DOI:10.7544/issn1000-1239.2020.20190079
57(1):188-201,2020

一种基于关联挖掘的服务一致化配置方法

王 焘[1,2] 陈 伟[2] 李 娟[3] 刘绍华[4] 苏林刚[2] 张文博[1,2]
[1](计算机科学国家重点实验室(中国科学院软件研究所) 北京 100190)
[2](中国科学院软件研究所 北京 100190)
[3](北京工业大学 北京 100124)
[4](北京邮电大学 北京 100876)
(wangtao@otcaix.iscas.ac.cn)

Association Mining Based Consistent Service Configuration

Wang Tao[1,2], Chen Wei[2], Li Juan[3], Liu Shaohua[4], Su Lingang[2], and Zhang Wenbo[1,2]
[1](*State Key Laboratory of Computer Science* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)
[3](*Beijing University of Technology*, *Beijing* 100124)
[4](*Beijing University of Posts and Telecommunications*, *Beijing* 100876)

Abstract Componentized service-oriented software systems always consist of loosely coupled heterogeneous service components, each of which contains a large number of configuration items configured with high flexibility. Complex dependencies exist between service components, resulting in their interrelated configuration items, so the operations of deploying, updating and migrating components are prone to errors. For configuration items related with each other, changing one configuration item requires to modify other related configuration items. Otherwise, violating constraints perhaps happens, which should cause system failure. Therefore, analyzing associations between configuration items is one key to ensure the reliability of a system. This paper proposes a service configuration approach based on association mining. Our approach crawls configuration files from Internet, narrows the analysis scope to frequently changed configuration items, generates association coefficients for item pairs according to the similarity of items' name, value and type, determines the set of candidate item pairs with rules, and then outputs a list of ordered configuration item pairs for query. We have deployed two typical open-source software systems to validate our approach for mining configuration associations between configuration items with case studies. Experimental results show that our approach can accurately detect most associations between configuration items.

Key words association mining; service configuration; configuration association; error detection; service reliability

摘 要 组件化服务化软件系统由松耦合的异构服务组件构成,每个服务组件都包含着大量可高度灵活配置的配置项.服务组件之间存在着复杂的依赖关系,导致其配置项相互关联,使得系统部署、更新或迁移易于出错.对于相互关联的配置项,更改一个配置项就需要修改与之关联的其他配置项,否则将违反

收稿日期:2019-02-13;**修回日期**:2019-11-08
基金项目:国家重点研发计划项目(2017YFB1400804);国家自然科学基金项目(61872344);北京市自然科学基金项目(4182070);中国科学院青年促进会人才专项项目(2018144)
This work was supported by the National Research and Development Program of China (2017YFB1400804), the National Natural Science Foundation of China (61872344), the Beijing Natural Science Foundation (4182070), and the Fund of the Youth Innovation Promotion Association of Chinese Academy of Sciences (2018144).

约束条件,导致系统出现故障.因而,分析配置项关联性对于保障系统可靠性至关重要,但需要跨产品的领域知识.提出了一种基于关联挖掘的服务一致化配置方法.该方法爬取配置文件样本数据以将搜索范围缩小到频繁改变的配置项,根据配置项的名称、取值和类型的相似性计算,为配置项对生成关联系数,使用定义的过滤规则确定候选关联配置项对集合,输出排序的配置项关联性列表以供查询.基于该方法部署了典型应用系统进行实验和评估,实验结果表明:该方法能够准确检测配置项的关联性.

关键词　关联挖掘;服务配置;配置关联;错误检测;服务可靠性

中图法分类号　TP311

服务化软件系统通常由许多异构服务组件构成,每个服务组件都有许多配置项.例如,MySQL 5.6 数据库服务器有 461 个配置参数,Apache 2.4 的所有模块中有超过 550 多个配置参数[1].服务组件规模巨大以及多层软件栈结构导致实际系统中通常包含成千上万的配置项,使得系统正确配置困难且易于出错.配置错误已经成为当今系统故障的主要原因之一[2].微软、亚马逊和 Facebook 等主要 IT 公司都经历过配置错误所导致的宕机事件[3-5].在配置错误中,配置项关联性(简称关联性)引起的错误占很大比例.研究表明,12.2%～29.7%的错误与配置关联性有关[6].

部分关联性由服务组件之间的依赖关系引入.例如服务组件需要进行数据库访问,那么,服务的数据库连接配置项需要与数据库信息关联,即服务与数据库的数据库名称、用户名和密码等参数值必须保持一致.研究报告表明,开源软件项目中有 27%～51%的配置项和另一个项目存在关联性[7].然而,分析配置项关联性,特别是跨服务组件关联性,非常困难.首先,关联配置可能会跨多个服务组件,每个服务组件存在大量配置项,分析配置信息的工作量巨大;其次,众多服务组件,尤其是开源软件仓库中的软件,文档可能与代码不一致,甚至没有文档[8];最后,服务组件使用多种编程语言,因而难以使用程序分析方法[9].即便是领域专家也很难拥有跨多种服务组件和软件的知识[10],而一旦忽略了一些配置项的关联性,就可能会违反配置约束,从而导致系统错误.

本文提出了一种基于关联挖掘的服务一致化配置方法.首先,从开源项目的代码库中爬取配置文件的样本数据,将搜索范围缩小到更改频繁的配置项;然后,根据配置项名称、取值比较和类型推断计算每个配置项对的关联系数,并且提供滤波器以确定可能关联的配置项候选集合;最后,输出配置项关联性的排序列表,以便系统管理员重点关注一些配置项,并可以通过查询操作检查系统配置,从而减少配置错误所导致的系统错误.进而,挖掘配置项的关联性,并且在召回率、准确率等方面对方法的有效性进行了实验评估.实验结果表明,所提出方法可以准确分析配置项的关联性,讨论了过滤器对最终结果的影响、关联性配置的分布、产生错误的原因等问题.

本文的主要贡献为:通过代码仓库挖掘与配置文件比较,评估配置项的关联性,从而为实现大规模分布式软件的自动化、智能化配置部署及错误诊断建立基础.与文献[7]相比,所提出方法无需掌握目标软件的系统架构、软件组件、交互行为、部署项含义等面向系统运维的用于部署配置的特定领域知识.可自动检测配置项的关联性,以有效减少系统配置并诊断配置错误的工作量.并且,搭建了典型的开源软件系统,基于准确性与召回率对方法的有效性进行了实验评价,分析讨论了过滤器对最终结果的影响及相关性配置的分布,比较了现有工作,并结合实验结果分析导致错误的问题原因.

1　研究动机

配置项关联是指在跨服务组件的软件系统中,某个服务组件的一个配置项依赖于其他配置项或环境对象[11].当一个配置项改变时,与之关联的配置项都需要作出相应修改.例 1 中的 MySQL 和 Tomcat 的配置项具有关联性,关联语义约束了 Tomcat 可以使用的持久连接数量 $mysql.max\_persistent$ 不能大于 MySQL 提供的总量 $max\_connections$,违反约束就会发生过多连接错误;例 2 中的 Web 服务组件 LogineService 与 EJB 服务组件 LoginEJB 的配置项"jndi-name"关联,关联语义约束这 2 个配置项具有相同的值,否则应用程序将发生登录失败.

例 1. MySQL 和 PHP 的配置关联.

MySQL 配置文件:

$max\_connections=300$.

