分类号：TP311.5　　单位代码：10335

密　级：无　　学　　号：21651072

浙江大学

硕士学位论文

中文论文题目：容器云平台跨层次异常检测系统研究

英文论文题目：Research on Cross-level Anomaly Detection System of Container Cloud Platform

申请人姓名：卢澄志

指导教师：贝毅君 副研究员

合作导师：

专业学位类别：工程硕士

专业学位领域：软件工程

所在学院：软件学院

论文提交日期 2018 年 8 月 13 日

容器云平台跨层次异常检测系统研究

论文作者签名:_____

指导教师签名:_____

论文评阅人 1: _____

评阅人 2: _____

评阅人 3: _____

评阅人 4: _____

评阅人 5: _____

答辩委员会主席: _____

委员 1: _____

委员 2: _____

委员 3: _____

委员 4: _____

委员 5: _____

答辩日期: _____

Research on Cross-level Anomaly Detection System of Container Cloud Platform

Author's signature:_____

Supervisor's signature:_____

Thesis reviewer 1: _____

Thesis reviewer 2: _____

Thesis reviewer 3: _____

Thesis reviewer 4: _____

Thesis reviewer 5: _____

Chair: _____
(Committee of oral defence)

Committeeman 1: _____

Committeeman 2: _____

Committeeman 3: _____

Committeeman 4: _____

Committeeman 5: _____

Date of oral defence: _____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 浙江大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名: 签字日期: 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名: 导师签名:

签字日期: 年 月 日 签字日期: 年 月 日

摘要

容器技术因其轻量化的特点自诞生以来就受到工业界和学术界的广泛关注。随着云计算的不断发展，越来越多的企业选择使用容器技术作为云服务的主要虚拟化技术。但是由于容器的隔离性相较于传统虚拟机而言稍显薄弱，因此运行在容器中的应用程序更加容易受到干扰。同时容器的轻量化特点导致了集群的规模非常庞大。面对这种情况，传统的异常行为检测算法不能适用于基于容器技术的云计算环境。

本文针对基于容器技术的 Web 应用设计了一套跨层次异常检测定位系统。系统通过分析各类性能数据指标对异常进行检测，并结合图论、贝叶斯网络等方法对异常源进行定位。

为了评估异常检测定位系统的效率，本文搭建了模拟容器云环境的平台，通过注入几种典型的异常来模拟实际使用中云计算系统可能出现的异常。利用收集到的性能数据分别构造异常检测模型与异常定位模型。最后使用测试数据对系统进行整体的评估与分析。经过分析，该系统能够有效的检测出典型的异常并定位出异常的根源。

关键词： 容器技术,跨层次,异常检测,异常定位,贝叶斯网络,图论

Abstract

Due to its lightweight nature, container technology has received widespread attention from industry and academia since its inception. With the continuous development of cloud computing, more and more enterprises choose to use container technology as the main virtualization technology of cloud services. However, because the isolation of the container is slightly weaker than the traditional virtual technology, applications running in the container are more susceptible to be interfered. At the same time, the lightweight nature of the container has led to the large scale of cloud system. Faced with this situation, the traditional fault behavior diagnosis algorithm cannot be applied to the cloud computing environment based on container technology.

This paper designs a cross-level anomaly detection and location system for web applications based on container technology. The system detects abnormalities by analyzing various performance data indicators, and then locates abnormal root cause combined with graph theory and Bayesian network.

To evaluate the efficiency of the anomaly detection system, we built a platform for simulating container cloud environment and injected several typical anomalies to simulate the abnormalities that may occur in actual cloud computing systems. The anomaly detection model and the anomaly location model are respectively constructed by the collected performance data. After analysis, the system can effectively detect typical anomalies and locate the root cause of anomalies.

Key Words: container technology, anomaly detection, anomaly location, Bayesian network, graph theory

目录

图目录

表目录

第1章 绪论

1.1 问题的提出

云计算（Cloud Computing）是利用互联网提供动态可扩展资源的一种服务模式。云计算提供的服务可以是基础的平台服务，也可以是完整的软件、应用服务。云计算的这种服务方式降低了个人用户和企业用户用于购买和维护服务器的成本，并且使得资源能够更加集中的进行管理。

作为一种新型服务提供方式，云计算获得了产业界和学术界的青睐，并且已经取得了许多的研究成果。国内外许多的互联网公司都推出了自己的云计算服务，例如国内的阿里巴巴的阿里云[1]，腾讯的腾讯云等，国外的有亚马逊的 AWS[2]，微软的 Azure[3]，谷歌的 Google CLOUD[4]等。学术界云计算领域的专家学者也积极与产业界的各个公司合作，开发出了多种开源的云计算平台以及多种云计算相关的软件，利用开源共享的优势促进了云计算的快速发展。例如，Apache 基金会对 Google 的 MapReduce 的开源实现 Hadoop[5][6][7]，针对大规模数据处理而设计的快速通用的计算引擎 Spark[8]等。得益于产业界和学术界的努力，云计算已经逐渐渗透到每个人的生活之中，对人们的生活产生了深远的影响。

然而，随着云计算的蓬勃发展和广泛应用，云计算的稳定性和安全性问题成为阻碍云计算进一步发展的重要原因之一。近年来，各大云平台的事故频频发生。2016 年 4 月，苹果的互联网服务发生长达数个小时的停机事故，造成大量的用户无法使用 iCloud 服务。2017 年 2 月，亚马逊位于弗吉尼亚州的数据中心的云存储系统出现异常，导致服务中断 4 小时，导致美国各个领域的亚马逊云用户出现不同程度的服务中断。随着越来越多的传统业务系统被部署到云平台上，如何提高云平台自身的稳定性成为了一个巨大的挑战。

当前，为了提高云计算平台的资源利用效率并且方便提供定制化的服务同时提高应用程序部署和销毁的效率，资源虚拟化技术成为各个云服务提供商所青睐的技术。随着类似 Xen，KVM，LXC 等虚拟机技术的成熟，云服务提供商能够提供各种各样的定制化服务，用户也开始逐步的将自己的应用程序部署到云计算平台中。应用程序部署速度越来越快造成了云计算系统的不断扩大。规模不断扩大的云计算系统已经变得非常复杂，主要体现在以下几个方面:

1. 规模巨大。一个传统的云数据中心拥有上万台服务器、交换机以及上千台路由器。如此庞大的节点数也就意味着异常的情况极其普遍。

2. 部署的程序结构复杂。由于应用部署的便捷性，拥有复杂交互模式的应用程序例如搜索引擎，电子商务等能够快速的部署在云平台上。这些应用往往都具有极其复杂的内部结构，一次页面请求可能会涉及到成百上千组件之间的协同工作[9]，任何一个中间组件的崩溃都会导致整个应用程序的服务不同程度的中断。

资源共享。云计算的一大特征就是资源共享。谷歌的云数据中心的一台服务器上共享的应用程序可达 5 到 18 个，平均每台服务器支持 10.69 个不同的虚拟机程序[10]，阿里云的一台服务器上多种应用同时运行[11]。这些虚拟机运行的过程中会产生资源竞争，对每个虚拟机内的程序造成影响，进而影响服务的质量。

由于云计算平台的复杂的内部系统架构以及资源共享，导致云计算系统相比于传统的服务器更容易产生异常，可以说异常也是云计算平台的一种常态[12]

想要解决云计算系统的异常问题，首要的便是明确异常的种类。异常是指系统因为某种原因无法以在预定的时间内达到正确的结果。其次是明确异常的来源。一般来说，云计算系统的异常来源包括以下几个：1. 资源竞争。由于使用了资源虚拟化技术，云计算系统中单个服务器可以同时运行多个操作系统，多个应用程序，每个应用程序之间会不可避免的产生资源竞争。2. 资源瓶颈。虽然云计算系统可以为用户提供海量的硬件资源，但是由于每个服务器提供的资源量不可能无限，云计算系统可能无法满足单个组件的极大的资源需求，从而造成组件服务阻塞。3. 硬件异常。云计算系统由数量众多的机器组成，硬件异常极其普遍，是异常的主要来源之一。4. 应用程序自身缺陷。5. 外部攻击。其中硬件异常，应用程序自身缺陷这类异常往往直接导致系统崩溃。这些异常通常无法通过软件手段去检测消除，只能等服务中断以后通过手动的重新配置参数，重启或更换硬件等手段来解决。但是由于资源竞争和资源瓶颈所导致的云计算系统异常则可以通过软件检测到，并在服务中断之前将其消除，保证服务能够相对持续的运行。所以由于共享资源产生的资源竞争和相互干扰成为了学者们研究的重点。

本文中检测的异常为不当的资源竞争或者相对轻微的硬件问题产生的异常。除了能够检测出异常外，还要考虑到云计算系统的实时性。云计算系统往往在同一台服务器上运行多种应用程序以提高集群的利用率。除了常规的计算应用，存储应用，还有服务类应用。而服务类应用往往强调实时性。因此，对于云计算系统的异常检测除了准确性之外还应该考虑到实时性，保证服务类应用持续稳定的运行。

1.2 国内外研究现状

有别于传统的服务器的是，云计算系统规模更大，程序结构更复杂，共享特性更加显著。因此针对云计算系统的异常检测不仅要保证异常检测的准确率和实时性，同时要快速确定异常发生的位置，以便于及时排除异常，避免影响到其他的用户的应用程序的运行。

1.2.1 异常检测方法的研究现状

国内外众多专家学者提出了不同的方法来解决云计算系统的异常检测和定位的问题。这些方法主要分为两大类，一类是基于日志的异常检测方法。通过分析云计算系统收集的日志信息来检测异常发生的位置，时间以及原因。第二类则是基于系统实时性能数据的异常检测方法。通过分析系统中实时性能数据来构造异常检测模型，利用异常检测模型对系统进行异常状态检测。

1. 基于日志的异常检测方法研究现状

日志系统记录系统运行过程中的硬件，软件和操作系统的信息，同时监视整个系统中发生的事件，并将这些信息通过文件形式存储起来。

美国北卡罗莱纳州立大学的 Kc 和 Gu 等人设计了一套基于日志的异常检测系统用以进行大规模云平台的异常[13]。该系统通过采用粗细粒度日志特征相结合的混合日志挖掘方法在保证检测精度的同时降低检测的开销。而且该系统能够自动提取日志信息，并且将日志信息按照归纳所得的定式进行判断，简化了系统管理员分析日志，查找异常的任务。

Nagaraj 等人提出了一个用于支持开发人员调查分布式系统中的性能问题的自动化工具 DISTALYZER[14]。该工具使用机器学习技术比较从大型系统提供的海量的日志数据中提取的系统行为，建立系统组件和性能之间的关联模型，推断两者之间的最强关联。

Ding 等人提出了一种成本感知的日志记录机制 Log2[15]。Log2 在给定预算（定义为在固定时间间隔内允许的日志输出最大量）的情况下通过两个阶段的过滤对日志做“记录与否”的决策。阶段一有效的剔除了大量不相关的日志，减少无关日志对后续步骤的干扰，阶段二将符合预算的有效日志缓存并输出，作为异常检测的数据来源。

国内华中科技大学的 Zou 等人设计了一套异常分析和诊断系统[16]，用于管理系统内部不同组件生成的日志，监控系统实时状态，检测系统内部是否出现了异常。该系统能够过滤噪音，提取用于分类的日志模板信息并过滤模板信息，通过

结合保存人工分类的关键词判断日志中的异常类型。

基于系统日志的异常检测方法的确能够精确的检测出系统异常，但是通过日志检测的方法存在延迟，系统往往只能通过异常发生后的日志来判断异常，管理员接收到的异常报告通常只能协助管理员找到异常发生的位置，排除异常。

2. 基于系统性能的异常判断的异常检测方法研究现状

美国佐治亚理工学院的 Wang 等人提出基于数据熵值的异常测试方法 EbAT 进行异常检测[17]。该方法采集云平台的性能指标数据并分析其分布特征，根据采集到的样本数据形成描述该分布的熵时间序列，最后利用数据分析法判别熵的正常和异常实现性能指标数据异常分布的变化的检测。

Sharma 等人提出云系统的异常管理框架 CloudPD[18]，利用操作环境的规范化表示来量化表示共享的影响，同时使用在线学习过程和基于相关性的性能模型来解决动态性和检测精度的问题。框架采用集成的端到端反馈回路来协同云来管理生态系统。

Pannu 等人提出了一种基于机制的自我进化异常检测框架检测云平台是否处于异常状态[19]。该框架采集云系统性能数据，对每条性能数据计算异常分，判断分数是否高于某个阈值。如果高于阈值，则向平台管理节点发送异常信息，待系统管理员处理完异常后，根据标记的正常和异常状态，框架对标记好的数据进行学习，实现框架的自我完善，提高检测器的检测性能。

Dean 等人针对 IaaS 平台设计了一套无监督行为学习系统，用于预测平台的性能异常[20]。该系统采用自组织映射的方法训练得出平台正常行为模式，之后将待检测的行为模式与训练模型得到的正常行为模式进行对比，判断性能是否出现异常。

国内林铭炜等人设计了一种虚拟机异常行为检测框架，该框架利用监控代理组件采集数据，利用异常件检测组件进行异常判断，同时利用混合推策略模型解决数据传输问题[21]。该框架对数据进行聚集并结合增量局部异常因子算法对虚拟机性能进行分析从而判断虚拟机的异常状态。

这类基于性能数据的异常检测算法通过发现性能的异常提早规避系统故障的发生。相较于基于日志的异常检测方法，基于系统性能数据的异常检测算法能够实时的监控系统状态，但是由于他们都只对某一层的性能数据进行分析，而忽略了现今云计算系统中普遍使用的多层次的结构产生的数据，更没有对于多层次之间的相关性进行协同分析，因而导致异常检测精度低，误报的情况时有发生。

1.2.2 异常定位方法研究现状

异常定位的方法基本分成两个步骤，首先就是理清云计算系统内部复杂的依赖关系。将云计算系统的复杂结构抽象化，形成结构图，利用图论的方法进行研究。其次便是对异常进行定位，找到异常的位置并在结构图中显示出来。

1. 依赖关系图构建

Apte 等人提出了 LWT 方法集用于判断虚拟机之间的依赖关系[22]。LWT 方法集分成三个阶段：监控，建模和聚类，主要针对的是。监控阶段主要是收集虚拟机的 CPU 使用率信息，然后为每个虚拟机构造一个自回归模型。最后利用机器学习算法对构造出的模型进行聚类，得到虚拟机之间的依赖关系。

Natarajan 等人设计了一种可以自动探测网络服务依赖的工具，NSDMiner[23]。NSDMiner 通过分析收集到的网络流量，获取服务之间的依赖关系。

Google 开发了一套 Dapper 的请求追踪框架[24]。该框架通过修改应用程序架构中通用关键代码库使得系统在请求处理过程中能够保存请求的相关信息，之后能够收集请求的执行状态。将请求的执行状态与组件的状态日志结合起来，便能够得到各个组件之间的依赖关系。

微软也推出了自己的事件跟踪系统 Magpie[25]。该系统采用 Windows 系统自带的事件跟踪器 ETW 捕获内核层的相关事件信息，以此根据发生的事件所属的任务 ID 进行整合和归类。

