软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(7):1997−2018 [doi: 10.13328/j.cnki.jos.006045]

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

基于日志数据的分布式软件系统故障诊断综述*

贾 统[1], 李 影[2], 吴中海[2]

[1](北京大学 信息科学技术学院,北京 100871)

[2](北京大学 软件工程国家工程研究中心,北京 100871)

通讯作者: 李影, E-mail: li.ying@pku.edu.cn

摘 要: 基于日志数据的故障诊断是指通过智能化手段分析系统运行时产生的日志数据以自动化地发现系统异常、诊断系统故障.随着智能运维(artificial intelligence for IT operations,简称 AIOps)的快速发展,该技术正成为学术界和工业界的研究热点.首先总结了基于日志数据的分布式软件系统故障诊断研究框架,然后就日志处理与特征提取、基于日志数据的异常检测、基于日志数据的故障预测和基于日志数据分析的故障根因诊断等关键技术对近年来国内外相关工作进行了深入分析,最后以所提出的研究框架为指导总结相关研究工作,并对未来研究可能面临的挑战进行了展望.

关键词: 日志数据;异常检测;故障预测;故障根因诊断

中图法分类号: TP311

中文引用格式: 贾统,李影,吴中海.基于日志数据的分布式软件系统故障诊断综述.软件学报,2020,31(7):1997−2018. http://www.jos.org.cn/1000-9825/6045.htm

英文引用格式: Jia T, Li Y, Wu ZH. Survey of state-of-the-art log-based failure diagnosis. Ruan Jian Xue Bao/Journal of Software, 2020,31(7):1997−2018 (in Chinese). http://www.jos.org.cn/1000-9825/6045.htm

Survey of State-of-the-art Log-based Failure Diagnosis

JIA Tong[1], LI Ying[2], WU Zhong-Hai[2]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](National Engineering Research Center for Software Engineering (Peking University), Beijing 100871, China)

Abstract: Log-based failure diagnosis refers to intelligent analysis of system runtime logs to automatically discover system anomalies and diagnose system failures. Today, this technology is one of the key technologies of artificial intelligence for IT operations (AIOps), which has become a research hotspot in both academia and industry. This study first analyzes the log-based failure diagnosis process, and summarizes the research framework of fault diagnosis based on logs and four key technologies in the field: Log processing and feature extraction technology, anomaly detection technology, failure prediction technology, and fault diagnosis technology. Next, a systematic review is conducted of the achievements of scholars at home and abroad in these four key technical fields in recent years. At last, the different technologies are summarized in this field based on the research framework, and the possible challenges are looked forwarded for future research.

Key words: log analysis; anomaly detection; failure prediction; fault diagnosis

分布式软件系统不仅广泛应用于互联网行业,也应用于政府、金融、航天、军事等关键领域中.这类软件系统服务和应用通常拥有数以亿计的海量用户群,任何微小的软件故障引发的服务中断或者服务质量下降都会带来巨大的损失.国际数据公司(International Data Corporation,简称 IDC)对 1 000 家企业的评估报告[1]表明,一小时的服务宕机平均会造成约 10 万美元的损失.因此,大规模分布式软件系统需要保障 7×24 小时无间断服

* 基金项目: 广东省重点领域研发计划(2020B010164003)
Foundation item: Key-Area Research and Development Program of Guangdong Province of China (2020B010164003)
收稿时间: 2019-10-17; 修改时间: 2020-02-07; 采用时间: 2020-03-09; jos 在线出版时间: 2020-04-21

务,具有高可用性和高可靠性需求.

分布式软件系统规模庞大、组成及运行逻辑复杂,导致系统故障频繁发生[2−4],且故障发生后难以发现、定位和诊断,系统出错后难以分析和调试.首先,大规模分布式软件系统故障种类繁多.从软件系统组成角度来看,系统组件及组件之间的交互均有可能出现故障;从软件系统运行逻辑角度来看,系统和用户请求路径上的任何一个节点均有可能出现故障,同时,大量并发请求又会引起更多故障的发生.其次,由于隐私保护和系统环境配置很难获取等因素,调试过程无法获得用户的输入和输出文件,难以重现与实际生产环境相同的系统配置,从而使基于错误重现和调试的故障诊断难以实施.再次,系统的监控能力有限,难以收集全方位、细粒度的系统运行数据以辅助故障诊断.最后,由于容错机制的存在,系统故障表现并不直观.分布式软件系统容错机制的目标是尽可能减少局部故障带来的影响,对系统屏蔽故障的发生.而恰恰是因为这类容错机制的存在,当系统发生故障时,其反应并不敏感,为系统管理人员进行故障排查增加了困难.因此,如何提升故障诊断效率,快速发现系统故障,定位故障根因成为保障大规模分布式软件系统的高可用性和可靠性的关键.

随着人工智能(artificial intelligence,简称 AI)的发展,智能运维(artificial intelligence for IT operations,简称 AIOps)的概念[5]于 2016 年被 Gartner 首次提出,即通过机器学习(machine learning)等算法分析来自于多种运维工具和设备的大规模数据,自动发现并实时响应系统出现的问题,进而提升信息技术(information technology,简称 IT)运维能力和自动化程度[6].在 AIOps 趋势下,以多源运维数据为驱动,以机器学习等算法为核心的智能化故障发现与根因诊断技术,引起了广泛关注.多源运维数据包括系统运行时数据和历史记录数据(如表单、系统更新文档等).与历史记录数据相比,系统运行时数据能够反映系统的动态特征及系统发生故障时的上下文信息,对未知故障具有更好的探测和表达能力.系统运行时数据主要包括日志数据和监控数据,日志数据是程序开发人员为辅助调试在程序中嵌入的打印输出代码所产生的文本数据,用以记录程序运行时的变量信息、程序执行状态等;监控数据是指系统运行状态下的资源占用情况,如中央处理器(central processing unit,简称 CPU)使用率、内存使用率、网络流量、进程数目以及进程资源使用率等.日志数据和监控数据所处层次不同,前者关注细粒度的应用状态和跨组件的程序执行逻辑,后者则关注系统状态和粗粒度的应用状态,如进程状态、服务状态等.对故障诊断任务而言,日志数据相较监控数据更具优势,表现为:

(1) 能够支持更加细粒度的故障根因诊断.基于系统监控的故障诊断往往只能定位到某一个资源指标出现异常波动,而基于日志数据的故障诊断技术可以定位到特定的出错日志及事件信息.

(2) 能够支持对程序执行逻辑的跟踪,跨组件、跨服务地捕捉程序执行异常和性能异常.每一条请求在系统内部的执行逻辑通常会经过多个组件或程序模块,每一个组件或程序模块均会输出日志,这些日志可以在一定程度上反映该请求的执行轨迹.通过对这些日志进行关联和建模,可以刻画复杂的请求执行路径,进而诊断故障组件及故障程序执行位置.

(3) 能够支持定位异常请求实例,捕捉故障细节.基于监控数据的故障诊断偏重于统计意义上的故障症状,而基于日志数据的故障诊断则可以跟踪每一个请求实例,识别异常,并刻画异常请求执行路径上的相关细节.

因此,基于日志数据的故障诊断对于提升分布式软件系统故障诊断效率,进而提升系统可靠性和可用性十分重要.

本文第 1 节对基于日志数据的大规模分布式系统故障诊断方法的研究框架进行总结,并总结归纳出其中的 4 个关键技术(日志处理与特征提取技术、基于日志数据的异常检测技术、基于日志数据的故障预测技术和基于日志数据的故障根因诊断技术).第 2 节对已有的日志处理与特征提取技术进行总结.第 3 节对基于日志数据的异常检测技术进行总结.第 4 节对基于日志数据的故障预测技术进行总结.第 5 节对基于日志数据的故障根因诊断技术进行分析.最后在第 6 节以本文提出的研究框架为指导总结分析相关研究工作,并对未来值得关注的研究方向进行初步探讨.

1 概 述

1.1 分布式软件系统日志

分布式软件系统日志通常指的是分布式软件系统各组件运行时输出的日志,例如 OpenStack 每个组件(如 Nova-compute,Neutron 等)、Hadoop 各个组件(如 Resource manager,Node manager)输出的日志.这些日志不包括

操作系统日志、网络日志、硬件设备日志以及上层应用日志.不同的分布式系统由于功能特性的不同导致日志序列和日志内容的形成,其最普遍和关键的日志特征是以请求(request)为核心(如 Openstack 的一次虚拟机启动请求,hadoop 的一次 job 执行等),本文所综述的故障诊断方法主要针对这种以请求为核心的日志数据.

典型的分布式软件系统日志包括两类:事务型日志(transactional log)和操作型日志(operational log).事务型日志表征请求/事务执行逻辑,操作型日志表征诸如心跳、消息分派等独立事件.以流行的开源 PaaS 云计算平台软件 Cloud Foundry 为例,其中负责管理应用程序生命周期的云控制器的日志记录查找容器、解析参数以及报告请求执行状态等请求执行步骤,日志之间有明显的因果关联关系,属于典型的事务型日志;而监控组件负责协调应用程序以及跟踪其状态的 HM 9000 系统监控组件,其每条日志代表一个独立事件,用于接收和保存来自其他组件的心跳,属于操作性日志,日志之间相互独立.值得一提的是,一个组件可能既输出事务型日志又输出操作型日志,一个典型的例子是某组件既需要执行事务流程,同时还需要定期报告心跳信息.针对事务型日志和操作型日志的不同特点,分布式软件系统故障诊断技术通常会采取不同的数据分析和建模方法.

1.2 基于日志的软件系统故障诊断过程

典型的软件系统故障过程包括 3 个阶段:故障根因出现、系统行为异常和系统运行故障.故障根因导致系统行为出现异常,进而引发系统故障.如图 1 所示,故障根因、异常和故障在软件系统故障过程中依次出现,具有时序和因果关系.故障诊断的目的是在系统行为异常阶段,检测系统表征的异常信息,预测未来可能发生的故障,诊断引发故障的根本原因.因此,软件系统故障诊断包括 3 个子任务:异常检测、故障根因诊断和故障预测.

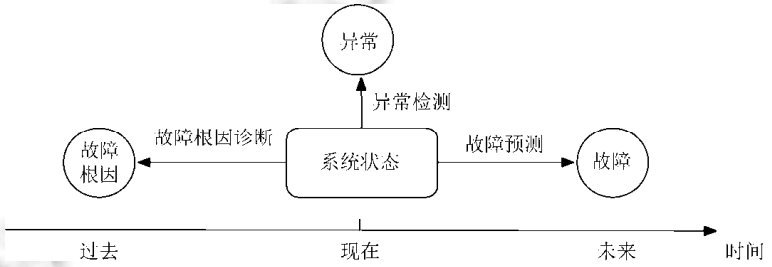

Fig.1 Relationships of failure, anomaly and fault/root cause

图 1 故障、异常、故障根因关系示意图

基于日志数据的软件系统故障诊断通过分析、挖掘系统运行日志,辅助运维人员理解系统行为,诊断软件系统故障.该技术的研究框架包括 4 个关键技术(如图 2 所示),即日志处理与特征提取、基于日志数据的异常检测、基于日志数据的故障预测和基于日志数据的故障根因诊断.

1. 日志处理与特征提取(见第 2 节)

日志作为程序开发人员打印的文本信息,具有半结构化特征;分布式系统的日志量十分庞大,且由于各个组件所使用的程序设计语言不同,日志打印的风格和质量都有很大差异,因此,为降低日志数据的复杂性,消除大量噪音并有效提取特征,日志处理与特征提取技术利用机器学习、模式识别、统计分析、自然语言处理等方法,分析和挖掘日志数据特征信息,为异常检测、故障预测和故障根因诊断提供数据基础.

