软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(8):1934−1947 [doi: 10.13328/j.cnki.jos.004936]

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

大规模软件系统日志研究综述*

廖湘科, 李姗姗, 董 威, 贾周阳, 刘晓东, 周书林

(国防科学技术大学 计算机学院,湖南 长沙 410073)

通讯作者: 李姗姗, E-mail: shanshanli@nudt.edu.cn

摘 要: 规范和充分的日志是良好代码质量的必要因素,也是软件故障诊断的重要手段.然而,代码的质量管理受限于大规模软件代码的高复杂程度,目前,利用日志信息进行软件故障重现和诊断的难度大、效率低.从日志特征分析、基于日志的故障诊断、日志的增强这 3 个方面综述了日志研究的现状.通过对几种常用的大规模开源软件的日志进行调研,发现了一些日志相关的特征和规律以及现有工具难以解决的问题.最后,对未来的研究工作进行了展望,并分析了可能面对的挑战.

关键词: 系统日志;特征分析;故障诊断;日志增强

中图法分类号: TP311

中文引用格式: 廖湘科,李姗姗,董威,贾周阳,刘晓东,周书林.大规模软件系统日志研究综述.软件学报,2016,27(8):1934−1947. http://www.jos.org.cn/1000-9825/4936.htm

英文引用格式: Liao XK, Li SS, Dong W, Jia ZY, Liu XD, Zhou SL. Survey on log research of large scale software system. Ruan Jian Xue Bao/Journal of Software, 2016,27(8):1934−1947 (in Chinese). http://www.jos.org.cn/1000-9825/4936.htm

Survey on Log Research of Large Scale Software System

LIAO Xiang-Ke, LI Shan-Shan, Dong Wei, JIA Zhou-Yang, LIU Xiao-Dong, ZHOU Shu-Lin

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

Abstract: Standardized and sufficient log is a necessary part of good code quality, and it plays an important role in failure diagnosis as well. Code quality management, however, is restricted by the high complexity of large-scale software. Currently, it's difficult and inefficient to reproduce and diagnose system failure with logs. This paper surveys log-related work from three aspects including log characterization, failure diagnosis with log and log enhancement. Through detailed study on several widely-used open-source software, the paper reveals some log-related observations, along with the problems which have not been well handled by existing tools. Finally, it proposes several possible log-related work, and analyzes potential challenges.

Key words: system log; log characterization; failure diagnosis; log enhancement

大规模软件系统,特别是国产基础软件正广泛应用于政府、金融、航天、军事等关键应用中,具有规模庞大、自身逻辑复杂、可靠性要求高、出错后难以分析和调试等特点.这些软件大都来源多样,集成了大量开源社区的代码,我们称其为混源软件系统.例如,国产操作系统基于开源软件进行改造和扩展,加入特定的代码并剪裁掉不需要的功能代码;具体应用可能采用商用软件(如办公软件、图像处理软件等)和专门开发的应用软件(如军事指挥控制软件等).然而,开源软件的开发缺乏集中管理,代码质量参差不齐,项目难以预测和控制,导致最终的混源软件系统质量缺乏统一的标准.另外,许多部门还保留了一些使用多年的遗留软件,这些软件往往缺乏

* 基金项目: 国家自然科学基金(61402496); 国家重点基础研究发展计划(973)(2014CB340703); 腾讯高校合作项目

Foundation item: National Natural Science Foundation of China (61402496); National Program on Key Basic Research Project of China (973) (2014CB340703); Tencent Cooperation Projects in Universities of China

收稿时间: 2015-06-17; 修改时间: 2015-08-24, 2015-10-13; 采用时间: 2015-10-27; jos 在线出版时间:2015-11-06

CNKI 网络优先出版: 2015-11-09 15:26:51, http://www.cnki.net/kcms/detail/11.2560.TP.20151109.1526.001.html

清晰的文档和结构良好的代码,开发运行环境也发生了变化.同时,随着软件规模的增大和版本的推进,软件中存在的代码缺陷越来越多,开源软件的可靠性受到越来越严重的威胁,表 1 给出了 Coverity 公司对典型的开源软件 Linux 操作系统进行可靠性分析后形成的报告[1].因此,混源软件系统的形成已经超越了传统软件的开发过程,其代码质量以及质量保证方法也因不同来源软件特征的差异以及系统的复杂程度,与传统软件质量保证过程存在明显不同,很多情况下需要对其来源、质量等方面进行单独考虑.

Table 1 Coverity 2012 Linux operating system analysis report[1]

表 1 Coverity 2012 Linux 操作系统分析报告[1]

时间	版本	代码量	新缺陷	已修复缺陷
2006	2.6.16	3 451 730	1 264	435
2007	2.6.16	3 458 369	425	217
2008	2.6.27	4 202 209	596	365
2009	2.6.32	4 862 567	527	417
2010	2.6.33 2.6.34 2.6.35	5 504 780	3 858	462
2011	2.6.38 2.6.39 3.0 3.1	6 849 378	2 331	1 283
2012	3.2 3.3 3.4 3.5 3.6 3.7	7 387 908	5 803	5 170

目前,大规模软件的故障诊断主要依靠人力排查,定位问题不及时且效率低.在多数大型软件系统中,由于隐私保护和系统环境设置等因素,错误重现和调试都十分困难.例如,开发人员一般缺少用户的输入和文件,或者难以重现相同的执行环境;同时,大型系统上的应用通常要并行处理用户提供的大数据集,运行相当长的一段时间,还可能部署在大型分布式系统环境中,不确定性因素很多.如果能从软件代码本身提高质量,在故障发生时提供用户或开发人员必要的提示信息,则可以有效地提高故障诊断和恢复的效率.系统日志是故障诊断的重要手段,它可以记录程序运行时的动态信息,帮助维护人员分析重现错误,进而更正系统错误,提高系统运行的可靠性.规范和充分的日志对于软件故障诊断非常重要,也是良好编程规范的必要因素.

关于软件系统日志,业界已经推出和制定了许多规范,但是编程人员在开发过程中多数是依赖经验和个人习惯,缺乏统一的被普遍接受的标准,通过日志进行故障诊断并非易事.然而有研究表明[2,3]:通过对部分大规模软件(Apache,Squid,PostgreSQL,SVN 和 Coreutils)的系统失效报告进行几百次随机采样,定位错误发生的位置,诊断上下文相关信息,发现大部分(77%)的系统失效都可以归结为几类常见的错误诊断模式,如返回错误码、switch 语句的默认处理等.而这些错误中有超过一半(57%)没有进行日志记录,导致维护人员需要花费大量的时间去定位和分析这些错误.由于代码量巨大,手动对软件代码进行分析和添加日志几乎不可能完成,因此,如何自动化地实现日志的分析、插入和管理,维护和增强大规模基础软件的可靠性的需求越来越迫切,同时也面临重大挑战.

本文拟对大规模软件日志研究现状和进展进行综述.首先,简要介绍故障诊断技术,分析所涉及的研究内容.然后,分别从日志特征分析、基于日志的故障诊断、日志的增强这 3 方面综述了当前日志的研究现状,并对它们进行了分析分类.针对大规模软件的代码质量需求以及来源异质等特点,对现有大规模开源软件代码进行调研和人工分析.本文分析和发现了一些日志相关的特征和规律,以及现有工具难以解决的问题.最后,本文对未来的研究工作进行了展望,并分析了可能面对的挑战.