PHP 的配置文件：

mysql.max_persistent＝400.

约束：在使用持久化连接的时候，PHP 中 *mysql.max_persistent* 值应该不超过 MySQL 中 *max_connections* 的值.

影响：引发“too many connections”错误.

例 2. 应用组件间的配置关联.

Web 服务 LoginService 的配置文件：

〈parameter *name*＝“jndi”〉Login〈/parameter〉.

EJB 组件 LoginEJB 配置文件：

〈jndi-name〉 LoginService 〈/jndi-name〉.

约束：LoginService 中的 jndi 必须和 LoginEJB 中的 jndi-name 保持一致.

影响：无法登录应用，抛出异常.

检测配置项关联性以及约束条件，对于保障系统配置的正确性至关重要.在部署、迁移和更新系统时，违反约束就会出现配置项错误，从而导致系统故障.如果事先获知配置项间的关联性，当某个服务组件更新造成配置信息改变时，管理员就可以对其他服务组件的配置信息做相应修改，从而减少错误的发生.同时，当系统出现故障时，管理员可以重点关注关联配置项，缩小配置错误检查的范围，从而降低系统故障风险并且减少人力投入.

为了确定跨服务组件配置项的关联性，研究了有代表性的开源软件，包括关系型数据库 MySQL①、应用服务器 Tomcat②、内存数据库 Redis③等.通过对这些软件特征的分析，发现了 3 种现象：

1) 如果 2 个服务组件相互依赖，可能存在跨服务组件的配置项关联性.服务组件依赖通常以资源供给、函数调用、数据共享和数据传输等方式实现.例 1 是资源持久连接所产生的关联性，例 2 是函数调用所产生的关联性.

2) 配置项根据其键值对的语法可分为不同的类型.常见的 3 种配置项类型是数字、布尔值和字符串[8]，可以根据配置项中键值对的语法模式推断其语义.例如，“*max_connections*＝300”是数字类型的配置项，可以根据推断出其表示的是资源(即最大连接数)数量.最典型的是字符串类型的配置项，例如在 MySQL 中，“*datadir*＝/var/lib/”可以推断为指定的文件路径.

3) 尽管服务组件具有大量配置项，但只有小部分经常使用.文献[1]研究表明，大多数用户只设置了一小部分配置项(6.1%～16.7%)，而高达 54.1% 的配置项使用默认值.还对开源软件 Redis 进行具体研究，例如从 Github④中多个项目中抓取 60 个 Redis 的配置文件并分析其配置项值，发现在 38 个配置项中只有 5 个配置项(即 13.2%)的值经常变化，而其他的配置项(即 86.8%)只有 5 个以下不同的值.表 1 给出了 Redis 经常变化的配置项.

Table 1 Configuration Change List

表 1 配置项数量列表

Configuration	Appearance No.	Configuration No.
port	56	21
pidfile	42	27
logfile	29	19
dbfilename	48	26
dir	40	20

基于观察发现：1)通过分析配置项的键和值信息，可以推断语义信息；2)通过分析配置项类型，可以推断其表示的对象与特征.因此，根据对配置项键、值和类型的分析，提出基于关联挖掘的服务一致化配置方法.

2 服务一致化配置方法

2.1 方法概述

方法技术路线如图 1 所示，主要包括配置项过滤、配置项类型推断、关联系数计算、过滤配置关联性、配置项关联性排序等 5 个步骤.

1) 配置项过滤.通过互联网使用爬虫技术从代码仓库(如 Github)中抓取开源软件(如 Redis)的配置文件作为样本数据，过滤掉几乎没有变化的配置项以缩小搜索范围，并关注那些频繁修改的配置项.在相同配置项的多个例集合里，通过配置值变化数量的绝对值和比例值来判断，比如少于 5 个不同值、少于 5%.

2) 配置项类型推断.将配置项的键值对与定义的正则表达式及关键字相匹配，可以推断其类型.

3) 关联系数计算.根据配置项的关键字、值和

① https://www.mysql.com

② http://tomcat.apache.org

③ http://redis.io

④ https://github.com

类型等特征,计算不同服务组件的每对配置项的关联系数.

4) 过滤配置关联性.基于关联系数过滤非关联配置对,将过滤结果作为确定关联性的依据.

5) 配置项关联性排序.将根据关联系数排序的关联配置项对的列表提供给用户,以供参考检查或修改系统配置.

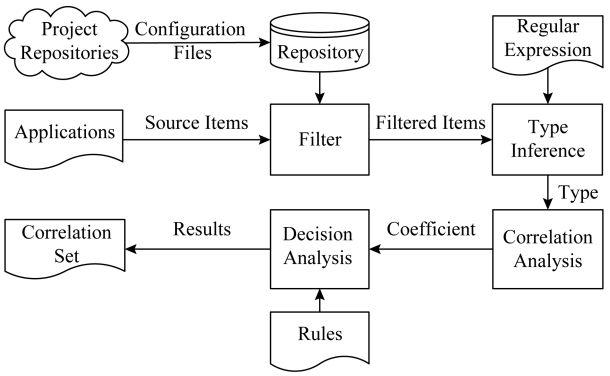


Fig. 1　Approach overview

图 1　技术路线

2.2　配置库构建

从在线技术论坛和代码托管网站,包括 ServerFault①, StackOverflow②, Database Administrators③, Github 中,抓取流行开源项目(例如 Web 服务器、数据库、消息中间件)的配置文件例.

方法针对具有大量配置项的系统软件作为目标软件,找到经常会发生变化的常用配置项.以 Github 为代表的开源软件仓库积累了大量软件项目,其中很多软件需要使用诸如数据库、消息队列等类型的系统软件,存在众多配置文件,从而能够支持方法对常用、常变配置项的识别.方法能够适用于以配置文件进行设置的系统软件.当然,方法的适用性受能够收集到的配置使用信息影响,因此对于广泛使用的常用软件具有更好的适用性.

建立配置库以检测配置项关联性包括 3 个步骤:1)确定目标系统及相关软件以限定分析对象(如数据库、消息中间件等).2)从 Github 库的项目中搜索相应的软件配置文件.3)检测相关组件配置文件中配置项的关联关系.以典型的 3 层架构企业应用为例:1)目标系统为企业应用,包括表现层、业务逻辑层、数据访问层.2)从 Github 库的项目中搜索表现层和业务逻辑层、业务逻辑层和数据访问层的相关软件.表现层典型软件为 Apache, Lighttpd, Nginx;业务逻辑层典型软件为 Tomcat, Jetty, JBOSS;数据访问层典型软件为 MySQL, PostgreSQL, InterBase.3)基于同类型软件集合建立配置仓库,挖掘不同类型软件组件间的关联关系.

基于同类型软件集合重点分析以扩充配置仓库,重点分析开源软件仓库中相关软件的配置文件,可以提升方法的针对性与应用效果.同时,构建配置项库过程中,每个软件和工具的配置项例数量是一个逐渐累积的过程,且数量越多,对于发现经常修改的配置项集合以及常用取值,尤其是系统软件的数值型配置项,起到促进作用.例如,从 Github 上找到了超过 100 个 Redis 的配置文件来构建实验所用的 Redis 相关配置项库.

对样本数据进行统计分析,获取每个配置项的例值,提出 2 个过滤规则以获取配置项的频繁项集.