2. 基于日志数据的异常检测(见第 3 节)

基于日志数据的异常检测即在系统日志数据中发现不符合预估行为的异常模式.其输出通常是日志片段是否属于异常的标签或日志片段包含系统异常信息的概率,评测指标通常包括精确率(precision)、召回率(recall)和综合评价指标 F 值(F-measure).精确率用于评测发现异常模式的正确性,召回率用于评测发现异常模式的全面性,F 值为精确率和召回率的加权调和平均,用于评测异常检测的整体性能.

3. 基于日志数据的故障预测(见第 4 节)

基于日志数据的故障预测即通过当前日志数据预测在不远的将来是否会发生系统故障.其输出通常是系统未来是否会出现故障的标签或出现故障的概率,评测指标通常包括精确率、召回率、综合评价指标 F 值和前

置时间(lead-time).精确率用于评测预测系统故障的正确性,召回率用于评测预测系统故障的全面性,*F* 值评测故障预测的整体效果,前置时间即预测系统故障至系统出现故障的时间间隔.前置时间反映了能够提前预知故障的能力,理论上讲,前置时间越长代表提供给运维人员解决或降低故障风险的时间越长,更有助于解决故障问题,降低故障产生的影响.

4. 基于日志数据的故障根因诊断(见第 5 节)

基于日志数据的故障根因诊断即通过日志数据诊断与系统故障相关的根因信息,如故障类型、故障位置、故障请求、故障代码片段等.故障根因诊断技术的输出即为多种类型的故障根因信息,评测指标与基于日志数据的异常检测技术相似,精确率用于评测输出故障信息的正确性,召回率用于评测输出故障信息的全面性,*F* 值用以评测故障诊断的整体性能.

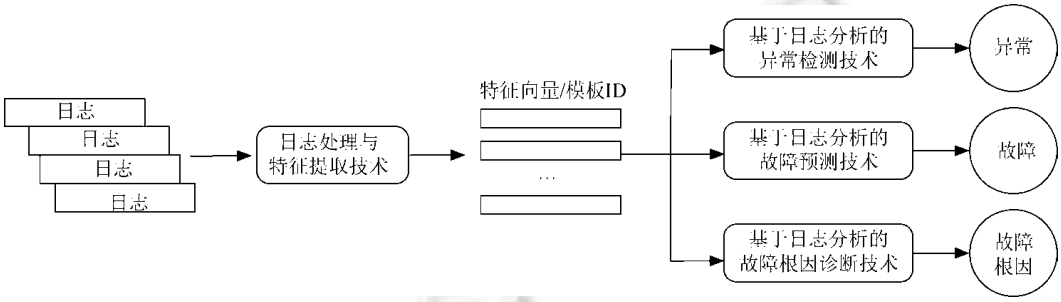

Fig.2 Research framework of log-based failure diagnosis for distributed systems

图 2 基于日志数据的分布式系统故障诊断研究框架

2 日志处理与特征提取

日志文本内容通常分为常量和变量,其中,常量描述当前程序的行为或功能,变量则反映程序运行过程中的动态上下文信息.例如,ip 地址、参数值、uuid 等.以 Hadoop NodeManager 组件的日志片段(如图 3 所示)为例,斜体部分代表变量,其余部分代表常量.同时,绝大多数日志均包含时间戳、日志等级等内容,时间戳记录了该日志打印的时间,日志等级代表日志所表征的事件的严重程度,通常包括 ERROR、WARN、INFO、DEBUG 等.

日志数据的复杂性为日志处理与特征提取带来了极大的困难与挑战.首先,日志数量庞大.例如,IBM 的一个公有云子平台每天产生超过 8 亿条,共计约 720GB 的日志数据.如何解析海量日志,并从中挖掘有价值的信息成为一个难题.其次,日志文本异构,不同组件的日志具有强异构性.分布式软件系统各个组件的功能逻辑不同、开发团队不同、所使用的程序设计语言不同、开发时所考虑的关键日志打印信息亦不同,导致日志的文本、结构、风格和质量大相径庭.如何支持异构日志的统一处理成为一个关键性问题.最后,日志文本具有半结构化特征,日志中自然语言和机器语言交织.日志中包含了时间戳、日志等级等结构化信息,同时也包含了由自然语言文本组成的非结构化信息.日志内容通常由机器语言和自然语言糅合交织而成,其中,机器语言通常包括一些标识信息和数值型指标信息,自然语言则主要包括对日志时间的描述文本.多层次、多维度、多类型的文本交织使得日志内容变得极为复杂,给日志的解析处理带来极大的困难.

为解决上述问题,研究人员提出了日志模板挖掘技术和日志特征提取技术,用以从降低日志文本的异构复杂性、从海量日志中提取有价值的信息.日志模板挖掘关注于日志中的常量部分,日志特征提取则关注于日志中的变量部分或其他特征.更进一步地,本文将日志模板挖掘技术划分为基于静态代码分析、基于频繁项集挖掘和基于聚类的日志模板挖掘技术,将日志特征提取技术划分为基于自然语言处理的日志特征提取技术、基于规则的结构化日志信息提取技术和基于统计模型的日志特征提取技术(如图 4 所示).文献分布表明,基于聚类的日志模板挖掘技术的研究和应用最多,而基于规则的结构化日志信息提取则是使用最广的日志特征提取技术.

(1) 2017-01-20 15:44:59,661 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.ContainerManagerImpl: Start request for container_1484893655240_0014_01_000003by user user1

(2) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.ContainerManagerImpl: Creating a new application reference for app application_1484893655240_0014

(3) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.NMAuditLogger: USER=user1
IP=192.168.5.65 OPERATION=Start Container Request
TARGET=ContainerManageImpl RESULT=SUCCESS
APPID=application_1484893655240_0014 CONTAINERID=container_1484893655240_0014_01_000003

(4) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.application.Application: Application application_ 1484893655240_0014 transitioned from NEW to INITING

(5) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.application.Application: Adding container_1484893655240_0014_01_000003to application application_1484893655240_0014

(6) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.application.Application: Application application_1484893655240_0014transitioned from INITING to RUNNING

(7) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.container.Container: Container container_1484893655240_0014_01_000003transitioned from NEW to LOCALIZING

(8) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.AuxServices: Got event CONTAINER_INITfor appId application_1484893655240_0014

(9) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.AuxServices: Got event APPLICATION_INITfor appId application_1484893655240_0014

(10) 2017-01-20 15:44:59,662 INFOorg.apache.hadoop.yarn.server.nodemanager.containermanager.AuxServices: Got APPLICATION_INIT for service mapreduce_shuffle

(11) 2017-01-20 15:44:59,663 INFOorg.apache.hadoop.mapred.ShuffleHandler: Added token for job_1484893655240_0014

Fig.3 An example of Hadoop logs

图 3 开源软件 Hadoop 日志示例

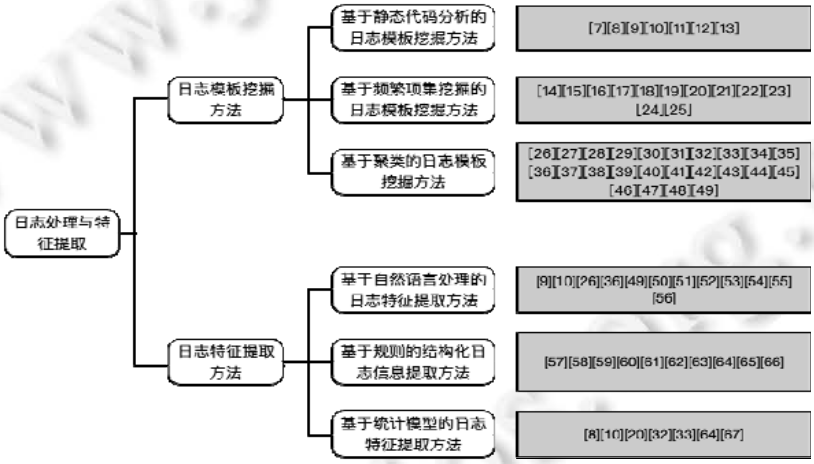

Fig.4 Related work of log parsing approaches

图 4 日志数据处理与特征提取方法分类及相关文献

2.1 日志模板挖掘

日志模板指的是以日志中的常量为标识的日志类型抽象.本质上,日志模板对应程序中的日志打印语句,相同日志打印语句输出的日志即属于同一个模板.如图 5 所示,日志中的常量通常在日志打印的代码语句中写明,变量则由系统运行时的变量取值决定.日志模板保留日志中的常量部分,将变量部分以占位符标识,从而将相同日志打印代码输出的日志映射到一个模板上.日志模板挖掘的意义在于:(1) 保留日志关键信息,解决日志的异构性问题.一方面,由于日志中的常量部分通常表征日志事件信息,而日志模板保留常量作为标识,因此日志模板可以保留关键日志事件信息;另一方面,日志模板对应程序中的日志打印语句,可以保留日志打印语句的位置

信息,从而记录程序执行的关键节点.与此同时,异构日志可以通过日志模板挖掘统一到一个日志模板集合中,在一定程度上解决了日志异构的问题.(2) 压缩日志,简化大规模日志处理.由于多条日志可以匹配相同的日志模板,因此日志模板可成千上万倍地压缩需要处理的日志数目,极大地减少了日志数量,降低了日志处理难度.

从上述分析可知,理论上讲,日志模板和日志打印语句应为一一对应关系.因此,精确、合理的日志模板应能表征程序中的每一条日志打印语句.然而,在现实中,日志模板的关键用途在于保留关键日志信息,压缩异构日志,从而更高效地处理和理解日志内容.在这种情况下,同一日志打印代码语句对应的日志模板或者精确以日志常量为标识的日志模板却不一定是最合适的日志模板.例如,分布式系统中的消息分发组件的作用是与其他若干组件通信并转发请求,这类组件的日志通常包含很多变量以记录转发状态、目的地、资源请求等信息.对于这些日志而言,仅仅提取其中的常量作为模板将会生成极少的无用模板而丢失大量的信息.因此,本文认为,有价值的日志模板应具备如下特点:(1) 模板数目适中,太多的模板带来较低的压缩比,进而增加日志后续工作的复杂度,太少的日志又会丢失有价值的信息;(2) 表征重要信息的变量应作为模板内容被提取出来.下面详细介绍日志模板挖掘的 3 种主要方法及相关工作.

11/12/2013::8:59 am: - Error in resp method: JobRequest Missing parameter serverId
11/12/2013::9:01 am: - Parsing resp method: GetLparNonBlockingDataBeans for cmd.
#日志示例：绿色部分代表变量，黑色部分代表常量
(1) print (time. time () +'Error in resp method' +method_name+'missing parameter' +params [i])
(2) print (time. time () +'Parsing resp method' +method_name+'for cmd')
#日志打印代码示例
(1) Error in resp method: (*) Missing parameter (*)
(2) Parsing resp method (*) for cmd
#日志模板示例

Fig.5 An example of logging statements and log templates
图 5 日志打印代码与日志模板示例

1. 基于静态代码分析的日志模板挖掘

该方法即利用静态代码分析等技术手段,从代码中搜索日志打印语句并从这些语句中挖掘模板.从代码中挖掘日志模板依赖于不同的程序设计语言.对于 C 语言或者类 C 语言,日志模板可以直接通过日志打印语句进行推理和提取.对于 Java 等面向对象的编程语言,从代码中提取日志模板就变得十分复杂,主要面临 3 个困难和挑战:(1) 必须预先获取日志打印类名(如 log4j 中的 Log 类);(2) 面向对象程序设计语言中的字符串输出语句中的变量对象实际上实现了 toString()方法,因此需要深入到变量对象的类内部代码进行搜索;(3) 由于类之间继承关系的存在,实际的调用 toString()函数的代码可能位于子类中.

针对这 3 个挑战,Xu 等人[8]首先将源代码转化为抽象语法树(abstract syntax tree,简称 AST),然后使用抽象语法树搜索日志打印类中所有的方法调用关系、对象名、类名等,对每一个方法调用都生成一个日志模板片段.接着,遍历所有函数中的 toString()方法调用,从其中的字符串连接语句中抽取所有的变量,将这些变量以占位符形式嵌入到日志模板片段中,如果日志打印语句中的对象相对应的类中没有 toString()函数,则递归地搜索其子类,直到发现该函数为止.嵌入变量占位符的日志模板片段即为最终的完整日志模板.