1 故障诊断的主要技术

故障诊断**是用于发现系统或系统组件中不正常现象的一种技术.随着软件系统的规模不断变大,逻辑变得更加复杂,故障诊断的难度也越来越大.一方面,大规模系统中不一定具备细致的监控能力;另一方面,由于一些容错机制的存在,故障有时并不会直观表现出来.故障诊断技术主要可以用于发现系统的不足.

当前的故障诊断技术主要依赖于人工枚举的大量输入,即枚举可能引发错误的各种测试用例,以尽可能地

** 本文中的故障主要指问题出现的根源,可能是硬件故障、软件 bug 或者是配置错误等.故障可以引发系统的不正常行为,比如异常或者系统调用返回出错.由于容错机制的存在,一些错误可能对用户不可见,但有些错误有可能会导致系统失效,如宕机.

暴露故障的原因和现象,缺乏针对性,效率低、不完备.因此,这个过程的自动化技术一直是故障诊断的研究热点.

故障诊断的自动化技术辅助开发人员定位问题可能的源头、可能出现的地方以及可能的故障类型,为故障诊断的过程进行初筛和预判,可以有效提高故障诊断的效率.已有的技术还不成熟,精确度也不高,常常会出现误报(false positive)或漏报(false negative)问题.因此,如何提高精度也成为了故障诊断自动化技术的主要研究目标之一.

当前,故障诊断技术正不断融入新的计算技术和数学方法,包括人工智能、机器学习、随机过程、贝叶斯推断、图论等.表 2 列举了主要的故障诊断技术及其优缺点.

Table 2 Summary of diagnosis techniques[4]

表 2 诊断技术汇总[4]

技术	定义、优势与局限
基于规则的技术	基于规则的技术主要通过将专家知识表示为一系列规则来进行故障诊断.规则是人为可扩展和可解释的,但这种技术不能诊断未知的错误,而且大量的知识库也不易维护.
基于模型的技术	基于模型的技术将系统定义为数学表示,通过测试观察到的行为来验证是否满足模型.基于模型的技术适合诊断应用级别的问题,然而,构建模型需要对系统有非常深刻的理解.
统计技术	统计技术通过对经验数据使用关联分析、对比和概率等理论来进行故障诊断.统计技术不需要对系统内部或者模型具备深入的了解,但是对于系统的非稳态(意料之外,情理之中)故障难以诊断,这类故障对于大规模系统而言是很常见的.
机器学习技术	机器学习技术采用聚类的方法识别模式,或者使用训练数据来确定系统状态是否健康,找出故障的潜在原因.机器学习技术可以自动地学习系统行为,但是当特征维度变大时,精确度会迅速下降.
计数和阈值技术	计数和阈值技术可以诊断出短暂和间歇性错误,这种技术在很大程度上依赖于参数的校正,可通过严格的数学公式和分析模型的配置参数.
可视化技术	可视化技术通过可视化数据的趋势和模式来识别异常点.它可以对问题出现的根源有各种假设,但是这种技术并不能自动识别问题.

故障诊断主要是发现系统中的不正常现象,而日志作为一种手段,主要目的就在于如何暴露这些不正常现象.在故障诊断过程中,日志主要用于记录程序执行的痕迹和线索,以帮助程序开发人员快速定位问题.文献[5]对微软 54 位有经验的开发人员做了调研,他们中的大多数(大概 96%)高度认同日志在系统开发和维护中的重要作用,认为日志是故障诊断的主要信息来源.然而,由于没有严格的日志规范或者要求,现实系统中的日志还相对随意和主观,业界人员对日志规范化的重视程度也不够高.开发人员往往按自己的习惯和经验决定在哪里加日志、加什么日志.即使存在规范,依然缺乏被普遍接受的标准.很多情况下,都是系统发生错误后,开发人员才把日志加入到代码中.下一节,本文将介绍日志相关的研究工作.

2 日志相关研究

目前,系统研究日志的工作并不多,主要从日志特征分析、基于日志的故障诊断和日志的增强这 3 个方面展开.

2.1 日志特征分析

文献[3]系统和量化地观察了 Apache httpd,OpenSSH,PostgreSQL 和 Squid 等常用开源软件过去 10 年间日志的修改内容和修改历史,对大规模开源软件进行了日志特征分析,对日志的质量和开发者的要求进行评估,分析和总结日志中值得关注的问题以及开发者需要注意的地方,为日志工具或语言支持工具的设计提供启发.分析结果表明,目前,在软件开发中进行日志记录是普遍的,平均 30 行代码中就有一行是日志;日志信息对实际部署系统的运行故障调试帮助较大,缩短故障调试时间的加速比为 2.2;日志代码的更新频率比其他代码要快约 1 倍,开发者较为重视日志代码的更新和维护;尽管日志代码在全部代码中所占的比例小,但是日志代码在软件版本进化过程中更新的比例比其他代码要大;开发者需要时会随时撰写日志代码,而之后往往会根据其实际效果对其进行更新;一般缺乏对日志消息进行解释的文档,使得开发者对日志消息的删除或移动持保守态度;开发者需要花大量精力确定错误事件发生的临界条件,并输出较多冗余日志,测试和代码分析工具需要更多地为开发者提供恰当的出错临界条件信息;约四分之一的日志修改是把新的程序变量写入日志,意味着开发者需要添加

更多程序运行时的动态信息来理解故障发生的原因;约一半的日志修改是对日志消息静态文本的修改,通常是为了处理日志消息和程序实际执行过程的不一致性问题,将自然语言处理和静态代码分析工具相结合,有望实现这种不一致性检测的自动化.这些特征分析对日志未来的研究工作有重要的指导作用.

微软的研究人员[5]认为,日志对于故障诊断起着非常重要的作用.日志既不能过多或也不能太少:过多会增加系统的运行开销,而且不利于挖掘有用信息;太少则会遗漏重要的调试信息.他们希望系统研究已有的日志开发的实践经验,使开发人员对日志有更清晰的认识,以帮助他们决策在哪里加日志、加什么内容.他们对微软开发的两个广泛使用的软件进行了代码分析,并对微软内部 54 个经验丰富的开发人员进行问卷调查,提问包括:哪些类型的代码段会加日志?加日志时应该考虑哪些因素?自动判断在哪加日志是否可行等.通过分析,对日志的常见出现位置、开发人员在代码中加日志的现状等问题有一些发现,并得出结论:添加日志的位置主要在函数返回、异常捕获以及一些程序相关的关键点等位置.

此外,还有部分从特定系统中分析日志特征的工作.文献[6]主要针对 10 000 个商业存储系统中的 636 108 个用户案例进行分析,总结了故障诊断的特征(软件故障难以诊断,诊断软件失效比硬件故障和配置错误要花费更长时间),同时观察到:在存在日志信息的情况下,故障诊断速度有明显提升.同时,文中分析单独的软件失效时的日志信息难以帮助诊断错误根源,多个事件日志和失效日志结合能够明显提升诊断速度.

2.2 基于日志的故障诊断

当软件运行出现问题时,可以通过日志推断出问题出现的原因和位置.最理想的情况是,开发人员期望用最少的输入(例如运行环境、系统的输入和输出、代码等)得到精确的诊断信息.但是目前,利用日志进行故障诊断的效率并不高:一方面,故障的分析一般强依赖于系统的运行环境和系统输入,而这些信息往往难以复现,可获得的信息量也有限;同时,软件系统的日志量庞大,而且日志信息缺乏结构化,难以自动化地提取有价值的信息.目前的故障分析主要依赖开发人员手工展开,很多时候还需要分析代码上下文.因此,一些研究工作借助程序分析方法研究基于日志的自动化故障诊断.同时,还有研究工作对日志能力展开了评估,包括分析影响日志能力的因素、如何评估日志的能力等.