1) 多值过滤.如果配置项例在样本数据集中的值有较大差异,则该配置项为频繁项.例如表 1 中的端口为 Redis 的监控端口,不同服务组件的端口值通常不同,收集了 56 个端口配置项例,其中的 21 个具有不同的值.

2) 异值过滤.如果样本数据中没有出现目标系统中的配置项例值,则该配置项为频繁项.这是由于在特定服务组件中,某些配置项可能配置为样本数据中未出现的特定值,例如配置项中设置文件路径、用户名和密码.

2.3　配置项类型推断

配置项类型通常包括数值型、布尔型和字符串型,每种类型可能具有多个子类型.例如 Redis 的"*pidfile*"表示文件路径,"*bind*" 表示 IP 地址,二者都是字符串类型配置项.推断配置项类型有助于获取其语义.配置项值与每个正则表达式匹配,遵循 3 个规则:

1) 如果配置项类型是数字或布尔类型,则配置项的值几乎没有语义信息.例如 Redis 的"*port*"表示监听端口号,"*timeout*"表示超时的时间,二者都是数字类型配置项.如果仅仅根据配置项的值,则没有属性及特征信息,配置项名称需要用其他正则表达式和关键字表示.

2) 如果配置项类型和子类型都被推断出来,则

① http://serverfault.com

② http://stackoverflow.com

③ http://dba.stackexchange.com

使用更具体的类型来描述配置项.例如“IP Address”(服务器的IP地址)既是定义的IP类型,又是字符串类型,那么将该配置项设置为IP类型.

3) 使用关键词可以推断出多个子类型.例如“*jdbc.pool.maxIdle*”表示数据库连接资源上限,是数字类型配置项.配置项名称中“*jdbc.pool*”可以推断为Resource类型,而“*maxIdle*”可以推断为Size类型.

每个配置项的推断类型以类型向量表示:$\boldsymbol{T}_{\text{entry}}=(t_1,t_2,\cdots,t_m)$,其中,$t_i(1\leqslant i\leqslant m)$表示一种配置项类型,当配置项属于此类型时,$t_i=1$,否则$t_i=0$;$m$表示配置项类型数量,向量长度固定.由于一个配置项可以同时具有多种推断类型,向量中可能有多个元素的值为1.表2给出了典型配置项类型描述,同时配置项类型是可扩展的以适应新的类型、正则表达式和关键字.

Table 2 Configuration Type Description

表2 配置类型定义

Type	Subtype	Regular Expressions	Semanteme
String	URL	[[a-zA-Z]+\\:]+//[^\\s]*	Internet URL
	IP	(([0-9]|[1-9][0-9]|[0-9]{2}|2[0-4][0-9]|25[0-5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])+(:(0|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-4][0-9]{3}|65[0-4][0-9]{2}|655[0-2][0-9]|6553[0-5]))?	IP
	File Path	/?([^/]+/)+[^/]*	Path
	Date Time	[0-9]{4}-(((0[13578]|(10|12))-(0[1-9]|[1-2][0-9]|3[0-1]))|(02-(0[1-9]|[1-2][0-9]))|((0[469]|11)-(0[1-9]|[1-2][0-9]|30)))	Date
	E-mail	^\\w+([-+.]\\w+)*@\\w+([-.]\\w+)*\\.\\w+([-.]\\w+)* $	E-mail
	Resource	buffer|cache|pool|thread|disk|cpu|memory|queue|message|table|connection|block|packet|file|socket|session|port|host|jdbc|dir|directory	Resources
	Host	server|host	Server
	Database	db|jdbc|database	Database
	ID	identy|identifier|id|name|uri|jndi	ID
	User	user|usr	User
	Group	Group	Group
	Password	password|pwd|pass	Password
Boolean	Common	[tT][rR][uU][eE]|[fF][aA][lL][sS][eE]|[oO][nN]|[oO][fF]{2}|[yYnN]|[yY][eE][sS]|[nN][oO]	Feature
	Mode	mode|enable|disable	Mode
Number	Common	[+-]?\\d+[.]?\\d+	Common Number
	Port	port|listen	Port
	Time	time|interval|day|month|year|hour|minute|second|millisecond	Time
	Size	size|number|length|max|min|threshold	Size
	Count	Count	Counter

2.4 关联系数计算

将配置项关联性分为一致关联性和类型关联性.

1) 一致关联性

一对配置项具有相同值,或者一个值是另一个值的子串,在配置项关联性中最为常见,可以用一致关联性系数衡量.2个配置项之间的一致关联性系数基于关键字、值和类型计算.这是由于如果2个配置项关联,其值相同或者相似.同时,由于配置项具有相似语义,关键字和类型也相似.

一致性关联是通过取值来推测2个配置项可能描述的是同一个对象,由于关注的配置项都是采用〈*key*,*value*〉形式存储,很难主动识别参数值的数据类型.例如,密码可能是“123456”,也可能是“qwe123”等,因此,统一作为字符串类型处理具有更好的通用性.所提出方法根据表2将值抽象化为正则表达式,正则表达式能够表达值的数据结构和类型,因此计算最长公共子串用来衡量值的数据类型和类型的相似度.

给定配置项$e_i=\langle k_i,v_i,\boldsymbol{T}_i\rangle$,其中,$k_i$为配置项$e_i$的关键字,$v_i$为$e_i$的取值,$\boldsymbol{T}_i$为类型向量.计算关键字、取值和类型之间的相似性,然后将这些相似性的平均值作为一致关联性系数.

基于“最长公共子串”方法计算配置项 e_i 与 e_j 的键和值的相似度为

$$sim(str1,str2)=\frac{mostCommonSubStr(str1,str2)}{maxlong(str1,str2)},\quad(1)$$

其中,函数 $mostCommonSubStr(str1,str2)$ 表示字符串 $str1$ 和 $str2$ 的公共子串,$maxlong(str1,str2)$ 表示字符串 $str1$ 和 $str2$ 的较长字符串长度值.

基于余弦计算类型向量 $\boldsymbol{T}_i$ 和 $\boldsymbol{T}_j$ 相似度为

$$sim(\boldsymbol{T}_i,\boldsymbol{T}_j)=\frac{\boldsymbol{T}_i\times\boldsymbol{T}_j}{\|\boldsymbol{T}_i\|\cdot\|\boldsymbol{T}_j\|}.\quad(2)$$

将配置项对 e_i 与 e_j 的键、值、类型相似度平均值作为一致关联性系数:

$$consis(e_i,e_j)=\alpha\times sim(k_i,k_j)+\beta\times sim(v_i,v_j)+\gamma\times sim(\boldsymbol{T}_i,\boldsymbol{T}_j).\quad(3)$$

相似度取值范围为[0,1],分数越高,配置项对存在一致性的可能性越高.对开源软件的例分析,发现以上 3 个相似度对最终结果的影响差别不大,因此采用均值计算总的关联系数.在未来的工作中,将进一步研究是否采用加权均值方式可以改进方法的效果.

2) 类型关联性

如果一个配置项的值改变了,另一个配置项应该变为相应的值,而不一定是相同的值,在大多情况下,可以从配置项类型中推断出来.例如 Resource 类型配置项与 Size 类型的配置项关联,即后者设置了前者所表示资源的数量.再如 URL 和 IP 这 2 种类型通常相互关联.