相关文献[9−11]直接使用了上述方法,Yuan 等人[12]同样要求程序开发人员给出日志打印函数以及在该函数中哪一个参数是以日志格式方式来控制参数.Zhao 等人[7]的方法则无需预先获取日志打印类名并修改了 AST 的遍历步骤.该方法在生成 AST 之后,遍历所有的方法调用,找到所有包含日志等级的特殊字符串,如 FATAL、ERROR、WARN、INFO、DEBUG 或 TRACE 的方法调用,这些方法调用即为所有的日志打印语句.

2. 基于频繁项集挖掘的日志模板挖掘

假设在不同日志中频繁出现的相同词语极有可能是常量,基于频繁项集挖掘的日志模板挖掘方法将每一条日志看作一条交易记录,将其中每一个词(token)看作一个商品,通过挖掘频繁词集合,确定日志中的常量,进而

通过组合这些词最终生成日志模板.Vaarandi[14]将基于频繁项集挖掘的方法划分为两个步骤:频繁词挖掘和频繁词集合挖掘,首先通过频繁项集挖掘算法找到出现频率大于支持度阈值的频繁词,然后,对于每一条日志,找到其中包含的频繁词,并记录其序列,再根据这些词序列的出现频率重新对这些频繁词序列进行挖掘,最终生成日志模板.

Vaarandi[18]对其工作进行了进一步优化,首先经过对若干软件系统日志中的词语出现频率进行统计分析发现绝大多数词语出现频次极少,少数词语出现频繁;频繁词语之间具有许多强关联.基于此,Vaarandi 提出日志模板挖掘算法和挖掘工具 SLCT[22,23,25].与 Apriori 算法相比,该算法充分利用日志的特征,省略了循环挖掘频繁项集的过程,因此在执行效率上有很大程度上的提升.为了解决长变量问题,Vaarandi[19]引入模糊匹配的思想对上述方法进行了优化.所谓长变量即日志中的变量包含多个词语,在模板挖掘时会被认为是多个变量,因而降低了模板挖掘的精确度以及算法效率.该方法在建立候选模板集合的步骤中验证某一个日志是否属于已有的模板时,会进行一次模糊的匹配,并根据匹配结果修改当前模板.与 Vaarandi 的工作[18,19]相比,Reidemeister 等人[17]对算法进行了更进一步的优化.首先,挖掘日志中的频繁区域(frequent region),通过设置频繁阈值挖掘日志中频繁出现的词或词序列作为频繁区域,这些频繁区域即候补日志模板;然后,对频繁区域聚类进一步过滤其中的变量.最后,编码聚类结果生成日志模板.

3. 基于聚类的日志模板挖掘

基于聚类的日志模板挖掘的特点是忽略日志中词语的出现频率,利用日志文本的相似度对日志聚类,从而挖掘日志模板.这类算法最早在 2008 年 Jiang 等人的研究工作[41,42]中被提出,之后,微软研究院的研究工作[27−31]将其发扬光大,成为了一种经典的模板挖掘算法.Jiang 等人[41]首先提出了一种基于少量先验知识的模板挖掘算法,通过日志变量替换、日志词语的切分与分类、日志模板提取和日志模板合并实现模板挖掘.Fu 等人[30]提出另一种基于聚类的日志模板挖掘方法.然后,对日志进行聚类,利用加权编辑距离计算日志之间的相似度,最后,切分聚类得到的日志组生成日志模板.

与 Fu 等人的方法相比,Chen 等人[32]在对日志进行聚类时,对编辑距离的计算方法作了进一步优化.通过对日志文本的观察,他们指出日志文本具有两个特点:位置靠前的日志文本往往比位置靠后的日志文本更重要;两条包含相反含义词语的日志,即使在文本上非常相似,理论上仍应属于不同的模板.基于这两个特点,Chen 等人提出了一种基于修正加权编辑距离的相似度(modified simple weighted Levenshtein ratio,简称 MSWLR)计算方法.在聚类过程中,他们使用了层次聚类的算法.Du 等人[33]指出,多数情况下属于同一个模板的日志有相同的日志等级(log level),因此可以将日志等级进行向量化处理,在计算日志之间的距离时,既考虑日志之间的编辑距离,又考虑日志等级之间的差异.

为了克服聚类算法迭代次数较多、执行速度较慢的缺点,近年来,基于聚类的在线日志模板挖掘方法[37,39,44−48]逐渐兴起.这类方法采用流式处理,对于当前每一条日志计算其与现有日志之间的距离,如欧式距离、编辑距离、汉明距离、曼哈顿距离等,将其划分到与之距离最近的日志类别中,最后将日志类别转化为日志模板.

4. 日志模板挖掘技术分析

表 1 从不同维度对现有日志模板挖掘技术进行了对比.从输入角度看,3 种日志模板挖掘技术都需要一定数量的日志集合,其中,基于静态代码分析的日志模板挖掘技术还需要系统源代码作为输入;从执行效率角度来看,静态代码分析方法扫描代码文件,从中获取日志打印语句进而生成日志模板,因此执行效率较高,频繁项集挖掘算法通过阈值调控,搜索出现频率超过阈值的日志文本 token,因此对迭代要求低,执行效率较高,而基于聚类的日志挖掘技术需要不断计算日志之间的文本编辑距离,并进行多次迭代以对日志进行分组,因此执行效率低.总体而言,基于静态代码分析的日志模板挖掘的优点在于日志模板挖掘速度快,且由于模板直接从源代码中获取,结果十分精确,缺点在于依赖系统源代码,对于不同的程序设计语言,不同的日志打印类实现都需要重新定义输入;基于频繁项集挖掘的日志模板挖掘的优点在于日志模板挖掘速度快,无需其他先验知识,缺点在于受用户指定置信度和支持度阈值影响较大,且由于其仅关注日志频率,因此无法从出现频率较低的日志中挖掘模

板,可能导致“冷门”日志无法匹配任何模板;基于聚类的日志模板挖掘的优点在于无需其他先验知识且无监督的聚类算法无需或仅需少量调参工作,缺点在于聚类算法迭代次数较多,执行速度较慢,且聚类算法仅仅根据日志的文本距离将日志划分为若干类型,还需要进一步识别日志模板.

Table 1 Analysis of log template mining approaches

表 1 日志模板挖掘方法分析

日志模板挖掘	输入	执行效率	优点	缺点
基于静态代码分析的日志模板挖掘技术	系统源代码/日志	高	1.日志模板挖掘速度快; 2.模板挖掘结果精确	需要系统源代码,对不同的程序设计语言,不同的 logger 都需要重新定义输入
基于频繁项集挖掘的日志模板挖掘技术	日志	高	1.日志模板挖掘速度快; 2.无需其他先验知识	1.受用户指定置信度和支持度阈值影响较大 2.无法从出现频率较低的日志中挖掘模板,可能导致“冷门”日志无法匹配任何模板
基于聚类的日志模板挖掘技术	日志	低	1.无需其他先验知识; 2.无需用户指定阈值,减少了大量调参工作	1.聚类算法迭代次数较多,执行速度较慢 2.聚类算法仅仅根据日志文本生成了若干类型的日志,还需进一步识别日志模板

2.2 日志特征提取

日志特征提取方法可分为基于自然语言处理、基于规则和基于统计模型的日志特征提取方法.基于自然语言处理的方法将每一条日志看作一段文本,利用自然语言处理领域的方法处理日志;基于规则的方法假设日志结构已知,通过关键词提取或者指定过滤规则,对日志的不同域进行提取;基于统计模型的方法通常基于日志模板挖掘技术,通过划分时间窗口,统计每一个时间窗口内日志模板的出现频率,从而将每一个时间窗口内的日志转化成日志模板频率向量.下面详细介绍这 3 种方法及其相关工作.

1. 基于自然语言处理的日志特征提取

基于自然语言处理的日志特征提取通常包括 3 种基本模型:*N*-gram 模型[53,54]、Word Count 模型和 TF-IDF (term frequency-inverse document frequency)模型.*N*-gram 指的是连续的 *n* 个字符或词组成的序列,*N*-gram 模型首先对每一条日志提取其中包含的所有 *N*-gram,然后统计每一个 *N*-gram 的出现频率,最终将这些频率组合成频率向量.Word Count 模型[49−51,69]首先将日志文本分词,然后统计日志文本中每一个位置上不同词的出现频率,最终生成词频向量.TF-IDF 模型[7,9,26]将每一个日志序列看作一个文本,将序列中的日志看作一个词,通过计算每一个日志的 TF-IDF 值,将每一个日志序列转化成 TF-IDF 向量.

2. 基于规则的结构化日志信息提取

基于规则的结构化日志信息提取假设或已知日志结构,通过关键词提取或者指定过滤规则,对日志的不同域(field)进行提取.Chuah 等人[58]主要关注关系型日志(rationalized log),这类日志具有特定的格式(如 POSIX 格式)和逻辑结构,因此容易通过正则表达式提取信息.Yen 等人[59]则针对网页代理日志,这类日志同样具有相对一致的格式,通过简单的正则匹配就可以将日志中的变量信息提取出来.另一种方法通过对系统进行修改或侵入,修改原始日志打印内容,从而对日志信息进行定制.Chow 等人[57]提出了一种基于系统侵入的日志特征提取方法,实现了一个 Ubertrace 的工具,用于整合系统不同组件的日志,并生成每一个请求的跨组件日志序列.同时,Ubertrace 还负责对这些请求取样分析,在尽可能地减少对系统性能损失的基础上保证请求日志序列的完整性.这样,Ubertrace 做到了对请求日志序列的自动获取和跟踪以及对日志事件内容的自动控制.

3. 基于统计模型的日志特征提取

基于统计模型的日志特征提取通常基于日志模板挖掘方法,统计每一个日志模板在时间窗口内的出现频率,从而将日志转化成频率向量(message count vector)[8].该方法忽略了日志的文本信息特征,将日志频率的统计信息作为表征系统状态的监控指标,用于故障诊断.Chen 等人[32]使用层次聚类的方法,将日志聚类成日志模板,然后计算每一个时间窗口内每一个日志模板的频率,最后使用绝对中位差(median absolute deviation,简称 MAD)作为频率变化的指标,找到频率突变的时间窗口作为故障时间.

4. 日志特征提取技术分析

表 2 对比分析各日志特征提取方法.从输入角度来看,3 种日志特征提取方法的输入均为一定数量的日志集

合,基于统计的日志特征提取方法还需要已挖掘的日志模板作为输入,用以计算日志模板频率作为统计特征;从输出角度来看,基于自然语言处理的日志特征提取技术和基于统计的日志特征提取技术的输出为数字型的特征向量,基于规则的日志特征提取技术输出为结构化的日志信息,主要是日志中各个域(包括时间戳、日志等级、特殊变量、消息文本等)的内容.总体而言,基于自然语言处理的日志特征提取技术的优点在于充分提取独立日志的文本特征,能够捕获日志中自然语言部分的语义信息,计算相对较为方便.缺点在于忽略了日志序列中的上下文因果和关联关系,以字符为基本特征提取单元的方法可能会导致特征向量维度巨大,导致故障诊断模型效率下降;基于规则的日志特征提取技术利用日志中的特殊间隔符将日志切分为各个域,其优点在于能够将日志转化为结构化的信息,计算过程简单,输出整洁精确,缺点在于要求日志数据为固定格式,且需要日志结构的先验知识以定义日志切分的正则表达式;基于统计的日志特征提取技术的优点在于充分提取日志在时间轴上的频率分布特征,计算速度快,适用于不同模式不同表征的日志数据(包括事务型日志和操作型日志),缺点在于忽略了日志的文本特征以及上下文关联特征,生成的特征向量所包含的信息有限,只能作为系统故障发现的一个简单指标.