控制台日志(console)因为其典型性、数量大、无结构而被许多工作首选作为研究目标.文献[7]设计了一种基于规则的方法处理控制台日志并检测故障,规则需要管理员预先用脚本的方式写好,这种情况下,管理员要对系统或者代码有深刻的理解,否则难以写出有效的脚本,定位故障的效率较低.文献[8,9]设计了一种模式挖掘方法,从 SNMP 数据中挖掘一些特定模式用于检测故障,如消息爆炸、周期性消息或多个消息之间的关联关系.文献[10]将 syslog 建模成了一个隐式马尔可夫序列,用于将消息和故障相关联.文献[11]通过为大规模电话系统的日志构建启发式过滤器将消息和故障关联.文献[12]通过构建有限状态自动机来对合法执行的行为进行建模,并与错误执行进行对比,给出导致错误的可能的异常事件.

这些工作的局限在于,他们将故障相关联的日志看成是一个时间序列.然而,在有大量独立进程并发运行的大规模系统中情况会复杂很多,当不同进程的日志消息交织出现时,模型会变得异常复杂.还有部分研究工作只是将日志单纯地看成一组英文单词组[10,11,13].

与这些工作不同,Xu 等人将与故障关联的日志看成一组消息,而非简单的英文单词组或时间序列,这样可以更精确地推测程序的执行,检测系统异常.他们系统地研究了如何自动化地从大量无结构化的控制台日志中挖掘故障信息[14].他们首先设计了一种离线的方法,采用源代码分析技术和信息检索技术理解控制台日志的结构如状态变量和对象标识,提取日志的特征,并设计了图形化的方式呈现[15].通过这些特征可以获取日志之间的关联关系,然后再采用数据挖掘和机器学习技术在这些特征上进行异常检测,定位故障的源头.具体来说,通过主要组件分析(PCA)的方法检测异常;通过采用决策树的方法,将检测的结果以用户可理解的方式呈现[16,17].在文献[18]中,他们又通过折中检测精度和时间,将方法扩展成了在线分析方法.他们还在接下来的工作中将上述的方法应用于 google 系统,并阐述了在超大规模系统中利用这些方法进行异常检测所面临的新挑战[19].

文献[20]设计了 LogSig,从文本日志信息中生成系统事件信息.通过分析发现,源码中日志输出语句中的固定子序列可以作为事件类型的签名,通过对具有代表性的签名进行分析,LogSig 算法可以将日志信息分类不同

的事件类型.文献[21]认为:在不同输入和负载下,系统的内部存在一种常量线性关系,可以作为程序的不变式.通过统计学习技术从控制台日志中挖掘出来这种常量线性关系,当新的日志信息破坏了这种不变量时,即可认定系统出现异常.实际实现时,首先使用日志解析器将非结构化日志解析为特定结构化日志;然后,依据不同日志参数将日志信息进行分组;接下来,可以自动从日志信息组中挖掘程序不变式;最后,通过与不变式进行对比发现系统异常.文献[22]针对大规模集群系统(如 HPC)的系统日志特点设计和实现了 Logmaster,展开了事件相关性挖掘.Logmaster 首先将日志解析成 n 维序列,其中每个事件被定义为一个 9 元信息组,并提出了 Apriori-LIS 算法进行事件挖掘;然后,设计提出事件相关图(events correlation graph,简称 ECG)来表示事件发生的时序.文献[23]提出了一种针对存储系统的低开销实时运行时异常检测方案,文中定义 stage 为存储服务器中的不同线程和不同模块的基本单位,并针对 stage-level 的日志 summaries 进行运行时分析;然后,使用统计分析方法进行跨 stage 实例的分析.通过对比,可以筛选出生成特殊执行流或者占用过长持续时间的 stage,这些即为可能的异常 stage.

文献[24]针对并发系统的故障诊断困难的问题,提出了一种新的模型推断技术 CSight.CSight 可以从系统运行的日志中以通信有限状态自动机(CFSM)的形式导出一个简单但精确的系统行为模型,以用于检测异常或系统调试.CSight 方法要求日志事件包含向量时间戳(vector timestamp).文中通过对 3 个不同网络系统进行实验以及用户调查来进行模型方法的验证.

一些研究工作针对现在广泛使用的分布式计算服务展开了基于日志的故障诊断研究.Fu 等人认为,大规模分布式系统执行异常主要包括两方面:工作流错误和性能降低[25].他们提出一种非结构化的日志分析技术来进行异常检测,假设每个日志信息均包含时间戳和线程 ID 或请求 ID 以区分来自不同线程,并设计了一种算法将日志无结构的文本信息转化成特定的关键字格式;然后,从这些日志关键字序列中学习建立有限状态自动机模型,以表示正常工作流;同时,基于日志相关时序信息建立系统性能度量模型.基于上述两个模型,系统维护者可以通过对比迅速发现系统中出现的执行异常.

Zhao 等人针对 HDFS,Cassandra,HB 等云服务设计了性能诊断工具 lprof[26],他们认为,理解服务请求的性能行为是进行性能诊断最关键的部分.因此,lprof 能够根据运行时日志自动地重建每个请求的运行流,由此推断出可能影响性能的每个阶段.它主要通过统计分析应用程序的二进制代码,判断哪些分散和交织的日志能组合关联到某个服务请求.在分布式系统中,由于这些服务是分布在许多节点上,而且每个服务都由多个子系统组成,如前端、缓存、后台数据库等,因此,实现日志分析并非易事.与文献[14]的工作类似,lprof 也是采用了机器学习的方法对日志进行自动化分析.不同的是,lprof 重点针对的是服务请求流的分析性能问题.实际上,lprof 可以和这些方法合并使用.SALSA[27]和 Mochi[28]也通过日志分析 Hadoop 的请求流,但它们需要手工产生模型,标识请求的开始、结束等等,而 lprof 则是全自动地完成整个过程.

针对网络应用的可用性问题,Li 等人提出了一种通过分析 Web 日志检测用户可见(user-visible)错误的方法[29].在 Web 应用出现问题时,用户经常不能及时正确地反馈信息给管理员以获得维护支持,因此,本文假设用户使用 Web 应用时满足一定行为模式.当应用出现错误时,操作行为会打破这种行为模式.具体实现时,作者采用一阶马尔科夫模型对用户行为建模,使用已有的 Web 日志对模型进行训练,并使用训练的模型对用户不正常行为进行推断.

文献[30]设计实现了一个自动工具 distalyzer,用来分析分布式系统的性能问题.首先,给定两个分布式系统的日志集合,一个系统具有高性能表现,另一个具有较差的性能表现;然后,distalyzer 从日志中提取系统行为并使用机器学习方法比较,自动推断出系统组件与性能之间的最强关联关系.distalyzer 输出结果包括:(1) 两个系统日志中差别最大的事件和相关程序变量值;(2) 对系统总体性能影响最大的事件和变量值.