“一致关联性”是不同配置项在表示同一个对象时,取值要保持相同或部分相同;而“类型关联性”是不同配置项存在语义关联,当一个发生变化,另一个也需要随之改变.例如用户名和密码就是关联类型.“一致关联性”中的“类型”是字段的数据类型,如数字型、布尔型、字符串等;“类型关联性”是某配置项的值随其他配置项做相应变化.

定义了配置项类型之间的共性关系,其中每种关系都隐含着 2 种实体之间的语义.通用类型关联以系统部署和运维管理的领域知识为基础,描述配置项之间的语义关联,类型关联包括:1)用户信息,包括用户名/密码/邮件地址;2)主机信息,包括 IP 地址/端口/URL/主机名;3)文件信息,包括文件名称/用户组别/访问权限.例如,对于数据库系统,数据库名称和数据库 IP 地址以及用户名和密码是类型相关的,当数据库变化,对应的 IP 地址和用户名/密码也可能发生变化.再如,$\langle FilePath,User\rangle$ 表示这 2 种类型的配置项因权限而关联,而 $\langle Host,IP\rangle$ 表示主机配置项具有此 IP 地址.另外,用户也可根据领域知识自行设置类型关联规则.

类型关联分数 $correl(e_i,e_j)$ 用于评估配置项类型关联性,首先计算 2 个配置项 (e_i,e_j) 的类型向量 (t_i,t_j) 之间的各类型关联的数量,而后将值归一化为范围为 0 到 1 之间:

$$correl(e_i,e_j)=\frac{2}{\pi}\times\arctan\sum_{i=1}^{n}\sum_{j=1}^{n}corrVal(t_i,t_j).\quad(4)$$

当 t_i 与 t_j 关联时,$corrVal(t_i,t_j)=1$,否则为 0.配置项对 (e_i,e_j) 的一致关联性和类型关联性是配置项对的 2 种不同的相似性 e_i 和 e_j,当 $consis(e_i,e_j)$ 增加,那么 $correl(e_i,e_j)$ 则随之减少,反之亦然.

2.5　配置项关联性确定

根据 2.4 节的方法,可以检测到众多配置项对之间存在着关联性,为了保证结果的正确性,本节提出多个过滤规则以去除错误的关联性结果.

1) 阈值过滤.为配置项一致关联性设定阈值 H_c,为配置项类型关联设定阈值 H_t,当关联性系数小于阈值,则 2 个配置项之间的关联关系较弱,过滤掉该配置项对.

2) 冗余过滤.观察发现,一个服务组件的配置项 e_i 很少会与另一个服务组件的多个配置项关联.因此,如果 1 个配置项在由 2 个服务组件组成的配置项对中出现了 3 次以上,仅将关联系数最高的 3 个配置项对作为关联配置项,过滤掉其他配置项对.

3) Top-K 过滤.根据配置项对的关联性系数按降序排序,得到一致关联性和类型关联性 2 个关联性排序列表,将 2 个列表中的前 K 个配置项对作为关联配置项对.

使用以上 3 个过滤规则,将关联性较低的配置项对过滤掉后,可以得到配置项一致关联性列表和类型关联性列表的并集作为最终候选列表.

3　实验评价

3.1　实验环境

使用服务化 Java 应用系统 Adventure 和基于云的存储服务 CloudShare① 等 2 个典型的开源软件系统以评估所提出方法,表 3 给出了 2 个实验系统的服务组件.

① https://www.cloudshare.com

Table 3 Experimental Service Components

表 3 实验系统服务组件

System	Component	Version	#Components	Function
CloudShare	Csenant.war		1	Portal
	Cloudshare.war		2	Frontend
	Service.war		2	Backend
	Index.war		1	Index
	Messageing.war		1	Messageing
	Apache Tomcat	7.0	6	Runtime
	Nginx	1.6.2	1	Request Distribution
	Redis	2.8.9	1	Cache
	ActiveMQ	5.8	1	Message Queue
	JDK	1.7	4	JVM
	Cs_global		2	Global Database
	Cs_tenant_default		2	Tenant Database
	Cs_msg_tenant Default		2	Message Database
	MySQL	5.6	2	MySQL Database
Adventure	Loging EJB		1	EJB Component
	Mountain EJB		1	
	Island EJB		1	
	Transport EJB		1	
	Hotel EJB		1	
	Bank EJB		1	
	Login Service		1	Web Service
	Mountain Service		1	
	Island Service		1	
	Transport Service		1	
	Hotel Service		1	
	Bank Service		1	
	Travel Plan Proces		1	WS-BPEL
	Adventure.war		1	Portal
	JDK	1.7	2	JVM
	Tomcat	7.0	1	Web Runtime
	Web Service Container		1	Webs Service Runtime
	WS-BPEL Container		1	WS-BPEL Runtime
	EJB Container		1	EJB Runtime
	Travelplan		1	Travel Database
	MySQL	5.6	1	MySQL Database

Adventure 是提供旅游安排服务的应用,采用 SOA(service oriented architecture)框架,具有 Web services, WS-BPEL(Web services-business process execution language),EJB(enterprise Java beans)和其他服务组件.在 3 个服务器上总共部署 22 个服务组件,包括应用的服务组件和系统软件(如 Tomcat, MySQL).

CloudShare 提供文件存储与共享、工作协同和即时消息等众多服务.将该系统部署在阿里云[①]环境中,其中的 28 个服务组件分布在 5 台云主机上,配置为 Intel® Core™ i7, 3.4 GHz CPU, 4 GB RAM, CentOS 6.5 操作系统.

① http://www.aliyun.com

服务一致化配置方法检测关联配置项对列表,当管理员进行系统部署、升级或迁移时,以该列表作为参考以辅助检查系统配置正确性,避免违反关联性约束条件.

3.2　实验步骤及结果

服务一致化配置方法的具体实现步骤包括:1)使用 Scrapy 爬取 Github 上目标系统的配置文件,解析〈*Key*,*Value*〉为类型配置项保存在 Redis 数据库;2)定义正则表达式用以判定配置项的语义类型;3)依据规则计算配置项之间的一致性和类型关联性;4)使用过滤器算法把得到的备选集合进一步过滤.

实验分为配置项过滤、配置项类型推断、关联系数计算及结果过滤 4 个步骤.方法涉及的参数包括:一致性关联的阈值(H_c)与类型关联的阈值(H_t),Top-K 排序过滤的阈值(K).根据实践经验,实验前两者设置为 0.6,K 则设置为 5.这 3 个参数都是阈值型参数,用以确定是否将备选的配置关联作为最终结果返回,H_c 主要用于一致性关联,H_t 用于类型关联,K 用于确定选取过滤的对象数量.

1) 配置项过滤

由于互联网上具有大量开源软件的配置文件样本数据,配置项过滤可以很大程度上减少需要分析的配置项数量.如图 2 所示,CloudShare 比 Adventure 的配置项过滤效果要好,这是由于前者使用众多的开源服务组件来构件系统,大部分配置项都被过滤掉了,例如 CloudShare 过滤掉了 50%以上 Nginx 的配置项和 80%以上 Redis 的配置项.

