云环境下基于统计监测的分布式软件系统故障检测技术研究

王 焘 张文博 徐继伟 魏 峻 钟 华
(中国科学院软件研究所 北京 100190)

摘 要 越来越多的分布式软件系统部署在公有云计算平台,通过互联网向外提供服务.云计算环境的复杂性、动态性和开放性使得分布式软件系统更易于出现故障,造成服务失效,从而影响大量用户正常使用,甚至造成巨大经济损失.故障检测技术旨在自动及时的检测系统故障的发生,以避免或减少服务失效所带来的损失,是保障分布式软件系统性能与可靠性的关键技术之一.云计算环境对该技术带来了新的挑战,该文首先分析了这些挑战.基于统计监测的故障检测技术在线搜集监测数据构建统计模型,并基于该模型对系统运行状态进行分析与预测,具有实时监测分析、自动化检测、无需领域知识等优势,能够满足云环境的需要,因此引起了学术界和工业界的广泛关注.该文提出了面向云计算环境的基于统计监测的分布式软件系统故障管理参考框架,包括分布式监测、监测数据处理、故障检测、故障诊断以及故障处理等模块;将已有工作分成基于规则、度量分析、日志分析和行为分析等四大类,逐类介绍其实现原理,并对比分析各类的优缺点;针对当前云计算环境的特点,从在线自动检测、运行环境感知和组件交互分析等 3 个方面展望了未来的研究方向.

关键词 云计算;软件监测;分布式软件系统;软件故障检测;性能异常检测;统计监测
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.00397

A Survey of Fault Detection for Distributed Software Systems with Statistical Monitoring in Cloud Computing

WANG Tao ZHANG Wen-Bo XU Ji-Wei WEI Jun ZHONG Hua
(*Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract More and more distributed software systems are deployed on public cloud computing platforms to provide services using the Internet. These systems are prone to many types of faults because of the complexity, dynamism and openness of cloud computing. These faults often lead to service failures affecting a large population of users and even resulting in serious economic loss. Fault detection, which aims at timely accurately detecting faults to avoid failures or reduce economic loss, has become one of the most key technologies for guaranteeing the performance and reliability of distributed software systems. Cloud computing environment has raised great challenges to the fault detection of distributed software systems, and then we first introduce these challenges. The fault detection method based on statistical monitoring builds statistical models with online collected monitoring data to analyze and predict system status, which is suitable for cloud computing environment. The method with advantages in online analysis and automatic

收稿日期:2015-06-04;在线出版日期:2016-01-14.本课题得到国家自然科学基金(61402450)、北京市自然科学基金(4154088)、国家科技支撑计划(2015BAH55F02-4)、国家"九七三"重点基础研究发展规划项目基金(2015CB352201)、国家"八六三"高技术研究发展计划项目基金(2013AA041301)资助.**王 焘**,男,1982 年生,博士,助理研究员,中国计算机学会(CCF)会员,主要研究方向为软件可靠性、软件故障检测、自主计算、云计算.E-mail: wangtao@otcaix.iscas.ac.cn. **张文博**,男,1976 年生,博士,研究员,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为分布式计算和软件工程. **徐继伟**,男,1985 年生,博士研究生,主要研究方向为分布式计算和软件工程. **魏 峻**,男,1970 年生,博士,研究员,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为分布式计算和软件工程. **钟 华**,男,1971 年生,博士,研究员,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为分布式计算和软件工程.

detection without domain knowledge has widely attracted the attention of industrial and academic communities. Then we propose a reference framework of fault management with statistical monitoring for distributed software systems, which includes distributed monitoring, data processor, fault detection, fault diagnosis and fault actor modules. After that, we categorize existing works as rule based, metric analysis, log analysis, and behavior analysis methods. Furthermore, we introduce typical works for each category, and compare these categories in strength and weakness. Finally, we direct the future works in online automatic detection, runtime environment awareness and component interaction analysis.

Keywords cloud computing; software monitoring; distributed software systems; software fault detection; performance anomaly detection; statistical monitoring

1 引 言

当前分布式应用的不断发展给人们带来了巨大便利,电子邮件、电子商务、网上银行等已经成为人们日常工作和生活中不可或缺的一部分,而这些应用通常依托云计算平台,部署成分布式软件系统以提供在线服务[1].然而应用的多样性以及部署环境的动态性使得分布式软件系统时常出现故障,从而无法正常对外提供服务,甚至造成商业方面的巨大经济损失.例如 2008 年 10 月 Google Gmail 账户服务失效长达 24 h,导致大量用户不能访问自己部署的云应用;2009 年 8 月 Paypal 系统出现故障导致服务中断一个小时,在处理顾客交易方面造成高达 720 万美元的经济损失①;2011 年 4 月 Amazon EC2 服务中断持续了 100 多个小时,导致美国东部地区依托其服务的上千家公司无法对外提供服务②.Tellme Networks 报告指出,分布式软件系统失效恢复时间的 75%用在检测故障,18%用在诊断故障[2],早期检测故障能够缓解或阻止 65%的失效发生[3].因此及时检测故障,并准确诊断问题原因,是确保分布式软件系统提供高效和可靠服务的关键.

分布式软件系统故障是由运行时复杂原因所造成的(如资源竞争、配置错误、软件缺陷、硬件失效),具有不确定性,难以重现(如并发处理造成的死锁、错误状态的传递)[4-5].这类故障占到了软件故障的 15%~80%[6],但无法在软件开发和测试过程中完全发现并消除,同时系统管理员难以人工跟踪系统运行状态并及时发现问题.因而故障检测技术被广泛应用,以在线监测系统运行,自动及时检测故障的发生,以避免或减少服务失效所带来的损失.

在云计算环境下,分布式软件系统的故障检测技术面临的挑战主要体现在以下几个方面:

(1)自动化分析.云计算环境下的分布式软件系统通常由成百上千个节点构成,同时又分为众多层次,面对如此规模巨大的系统,系统管理员无法根据经验人工分析系统所出现的问题.

(2)问题组件定位.分布式软件系统通常由众多组件构成,分布在不同节点,调用各种服务,组件间交互关系复杂、关联度高,难以准确定位引起系统故障的故障组件.

(3)领域知识缺少.租户部署在云计算平台的应用通常对平台提供者和管理者是透明的,这就使得系统运维人员通过设计文档或代码注入获取应用结构,进而建模分析变得困难.

(4)在线检测.许多软件系统的故障经常是在大规模运行过程中表现出来的,而系统运维人员难以在离线环境下重现产品运行中所出现的问题,以跟踪定位问题原因.

(5)环境适应性.执行环境在应用运行过程中会发生动态变化(如外部负载波动、应用在多主机迁移、虚拟机资源动态调整),应用也会随之表现出与环境相应的行为,难以采用离线建立的模型对系统状态进行准确刻画.

为了应对以上挑战,近年来,统计监测方法引起了工业界和学术界的关注,被广泛应用在故障检测技术方面,获得了较多研究成果.统计监测方法是基于在线搜集的监测数据构建概率统计模型,运用模型对系统运行状态进行预测与分析[7].将统计监测应用于故障检测具有以下优势:

(1)无需领域知识对系统行为和故障特征进行

① Shankland S. PayPal suffers from e-commerce outage. Available: http://news.cnet.com/8301-1023_3-10302072-93.html, August 3, 2009

② AWS. Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. Available: http://aws.amazon.com/message/65648/, April 29, 2011

事先刻画，具有较广的适用范围.

(2) 实时监测和在线分析系统行为，能够有效监测到难以在测试环境下获取的系统行为.

(3) 自动化分析系统状态，能够简化系统管理操作，更好应对规模巨大、复杂性高的系统.

本文面向云计算环境下提供在线服务的分布式软件系统，首先提出了基于统计监测的分布式软件系统故障管理参考框架. 然后，对已有方法进行分类，介绍了各类代表性的工作，并分析各类方法所存在的问题. 最后，对未来的研究方向提出了建议.

2　故障管理参考框架

基于统计监测的方法根据系统历史监测数据制定报警规则，或建立统计模型作为基准. 这类方法通常需要满足假设：系统在大多数情况下能够正常运行，而在特定环境下触发故障. 例如某个请求引发未经测试的内存泄露，在某种访问序列下并发线程竞争共享资源造成死锁，或是动态迁移到相同主机的多个应用产生资源竞争等. 基于统计监测的故障检测方法通常分为监测、建模和检测三个阶段.

(1) 在监测阶段. 利用监测工具搜集各层次的监测数据.

(2) 在建模阶段. 在系统正常运行情况下，建立系统运行状态的概率统计模型.

(3) 在检测阶段. 将当前运行状态与建立的模型进行比较，当监测状态与模型偏离则检测为系统故障.

基于统计监测的故障管理系统通常在线搜集各节点的监测数据，进而自动化分析以提前预测、及时检测并准确诊断故障，最后针对问题原因作出相应处理. 基于以上分析，本文提出了故障管理参考框架，如图 1 所示，由五个核心模块构成，包括分布式监测、监测数据管理、故障检测、故障诊断及故障处理等，基本功能如下：

(1) 分布式监测. 用于搜集分布式软件系统运行时各节点的监测信息. 每个服务器节点上都需要部署一个监测代理，从物理主机、虚拟机、中间件以及应用等多个层次分别搜集监测数据，而后将获取的监测数据通过网络传输给监测数据管理模块. 监测对象主要包括以下几个层次：

① 基础设施层. 各物理设备的资源使用信息(如 CPU 利用率、内存占用率、磁盘读写率、网络链接数据传输速率)；

② 操作系统层. 各虚拟机的资源使用信息(如虚拟机内存占用)，尤其是共享敏感资源(如多级缓存)；

③ 中间件层. 各种逻辑资源信息(如线程、队列、数据库连接池)；

④ 应用层. 应用相关信息(如响应时间、吞吐量、用户访问模式、应用组件交互行为).