可以看到，大多数的依赖关系构建都是基于网络流量进行分析构造。基于流量分类方法的主要问题在于云计算系统内部组件过多，组件之间通信频繁，流量数据庞大，如何筛选，存储和分析这些流量数据成为了一大挑战。

2. 异常定位研究现状

常用的程序执行跟踪方法主要有基于程序插桩的方法和基于采样分析的方法。

Attariyan 等人提出性能摘要工具 X-ray，一种自动诊断异常根源的技术[26]。通过在应用程序执行时对可执行文件进行插桩获取程序执行过程中的动态信息流，进而推测出异常位置。

Nguyen 等人提出在线黑盒异常定位系统定位异常组件[27]。该系统利用虚拟机性能指标构造性能指标的正常波动模型，判断异常变化的数据点。同时结合发生变化的数据点的时间信息和组件之间的依赖关系定位异常组件。

1.3 论文研究目标与章节安排

目前大多数的异常检测研究都着眼于采用传统虚拟化技术的云平台进行检测，

而很少有对采用容器技术的云平台进行异常检测。云计算系统具有复杂的内部结构，系统层次复杂，涉及到的硬件，虚拟机管理程序，操作系统，应用程序可达上千个，每个应用程序的内部结构也相当复杂，一个典型的 Web 程序涉及上百个关联模块。采用容器技术的云计算系统，由于容器的部署速度快，启动快的特点，使得虚拟机的创建、启动和迁移相较于其他虚拟化技术的云计算系统更加富有动态性。云系统的应用程序之间也往往需要多个不同的部件相互协作才能完成。大量的请求进入平台内部，触发整个云平台内部的各个组件的运行，具有显著的跨节点、跨层次的特征。传统异常检测方法缺少层次间的关联信息，无法准确反映异常传播特性，使得其在基于容器技术的云系统上检测准确率不高，误报率高。

因此，本文针对采用容器技术的云平台提出了一种跨层次的异常检测系统，利用贝叶斯网络与图论的相关知识对异常发生的源头进行定位。

本论文分为六章，各章节内容安排如下:

第一章: 绪论。介绍云计算系统中异常检测的背景，国内外研究现状。

第二章: 相关技术介绍。对容器云平台的结构，异常类型以及异常检测技术进行分析。

第三章: 容器云跨层次异常检测系统的框架设计。根据基于容器技术的云平台的特点对容器云异常检测系统的框架进行总体描述。

第四章: 容器云跨层次异常检测系统的关键技术设计与实现。对容器云平台异常检测系统的关键技术如异常检测，异常定位，数据收集等进行详细分析。

第五章: 容器云平台跨层次异常检测系统测试与分析。搭建容器云实验环境，构建容器云异常检测系统，通过异常注入的方式对系统进行测试并展示异常检测结果。

第六章: 总结与展望。对本论文的研究内容进行总结，同时对今后的工作进行展望。

第2章 相关技术介绍

2.1 容器云技术介绍

云计算系统的实质就是一个远端的大型分布式系统。按照 NIST 对于云计算平台的定义，经典云计算架构主要分成三层，如图 2.1 所示，包括 IaaS（Infrastructure as a Service，基础设施即服务），PaaS（Platform as a Service，平台即服务）以及 SaaS（Software as a Service，软件即服务）。

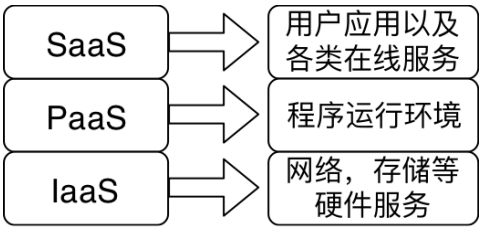

图 2.1 经典云计算架构示意图

IaaS 层为用户提供基本的计算，网络，存储以及其他的服务器基础资源，用户可以在上面任意部署和运行操作系统和软件，而基础资源的管理则由云计算服务提供商负责处理，大大降低了用户维护服务器基础设施的成本。PaaS 层为用户提供了支撑应用运行的运行时环境、相关工具和服务。用户无需再为配置环境而烦恼，从而可以专注于开发应用的核心业务。SaaS 层为用户提供了完整的软件服务，用户只需要从浏览器或者客户端等使用部署在云计算系统上的应用，无需关心软件的开发维护等工作。

随着技术的不断更新，应用规模也越来越大，应用逻辑越来越复杂，与此同时，用户对于快速部署和更新应用的需求越来越强烈。但是传统的 IaaS 的最小粒度的资源调度单位以虚拟机为主。虽然虚拟机具有良好的隔离性，但是虚拟机本身过于庞大，因此导致了资源利用率低，调度缓慢等问题。PaaS 建立在 IaaS 的基础上，通过容器技术提高资源利用率，但是同样的，PaaS 对于应用架构的选择以及软件环境等受到 IaaS 的较大限制。因此应用和平台之间存在强耦合，给应用迁移造成巨大的麻烦，运维人员控制力显著下降。人们需要一个真正可行的解决方案。

其中一个解决方案就是采用容器技术虚拟化代替传统的 hypervisor 虚拟机。如图 2.2 所示，容器技术虚拟化直接将虚拟化层建立在操作系统上，通过共享的操作系统内核来支持应用的运行，同时使用操作系统的进程隔离机制保证不同的容器之间的隔离。容器技术虚拟化直接利用了宿主机的内核，减少了抽象层，更加轻量化，启动更快。相比于传统的虚拟化技术，容器虚拟化技术具有以下几个优

点：1.跨平台支持。2.高资源利用率。3.精确的资源隔离。4.便捷的版本控制。5.毫秒级的启动速度。

容器云是使用容器虚拟化技术建立的云计算系统。容器云的资源分割调度的基本单位是容器，通过容器封装软件的运行时环境，为开发者和系统管理员提供用于构建、发布和运行分布式应用的平台[28]。容器云既可以作为 IaaS 为用户提供服务，也可以作为 PaaS 为用户提供服务。当用户更加关注容器之间的资源共享与隔离，容器编排方式以及底层网络通信等时，容器云可以看作是传统的 IaaS；当用户更加注重应用程序的运行环境与应用支撑时，容器云可以看作是 PaaS。

当前容器云所使用的技术包括基于 Linux container 的 Docker，基于 rocket 容器的 CoreOS 等。Docker 作为其中发展最快，使用范围最广的技术已经成为云计算领域研究的重点。

Docker 是一个开源软件项目，提供一个额外的软件抽象层，让应用程序可以便捷的部署在容器上，同时对操作系统虚拟化进行自动化管理[28]。相较于传统的虚拟化技术，Docker 不对硬件进行虚拟化，也不会为每个虚拟机创建单独的内核，而是共享宿主机的内核，容器仅仅只是运行在宿主机上的一个特殊的进程。Docker 的核心技术包括：1）Linux namespace。Docker 通过 Linux 内核的不同类型的 namespace 将不同容器的用户空间、网络、消息、文件系统、UTS（UNIX Time-sharing System）、进程等隔离开来。2）Control Groups。Docker 采用 Control Groups（cgroups）对每个容器的资源进行配额和度量。通过 cgroups，Docker 可以精确的为每个容器分配 CPU 数量，CPU 占用率，内存大小，磁盘 I/O，网络，设备以及命名空间。3）AUFS。AUFS 是一种 Union File System，是一种支持联合挂载的文件系统，即将不同目录挂载到同一个目录下。AUFS 将目录进行分层，最上层是读写层，最下层是只读层。当用户需要读文件，AUFS 将会从最上层开始寻找，如果找到文件，就打开文件。如果没找到，便根据层与层的关系到下一层中寻找。当用户需要写文件，如果文件不存在，AUFS 将会在读写层创建一个新的文件，如果存在，AUFS 将会一层一层的寻找，找到后就在读写层复制一个相同的文件进行修改。通过 AUFS，Docker 保证了 Linux 内核不会被恶意的进行修改。

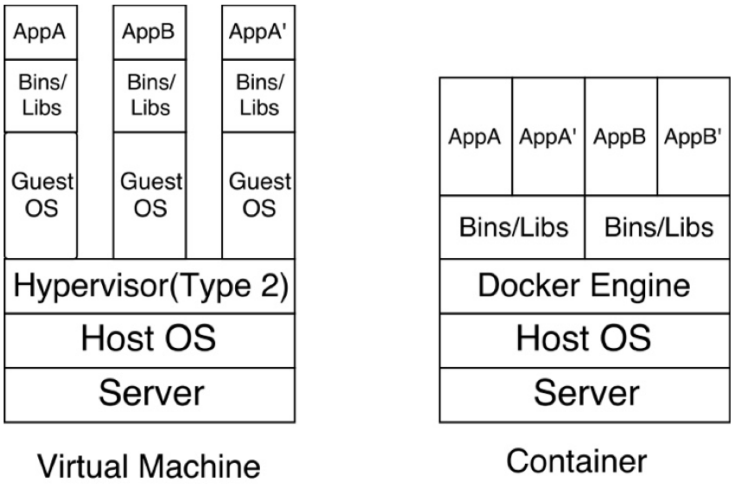

图 2.2 传统虚拟机结构和基于容器的虚拟机结构对比

2.2 异常检测技术介绍

2.2.1 容器云平台异常类型

基于容器的云计算系统异常通常分成三类。

第一类是硬件异常，是由于服务器的长时间工作而导致的硬件损坏。这类的异常一般是由服务器主板进行检测，由服务器自带的异常诊断面板或者主板上自带的蜂鸣器提示异常类型，再由服务器的运维人员负责处理异常。

第二类是网络入侵导致的异常。这类异常来源于他人恶意的对系统的资源完整性、保密性和可用性进行攻击。检测这类异常的关键就是检测入侵活动。入侵通常分成三种，即系统外入侵，系统内相关人员渗入以及滥用职权。目前入侵检测技术主要有两种：异常行为检测和滥用检测[29]。滥用检测是对已知攻击进行特征描述和模式归纳，并将这些特征进行编码存入知识库内。当有事件发生的时候，将事件的特征编码与数据库内存储的特征编码进行比较，如果超过一定的阈值，则认为是入侵行为。这种技术能够高效的识别已知类别的入侵活动，但是对于未知的入侵活动或者是存在有较大差异的已知类型的入侵活动的变种检测识别成功率低，需要不断的更新知识库。异常行为检测通过实时监测检测对象的行为，将实时的行为与知识库内保存的检测对象的正常行为进行比对，若发现实时行为相比于正常行为有较大幅度的改变，可以认为是遭到了入侵。

第三类是不同容器之间的相互影响的异常。这类异常在基于容器的云计算系统上非常普遍。由于容器的轻量化特征，部署在同一台服务器上的容器数量众多，但是容器并不具备传统虚拟机的良好的隔离性，因此，基于容器技术的云计算系

统容器间的资源竞争会比较明显。对于容器之间相互影响造成的异常。检测和避免这类异常的一个关键技术就是对容器的运行状态进行实时的监控。目前检测容器运行状态的方案主要有两种，基于日志的检测方案和基于实时性能数据的检测方案。

2.2.2 容器云平台异常检测问题组成

对于基于容器技术的云计算系统进行的异常检测，通常可以归纳成对于容器的异常状态检测。通过检测容器的异常状态对容器进行及时的暂停，迁移以及诊断等一系列的措施。异常检测问题通常由五个部分组成，分别是输入数据，异常类型，数据标签，异常检测方法以及异常结果输出。

1. 输入数据

如何选择输入数据是异常检测的关键问题。输入数据通过一组属性描述。数据的属性可以是离散的，也可以是连续的。数据属性的数量和类型由数据本身和数据的使用者共同决定的。输入数据的特征则决定了检测技术的选择。例如，对于离散的输入数据，基于最近邻的方法就可以使用；对于连续的输入数据，最近邻的方法就不合适。

通常而言，输入数据是相关的。例如在容器云中，两个容器之间的性能数据往往是互相影响的，同一台服务器的虚拟层和物理层之间也是互相关联的。不同服务器如果存在同一应用的不同组件那么同样是互相关联。

2. 异常类型

对于异常类型判断也是异常检测问题的重点。异常大致可以分成三类[30]。

第一类是点异常。如果一个数据与其他所有的数据不相关，且与其他所有的数据相比处于异常状态，则该数据就是一个异常点。这类的异常称为点异常[31]。如图 2.3 所示，内存使用率在 t1 时刻突然增加，而其他的时刻保持比较稳定的状态，t1 时刻从整体上看就是一个点异常。虽然点异常的假设性较强，但是点异常最为直观，因此成为了异常检测研究方向关注的重点。例如在容器云平台中，内存的使用是一个比较稳定的曲线，如果内存的使用在某个时刻超过了正常的使用比例，但系统中没有新的申请记录的话，就认为这个时刻就是点异常。

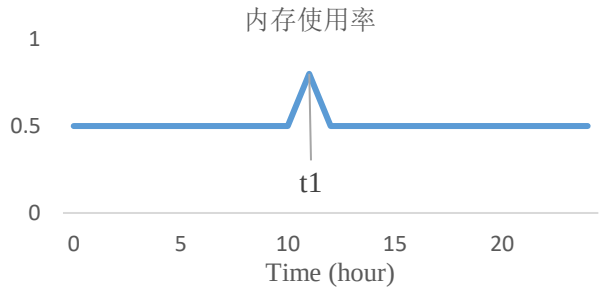

图 2.3 点异常示意图

第二类是上下文异常。如果某个数据在某种上下文条件下属于异常，这种异常被称为上下文异常[31]。上下文的概念由数据来源引起的，不同的数据来源造成了数据集上下文的不同。因此，每条数据实例都具有一个或者多个上下文属性。例如在容器云的 CPU 数据中，时间是上下文属性，应用类型也是上下文属性。区别于上下文属性的属性被称为行为属性。例如在容器云的 CPU 数据中，CPU 的使用率是行为属性。上下文异常由特定上下文的行为属性来定义。一条数据可能在某个上下文属性中是异常的，但是在另外一条上下文中是正常的。这个特性可以用来识别上下文属性和行为属性。上下文异常在与时间、空间等相关的数据中广泛存在，包含了时间、空间上的连续性。图 2.4 描述了一个 Web 应用程序的 CPU 使用率的上下文异常示例。该图记录一天之内 CPU 使用率的变化情况，t1 与 t2 时刻 CPU 使用率为 40%，由于 t2 时刻是在白天，所以 CPU 使用率可能是正常的，而 t2 时刻是在凌晨，其 CPU 使用率就是异常的。上下文异常的检测需要对于应用的领域有足够多的知识判断其异常是否符合其实际意义。另一个关键因素则是上下文属性的可定义性。如果定义一个数据集的上下文简单可用，采用上下文检测技术是可行的，但如果定义一个数据集的上下文非常复杂且难以理解，则采用上下文检测技术是不可行的。