Fu 等人在已有工作[25]的基础上提出了一种日志文件的上下文分析方法来理解系统行为[31].通过定义执行模式的概念来反映系统的运行行为和执行路径,并提出了从日志信息中挖掘执行模式的算法.通过对执行模式的挖掘,系统管理员可以进一步理解系统在不同条件下的执行路径以及相应的上下文信息(如某个特点参数的取值),为执行例如错误诊断的任务提供帮助.

在软件出现故障时,即使提取了日志信息诊断也并非易事,比如缺乏用户输入、难以重建相同的运行时环境或者并行运行环境的不确定性等.协同调试(cooperative debugging)广泛应用于商业软件,例如,Windows Error Reporting[32],Mozilla Quality Feedback Agent[33]通常利用有限的内存转储文件(出于隐私考虑)对错误报告进行收集,通过大量的统计数据得出经常出现的错误失效模式.但由于内存转储不能反映当时执行相关的上下文,从而无法通过其诊断失效的根源.文献[34]设计了工具 SherLog,通过分析程序运行产生的 trace,结合源码进行确定性重演[35−38]来定位发生软件错误时的执行路径和上下文,以减少故障诊断难度.SherLog 既不需要重新运行软件也不需要了解日志的语义,它主要采用路径和上下文敏感的程序分析技术提取有用的控制和数据信息.为了提高工具的可扩展性,SherLog 还采用了一个可满足的约束解析器进行剪枝,以支持更大规模的软件,如操作系统等.

在日志能力的评估方面,文献[39]分析了用日志进行故障诊断时影响精度的因素.分析表明,日志的检测能力不受失效类型的影响.而对于采用类似日志机制的不同软件,其日志的检测能力却存在明显差别,原因在于:不同的软件系统架构、日志分布以及执行环境存在差异,均会有显著影响.文献[40]利用故障注入方法比较了日志与其他失效检测手段在 Web 应用下的能力.实验指出,日志信息能够有效诊断资源枯竭和程序执行环境引起的失效,但难以应对应用相关的问题.文献[41]分析了现有日志机制中存在的限制,并提出了一种基于规则的方法提高日志的有效性.该方法从软件需求中提取出一系列规则,用于定位需要加日志的地方.文献[42]提出了一种通过错误注入来评估日志能力的方法,通过错误注入,可以针对系统:(1) 检测出常见错误模型;(2) 识别日志不足与缺陷;(3) 为日志改进提供指导.论文分别针对 apache Web server,tao open data distribution system,mysql dbms 进行实验验证,实验结果表明,仅有 35.6%~42.1%的错误存在日志记录,大量的软件错误并没有生成任何日志记录.

2.3 日志的增强

日志的增强主要包括日志条目的增加和日志内容的增强.现有的大规模软件大都是多人开发,或者采用多种来源的软件,集成了大量开源社区的代码,软件内部的代码风格存在许多不一致,并不是该有日志的地方都加了日志,一些地方即使加了日志,信息量也不够充分,难以达到故障诊断的目的.因此,一些研究工作考虑如何为软件增加规范的日志、如何增强已有日志的信息量等.另外,软件中的日志并不是越多越好[5]:首先,太多的日志会消耗许多的运行时开销,如 CPU、I/O、存储开销等;其次,太多的日志可能是冗余和无用的,不利于从中挖掘有用的信息,即有时候过多的信息量意味着没有信息量;另外,日志本身也是代码,有开发和维护的成本.

文献[2]重点研究了日志的自动化添加技术,设计并实现了工具 Errlog.它认为,软件系统的出错情况主要可以分成几类:函数返回错、访存安全检查出错、Switch 语句不完备、异常信号处理、输入合法性验证出错、溢出、非正常退出以及其他原因等,如图 1 所示.

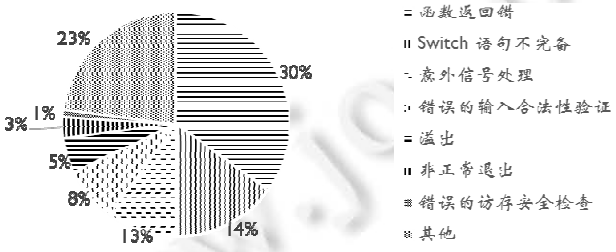


Fig.1 Classification of common software exceptions[2]

图 1 软件系统出错情况的分类[2]

根据错误分类找到其故障原因,更容易针对性地确定日志添加位置.进一步地,又将错误分为两类:程序检测到的错误与程序未检测到的错误.程序检测到的错误即异常,如代码库检查异常、系统调用返回值异常等,它们有可能会被异常处理程序正确处理,也有可能无法正确处理从而导致失效;程序未检测到的错误,如不正确的

变量值等,开发人员无法事先预测,最终会导致系统失效.当程序检测到的错误时,可能有 3 种处理结果:一是程序发现该异常后提前退出;二是程序正确处理该异常;三是程序不能正确处理该异常从而导致失效.不管是哪种情况,在程序检测到错误的位置添加日志是较好的选择,因为即便进入正确的异常处理程序,异常处理的过程也可能是有漏洞的(目前,针对异常处理程序的测试极少),其他两种情况就更需要系统日志.Errlog 在这些分析的基础上,采用了基于控制流和数据流的分析方法去识别程序中潜在的日志添加点,并自动添加日志.LogAdvisor[43]利用机器学习的方法总结软件开发中已有日志行为的上下文特征,经过特征减重和噪声控制处理后,建立了良好的日志预测模型.通过集成于 IDE 中,可在软件开发过程中在线地为程序员提供是否需要加日志的建议.

Zhou 等人设计了 LogEnhancer[44],通过对日志内容进行增强,减小软件不确定分支的数目,从而降低分支临界条件判断和错误重现的难度,提高诊断效率.具体来说,假定有一段包含多个分支的代码,存在较多的分支语句使得该程序难以调试,这时,将条件变量写入到日志信息中去,就能通过日志信息中条件变量的值判断之前条件语句的判断结果,从而消除一段不确定分支.需要指出的是,即使采用了 LogEnhancer,大部分的日志行为还是需要由开发人员来决策.目前,大规模软件的日志虽然量大,但日志里包含的信息量并不充分,开发者通常需要添加更多程序运行时的动态信息来理解故障发生的原因,即频繁地更新日志,向日志消息中不断加入新的程序变量.研究自动对日志信息进行增强的方法,可以极大地降低开发者的工作量,提升代码调试的效率.

为了帮助开发人员理解日志,文献[15]提出了一种控制台日志的可视化方法,展示了系统中每个日志实体所记录的信息量以及实体之间的联系,并能够对比分析不同应用或者同一应用下的不同配置的日志结构,增强了日志的展现能力.文献[45]为提高系统日志的展示能力给出了一组建议,包括区分引发日志的事件类型、设计分层的日志编码体系、合理分类日志信息、设计日志功能时要考虑信息的密度等.文中还提出了一种度量日志信息熵的方法.

2.4 日志相关研究分析

本节对日志相关研究方法进行梳理,将从基于故障诊断方法分类、应用领域分类和研究目标分类这 3 种分类方法展开.

2.4.1 基于故障诊断方法分类

结合第 1 节中故障诊断技术的分类,本节进一步对基于日志的故障诊断研究进行分类.其中,大部分工作采用基于模型的技术和机器学习技术.除表 3 所列的诊断方法外,基于日志的故障诊断还涉及其他技术,如程序分析[31,35]和错误注入[41,43]等.