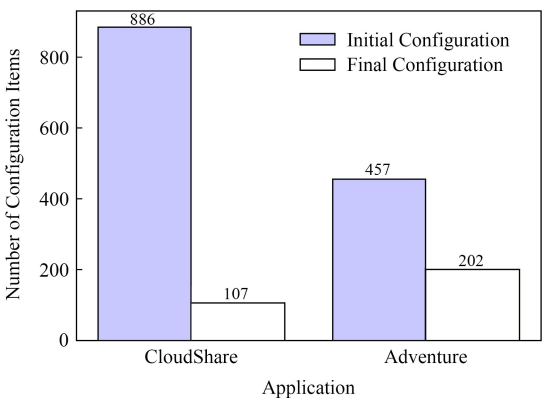


Fig. 2　Filtering Results

图 2　配置项过滤结果

2) 配置项类型推断

对于系统中的每个服务组件,建立频繁配置项集,并推断配置项类型.表 4 展示了配置项类型推断的结果,通过人工比对,大多数配置项的类型推断都是正确的.表 4 中的配置项总数为 202 项,比图 2 中的配置项总数要多,这是由于很多配置项有多种类型.例如"*db.default.USER*=app"用于设置数据库用户名,可以根据键中的关键字来推断数据库和用户类型.配置推断错误与错误率如表 5 所示,在 CloudShare 的 202 个类型推断中有 11 个错误,错误率为 5.45%.例如"*mail.username* = noreply@cloudshare.im"根据正则表达式推断为电子邮件类型,但是这个配置项实际上是一个用电子邮件设置的用户名.再如"*server_id* =1"为数字类型的配置项,实际上用来作为服务器 ID.在 Adventure 的 212 个类型推断中有 17 个错误,错误率为 8.02%.

Table 4　Configuration Type Distribution of CloudShare

表 4　CloudShare 配置类型分布

Common Type	Subtype	Appearance Number
String	Password	18
	URL	23
	IP	22
	Flie Path	31
	Date Time	0
	E-mail	3
	Resource	5
	Host	1
	Database	21
	ID	0
	User	12
	Group	0
	Else Type	11
Boolean	Mode	1
	Else Type	5
Numeric	Count	6
	Port	28
	Time	4
	Size	17
	Else Type	0

Table 5　Fault Rate of Configuration Inference

表 5　配置推断错误率

Application	Configuration Type	Fault	Fault Rate/%
CloudShare	202	11	5.45
Adventure	212	17	8.02

3) 关联系数计算及结果过滤

基于第 2 节所提出的配置项关联性检测方法,

为配置项对生成一致关联系数和类型关联系数.通过人工手动判断找到的关联性是否正确,使用准确率(precision, P)与召回率(recall, R)评价所提出方法的效果:

$$P=\frac{TP}{TP+FP},\quad R=\frac{TP}{TP+FN},\tag{5}$$

其中,TP(true positive)表示正确发现的关联数量,FP(false positive)表示错误判断的关联数量,FN(false negative)表示存在关联但被判断为无关联的数量.

通过对 CloudShare 和 Adventure 的配置项做逐条深入分析,人工在 CloudShare 中发现 91 个配置项关联关系,在 Adventure 中发现 84 个配置项关联关系,以之作为基准进行评价.根据所提出的方法,在 CloudShare 中发现了 65 个正确关联关系,在 Adventure 中发现了 69 个正确关联关系.因此,CloudShare 的召回率为 65/91=71.43%,Adventure 的召回率为 69/84=82.14%.

使用阈值过滤器、冗余过滤器和 Top-K 过滤器对 1)2)步骤检测的关联配置项对进行过滤操作,以输出关联配置项对的最终候选集合.根据经验,设置阈值的默认值为 0.6,k 的默认值为 5.对于准确率,不同过滤规则的组合会对最终结果有着不同的影响,分别进行评价.图 3 和图 4 中 T 表示实验过程中使用阈值过滤器,R 表示实验过程中使用冗余过滤器,Top-K 表示实验过程中使用 Top-K 过滤器,T+R表示同时使用阈值过滤器和冗余过滤器,T+Top-K 表示同时使用阈值过滤器和 Top-K 过滤器,T+R+Top-K 表示同时使用所有过滤器.实验结果如图 5 所示,对于 CloudShare,只使用阈值过滤器(T)的精度是最低的,约为 65/140=46.43%,这是由于存在很多假阳性结果.进而,通过与其他不同的过滤器组合来减少假阳性结果以提高精度,最高能够达到约 65/97=67.01%.Adventure 与 CloudShare 的结果类似,当只使用阈值过滤器(T)时,精度最低,约为 69/132=52.27%.然后,精度增加到 53.91%,这是由于冗余滤波器去除了一些假阳性结果.Adventure 中 T+Top-K 和 T+R+Top-K 的 2 个实验的最终准确度相同,约为 78.41%,这是由于大多数假阳性结果都被 Top-K 过滤器过滤掉了,不存在多余的候选配置项关联,因此,当进一步使用冗余滤波器时,冗余滤波器对最终结果没有影响.

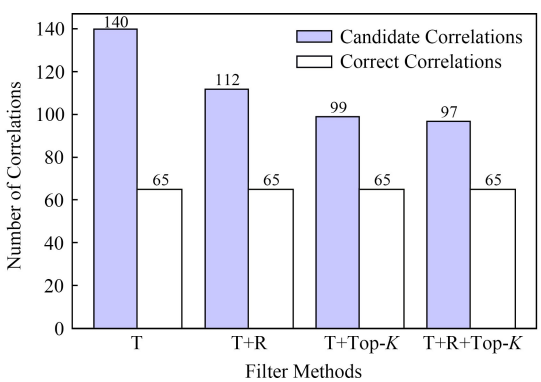


Fig. 3 Experimental results of filters in CloudShare

图 3 CloudShare 过滤结果

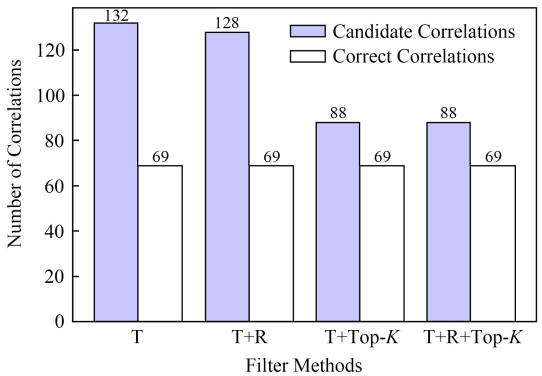


Fig. 4 Experimental results of filters in Adventure

图 4 Adventure 过滤结果

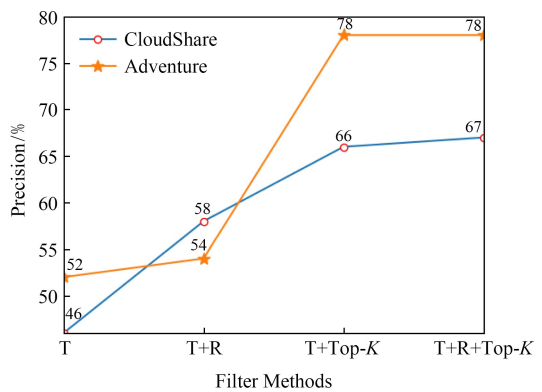


Fig. 5 Precision comparison of filters

图 5 过滤准确率比较

进一步分析发现关联性排序前 5 名(即 $K=5$)的正确配置项对的数量.如图 6 所示,排名第 1 的数量分别为 45 和 54,分别占 69.23%和 78.26%,实验结果表明关联性排序可以准确表现配置项的关联程度.