Table 2 Analysis of log feature extraction approaches

表 2 日志特征提取方法分析

日志特征提取	输入	输出	优点	缺点
基于自然语言处理的日志特征提取	日志	特征向量	充分利用日志的文本特征,能够捕获日志中自然语言部分的语义信息,生成数字化特征向量	忽略日志序列的上下文关联关系,特征向量维度可能会很大,影响后续计算复杂度
基于规则的日志结构化信息提取	日志	日志中每个域中的信息	将日志转化为结构化的文本信息,计算过程简单,输出整洁精确	日志数据需要属于同一个格式,且需要日志结构的先验知识以定义日志切分正则表达式;适用范围较窄,只能针对包含特定间隔符的日志,难以广泛应用
基于统计模型的日志特征提取	日志/日志模板	特征向量	充分提取日志在时间轴上的频率分布特征,将日志序列转化为数字化的特征向量,计算效率高,通用性强	忽略日志的文本特征以及前后关联关系,特征向量包含信息有限,只能作为系统故障发现的一个简单指标,仅能捕获日志模板分布的变化情况,难以适用于构建复杂情况下异常的故障诊断模型

3 基于日志数据的异常检测

基于日志数据异常检测的目的是在线地对系统进行日志收集与分析,在系统日志中找到不符合预估行为的模式.其主要思路是从系统正常运行的日志数据中学习“健康”状态模型,故障探测过程通过寻找是否有与“健康”状态模型冲突的在线数据.由于日志数据与分布式软件系统的复杂性,基于日志数据的异常检测面临两个关键性挑战.(1) 日志数据中的异常表征复杂多样,难以有效捕获.日志数据中的异常可能包含分布异常、序列异常、变量异常等,如何构造有效的异常检测模型以捕获复杂的异常成为一个技术上的挑战.(2) 从海量日志数据中精准发现少量异常是一个难点.日志数量庞大,异常表征往往会隐藏在海量日志中,如何构造高效的异常检测模型快速过滤海量日志并发现其中的异常成为一个技术上的挑战.

现有技术可分为 3 类:基于图模型的异常检测、基于概率分析的异常检测和基于机器学习的异常检测.基于图模型的异常检测对日志的序列关系、关联关系以及日志文本内容进行建模;基于概率统计的异常检测采用关联分析、对比等,计算日志与异常的关联概率;基于机器学习的异常检测采用聚类算法找到离群值(outlier)或分类算法学习故障时的日志模式,判断在线日志数据是否符合这些日志模式.该技术的方法分类及相关文献如下文的图 6 所示.

3.1 基于图模型的异常检测

基于图模型的异常检测采用有向图(DAG)模型对表征系统执行逻辑的日志序列进行建模.该方法要求日志之间存在因果关联关系,具备表征请求/事务执行逻辑的能力,因此适用于事务型日志.文献[15]是以日志模板为节点建模的典型工作,通过挖掘日志模板,根据系统正常运行时产生的日志集合建立控制流图(control flow graph)模型.在控制流图模型中节点代表日志模板,边代表日志模板之间的转移关系;在异常检测阶段,对比系统运行时的日志和已建立的控制流图模型,两者之间的差异(可能是转移路径不同,节点不同等)即为异常,包

括序列异常和分布异常.序列异常指的是图模型中子节点丢失或错误,分布异常指的是图模型中父节点向不同子节点的转移概率变化.

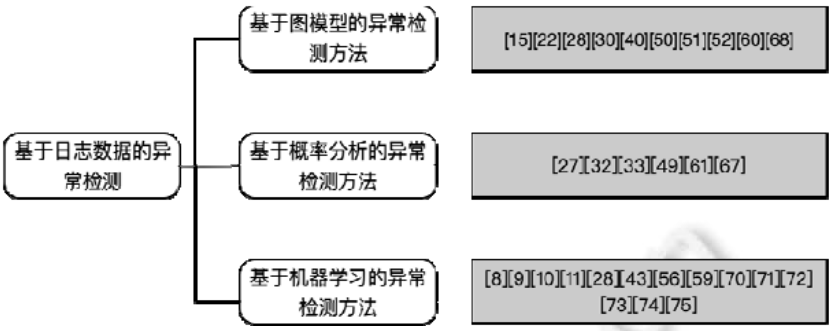

Fig.6 Related work of anomaly detection approaches

图 6 基于日志数据的异常检测方法分类及相关文献

文献[28]以日志模板为边,通过分析表征程序执行逻辑的日志序列,构建有向图模型.通过 Job ID 和 Transaction ID 将属于同一个请求或者同一个事务的日志序列提取出来,然后利用 ID 信息建立有向图模型.采用 FCA(formal concept analysis)方法对这些日志序列进行建模,并通过决策树模型找到影响系统执行分支的因素(日志或日志集合).文献[10]同样使用有向图模型对日志序列进行建模.首先挖掘日志模板,然后通过 Process ID 和 Thread ID 将同一个任务的日志序列切分出来,由这些日志序列建立有向状态机 FSA(finite state automaton)模型.文献[60]提出已有的基于日志数据的异常检测工作通常对日志内容有一定的假设,假设日志中包含 Request ID、Transaction ID、Process ID、Thread ID 等,从而有向图模型对不同请求的交错日志进行建模,使用日志中的多种共享变量关联同一个请求输出的日志,并以日志模板为边构建有向图模型,实现异常检测的目标.

除了有向图模型外,一部分研究工作尝试采用 Page-Rank 模型、信息熵模型等对日志序列进行建模.文献[50]通过香农信息熵对日志中的词语或变量在不同分布式组件或节点上的分布进行建模.首先使用基于 Word Count 模型的日志特征提取技术生成日志中的词语或变量在不同组件或节点上的分布矩阵,然后使用香农信息熵计算矩阵中每一个元素的权重,进而对每一个组件或节点上的日志分布建模.当一段时间内不同组件的日志分布出现差异时,则判断系统出现异常.文献[68]使用 Page-Rank 模型对日志以及日志中的标识符(identifier)进行建模.日志中的标识符被定义为包含 3 种类型的词语或词组:(1) 在所有日志中出现频率较高的词语;(2) 从代码中提取的类名;(3) 由程序开发人员指定的关键标识词语.首先对系统正常运行时产生的日志及日志中的标识符建立图模型 G_1,然后计算 G_1 中每一个点的 page-rank 值.在线异常检测阶段按照同样的方法利用在线日志数据生成图模型 G_2,计算 G_2 中每一条日志的 page-rank 值并与 G_1 中每一条日志的 Page-Rank 值的差,并以此作为判断该日志是否异常的标志.

3.2 基于概率分析的异常检测

基于概率分析的异常检测技术从日志中挖掘频率向量、规则等,通过计算在线日志频率分布与已挖掘的频率向量、规则等的匹配程度或相似概率,进而计算在线日志的异常概率.该方法主要通过抽取不同时间窗口内日志的频率或序列特征,对日志序列的上下文关联没有要求,因此适用于事务型日志和操作型日志.

文献[69]是该方法的典型工作.首先将日志转化为信号(signal),然后记录每一个时间窗口内的信号分布,最后使用 K-L 散度计算当前时间窗口内的信号分布和以往时间窗口内的信号分布差异,借此判断时间窗口的异常程度.文献[33]则将每一个日志模板在时间轴上的频率分布看作一个向量,称为频率序列(frequency sequence).定义一个日志模板的行为子序列(behavior subsequence)为定长滑动时间窗口内的频率序列;使用欧拉距离计算任意两个行为子序列之间的相似度,然后基于该距离计算方法使用层次聚类的方法将所有行为子序列聚类,即如果行为子序列的分布相似,则将其划分为一个行为模式.在异常检测阶段,生成在线日志数据的

行为子序列集合,计算每一个行为子序列的异常评分(anomaly score).

3.3 基于机器学习的异常检测

基于机器学习的异常检测从日志特征向量中学习系统正常运行的关键特征,对比这些关键特征检测异常,或对日志集或日志特征向量集进行无监督聚类检测出的离群点即为异常点.该方法通常使用单条日志或日志集合的特征向量构建机器学习模型,对日志序列的上下文关联没有要求,因此适用于事务型日志和操作型日志.

文献[8−10]根据日志中的公共变量信息(如 Tid,Pid 等)将日志分组,对每一个组生成一个日志模板频率向量(message count vector).然后用 PCA 算法计算日志模板频率向量中每个特征维度的影响,并将其中关键特征维度组合成低维度特征空间.最后,计算大多数日志模板频率向量在这些关键特征维度上的取值范围,例如 95%的数据在这些关键特征维度上的取值分布范围,记录这些取值生成正常子特征空间(normal subspace).在异常检测阶段,将在线日志的日志模板频率向量通过矢量计算映射到关键特征维度上,并计算该向量与正常子特征空间之间的矢量距离,如果距离超过某阈值,则判定系统出现异常.文献[50,69]采用决策树模型构造主特征识别器,用以识别与注入故障最相关的日志信息.异常检测阶段则检查在线日志是否包含故障相关的日志信息,进而判断是否出现异常.文献[59]从日志中提取了 15 个特征,使用 PCA 方法对 15 个特征进行降维,使用 *K*-means 算法对所有日志进行聚类,找到的离群点即为检测出的异常.

3.4 异常检测方法分析

表 3 总结了基于日志数据的异常检测技术.基于图模型的异常检测方法的输入是已挖掘的日志模板,其优点在于充分利用日志序列的上下文关系,构建请求执行逻辑图,具备强大的异常检测能力,能够有效细粒度地探测包括分支异常、性能异常等在内的多种类型的复杂系统异常;缺点在于忽略了日志的文本特征和统计特征,特别是变量信息,无法探测日志中变量值变化表征的系统异常,同时,该方法对日志序列进行建模往往需要日志文本中包含一定的关联信息,具备一定的限制;另外,刻画系统多线程和并发的复杂情形十分困难,难以保障精确度,需要大量的离线日志建模过程.基于概率分析方法的输入是日志模板频率向量,执行效率很高,能够在线快速找到系统运行时的异常,无需离线训练过程;缺点在于其仅仅关注日志数据在时间轴上的表征,异常检测的范围和质量受限.基于机器学习的方法则需要将日志转化为可以计算的数字化的特征向量,该方法关注于日志特征的处理和使用,屏蔽了日志的异构性,适用范围广,可以反映出与异常相关的关键特征,进而帮助理解系统行为;缺点在于其十分依赖日志数据特征的选取和质量,可能需要大量的标注日志数据用于离线的异常检测模型训练,异常检测的范围和质量受限,难以适用于复杂异构多变的大规模分布式系统.

Table 3 Analysis of log-based anomaly detection approaches

表 3 基于日志数据的异常检测方法对比分析

异常检测	输入	效率	适用日志类型	优点	缺点
基于图模型的异常检测	日志/日志模板	低	事务型日志	充分利用日志序列的上下文关系,构建请求执行逻辑图,具备强大的异常检测能力,能够有效细粒度地探测包括分支异常、性能异常等在内的多种类型的复杂系统异常	忽略了日志的文本特征和统计特征,特别是变量信息,无法探测日志中变量值变化表征的系统异常,同时,该方法对日志序列进行建模往往需要日志文本中包含一定的关联信息,具备一定的限制;另外,刻画系统多线程和并发的复杂情形十分困难,难以保障精确度,需要大量的离线日志建模过程
基于概率分析的异常检测	日志/日志模板频率向量	高	事务型日志/操作型日志	执行效率很高,能够在线快速找到系统运行时的异常.无需离线建模过程	仅仅关注日志数据在时间轴上的表征,对系统异常探测的范围和质量有限
基于机器学习的异常检测	日志/日志特征向量	低	事务型日志/操作型日志	关注于日志特征的处理和使用,屏蔽了日志的异构性,方法适用范围广.机器学习结果分析可以反映出与异常相关的关键特征,进而帮助理解系统行为	十分依赖日志数据特征的选取和质量,可能需要大量的标注日志数据用于离线的异常检测模型训练,异常检测的范围和质量受限,难以适用于复杂异构多变的大规模分布式系统

4 基于日志数据的故障预测

基于日志数据的故障预测在系统运行过程中预测未来一定时间范围内是否会发生系统故障.通过学习故障发生前若干时间内的日志模式或变化趋势,在故障发生之前就是否会出现故障(label)或出现故障的概率(score)及时预警.基于日志数据的故障预测面临两个关键性技术挑战.(1) 系统故障前的日志数据往往不具备明显的异常特征,如何从微弱的异常趋势中判断系统问题是一个挑战.(2) 系统故障前往往伴随持续性的异常变化,如何尽早预测出系统故障成为另一个挑战.相关工作如图 7 所示.