(2) 监测数据管理. 用于预处理监测数据. 根据故障预测、检测和诊断方法的需要提取监测数据，对数据进行清洗(填补遗漏和平滑噪声)、集成(多节点监测数据合并)和转换(规格化数据). 对历史数据进行处理并存储，用以建立统计模型，并基于在线数据

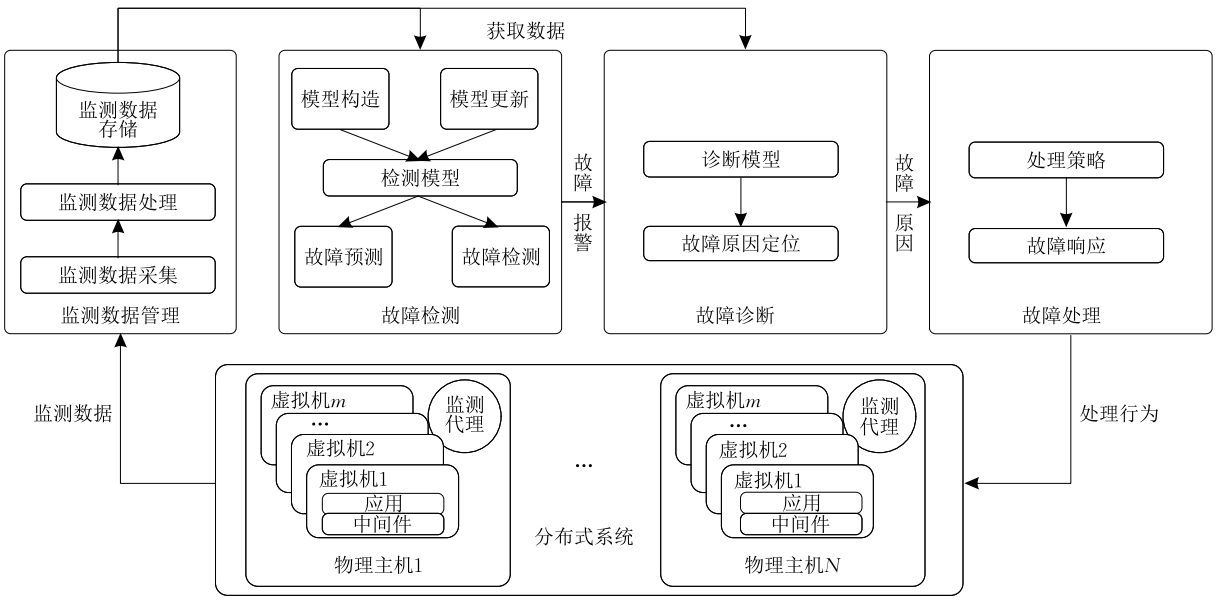

图 1　故障管理参考框架

更新模型.

(3)故障检测.用于根据监测数据预测系统出现故障的可能性,并分析系统当前运行状态.在启动阶段,根据监测数据建立系统状态模型;在运行过程中,根据运行环境变化(如负载波动、资源调整以及更新升级),对模型进行动态更新;同时根据模型在线预测并检测系统运行状态是否异常.当预测或检测到异常状态时,将报警信息汇报给故障诊断模块以做进一步分析.

(4)故障诊断.用于对问题进行分析以定位原因.从资源角度考虑,通过分析各资源使用状况和变化趋势以定位异常系统度量.将问题原因汇报给故障管理模块,以便对可能出现或者即将发生的故障进行主动响应.

(5)故障处理.用于针对特定问题做相应处理.例如在云计算环境下,通过动态调整虚拟机分配的资源、虚拟机在线迁移和虚拟机镜像克隆等技术实现自动调整、获取、释放资源,以达到资源按需分配,保障服务质量的效果.再如对于基于组件的软件系统,利用组件重启机制进行软件恢复.

故障管理包括数据采集与预处理、故障预测与检测、故障诊断、故障恢复(或故障规避)等,本文在故障管理参考框架下对故障检测技术进行综述.

3 故障检测方法

故障检测方法通常可以分为基于规则和异常检测等两类,方法分类如图 2 所示.

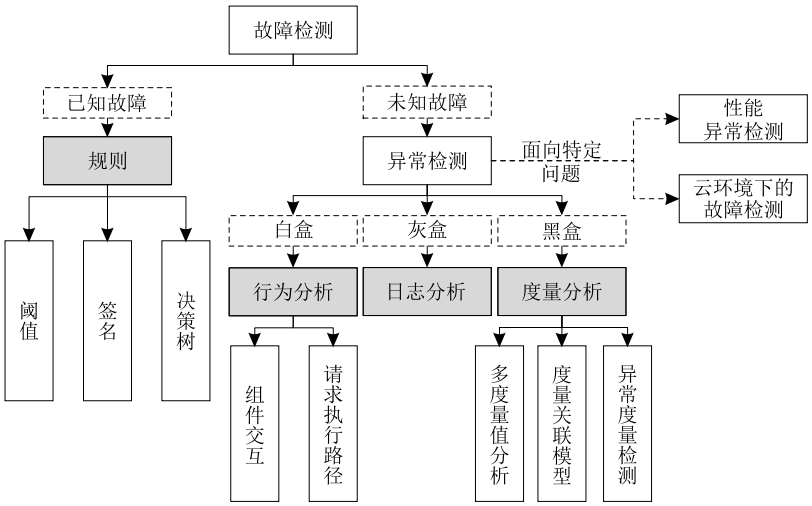

图 2 故障检测方法分类

基于规则的方法根据历史故障所表现的现象来定义故障出现时可辨别的特征,而后将观察到的现象与已定义的故障特征进行匹配.当匹配成功则检测为有故障,发出警报;否则认为系统运行正常.另一方面,基于异常检测的方法为目标系统建立模型作为基准,将系统行为与基准进行对比.根据对系统内部结构的了解程度以及监测分析对象的不同,基于异常检测的方法可以分为黑盒、灰盒和白盒.

(1)黑盒方法.典型的是基于度量分析的方法,不需要了解系统内部结构,通过调用操作系统提供的接口搜集监测数据进行分析.

(2)灰盒方法.典型的是基于日志分析的方法,通过分析日志信息可以了解一部分系统执行路径.

(3)白盒方法.典型的是基于行为分析的方法,通过代码注入等方式搜集各组件行为.

本节将对各类方法的原理以及具有代表性的工作进行介绍.

由于分布式软件系统所提供服务的性能直接影响着用户体验,关乎着企业的经济效益,一些工作关注于系统的性能表现.同时近年来众多工作针对虚拟化云计算环境的特点展开研究.性能异常检测和面向云环境的故障检测通常基于异常检测方法展开研究,本节也将进行介绍.

最后,本节针对第 1 节分析的云计算环境对分布式软件系统提出的挑战,对各类方法的优缺点进行对比分析.

3.1 基于规则的方法

基于规则的故障检测方法描述一系列规则,当监测数据与其中任何规则匹配成功时,则检测为某种故障,并执行相应动作.

3.1.1 基于阈值的方法

基于阈值的方法是一种简单的基于规则的方

法，其事先设置各度量的阈值，当监测的度量值超出阈值，故障检测系统则根据规则自动向管理员发出警报. 例如：

规则 1. IF 80%≤CPU_Util.<95% Then Alert=Yellow and Send Email to Operator

表示：当目标系统节点的 CPU 利用率大于 80%且小于 95%时，为黄色警报，故障检测系统向管理员发送 Email 报警.

规则 2. IF CPU_Util.≥95% Then Alert=Red and Send Email&SM to Operator

表示：当目标系统节点的 CPU 利用率大于 95%时，则为红色警报，故障检测系统向管理员同时发送 Email 和短信报警.

这种方法简单易用，在工业产品中被广泛采用（如 IBM Tivoli，HP OpenView，EMC Smarts），但分布式软件系统监测度量数量众多，人工为每个度量设置合理的阈值需要领域知识，难度较大.

3.1.2　基于签名的方法

基于签名的方法事先将故障发生时的特征定义为签名，在运行过程中将监测到的系统状态与已知故障签名进行匹配，如果匹配成功，则检测为与签名相对应的故障. FlowDiff 框架[8] 面向数据中心，对系统各层次进行监测，从框架、应用和操作等几个层次分别定义了故障签名：框架签名获取网络物理拓扑结构，应用到服务器的映射关系以及基准性能参数（如网络链接利用率和端到端的延迟）；应用签名获取每个应用的行为（如响应时间和吞吐量）以及应用间的交互行为；操作签名获取用户或应用所进行的操作（如迁移虚拟机和添加服务器）. FlowDiff 将当前数据中心的状态与过去已知健康状态进行比较，获取产生的偏差签名，将这些偏差签名集合与已定义的故障特征进行匹配以分析问题. 例如作者在应用层定义了 5 个签名如表 1 所示，并且定义了 12 种签名组合所对应的故障类型如表 2 所示.

表 1　应用层签名

名称	签名	意义
连接图	CG，Connectivity Graph	节点间通信关系
流统计	FS，Flow Statistics	流传输数据的数量以及持续时间
组件交互	CI，Component Interaction	每个节点进/出边流的数量
延迟分布	DD，Delay Distribution	节点间的传输延迟
偏相关	PC，Partial Correlation	流之间的关联强度

表 2　应用层签名与故障的对应关系

故障类型	主机失效	主机性能	应用失效	应用性能	网络连接	网络瓶颈	交换机配置	交换机开销	控制器开销	交换机失效	控制器失效	未授权访问
签名	CG	DD	CG	DD	CG	DD	CG	DD	DD	CG	CG	CG
	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	CI
	CI	FS	CI	FS	CI	FS	CI	FS	FS	CI	CI	FS
	FS		FS		FS		FS			FS	FS	
	DD		DD		DD		DD			DD	DD	

文献[9]利用历史经验建立知识库来记录以往故障信息（包括故障原因、故障症状、失效恢复策略），每次检测故障，在故障数据库中查找症状相似的故障实例. 作者利用网络图来定义故障签名，图中节点为属性，链接为属性间的关联性. 故障发生时，属性间的关联性打破，记录此时的网络图作为故障签名，当故障再次发生时，提取相似网络图以分析问题.