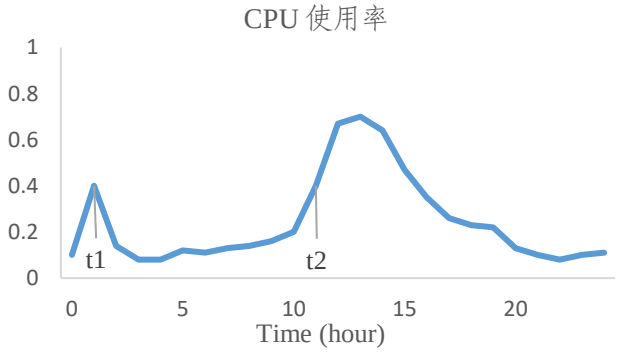

图 2.4 上下文异常示意图

第三类是聚集异常。聚集异常是指一组相关数据在整个数据集中所表现的都处于异常状态[31]。如果单独看异常组里的每个数据，其并不是异常，但是这组数据如果一起出现则是异常。如图 2.5 所示是应用的某个组件产生的 TCP 延迟。如果单独出现一个高 TCP 延迟，不认为是异常，但是如果一段时间 TCP 延迟一直较高，则认为是异常情况。

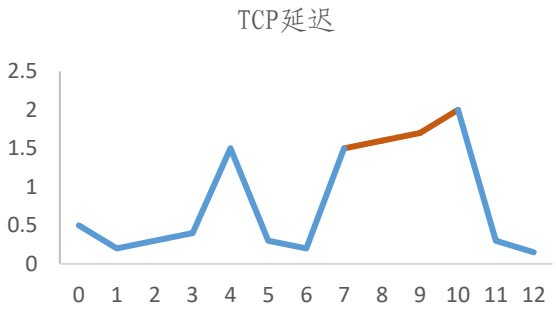

图 2.5 聚集异常示意图

上述三种异常发生的情况不一。点异常可以出现在任何数据集中，集体异常只能出现在数据具有相关性的数据集中，而上下文异常则取决于数据集是否有上下文含义。广义上来说，任何一个数据集都有自己的上下文环境，都需要考虑到诸如时间，空间这类上下文属性，因此，如果数据集的上下文属性容易表达，那么对于点异常的检测和对于聚集异常的检测可以归纳为对上下文异常的检测。

3. 数据标签

数据标签是指对于某个数据是否是异常数据由人工或者系统打上的标签。但是云计算系统每秒产生的数据量极大，只能选取其中部分数据进行标签化，无法对整个集群内所有的异常状况进行分类，同时由于不同的应用部署对应着不同的异常状态，因此，异常状态也具有动态性，因而无法判断基于之前的标签所得到的状态信息是否符合当前的状态。因此，需要通过机器学习技术对海量数据进行学习，让系统“学会”打标签，识别未知的标签。一般来说，根据获得标签的数据的多少，基于机器学习的异常检测技术可以分成三类：有监督异常检测，半监督异常检测和无监督异常检测。

有监督异常检测通常是需要人工对一组待训练的数据打上非常准确的标签。利用这些已经打上精确标签的数据训练一个模型用于预测正常类和异常类。然后通过这个分类器的所有数据都会放入这个模型中进行判断，最后为这些数据打上标签。有监督学习往往能够非常准确地辨别出应用的异常状态，但是需要大量的人工操作。

半监督异常检测与有监督异常检测不同，通常只对正常状态的数据进行标签化并训练模型。相比于有监督学习，半监督学习适用范围更广。通过对正常状态的不断迭代自学习，从而形成一个完整的正常状态模型，从而避开了未知异常类型的限制同时能够准确的识别已知或者未知的异常状态。

无监督异常检测通过对数据进行计算，将所有数据聚成多个集合，处于集合的数据被视为状态相同。无监督异常检测的基本假设是处于异常状态的数据要远少于处于正常状态的数据。因此，如果异常状态的产生过于频繁，会增加误报率。所以通常需要用特意挑选的异常状态远少于正常状态的数据集训练出基本的模型，增加模型的鲁棒性。

4. 异常检测方法

研究者们针对不同的异常类型提出了不同的异常检测方法。

点异常是为了简化问题而将异常中的上下文属性忽略而得到的异常，因而这类检测只需要对正常和异常进行准确的区分，即假设与正常状态数据差别较大的都是异常状态数据。检测点异常的技术有：1）基于分类的检测技术。通过机器学习或者深度学习技术在已有标签的训练集上训练出模型，将未知数据集导入到模型中，根据生成模型判断数据集中的数据的状态。2）基于聚类的检测技术。聚类检测技术对数据集进行聚类，正常的数据和异常的数据最后会分别聚在一起，由于正常状态的数据量远大于异常状态的数据量，因此判断那些数据量极少的类属于异常类。3）基于统计的检测技术。基于统计的异常检测技术认为正常数据出现在数据集中的概率高，异常数据出现在数据集中的概率低。通过分析正常状态数据，构造正常状态生成模型。如果未知数据在该模型中出现的可能性高，则认为是正常状态，如果出现的可能性低，则是异常状态。

上下文异常检测需要综合考虑上下文属性与数据本身。马尔可夫模型是一种常用于检测带有上下文属性异常的模型。通过计算数据在正常状态轮廓内状态转移的概率，模型能够判断数据处于正常或异常状态。累积和(Cumulative Sum, CUSUM)算法是一种序贯分析法，即计算一段时间内的序列数据平均数或方差，判断其是否出现了较大的改变的方法。

聚集异常通常而言需要严格的上下文定义，所以在检测方法上同上使用上下文异常检测方法。

5. 异常检测结果的输出

异常检测的最终目的是为云计算系统维护人员提供异常信息。因此，如何输

出表示异常检测的结果也是重要一环。一般来说，云计算系统维护人员希望异常检测的输出是正常或者异常的标签，但是由于无法绝对准确地检测异常，通常检测技术输出的结果是可能性最大的异常类型，通过打分的机制对异常的程度进行衡量。如果系统维护人员给异常程度设置一个阈值，那么异常检测技术可以通过比较异常的可能性和阈值来为异常状态进行标记。

2.3 异常定位技术介绍

异常定位的目的是确定异常发生的位置，减少后续搜索异常源的时间，提高云计算维护人员的维护效率。异常定位根据观察到的数据和现象进行推理，得到异常源的位置信息。但是在容器云平台中，层与层，组件与组件，组件内部均存在大量耦合，相互影响，因此如何定位异常成为了一个难题。

2.3.1 贝叶斯网络

贝叶斯网络是一种基于概率的不确定性推理网络[32]。它是一种用来表示变量之间连接概率的图形模型，通过图形化的方式清晰地表示节点之间的因果关系。如图 2.6 所示，下雪有可能导致堵车和摔跤，而堵车和摔跤都有可能导致迟到，严重一点的摔跤则会导致骨折，通过贝叶斯网络能很简洁明了地反映出事件之间的因果关系。作为一种基于概率的不确定性推理网络，贝叶斯网络已经广泛的被应用于医疗诊断，统计决策等领域的不确定信息的智能化系统中。

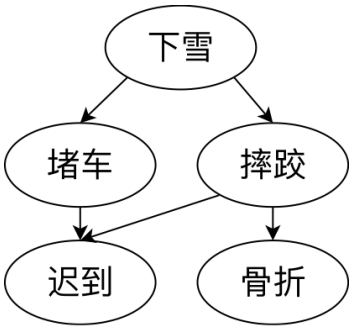

图 2.6 贝叶斯网络示意图

贝叶斯网络模型包含一个顶点集和一个边集，顶点集里的元素表示随机变量，用来表示实际问题的现象，属性，状态等。边集里的元素则表示顶点所代表的随机变量之间的因果关系。有向边的指向代表因果关系的方向，边的权值表示的是两个顶点的条件概率。如果两个顶点之间没有边连接，则表示顶点所代表的变量之间条件独立。

图 2.7 是一个简单的贝叶斯网络，其中 A 会引起 B，B 会引起 C，A 会引起 C。所以有

$$P(A,B,C) = P(C|A,B)P(B|A)P(A)。 \quad (2.1)$$

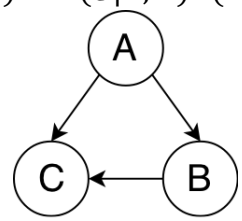

图 2.7 简单贝叶斯网络

贝叶斯网络在已知父节点的情况下，假设每个随机变量和其非直接后继随机变量相互独立，于是贝叶斯网络 G=(I,E)便有随机变量 X 的联合概率分布:

$$p(X) = \prod_{i\in I} p(x_i|x_{\pi i}) \quad (2.2)$$

其中πi表示的是表示 X 的节点 i 的父节点。以图 2.8 为例，条件一为 x1 和 x3 相互独立，条件二为 x5 和 x6 在 x4 给定的情况下独立，条件三为 x2 和 x5 在 x4 给定的情况下独立，其联合概率为:

$$p(x1,x2,x3,x4,x5,x6) = p(x1)P(x2|x1,x3)p(x3)p(x4|x2)p(x5|x4)p(x6|x4,x3)$$

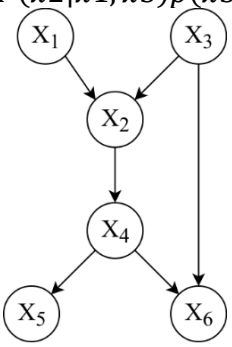

图 2.8 较复杂的贝叶斯网络

上述这三种条件对应了贝叶斯网络三种结构，第一种对应的是条件一，被称为 head-to-head 结构，第二种对应的是条件二，被称为 tail-to-tail，第三种对应的是条件三，被称为 head-to-tail。这种判断随机变量是否条件独立的方法被称为 D-分离。

贝叶斯网络作为一种表示对象间概率的图形模式，数学基础良好，可解释性较强。相比于决策树，神经网络等，贝叶斯网络具有表示形式直观，局部分布式推理，适用于表示不确定性与概率性问题等优点。

2.3.2 朴素贝叶斯

朴素贝叶斯是一种以假设特征之间强独立下运用贝叶斯定理构成的简单的概率分类器[33]，相比于贝叶斯网络其拓扑结构简单，速度更快。朴素贝叶斯假设目标值给定的时候目标属性相互独立。基于这个假设，朴素贝叶斯可以简单使用贝

叶斯定理构造网络。如图 2.9 所示，朴素贝叶斯仅有两层，一个父节点，若干子节点。相比于贝叶斯网络，朴素贝叶斯的结构简单。贝叶斯网络在构建完基础的拓扑结构后，需要对逐层的对节点间的因果关系进行计算。计算的过程开始于外围节点，按照层次进行，对于拓扑结构复杂的网络，计算这些条件概率的开销将会非常大，而且通过条件概率计算出后验概率存在误差。相对的，朴素贝叶斯由于其简单的拓扑结构，计算条件概率速度快，概率推理也非常的迅速。

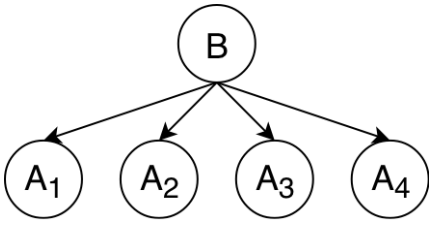

图 2.9 朴素贝叶斯示意图

例如上图所示的例子，令 B 表示对象的类别。B 节点有多个属性子节点，由贝叶斯定理可以得到 B 的后验概率:

$$P(B=b|A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4)=\frac{P(B=b)*P(A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4|B=b)}{P(A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4)} \quad (2.3)$$

如果使用贝叶斯网络对 B 进行分类，即计算:

$$B(b)=argmax(P(B=b|A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4)) \quad (2.4)$$

通过 B 的后验概率公式可以将问题转换为计算以下三项的值:

$$P(B=b) \quad (2.5)$$

$$P(A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4|B=b) \quad (2.6)$$

$$P(A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4) \quad (2.7)$$

其中式(2.5)的值可以通过数据集中得到，式(2.7)的值可以视为 1，关键在于求式(2.6)的值。在贝叶斯网络中，不同属性之间相互关联，因此求解式(2.6)的值相当于重构一个贝叶斯网络，使得问题变成了一个 NP 问题。所以贝叶斯网络的计算相对复杂。朴素贝叶斯的重要假设是属性相互独立，因此式(2.6)可以做以下转化

$$P(A_1=a_1,A_2=a_2,A_3=a_3,A_4=a_4|B=b)=\prod_{i=1}^{4}P(A_i=a_i|B=b) \quad (2.8)$$

单独计算$P(A_i=a_i|B=b)$的值相对容易的多。虽然一些特殊的应用中，对于给定对象，朴素贝叶斯分类性能较贝叶斯网络更好。但是朴素贝叶斯的假设过强，在实际应用中很难满足，而且朴素贝叶斯无法对样本缺失的情况做出处理。

2.3.3 人工神经网络

人工神经网络最早的模型是心理学家 Warren MaCullohc 与数学家 Walter Pitts

共同提出的兴奋与抑制型建立的 MP 模型[34]。利用 MP 模型，他们证明了单个神经元能够执行逻辑功能，奠定了神经网络基础。进入上个世纪八十年代，随着计算机的运算速度的加快以及互联网的普及，数据量的增大，神经网络再次成为研究的重点。如今，神经网络理论已经被应用于诸如模式识别，机器人，生物医学等领域，与不同领域的特色结合形成很多应用研究分支。

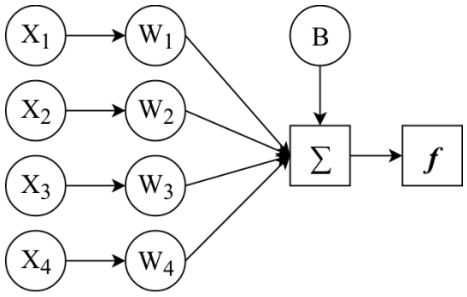

图 2.10 神经元示意图

神经元是神经网络的基本单位。神经网络由大量独立的神经元连接形成非线性且自适应的信息处理系统。如图 2.10 所示，神经元是一个拥有多个输入但是只有一个输出的非纯属组件，每个输入变量通过一个特定的权值输入神经元，并将它们累加。输出函数将输入的累加结果和与输入阈值进行计算，得到输出值。输入阈值即偏差，调节输入阈值的大小，就可以改变输出值，从而增加解决问题的种类。神经网络的基本特征是：1. 非线性。神经网络能够更好的模拟非线性的现实问题。2. 非局限性。神经网络的行为不仅和内部神经元的特征有关，还和神经元之间的连接作用方式相关。3. 非常定性。神经网络能够通过自学习，自适应等方式不断调节内部的结构，从而实现模拟现实问题的多变性。4. 非凸性。神经网络能够形成多个稳定的状态，能够很好的适应现实问题的非凸性。通过将系统所有节点的状态作为输入，神经网络能够逐步的分析节点之间的关联关系，最终达到一种稳定的关联关系状态，用以进行异常定位

虽然神经网络能够很好的对大量数据进行分析，但是神经网络的执行过程会消耗大量的资源。

2.4 本章小结

本章介绍了容器云相关技术以及异常检测的相关技术，以及其应用的场景。

人们希望能够对容器云平台进行异常检测，即对容器云异常状态进行检测。异常状态被定义为由于硬件问题，用户误操作或者资源分配不合理等因素造成的例如 CPU 使用率，内存使用率，IO 读写等容器云平台性能的不能由系统本身解

决的变化，因此需要通过分析容器云平台多层次的实时性能数据达到识别容器云异常状态的目的。容器云平台的异常类型主要是上下异常，对上下异常的检测适合本文提出的问题。

容器云平台异常检测的另外一个重要的部分是对异常进行定位，根据 2.2.3 节对于异常定位技术的介绍，贝叶斯网络能够反映组件之间，应用之间的依赖关系，适合于构造容器云内部复杂的依赖关系图。而异常定位的根本则是对这个依赖关系图进行分析，因而本文将会结合图论的相关知识对异常进行定位。