Table 3 Categories of log research based on diagnosis techniques

表 3 基于故障诊断方法的日志相关研究分类

故障诊断技术	相关研究工作
基于规则的技术	[7,41]
基于模型的技术	[8–12,20,24,25,27–29,31]
统计技术	[21,23,26]
机器学习技术	[14,18,19,22,30]
可视化技术	[15–17]
其他技术	[34,40,42,44]

2.4.2 应用领域分类

基于日志相关工作研究的领域,可分为控制台日志、分布式计算服务、网络应用、存储系统、并发系统以及一般性系统,见表 4.

控制台日志是典型的日志形态,现有相关工作主要通关关联系统故障和日志消息,基于规则和模型的方法实现故障诊断.在分布式并发系统方面,主要基于日志进行故障诊断或者性能诊断,实现对于复杂系统的故障快速检测和处理.在网络应用方面,重点是针对可用性问题,利用日志分析用户行为,或者通过错误注入对比了日志检测能力和其他检测手段.在存储系统中,主要通过观察日志信息来提升故障诊断速度.在并发运行的大规模

系统中,不同进程的日志交织会导致系统日志分析变得异常复杂,利用日志进行异常检测或系统调试难度增加.还有工作研究可针对多种软件系统进行日志相关的故障诊断,这类工作具有一定的普适性,更加关注基于日志进行故障诊断的一般特征.

Table 4 Categories of log research based on fields and target

表 4 基于应用领域和研究目标的日志相关研究分类

应用领域	研究目标		
	功能性故障分析和诊断	性能分析	日志诊断能力分析
控制台日志	[7−12,14,21]	–	[15]
分布式计算服务	–	[25−28,30]	–
网络应用	[29]	–	[40]
存储系统	–	[23]	[6]
并发系统	[24]	–	–
一般性系统	[2,44]	–	[3,39,42]

2.4.3 研究目标分类

针对不同的研究目标,现有研究工作可以分为功能性故障分析和诊断、性能分析以及日志的诊断能力分析,见表 4.

从研究目标上看,功能性故障分析和诊断是当前的研究重点,包括面向系统崩溃、抛出异常或错误信息等功能异常,系统运行时的被动诊断为主要方式,即在软件运行出现错误之后,通过对系统日志的分析进行故障诊断.性能问题的分析这几年受到越来越多的关注,尤其在分布式计算服务领域中,性能问题是影响系统可用性的关键因素.一般通过运行时日志重建系统的运行过程,并由此推断出可能影响性能的每个阶段.此外,还有部分工作研究日志自身的错误诊断能力,主要采用规则和错误注入等方法,通过向系统中注入不同类型的错误代码,进一步分析已有日志检测注入错误的能力.

3 观察、发现及现有工具的困难

现有日志相关工作主要集中于利用已有日志进行故障诊断,绝大多数工作未考虑现有的日志机制和日志质量.少数日志特征分析工作仅关注日志的密度、更新频率、日志内容修改等直接特征的统计数据,并未从开发人员因素(如开发风格、错误处理方式)、性能因素(如决定是否加日志、日志分布)以及软件内部环境因素(如日志行为的上下文、语义相关缺陷)等方面考虑.本文通过对软件缺陷特征的调研和 Httpd,MySQL,Squid,CentOS 等大量开源代码的人工分析,分析和发现了一些日志相关的特征和规律以及现有工具难以解决的问题.

3.1 开发风格不一致

我们在前期调研和分析中发现,即使被广泛应用的成熟的开源软件,仍然存在很多不规范、不清晰的日志代码编写风格.特别地,由于不同的代码开发人员的编写习惯不一致,导致的某些关键代码位置缺乏日志,难以人工检查.图 2 给出一个返回值为非 bool 型的例子,左图采用 if 的形式,而右图采用 switch 的形式.由于代码风格不同,又存在多个返回码,导致一些地方日志缺漏不容易被发现.在未来的研究工作中,可以借助程序分析工具,指导自动化的日志判别、匹配和分析.

```
/* Httpd-2.0.65/modules/proxy/proxy_ftp.c */
rc = proxy_ftp_command(apr_psprintf(p, "PORT %d,%d, ..."
if (rc == -1 || rc == 421) {
        return ap_proxyerror(…);
}
if (rc != 200) {
        return ap_proxyerror(…);
}
```

```
/* Httpd-2.0.65/modules/proxy/proxy_ftp.c */
switch (proxy_ftp_command("PWD" CRLF, r, ftp_ctrl, bb, ...
        case -1:
        case 421:
        case 550:
            ap_proxyerror(…);
            break;
}
```

Fig.2 Different coding styles

图 2 编码风格不一致

同时,我们还发现函数的异常处理方式具有很强的文件相关性.一般地,一个文件由一个开发人员书写.在大规模软件中,由于多人开发的风格不一致,对相同函数的异常处理也存在不同.以 C/C++程序为例,常见 3 种函异常误处理分为 3 种模式:(1) 记录错误码;(2) 返回错误码;(3) 输出日志信息,或者是多种方式的综合.图 3 给出了 Linux-3.12 内核 ext4 文件系统中不同文件对 sb_bread 函数的异常处理方式.

这种不一致的情况在现有的工程代码中很多,可以设计和实现相应的自动化的工具,通过统计分析,为一些潜在的出错位置加上日志.

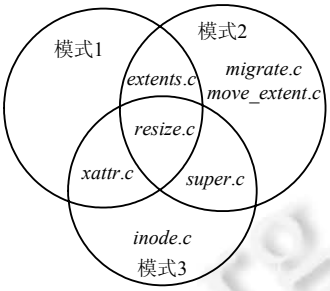


Fig.3 Exception dispose of sb_bread in different files in linux-3.12/fs/ext4

图 3 Linux-3.12 内核 ext4 文件系统中不同文件对 sb_bread 函数的异常处理

3.2 日志行为受上下文影响

我们在对实际项目的日志特征学习中发现,函数是否需要加日志、加什么日志常常与函数的上下文相关.例如图 4 中的函数 ap_regkey_open,有的地方在它执行之后加了日志,而有些地方却没有加.这对于自动统计日志行为带来了一些困难.通过统计,31%(5/16)的调用加了日志,当第 4 个参数包含 APR_CREATE 时,80%(4/5)的调用加了日志,否则只有 9%(1/11)的调用加了日志.即当第 4 个参数包含了“APR_CREATE”时,大部分都加了日志函数;反之则没有加,见图中的右例.进一步通过人工审查,涉及到 APE_CREATE 的操作与创建相关,应该加日志,符合常理.因此,对于函数的检查,除了单纯地查看函数名、返回值以外,还应该考虑上下文等因素.

