 10.1145/2642937.2643009]

20 曾广福, 何浩辰, 周书林. 一种面向枚举类型的配置约束提取方法. 计算机工程与科学, 2020, 42(4): 634–640. [doi: 10.3969/j.issn.1007-130X.2020.04.009]

21 Sayagh M, Dong Z, Andrzejak A, *et al*. Does the choice of configuration framework matter for developers? Empirical study on 11 Java configuration frameworks. Proceedings of the IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM). Shanghai, China. 2017. 41–50.

22 Xu TY, Legunsen O. Configuration testing: Testing configuration values as code and with code. arXiv: 1905.12195, 2019.

23 Wang T, Liu XD, Li SS, *et al*. MisconfDoctor: Diagnosing misconfiguration via log-based configuration testing. Proceedings of 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). Lisbon, Portugal. 2018. 1–12.

24 Zhang S, Ernst MD. Which configuration option should I change? Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India. 2014. 152–163. [doi: 10.1145/2568225.2568251]

25 Zhang S, Ernst MD. Automated diagnosis of software configuration errors. Proceedings of the 35th International Conference on Software Engineering (ICSE). San Francisco, CA, USA. 2013. 312–321.

26 Nguyen T, Koc U, Cheng J, *et al*. iGen: Dynamic interaction inference for configurable software. Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seattle, WA, USA. 2016. 655–665.

27 Sayagh M, Kerzazi N, Adams B. On cross-stack configuration errors. Proceedings of the IEEE/ACM 39th International Conference on Software Engineering. Buenos Aires, Argentina. 2017. 255–265.

28 Santolucito M, Zhai EN, Dhodapkar R, *et al*. Synthesizing configuration file specifications with association rule learning. Proceedings of the ACM on Programming Language, 2017, 1: 64. [doi: 10.1145/3133888]

29 Xu TY, Jin XX, Huang P, *et al*. Early detection of configuration errors to reduce failure damage. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation. Savannah, GA, USA. 2016. 619–634.

30 Potharaju R, Chan J, Hu LH, *et al*. ConfSeer: Leveraging customer support knowledge bases for automated misconfiguration detection. Proceedings of the VLDB Endowment, 2015, 8(12): 1828–1839. [doi: 10.14778/2824032.2824079]

31 Xu XY, Li SS, Guo Y, *et al*. Automatic type inference for proactive misconfiguration prevention. Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering. Pittsburgh, PA, USA. 2017. 295–300.

32 Dong Z, Andrzejak A, Lo D, *et al*. ORPLocator: Identifying read points of configuration options via static analysis. Proceedings of the IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). Ottawa, ON, Canada. 2016. 185–195.

33 Sarkar A, Guo JM, Siegmund N, *et al*. Cost-efficient sampling for performance prediction of configurable systems (T). Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). Lincoln, NE, USA. 2015. 342–352.

34 Wang S, Li C, Hoffmann H, *et al*. Understanding and auto-adjusting performance-sensitive configurations. Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems. Williamsburg, VA, USA. 2018. 154–168.

35 Zhang Y, Guo JM, Blais E, *et al*. Performance prediction of configurable software systems by Fourier learning (T). Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). Lincoln, NE, USA. 2015. 365–373.

36 Feng ZM, Li SS, Liao XK, *et al*. Relax: Automatic contention detection and resolution for configuration related performance tuning. Proceedings of the 25th Asia-Pacific Software Engineering Conference (APSEC). Nara, Japan. 2018. 239–248.

37 Nair V, Menzies T, Siegmund N, *et al*. Using bad learners to find good configurations. Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany. 2017. 257–267. [doi: 10.1145/3106237.3106238]

38 Kaltenecker C, Grebhahn A, Siegmund N, *et al*. Distance-based sampling of software configuration spaces. Proceedings of the 41st International Conference on Software Engineering. Montreal, QC, Canada. 2019. 1084–1094. [doi: 10.1109/ICSE.2019.00112]

39 Han J, Kim C, Huh J, *et al*. Configuration guidance framework for molecular dynamics simulations in virtualized clusters. IEEE Transactions on Services Computing, 2017, 10(3): 366–380. [doi: 10.1109/TSC.2015.2477835]

40 Jha AK, Lee S, Lee WJ. An empirical study of configuration changes and adoption in Android apps. Journal of Systems and Software, 2019, 156: 164–180. [doi: 10.1016/j.jss.2019.06.095]

41 Sayagh M, Adams B. Multi-layer software configuration: Empirical study on wordpress. Proceedings of the IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM). Bremen, Germany. 2015. 31–40.

42 Dehak N, Kenny PJ, Dehak R, *et al*. Front-end factor analysis for speaker verification. IEEE Transactions on Audio, Speech, and Language Processing, 2011, 19(4): 788–798. [doi: 10.1109/TASL.2010.2064307]

43 Snyder D, Garcia-Romero D, Povey D, *et al*. Deep neural network embeddings for text-independent speaker verification. Proceedings of the 18th Annual Conference of the International Speech Communication Association. Stockholm, Sweden. 2017. 999–1003.

44 Xu TY, Marinov D. Mining container image repositories for software configuration and beyond. Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results. Gothenburg, Sweden. 2018. 49–52.