第3章 容器云跨层次异常检测系统设计

3.1 容器云平台结构

本文研究的是基于容器虚拟化技术的云计算平台（以下简称容器云平台）的异常检测。容器云平台的体系结构如图 3.1 所示。容器云平台中，根据服务器的不同类型和数量，整个数据中心被分成多个集群，如计算集群、存储集群、高可用集群等。计算集群由多个高性能服务器组成，通过虚拟化技术将服务器虚拟出多个虚拟机。存储集群则是由多个存储设备组成，存储设备通过网络提供服务以及通信。交换机负责连接多个集群和管理服务器。管理服务器负责管理整个数据中心的资源的分配和管理，如容器的创建，删除，迁移等。用户通过网络访问自己的容器集群，并在容器集群上创建自己的应用。本文中对于基于容器的云计算系统的研究工作主要放在高可用集群上即 Web 应用。

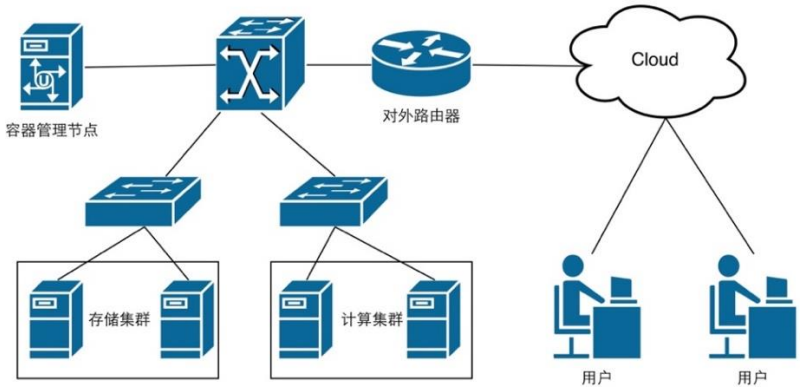

图 3.1 容器云平台体系结构示意图

集群中的每个物理主机的结构层次如图 3.2 所示。每台物理主机上部署了多个容器。所有容器共享物理机的资源，通过容器管理软件管理每台容器的具体资源使用。利用容器的快速部署的特点，云服务提供商可以非常便捷的创建或删除容器，从而可以灵活地调节资源的分配，达到减少资源的浪费，提高容器云平台的整体资源利用率的目的。用户在容器中运行自己的应用程序，也能够非常便捷的对应用程序的版本进行更迭。利用容器，应用程序不同组件之间实现了解耦。维护应用不需要将所有的组件全部停止，而仅需要更新某个组件所在的容器，大大减少了维护所需要的时间。

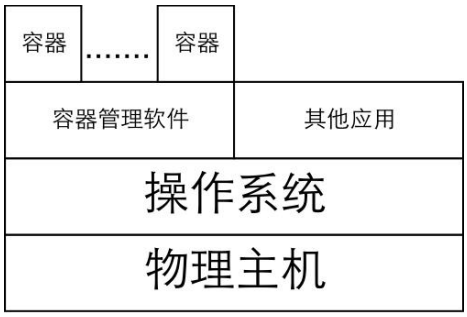

图 3.2 容器云主机层次结构示意图

3.2 容器云异常检测系统结构设计

本文中的容器云平台的异常检测系统主要针对 Web 应用进行异常检测。由于 Web 应用的不同组件对于资源的需求不同，如数据库组件主要消耗磁盘 I/O 资源，memory cache 组件主要消耗内存资源，而负载均衡组件主要消耗 CPU 资源。因而这三种组件的异常状态各不相同。容器云异常检测系统的结构如图 3.3 所示。

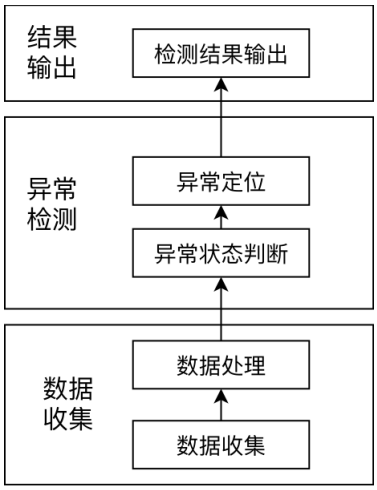

图 3.3 容器云异常检测系统结构示意图

容器云异常检测系统主要包含以下几个大的模块:

1. 数据收集

数据收集是容器云异常检测系统的重要模块之一。系统将会采集容器层和物理机层性能数据指标，如图 3.4 所示。其中容器的性能数据分成两类，一类是资源使用数据，一类是服务性能指标。服务性能指标主要为应用程序的服务状态信息。资源使用数据主要包括 CPU 使用情况，内存使用情况，I/O 使用情况，网络传输量等信息。由于物理机没有参与到整个应用集群的服务工作中而只是为容器提供相应的资源，物理机层将只会收集资源使用数据。

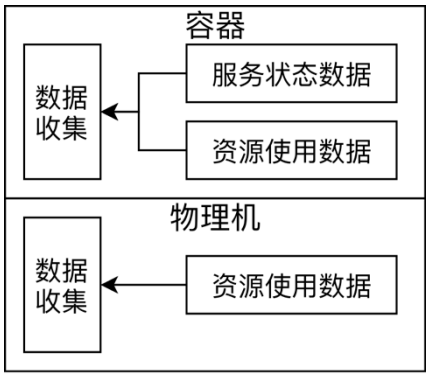

图 3.4 物理主机以及容器数据收集示意图

2. 数据处理

由于采集到的数据中包含许多冗余数据，系统在采集完性能数据之后需要对数据进行初步的处理和筛选。数据处理的过程如图 3.5 所示。数据收集完成后首先对数据进行相关性分析，去除冗余数据，然后通过主成分分析筛选出主要特征，最后将筛选之后的性能数据输出作为异常检测阶段。

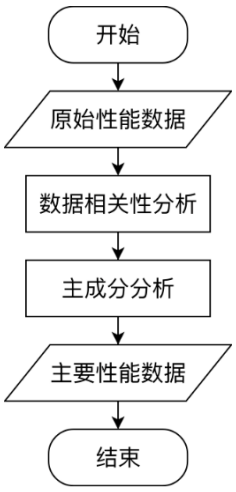

图 3.5 数据处理流程

3. 异常状态分析

为了降低系统开销，系统在执行异常定位之前需要进行系统异常状态的分析。异常状态分析过程如图 3.6 所示。数据筛选处理完数据后，首先由异常状态分析模块进行异常状态的分析。如果处于正常状态则继续等待输入数据并判断系统状态；如果处于异常状态，则调用异常定位模块对异常进行定位。

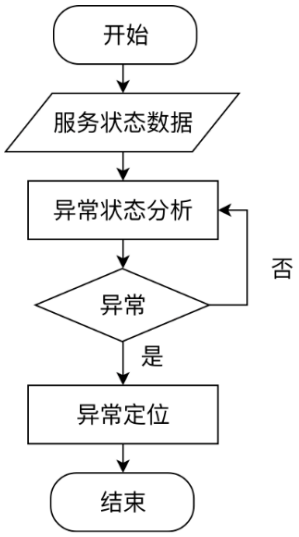

图 3.6 异常状态判断流程

4. 异常定位

当发现系统处于异常状态的时候，便开始进行异常的定位。异常定位的流程如图 3.7 所示。本文中所设计的异常检测系统仅对单个异常源进行检测与定位。异常的定位主要分为两个部分。一是节点的定位，即找到具体是哪个节点的故障，二则是性能指标的定位，即找到具体的是哪个性能指标出现异常。最后将两个部分的结果结合起来作为异常定位的结果输出。

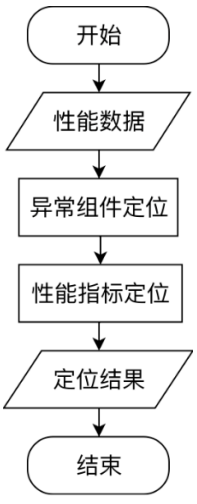

图 3.7 异常定位流程

5. 结果输出

容器云异常检测系统的最终目的是为系统管理员展示系统的异常检测结果，结果数据模块将检测将异常状态分析与定位结果输出，为系统管理员提供实时的

系统异常信息与可能发生的位置。

3.3 本章小结

本章介绍了容器云异常检测系统的总体架构设计。首先介绍了基于容器技术的云计算系统的总体架构，然后介绍了采用容器技术的服务器的架构，分析了采用容器技术的优势。紧接着介绍了容器云异常检测系统的基本模块，并对其中的数据收集，异常分析与定位，结果输出几个模块按照不同的应用类型进行了简单介绍

第4章 容器云跨层次异常检测系统关键技术实现

4.1 性能指标数据收集

面对基于容器技术的云计算系统越来越复杂的系统结构和行为，传统的针对单一方面的异常检测技术如日志审计分析[35]，网络流量监控[36]，操作系统监控[37]等方法已经不能准确的判断出异常了。

虽然传统的针对单一方面的异常检测技术不能直接判断出异常，但是通常来说，异常可以通过机器相关性能指标如 CPU 使用情况，内存使用情况，网络带宽占用率，磁盘 I/O 资源消耗等来体现。对于 Web 应用来说，短时间内服务更多的人以及每个人都能够更快的获得服务器的响应是这类应用追求的目标。对于批处理类任务，尽快的得到运算结果，提高运算效率是这类任务的追求。而异常行为，或者说异常行为，会造成机器相关性能的改变，使得 Web 应用服务效率降低，即增加了反应时间，提高了延迟。因而性能指标数据收集主要需要考虑到机器的性能，容器的性能以及应用的服务状态。

物理机数据的收集主要使用的是 python 提供的 psutil 模块，该模块能够轻松实现获取系统当前运行进程和系统各类资源的利用率信息，主要用与做系统监控，性能分析，进程管理。它实现了如 ps、top、lsof、netstat 等命令行工具提供的功能。通过 psutil 模块，系统的相关信息能够非常便捷的进行处理分析。

容器数据的收集依赖于 Docker 本身提供的信息。由于 Docker 的资源隔离是基于 cgroups，关于容器的性能指标能够从 cgroups 中获取。但是在 cgroups 中，内存，CPU，网络等分别属于不同的 namespaces,直接通过 cgroups 获取性能数据将会变得非常麻烦。Docker 技术方案作为一种成熟的技术方案，为了便于用户对容器进行实时的监控与操作，Docker 提供了远程 API 用于监控和操作容器。Docker 远程 API（Docker Remote API）允许用户通过 Unix socket 通信操作 Docker daemon，同时也可以通过 HTTP 调用 Rest API。用户可以用过发送 HTTP 请求获取到当前正在运行的容器的所有性能指标信息。由于处理 HTTP 请求的是 Docker daemon，因此对于容器内运行的程序影响相对较小，提高了性能数据的可靠性。

服务状态数据是指 SLO（服务等级目标）指标，通常和使用系统的用户的体验直接相关。由于 TCP 请求延迟能够反映 Web 系统的状态，且 TCP 协议是多种应用的基础通信协议如 MySQL，tomcat，nginx 等，同时 TCP 延迟相对较为简单，容易处理，因此本文主要采用 TCP 延迟作为 Web 的 SLO 指标。收集 TCP 延迟主

要采用 tcprstat，该工具通过监控网络传输来统计分析请求的响应时间,能够对特定端口的 TCP 数据包进行 TCP 延迟分析。网络工具 tcprstat 处理数据包的流程如图 4.1 所示。首先利用 libpcap 捕获 TCP 数据包，接着判断数据包的数据部分的长度，如果长度为 0 则表示该数据包是 TCP 控制包，如果长度大于 0 则表示该数据包是正常的通信数据包继续执行后续操作。接着提取该包的源 IP，目的 IP 以及源端口，目的端口，判断该包是请求包还是响应包。如果是请求包则将数据包的源 IP，目的 IP 等报文信息与时间戳信息记录下来，如果是响应包，则找到对应的请求包报文信息，计算时间差。最后将该请求包信息删除。

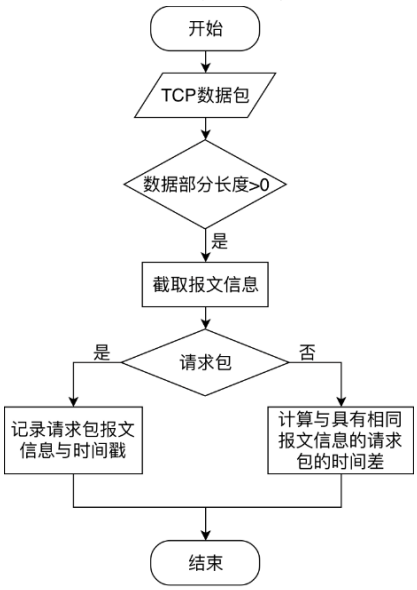

图 4.1 tcprstat 处理流程

利用 tcprstat 收集到的 TCP 延迟数据指标如表 4.1 所示。但是本文仅关注 TCP 延迟信息，所以其中一些数据将会在采集阶段就筛选掉，例如 stddev、95_std 等。

表 4.1 TCP 延迟数据指标

延迟数据指标	具体含义
timestamp	时间戳
count	TCP 请求总数
max	TCP 延迟最大值
min	TCP 延迟最小值
avg	TCP 延迟平均值
med	TCP 延迟中位数

续表 4.1

延迟数据指标	具体含义
stddev	TCP 延迟标准差
95_max	95%TCP 包的延迟最大值
95_avg	95%TCP 包的延迟平均值
95_std	95%TCP 包的延迟标准差
99_max	99%TCP 包的延迟最大值
99_avg	99%TCP 包的延迟平均值
99_std	99%TCP 包的延迟标准差

性能指标收集的频率要适中。采集频率过高会占用过多的系统资源，对异常的检测造成影响。但是性能指标频率太低又不能反映系统的实时状态，而且在发现异常状态的时候，数据收集组件又需要频繁的采集系统性能数据，所以数据收集组件应当能够根据系统状态自动的按照当前系统的状态调整采样频率。本文中容器云异常检测系统的数据收集模块采用动态数据收集策略，当管理节点没有检测到异常的时候，采用长的固定时间间隔的采样，当管理节点检测到异常的时候再减小采样间隔时间。这样既可以降低收集数据的开销，同时又能满足异常检测对于数据收集周期的要求。

4.2 性能指标数据处理与筛选

本文中的数据指标变化的幅度相对于系统资源总量而言过小，直接使用原始数据进行分析将会十分困难。本文通过对数据的简单分析，发现数据的变化幅度也服从正态分布，且其同样能够很好的体现性能指标之间的相互影响，因此，本文将会基于数据的增量进行数据的分析与处理。同时由于从异常出现到最终异常被检测到，中间存在一定的时间差。因此采用数据增量的方案需要对性能数据指标进行缓存。在进行异常定位的时候，检测系统将会从缓存中读取数据，在其中判断是否存在异常数据。

