

学校代码: 10286

分 类 号: TP311

密 级: 公开

U D C: 004.6

学 号: 184607



东南大学

# 工程硕士学位论文

## 基于知识图谱的 IT 运维辅助系统的 设计与分析

研究生姓名: 方苏东

导师姓名: 漆桂林 教授

丁岩 高工

申请学位类别 工程硕士 学位授予单位 东南大学

一级学科名称 软件工程 论文答辩日期 2021 年 5 月 10 日

二级学科名称 软件工程 学位授予日期 2021 年 5 月 10 日

答辩委员会主席 \_\_\_\_\_ 评 阅 人 \_\_\_\_\_

2021 年 5 月 10 日



東南大學  
碩士學位論文

基于知識圖譜的 IT 運維輔助系統的設  
計與分析

專業名稱: 軟件工程

研究生姓名: 方蘇東

導師姓名: 漆桂林教授

丁岩高工



# DESIGN AND ANALYSIS OF IT OPERATION AND MAINTENANCE ASSISTANT SYSTEM BASED ON KNOWLEDGE GRAPH

A Thesis submitted to  
Southeast University  
For the Professional Degree of Master of Engineering

BY  
FANG Sudong

Supervised by:  
Prof. QI Guilin  
and  
Ir. DING Yan

SouthEast University  
Southeast University

2021/5/10



## 东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，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，也不包含为获得东南大学或其它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了谢意。

研究生签名: \_\_\_\_\_ 日期: \_\_\_\_\_

## 东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名: \_\_\_\_\_ 导师签名: \_\_\_\_\_ 日期: \_\_\_\_\_



## 摘要

随着互联网技术的迅猛发展，目前大多数应用软件都建立在一个庞大、繁杂、跨协议层的大型分布式集群中。这类分布式集群的技术、软件、配置通常会不断地演变，难以避免会发生故障。面对海量的监控数据和庞大的系统，IT（Information Technology）运维人员很难做出迅速、准确的运维决策来预防解决各种故障。智能运维（Artificial Intelligence for IT Operations, AIOps）旨在借助人工智能技术辅助IT运维人员提升运维效率。

已有的AIOps方案均停留在算法层面，仅使用历史数据训练机器学习模型，却忽略了历史数据中蕴含的运维知识。这些运维知识，不仅能够解释故障产生过程，还具有通用性能够指导运维。知识图谱可以结构化整合这些历史数据中的知识，进一步辅助线上诊断、故障预测等运维工作。

本文主要研究了基于知识图谱的IT运维技术，主要工作如下：

(1) 提出了自动化构建组件-事件知识图谱的方法，利用机器学习模型发掘事件间因果关系，从历史数据沉淀出了与故障类型对应的组件-事件知识图谱，减少了知识图谱构建的人工消耗。

(2) 提出了组件-事件知识图谱的表示学习模型，考虑了实体在不同上下文中含义不同，把实体表示分为了语义表示和结构表示，实现了实体随上下文动态地表示，在组件-事件知识图谱三元组分类和链接预测任务上取得了最好的效果。

(3) 提出了引入组件-事件知识图谱的故障预测模型，利用各类故障对应的知识图谱识别事件序列中的关键信息，把最匹配事件序列的知识图谱作为预测结果，提高了预测结果的细粒度，增强了可解释性。

(4) 基于上述工作，设计并实现了一个基于知识图谱的IT运维辅助系统。

**关键词：** 故障预测，IT运维，知识图谱，表示学习



## Abstract

With the rapid development of internet technology, most applications are built in a large, complicated and distributed cluster across protocol layers. The technology, software and configuration of this kind of distributed cluster are always evolving, and it is difficult to avoid failures. Faced with massive monitoring data and huge systems, it is difficult for IT (Information Technology) operation and maintenance personnel to make quick and accurate operation and maintenance decisions to prevent and solve various failures. Artificial intelligence for IT operations (AIOps) aims to assist IT operation and maintenance personnel to improve operation and maintenance efficiency by means of artificial intelligence technology.

Existing AIOps schemes all stay at the algorithm level, only use historical data to train machine learning models, but ignore the operation and maintenance knowledge contained in historical data. This knowledge of operation and maintenance can not only explain the process of fault generation, but also be universal and guide operation and maintenance. Knowledge graph can integrate the knowledge in these historical data structurally, and then further assist the operation and maintenance work such as online diagnosis and fault prediction.

This paper mainly studies the IT operation and maintenance technology based on knowledge graph. The main work is as follows:

(1) The method of automatically constructing component-event knowledge graph is proposed, and the causal relationship between events is discovered by machine learning model, and the component-event knowledge graph corresponding to fault types is precipitated from historical data, which reduces the labor consumption of constructing knowledge graph.