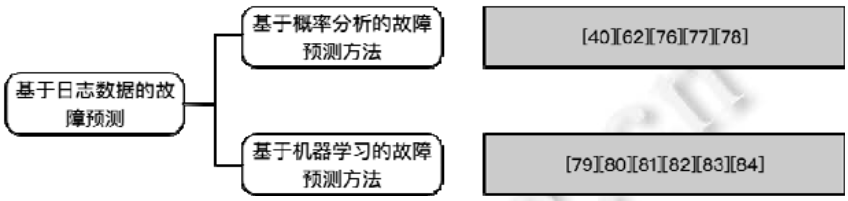

Fig.7 Related work of failure prediction approaches

图 7 基于日志数据的故障预测方法分类及相关文献

4.1 基于概率分析的故障预测

基于概率分析的故障预测首先找到系统故障点,使用 Error、Fatal 等日志信息标注;然后在故障点之前划分时间窗口,通过概率模型学习与故障相关的日志序列模式,计算每一种日志序列模式与故障的关联概率,进而预测故障.该方法依赖于日志序列模式和日志之间的因果关联关系,因此仅适用于事务型日志.

文献[40]计算每一个日志模板序列与故障之间的关联概率(频繁共现概率),并记录故障强相关的日志模板序列.比对每一个时间窗口内的日志模板序列与所记录的故障强关联的日志模板序列之间的差异,若相似,则认为未来会出现故障.文献[78,79]使用回归分析的方法挖掘日志模板与故障之间的关联关系,识别系统故障前不同类型日志的分布特征,通过比对在线日志与故障前日志的分布特征,预测是否会发生故障.文献[76]使用频繁项集挖掘算法识别动态时间窗口内的频繁日志模板序列,获得日志模板之间的关联和时序关系,使用链式结构记录日志模板之间的关联概率.通过这种链式结构能够有效预测下一段时间的日志模板及日志模板出现的频率,当日志模板及其频率与预测结果出现明显偏差时,则判断系统即将出现故障.

4.2 基于机器学习的故障预测

基于机器学习的故障预测方法依赖于有故障标签的日志数据,并将故障预测转化为机器学习中的分类问题.该方法通常需要提取故障发生前一段时间内的日志序列特征,要求日志序列具备上下文关联的特性以预测未来的日志表征,因此仅适用于事务型日志.文献[80]首先采用基于规则的结构化日志信息提取技术获得日志中每个域的结构化信息.然后,以日志等级为 FATAL 的日志作为系统故障的标签,进而生成训练数据.具体而言,挑选原始日志集合中所有日志等级为 FATAL 的日志,将该日志之前一个或数个时间窗口内的日志标记为与故障相关,其他时间窗口内的日志标记为与故障无关.采用贝叶斯网络训练分类模型判断系统未来是否会出现故障.

与文献[80]类似,文献[81]同样使用日志严重性等级最高的日志作为系统发生故障的标识(Error 等级),但是特征提取阶段则采用日志模板挖掘的方法提取日志模板,然后根据预定规则(已知一个请求的起始日志和终止日志)确定故障前的日志序列.这样,Error 日志之前的日志序列被标记为故障相关,其他的日志序列则被标记为故障无关.采用基于自然语言处理的日志特征提取方法构建每一个日志序列的特征向量,基于所生成的特征向量,训练支持向量机模型(SVM),进而判断在线日志序列是否会出现故障.文献[77]使用基于聚类的日志模板挖掘方法将日志数据划分为若干类型,并利用集成学习构建故障预测模型.

4.3 故障预测方法对比分析

表 4 列举了基于日志数据的故障预测技术的对比分析.基于概率分析方法的优点在于利用日志序列的上

下文关联关系,通过故障前日志序列的出现频率计算日志序列与故障的关联概率,执行效率高且无需训练过程;缺点在于需要先验知识切分日志序列,且需要大量带标签的日志数据计算日志序列与故障的关联概率,同时,从方法本身角度来看,该方法精确率较低,无法处理新出现的日志序列,无法适应异构复杂的系统运行情况.基于机器学习的技术关注于日志特征的处理和使用,屏蔽了日志的异构性,适用范围广,可以通过学习故障前日志序列的模式,充分利用日志的序列特征和文本特征,预测精确度高,对于新出现的日志序列也可以进行预测;缺点在于其十分依赖日志数据特征的选取和质量,可能需要大量的标注日志数据用于离线的故障预测模型训练,故障预测的范围和质量受限.

Table 4 Analysis of log-based failure prediction approaches

表 4 基于日志数据的故障预测方法对比分析

故障预测	输入	效率	适用日志类型	优点	缺点
基于概率分析的故障预测	日志/日志模板	高	事务型日志	利用日志的前后关联关系,通过故障前日志序列的出现频率计算日志序列与故障的关联概率,执行效率高且无需训练过程	1.需要先验知识切分日志序列; 2.需要大量带标签的日志数据计算日志序列与故障的关联概率; 3.方法简单,精确度较低,无法处理新出现的日志序列,无法适应异构复杂的系统运行情况
基于机器学习的故障预测	日志/日志特征向量	低	事务型日志	通过学习故障前日志序列的模式,利用日志的序列特征和文本特征,预测精确度高,对于新出现的日志序列也可以进行预测	十分依赖日志数据特征的选取和质量,可能需要大量的标注日志数据用于离线的故障预测模型训练,故障预测的范围和质量受限

5 基于日志数据的故障根因诊断

日志是系统管理或者开发人员理解系统行为的重要数据源,因此日志数据在辅助定位故障位置、诊断故障原因方面占据最重要的地位.基于日志数据的故障诊断是当故障或异常发生后,自动化地快速、准确定位故障位置,诊断引发系统故障的根本原因.其主要思路是通过机器学习、文本挖掘以及图模型算法从海量日志中找到与故障敏感的各种信息,包括故障相关的代码片段、日志集合、异常请求的执行路径或者故障类型等.代码片段是指与故障相关的代码及其上下文,日志集合是指与故障相关的一条或若干条日志,异常请求执行路径是指与故障相关的以日志序列为表征的请求执行路径,故障类型是指预设的故障根因类型(如网络中断、服务挂起、程序死锁等).从方法角度来看,主要包括基于关联推断的技术和基于机器学习的技术.图 8 总结了该技术的分类以及相关研究工作.多数研究工作集中在使用基于关联推断的技术恢复程序执行逻辑以及诊断故障时自动发现异常请求的执行路径等方面.

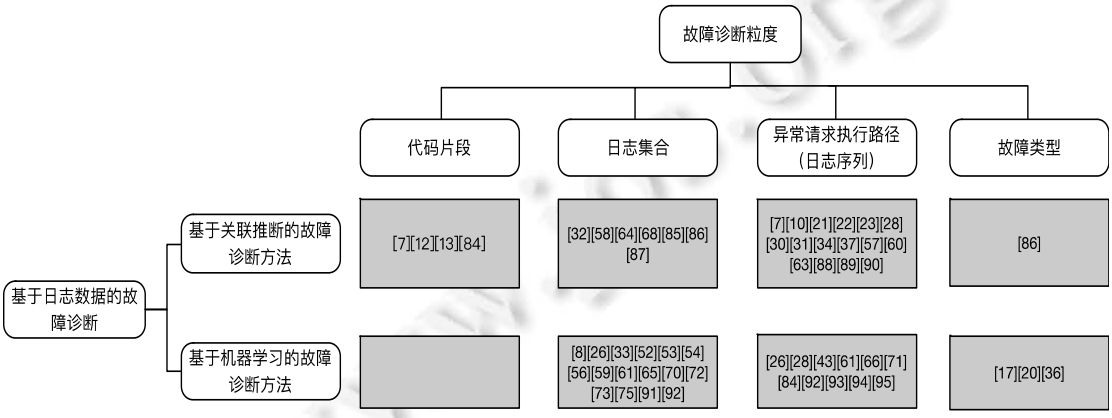

Fig.8 Related work of failure diagnosis approaches

图 8 基于日志数据的故障根因诊断方法分类及相关文献

5.1 基于关联推断的故障根因诊断

基于关联推断故障根因诊断技术通过代码分析、系统侵入、日志文本挖掘等手段对日志序列、日志内容、代码中的日志打印点进行描述性建模,推断与故障关联的日志信息,定位与故障相关的代码片段、日志集合以及异常请求执行路径等.该方法依赖于将日志序列与代码、请求执行路径、故障类型等进行关联,因此往往仅适用于事务型日志.

1. 故障诊断粒度为代码片段

Yuan 等人[12]通过代码分析的方法找到日志序列所对应的代码的执行逻辑,使用图模型对执行逻辑建模,用以诊断故障.他们提出一个名为 sherlog 的工具,当系统发生故障时,通过路径推断和变量推断,找到故障发生的代码片段,从而帮助系统管理人员了解故障前后程序执行的上下文.

Zhang 等人[84]对上述工作进行了拓展,通过日志找到关键的函数调用层级.其目的是通过代码分析、关联分析的方法细粒度地定义一系列规则关联运行时事件,用于重现故障.其基本前提假设是分布式系统为了使结构松耦合以方便自动化测试和单元测试,会定义一系列外层接口(API)调用(直接暴露给外界用户,用户可以通过若干 API 调用操作执行不同 task 或者 job),这些外层 API 调用也被称作用户指令(user command).当系统发生异常时,该方法以与异常相关联的日志为分析起点,逐层向上分析代码的调用和执行逻辑,并利用链式结构对其建模,直至用户指令层.这样,当执行某一个或一组用户指令时,就可以复现此故障.

与此类似,Zhao 等人[7]同样使用静态代码分析的方法找到包含日志打印语句的代码执行路径(函数调用路径).虽然他们的关注点是在分布式软件系统中如何将不同组件输出的日志整合起来,从中发掘请求的执行路径,但是静态代码分析可以用于关联故障与代码,也具备诊断代码片段的能力.与该方法不同,Ghanbari 等人[13]修改了 log4j 日志打印库,在其中添加 stageId 信息,用以关联单线程上的相同任务输出的日志.针对发布-订阅模式和生产者-消费者模式的代码模式解析,从而确定任务的起始位置,拦截不同线程对日志打印程序的调用,确定每一条日志的 stageId 的值.接着,stageId 信息被用于生成任务执行路径,时间戳被用于计算任务执行时间,使用 t-test 的方法发现异常任务(包括异常任务执行路径和异常日志任务执行时间).该方法同时也具备定位异常代码片段的能力.

2. 故障诊断粒度为日志集合

这类方法通常包括基于时间窗口切分的方法和基于关联分析的方法.基于时间窗口切分的方法按照时间窗口切分日志,对时间窗口内的日志提取特征,对日志建模发现故障时间窗口.基于关联分析的方法则对日志条目提取特征,针对故障日志条目,例如 ERROR 级别的日志等,挖掘这类日志条目与其他日志之间的关联关系,进而诊断出故障相关的日志集合.