本文中的描述容器应用，容器本身以及宿主机性能的指标有可达上百个，如果采集上千条数据，整体的数据量将达到上万个。因此数据具有维度高，数据量大的特点。与此同时，采集的数据本身也有可能会有大量冗余。

首要的工作是去除冗余数据。系统采集到的数据很多的存在极高的相关性，这些冗余的数据可能会在后期构成网络的时候形成环，这会造成推理结果的不稳定性。因此，本系统将会从这些冗余数据中选取一个，将其他的数据从性能指标

表中移除。本文中冗余数据的筛选标准是：1）如果两个指标的数据相关系数超过或低于某个阈值例如 0.8 和-0.8，则从这两个指标中选取一个作为代表，移除另外一个。2）保留一些特定的指标，即使他们的相关系数超过了阈值，例如 net_tx_bytes 与 net_rx_bytes。筛选的过程如图 4.2 所示。

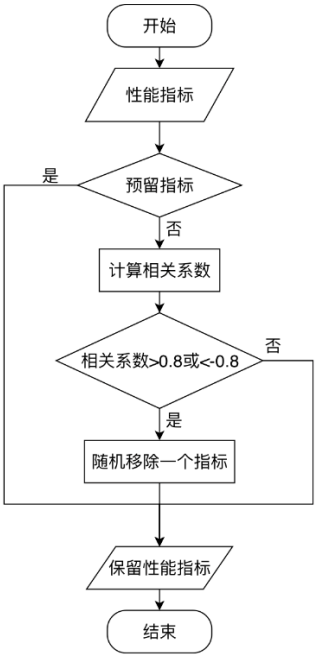

图 4.2 性能数据筛选流程

为了能够找到某些特定的特征来表示容器性能指标，降低后续模型训练的操作的计算复杂度，需要对原始数据进行特征分析。由于本文所收集到的数据并非相互独立，考虑到主成分分析仅仅只对数据进行线性变化，保持数据的方差最大化，保持了数据原有的分布。而且主成分分析简单高效。所以本文将采用主成分分析对容器的性能指标进行主要成分的分析。

令采集的容器应用性能指标数据构成矩阵 X，X 为 m*n 的矩阵，其中 m 是采集的数据总数，n 为每条数据所包含的性能指标的数量。

首先，计算矩阵 X 的转置X^T，并且将 X 转置去平均值。

第二步，对 X 进行奇异值分解从而获得特征向量矩阵

$$X = W\Sigma V^T \tag{4.1}$$

其中 W 是XX^T的特征向量矩阵，Σ是非负对角矩阵，V^T是X^TX的特征向量矩阵。

第三步，计算Y^T

$$Y^T = X^TW = V\Sigma^TW^TW = V\Sigma^T \tag{4.2}$$

当 m<n-1 的时候，V 不唯一，但是 Y 唯一，W 是正交矩阵且$Y^T W^T = X^T$，由此Y^T的第一列为第一主成分，第二列为第二主成分，由此递推。

第四步，降低数据维数。

$$Y = {W_L}^T X = \Sigma_L V^T \tag{4.3}$$

利用W_L将数据维数降低到 L 维，剔除掉对于整体分布影响不大的变量，从而达到降维的目的。

4.3 系统异常状态分析

容器云平台的异常分析指的是通过异常检测方法对运行在容器中的应用程序的性能指标进行分析，判断系统的运行状态。本文中对 Web 系统异常状态的检测是通过对 SLO 指标的状态进行分析。本文选取的 SLO 指标为 TCP 延迟指标，因此 Web 异常状态的检测是通过分析 TCP 延迟的变化实现的。显然，TCP 延迟指标异常的异常类型属于典型的上下文异常。因此采用结合滑动窗口的累积和(Cumulative Sum, CUSUM)算法判断系统是否处于异常状态。采用滑动窗口 CUSUM 算法的异常状态分析过程如下所示:

算法 滑动窗口 CUSUM 异常状态检测算法

输入：集群的 SLO 值即 TCP latency，阈值 h，窗口大小 k

输出：异常状态判断的结果 R

1.初始化滑动窗口$[L_0, L_k]$，初始化平均值$\overline{L_0}$，初始化累积和S_0=0

2.输入 TCP latency，设为L_t，t>k，将窗口右移$[L_{t-k}, L_t]$,计算 k 次内的平均值$\overline{L_t}$，令累积和值为S_{i-1}

3.输入 TCP latency，设为L_{t+1}，将窗口右移$[L_{(t+1)-k}, L_{t+1}]$,计算这 k 次内的平均值$\overline{L_{t+1}}$

4.计算$S_{i+1}=S_i+(L_{t+1}-\overline{L_t})$

5.计算$[L_{(t+1)-k}, L_{t+1}]$范围内 S 的最大值与最小值的差值S_{diff}作为预警值

5.判断S_{diff}是否大于等于 h 或小于等于-h，如果是则令 R=1，如果否，则令 R=0

6.输出异常状态判断结果 R

4.4 异常定位

由于容器云平台本身以及运行于其上的应用程序结构，当一个节点发生故障的时候，其造成的影响将会随着 Web 应用的结构网络迅速传播。因此异常定位的

首要工作就是对 Web 应用进行因果关系图构建。Web 应用的因果关系图分成两个部分，一个是组件之间依赖关系图，另一个是组件内部的性能指标的因果关系图。

4.4.1 组件异常源定位

组件异常源的定位主要分为两个部分，一是组件依赖关系图的构建，而是组件异常源的推理。

对系统进行异常定位的首要工作是确定系统内组件依赖关系。不同的应用程序的依赖关系不一致，因此需要通过对应用程序间的数据包进行分析，然后利用图论的知识构造出整个系统的依赖关系图，用于后期的异常源推理。

由于每一个节点运行唯一的组件，组件的依赖关系可以转化为节点的依赖关系。节点依赖关系图构建的过程如图 4.3 所示。主要分为三个步骤，网络流量数据分析，节点依赖关系判断，节点依赖关系网构造。

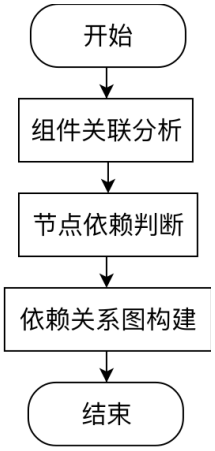

图 4.3 组件依赖关系图构建流程

1. 网络流量数据分析

网络流量数据分析的主要工作是提取网络流量中的信息，剔除掉无用信息。应用程序在运行过程中会产生大量的网络流量数据，其中组件之间的会频繁的产生网络通信。网络流量分析的第一步是将采集到的网络数据进行解析，从中获得源 IP，目的 IP，源端口，目的端口，从而获得组件之间的连接信息，即 channel。通常而言服务与端口是绑定的，所以通过查询系统的端口信息便可以将端口和服务进行一一对应。由此能够逐渐构造出一个节点之间的通信图。然而，由于节点与节点之间会相互通信，通过这种方式构造出来的图是无向图，只能反映出节点之间具有关联性，无法判断节点之间的依赖关系。因此需要对节点之间的依赖关系进行进一步的分析。分析系统内 Web 应用组件之间的关联关系的流程如图 4.4 所示，通过遍历整个应用的所有组件节点，找到组件节点之间的关联关系，并将

关联关系图输出作为节点依赖关系图构造的基础。

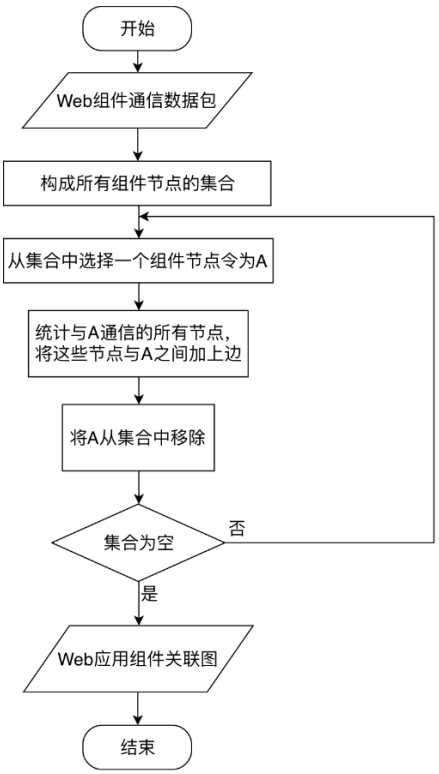

图 4.4 Web 组件关联图构造流程

2. 节点依赖关系判断

系统通过网络流量分析，得到了节点的关联关系图，下一步便是判断节点之间的依赖关系。网络依赖结构主要分成两种，一种是 Local-Remote 结构，一种是 Remote-Remote 结构。Local-Remote 结构如图 4.5 所示，Client 向 Serivce1 发送请求，由于 Service1 依赖于 Service2，因此 Service1 向 Service2 发送请求。在服务 1 需要等待服务 2 响应后才能继续执行。典型应用如传统的 Web 服务。Remote-Remote 结构如图 4.6 所示。Client 发送请求给 Service1，Service1 返回响应告知 Client 请求 Service2，Client 随后请求 Service2，Service2 返回响应。典型应用如 DNS 服务。本文研究的是 Web 应用程序，因此需要分析的结构主要是 Local-Remote。

图 4.5 Local-Remote 结构示意图

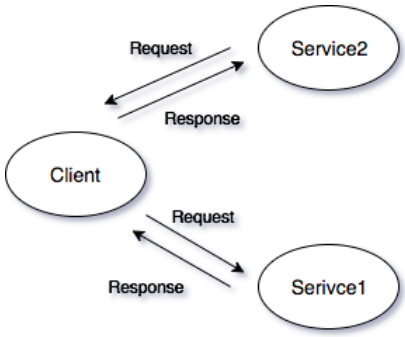

图 4.6 Remote-Remote 结构示意图

从 Local-Remote 结构可以看出 Service1 的 TCP 延迟与 Service1 的处理时间，Service2 的处理时间以及 Service1 与 Service2 的通信延迟相关，而由于 Service2 不会向 Service1 发送请求，因此 Service2 的 TCP 延迟只与 Service2 的处理时间相关。因此可以采用延迟相关性分析来区分两个节点的依赖关系[38]。首先手动增加 Service1 的输出 TCP 延迟，若 Service2 的输出 TCP 延迟随着 Service1 的 TCP 延迟增高而增高，则认为 Service2 依赖于 Service1，反之若 Service2 的输出 TCP 延迟基本不发生波动，则认为 Service2 不依赖于 Service1。通过对所有相互通信的节点进行依赖关系判断，系统将无向的边转化为有向的边，其中边的起点为被依赖的节点。

最后系统通过将有向的节点间的依赖关系进行组合，得到整个集群的依赖关系有向图。图 4.7 则给出了构建组件之间依赖关系图示例。整个依赖关系图的构建流程如图 4.8 所示。

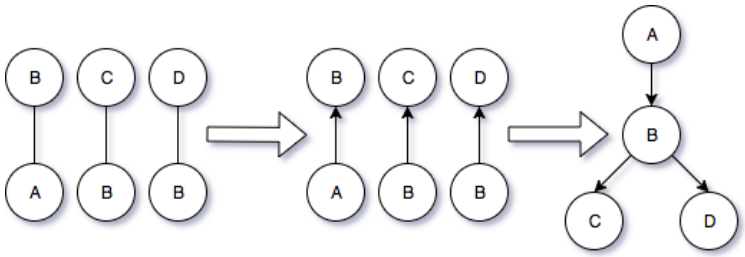

图 4.7 系统组件依赖图构建示意图

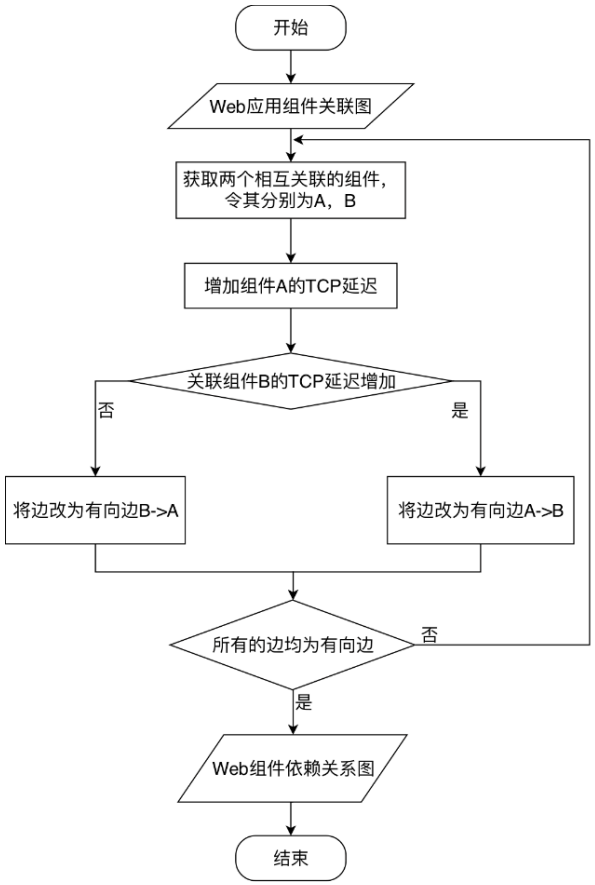

图 4.8 系统组件依赖关系图构件流程

常规的异常定位方法是通过遍历系统组件依赖图，逐个判断每个节点的正常与异常状态。但是基于容器的云计算系统内一个应用程序往往具有数量庞大的组件，如果每次定位异常都需要遍历一遍整个组件依赖图将会产生极大的开销。因此本文将会结合图论，简化图的遍历过程。组件异常源的推理流程如图 4.9 所示。首先确定每个组件的状态，可以通过系统异常状态分析的方法判断每个组件的状态。然后将异常状态子图从整个组件依赖关系图中提取出来。由于异常源的存在不依赖于其他的异常，在异常状态子图中作为异常源的顶点不存在入度，因此可以通过异常状态子图判断异常源。例如图 4.9，图中异常状态向量中 0 表示正常，1 表示异常。当获得一个异常状态矩阵的时候，首先将正常的节点从因果关系图中移除，得到一个异常子图，然后判断节点的入度，如果入度为 0 则是异常源。

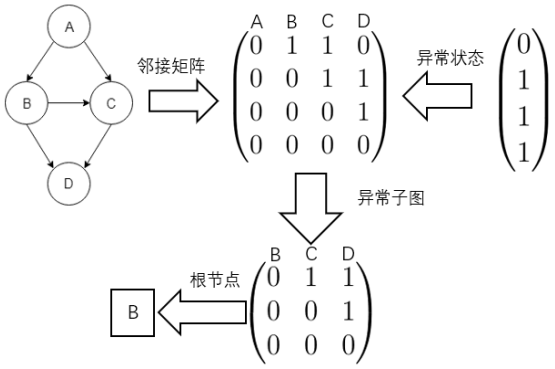

图 4.9 异常位置推理示意图

4.4.2 性能指标异常源定位

构造完成异常组件定位后之后，接下来就是对异常组件内的异常源进行推理。由于组件内部的性能指标相互影响，当组件中某个性能指标状态发生异常了，异常随后影响其他的性能指标的状态，最后形成一个异常性能指标构成子图。性能指标异常源定位的主要任务便是从异常性能指标构成的子图中识别出真正的异常源。本文中性能指标异常源定位总共分为两个部分，一个是状态因果关系网构造，一个是异常位置推理。