(2) A representation learning model of component-event knowledge graph is proposed. Considering the different meanings of entities in different contexts, entity representation is divided into semantic representation and structural representation, which realizes the dynamic representation of entities with context, and achieves the best results in the task of triple classification and link prediction of component-event knowledge graph.

(3) A fault prediction model with component-event knowledge graph is proposed, which uses the knowledge graphs corresponding to various faults to identify the key information in the event sequence, and takes the knowledge graph that best matches the event sequence as the prediction result, thus improving the fine granularity of the prediction result and enhancing the interpretability.

(4) Based on the above work, an IT operation and maintenance assistant system based on knowledge graph is designed and implemented.

**Keywords:** Failure Prediction, IT operation and maintenance, Knowledge Graph, Knowledge Graph Embedding

# 目录

摘要 . . . . .	I
<b>Abstract . . . . .</b>	<b>III</b>
<b>第一章 绪论 . . . . .</b>	<b>1</b>
1.1 研究背景 . . . . .	1
1.2 研究现状 . . . . .	2
1.2.1 运行状态监测模型研究现状 . . . . .	2
1.2.2 知识表示学习研究现状 . . . . .	3
1.2.3 故障预测研究现状 . . . . .	3
1.3 论文研究内容 . . . . .	4
1.4 论文组织架构 . . . . .	5
<b>第二章 组件-事件知识图谱构建 . . . . .</b>	<b>6</b>
2.1 组件拓扑关系获取 . . . . .	6
2.2 事件因果关系获取 . . . . .	8
2.2.1 事件生成 . . . . .	8
2.2.2 事件因果关系挖掘 . . . . .	10
2.3 组件-事件知识图谱生成 . . . . .	11
2.4 本章小结 . . . . .	13
<b>第三章 组件-事件知识图谱表示学习 . . . . .</b>	<b>15</b>
3.1 场景分析 . . . . .	15
3.1.1 上下文背景信息 . . . . .	15
3.1.2 动态嵌入表示 . . . . .	16
3.2 基于图卷积网络的上下文图结构嵌入算法 . . . . .	17
3.2.1 模型结构 . . . . .	17
3.2.2 模型训练 . . . . .	18
3.3 组件-事件知识图谱表示学习模型 . . . . .	19
3.3.1 模型结构 . . . . .	20
3.3.2 参数学习 . . . . .	22
3.3.3 预测 . . . . .	22
3.4 本章小结 . . . . .	22
<b>第四章 基于知识图谱的故障预测 . . . . .</b>	<b>23</b>
4.1 场景分析 . . . . .	23
4.2 基于记忆网络的故障预测算法 . . . . .	24

---

4.2.1 模型结构 . . . . .	24
4.2.2 模型训练 . . . . .	26
4.3 基于记忆网路和知识图谱的故障预测模型 . . . . .	26
4.3.1 模型结构 . . . . .	26
4.3.2 参数学习 . . . . .	28
4.3.3 预测 . . . . .	28
4.4 本章小结 . . . . .	28
<b>第五章 实验与评估 . . . . .</b>	<b>29</b>
5.1 数据集构建 . . . . .	29
5.2 事件因果关系判别实验 . . . . .	32
5.3 组件-事件知识图谱表示学习实验 . . . . .	32
5.3.1 数据标注 . . . . .	32
5.3.2 评测标准 . . . . .	34
5.3.3 实验方法 . . . . .	34
5.3.4 实验结果与分析 . . . . .	35
5.4 基于知识图谱与实时事件序列的故障预测实验 . . . . .	37
5.4.1 数据标注 . . . . .	37
5.4.2 评测标准 . . . . .	37
5.4.3 实验方法 . . . . .	37
5.4.4 实验结果与分析 . . . . .	38
5.5 本章小结 . . . . .	39
<b>第六章 系统设计与实现 . . . . .</b>	<b>40</b>
6.1 系统框架设计 . . . . .	40
6.1.1 系统框架 . . . . .	40
6.1.2 系统功能 . . . . .	40
6.2 数据获取 . . . . .	42
6.2.1 集群拓扑结构 . . . . .	42
6.2.2 微服务拓扑结构 . . . . .	42
6.2.3 日志数据 . . . . .	44
6.2.4 指标时序数据 . . . . .	44
6.3 系统实现和功能展示 . . . . .	46
6.3.1 系统实现 . . . . .	46
6.3.2 功能展示 . . . . .	47
6.4 本章小结 . . . . .	50
<b>第七章 总结与展望 . . . . .</b>	<b>51</b>
7.1 本文工作总结 . . . . .	51
7.2 未来工作展望 . . . . .	51
<b>致谢 . . . . .</b>	<b>52</b>