3.1.3　基于决策树的方法

决策树是一种预测模型，可以用来表示系统各属性与最终状态之间的映射关系. 树中每个节点代表某个属性，从根节点到叶子节点经历的路径可抽取为一条规则，表示在某种情况下系统出现故障的概率. Pinpoint[10] 在检测到可能出现故障的请求后，记录每个组件的特征，而后利用 ID3 算法来学习决策树. 这些特征对应于 Pinpoint 搜集到的路径信息（如集群主机的 IP 地址和 EJB 组件名）. 一旦建立决策树，从树根到叶子节点形成执行路径，这样就可以将决策树转换为规则集合，利用这些规则进行故障检测.

3.2　基于度量分析的方法

基于系统度量分析的方法并不需要已知系统内部结构及请求处理流程，而只是利用操作系统提供的接口即可搜集监测数据，以分析度量值的变化或建立度量间的关联模型. 这类方法无需对系统进行代码注入，监测开销较小，且适用范围较广. 该方法需要满足以下假设：系统故障的触发能够反映到可监测度量. 例如内存泄露会反应到内存分配数量持续增加，自旋锁造成一段时间内线程陷入循环等待或死循环占用 CPU 资源.

3.2.1　基于多度量值分析的方法

基于多度量值分析的方法分析搜集到的多个度

量值信息. Aniketos[11]利用计算几何学方法来检测系统故障,同样分为训练和检测两个阶段. 在训练阶段,搜集系统正常运行时多维度监测数据,每个时刻的监测数据作为一个点,来建立几何闭包以表示系统的正常运行状态空间. 在检测阶段,如果运行时搜集的监测数据不能够包含在计算几何闭包之内,则检测系统处于异常状态.

文献[5]利用信息熵的方法进行故障检测,分为以下 3 个步骤:

(1)对度量进行分组. 在无故障发生的系统操作期间,周期性的从目标系统搜集度量,利用归一化互信息(Normalized Mutual Information,NMI)来测量任意两个度量的相似性. 进而获得相似度矩阵,应用分层聚类(Hierarchical Agglomerative Clustering,HAC)将相关联的度量划分为簇,簇间距离为两个簇中元素间的最大距离. 将每个度量作为单一簇,而后依次合并距离最近的簇,直到两个簇间距离超出预定阈值,或是所有度量属于同一个阈值,则完成聚类过程.

(2)计算每个簇的熵值. 熵用来衡量簇的不确定性,由于同一簇内的度量是强相关的,那么在给定时间内,相同簇内度量值间的不确定性要低于不相关度量的不确定性,而且应保持稳定. 所以熵可以作为簇的特征来表现簇中多度量的当前状态,而不必考虑度量的实际值.

(3)利用簇熵进行故障检测. 相同簇在不同时间的熵具有可比性,较大的波动意味着度量间关联强度变化,因此可以通过监测各簇熵的变化来检测故障的发生. 作者对于每个簇建立两个滑动窗口,一个为最近的 n 个熵,另一个为先前的检验窗口. 利用非参数化的 Wilcoxon Rank-Sum 方法来检验两个窗口的采样集合是否符合同分布,以监测熵是否发生了较大的变化,从而检测故障的发生.

以上的方法适用于运行环境稳定的情况,环境的变化会引起所建立模型的改变,从而造成检测结果的不准确. 针对此问题,我们此前的工作[12]面向采用多层架构的 Web 系统,通过实验结果分析发现负载的变化(包括并发数量和访问序列)会造成系统度量值的变化以及关联模型的改变. 进而根据 Web 应用的负载波动呈现出周期性的特点,提出了一种负载感知的故障检测方法. 如图 3 所示,我们首先引入了负载向量对动态负载进行刻画,而后提出一种在线增量式聚类方法对负载模式进行训练与识别. 在此基础上,在一定负载模式下利用局部离群因数(Local Outlier Factor,LOF)来表示系统运行状态,通过监测 LOF 值的变化来监测系统健康状况以检测故障. 由于考虑到了负载对故障检测造成的影响能够有效提高检测的准确性.

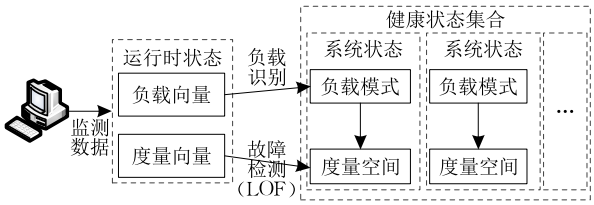

图 3 负载感知的 Web 系统故障检测

3.2.2 基于度量关联模型的方法

基于度量关联模型的方法建立度量间的关联关系,通过监测关联性的变化来分析系统运行状态. 该类方法基于假设:正常运行的软件系统的监测度量间呈现长期稳定的关联性. 例如对于多层架构的 Web 应用,前端 Web 服务器 HTTP 请求的激增,通常会引起后端数据库服务器 SQL 请求的激增,那么 HTTP 请求和 SQL 请求之间显然存在着关联性. 这类工作主要包括以下几个阶段:搜集系统运行时的监测度量,根据应用需要选择特定类型的统计模型,计算参数得到模型,利用新的监测数据检验得到的模型. 建立的模型可以分为线性和非线性回归两类:

(1)线性回归模型

由于线性关系在现实系统中普遍存在,较为简单且时间复杂度低,许多工作利用其建立系统度量间的关联性. 文献[13]为感兴趣的度量创建多个线性回归模型,利用最小二乘回归来对各变量的参数进行估算,如下式所示:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \tag{1}$$

其中,y 为关注度量的预测值,x_i 为已知各度量实际值,b_i 为参数.

作者只是关注于建立度量间的线性关联模型以帮助理解系统,而未提及如何进行系统故障检测.

文献[14]尝试利用简单线性回归(Simple Linear Regression,SLR)和鲁棒线性回归(Robust Linear Regression,RLR)模型,在足够长的周期内搜集各度量的监测数据,利用关联检验来发现具有关联关系的度量,以建模度量间的稳定关联性,进而采用交叉验证检验模型的有效性. 建立关联性模型可以采用增量式方式,在上个监测周期的基础上逐步增加监测度量,直到达到定义的最大度量数量或者诊断结果不会改变. 当监测所有度量的开销较高时,则可以采用增量式;当监测开销可以接受时则对所有度量都进行考虑.

(2)非线性回归模型

Jiang 等人[4]关注于存在稳定关联关系的度量对，利用自回归(Auto-regressive Regression with eXogenous input，ARX)来建模度量间的关联性，如下式所示：

$$y(t)+a_1y(t-1)+\cdots+a_ny(t-n)=b_0x(t-k)+\cdots+b_m(t-k-m) \quad (2)$$

其中，n，m，k 是模型的阶数，决定了此前多少步骤影响当前步骤的结果.

作者通过监测度量间的关联性变化来分析系统运行状态，当多个关联性出现较大波动，则认为系统出现故障. 与简单线性回归模型相比，ARX 模型能够考虑此前多个运行状态，并且具有动态更新的特点. 但是建立 ARX 模型的时间复杂度较高，为发现度量对间存在关联性的时间复杂度 $O(n^2)$ 乘以模型建立的开销(n 为度量数量). 因此作者讨论两个算法以加快发现度量关联性：第一个算法，将相关的度量划分为若干簇，搜索每个簇中的稳定关联性；第二个算法，利用度量关联的传递性优化模型的学习过程[15]. 该方法效果很大程度上取决于建立 ARX 模型的参数设定，但设置合理的参数较为困难. 如果阶数范围设定过小，则不能获得合理的模型. 相反如果阶数范围设定过大，则出现过度拟合，计算复杂度将成倍增加. 作者对这些参数设定了范围($0\leq n,m,k\leq 2$)而不是固定的数字，以学习多个候选模型，根据实验结果来选择一个合适的模型.

文献[16]比较五种回归模型，包括 SLR、转换数据的简单线性回归(SLR using Transformed Data，SLRT)、平滑数据的简单线性回归(SLR using Smoothed Data，SLRS)、ARX 和局部加权回归(Locally Weighted Regression，LWR). 其中，SLRT 尝试通过对线性模型进行数据变换，引入非线性关系，具有以下形式：

$$y'(x)=b_0+b_1T(x) \quad (3)$$

其中，$T(x)$为 x 的非线性转换，$y'(x)$为 $y(x)$经过转换后的表示，b_0 和 b_1 为参数.

找到合适的非线性转换方法相当重要，作者尝试一些简单的函数形式来描述多种非线性关系，如 $T(x)=\log(1+x)$，其逆转形式 $T(x)=(x+1)^{-1}$，平方根形式 $T(x)=x^{0.5}$.

该方法包括建立模型、检验模型和检测故障等三个阶段. 在建模阶段，搜集监测数据对度量对进行建模，利用系数 R^2(coefficient of determination)来辨别关联模型的适合程度，选择具有较高 R^2 的模型. 在检验阶段，进行更多的实验，搜集更多的监测数据，检验模型是否能够解释这些采样，若能成功解释则认为建立的模型是稳定的. 在检测阶段，计算当前监测数据与目标模型的偏差，当其超出预先定义的阈值，则认为与已知模型相冲突. 从系统总体进行考虑，对所有建立的模型都进行检查，以检测发现了多少异常. 当所有模型中很大比例都报告异常，就怀疑系统出现了故障. 与某个度量相关的模型发生冲突的越多，这个度量的异常分数也就越高，对这些分数进行排序，而后提取这个度量所属的组件. 然而这种方法的有效性取决于众多阈值的设定，包括 R^2、偏离程度以及冲突模型占总模型的比例等. 这些阈值通常难于事先确定，并且不同的应用通常会有不同的阈值.