根据上文中对贝叶斯网络进行的简单的介绍，贝叶斯网络反映的是变量之间的因果联系。本文中状态因果网络主要是对 Web 应用程序不同组件的相关性能指标进行状态因果分析构造出来的贝叶斯网络。状态因果关系网络构造的主要工作是通过贝叶斯网络的构造方法构造节点内部的性能指标的依赖关系网络。由于容器运行在物理机上，物理机的故障也会造成容器的性能指标出现异常。在容器云平台中状态因果关系网络需要同时涉及到物理机和容器的性能指标。又因为状态因果关系网络需要检测的是异常状态的来源，所以本文中状态因果关系网络需要反应性能指标异常状态之间的相关性。状态因果网络的构造流程如图 4.10 所示。

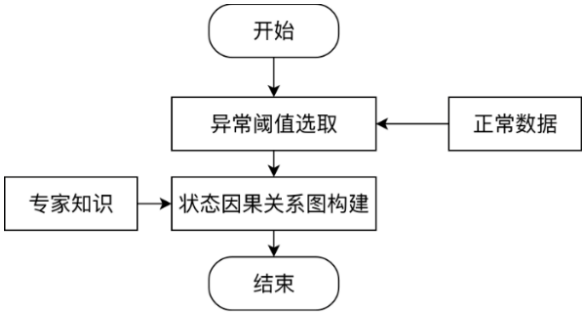

图 4.10 状态因果关系图构建流程

与系统异常状态不同，系统性能指标的异常更多的是点异常，即当前状态的

与前一时刻的状态无关。本文中系统性能指标的异常将采用基于统计的异常检测方法进行判断，即分析正常状态，找到正常状态的分布，从而判别出异常状态。如图 4.11 到图 4.14 所示，通过分析性能指标的增量可以看到，系统性能指标的增量服从高斯分布，因此其分布可以表示成$\mathcal{N}(\mu,\sigma^2)$。由正态分布的性质可知，若一个变量服从正态分布，其数值分布满足3σ原则。因此可以认为正常的数据主要分布在$(\mu-3\sigma,\mu+3\sigma)$之间。当某个值的增量不处于这个范围内的时候，将其判断为异常。μ和σ的值可以通过最大似然估计估计出来。估计的结果:

$$\hat{\mu}=\bar{X} \tag{4.4}$$

$$\hat{\sigma}^2=\frac{1}{n}\sum_{i=1}^{n}(X_i-\bar{X})^2 \tag{4.5}$$

通过不断的学习，可以更加准确获得μ和σ的值，从而能够更加准确地进行异常的判断。

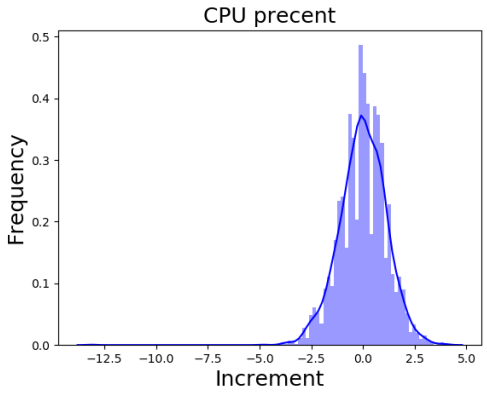

图 4.11 CPU percent 指标增量的分布

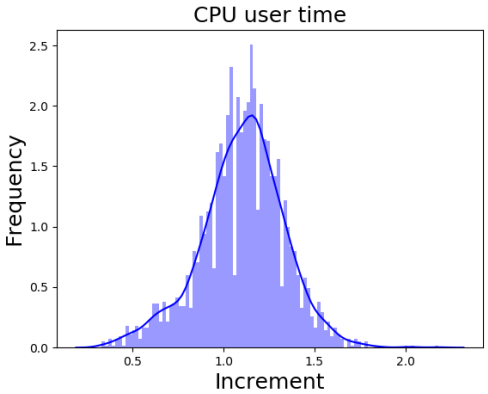

图 4.12 CPU user time 指标的增量分布

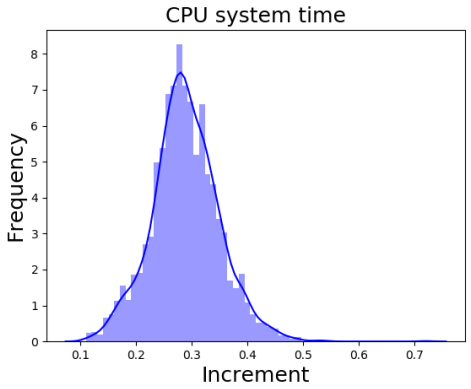

图 4.13 CPU system time 指标的增量分布

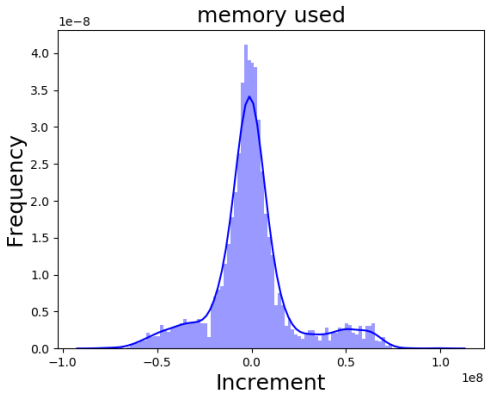

图 4.14 memory used 指标的增量分布

通常而言，输入请求的状态异常与节点内部指标无关，而输入请求的异常能够造成整个节点的性能指标增量的异常。因此，本文中的状态因果关系网络将输入的请求数量作为根节点。同样的 TCP 延迟不会影响到节点内的其他性能指标的改变，而其他性能指标则会影响到 TCP 延迟的状态。所以本文所构造的状态因果关系网络的结构满足两个基础规则:

1. 输入的请求数量没有父节点
2. 输出的 TCP 延迟没有子节点

状态因果关系网络的构造主要是找到各个性能指标之间的相关性。其相关性依据的来源主要从专家那里获取知识，通过专家的帮助手工搭建网络。这个过程通常还需要进行模型测试，通过估计一些“典型”的查询结果检查模型是否会返回可能的回答。然而事实上，大多数情况下，从专家那里获取全部知识几乎不可能，没有人能够精通所有的领域，而且依靠人工构建网络将花费大量的时间。同时大多数的领域里，数据分布的性质会随着各种各样的条件的变化呈现不同的分布，需要经常对网络进行重新构建。因此，一种更好的办法就是通过大量的数据

构建总体的分布模型，而且后期能够通过收集异常数据不断地对模型进行反复验证与调整。

本文通过模拟的方法收集服务状态异常时系统性能指标，并分析这些性能指标之间的相互关联，利用禁忌搜索算法[39]对性能指标进行分析形成贝叶斯网络的初期架构。然后根据经验对该贝叶斯网络的部分结构进行调整。随后系统将通过日志记录下服务异常状态下系统性能指标，并定期根据这些异常状态数据对状态因果关系网络结构进行验证和调整。

与组件异常源推理类似，性能指标异常源的推理也是采用图论的方法，利用异常源在异常子图中不存在前置节点的特性，对异常源进行推理。

4.5 本章小结

本章主要介绍了容器云异常检测系统的关键技术包括性能指标收集，异常状态分析以及异常的定位。系统首先通过训练数据以及专家知识训练出用于异常检测的状态因果关系网络，之后实时收集节点的服务状态信息，然后判断服务状态是否处于异常状态，一旦发现异常状态就进入异常定位阶段。在该阶段，首先要对结合组件依赖关系图，找到导致系统整体出现异常的根组件，然后对异常根组件的系统性能指标进行异常判断，最后通过性能指标的状态因果关系网络推理出具体的异常来源。

第5章 容器云跨层次异常检测系统测试与分析

5.1 系统构建环境

由于本文研究的内容是容器云环境，系统构造的环境主要是使用 Docker 构建容器云环境。测试环境有 2 台服务器组成。主机硬件配置如表 5.1 与表 5.2 所示。

表 5.1 主节点配置

主节点	
CPU	Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz
内存	64GiB
硬盘	16TB
网卡	1000Mbps

表 5.2 从节点配置

从节点	
CPU	Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz
内存	64GiB
硬盘	16TB
网卡	1000Mbps

测试环境的容器云平台为 Web 应用程序分配两台服务器，每台物理机上运行 3 到 4 个容器实例以模拟容器间的资源竞争，容器内的操作系统为 CentOS6.5，物理机操作系统为 Ubuntu14.04。

Web 应用由于差异性较大，因此本文仅选取具有代表性的三层结构，即负载均衡器, Web 服务器, 数据库。Web 应用的负载均衡器选用的是 nginx，容器镜像是基于 nginx 构造的镜像，Web 服务器选用的是 Tomcat7.0，容器镜像是基于 tomcat 官方镜像，数据库选用的是 MariaDB，MariaDB 是 MySQL 的开源版本。容器的部署方式是 swarm 方式，采用 docker-compose 部署 Web 应用集群。Swarm 是 Docker 公司发布的用于管理容器集群的工具。利用 Swarm，用户能够很方便的多个 Docker engine，而不需要频繁的切换物理机。Docker-compose 是一种用来定义相关联容器的文件，用户可以将所需要配置的容器的属性写到文件中，通过 docker-compose 启动容器或者容器组。

5.2 负载选择与故障注入

5.2.1 负载选择

本文中的 Web 应用的负载主要选择的是 TPC-W[40]。TPC-W 是一个适用于电子商务测试的基准，仿真模拟了一个在线书店的应用场景。TPC-W 测试的主要是支持服务的硬件和软件，因此能够很好的为本文的系统提供负载。TPC-W 包含四层，客户端，http 服务器层，Web 服务器层以及数据库层。Web 服务器运行 Tomcat，并且连接数据库，数据库选择的是 MySQL。为了保证多个 Web 服务器能够正常运行，本文为 Web 集群添加了 Memory cache 层。Web 负载选择的是 TPC-W 的客户端，TPC-W 客户端模拟用户正常交互的流程,能够充分的调用系统中的所有组件进行服务。

5.2.2 故障注入

基于容器的云计算系统纵向分为三层。为了对容器云平台的异常进行研究，需要将异常注入到容器云平台中模拟不同的异常状态。异常注入主要分成两个部分，一个是容器层的异常注入，一个是物理机的异常注入。本文通过系统压力测试软件 stress 软件模拟系统资源消耗，通过 Linux 网络防火墙 iptables 模拟网络丢包，traffic control 工具模拟网络延迟。容器层注入包括:

1. CPU 异常。通过 Linux 系统的压力测试软件 stress，创建多个计算 sqrt() 的线程占用多个 CPU。
2. 内存异常。通过 stress 软件产生多个处理 malloc()内存分配的进程不断消耗系统的内存，但是不会导致系统崩溃。
3. I/O 异常。通过 stress 软件产生多个读写文件的函数占用磁盘 I/O。
4. 网络异常。利用 iptables 增加容器的网络丢包，利用 traffic control 模拟网络延迟。

物理机层异常的产生与容器层的异常相同。

5.3 数据采集与处理

由于构建贝叶斯网络需要大量的数据进行支撑，、本文测试部分数据采集的频率将会超过正常应该采用的时间间隔，初步设定为 5 秒采集一次。按照 4.1 节所述的数据类型采集两类主要的数据指标: 一类是每个节点的 TCP 延迟数据，一类的每个节点的性能指标数据。TCP 延迟数据用于对系统状态进行检测以及故障源节点的判断，而性能指标数据主要用于学习因果关系网络结构以及定位具体的故障类型。为了能够更好的对系统性能指标进行异常判断，需要采集足够多的正常

数据，使得数据能够更好的拟合分布函数。同时为了能够找到系统异常传播的规律，需要使用异常数据进行分析。本实验中将采集 3000 条正常的性能指标数据用以模拟高斯分布，以及 1600 条不同的异常性能指标数据，每种异常 200 条用以构造因果关系网络。

物理机性能数据通过 python 的 psutil 模块进行收集并保存至 csv 文件中，便于下一步的分析。采集到的部分物理机性能数据如表 5.3 所示:

表 5.3 部分物理机性能数据

编号	user_time	system_time	interrupts	soft_interrupts
1	2178.12	1172.76	61272074	53496478
2	2178.65	1173.46	61285833	53503647
3	2178.86	1173.71	61296065	53511387
4	2179.12	1173.96	61305382	53517960
5	2179.38	1174.23	61314757	53524845
6	2179.6	1174.56	61324335	53531904
7	2179.77	1174.82	61332720	53537652
8	2179.97	1175.17	61342088	53544796
9	2180.18	1175.43	61352183	53552160
10	2180.42	1175.73	61361875	53559649
11	2180.63	1176.01	61371275	53567039
…	…	…	…	…

随后分析性能指标之间的相关性，如图 5.1 所示。按照本文对于性能指标的筛选方法，对数据进行筛选，例如 x_switches 和 interrupts 的相关性大于 0.8，则从 interrupts 和 x_switches 选取一个，另外一个则从指标集中移除。

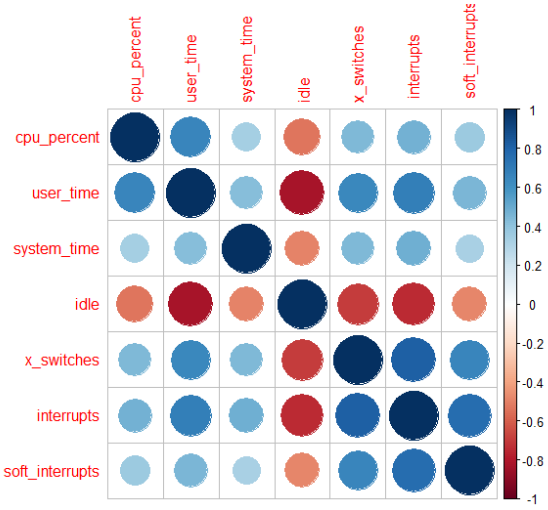

图 5.1 部分性能数据的相关性系数

经过相关性分析筛选过后，保留的物理机性能指标如表 5.4 所示:

表 5.4 保留的物理机性能指标

指标名称	具体含义
CPU percent	CPU 使用率
User time	CPU 用户态使用时间
System time	CPU 系统态使用时间
Interrupts	中断次数
Soft interrupts	软中断次数
Bytes sent	网络发送字节数
Bytes receive	网络接收字节数
Read bytes	磁盘读取字节数
Write bytes	磁盘写入字节数

容器性能数据可以通过访问 Docker daemon 获取。首先开启 Docker daemon 的 Remote API 选项，即修改 docker 配置文件，之后重启容器。然后通过 HTTP 请求或者 Python 的 docker 模块来获取 docker 容器的相关信息。由于从 Remote API 获取的数据是 Json 格式，在获取到数据之后，系统首先需要解析 Json 数据。之后与处理物理机性能指标数据类似，通过分析性能指标之间的相关性对容器性能数据进行筛选。同样的之后通过主成分分析获取主要性能数据指标。