参考文献 . . . . .	53
作者攻读硕士学位期间的研究成果 . . . . .	57

# 第一章 绪论

## 1.1 研究背景

近年来，由于云计算灵活、便捷、可扩展的特性，越来越多电子商务、社交网络、金融、医药等领域的企业都选择将其服务应用部署于云计算环境中。在云计算环境中，这些服务应用会被部署在复杂庞大、跨协议的分布式集群上。随着分布式集群中的软硬件变迁、技术升级、配置更新等，各种故障时有发生，如资源争用死锁、软件运行错误、硬件故障等。另外，分布式集群结构的复杂性和资源的共享关联性会形成阻力，导致运维人员很难快速准确地发掘、分析分布式集群故障数据，进而导致故障未被及时发现并进一步扩散，最终导致应用不可用，造成巨大的经济损失。例如，云提供商 AmazonWeb Service (AWS) 在 2015 年 9 月 20 日上午 2:13 到 7:10 之间发生了一次服务宕机，当时用户无法访问 Netflix, Tinder, Airbnb, Reddit 和 IMDb 等流行的在线服务，失败的根源在于美国东部地区亚马逊 DynamoDB 服务的读写操作异常。

因此，如何借助人工智能技术辅助运维人员自动检测、分析、预测故障成为了研究的热点及难点。近些年被提出的 AIOps 就尝试将人工智能技术与运维相结合，通过机器学习的方法来提升运维效率。但大量已有 AIOps 方案均使用历史数据训练各种机器学习模型，都只停留在算法层面，忽略了历史数据中蕴含的运维知识。这些历史数据中的运维知识，不仅能够解释故障产生过程，还具有通用性能够指导运维。

分布式集群中可用的数据是庞大且多样的，有组件间的拓扑关系、硬件的指标时序数据、软件的运行日志数据，如图1-1所示为分布式集群信息关系示意图。组件间的拓扑关系指明了组件间的交互、依赖等关系，如 SLB 会调控多台 ECS 实现负载均衡；硬件的指标时序数据记录了硬件的内存占用率、读写速度、CPU 负载率等随时间的变化曲线；软件的运行日志数据记录了各个微服务在具体时间戳打印的日志内容。分布式集群中的这些数据不仅能用来训练模型以解决某个检测、分类或预测的子任务，还能将其中隐含的运维知识沉淀为结构化的知识图谱。生成的知识图谱对于云服务提供商，可以为客户提供更稳定的服务以增加收益；对于运维人员，可以更直观，更有效地管理分布式集群减少人力消耗，并做出可解释、可靠的运维决策；对于云应用程序开发人员，可以减少应用程序部署，迭代和发布的阻力。

基于以上分析，首先本文分析了现有的分布式集群运行状态监测方案，发现这些方案仅使用了分布式集群部分信息且挖掘的数据特征有限，导致构建的监测拓扑图比较片面且其构建过程需要大量人工。本文收集了横跨软硬件的所有信息，并设计了 6 种数据特征，构建了组件-事件知识图谱。其次，本文针对云计算场景下异常沿组件图传播、事件发生与上下文相关的特性，提出了组件-事件知识图谱实体的动态表示方案。最后，本文用双向记忆网络编码事件序列，再用组件-事件知识图谱识别关键信息，从而将运维知识引入到故障预测中，增强了预测结果的可靠性与可解释性。