3.3 方法比较

文献[11]提出一种配置参数关联分析方法,当配置文件中参数值是相同字符串或者一个值是另一个值的子串,则检测为配置关联.在实验中,将所提

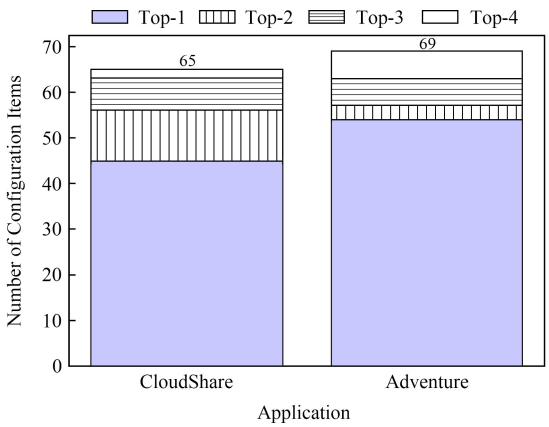


Fig. 6　Rank of configuration association

图 6　配置项关联排序

出方法与该方法进行比较.如图 7 和图 8 所示,CloudShare 和 Adventure 表示所提出方法的效果,而 CloudShare_N 和 Adventure_N 表示文献[11]所提出方法的效果.实验结果表明这 2 种方法的召回率相近,但是所提出方法的准确率却远高于已有工作.例 3 和例 4 描述了错误检测的配置关联性,如例 3,

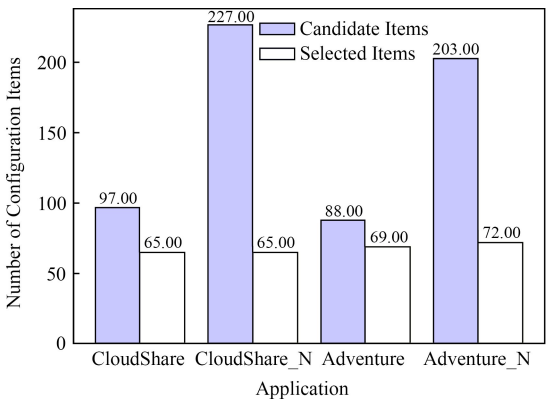


Fig. 7　Comparison of experimental results

图 7　实验结果比较

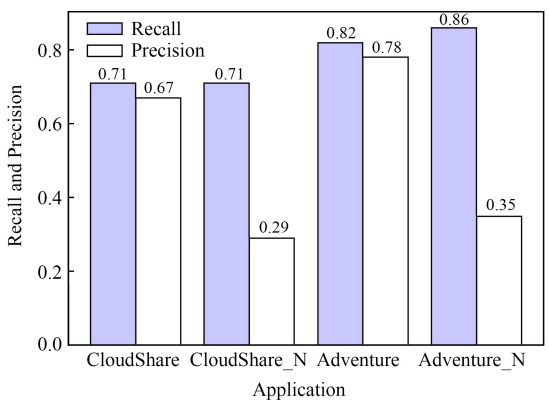


Fig. 8　Comparison of precision and recall

图 8　准确率和召回率比较

已有方法仅比较配置项的取值,所以存在许多假阳性结果.另一方面,如例 4,2 个类型关联的配置项会由于取值不同而被忽略,从而造成假阴性结果.

例 3. Nginx 和服务组件的错误关联.

Nginx 配置文件:

upstream.msg.server=133.133.134.174:8082.

服务配置文件:

redis.host=133.133.134.174.

约束:前者设置消息服务的负载均衡器,后者设置 Redis 的 IP 地址,二者值相似但意义不同.

例 4. 索引服务和数据库的遗漏关联.

Index 的配置:

jdbc.username=index-app.

数据库的配置:

password=pwd.app.

约束:2 个配置相关联,如果前者的值改变,后者的值相应改成该用户在数据库中对应的密码.

3.4　实验结果讨论

1) 假阴性错误

通过分析实验结果,发现大多数遗漏的关联关系涉及 2 个以上配置项.

例 5. 配置项 1 对多关联关系.

Nginx 配置项:

upstream.msg.server=133.133.134.174:8082.

Node2 配置项:

redis.host=133.133.134.174.

Tomcat 配置项:

Connector.port=8082.

例 6. 配置项间关联关系.

MySQL 配置项:

database.name=*cs_global.cs_tenant_default.port*=3306.

service.war 配置项:

jdbc.url=jdbc:mysql://133.133.134.175:3306/cs_tenant_default.

Node5 配置项:

node.host=133.133.134.175.

例 5 显示了 1 对多的关联关系,其中 Nginx 的配置项与服务器节点 2 的 IP 地址和 Tomcat 的端口关联.例 6 涉及到 4 个配置项,其中,*jdbc.url* 与 MySQL、服务器的其他 3 个配置项关联.所提出的方法只关注 1 对 1 的关联关系,仅发现了 *upstream.msg.server* 和 IP 的关系,这是由于差异较大的字符串导致最终的关联性系数很低.

2) 关联性分布

将系统的服务组件具体分为 2 类.

① 应用服务组件(图 9 中表示为 App).提供业务相关的功能和服务,例如 CloudShare 中的 Web 模块(即 WAR 包)和 Adventure 系统中的 Web 服务、EJB 和 WS-BPEL 流程等;

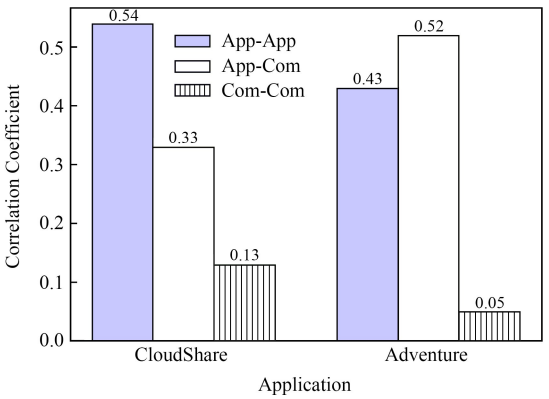


Fig. 9 Configuration association

图 9 配置项关联

② 实例通用服务组件(图 9 中表示为 Com).提供公共服务的服务组件以支持多种业务应用,如 Nginx,Redis,Tomcat,ActiveMQ,MySQL.

服务组件之间的依赖关系分为 3 类: App 与 App,App 与 Com,Com 与 Com.根据 3 种类型的服务组件依赖关系对关联关系进行分组,分布情况如图 9 所示,发现大多数关联存在于 App 与 App 之间以及 App 与 Com 之间.这是由于应用的服务组件依赖于系统软件所提供的服务,造成许多配置项互相关联.例如,service.war 依赖于 Redis 的缓存服务,因此它们之间有 3 个配置项关联,即端口号、IP 地址和密码.另外,应用的服务组件之间的数据通信和功能依赖也产生了许多 App 与 App 的配置项关联.例如,Web 服务 HotelService 和 HotelEJB 之间有 4 个关联,即 jndi-name,jndi-provider,URL,以及其他一些参数.因此,大多数配置项关联都是应用在程序与其他软件之间.