Peerwatch 面向虚拟化数据中心，引入典型关联分析(Canonical Correlation Analysis，CCA)来建模部署在不同虚拟机上的，相同应用的多个实例间的关联性，用以检查每个应用实例的状态[17]. 在操作过程中，如果一些关联性发生较大幅度的下降，Peerwatch 认为系统处于故障状态，并且发出警报. 与其他众多实例的关联性发生改变的实例，则检测为故障实例. 文献[18]观察到某些度量的监测数据点紧密围绕着中心点，因此采用高斯混合模型(Gaussian Mixture Model，GMM)来建模度量间的关联性.

这些方法虽然具有一定故障检测能力，然而在实践当中面临着很多问题. 很多方法仅能够建模度量关联中的一些特定形式. 例如 SLRT 通常只是涉及到一些特殊的数据转换形式(如逆转、平方函数). 另外这些方法的有效性很大程度上取决于关键参数的设定，例如 GMM 要求设定适当的聚类数量，LWR 要求选择合适的平滑参数. 同时这些模型对于不同类型的应用，在不同场景都需要配置不同的参数. 此外建立这些模型(如 GMM、LWR)会具有较高的时间复杂度. 例如训练 GMM 模型通常需要期望最大化算法，其开销是 $O(sck)$，其中，c 是聚类数量，s 是用于学习的采样大小，k 是收敛所需要递归的次数. 再如 LWR 模型中，需要找到最近邻居以适应局部回归，其开销为 $O(s\log(s))$. 对于复杂系统，大量的监测数据需要在线分析，这种方法的使用会产生较高的开销.

3.2.3　异常度量检测方法

异常度量检测就是根据监测数据来检测出现异常的系统度量. 常见的方法包括基于贝叶斯网、基于

特征选择、基于相似度和基于数理统计的方法等等，下边分别进行介绍.

(1) 基于贝叶斯网的方法

贝叶斯网(Bayesian Network)是一种有向图结构，是人工智能、概率理论、图论、决策理论相结合的产物.它利用网络结构的有向图表达各要素之间的关联关系及影响程度，图中节点表示信息要素，有向边表示各信息要素之间的关联关系，用条件概率表表达各信息要素之间的影响程度.

基于贝叶斯网的故障检测技术目的在于在故障表现和系统度量之间建立基于概率的因果关系.通过学习故障出现的先验概率来对贝叶斯网络进行训练.诊断时根据先验概率求解故障现象下各种系统度量的后验概率.贝叶斯网采用图形化的网络结构直观表达变量的联合概率分布，同时能够计算各特征造成故障的概率.文献[19-20]定义系统 SLA 状态为 $S=\{S^+,S^-\}$，其中，S^+ 为 SLA 符合，S^- 为 SLA 冲突；度量值向量为 $\boldsymbol{M}=[m_0,m_1,\cdots,m_n]$，其中，$m_i$ 为第 i 个度量值.那么，每个度量 m_i 对 SLA 冲突的影响可以量化为概率模型 $\mathrm{Prob}(m_i|S^-)$，概率越高，则该度量 m_i 引起故障的可能性也就越高.

(2) 基于数理统计的方法

基于数理统计的方法基于基本假设：正常数据实例发生在随机模型概率较高的区域，而故障实例发生在概率较低的区域.该技术引入统计模型以符合已有数据，并使用某种统计推断检验(Statistical Inference Test)来判断监测数据是否符合该模型，将那些符合度较低的监测数据判定为故障.例如 χ^2 检验是较为常见的假设检验方法，用来判定两个组件监测数据集是否符合同分布，若不符合，该组件则检测为故障[10,21].

我们此前的工作[22]基于在相同负载模式下，度量遵循稳定分布的假设，利用学生 T 检验的方法，计算各度量与正常运行状态下搜集监测数据的符合程度以检测可疑度量.我们观察到在正常状态下，单位负载组件的资源占用保持稳定，但当出现性能异常，资源占用将会出现较大波动[23].因此，我们利用 XmR 控制图来监测各组件资源利用是否稳定，由此来检测出现性能异常的组件.同时我们在[24]中应用 EWMA 来检测负载与资源和性能关联系数的异常波动.该类方法具有较小的计算复杂度，适应于在线检测的场景.

(3) 基于特征选择的方法

我们观察到，系统故障的发生通常是由于某种系统资源使用异常引起的，而该资源问题通常会表现为相应的度量发生巨大波动.根据此观察结果，我们此前的工作[24]提出基于特征选择的可疑度量检测方法.将检测到故障发生前后搜集到的数据实例标记为正例和反例，而后选择变化较大的度量作为与故障相关的可疑度量.该问题可抽象为特征选择问题，以获得故障发生前后分布发生较大变化的系统度量，并量化度量异常程度以检测异常资源.如图 4 所示，在检测到异常发生前的监测实例标记为 True，异常发生后的监测实例标记为 False，则各度

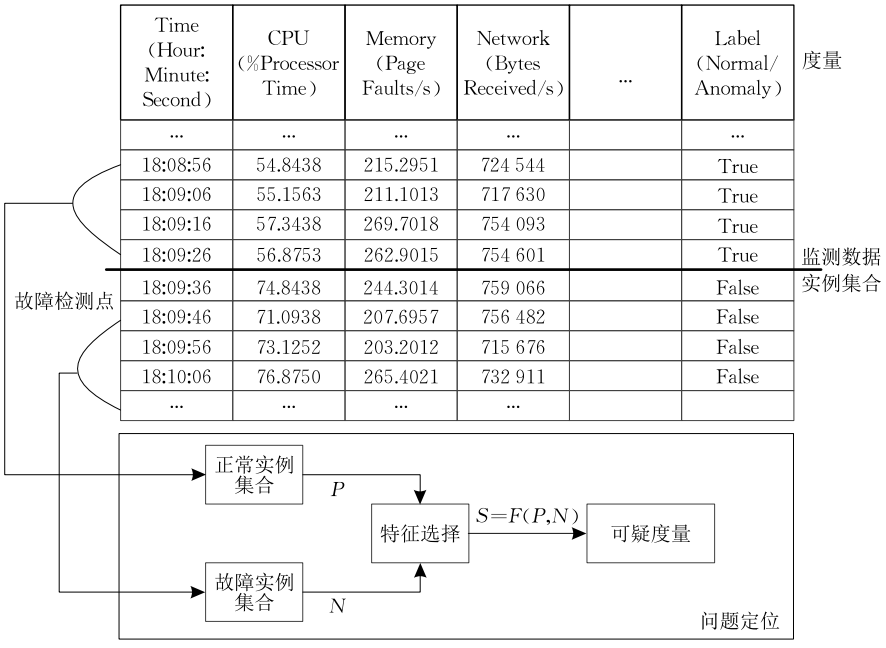

图 4 基于特征选择的可疑度量定位

量异常程度集合 S 为 $S=F(P,N)$，其中，P 为标记为 True 的监测数据实例集合；N 为标记为 False 的监测数据实例集合；F 为特征选择方法.

(4) 基于相似度的方法

基于相似度的方法检测故障节点(或可疑度量)，通过观察，当某个节点发生故障，或某个度量出现异常时，则与其关联的多个模型都可能发生冲突；对于未发生故障的节点(或度量)，可能只会有少量与之关联的模型发生冲突. 这类方法通常使用 Jaccard 系数(Jaccard coefficient)来量化节点(或度量)与故障的相关性.

假设系统中存在 n 个子节点(或度量)，任意两个节点(或度量)间都存在关联模型，那么，系统中共有个 $L=0.5\times n\times(n-1)$ 个关联模型. 模型冲突向量为 $\boldsymbol{T}_b$，它有 L 个分量，代表 L 个节点对，发生模型冲突的分量为 1，否则为 0. 每个节点对应一个向量 $\boldsymbol{T}_i(1\leq i\leq n)$，它有 L 个分量，当该节点包含在第 i 个节点对中，该节点第 i 个分量置为 1，否则置为 0.

表 3　四个节点例子

节点对	阈值违反($\boldsymbol{T}_b$)	T_1	T_2	T_3	T_4
1	1	1	1	0	0
2	0	1	0	1	0
3	0	1	0	0	1
4	1	0	1	1	0
5	1	0	1	0	1
6	0	0	0	1	1

如表 3 所示，以四个节点(六个节点对)为例，计算节点 Jaccard 系数的公式如下：

$$J_i=\frac{|T_i\bigcap T_b|}{|T_i\bigcup T_b|},\ 1\leq i\leq n \tag{4}$$

Jaccard 系数越大，则相应节点是故障节点的可能性越大，将所有节点按 Jaccard 系数从大到小排序. 表 3 中的四个节点的 Jaccard 系数分别为 0.2，1.0，0.2 及 0.2，则节点 2 更可能是故障节点. Jaccard 系数被广泛应用在基于度量关联的方法中以检测可疑度量[16,18].

文献[5]将组件的重要性引入到故障组件检测当中，假定组件的依赖关系可以通过源代码、设计文档或执行路径获得，那么计算组件的重要性时需要同时考虑与之交互的其他组件，并对其进行动态调整. 作者将组件的重要性定义为被调用的可能性，将计算组件重要性的问题抽象为计算 Internet 上 Web 页面访问可能性的问题. PageRank 是经典的计算页面访问可能性的算法，因而作者引入 PageRank 算法来计算组件的重要性. PageRank 通过将浏览 Web 页面的行为建模为一个随机过程，并且使用静态概率分布作为流行度的测量. 作者将组件对应于页面，组件依赖对应于页面索引，组件调用对应于状态转移. 同时，作者考虑到流行组件应该具有更高的权重，通过评估组件的流行程度来调整组件的异常分数.