文献[5]对日志函数的上下文进行了分析,提出了通过搜索代码中的关键字(如 delete,load,remove 等)对预测日志添加位置准确率的影响,这要求软件中有非常好的编码风格,并不具有普适性.此外,日志还可能与敏感信息有关,例如密码文件的读写、信用卡账号的处理等.

```
/*Httpd-2.0.65/server/mpm/winnt/service.c */
rv = ap_regkey_open(…, APR_READ, …);
if (rv != APR_SUCCESS) {
        mpm_display_name = apr_pstrdup(p,
set_name);
}
```

```
/* Httpd-2.0.65/server/mpm/winnt/service.c */
rv = ap_regkey_open(…, APR_READ | APR_WRITE | APR_CREATE, …);
if (rv != APR_SUCCESS) {
        ap_log_error(…Failed to create the registry service key."...
        return (rv);
}
```

Fig.4 Log behavior is influenced by the 4th argument of ap_regkey_open

图 4 ap_regkey_open 之后的日志行为受其第 4 个参数的影响

3.3 并非所有的潜在出错点都需要加日志

一般情况下,内存分配函数执行以后都要加上日志,以检查空间是否分配成功.但在一些项目中,由于性能考虑,未必都加了日志.图 5 给出了 Httpd-2.4.10 中的一个例子,apr_proc_create 后,并未添加日志.我们调研了 Httpd 的 bug 管理库和邮件列表,发现不少开发人员指出这个问题,但是 Httpd 的开发人员回复这种处理方式主要是出于性能考虑.因此,是否需要添加日志并非完全遵从固定的模式,需要从软件中学习和挖掘特定的需求和特点.

```
/* Http-2.4.10/modules/metadata/mod_mime_magic.c */
procnew = apr_pcalloc(child_context, sizeof(*procnew));
rc = apr_proc_create(procnew, compr[parm->method].argv[0],
        new_argv, NULL, procattr, child_context);
```

Fig.5 No log for check after memory allocation in Httpd

图 5 Httpd 中内存分配函数后未加日志检查

3.4 错误码的处理

错误码处理也是日志处理中比较棘手的问题,如图 6 所示.错误码往往会从内层函数向外传递很多层,每一层都可能包含 log 的信息.如果统一在某一层加日志,则难以确定最合适的地方,而每一层调用都加又会导致日志过多或冗余.统一到最上层处理可以简化程序,但是越往函数调用栈的上层走,软件的出错现场信息有可能越缺失.

```
/* Linux-3.12/fs/ext4/migrate.c */
bh = sb_bread(inode->i_sb, pblock);
if (!bh)
        return -EIO;
```

Fig.6 Returning erron number to calling function

图 6 函数调用后将错误码返回上一层函数

3.5 test模块需要独立考虑

一般地,大规模软件里都包含一个 test 模块,部分软件的每一个模块中都会包含一个 test 子模块.在 test 模块中,日志往往被大量使用.由于该模块是为了测试功能或性能,完全不用考虑日志输出对性能的影响.我们通过对一些大型软件进行分析,发现很多时候 test 模块中的日志密度很高,例如图 7,在 MySQL-5.6.17 和 SVN-1.8.10 的 test 模块中,日志所占的整个软件日志的比例分别为 43%和 74%,而 test 模块的代码行数(LOC)只占整个软件的 1.23%和 1.45%.因此在实际的日志特征分析中,test 模块需要单独考虑,多数情况下不应该纳入系统日志分析的样本.

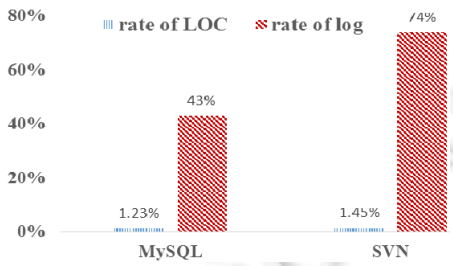
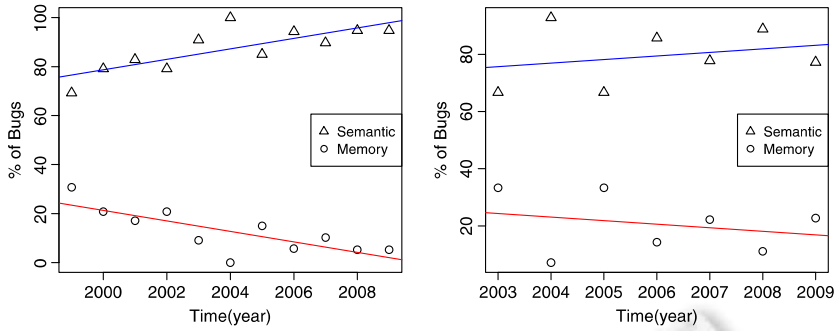


Fig.7 Log density of test module in MySQL and SVN

图 7 MySQL 和 SVN 中 test 模块的日志密度

3.6 语义相关的缺陷比重加大

文献[46]分析指出,随着软件发展的不断成熟,在目前的大规模开源软件中:(1) 内存相关的缺陷在逐渐减少;(2) 语义相关的缺陷在逐渐增加,如图 8 所示.例如,Linux 操作系统缺陷大部分是驱动程序的缺陷.在这种趋势下,单纯地依靠类似 Errlog[2]的方法,通过总结错误模式来添加日志将难以满足需求,因为许多语义错误难以总结成特定的错误模式.因此,我们将基于机器学习、统计等理论,探索如何寻找语义相关的潜在出错点,以加入日志,快速进行错误检测和诊断.



(a) Mozilla (b) Linux 内核

Fig.8 Bug trend in different open source software[46]

图 8 不同开源软件的缺陷趋势[46]

4 建议研究方向和挑战

在大量分析和调研已有大规模开源软件日志规律的基础上,我们思考与列出了一些未来日志研究的问题与方向,主要包括:

(1) 多角度细粒度地分析软件日志的特征.包括日志出现的一般模式、日志的上下文、日志的分布特征等等,形成更加符合程序员的日志编写规范,提高软件代码质量.提出日志评价标准和打分规则,设计自动化的软件日志评价和打分工具,以辅助程序员写出更好质量的软件代码.

(2) 研究程序相关的日志行为学习技术.传统的方法将程序缺陷总结成一些特定的模式,通过在程序中寻找这些模式来研究日志行为.实际上,随着软件中语义相关错误的不断增加,不是所有日志行为都可以归纳成一定的模式.第 3.4 节所提到的 Httpd 就是一个例子,并非内存分配之后都要加日志,而是需要平衡日志开销与性能的关系.

(3) 研究自动化的日志增强和管理技术,包括日志函数的自动识别、日志行为的自动识别、日志的自动插入以及日志信息的自动增强等.从统计分析和机器学习的思路出发,有效结合传统程序分析方法的优势,避免其规模受限等问题,使得设计的方法具有较好的可扩展性.分析时间随代码规模的增加线性增长,同时不需要预先了解程序代码的知识.

(4) 分析已有的软件故障与日志之间的关系,研究多事件日志预警方法.现有的故障往往难以从单个事件的信息进行诊断,可以有针对性地分析特定的故障的特征,如常见的配置故障、国产软件中的驱动程序错等,研究故障与多个事件产生的日志之间的关联关系和相应的日志预警策略.

设计和实现解决上述研究问题并非易事,尤其是自动化地实现一些功能.以下我们列出了部分可能遇到的设计挑战.

(1) 如何在没有预备知识的情况下自动识别日志函数

在大规模软件中,日志函数往往不只一个,它们可能是系统库里自带的打印函数,如 printf,也可能是基于这些打印函数封装而来的函数.识别这些日志函数需要分析日志函数的特征、函数的调用关系等.