4 相关工作

在配置错误检测方面,通常采用程序分析方法,主要包括静态分析与动态分析.基于静态数据流的方法剖析软件源代码并分析数据执行流,预先计算可能出现的配置错误[12].ConfAid 动态注入程序执行的源码以跟踪程序执行流程,检测错误的根本原因[13].ConfDiagnoser 将静态分析与动态分析相结合,基于统计分析技术将不希望的行为与特定的配置项联系起来[14].CODE 基于统计分析技术设定在特定背景下访问配置项的规则,通过检测访问配置的行为自动发现软件配置错误[15].基于签名的方法提取与特定错误配置相关联的程序行为,将其定义为签名,从而诊断配置错误类型[16-17].基于重放的方法(如 Chronus[18], AutoBash[19], Traight[20])在沙箱中尝试可能的配置变化以修复配置错误.基于比较的方法(如 Strider[21], PeerPressure[22])将错误配置与正确配置相比较,根据差别检测配置错误原因.在简化系统配置方面,当前工作可以降低错误配置率的方式有:提供自动化的部署和配置;最小化配置项数量并找出频繁设置的配置项;设置用户友好的配置约束.文献[1]通过实例研究在配置项设计方面提供给软件架构师和开发人员有益经验以供借鉴.ConfValley 是由声明性语言、推理机和检查器组成的通用配置验证框架,以易于软件系统配置[23].

在配置错误修复方面,当前工作通过拒绝错误的配置和打印有用的日志信息来查明错误.Conferr 是用来测试和评估软件系统对人为造成配置错误的恢复能力[24].文献设置要改变的配置项,并给出这些值的建议取值范围,从而修复配置错误[25].ConfDiagDetector 在测试阶段注入配置错误,并观察输出信息,运行时基于配置变异与自然语言处理检测配置错误[26].文献[27]提出了一种数据量感知的内存集群自动配置方法,可有效识别程序的高维配置,通过分层方式组合了多个独立子模型以构建性能模型,采用遗传算法搜索最优配置,从而在给定集群上实现最佳性能.文献[28]用归纳方法调研了运营商对安全配置错误的看法,探讨这类安全问题中的人为因素,定性研究如何达到目标群体并检测错误配置,为减少错误配置的频率和影响提供了建议.文献[29]对 5 种广泛使用的开源软件源代码的配置约束及变化进行例研究,发现配置数据总体的统计、特定类型约束的特征以及配置约束提取的障碍 3 种情况,进而提出建议以自动提取配置约束.MisconfDoctor 通过错误配置测试,提取每个错误配置的日志特征,并构建特征数据库,通过计算新异常日志与特征数据库的相似性来发现潜在的错误配置[30].PCHECK 帮助软件系统早期检测隐性配置错误,分析源代码并自动生成配置检查代码,使用配置值模仿后期执行以捕获错误表现[31].

一些工作关注于配置关联性检测.Rabkin 将配置项分为数字、模式、标识符和其他等 4 种类型,基于静态程序分析学习程序使用配置项的模式以推断配置项类型[8].SPEX 根据软件源码分析控制流图以推断配置项间的控制依赖,并比较语句以推断配置项值的关系,沿着参数的整个数据流路径学习配置模式以确定其语义[32].Encore 使用数据的语法模式和系统的环境信息推断配置项类型,基于机器学习以模板的形式给出配置项关联性[7].与 SPEX 和 Encore 不同,所提出方法仅基于配置文件而不是分析源代码来确定配置项关联,因此与编程语言无关.此外,用一组预定义的正则表达式推断配置项可能的多种类型,而不是仅表示单一的类型信息,具有更强的表达能力.文献[11]基于配置项值的相似性计算其关联概率,同时提出了一些过滤器,例如异值过滤器、非频繁值过滤器和归一化 Google 距离过滤器.然而,这些过滤器在实际应用中受到限制而不能使用.异值过滤器和非频繁值过滤器要求多个服务组件例,在只有一个例的情况下无法应用.另外,标准化 Google 距离过滤器利用 Google 搜索结果中 2 个配置项的出现频率作为过滤度量.然而如果至少有一个是特定应用软件的配置项,则很难找到配置项对的出现.所提出方法分析了配置项的键、值和类型,并且过滤检测不需要额信息,具有准确性与实用性.

5　讨　　论

采用例研究方法系统调研了 3 层架构企业应用,提取了表 2 典型配置的数据类型的正则表达式形式,分析了通用类型关联规则,实验结果及系统实践表明,能够较好解决 3 层架构企业应用的配置关联性检测问题.同时,配置的数据类型和配置类型的关联性规则具有可扩展性,面向不同的应用系统可以在实际运行过程中增量式添加新的正则表达式和类型关联规则.由于难以穷举配置文件中所有数据类型,因此在今后工作中,计划应用自然语言处理或语义分析技术更好理解配置项中的关键字.

方法粗粒度定义了一些通用类型关联规则,覆盖面较窄,因此在今后工作中,计划面向具体应用领域广泛分析更多的开源软件系统,以定义更多领域相关的类型关联规则.方法难以发现 2 个以上配置项之间的关联性,且服务组件之间还可能存在关联性的传递[32],因此在今后工作中,计划应用统计学习和推断技术更准确地发现这类关联性.所提出方法无法发现在程序中被硬编码为常量或变量,而不出现在配置文件中的配置项,因此在今后工作中,计划引入程序分析技术以更全面发现配置信息.

所提出方法设计多个参数设置,如一致关联性中的 α,β,γ,以及阈值过滤中的阈值等.这些参数根据经验设置,实践过程及实验结果表明,能够得到较好效果.分布式系统的多参数设置是一个重要的研究方向,目前已有较多研究成果[33-34],因此未对该方向开展深入研究.在未来工作中,将尝试采用已有基于智能搜索的参数设置方法(如爬山算法)以合理、高效配置参数.

6　结　　论

分布式软件系统在部署、更新或迁移过程中,由于服务组件配置项之间存在着关联性,配置项设置不一致会引发配置错误.人工手动确定配置项的关联性需要跨多个软件的领域知识,既耗时又繁琐.针对该问题,提出了一种基于关联挖掘的服务一致化配置方法,以自动发现服务组件之间配置项的关联性,并基于 2 个典型开源软件系统对其效果进行了评估.

参　考　文　献

[1] Xu Tianyin, Jin Linlong, Fan Xuepeng, et al. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software [C] //Proc of the 10th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2015: 307-319

[2] Xu Gang, Wang Zhan, Zang Dawei, et al. Anomaly detection algorithm of data center network based on LSDB [J]. Journal of Computer Research and Development, 2018, 55(4): 815-830 (in Chinese)
(许刚, 王展, 臧大伟, 等. 基于链路状态数据库的数据中心网络异常检测算法[J]. 计算机研究与发展, 2018, 55(4): 815-830)

[3] Svrdlik Y. Microsoft: 10 things you can do to improve your datacenters [EB/OL]. [2019-02-01]. https://www.datacenterdynamics.com/news/microsoft-10-things-you-can-do-to-improve-your-data-centers/

[4] AWS Team. Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region [EB/OL]. [2019-02-01]. https://aws.amazon.com/cn/message/65648/

[5] Facebook. More details on todays outage [EB/OL]. [2019-02-01]. https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919/

[6] Yin Zuoning, Ma Xiao, Zheng Jing, et al. An empirical study on configuration errors in commercial and open source systems [C] //Proc of the 23rd ACM SOSP. New York: ACM, 2011: 159-172