3.3　基于日志分析的方法

当前复杂分布式软件系统中，大量日志能够通过相对简单的机制搜集，这些从不同层次搜集的日志对于故障检测具有较高的价值. 然而在实际系统中，这些日志信息并未得到充分利用，本节首先介绍日志的搜集与解析，而后讨论通过挖掘日志信息来检测故障的相关工作.

传统日志信息是通过标准的打印语句输出的(如 C 语言中的 printf、C++语言中的 cout 流). 许多高级的日志框架(如 log4j①、SLF4j)旨在提高日志的产生方式，支持柔性(即运行时可配置)的日志形式，能够提升日志输出的 I/O 性能，并且提供了多种存储方式. 这些工具的部署减少了程序中搜集日志的操作，极大提高了大规模系统中日志的可管理性. 程序员使用这些框架可以方便的产生日志，并且得到更多信息(如时间戳和与每个消息相关的线程 ID). 当前实际系统中，日志通常利用 Syslog② 来搜集，由于 Syslog 的日志信息通过远程过程调用(Remote Procedure Call，RPC)发送到 Syslog 服务器，因此，难以应用在大规模集群系统. 文献[25]实现了一个分布式日志搜集系统，在 Hadoop 文件系统③存储日志文件夹，这种设计不只对日志文件夹提供可靠性保障，而且 Map-Reduce 编程模式使得访问与处理日志文件更为有效.

日志搜集工具在实践中被普遍应用，但分析工具的缺乏使得大量文本日志未得到充分利用. 已有工作大都基于能够准确检测消息类型的假设，例如文献[26-27]手动为 SNMP 数据标识类型，但消息类型会多达成千上万[28]，使得这种方式难以实施. 也有工作利用数据挖掘技术(如频繁项集)来找到频繁出现的字符串及其所在的位置以检测消息类型. 文献[29]利用聚类和模式为日志抽取消息模板. IPLoM[30-31] 递归采用优化方法来检测相似日志信

① Gulcu C. Short introduction to log4j. Available: http://logging.apache.org/log4j, 2002

② Lonvick C. RFC3164 — the BSD syslog protocol. Available: http://www.faqs.org/rfcs/rfc3164.html, 2001

③ Borthakur D. The hadoop distributed file system: Architecture and design. Available: http://hadoop.apache.org, 2007

息中不同的部分,以决定消息类型.这种方法对于日志中频繁出现的消息类型具有较好的效果,但不能够应对运行时偶尔出现的消息类型.

基于模式匹配的方法(如 Logsurfer[32]、Swatch[33]和 OSSEC①)是最常用的日志分析方法,操作人员需要指定正则表达式以从日志中提取文本模式,而后对抽取的模式进行简单的分类聚集.此类系统存在的主要问题是这些模式通常难以获得和维护,大多数模式并不能反映真实问题,或者会产生太多的警报,误报率较高[34].此类系统和传统的脚本在本质上并无差别,它们将日志建模成为英文单词集合,只考虑日志的文本属性,但单词语义并不能够全面表现系统问题.文献[35]首先对系统源码进行分析,从源代码中获得日志模版,以〈name,type〉形式展现变量列表和消息类型,将这些模版与日志消息相匹配,从而将日志消息从非结构化文本转换为结构化数据.通过选择变量将日志消息分组,从中抽取信息构建特征向量.相同组的日志消息是强关联的,特征之间存在着关联性,偏离关联模式的为异常向量.作者利用 PCA 对特征向量进行异常检测,将特征向量标识为正常和异常,同时为了更好理解异常检测结果,利用决策树来可视化结果.

基于时序分析的方法,建模日志序列(如隐式马尔科夫模型[28]),建立时间与失效之间的关联性,以检测与系统失效相关的消息.然而这种方法仅将日志作为单一的时间序列,在产生相互交叉日志的大规模集群系统中,模型将会变得过于复杂,并且相互交叉的日志使得参数难于调整,不会有很好的表现.文献[31]从日志流中抽象出有限状态机(Finite State Machine,FSM),每个消息类型抽象成为一种状态,所期望的消息状态序列由 FSM 捕获,当日志序列不符合 FSM 则检测为故障.

3.4 基于行为分析的方法

基于系统行为分析的方法,通过监测框架(如 X-trace[36])对中间件进行注入,以跟踪组件交互行为和请求处理路径,基于此对系统行为进行建模.该方法需要了解系统内部执行流信息(如数据流和控制流),某些系统信息可以在设计文档、源代码或配置源文件中分析得到.但大多数情况下,系统对于管理员是透明的,这时,只能够利用面向方面编程(Aspect Oriented Programming,AOP)[37]等方法进行代码注入,以搜集运行时的系统行为.该方法根据关注点的不同分为面向组件交互行为和面向请求执行路径的方法.前者关注于基于组件的系统中组件之间的交互行为,后者则面向"请求-响应"交互型系统(如电子商务应用)的请求处理路径.这种方法与系统或应用特点相关,能够发现应用层故障,但应用场景和用户访问模式的变化会导致组件交互行为和请求处理执行路径的改变,因此,这种方法不适应负载模式或部署环境动态变化的情况.

3.4.1 面向组件交互行为

面向组件交互行为的故障检测方法关注于单个组件与其他组件的交互行为,当交互行为发生了改变,则表明该组件的状态发生了变化.Pinpoint 将组件实例间调用频率作为检验的参考分布,当组件实际行为与参考分布不符,则表明组件发生故障.

组件交互行为建模为

$$Q=\sum_{j=1}^{k}\frac{(N_j-w_j)^2}{w_j} \tag{5}$$

其中,当前组件与组件 j 的实际调用频率为 N_j,期望调用频率为 w_j,与当前组件交互的组件数量为 k.

该模型为符合自由度为 $(k-1)$ 的 χ^2 分布,利用其检验当前行为与参考模型的偏差.Q 是组件当前行为与期望行为符合相同潜在分布的程度,Q 值越高则监测状态与正常状态分布的偏差也就越大.计算置信度为 α 的阈值,将 Q 与阈值进行比较,以此来检验组件是否出现故障.

Pinpoint 从系统内部组件的角度考虑,而文献[21]则关注于用户访问行为的变化.通过分析 HTTP 访问日志来分析用户行为,其改变是站点失效的表现.该方法基于假设,在正常条件下,不同时间间隔搜集到的各页面点击次数应该符合相同分布.通过系统监测,可以获得两个数据集合 $A=\{a_1,a_2,\cdots,a_n\}$ 和 $B=\{b_1,b_2,\cdots,b_n\}$,其中,A 和 B 为不同时间段各组件点击率的集合,a_i 和 b_i 分别为组件 i 在不同时刻的调用频率:

$$S_a=\sum_{i=1}^{n}a_i,\ S_b=\sum_{i=1}^{n}b_i,\ s_i=a_i+b_i \tag{6}$$

对于每个 a_i 和 b_i,计算期望值:

$$E_i^A=s_iS_a/(S_a+S_b),\ E_i^B=s_iS_b/(S_a+S_b) \tag{7}$$

$$\chi^2=\sum_{i=1}^{n}(a_i-E_i^A)^2/E_i^A+(b_i-E_i^B)^2/E_i^B \tag{8}$$

则可以用自由度为 $(n-1)$ 的 χ^2 分布来检验两个向量是否符合相同潜在分布.

以上的工作单纯从组件交互行为考虑,而组件

① OSSEC. OSSEC Manual. Available: http://www.ossec.net/doc/, 2008

间的交互行为会引起各组件资源使用的变化.对此，我们此前的工作[38]面向基于组件的软件，通过分析组件间的服务调用，来监测各组件资源使用，提出了一种基于线程染色的组件资源监测技术.由于线程是物理资源消耗的实体，线程的执行会占用 CPU 时间，而线程创建对象会占用内存空间.同时组件的交互会造成线程在组件间的转移，而组件的交互通常分为静态方法调用和动态服务交互两种，我们利用 AOP 对方法或服务进行接口注入，在线监测线程在组件间的转移和执行.进而将系统资源的使用划分到组件粒度，通过分析各组件资源使用的波动情况检测问题组件.

3.4.2　面向请求执行路径

面向请求执行路径的故障检测方法关注于“请求-响应”交互式系统处理请求的执行路径.CloudDiag 利用代码注入的方法监测各方法的请求响应时间以及方法的调用关系[39].监测数据格式如表 4 和表 5 所示，从中可以获取：(1)请求处理时间：获取相同主机(Host)，相同请求(RequestID)，标识为 Flag=Start/End 的记录，计算 Timestamp 的差值；(2)请求调用树：属于同一个请求(RequestID)的监测数据分为相同组，每类请求根据 CallerMID 和 CalleeMID 建立调用树.同类请求具有相同的调用树，延迟相近，当延迟的变化系数超出阈值，则检测为请求异常.对于每个异常分类，建立 $m\times n$ 矩阵 $\boldsymbol{M}$，m 为调用树的请求数量，n 为调用树中被调用方法的数量，M_{ij} 表示第 i 个请求调用第 j 个方法的执行时间，$\boldsymbol{M}(j)$ 为第 j 个方法的执行时间向量.CloudDiag 将辨别分类中的异常方法调用的问题抽象为，检测矩阵中噪音数据的过程，利用鲁棒主成分分析方法(Robust Principle Component Analysis，RPCA)检测异常方法调用.