Chen 等人[32]计算每一个时间窗口内每一个日志模板的频率,并分析这些日志模板频率的变化,根据日志模板频率统计结果的长尾分布特征,使用 MAD 作为频率变化的指标,找到频率突变的时间窗口作为故障窗口,该时间窗口内的日志数据即为故障日志集合.Chuah 等人[64]通过对不同日志事件在时间上的频率分布建模,计算日志事件之间的相似度,当发现某故障日志事件时,可以快速找到与该故障日志事件相关的其他日志事件集合以及这些日志在时间上的分布情况,帮助诊断系统故障原因.与该工作不同,Shang 等人[87]不考虑关联系统日志,而是关联开源社区中的开发人员知识.将故障日志事件关联源代码中的日志打印位置以及其所在方法的调用路径等,这些位置的日志打印语句输出的日志作为故障诊断依据;根据这些日志打印位置关联版本升级记录及软件缺陷库.在这个过程中,版本升级记录和软件缺陷库中包含的日志同样可以作为故障诊断依据.

3. 故障诊断粒度为异常请求执行路径

这类方法的主要思路是利用日志构建图模型,用以刻画软件系统的请求执行路径,进而与在线日志序列比对,发现异常日志序列并定位异常请求执行路径.

Zhao 等人[7]使用静态代码分析的方法找到包含日志打印语句的代码执行路径(函数调用路径),其关注点在于如何将分布式软件系统中不同组件产生的日志整合起来,从中发掘程序的执行逻辑.输出一个请求在跨组件和节点执行过程中打印的日志序列,而非更细粒度的代码执行逻辑,这个日志序列包含了清晰而全面的系统故

障信息.

Nguyen等人[88]利用虚拟机技术在软件系统运行过程中对服务器保存影子复制,当系统发生故障时,通过环境数据(配置文件、相关组件状态、系统调用等)和日志数据,在影子复制中重现该系统故障,推断能够输出这些日志数据的请求执行路径,即为最终故障诊断的结果.Chow 等人[57]提出一种基于日志的服务请求端到端跟踪的方法,该方法无需系统源代码或系统环境数据,仅依靠日志构建请求执行路径.首先使用基于规则的方法提取Facebook 系统日志中的信息,包括请求ID、节点信息、时间戳、事件名等;然后将每一个请求中的所有事件以事件片段为组织单位,表示为⟨task,start_event,end_event⟩3 元组;接着,建立以所有事件片段为节点的全连接有向无环图;最后,利用Facebook的每种类型服务均包含大量请求的特点,使用同类型服务中的大量请求不断地对该有向无环图剪枝,只有同类型服务中的所有请求均未出现冲突的关系结构才会被保留.通过这种方式,当系统发生故障时,该方法可以通过上述有向无环图准确定位异常请求执行路径.

Fu 等人[30]同样使用有向图模型对日志序列进行建模.通过进程ID和线程ID将同一个job的日志序列切分出来,利用这些日志序列建立有限状态机(finite state automaton,简称FSA)模型.其中,节点代表日志模板,边代表日志模板之间的转移关系.文献[21,60]同样关注于通过日志恢复一个请求的执行逻辑,进而逐步构建有向图模型.与文献[7,30]的不同之处在于,这些方法提出,在真实环境中系统源代码很难获取,且日志中通常不会包含精确的请求ID、线程ID或进程ID.因此,考虑根据日志中的关键变量对日志进行关联,利用不同日志中的多个相同变量值挖掘日志的关联关系,挖掘一个请求执行产生的完整日志序列.

5.2 基于机器学习的故障根因诊断

基于机器学习的故障根因诊断的基本思路是对日志或日志序列提取特征,利用分类或聚类算法检测故障日志或故障日志序列.该方法既可以利用不同时间窗口内的日志或日志模板数量分布特征,又可以利用日志序列的上下文关联特征,因此能够适用于事务型日志和操作型日志.

1. 故障诊断粒度为日志集合

这类方法通常划分时间窗口,对每一个时间窗口内的日志提取特征,采用机器学习算法学习这些特征进而判断异常时间窗口,并定位故障时间窗口内的日志集合.

饶翔等人[70]利用决策树模型学习故障日志和正常日志之间的差异,构造分类模型检测异常日志集合.将通过故障注入获得的故障相关日志作为特征提取对象,对这些特征日志进行建模,构造基于决策树模型的主特征识别器.进而对一段时间内的日志集合是否故障相关进行判断.

Yen 等人[59]提出一种基于无监督聚类的故障诊断方法.使用基于规则的日志结构化特征提取方法,从日志中提取了15个特征.使用主成分分析法(principal component analysis,简称PCA)对15个特征进行降维,使用*K*-means对所有日志聚类,最终找到的离群点即为检测出的异常.Xu 等人[8]同样使用无监督聚类的思路,但是与常用的切分时间窗口做法不同.该工作根据日志中的公共变量信息(如Tid,Pid等)将日志分组,对每一个组生成一个日志模板频率向量(message count vector).用PCA算法计算日志模板频率向量中的每个特征维度的影响,并将其中的关键特征维度组合成新的低维度特征向量,并计算大多数日志模板频率向量在这些关键特征维度上的取值范围.故障根因诊断阶段,将在线日志模板频率向量通过矢量计算的方法映射到关键特征维度上,并计算该向量与正常日志的特征向量之间的矢量距离,如果距离超过某阈值,则诊断出故障日志集合.

Davidsen等人[61]通过Task ID提取表征一个请求执行路径的日志序列,对日志序列进行聚类并存入知识库,通过人工校验以及历史知识,对这些日志序列进行标记.故障根因诊断过程通过查询并匹配知识库,定位故障敏感日志序列的集合.

2. 故障诊断粒度为异常请求执行路径

这类方法关注于处理和分析以请求为关注核心的日志序列.部分工作[26,92]同时考虑两种故障诊断粒度.

Fu 等人[28]通过对系统日志的挖掘获取表征程序执行逻辑的日志序列,提出了一种自动学习关键上下文因素(特殊的事件标识,如出现某些特殊的日志,出现特殊变量等)的方法.通过Job ID和Transaction ID提取属于同一个请求或者同一个事务的日志序列,使用形式概念分析(formal concept analysis,简称FCA)方法,对这些日志

序列进行建模,最后通过决策树模型找到影响系统执行分支的因素(日志或日志集合).

Lo 等人[93]假设日志序列已被组织为事件序列(event trace),针对这些事件序列提出了一种基于分类模型的故障根因诊断方法.该方法从时间序列中挖掘独立频繁序列模式,既考虑序列特征又考虑迭代特征;这些独立频繁序列模式被用于提取特征,并使用 LIBSVM 机器学习算法学习正常事件序列和故障事件序列的特征,进而检测故障事件序列.

Xu 等人[95]针对 kubernetes 管理的微服务架构系统同时应用分类算法与聚类算法,采用自然语言处理方法从微服务系统部署配置文件中提取特征,采用朴素贝叶斯进行分类,进而检测异常部署.同时,使用日志中不同的 ID 信息关联日志模板序列,采用 *k*-means 算法进行聚类进而找到异常类别,借此对数据进行标记,使用逻辑回归模型对日志模板序列训练分类模型以诊断故障序列.

近年来,深度学习在图像、语音等领域取得了非凡的成就,有研究人员将深度学习模型应用于基于日志数据的故障诊断领域,取得了良好的效果.其中的一个代表性工作是:Du 等人[43]把一个日志序列看作一个文本,日志序列中的每条日志对应的日志模板当作一个词.基于此,将多用于自然语言处理的 LSTM 模型对系统正常运行状态的日志序列建模(DeepLog 模型),在线比对日志序列与 LSTM 预测的日志序列之间的差异进行异常检测.首先从系统正常运行状态下产生的日志提取日志模板序列和日志变量向量,分别用于训练日志模板异常检测模型和日志变量异常检测模型,其中,日志模板序列还用于训练工作流模型,用于后续的对异常的理解和诊断;在预测阶段,DeepLog 会对系统新产生的每一条日志进行异常检测,将新产生的日志转换为日志模板和日志变量向量,随后用日志模板异常检测模型和日志变量异常检测模型进行检测.

3. 故障诊断粒度为故障类型

针对易复发故障(recurrent failure),Reidemeister 等人[20]通过统计模型提取特征并采用朴素贝叶斯分类算法对故障相关的日志进行识别.Reidemeister 等人[17]使用基于频繁项集挖掘的方法提取日志模板,然后将这些日志模板映射到已标注的日志文件,最后训练决策树分类模型对复发故障进行识别.与此不同,Jia 等人[36]关注于识别一类特殊故障,即软件缺陷导致的故障.利用开源软件缺陷库中包含的日志,使用多种机器学习算法学习这些日志特征,从而诊断软件缺陷导致的故障.使用基于聚类的日志模板挖掘算法将日志聚类,将开源软件缺陷报告中的日志序列转化为日志模板序列,通过自然语言处理的方法将日志模板序列转化为特征向量,使用 SVM、KNN 和多层感知器训练分类器识别故障.

5.3 故障根因诊断方法对比分析

表 5 列出了对不同的基于日志数据的故障根因诊断方法的对比分析.

Table 5 Analysis of log-based fault diagnosis approaches

表 5 基于日志数据的故障根因诊断方法分析

故障根因诊断	输入	适用日志类型	优点	缺点
基于关联推断的故障根因诊断	日志/日志模板	事务型日志	1.充分利用日志之间的上下文关联关系,恢复请求执行逻辑,可以细粒度地定位不同类型的系统故障; 2.可以定位与故障敏感的请求执行路径; 3.故障诊断效率较高,模型易于理解,能够有效地为运维人员修复故障提供支持	1.对日志序列进行建模难以精确刻画系统多线程和并发的复杂情形; 2.可能需要大量的系统知识辅助建模.需要离线的日志建模过程; 3.构建故障根因诊断模型可能需要日志文本中包含一些关联信息,具有一定的适用范围限制
基于机器学习的故障根因诊断	日志/日志模板	事务型日志/操作型日志	1.可以输出与故障敏感的日志集合或输出故障类型; 2.对日志的内容和系统知识的假设较少,方法适用范围广	1.需要离线训练阶段以及大量带标签的训练数据; 2.故障定位粒度较粗,辅助故障诊断的能力弱于基于模型的技术; 3.机器学习模型和深度学习模型往往难以理解,对运维人员修复故障的辅助能力不足

基于关联推断的方法可以充分利用日志之间的上下文关联关系,恢复请求执行逻辑,可以细粒度地定位不同类型的系统故障,并且可以定位与故障敏感的请求执行路径,故障诊断效率较高,模型易于理解,能够有效地为运维人员修复故障提供支持;缺点在于仅适用于事务型日志序列,无法检测操作型日志中的异常表征,对日志

序列进行建模难以精确刻画系统多线程和并发的复杂情形,可能需要大量的系统知识辅助离线建模过程,构建故障根因诊断模型可能需要日志文本中包含一些关联信息,如请求 ID、线程 ID 等,具有一定的适用范围限制;基于机器学习的方法可以方便输出与故障敏感的日志集合或输出故障类型,对日志的内容和系统知识的假设较少,能够有效地从不同的日志类型中提取较为全面的特征并构建模型,适用事务型日志和操作型日志;缺点在于十分依赖日志特征的选取和处理,需要离线训练阶段以及大量带标签的训练数据,故障定位粒度较粗,辅助故障诊断的能力弱于基于关联推断的方法.同时,机器学习模型和深度学习模型往往难以理解,对运维人员修复故障的辅助能力不足.

6 总结与未来展望

6.1 总 结

以第 1 节所述基于日志数据的分布式系统故障诊断研究框架为指导,对日志处理与特征提取技术、异常检测、故障预测及故障根因诊断技术的相关工作进行总结分析(见表 6),并得出如下结论:(1) 日志模板挖掘方法是日志处理与特征提取技术的主流,且该方法通常用于异常检测和故障根因诊断;(2) 日志特征提取方法,特别是基于规则的结构化信息提取方法通常与基于日志数据的故障预测组合;(3) 基于日志数据的故障根因诊断技术是当前的研究热点,相关研究工作应用了多种的日志模板挖掘方法和日志特征提取方法.