[7] Zhang Jiaqi, Renganarayana L, Zhang Xiaolan, et al. Encore: Exploiting system environment and correlation information for misconfiguration detection [J]. ACM SIGPLAN Notices, 2014, 49(4): 687-700

[8] Jiang He, Chen Xin, Zhang Jingyi, et al. Mining software repositories: Contributors and hot topics [J]. Journal of Computer Research and Development, 2016, 53(12): 2768-2782 (in Chinese)
(江贺, 陈信, 张静宣, 等. 软件仓库挖掘领域: 贡献者和研究热点[J]. 计算机研究与发展, 2016, 53(12): 2768-2782)

[9] Zhong Linhui, Xie Bing, Shao Weizhong. Supporting component-based software development by extending the CDL with software configuration information [J]. Journal of Computer Research and Development, 2002, 39(10): 1361-1365 (in Chinese)
(钟林辉, 谢冰, 邵维忠. 扩充 CDL 支持基于构件的系统组装与演化[J]. 计算机研究与发展, 2002, 39(10): 1361-1365)

[10] Rama V, Gupta M, Sethi M, et al. Determining configuration parameter dependencies via analysis of configuration data from multi-tiered enterprise applications [C] //Proc of the 6th ICAC. New York: ACM, 2009: 169-178

[11] Xu Tianyin, Zhou Yuanyuan. Systems approaches to tackling configuration errors: A survey [J]. ACM Computing Surveys, 2015, 47(4): 70-82

[12] Rabkin A, Katz R. Precomputing possible configuration error diagnoses [C] //Proc of the 26th ASE. Piscataway, NJ: IEEE, 2011: 193-202

[13] Attariyan M, Flinn J. Automating configuration troubleshooting with dynamic information flow analysis [C] //Proc of the 9th USENIX OSDI. Berkeley, CA: USENIX Association, 2010: 237-250

[14] Zhang Sai, Ernst M. Automated diagnosis of software configuration errors [C] //Proc of the 35th ICSE. Piscataway, NJ: IEEE, 2013: 312-321

[15] Yuan Ding, Xie Yingliang, Panigrahy R, et al. Context-based online configuration-error detection [C] //Proc of ATC. Berkeley, CA: USENIX Association, 2011: 28-41

[16] Yuan Chun, Lao Ni, Wen Jirong, et al. Automated known problem diagnosis with event traces [J]. ACM SIGOPS Operating Systems Review, 2006, 40(4): 375-388

[17] Ding Xiaoning, Huang Hai, Ruan Yaoping, et al. Automatic software fault diagnosis by exploiting application signatures [C] //Proc of the 22nd LISA. Berkeley, CA: USENIX Association, 2008: 23-39

[18] Whitaker A, Cox R, Gribble S. Configuration debugging as search: Finding the needle in the haystack [C] //Proc of the 3rd OSDI. Berkeley, CA: USENIX Association, 2004: 6-19

[19] Su Yayunn, Attariyan M, Flinn J. Autobash: Improving configuration management with operating system causality analysis [C] //Proc of the 15th OSDI. Berkeley, CA: USENIX Association, 2016: 237-250

[20] Tucek J, Lu Shan, Huang Chengdu, et al. Triage: Diagnosing production run failures at the user's site [J]. ACM SIGOPS Operating Systems Review, 2007, 41(3): 131-144

[21] Wang Yimin, Verbowski C, Dunagan J, et al. Strider: A black-box, state-based approach to change and configuration management and support [J]. Science of Computer Programming, 2004, 53(2): 143-164

[22] Wang Helen, Platt J, Chen Yu, et al. Automatic misconfiguration troubleshooting with peer pressure [C] //Proc of the 3rd OSDI. Berkeley, CA: USENIX Association, 2004: 245-257

[23] Huang Peng, Bolosky W, Singh A, et al. Confvalley: A systematic configuration validation framework for cloud services [C] //Proc of the 10th ECCS. New York: ACM, 2015: 19-32

[24] Keller L, Upadhyaya P, Candea G. Conferr: A tool for assessing resilience to human configuration errors [C] //Proc of DSN. Piscataway, NJ: IEEE, 2008: 157-166

[25] Xiong Yingfei, Hubaux A, She S, et al. Generating range fixes for software configuration [C] //Proc of the 34th ICSE. Piscataway, NJ: IEEE, 2012: 58-68

[26] Zhang Sai, Ernst M D. Proactive detection of inadequate diagnostic messages for software configuration errors [C] //Proc of ISSTA. New York: ACM, 2015: 12-23

[27] Yu Zhibin, Bei Zhendong, Qian Xuehai. Data size-aware high dimensional configurations auto-tuning of in-memory cluster computing [C] //Proc of the 23rd ASPLOS. New York: ACM, 2018: 564-577

[28] Dietrich C, Krombholz K, Borgolte K, et al. Investigating system operators' perspective on security misconfigurations [C] //Proc of SIGSAC. New York: ACM, 2018: 1272-1289

[29] Liao Xiangke, Zhou Shulin, Li Shanshan, et al. Do you really know how to configure your software? Configuration constraints in source code may help [J]. IEEE Transactions on Reliability, 2018, 67(3): 832-846

[30] Wang Teng, Liu Xiaodong, Li Shanshan, et al. MisconfDoctor: Diagnosing misconfiguration via log-based configuration testing [C] //Proc of SQR. Piscataway, NJ: IEEE, 2018: 1-12

[31] Xu Tianyin, Jin Xinxin, Huang Peng, et al. Early detection of configuration errors to reduce failure damage [C] //Proc of the 12th OSDI. Berkeley, CA: USENIX Association, 2016: 619-634

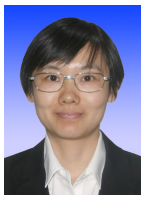
[32] Xu Tianyin, Zhang Jiaqi, Huang Peng, et al. Do not blame users for misconfigurations [C] //Proc of the 24th SOSP. New York: ACM, 2013: 244-259

[33] Zhang Fan, Cao Junwei, Liu Lianchen, et al. Fast autotuning configurations of parameters in distributed computing systems using ordinal optimization [C] //Proc of ICPPW. Piscataway, NJ: IEEE, 2009: 190-197

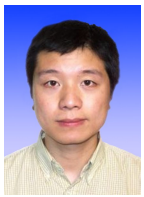
[34] Bilal M, Canini M. Towards automatic parameter tuning of stream processing systems [C] //Proc of SoCC. New York: ACM, 2017: 189-200

Wang Tao, born in 1982. PhD, associate professor. Senior member of CCF. His main research interests include fault diagnosis, software reliability, and autonomic computing for cloud computing systems.

Chen Wei, born in 1980. PhD, associate professor. Member of CCF. His main research interests include data-driven software engineering, software configuration, micro-service and cloud computing.



Li Juan, born in 1977. PhD, associate professor. Her main research interests include requirements engineering, big data analysis and cloud computing.



Liu Shaohua, born in 1976. PhD, associate professor. His main research interests include distributed computing, cloud computing and Internet of things.

Su Lingang, born in 1994. Master candidate. His main research interests include service computing and cloud computing.

Zhang Wenbo, born in 1976. PhD, professor. Member of CCF. His main research interests include distributed computing, cloud computing and middleware.