表 4　方法执行时间统计表

Host	Timestamp	RequestID	MID	Method	Flag
Host 1	2012-01-01　16:31:31690272	169	739	ReadFile	Start(or End)

表 5　方法调用关系统计表

Host	Timestamp	RequestID	Caller MID	Callee MID
Host 1	2012-01-01　16:31:31690272	169	739	991

Pinpoint 将执行路径集合建模为概率上下文无关文法(Probabilistic Context-Free Grammar，PCFG)，语法符号表示提供服务组件，语法规则表示请求在组件间的转移概率，当实际监测与模型定义的转移概率出现的偏差大于定义的阈值，即认为请求处理路径上出现的故障[10].

Magpie 合并日志信息，而后采用事件字符串来表示序列化请求事件，以跟踪请求的执行路径，并使用资源使用信息对其进行标注[40].利用标准化的 Euclidean 距离来表示资源使用间的距离，将请求事件字符串划分为簇，当事件字符串不符合其所对应的簇则检测为故障.为了识别事件序列，将系统运行状态建模为概率状态机，每个状态间的转换抽象为一个事件，并具有一定的概率.

Spectroscope 跟踪组件内部或者跨组件的请求流，辨别请求流的处理时间变化，通过比较系统的不同版本，或不同时间段请求流的执行处理时间进行故障检测[41].该方法能够检测由于系统变化(如代码、配置)所造成的性能问题，但负载在不同时间段发生巨大变化造成资源竞争，所带来的性能问题就难以准确定位.

3.5　面向特定问题的故障检测

性能异常检测与云环境下的故障检测是近年来尤其受关注的问题，通常采用以上基于度量、日志和行为分析方法中的一种或结合几种方法来解决性能和云环境下所面对的问题，本节将对这两类特定问题进行分析.

3.5.1　性能异常检测

分布式软件系统的性能问题往往会影响用户体验，甚至导致客户流失、收益受损等严重后果，因此，面向性能表现的异常检测方法近年来得到了较多的关注.通常采用的方法是建立性能模型，以预测系统或其组件的一些性能属性，包括资源(如 CPU 和内存)利用率、响应时间、吞吐量等.性能模型能够应用在能力规划和资源供给等方面，同样可以用于检测系统性能故障[42].

在已知系统内部结构的情况下，可以建立软件体系结构，基于此分析系统性能，例如程序的执行时间是累加所有其调用的函数所花费时间的总和.这类研究大多面向某个具体领域，例如文献[43]面向数据流处理，文献[44]面向多组件集群系统，文献[45]面向 I/O 系统.这种方法直观，易于理解，但建立性

能模型需要知道底层细节(如资源消耗、内部算法和相关参数),而这样的信息通常难以获得.同时人工手动建立复杂系统的分析模型需要花费较多时间且操作复杂.

另一类方法利用数学模型对请求处理过程进行建模,对于多用户系统,最常见的是队列模型.队列模型将系统抽象为多个互相连接的队列集合,一个服务或资源关联一个队列,等待服务的请求在队列中排列.队列模型考虑到了目标系统在不同层次上的特点,可以用来建模各种元素(如磁盘、CPU 等)[46].服务器软件(如 Web 服务器)是相当常见的队列模型的建模对象[47].文献[48]利用队列模型建模多层应用,同时考虑到了用户会话、各层间的缓存等影响性能的要素.文献[49]考虑到了负载变化对性能的影响,关注于各组件处理时间的波动,利用异常签名衡量波动的严重程度以检测故障组件.这些模型可以用来调整配置参数(如最大进程、连接数量)以最大化性能,或提供一定层次的服务质量保障.然而性能模型只是考虑到了系统所监测的特定部分的性能,并且评估模型参数困难(如服务时间).

基于统计的方法近年来尤其受到关注.文献[19]建立分析引擎处理从标准操作系统接口搜集到系统层监测数据.由于特定度量与特定的组件、资源、进程和时间相关,引入分类器可以辨别与 SLA 冲突相关的度量集合.系统 SLA 状态描述为 $S=\{S^+,S^-\}$,其中,S^+ 为 SLA 符合,S^- 为 SLA 冲突;度量值向量为 $\boldsymbol{M}=[m_0,m_1,\cdots,m_n]$,其中,$m_i$ 为第 i 个度量值.作者将性能异常检测抽象为模式分类问题,通过引入分类函数 F 实现 $\boldsymbol{M}$ 到 S 的映射,分类器的输入为 $\boldsymbol{M}$ 而输出为 S.训练过程是监督式学习,训练集是观察到的日志$\langle \boldsymbol{M},S\rangle$集合,利用 F 可以判断在给定的度量值条件下,SLA 是否会出现冲突.模型可以表示为在给定的观察度量值下的条件概率分布 $\text{Prob}(S|\boldsymbol{M})$,利用分类器来评估是否 $P(S^+|\boldsymbol{M})>\text{Prob}(S^-|\boldsymbol{M})$.作者选择 Bayesian 网络作为分类器,优势在于可以利用模型来辨别度量向量中影响分类器选择的特定度量.与神经网络和支持向量机相比,贝叶斯网络具有较好的可理解性.而与决策树相比,贝叶斯网络可以将专家知识加入到模型中,例如用户可以指定度量间的关联性.由于训练数据噪音,Bayesian 网络会出现数据的过度拟合从而影响准确性,作者选择度量子集将 TAN(Tree Augment Naive Bayesian Network)作为分类器.

负载的波动以及部署环境的改变,会造成系统度量发生较大波动,从而导致模型改变,使得预测结果不准确.针对该问题,文献[20]综合多个模型,以适应环境的变化.在系统运行过程中,当已有模型并不能捕捉系统行为时,则引入新的模型以加强总体效果.作者利用特征选取的方法选择与建模最相关的度量,具有两方面优势,首先,抛弃了与 SLA 没有影响的度量,使得操作人员关注于一小部分的度量;另外,减少了需要推断模型的数据采样数量.方法利用滑动窗口搜集一段时间内的监测数据作为训练数据,基于此建立 TAN 模型,采用交叉验证的方法评估模型的平衡精度(Balance Accuracy,BA):

$$BA=0.5(P(S^-=F(\boldsymbol{M})|S^-)+P(S^+=F(\boldsymbol{M})|S^+)) \tag{9}$$

如果当前模型比已有模型都有显著提高,则加入新的模型,否则抛弃当前模型.这种方法是在系统处于正常状态的前提下,预测性能是否会发生 SLA 冲突.

文献[49]基于各组件的服务时间对于不同负载相对稳定的假设,建立组件服务时间与系统 CPU 时间的线性回归模型,以估算各组件的服务时间,服务时间波动较大的组件则检测为性能异常组件.该方法只关注 CPU 利用率,而事实上系统中具有众多的度量需要考虑,因而,其异常检测能力有限.也有工作利用 AOP 监测事务响应时间,利用 Pearson 系数和 DTW 算法建立事务数量与响应时间的关联性,并对其进行适应性更新,最后利用方差分析检测引起性能异常的组件[50].这种方法仅能分析单个事务与单个性能参数,而实际应用往往是多个事务的混合,而且需要关注多个性能属性;同时利用 AOP 监测事务与方法的响应时间会带来较大的开销,且监测的准确性受限于操作系统提供的接口.

以上方法通常需要分析应用的处理逻辑以选择监测注入点,其实际应用具有较大的局限性.我们此前的工作[51]提出一种基于关联分析的 Web 应用性能异常检测方法.该方法在运行过程中动态搜集负载与性能信息,利用核典型关联分析(Kernel Correlation Analysis,KCCA)的方法自动建立起负载与性能的关联模型,以刻画二者在正常状态下的关联性.而后,根据关联系数动态构造并更新 X-mR 控制图,通过监测关联系数的变化检测性能故障发生.该方法具有无需领域知识对系统进行建模,能够同时考察多个性能属性的优势.

3.5.2 云环境下的故障检测

软件可靠性技术可以分为故障避免、故障移除、

故障容忍和故障预测等四个方面. 故障避免是在开发阶段就考虑到故障发生的可能性，进行避错操作；故障移除是系统在运行过程中发生故障后消除出现的问题；故障容忍是系统在运行过程中发生故障的时候依然能够提供服务；故障预测是及早分析预测出系统可能出现的故障[52].

分布式特别是云计算环境对系统的可靠性有更高要求，通常采用故障容忍技术. 故障容忍可以分为主动和被动副本策略. 主动策略并行调用不同的副本来处理相同的请求，采用首先返回的响应作为最终结果，从而能够获得较高的性能. 而被动策略采用主要服务来处理请求，当其失效时调用另外的可选副本服务，可以分为以下三种策略：(1) 重试. 当原始服务调用失败则重复调用几次；(2) 恢复块. 按照设定的顺序执行多个副本；(3) N 版本编程. 编写多个实现相同功能的软件副本通过投票方式决定服务调用结果[53]. 这些策略各有利弊，静态同构的策略不能够适应动态的互联网环境，文献[54]提出了动态的故障容忍策略以适应运行环境的变化.

本文研究的故障检测是故障容忍的前提，检测到故障后才能够采用相应的故障容忍技术来处理故障. 面向虚拟化云计算环境，文献[55]结合异常预测技术提出一种性能异常预防系统. 该系统根据历史数据建立 Markov 模型以预测每个属性值以及 TAN(Tree-Augmented Naive Bayes)模型以检测可疑度量值. 文献[56]面向云计算环境下由多组件构成的分布式系统，首先定义组件的依赖关系，而后分析异常传播模式以分析各组件的异常程度. 文献[57]提出一种端到端的云计算系统故障管理框架，整合了反馈控制机制以实现云计算系统故障的自动化处理. 文献[58]面向云环境下的应用性能异常，辨别最能够反应应用性能的度量，通过检测这些度量的异常改变点以改进已有方法.