Table 6 Summary of log-based failure diagnosis approaches

表 6 基于日志数据的故障诊断技术相关文献总结

		日志处理与特征提取					
		日志模板挖掘方法			日志特征提取方法		
		基于静态代码分析的日志模板挖掘方法	基于频繁项集挖掘的日志模板挖掘方法	基于聚类的日志模板挖掘方法	基于自然语言处理的日志特征提取方法	基于规则的结构化信息提取方法	基于统计模型的日志特征提取方法
基于日志数据的异常检测	基于图模型的异常检测方法	–	[15,22]	[28,30,40]	[50–52]	[60]	–
	基于概率分析的异常检测方法	–	–	[27,32,33]	[49]	–	[67]
	基于机器学习的异常检测方法	[8,9] [10,11]	–	[28,73,74]	[56]	[59]	–
基于日志数据的故障预测	基于概率分析的故障预测方法	–	–	[40]	–	[62,76–78]	–
	基于机器学习的故障预测方法	–	–	–	[81,82]	[79,80,83]	–
基于日志数据的故障根因诊断	基于关联推断的故障根因诊断	[7,10,12] [13,84,89]	[21–23]	[28,30–32,37]	–	[58,60,63,64]	[32]
	基于机器学习的故障根因诊断	[8,92]	[17,20,95]	[26,28,33,36, 43,73,93]	[52–54,56]	[59,61,65,66]	[33]

进一步地,通过对日志处理与特征提取技术、异常检测、故障预测及故障根因诊断所使用的重要技术进行总结分析(见表 7),得出如下结论:(1) 机器学习和统计分析是基于日志数据的分布式软件系统故障诊断最常用的关键技术,在 4 个子技术中均有所应用;(2) 关联概率分析和有向图构建与计算是构建异常检测、故障预测、故障根因诊断模型的重要手段;(3) 日志处理与特征提取和基于日志数据的故障根因诊断中应用的技术最多.

Table 7 Summary of log-based failure diagnosis techniques

表 7 基于日志数据的故障诊断技术手段总结

	机器学习	静态代码分析	自然语言处理	统计分析	关联概率分析	频繁项集挖掘	有向图构建与计算
日志处理与特征提取	√	√	√	√	–	√	–
基于日志数据的异常检测	√	–	–	√	√	–	√
基于日志数据的故障预测	√	–	–	√	√	√	–
基于日志数据的故障根因诊断	√	√	√	√	√	–	√

6.2 未来展望

本文就日志处理与特征提取、基于日志数据的异常检测、基于日志数据的故障预测和基于日志数据的故障根因诊断 4 个关键技术,对基于日志数据的故障诊断研究领域的相关研究工作进行了综述和分析.虽然现有研究工作已经取得了一定的成果和进展,但该领域仍存在许多关键问题值得深入探讨和研究.

1. 故障诊断任务驱动的日志嵌入表示方法

现有日志处理与特征提取技术存在两个关键问题:(1) 该技术以简化日志表示、降低日志复杂性为根本目的而非提升故障诊断效率.例如,日志模板的本质是对海量日志的抽象,以模板的形式简化复杂的日志文本;日志特征提取方法则从日志文本中摄取少量的简单信息,如词频信息、特殊标识符信息等.这些方法提取的信息或特征可能与故障诊断任务无关,同时导致关键故障信息丢失.(2) 该技术难以做到精准和普适.例如,日志中的常量和变量交织分布情况可能及其复杂,以概率统计为基础的日志模板挖掘方法往往难以完全正确生成日志模板;而日志特征提取方法则依赖于日志文本的内容和结构,对于不同系统的日志而言,其提取的特征质量千差万别.为解决上述问题,未来可能从全新的角度思考日志处理与特征提取技术,借鉴基于深度学习的自然语言处理技术中的词嵌入表示(word embedding)方法,以故障诊断任务为目标构建或训练日志的嵌入表示方法.

2. 面向复杂系统的精准的故障预测模型与方法

现有基于日志数据的故障预测技术的相关研究工作较为薄弱,主要存在如下几个关键问题:(1) 现有故障预测模型多以机器学习算法为基础,方法较为简单,预测粒度粗,无法支持复杂的甚至是多故障集中爆发情况的预测;(2) 现有故障预测模型多以单日志序列为训练集,无法支持分布式系统中故障在多组件中蔓延和传播情况下的故障预测;(3) 现有故障预测方法往往仅利用了日志数据的少量文本或统计特征,在故障预测精确度方面仍有很大的提升空间.因此,未来可能以多组件协作的复杂分布式软件系统为目标,以时间序列分析技术、机器学习技术、图计算技术等为基本手段,构建面向复杂系统的精准故障预测模型.

3. 在线自学习、自更新异常检测、故障预测及故障根因诊断模型与方法

在当今 DevOps 的快速开发迭代环境中,系统更新极为频繁,随之带来的是日志的频繁更新与改变.这种频繁的系统更新要求故障诊断模型具备快速在线训练、更新和适应的能力,以保障故障诊断模型的可用性和有效性.然而,现有异常检测、故障预测及故障根因诊断模型均采用离线训练、在线使用的模式.这种模式仅支持离线重训练(re-training),速度慢、效率低,无法适应频繁的系统更新.因此,未来需要研究异常检测、故障预测及故障根因诊断模型的在线训练、更新方法与策略,故障诊断方法的自适应扩展及集成技术,实现异常检测、故障预测及故障根因诊断模型的在线动态快速自训练、自更新、自学习及自适应.

References:

[1] Elliot S. DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. International Data Corporation (IDC), 2014.

[2] The 10 biggest cloud outages of 2018. 2018. https://www.crn.com/slide-shows/cloud/the-10-biggest-cloud-outages-of-2018

[3] https://yq.aliyun.com/articles/603866/, 2018.

[4] https://www.ithome.com/html/it/372627, 2018.

[5] Market guide for AIOps platforms. 2016. https://www.gartner.com/doc/3892967/market-guide-aiops-platforms

[6] What is AIOps. 2018. https://www.bmc.com/blogs/what-is-aiops

[7] Zhao X, Zhang Y, Lion D, *et al*. Lprof: A non-intrusive request flow profiler for distributed systems. In: Proc. of the 11th Symp. on Operating Systems Design and Implementation. USENIX, 2014. 629–644.

[8] Xu W, Huang L, Fox A, *et al*. Detecting large-scale system problems by mining console logs. In: Proc. of the Int'l Conf. on Machine Learning. IEEE, 2009. 117–132.

[9] Xu W, Huang L, Fox A, *et al*. Mining console logs for large-scale system problem detection. In: Proc. of the Workshop on Tackling Computer Systems Problems with Machine Learning Techniques. San Diego: IEEE, 2008. 4.

[10] Xu W, Huang L, Fox A, *et al*. Online system problem detection by mining patterns of console logs. In: Proc. of the 9th Int'l Conf. on Data Mining. Miami: IEEE, 2009. 588–597.

[11] Xu W. System problem detection by mining console logs [Ph.D. Thesis]. Berkeley: University of California, 2010.

[12] Yuan D, Mai H, Xiong W, *et al*. SherLog: Error diagnosis by connecting clues from run-time logs. In: Proc. of the 15th Edition on Architectural Support for Programming Languages and Operating Systems. ACM, 2010. 143–154.

[13] Ghanbari S, Hashemi AB, Amza C. Stage-aware anomaly detection through tracking log points. In: Proc. of the 15th Int'l Middleware Conf. ACM, 2014. 253–264.

[14] Vaarandi R. A breadth-first algorithm for mining frequent patterns from event logs. In: Proc. of the Int'l Conf. on Intelligence in Communication Systems. IEEE, 2004. 293–308.

[15] Nandi A, Mandal A, Atreja S, *et al*. Anomaly detection using program control flow graph mining from execution logs. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2016. 215–224.

[16] Makanju AAO, Zincir-Heywood AN, Milios EE. Clustering event logs using iterative partitioning. In: Proc. of the 15th Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2009. 1255–1264.

[17] Reidemeister T, Jiang M, Ward PAS. Mining unstructured log files for recurrent fault diagnosis. In: Proc. of the Int'l Symp. on Integrated Network Management. DBLP, 2011. 377–384.

[18] Vaarandi R. A data clustering algorithm for mining patterns from event logs. In: Proc. of the Int'l Conf. on IT Operations & Management. IEEE, 2003. 119–126.

[19] Vaarandi R, Pihelgas M. LogCluster: A data clustering and pattern mining algorithm for event logs. In: Proc. of the Int'l Conf.on Network and Service Management. IEEE, 2015. 1–7.

[20] Reidemeister T, Munawar MA, Ward PAS. Identifying symptoms of recurrent faults in log files of distributed information systems. In: Proc. of the Network Operations and Management Symp. IEEE, 2010. 187–194.

[21] Tak BC, Tao S, Yang L, *et al*. LOGAN: Problem diagnosis in the cloud using log-based reference models. In: Proc. of the Int'l Conf. on Cloud Engineering. IEEE, 2016. 62–67.

[22] Babenko A, Mariani L, Pastore F. AVA: Automated interpretation of dynamically detected anomalies. In: Proc. of the Int'l Symp. on Software Testing and Analysis. IEEE, 2009. 237–248.

[23] Mariani L, Pastore F. Automated identification of failure causes in system logs. In: Proc. of the Int'l Symp. on Software Reliability Engineering. IEEE, 2008. 117–126.

[24] Gainaru A, Cappello F, Trausan-Matu S, *et al*. Event log mining tool for large scale HPC systems. In: Proc. of the European Conf. on Parallel Processing. Berlin, Heidelberg. Springer-Verlag, 2011. 52–64.

[25] Vaarandi R. Mining event logs with SLCT and loghound. In: Proc. of the Network Operations and Management Symp. IEEE, 2008. 1071–1074.

[26] Lin Q, Zhang H, Lou J G, *et al*. Log clustering-based problem identification for online service systems. In: Proc. of the 38th Int'l Conf. on Software Engineering Companion. ACM, 2016. 102–111.

[27] Lou JG, Fu Q, Yang S, *et al*. Mining invariants from console logs for system problem detection. In: Proc. of the ATC. USENIX, 2010. 231–244.

[28] Fu Q, Lou JG, Lin Q, *et al*. Contextual analysis of program logs for understanding system behaviors. In: Proc. of the Int'l Conf. on Mining Software Repositories. IEEE, 2013. 397–400.

[29] Ding R, Fu Q, Lou JG, *et al*. Healing online service systems via mining historical issue repositories. In: Proc. of the Int'l Conf. on Automated Software Engineering. IEEE, 2012. 318–321.

[30] Fu Q, Lou JG, Wang Y, *et al*. Execution anomaly detection in distributed systems through unstructured log analysis. In: Proc. of the Int'l Conf. on Data Mining. IEEE, 2009. 149–158.

[31] Lou JG, Fu Q, Wang Y, *et al*. Mining dependency in distributed systems through unstructured logs analysis. ACM SIGOPS Operating Systems Review, 2010,44(1):91–96.

[32] Chen C, Singh N, Yajnik S. Log analytics for dependable enterprise telephony. In: Proc. of the 9th European Dependable Computing Conf. IEEE, 2012. 94–101.

[33] Du S, Cao J. Behavioral anomaly detection approach based on log monitoring. In: Proc. of the Int'l Conf. on Behavioral, Economic and Socio-cultural Computing. IEEE, 2015.

[34] Li T, Liang F, Ma S, *et al*. An integrated framework on mining logs files for computing system management. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. Chicago: ACM, 2005.

[35] Aharon M, Barash G, Cohen I, *et al*. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In: Proc. of the Joint European Conf. on Machine Learning and Knowledge Discovery in Databases. Berlin, Heidelberg: Springer-Verlag, 2009. 227–243.