(2) 如何判断一个易错点是否已经添加了日志

代码中往往会存在复杂的控制流和数据流,判断一个易错点是否已经存在日志并非易事,如图 9 所示,通过数据流分析,apr_procattr_error_check_set 和 apr_procattr_cmdtype_set 的返回值都流向了 fprintf 的判断条件,可以判断 fprintf 同时是的两个函数的日志.例 2 中,通过控制流分析发现同一函数在不同条件下会有多个日志.另外如前所述,大规模软件经常使用统一的错误码进行出错处理.错误码往往会从内层函数向外传递多层,要确定是否包含日志语句,则需要从最底层的函数沿着调用栈一直向上遍历,代价较大,如果在分析过程中遇到钩子函数则更加困难.

```
/* Httpd-2.4.10/support/rotatelogs.c */
rv = apr_procattr_error_check_set(pattr, 1);
if (rv == APR_SUCCESS)
    rv = apr_procattr_cmdtype_set(pattr, ,,,);
if (rv != APR_SUCCESS) {
    fprintf(stderr, "post_rotate: could not set up …);
    return;
}
```

```
/* Mysql-5.6.17/mysys_ssl/my_default.cc */
if ((error= search_default_file_with_ext(…) < 0)
    goto err;
if (error > 0) {
    fprintf(stderr, "Could not open required file…);
    goto err;
}
err:
    fprintf(stderr,"Fatal error in defaults handling. …");
```

Fig.9 Example of complex logic of the log analysis

图 9 复杂的日志代码分析示例

(3) 如何判断一个潜在的易错点是否应该添加日志

在通过统计分析寻找日志点时,可以这样的思路:当函数调用后出现日志检查超过一定次数,即可推断这后面应该加日志.然而,这个阈值常常难以确定,表 5 给出了 Httpd 中一组统计数据.是交给开发人员决定,还是根据程序动态决定阈值,都是需要考虑的问题.

Table 5 Log number and log rate after some function invoke in Httpd-2.4.10

表 5 Httpd-2.4.10 中一些函数调用发生后出现日志的次数和比例

函数	调用次数	日志次数	日志比率
proxy_ftp_cmmand	18	15	0.83
send_response_header	5	4	0.8
apr_proc_mutex_child_init	5	4	0.8
ap_regkey_open	9	7	0.78
apr_proc_mutex_create	8	6	0.75

(4) 如何确定故障与日志之间的关联关系

大规模软件中经常存在复杂的程序逻辑,故障的产生原因往往和多个事件相关,需要将多个事件日志和失效日志结合才能进行诊断.如何确定故障与日志之间的关联关系并非易事.

5 结束语

软件系统规模的爆炸式增长,为软件的故障诊断提出了挑战.作为故障诊断的重要手段,日志的质量也成为了开发人员和研究人员关注的重点.本文系统地综述了日志研究的进展,分别从日志的特征分析、基于日志的故障诊断、日志的增强方面介绍了如何有效地提高日志质量,以提高大规模软件系统的故障诊断效率.本文通过对已有大规模软件的日志代码展开分析,发现了一些日志相关的特征和规律以及现有工具难以解决的问题.本文还给出了建议的研究方向和挑战.

我们认为,日志领域的研究将进一步与具体的故障类型相结合,综合多点的日志信息进行推理和诊断,如针对广泛使用的分布式系统相关的故障、出错类型较多的操作系统驱动程序故障、难以定位的配置故障等.同时,提高日志的可利用能力,如在开发、调试和测试过程中增加提示,增强日志信息的可视化程度,辅助开发人员形成规范和信息量充分的日志.

致谢 在此,我们向对本文的工作给予支持和建议的同行,特别是清华大学的孙家广院士、顾明教授、贺飞副教授和腾讯公司的专家表示感谢.

References:

[1] Coverity. Coverity Scan: 2012 Open Source Report. 2013. http://www.coverity.com/

[2] Yuan D, Park S, Huang P, Liu Y, Lee MM, Tang X, Zhou Y, Savage S. Be conservative: enhancing failure diagnosis with proactive logging. In: Proc. of the 10th Symp. on Operating Systems Design and Implementation (OSDI). 2012. 293−306.

[3] Yuan D, Park S, Zhou Y. Characterizing logging practices in open-source software. In: Proc. of the 2012 Int'l Conf. on Software Engineering. 2012. 102−112. [doi: 10.1109/ICSE.2012.6227202]

[4] Kavulya SP, Joshi K, Di Giandomenico F, Narasimhan P. Failure Diagnosis of Complex Systems, Resilience Assessment and Evaluation of Computing Systems. Springer-Verlag, 2012. 239−261. [doi: 10.1007/978-3-642-29032-9]

[5] Fu Q, Zhu J, Hu W, Lou JG, Ding R, Lin Q, Zhang D, Xie T. Where do developers log? An empirical study on logging practices in industry. In: Proc. of the 36th Int'l Conf. on Software Engineering. 2014. 24−33. [doi: 10.1145/2591062.2591175]

[6] Jiang W, Hu C, Pasupathy S, Kanevsky A, Li Z, Zhou Y. Understanding Customer Problem Troubleshooting from Storage System Logs. In: Proc. of the 7th USENIX Conf. on File and Storage Technologies (FAST). 2009. 43−56.

[7] Prewett JE. Analyzing cluster log files using logsurfer. In: Proc. of the 4th Annual Conf. on Linux Clusters. 2003.

[8] Hellerstein JL, Ma S, Perng CS. Discovering actionable patterns in event data. IBM Systems Journal, 2002,41(3):475−493. [doi: 10.1147/sj.413.0475]

[9] Ma S, Hellerstein JL. Mining partially periodic event patterns with unknown periods. In: Proc. of the 17th Int'l Conf. on Data Engineering. 2001. 205−214. [doi: 10.1109/ICDE.2001.914829]

[10] Yamanishi K, Maruyama Y. Dynamic syslog mining for network failure monitoring. In: Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery in Data Mining. 2005. 499−508. [doi: 10.1145/1081870.1081927]

[11] Lim C, Singh N, Yajnik S. A log mining approach to failure analysis of enterprise telephony systems. In: Proc. of the IEEE Int'l Conf. on Dependable Systems and Networks with FTCS and DCC (DSN 2008). 2008. 398−403. [doi: 10.1109/DSN.2008.4630109]

[12] Mariani L, Pastore F. Automated identification of failure causes in system logs. In: Proc. of the 19th Int'l Symp. on Software Reliability Engineering (ISSRE 2008). 2008. 117−126. [doi: 10.1109/ISSRE.2008.48]

[13] Liang Y, Zhang Y, Sivasubramaniam A, Sahoo RK, Moreira J, Gupta M. Filtering failure logs for a bluegene/l prototype. In: Proc. of the Int'l Conf. on Dependable Systems and Networks (DSN 2005). 2005. 476−485. [doi: 10.1109/DSN.2005.50]

[14] Xu W. System Problem Detection by Mining Console Logs. 2010.

[15] Rabkin A, Xu W, Wildani A, Fox A, Patterson D, Katz R. A graphical representation for identifier structure in logs. In: Proc. of the Workshop Managing Systems via Log Analysis and Machine Learning Techniques. 2010.

[16] Xu W, Huang L, Fox A, Patterson DA, Jordan MI. Mining console logs for large-scale system problem detection. In Proc. of the 3rd Conf. on Tackling Computer Systems Problems with Machine Learning Techniques. 2008. 4.

[17] Xu W, Huang L, Fox A, Patterson D, Jordan MI. Detecting large-scale system problems by mining console logs. In: Proc. of ACM SIGOPS the 22nd Symp. on Operating Systems Principles. 2009. 117−132. [doi: 10.1145/1629575.1629587]

[18] Xu W, Huang L, Fox A, Patterson D, Jordan M. Online system problem detection by mining patterns of console logs. In: Proc. of the 9th IEEE Int'l Conf. on Data Mining (ICDM 2009). 2009. 588−597. [doi: 10.1109/ICDM.2009.19]

[19] Xu W, Huang L, Fox A, Patterson D, Jordan M. Experience mining Google's production console logs. In: Proc. of the SLAML. 2010.