以上方法多是针对虚拟化环境，从底层物理资源的利用角度对系统状态进行分析，难以检测到具体问题组件. 我们此前的工作[59]针对云应用运行环境的动态性、类型的多样性和系统结构的复杂性，提出一种云应用故障检测框架，将外部负载变化、应用性能表现以及虚拟物理资源利用等因素引入到故障检测模型，建立外部负载与内部资源利用和性能表现的关联性，通过跟踪关联系数变化检测系统故障，通过分析资源使用变化定位系统故障.

3.6　对比分析

基于规则的方法由于事先已知故障及其表现，具有较高的准确性和及时性. 然而当故障此前未曾出现，或者故障表现难以刻画为规则，基于规则的方法就不能够识别，因此，该方法虽然查准率高，但查全率却较低. 同时云环境下应用类型多样、系统层次众多，大量度量需要监测分析，系统管理员难以根据经验人工制定规则.

基于度量分析的方法，不需要了解系统内部结构，通过调用操作系统提供的接口搜集监测数据，适用范围广. 其优势在于，无需事先知道故障类型并描述其特征. 然而由于网络环境的动态性与复杂性，建立具有鲁棒性和普适性的基准相当困难，基于异常检测的方法通常具有较高的误报率. 同时难以在代码层细粒度检测问题.

基于行为分析的方法，通过代码注入等方式搜集各组件行为，能够将故障定位到组件或代码片段，但需要了解应用的内部结构，且细粒度监测开销较高. 同时由于不同的应用的处理逻辑不同，需要注入不同的监测点，适应性较差.

基于日志分析的方法，通过分析日志信息可以了解一部分系统执行路径，故障检测的准确性取决于日志记录的数量和位置. 同时由于需要搜集大量的日志文件，从中抽取固定的模式，难以满足在线故障检测的需求.

如表 6 所示，本文根据在第 1 节提出的云计算环境下，分布式软件系统故障检测技术面临的在自动化分析、问题组件定位、领域知识要求、在线检测和适应环境能力等五个方面的技术挑战，对基于规则、度量、行为和日志等四类方法的优缺点进行了归纳总结. 其中，A 表示完全满足该方面的要求，C 表示完全不能满足，B 表示界于 A 和 C 之间部分满足.

表 6　故障检测方法优缺点对比

	自动化	问题定位	领域知识	在线检测	适应环境
规则	C	A	C	A	C
度量	A	C	A	A	B
行为	B	A	C	A	C
日志	B	B	B	C	B

4　未来工作展望

云计算环境巨大的部署规模、复杂的拓扑结构、动态的负载变化以及多样的应用类型给故障检测带来了新的问题. 本节针对第 1 节分析的云计算环境对分布式软件系统提出的挑战，给出了进一步的研究方向：

(1)自动化在线故障检测.云计算系统通常部署在大规模数据中心,成千上万的节点以及众多层次(如网络层、硬件层、虚拟机层、操作系统层、中间件层、应用软件层)的大量度量需要分析.特别是,白盒和灰盒技术需要注入应用或底层中间件平台,具有较大计算、内存、网络等开销,难以满足不影响系统运行、快速故障报警的在线检测需求.那么,如何以较小的计算开销,快速分析如此大规模的数据成为挑战.

减少在线故障检测开销可以从数据分析对象、数据分析频率、局部与全局分析相结合等三个方面考虑.首先选择具有代表性的度量进行分析,推断另外的度量值;而后分析系统运行状态,预测故障发生的可能性,据此动态调整数据分析频率;最后先在本地节点进行初步分析,检测简单的故障,如需进一步分析再汇总各节点信息,基于全局关联的方法分析系统状态.

(2)运行环境感知的故障检测.云计算平台是多承租的,多个虚拟机部署于相同物理主机.由于虚拟化技术对于缓存、磁盘、网络资源隔离的局限性,应用的性能会受到与其共存于同一物理主机的其他应用的影响.故障检测技术如何考虑多应用共享敏感资源因素造成的性能异常成为挑战.可以通过仿真的方式模拟应用间的性能干扰,据此分析能够反映性能干扰的关键度量,通过监测这些度量就可以分析性能干扰造成性能衰减.

另外,云计算系统处于高度动态的运行环境当中.云计算系统需要根据负载动态变化,通过虚拟机资源调整、虚拟机迁移和虚拟机克隆等技术按需自动获取和释放资源.部署环境的频繁改变使得问题定位需要利用加强学习等机器学习方法在线学习和演化状态模型,利用模式匹配的方法动态选择模型,在特定模型下进行故障检测,以适应不断变化的运行环境.

(3)面向组件交互行为的故障检测.云计算系统由众多组件构成,分布在不同节点,调用各种服务,组件间交互关系复杂,组件关联度高.复杂的交互行为使得组件相互影响,故障在组件间快速传播,某个组件的故障往往会造成多个组件出现异常,准确定位故障组件值得深入研究.定位故障组件,首先需要监测各组件间的调用关系,并且引入时间因素来建立故障传播模型.

另外,云计算平台上部署的应用通常对平台提供者和管理者是透明的,这就使得通过设计文档或代码注入对应用的软件体系结构进行建模分析变得困难.而黑盒方法只能够粗粒度分析异常的系统度量,不能定位到故障组件,这给细粒度(软件组件或代码段)故障检测带来了挑战.可以分析跟踪内核Trace,这样无需对源码进行修改,具有较小开销,进而结合和扩展异常检测方法,可以细粒度的检测故障组件.

5 总 结

故障检测技术成为保障分布式软件系统性能与可靠性的关键之一,近年来引起了工业界和学术界广泛关注,出现了较多研究成果并得到了应用.本文分析了云环境下面临的技术挑战,提出了基于统计监测的分布式软件系统故障管理参考框架,为此类系统的研究、设计与实现提供了参考借鉴.然后,对故障检测的方法进行分类,介绍了各类方法中有代表性的工作,并对比分析各类方法的优缺点.最后,针对当前云计算环境的特点,分析该技术所面临的问题与今后可能的研究方向.

参 考 文 献

[1] Patterson D A. A simple way to estimate the cost of downtime //Proceedings of the 16th USENIX Conference on System Administration. Berkeley, USA, 2002: 185-188

[2] Chen M Y, Accardi A, Kiciman E, et al. Path-based failure and evolution management//Proceedings of the 1st Symposium on Networked Systems Design and Implementation. Berkeley, USA, 2004: 23-36

[3] Oppenheimer D, Ganapathi A, Patterson D A. Why do internet services fail, and what can be done about it?// Proceedings of the 4th Symposium on Internet Technologies and Systems. Seattle, USA, 2003: 1-16

[4] Jiang G, Chen H, Yoshihira K. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. IEEE Transactions on Dependable and Secure Computing, 2006, 3(4): 312-326

[5] Jiang M, Munawar M A, Reidemeister T, Ward P A S. Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring. IEEE Transactions on Dependable and Secure Computing, 2011, 8(4): 510-522

[6] Trivedi K, Ciardo G, Dasarathy B, et al. Achieving and assuring high availability//Proceedings of the IEEE International Symposium on Parallel and Distributed Processing. Miami, USA, 2008: 1-7

[7] Fox A, Kiciman E, Patterson D A. Combining statistical monitoring and predictable recovery for self-management// Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems. Newport Beach, USA, 2004: 49-53

[8] Arefin A, Singh V K, Jiang G, et al. Diagnosing data center behavior flow by flow//Proceedings of the IEEE 33rd International Conference on Distributed Computing Systems. Philadelphia, USA, 2013: 11-20

[9] Chen H, Jiang G, Yoshihira K, Saxena A. Invariants based failure diagnosis in distributed computing systems//Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems. New Delhi, India, 2010: 160-166

[10] Kiciman E, Fox A. Detecting application-level failures in component-based Internet services. IEEE Transactions on Neural Networks, 2005, 16(5): 1027-1041

[11] Stehle E, Lynch K, Shevertalov M, et al. On the use of computational geometry to detect software faults at runtime //Proceedings of the 7th International Conference on Autonomic Computing. Washington, USA, 2010: 109-118

[12] Wang T, Zhang W, Wei J, Zhong H. Workload-aware online anomaly detection in enterprise applications with local outlier factor//Proceedings of the IEEE 36th Annual Computer Software and Applications Conference. Izmir, Turkey, 2012: 25-34

[13] Diao Y, Eskesen F, Froehlich S, et al. Generic on-line discovery of quantitative models for service level management// Proceedings of the IFIP/IEEE 8th International Symposium on Integrated Network Management. Colorado Springs, USA, 2003: 157-170

[14] Munawar M, Ward P S. Leveraging many simple statistical models to adaptively monitor software systems//Proceedings of the 5th International Symposium on Parallel and Distributed Processing and Applications. Niagara Falls, Canada, 2007: 457-470

[15] Jiang G, Chen H, Yoshihira K. Efficient and scalable algorithms for inferring likely invariants in distributed systems. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(11): 1508-1523

[16] Munawar M A, Ward P A S. A comparative study of pairwise regression techniques for problem determination//Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research. Toronto, Canada, 2007: 152-166

[17] Kang H, Chen H, Jiang G. PeerWatch: A fault detection and diagnosis tool for virtualized consolidation systems// Proceedings of the 7th International Conference on Autonomic Computing. Washington, USA, 2010: 119-128

[18] Zhen G, Jiang G, Chen H, Yoshihira K. Tracking probabilistic correlation of monitoring data for fault detection in complex systems//Proceedings of the International Conference on Dependable Systems and Networks. Pittsburgh, USA, 2006: 259-268

[19] Cohen I, Goldszmidt M, Kelly T, et al. Correlating instrumentation data to system states: A building block for automated diagnosis and control//Proceedings of the 6th Symposium on Operating Systems Design and Implementation. San Francisco, USA, 2004: 16-29

[20] Zhang S, Cohen I, Symons J, Fox A. Ensembles of models for automated diagnosis of system performance problems// Proceedings of the International Conference on Dependable Systems and Networks. Los Alamitos, USA, 2005: 644-653