[36] Jia T, Li Y, Tang H, *et al*. An approach to pinpointing bug-induced failure in logs of open cloud platforms. In: Proc. of the Int'l Conf. on Cloud Computing. IEEE, 2016. 294–302.

[37] Debnath B, Solaimani M, Gulzar MAG, *et al*. LogLens: A real-time log analysis system. In: Proc. of the 38th Int'l Conf. on Distributed Computing Systems (ICDCS). IEEE, 2018. 1052–1062.

[38] He P, Zhu J, He S, *et al*. Towards automated log parsing for large-scale log data analysis. IEEE Trans. on Dependable & Secure Computin, 2017,(99):1.

[39] He P, Zhu J, Zheng Z, *et al*. Drain: An online log parsing approach with fixed depth tree. In: Proc. of the Int'l Conf. on Web Services. IEEE, 2017. 33–40.

[40] Watanabe Y, Otsuka H, Sonoda M, Kikuchi S, Matsumoto Y. Online failure prediction in cloud datacenters by real-time message pattern learning. In: Proc. of the Int'l Conf. on Cloud Computing Technology and Science (CloudCom). IEEE, 2012. 504–511.

[41] Jiang ZM, Hassan AE, Hamann G, *et al*. An automated approach for abstracting execution logs to execution events. Journal of Software Maintenance & Evolution Research & Practice, 2008,20(4):249–267.

[42] Jiang ZM, Hassan AE, Flora P, *et al*. Abstracting execution logs to execution events for enterprise applications. In: Proc. of the 8th Int'l Conf. on Quality Software. IEEE, 2008. 181–186.

[43] Du M, Li F, Zheng G, *et al*. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proc. of the 2017 Conf. on Computer and Communications Security. ACM, 2017. 1285–1298.

[44] Zhu KQ, Fisher K, Walker D. Incremental learning of system log formats. ACM SIGOPS Operating Systems Review, 2010,44(1): 85–90.

[45] Hamooni H, Debnath B, Xu J, *et al*. Logmine: Fast pattern recognition for log analytics. In: Proc. of the 25th Int'l Conf. on Information and Knowledge Management. ACM, 2016. 1573–1582.

[46] Mizutani M. Incremental mining of system log format. In: Proc. of the Int'l Conf. on Services Computing. IEEE, 2013. 595–602.

[47] 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. ACM, 2011. 785–794.

[48] Du M, Li F. Spell: Streaming parsing of system event logs. In: Proc. of the 16th Int'l Conf. on Data Mining. IEEE, 2016. 859–864.

[49] Oliner AJ, Aiken A. Online detection of multi-component interactions in production systems. In: Proc. of the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Hong Kong: IEEE, 2011. 49–60.

[50] Oliner AJ, Aiken A, Stearley J. Alert detection in system logs. In: Proc.of the Int'l Conf. on Data Mining. IEEE, 2008. 959–964.

[51] Stearley J, Oliner AJ. Bad words: Finding faults in spirit's syslogs. In: Proc. of the Int'l Symp. on CLUSTER. IEEE, 2008. 765–770.

[52] Juvonen A, Hamalainen T. An efficient network log anomaly detection system using random projection dimensionality reduction. In: Proc. of the 6th Int'l Conf. on New Technologies, Mobility and Security. IEEE, 2014. 1–5.

[53] Sipola T, Juvonen A, Lehtonen J. Anomaly detection from network logs using diffusion maps. In: Proc. of the IFIP Advances in Information & Communication Technology, IEEE, 2011,363:172–181.

[54] Sipola T, Juvonen A, Lehtonen J. Dimensionality reduction framework for detecting anomalies from network logs. In: Proc. of the Engineering Intelligent Systems. 2012.

[55] Bertero C, Roy M, Sauvanaud C, *et al*. Experience report: Log mining using natural language processing and application to anomaly detection. In: Proc. of the 28th Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2017. 351–360.

[56] Brown A, Tuor A, Hutchinson B, *et al*. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. arXiv Preprint arXiv:1803.04967, 2018.

[57] Chow M, Meisner D, Flinn J, *et al*. The mystery machine: End-to-end performance analysis of large-scale internet services. In: Proc. of the 11th Symp. on Operating Systems Design and Implementation. USENIX, 2014. 217–231.

[58] Chuah E, Jhumka A, Narasimhamurthy S, *et al*. Linking resource usage anomalies with system failures from cluster log data. In: Proc. of the 32nd Int'l Symp. on Reliable Distributed Systems. IEEE, 2013. 111–120.

[59] Yen TF, Oprea A, Onarlioglu K, *et al*. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proc. of the Int'l Conf. on Computer Security Applications. IEEE, 2013. 199–208.

[60] Yu X, Joshi P, Xu J, *et al*. CloudSeer: Workflow monitoring of cloud infrastructures via interleaved logs. SIGOPS Operating Systems Review, 2016,50(2):489–502.

[61] Davidsen B, Kristensen E. Pinpoint: Problem determination in large, dynamic Internet services. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2002. 595–604.

[62] Liang Y, Zhang Y, Sivasubramaniam A, *et al*. BlueGene/L failure analysis and prediction models. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2006. 425–434.

[63] Beschastnikh I, Brun Y, Ernst MD, *et al*. Inferring models of concurrent systems from logs of their behavior with CSight. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014. 468–479.

[64] Chuah E, Kuo S, Hiew P, *et al*. Diagnosing the root-causes of failures from cluster log files. In: Proc. of the Int'l Conf. on High Performance Computing. IEEE, 2010. 1–10.

[65] 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. USENIX, 2012. 26.

[66] Mi HB, Wang HM, Zhou YF, *et al*. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. Science China Information Sciences, 2012,55(12):2757–2773.

[67] Lim C, Singh N, Yajnik S. A log mining approach to failure analysis of enterprise telephony systems. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2008. 398–403.

[68] Yan X, Zhou W, Gao Y, *et al*. PADM: Page rank-based anomaly detection method of log sequences by graph computing. In: Proc. of the Int'l Conf. on Cloud Computing Technology and Science. IEEE, 2014. 700–703.

[69] Oliner AJ, Kulkarni AV, Aiken A. Using correlated surprise to infer shared influence. In Proc. of the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Chicago: IEEE, 2010. 191–200.

[70] Rao X, Wang HM, Chen ZB, *et al*. Detecting faults by tracing companion states in cloud computing systems. Chinese Journal of Computers, 2012,35(5):856–870 (in Chinese with English abstract).

[71] Rao X, Tian Q, *et al*. Sub-sequence feature vector-based massive system log anomaly detection in cloud computing systems. In: Proc. of the National Conf. of Information Storage. 2012 (in Chinese with English abstract).

[72] Vinayakumar R, Soman KP, Poornachandran P. Long short-term memory-based operation log anomaly detection. In: Proc. of the Int'l Conf. on Advances in Computing, Communications and Informatics. IEEE, 2017. 236–242.

[73] Lu S, Wei X, Li Y, *et al*. Detecting anomaly in big data system logs using convolutional neural network. In: Proc. of the 16th Int'l Conf. on Dependable, Autonomic and Secure Computing. IEEE, 2018. 151–158.

[74] Meng WB, Liu Y, *et al*. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. IEEE, 2019.

[75] Cui Y, Sun Y, Hu J, *et al*. A convolutional auto-encoder method for anomaly detection on system logs. In: Proc. of the Int'l Conf. on Systems, Man and Cybernetics (SMC). IEEE, 2018. 3057–3062.

[76] Gainaru A, Cappello F, Fullop J, *et al*. Adaptive event prediction strategy with dynamic time window for large-scale HPC systems. In: Proc. of the Managing Large-scale Systems Via the Analysis of System Logs & the Application of Machine Learning Techniques. ACM, 2011. 1–8.

[77] Lan Z, Gu J, Zheng Z, *et al*. A study of dynamic meta-learning for failure prediction in large-scale systems. Journal of Parallel & Distributed Computing, 2010,70(6):630–643.

[78] Navarro JM, Parada GHA, Duenas JC. System failure prediction through rare-events elastic-net logistic regression. In: Proc. of the Int'l Conf. on Artificial Intelligence, Modelling and Simulation. IEEE, 2014. 120–125.

[79] Shalan A, Zulkernine M. Runtime prediction of failure modes from system error logs. In: Proc. of the Int'l Conf. on Engineering of Complex Computer Systems. IEEE, 2013. 232–241.

[80] Yu L, Zheng Z, Lan Z, *et al*. Practical online failure prediction for Blue Gene/P: Period-based vs. event-driven. In: Proc. of the Int'l Conf. on Dependable Systems and Networks Workshops. IEEE, 2011. 259–264.

[81] Fronza I, Sillitti A, Succi G, *et al*. Failure prediction based on log files using random indexing and support vector machines. Journal of Systems & Software, 2013,86(1):2–11.

[82] Fulp EW, Fink GA, Haack JN. Predicting computer system failures using support vector machines. In: Proc. of the 1st USENIX Workshop on the Analysis of System Logs. San Diego: USENIX, 2008.

[83] Sahoo RK, Oliner AJ, Rish I, *et al*. Critical event prediction for proactive management in large-scale computer clusters. In: Proc. of the ACM Int'l Conf. on Knowledge Discovery and Data Mining. Washington: ACM, 2003. 426–435.

[84] Zhang Y, Makarov S, Ren X, *et al*. Pensieve: Non-intrusive failure reproduction for distributed systems using the event chaining approach. In: Proc. of the 26th Symp. on Operating Systems Principles. ACM, 2017. 19–33.

[85] Yamanishi K, Maruyama Y. Dynamic syslog mining for network failure monitoring. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2005. 499–508.

[86] Liang Y, Zhang Y, Sivasubramaniam A, *et al*. Filtering failure logs for a bluegene/l prototype. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2005. 476–485.

[87] Shang W, Nagappan M, Hassan AE, *et al*. Understanding log lines using development knowledge. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 21–30.

[88] Nguyen H, Dean DJ, Kc K, *et al*. Insight: In-situ online service failure path inference in production computing infrastructures. In: Proc. of the Annual Technical Conf. USENIX, 2014. 269–280.

[89] Zhao X, Rodrigues K, Luo Y, *et al*. Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle. In: Proc. of the 12th Symp. on Operating Systems Design and Implementation. USENIX, 2016. 603–618.

[90] Tan J, Pan X, Kavulya S, *et al*. SALSA: Analyzing logs as state machines. WASL, 2008,8:6.

[91] Rao X. Research on log-based trust management in large-scale distributed software system [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2011 (in Chinese with English abstract).

[92] Kc K, Gu X. ELT: Efficient log-based troubleshooting system for cloud computing infrastructures. In: Proc. of the Int'l Symp. on Reliable Distributed Systems. IEEE, 2011. 11–20.

[93] Lo D, Cheng H, Han J, *et al*. Classification of software behaviors for failure detection: A discriminative pattern mining approach. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2009. 557–566.

[94] Mi H, Wang H, Yin G, *et al*. Performance problems diagnosis in cloud computing systems by mining request trace logs. In: Proc. of Network Operations and Management Symp. IEEE, 2012. 893–899.

[95] Xu J, Chen P, Yang L, *et al*. LogDC: Problem diagnosis for declartively-deployed cloud applications with log. In: Proc. of the 14th Int'l Conf. on e-Business Engineering (ICEBE). IEEE, 2017. 282–287.

附中文参考文献:

[70] 饶翔,王怀民,陈振邦,等.云计算系统中基于伴随状态追踪的故障检测机制.计算机学报,2012,35(5):856–870.

[71] 饶翔,田青,朱鸿宇,等.云计算系统中基于子序列特征向量的海量日志异常检测方法.见:全国信息存储技术学术会议.2012.

[91] 饶翔.基于日志的大规模分布式软件系统可信保障技术研究[博士学位论文].长沙:国防科学技术大学,2011.

贾统(1993－),男,博士,主要研究领域为分布式计算,智能运维.

吴中海(1968－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为大数据技术,系统安全,嵌入式软件.

李影(1975－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算,可信计算.