[20] Tang L, Li T, Perng CS. LogSig: Generating system events from raw textual logs. In: Proc. of the 20th ACM Int'l Conf. on Information and Knowledge Management. 2011. 785−794. [doi: 10.1145/2063576.2063690]

[21] Lou JG, Fu Q, Yang S, Xu Y, Li J. Mining invariants from console logs for system problem detection. In: Proc. of the USENIX Annual Technical Conf. 2010.

[22] Fu X, Ren R, Zhan J, Zhou W, Jia Z, Lu G. LogMaster: Mining event correlations in logs of large-scale cluster systems. In: Proc. of 2012 IEEE the 31st Symp. on Reliable Distributed Systems (SRDS). 2012. 71−80. [doi: 10.1109/SRDS.2012.40]

[23] Ghanbari S, Hashemi AB, Amza C. Stage-Aware anomaly detection through tracking log points. In: Proc. of the 15th Int'l Middleware Conf. 2014. 253−264. [doi: 10.1145/2663165.2663319]

[24] Beschastnikh I, Brun Y, Ernst MD, Krishnamurthy A. Inferring models of concurrent systems from logs of their behavior with CSight. In: Proc. of the 36th Int'l Conf. on Software Engineering. 2014. 468−479. [doi: 10.1145/2568225.2568246]

[25] Fu Q, Lou JG, Wang Y, Li J. Execution anomaly detection in distributed systems through unstructured log analysis. In: Proc. of the 9th IEEE Int'l Conf. on Data Mining (ICDM 2009). 2009. 149−158. [doi: 10.1109/ICDM.2009.60]

[26] Zhao X, Zhang Y, Lion D, Faizan M, Luo Y, Yuan D, Stumm M. lprof: A nonintrusive request flow profiler for distributed systems. In: Proc. of the 11th Symp. on Operating Systems Design and Implementation. 2014.

[27] Tan J, Pan X, Kavulya S, Gandhi R, Narasimhan P. SALSA: Analyzing logs as state machines. In Proc. of the 1st USENIX Conf. on Analysis of System Logs. 2008. 6.

[28] Tan J, Pan X, Kavulya S, Gandhi R, Narasimhan P. Mochi: Visual log-analysis based tools for debugging hadoop. In: Proc. of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud). San Diego, 2009.

[29] Li W, Gorton I. Analyzing Web logs to detect user-visible failures. In: Proc. of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML 2010). Vancouver, 2010. 6−6.

[30] Nagaraj K, Killian C, Neville J. Structured comparative analysis of systems logs to diagnose performance problems. In: Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation. 2012. 26−26.

[31] Fu Q, Lou JG, Lin Q, Ding R, Zhang D, Xie T. Contextual analysis of program logs for understanding system behaviors. In: Proc. of the 10th Working Conf. on Mining Software Repositories. 2013. 397−400. [doi: 10.1109/MSR.2013.6624054]

[32] Glerum K, Kinshumann K, Greenberg S, Aul G, Orgovan V, Nichols G, Grant D, Loihle G, Hunt G. Debugging in the (very) large: 10 years of implementation and experience. In: Proc. of ACM SIGOPS the 22nd Symp. on Operating Systems Principles. 2009. 103−116. [doi: 10.1145/1629575.1629586]

[33] Mozilla quality feedback agent. http://goo.gl/V9zl2

[34] Yuan D, Mai H, Xiong W, Tan L, Zhou Y, Pasupathy S. SherLog: Error diagnosis by connecting clues from run-time logs. In: Proc. of the ACM SIGARCH Computer Architecture News. 2010. 143−154. [doi: 10.1145/1736020.1736038]

[35] Veeraraghavan K, Lee D, Wester B, Ouyang J, Chen PM, Flinn J, Narayanasamy S. DoublePlay: Parallelizing sequential logging and replay. In Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2011. 15−26. [doi: 10.1145/2110356.2110359]

[36] Subhraveti D, Nieh J. Record and transplay: Partial checkpointing for replay debugging across heterogeneous systems. In: Proc. of the ACM SIGMETRICS Joint Int'l Conf. on Measurement and Modeling of Computer Systems. 2011. 109−120. [doi: 10.1145/1993744.1993757]

[37] Altekar G, Stoica I. ODR: Output-deterministic replay for multicore debugging. In: Proc. of ACM SIGOPS the 22nd Symp. on Operating Systems Principles. 2009. 193−206. [doi: 10.1145/1629575.1629594]

[38] Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. ACM SIGOPS Operating Systems Review, 2002,36(SI):211−224. [doi: 10.1145/844128.844148]

[39] Pecchia A, Russo S. Detection of software failures through event logs: An experimental study. In: Proc. of 2012 IEEE the 23rd Int'l Symp. on Software Reliability Engineering (ISSRE). 2012. 31−40. [doi: 10.1109/ISSRE.2012.24]

[40] Silva LM. Comparing error detection techniques for Web applications: An experimental study. In: Proc. of the 7th IEEE Int'l Symp. on Network Computing and Applications (NCA 2008). 2008. 144−151. [doi: 10.1109/NCA.2008.57]

[41] Cinque M, Cotroneo D, Pecchia A. Event logs for the analysis of software failures: A rule-based approach. IEEE Trans. on Software Engineering, 2013,39(6):806−821. [doi: 10.1109/TSE.2012.67]

[42] Cinque M, Cotroneo D, Natella R, Pecchia A. Assessing and improving the effectiveness of logs for the analysis of software faults. In: Proc. of the 2010 IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). 2010. 457−466. [doi: 10.1109/DSN.2010.5544279]

[43] Zhu J, He P, Fu Q, Zhang H, Lyu MR, Zhang D. Learning to log: Helping developers make informed logging decisions. In: Proc. of the 37th Int'l Conf. on Software Engineering. 2015. [doi: 10.1109/ICSE.2015.60]

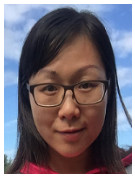
[44] Yuan D, Zheng J, Park S, Zhou Y, Savage S. Improving software diagnosability via log enhancement. In Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2011. 3−14. [doi: 10.1145/2110356.2110360]

[45] Salfner F, Tschirpke S, Malek M. Comprehensive logfiles for autonomic systems. In: Proc. of the 18th Int'l Parallel and Distributed Processing Symp. 2004. [doi: 10.1109/IPDPS.2004.1303243]

[46] Tan L, Liu C, Li Z, Wang X, Zhou Y, Zhai C. Bug characteristics in open source software. Empirical Software Engineering, 2014, 19(6):1665−1705. [doi: 10.1007/s10664-013-9258-8]

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

贾周阳(1994－),男,博士生,主要研究领域为操作系统,软件可靠性.



李姗姗(1980－),女,博士,副研究员,CCF 专业会员,主要研究领域为操作系统,软件可靠性.

刘晓东(1983－),男,博士,助理研究员,CCF 专业会员,主要研究领域为操作系统.

董威(1976－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为软件工程.

周书林(1992－),男,硕士生,主要研究领域为操作系统,软件可靠性.