[21] Bodic P, Friedman G, Biewald L, et al. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization//Proceedings of the 2nd International Conference on Autonomic Computing. Seattle, USA, 2005: 89-100

[22] Wang T, Wei J, Zhang W, et al. Workload-aware anomaly detection for Web applications. Journal of Systems and Software, 2014, 89(3): 19-32

[23] Wang T, Wei J, Zhang W, Zhong H. A framework for detecting anomalous services in OSGi-based applications// Proceedings of the IEEE 9th International Conference on Services Computing. Honolulu, USA, 2012: 250-257

[24] Wang T, Zhang W, Wei J, Zhong H. Fault detection for cloud computing systems with correlation analysis//Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management. Ottawa, Canada, 2015: 652-658

[25] Boulon J, Konwinski A, Qi R, et al. Chukwa: A large-scale monitoring system//Proceedings of the International Conference on Cloud Computing and Its Applications. Chicago, USA, 2008: 1-5

[26] Hellerstein J, Ma S, Perng C. Discovering actionable patterns in event data. IBM System Journal, 2002, 41(3): 475-493

[27] Sheng M, Hellerstein J L. Mining partially periodic event patterns with unknown periods//Proceedings of the 17th International Conference on Data Engineering. Heidelberg, Germany, 2001: 205-214

[28] Chinghway L, Singh N, Yajnik S. A log mining approach to failure analysis of enterprise telephony systems//Proceedings of the IEEE International Conference on Dependable Systems and Networks. Anchorage, USA, 2008: 398-403

[29] Fisher K, Walker D, Zhu K Q, White P. From dirt to shovels: Fully automatic tool generation from ad hoc data// Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. San Francisco, USA, 2008: 421-434

[30] Makanju A, Heywood A, Milios E. Clustering event logs using iterative partitioning//Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Paris, France, 2009: 1255-1264

[31] Fu Q, Lou J G, Wang Y, Li J. Execution anomaly detection in distributed systems through unstructured log analysis// Proceedings of the 9th IEEE International Conference on Data Mining. Miami, USA, 2009: 149-158

[32] Prewett J E. Analyzing cluster log files using logsurfer// Proceedings of the Annual Conference on Linux Clusters. San Francisco, USA, 2003: 169-176

[33] Stephen E H, Todd E. Automated system monitoring and notification with Swatch//Proceedings of the 7th Conference on System Administration. Monterey, USA, 1993: 101-108

[34] Oliner A, Stearley J. What supercomputers say: A study of five system logs//Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Edinburgh, Scotland, 2007: 575-584

[35] Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. Big Sky, USA, 2009: 117-132

[36] Fonseca R, Porter G, Katz R H, et al. X-trace: A pervasive network tracing framework//Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation. Cambridge, USA, 2007: 20-33

[37] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming//Proceedings of the 15th European Conference on Object-Oriented Programming. Jyväskylä, Finland, 1997: 220-242

[38] Wang T, Zhang W, Wei J, Zhong H. Online anomaly detection for components in OSGi-based software//Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering. San Francisco Bay, USA, 2012: 188-193

[39] Mi H, Wang H, Zhou Y, et al. Towards fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1245-1255

[40] Barham P, Donnelly A, Isaacs R, Mortier R. Using magpie for request extraction and workload modelling//Proceedings of the 6th International Symposium on Opearting Systems Design and Implementation. California, USA, 2004: 18-31

[41] Sambasivan R, Zheng A X, Rosa M D, et al. Diagnosing performance changes by comparing request flows//Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. Boston, USA, 2011: 4-17

[42] Wang C, Kavulya S P, Tan J, et al. Performance troubleshooting in data centers: An annotated bibliography? SIGOPS Operation System Review, 2013, 47(3): 50-62

[43] Uysal M, Kurc T, Sussman A, Saltz J. A performance prediction framework for data intensive applications on large scale parallel machines//Proceedings of the 4th International Workshop. Pittsburgh, USA, 1998: 243-258

[44] Stewart C, Shen K. Performance modeling and system management for multi-component online services//Proceedings of the 2nd Symposium on Networked Systems Design and Implementation. Washington, USA, 2005: 71-84

[45] Shen K, Zhong M, Li C. I/O system performance debugging using model-driven anomaly characterization//Proceedings of the 4th USENIX Conference on File and Storage Technologies. San Francisco, USA, 2005: 23-36

[46] Dilley J, Friedrich R, Jin T, Rolia J. Measurement tools and modeling techniques for evaluating web server performance//Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools. St. Malo, France, 1997: 155-168

[47] Urgaonkar B, Pacifici G, Shenoy P, et al. An analytical model for multi-tier internet services and its applications//Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Canada, 2005: 291-302

[48] Lu Y, Abdelzaher T, Lu C, et al. Feedback control with queuing-theoretic prediction for relative delay guarantees in Web servers//Proceedings of the 9th International Symposium on Real-Time and Embedded Technology and Applications. Toronto, Canada, 2003: 208-217

[49] Cherkasova L, Ozonat K, Mi N, et al. Automated anomaly detection and performance modeling of enterprise applications. ACM Transactions on Computer Systems, 2009, 27(3): 1-32

[50] Magalhes J P, Silva L M. SHoWA: A self-healing framework for Web-based applications. ACM Transactions on Autonomous and Adaptive Systems, 2015, 10(1): 1-28

[51] Wang T, Wei J, Qin F, et al. Detecting performance anomaly with correlation analysis for Internetware. Science China Information Sciences, 2013, 56(8): 1-15

[52] Jhawar R, Piuri V, Santambrogio M. Fault tolerance management in cloud computing: A system-level perspective. IEEE Systems Journal, 2013, 7(2): 288-297

[53] Zheng Z, Lyu M, Wang H. Service fault tolerance for highly reliable service-oriented systems: An overview. Science China Information Sciences, 2015, 58(5): 1-12

[54] Zheng Z, Lyu M. Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints. IEEE Transactions on Computers, 2015, 64(1): 219-232

[55] Tan Y, Nguyen H, Gu X, et al. PREPARE: Predictive performance anomaly prevention for virtualized cloud systems//Proceedings of the 32nd International Conference on Distributed Computing Systems. Macau, China, 2012: 285-294

[56] Nguyen H, Shen Z, Tan Y, Gu X. FChain: Toward black-box online fault localization for cloud systems//Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems. Philadelphia, USA, 2013: 21-30

[57] Sharma B, Jayachandran P, Verma A, Das C. CloudPD: Problem determination and diagnosis in shared dynamic clouds//Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Budapest, Hungary, 2013: 1-12

[58] Xiong P, Pu C, Zhu X, Griffith R. vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments//Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. Prague, Czech, 2013: 271-282

[59] Wang T, Zhang W, Ye C, et al. FD4C: Automatic fault diagnosis framework for web applications in cloud computing. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016, 46(1): 61-75

WANG Tao, born in 1982, Ph. D., assistant professor. His research interests include fault diagnosis, software reliability, autonomic computing, cloud computing.

ZHANG Wen-Bo, born in 1976, Ph. D., professor, Ph. D. supervisor. His research interests include distributed computing, software engineering.

XU Ji-Wei, born in 1985, Ph.D. candidate. His research interests include software engineering and distributed computing.

WEI Jun, born in 1970, Ph. D., professor, Ph. D. supervisor. His research interests include distributed computing and software engineering.

ZHONG Hua, born in 1971, Ph. D., professor, Ph. D. supervisor. His research interests include software engineering and distributed computing.

Background

Nowadays, more and more applications (e. g., Netflix, Yelp, Pinterest) are deployed on public cloud computing platforms (e. g., Amazon EC2) to provide services. Due to the complexity, dynamism and openness of cloud computing, the development and deployment of services are prone to many types of faults. These faults may lead to service failures that affect a large population of users and result in serious economic loss. For example, Amazon. comwas down for almost 45 minutes in August 2013 due to an unexpected fault, causing an economic loss of five million dollars. Detecting faults accounts for 75 percent of failure recovery time, and detecting faults in time could prevent 65 percent of failures, according to the report of Tellme Networks. Thus, fault diagnosis is essential to guarantee the reliability and performance of Internet-based services. However, the faults inside services (e. g., resource contention, configuration faults, software faults, hardware failures) are usually triggered by complex factors in the deployment environment at runtime. This makes it difficult to reproduce the faults (e. g., deadlock caused by concurrency, fault transmission between components). Therefore, debugging and testing services cannot effectively eliminate the inevitable faults triggered in specific contexts.

Fault diagnosis technologies have widely attracted the attention of industrial and academic communities in recent years. For example, commercial monitoring tools (e. g., IBM Tivoli, HP OpenView, Amazon CloudWatch) allow system operators to set rules for some particular system metrics manually. Then they raise alerts automatically when the value of a metric exceeds the pre-defined threshold. However, setting suitable thresholds for thousands of metrics is difficult. Similarly, by modeling the status of distributed systems during their normal operation periods, traditional fault diagnosis methods can detect faults when the monitored status deviates from the built model. However, existing solutions are difficult to detect the faults of distributed services deployed in a large-scale dynamic cloud computing environment.

Fault detection and diagnosis has become one of the most important technologies for guaranteeing the performance and reliability of distributed services, which has been extensively studied and widely used in various fields. This paper proposes a reference framework of fault detection and diagnosis with statistical monitoring for service systems. After categorizing existing works, it analyzes the main ideas and limitations of them, and introduces typical works for each category. Finally, it introduces new challenges and future works.

This work was supported in part by the National Natural Science Foundation of China under Project 61402450, the Beijing Natural Science Foundation under Project 4154088, the National Key Technology R&D Program of China under Project 2015BAH55F02-4, the National Key Basic Research and Department (973) Program of China under Project 2015CB352201, and the National High Technology Research and Development Program (863 Program) under Project 2013AA041301.