TCP 延迟数据的收集使用的工具是 tcprstat。根据容器内运行的 Web 组件的不同，分别监控不同端口的 TCP 延迟。本文中容器内应用启动的端口为默认端口，例如 tomcat 容器使用 8080 号端，Memcached 使用 11211 端口，database 容器使

用 3306 端口。

5.4 异常检测定位模型训练与评估

5.4.1 异常检测模型构建

为了减少异常定位的次数，降低系统额外资源开销，需要首先对系统异常状态进行判断。本文中系统的状态信息主要由 TCP 延迟信息提供，因此分析系统的异常问题就转化成了分析 TCP 延迟信息的异常。当节点发生故障的时候，若节点对于出现故障的资源较为依赖，TCP 延迟就会随之升高，进而影响其他节点的 TCP 延迟。如图 5.2 所示，当 Database 节点出现 CPU 异常的时候，Database 本身 TCP 延迟会增加。

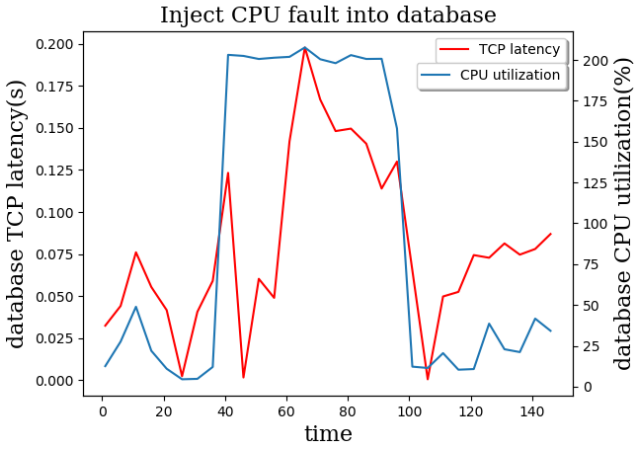

图 5.2 Database 节点的 CPU 故障与 TCP 延迟

如图 5.3 所示，在 40 秒处向 Database 节点注入时长为 60 秒的故障，导致 DB 节点在 40 秒到 100 秒的时间段内 TCP 延迟上升。

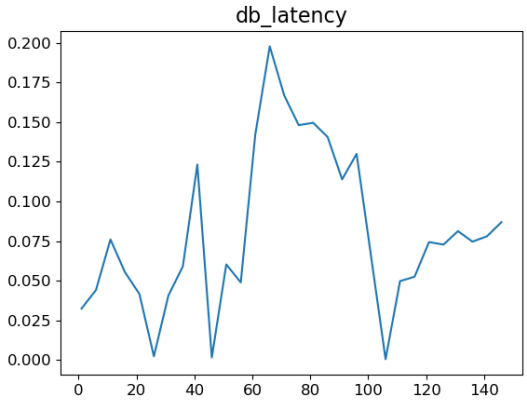

图 5.3 Database 节点注入故障

随后观察其余各类组件的 TCP 延迟变化。memory cache 组件的 TCP 延迟变化如图 5.4 所示，其 TCP 延迟的变化没有明显的波动。

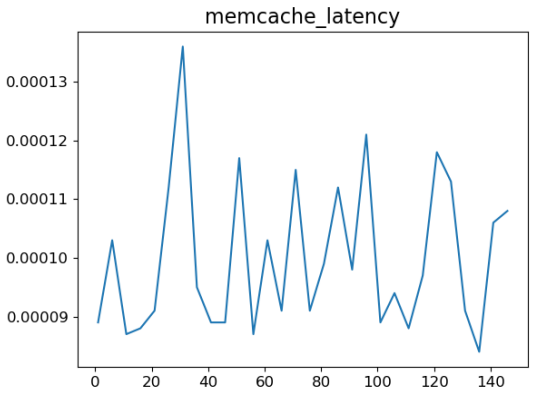

图 5.4 memory cache 节点 TCP 延迟变化

而 tomcat 组件与 nginx 组件的 TCP 延迟则随着 DB 节点升高而升高，如图 5.5 与图 5.6 所示。

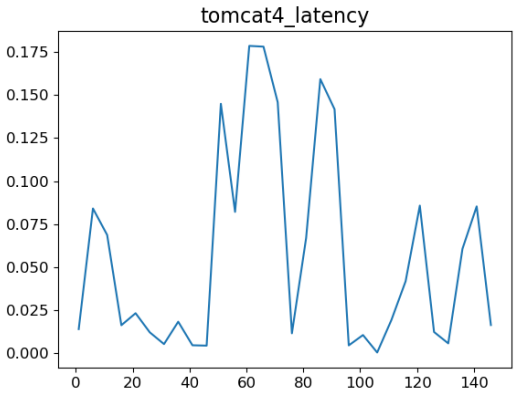

图 5.5 tomcat 节点 TCP 延迟变化

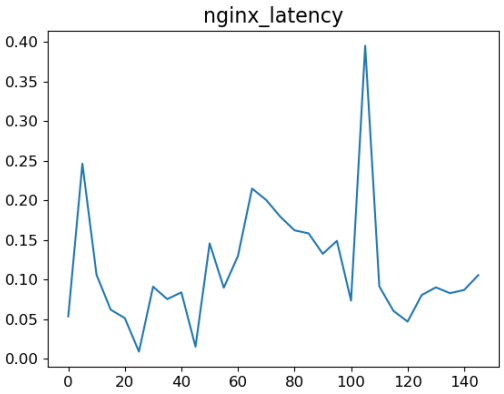

图 5.6 nginx 节点 TCP 延迟变化

本文采用滑动窗口 CUSUM 方法判断系统是否处于异常状态。例如，nginx 节点的 TCP 延迟信息利用 CUSUM 方法分析的部分结果如图 5.7 所示。当 TCP 延迟变化的幅度超过或小于某个设定的阈值的时候，就认为当前系统处于异常状态，

如图 5.7 中异常点所示。阈值的选取将会影响异常检测的频率。本文中阈值的选取考虑 TCP 延迟变化的分布，一般而言正常情况下 TCP 延迟的波动服从正态分布，因此，其正常状态下最大增量的绝对值一般会小于某个值。如图 5.7 所示，nginx 节点的 TCP 延迟的波动均小于 0.15 秒，因此选取 0.15 秒作为 CUSUM 判断时候的阈值。

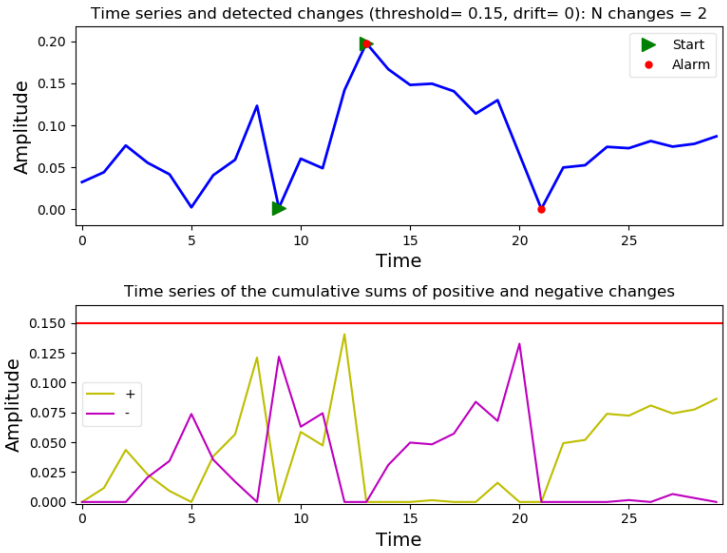

图 5.7 Database 节点 TCP 延迟 CUSUM 部分分析结果

同时可以从图 5.2 看出，尽管某个组件处于异常状态，仍然有部分处于系统异常状态的时间段内，该组件并不处于异常的状态。这是因为不同的 Web 系统组件主要消耗的系统资源不同，对系统异常的敏感度不同所致。例如 nginx 节点较多的使用 CPU 资源，对于 CPU 异常的敏感度较高，而对于磁盘 IO 异常敏感度不高。因此，在对异常进行定位的时候需要考虑到组件对于系统异常的敏感度。

5.4.2 异常定位模型的构建

1. 组件依赖关系图构建

根据 4.4.1 节的依赖关系图构建方法，将容器两两分组，分别增加 TCP 延迟，观察容器之间的 TCP 延迟的变化。如图 5.8 所示，当增加 database 节点的 TCP 延迟时，tomcat3 节点的 TCP 延迟随之升高，而增加 tomcat3 的 TCP 延迟时，database 节点的 TCP 延迟并无显著增加如图 5.9 所示。可以得出结论 tomcat3 依赖与 database 节点。

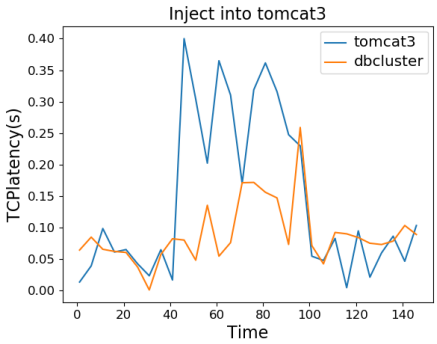

图 5.8 tomcat3 节点注入故障

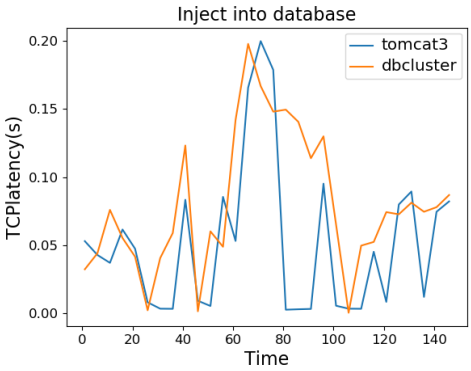

图 5.9 Database 节点注入故障

通过两两组件之间进行判断，可以得到最终的系统内部组件节点的依赖关系图，如图 5.10 所示。

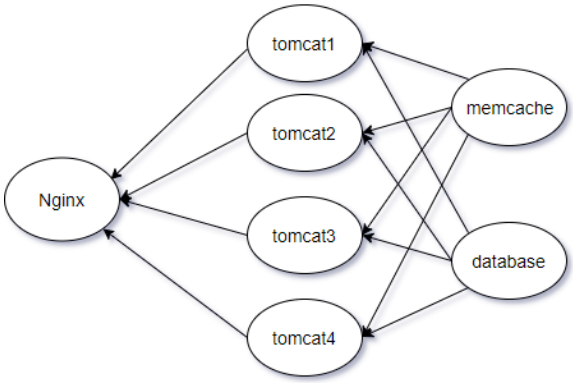

图 5.10 组件节点依赖关系图

2. 状态因果关系网络的构造

根据 4.4.2.1 中对于性能指标异常阈值的选择方法，利用正常数据的性能指标增量模拟高斯分布，通过最大后验估计计算$\hat{\mu}$与$\hat{\sigma}^2$的值，得到正常的系统增量范围$(\hat{\mu}-3\hat{\sigma},\mu+3\hat{\sigma})$。

状态因果关系网络的构造需要一定的专家知识，因此本文将会结合 R 语言的贝叶斯网络包、pcalg 包以及一定的常识构造贝叶斯网络。

首先利用 PCA 方法处理各层的性能指标数据，从这些数据中选取主要的影响因素，将这些指标作为贝叶斯网络的训练数据的指标。

将训练数据进行离散化处理。考虑到组件对于不同性能指标具有不同的敏感度，可以适当调整区间大小。当性能指标数据不属于阈值区间的时候将其标记为异常，当性能数据属于阈值区间的时候将其标记为正常。打完标签后的性能指标数据如图 5.11 所示。其中 0 表示正常，1 表示异常

Con_blkio	Con_CPU_total	Con_mem_usage	Con_net_rx_bytes	Con_net_tx_bytes	request	TCPlatency	CPU_user_time	mem_used	write_bytes	tx_bytes
1	1	1	1	1	0	1	1	1	1	1
0	1	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0

图 5.11 系统性能指标标签化

利用标签化的性能指标数据以及部分专家只是构造完成的贝叶斯网络的部分如图 5.12 所示，其中带 Con 的表示容器指标。由于性能指标过多，这里仅展示数据库节点的部分贝叶斯网。其中以 Con 开头的为容器性能指标，物理机性能指标由多个容器的指标共同影响。组件 TCP 延迟的依赖关系与组件依赖关系相同。

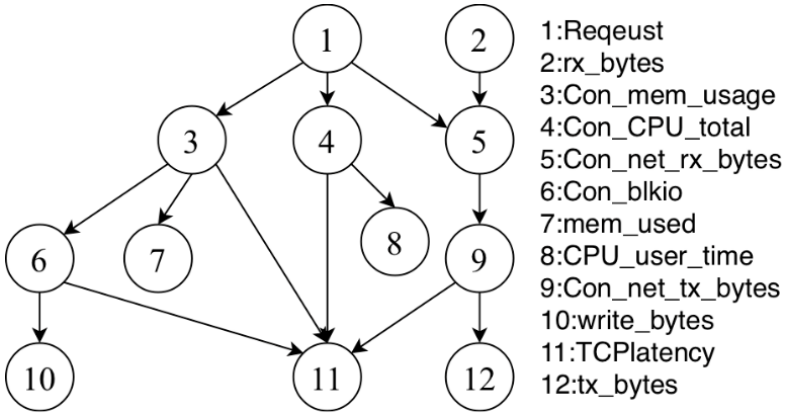

图 5.12 部分因果关系网络

之后将因果关系网络转换为邻接矩阵如图 5.13 所示为部分邻接矩阵。其中 1 表示存在边，0 表示不存在边

```
[[ suppressing 12 column names 'request', 'TCPlatency', 'CPU_user_time' ... ]]

request          . . . . . 1 1 . 1 .
TCPlatency       . . . . . .
CPU_user_time    . . . . . .
mem_used         . . . . . .
write_bytes      . . . . . .
tx_bytes         . . . . . .
rx_bytes         . . . . . 1 .
Con_mem_usage    . 1 . 1 . . . . 1 . .
Con_CPU_total    . 1 1 . . . . . .
Con_blkio        . 1 . . 1 . . . . .
Con_net_rx_bytes . . . . . 1
Con_net_tx_bytes . 1 . . . 1 . . . . .
```

图 5.13 部分邻接矩阵

5.4.3 异常定位

当系统处于异常状态的时候，将采集到的性能数据按照性能数据异常的判别方法打上标签，取出其中为异常的性能数据指标。按照 4.4.2.3 节的推理方法进行推理，可以定位到系统性能异常的根源。

5.4.4 模型评估

模型评估主要分成两个部分，一个是评估异常检测的准确率与召回率，一个是评估异常定位的准确率与召回率

异常检测的准确率与召回率如图 5.14。可以看到准确率和召回率均较高，但是由于滑动窗口 CUSUM 算法的特点与阈值的选择，异常检测的准确率相比于召回率不高。

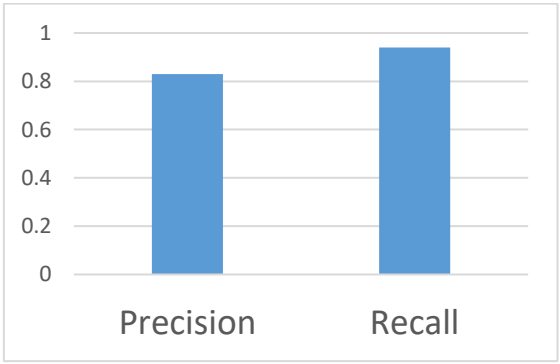

图 5.14 异常检测准确率与召回率

当系统检测到异常的时候，将会调用异常定位算法。本文评估的是当系统确定发生异常的时候，系统定位到异常来源的准确率与召回率。当位于同一主机上的所有的容器的相同指标均被判断成异常来源的时候，就认为是主机相应的指标是异常来源。按照 5.2.2 节所描述的方法注入的异常。评估结果如图 5.15 与图 5.16 所示。

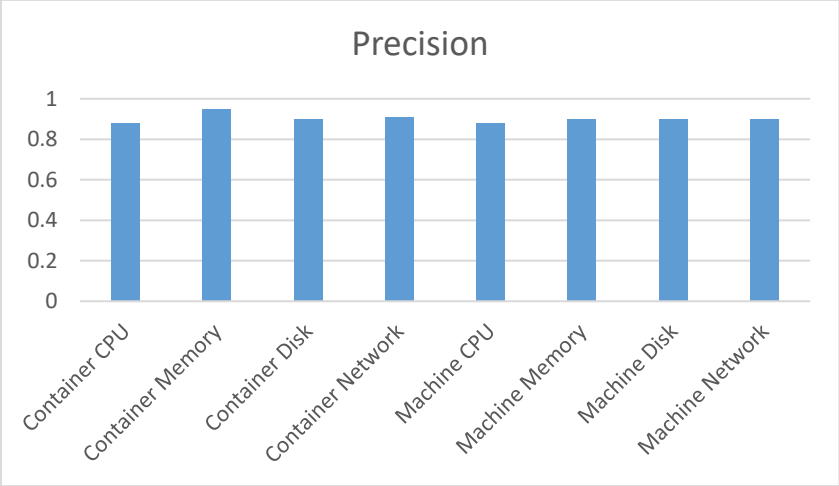

图 5.15 异常定位准确率

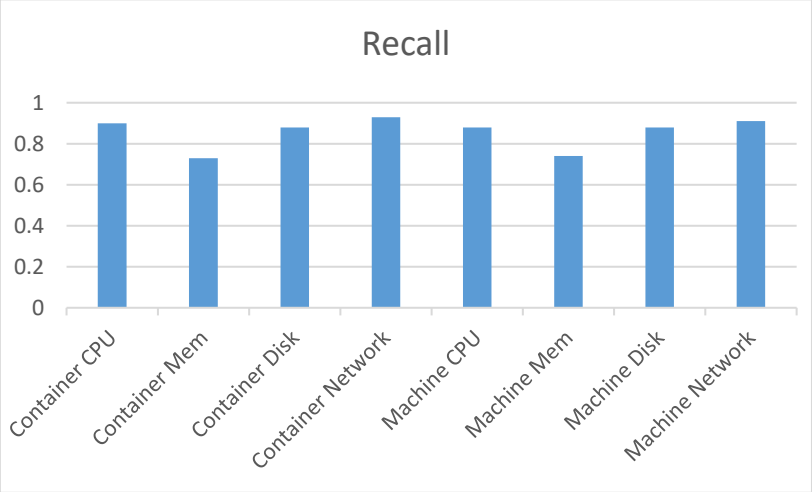

图 5.16 异常定位召回率

可以看到，对于内存异常，异常检测系统的准确率较高而召回率不高，这是因为本文中对于异常的判断首先需要对异常进行检测。相比 CPU，磁盘 I/O 和网络等异常，内存发生异常需要更长的时间才能对组件的状态造成较大的影响，因此其召回率比较低。而其他的性能指标相对来说变化会更加显著，其准确率和召回率相对较高。

5.5 本章小结

本章对容器云跨层次异常检测系统进行了测试与分析。详细介绍了系统部署的环境，系统所使用的负载，并介绍了系统故障注入，数据采集与处理的方法。之后对介绍了模型构建的方法，包括组件节点之间的依赖关系图的构建以及性能指标的因果关系网的构建。随后介绍了系统异常状态的判断方法以及性能指标异常的判断方法。最后对系统的准确率和召回率进行了分析。

第6章 总结与展望

6.1 总结

容器以其轻量级的特点成为了研究的热点之一。然而同样是因为容器的轻量化的特点，其布置开销降低，因此容器云平台的结构越来越复杂，规模越来越大，传统的异常检测系统已经无法满足容器云平台的环境。同时由于容器隔离性较差，导致异常的传播更为广泛更为迅速，在规模庞大的容器云环境中，传统的异常定位方法已经不能很好的实现定位。本文针对容器云平台的这一特点，设计了一个容器云异常检测系统。并且对系统进行了详细的描述，其中异常分析与定位是本文的重点。该异常检测系统针对运行 Web 应用的容器云平台进行了异常检测与异常定位。系统利用结合滑动窗口机制的 CUSUM 算法对 Web 应用进行异常状态监测。之后通过对 Web 应用组件之间的数据流量进行分析，系统建立起 Web 应用组件之间的关联关系。随后利用贝叶斯网络的构造方法构造出组件内的性能指标数据的因果关系网。然后结合图论的相关知识构造异常定位模型。最后对检测和定位模型进行了评估。

为了评估该系统，本文搭建了容器云平台，并且在平台内运行 Web 应用。通过人为注入故障模拟系统异常，最后对异常检测以及异常定位的准确性进行了评价

本文主要的内容和创新点有:

1. 利用 Docker 搭建了容器云环境，在容器云环境上建立了模拟 Web 应用，并详细描述了 Web 应用的特点，针对 Web 应用的特点设计了跨层次的异常检测系统，这种跨层次的异常检测研究提高了容器云平台的可用性，目前针对容器云平台的跨层次进行异常检测与定位的研究较少。

2. 针对 Web 应用的特点设计了对应的性能收集模块，有效的表示了 Web 集群运行状态，并对收集到的性能数据指标进行有效的筛选和过滤，从而减少了由于冗余数据带来的额外开销。

3. 研究了基于容器技术的 Web 应用的异常检测方法。在异常检测的过程中，采用了结合窗口机制的 CUSUM 方法对异常进行检测，提高了异常检测效率。

4. 研究了依赖关系构建方法。先利用组件节点之间的通信状态构造了组件关联关系图，然后分析组件之间的依赖，最后构造组件的依赖关系图。在构造

组件内部性指标的因果关系图时采用了贝叶斯网络的构造方法。

5. 研究了异常定位的方法。利用图论的相关知识对异常进行定位。

6.2 展望

本文对基于容器技术的云计算系统的异常检测系统，并针对不同的应用框架采取了不同的异常检测和异常定位方法。然而由于基于容器的云计算系统复杂多变，仍需要针对以下几个方面进行完善:

1. 本文 Web 应用框架虽然能够已经能够准确的检测出异常状态与定位异常，但是仅针对 Web 应用框架不能够反映多变的容器云环境，后续会考虑更多典型应用框架的异常检测与异常定位。

2. 本文对于性能数据指标的增量进行了研究与分析，对于内存异常无法很好的判断集群是否处于异常状态。后续将会对性能指标本身进行更加深入的研究，提高对异常状态的识别准确度。

3. 本文因果关系网络的构造依赖经验，会造成部分失真，因此后续会考虑更换无需先验知识的贝叶斯网络的变种，降低经验带来的识别率降低。

参考文献

[1]. Jiang Q. Virtual machine performance comparison of public IaaS providers in China [C]. Proceedings of 2012 IEEE Asia Pacific Cloud Computing Congress, 2012: 16-19.

[2]. Mehrotra P, Djomehri J, Heistand S, *et al*. Performance evaluation of amazon EC2 for NASA HPC applications [C]. 2012 3rd ACM Workshop on Scientific Cloud Computing, 2012: 41-49.

[3]. Monma H. FGCP/A5: Paas service using windows azure platform [J]. Fujitsu Scientific and Technical Journal, 2011, 47(4): 443-450.

[4]. Malawski M, Kuzniar M, Wjcik P, Bubak M. How to use google app engine for free computing [J]. IEEE Internet Computing, 2013, 17(1): 50-59.

[5]. Shvachko K, Kuang H, Radia S *et al*. The Hadoop distributed file system [C]. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010: 1-10.

[6]. Apache Hadoop [EB/OL]. http://hadoop.apache.org/.

[7]. White T. Hadoop: The Definitive Guide [M]. O'Reilly Media/Yahoo Press, 2012.

[8]. Apache Spark[EB/OL]. https://spark.apache.org/

[9]. Veeraraghavan K, Meza J, Chou D, *et al*. Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services[C] Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016.

[10]. Reiss C, Tumanov A, Ganger G R, *et al*. Heterogeneity and dynamicity of clouds at scale:Google trace analysis[C] ACM Symposium on Cloud Computing. ACM, 2012:1-13.

[11]. Lu C, Ye K, Xu G, *et al.* Imbalance in the cloud: An analysis on Alibaba cluster trace[C] IEEE International Conference on Big Data. IEEE, 2017:2884-2892.

[12]. Vahdat A, Vahdat A, Vahdat A, *et al*. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure[C] Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:58-72.

[13]. Kc K, Gu X. ELT: Efficient Log-based Troubleshooting System for Cloud Computing Infrastructures[C] Reliable Distributed Systems. IEEE, 2011:11-20.

[14]. Nagaraj K, Killian C, Neville J. Structured comparative analysis of systems logs to diagnose performance problems[C] Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2012:26-26.

[15]. Ding R, Zhou H, Lou J G, *et al*. Log 2 : a cost-aware logging mechanism for performance diagnosis[C] Usenix Conference on Usenix Technical Conference. USENIX Association, 2015:139-150.

[16]. Zou D, Qin H, Jin H, *et al*. Improving Log-Based Fault Diagnosis by Log Classification[M] Network and Parallel Computing. Springer Berlin Heidelberg, 2014:446-458.

[17]. Wang C, Talwar V, Schwan K, *et al.* Online detection of utility cloud anomalies using metric distributions[C] IEEE/IFIP Network Operations & Management Symposium. IEEE, 2010:96-103.

[18]. Sharma B, Jayachandran P, Verma A, *et al.* CloudPD: Problem determination and diagnosis in shared dynamic clouds[C] IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, 2013:1-12.

[19]. Pannu H S, Liu J, Fu S. A Self-Evolving Anomaly Detection Framework for Developing Highly Dependable Utility Clouds[J]. 2012:1605-1610.

[20]. Dean D J, Nguyen H, Gu X. UBL: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems[C] International Conference on Autonomic Computing. ACM, 2012:191-200.

[21]. 林铭炜. 面向云平台的虚拟机异常行为检测方法研究[D]. 重庆大学, 2014.

[22]. Apte R, Hu L, Schwan K, *et al.* Look who's talking: discovering dependencies between virtual machines using CPU utilization[C] Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. USENIX Association, 2010:157-197.

[23]. Natarajan A, Ning P, Liu Y, *et al.* NSDMiner: Automated discovery of Network Service Dependencies[C] INFOCOM, 2012 Proceedings IEEE. IEEE, 2012:2507-2515.

[24]. Sigelman B H, Barroso L A, Burrows M, *et al*. Dapper, a Large-Scale Di stributed Systems Tracing Infrastructure[J/OL]. https://static.googleusercontent.com /media/research.google.com/zh-CN//archive/papers/dapper-2010-1.pdf

[25]. Barham P, Isaacs R, Mortier R, *et al*. Magpie: online modelling and performance-

aware systems[C] Conference on Hot Topics in Operating Systems. USENIX Association, 2003:15-15.

[26]. Attariyan M, Chow M, Flinn J. X-ray: automating root-cause diagnosis of performance anomalies in production software[C] Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2012:307-320.

[27]. Nguyen H, Dean D J, Kc K, *et al.* Insight: in-situ online service failure path inference in production computing infrastructures[C] Usenix Conference on Usenix Technical Conference. USENIX Association, 2014:269-280.

[28]. 丁轶群，张磊等，Docker 容器与容器云[M] 北京：人民邮电出版社，2016

[29]. 阎巧, 谢维信. 异常检测技术的研究与发展[J]. 西安电子科技大学学报(自然科学版), 2002, 29(1):128-132.

[30]. V. Chandola, A. Banerjee, V. Kumar. Anomaly detection: a survey. ACM Computing Surveys, 2009, 41(3): 1-58.

[31]. Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey[J]. ACM Computing Surveys (CSUR), 2009, 41(3): 15.

[32]. Pearl J. Probabilistic Reasoning in Intelligent Systems: networks of plausible inference. Morgan Kaufman,1988

[33]. Rish I. An empirical study of the naive Bayes classifier[J]. Journal of Universal Computer Science, 2001, 1(2):127.

[34]. Pitts W. A logical calculus of the ideas immanent in nervous activity[M] Neurocomputing: foundations of research. MIT Press, 1988:115-133.

[35]. 韦勇, 连一峰. 基于日志审计与性能修正算法的网络安全态势评估模型[J]. 计算机学报, 2009, 32(4): 763-772.

[36]. 张焕国, 陈璐等. 可信网络连接研究[J]. 计算机学报, 2010, 33(1): 706-717.

[37]. 沈胜庆. 嵌入式操作系统的内核研究[J]. 微计算机信息, 2006 (02Z): 72-74.

[38]. Zand A, Vigna G, Kemmerer R, *et al*. Rippler: Delay injection for service dependency detection[C]INFOCOM, 2014 Proceedings IEEE. IEEE, 2014: 2157-2165.

[39]. Glover F. Future paths for integer programming and links to artificial intelligence e[J]. Computers & Operations Research, 1986, 13(5):533-549.

[40]. TPC-W[EB/OL].http://www.tpc.org/tpcw/default.asp

作者简历

教育经历:

2016 年 9 月 – 至今　　浙江大学 软件工程　硕士研究生

2012 年 9 月 – 2016 年 6 月　　武汉大学 信息安全 大学本科

工作经历

2017 年 9 月 - 至今　　中国科学院深圳先进技术研究院 实习

攻读硕士期间发表的论文和完成的工作简历:

[1]. Lu C, Ye K, Xu G, *et al.* Imbalance in the cloud: An analysis on Alibaba cluster trace[C] IEEE International Conference on Big Data. IEEE, 2017:2884-2892.

致谢

时光飞逝，岁月如梭，转眼间两年的硕士生涯结束了。两年的硕士生涯给我带来了许多美好的回忆。这两年里我学到了许多的知识，认识了许多的人，结交了许多的好友。他们帮助了我很多，也教会了我很多。

首先我要感谢我的父母，他们不遗余力的支持我，帮助我，在我对未来最迷茫的时候给予了我支持。

其次我要感谢贝毅君老师，他带我认识了大数据领域，教会了我许多大数据领域的相关知识，还帮助我完成了毕业论文的撰写。

还有，我也要感谢我的好友们，你们给我提供了许多的帮助，也与我一起创造了许许多多美好的记忆。我们之间的友谊终生难忘，祝愿你们前程似锦。

最后，再次感谢那些帮助我的，关心我的人，谢谢你们！

卢澄志

于浙江大学软件学院

2018 年 7 月 18 日