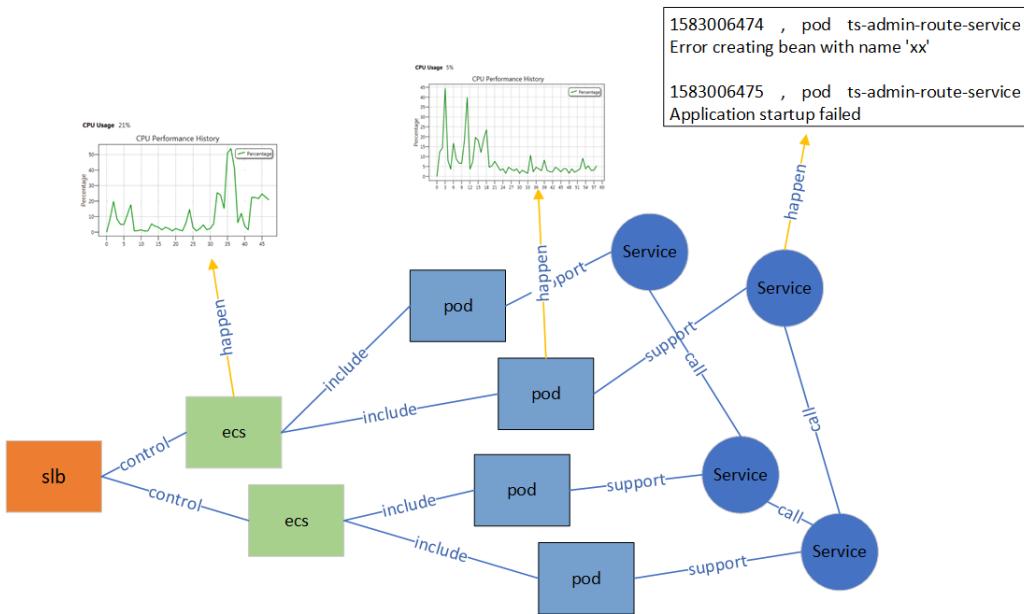


图 1-1 分布式集群信息关系示意图

## 1.2 研究现状

针对分布式集群信息多样繁杂不易整合、运维知识表示不足、故障难以预知等问题，本文主要从分布式集群运行状态监测模型、知识表示学习和故障预测三方面展开了调研。

### 1.2.1 运行状态监测模型研究现状

有效的分布式集群运行状态监测模型可以全面完善地获取、表示系统运行状态。高细粒度的运行状态数据是展开进一步运维工作的基础。

已有的模型分为基于系统拓扑图的方法、基于事件因果图的方法和基于运维知识图谱的方法。在基于系统拓扑图的方法中，eBay 提出的 GRANO<sup>[1]</sup> 通过构建组件拓扑图，捕获警报信息和程序运行状态，实现了用于云计算分布式数据平台的端到端状态检测分析系统。具体实现上，GRANO 首先处理大量指标时序数据来检测硬件和软件系统组件的异常信息，随后让专家利用领域知识给异常信息打上严重分数，然后在组件拓扑图上使用传播算法对各个组件的严重分数进行更新，最终辅助运维人员通过交互界面沿着系统拓扑图查看严重的系统组件。这种基于系统拓扑图的方法虽可以有效监视和识别系统异常，但不能沉淀出运维知识，也不能给出可靠的异常触发链。在基于事件因果图的方法中，文献 [2] 首先利用异常检测方法<sup>[3]</sup> 获取各个系统组件实时运行发生的异常事件，随后借助有监督的机器学习<sup>[4]</sup> 构建事件间可靠的因果关系，最终形成可辅助运维的事件因果图。这种基于事件因果图的方法不需要了解云计算场景中应用的设计和实现细节，也不需要检测服务的源代码，但只使用概率特征的因果发掘模型需要领域专家循环标注海量数据才可以达到良好的效果，且缺乏对系统组件拓扑关系的有效利用。在基于运维知识图谱的方法中，文献 [5] 根据组件间的访问与部署关系、指标与组件间的来源产生关系，构建了运维知识图谱，但对于异常信息间的触发关系，该图谱只能精确到指标时序类型之间的因果关系，如“Container 1 CPU usage” 导致了“Microservice 1 Response Time”<sup>[5]</sup>，不能进一步知道什么样的 CPU 变化导致了什么样的响应时间变化。

可见始终缺乏一个横跨硬件、软件、异常数据，并能在高细粒度上将异常信息触发链和

系统状态作为知识沉淀下来的运行状态监测模型。

### 1.2.2 知识表示学习研究现状

在运行状态监测模型沉淀出知识后，需要利用知识表示学习将运维知识转为计算机可理解的数据形式，即利用计算机可以处理的实值向量来表示知识实体和关系。运维知识被转化为实值向量后，可以通过计算机引入到下游任务中（如故障预测），有助于实现自动化运维。

知识表示学习方面，已经存在着大量工作。基于距离的模型中，文献 [6] 只利用了头实体和尾实体的共现信息来进行表示学习；在此基础上文献 [7] 将实体关系建模为两个分别针对头尾实体的矩阵，从而引入了关系信息。从翻译视角展开工作的模型中，最开始 TranE<sup>[8]</sup> 将每个知识三元组中的关系视作头实体到尾实体的翻译过程，但它不能处理复杂的一对多、多对多关系；TransH<sup>[9]</sup> 选择将头尾实体都映射到关系所在的超平面上，解决了复杂关系的问题；TransR<sup>[10]</sup> 认为每一种关系间是有区别的，应该对应着不同的语义向量空间，所以其将关系嵌入为矩阵；TransD<sup>[11]</sup> 为了克服同一关系头尾实体种类属性差别过大的问题，将每个对象（实体、关系）嵌入为语义向量和映射向量这两个向量；TranSparse<sup>[12]</sup> 根据关系连接的头尾实体对数量，自适应的调整映射矩阵的稀疏程度；TransF<sup>[13]</sup> 放宽了损失函数的约束条件，只要求头实体加关系后的张量与尾实体关系一致；TransG<sup>[14]</sup> 将实体的不确定度、关系的多语义用高斯分布协方差来表示。基于语义匹配的模型中，LFM<sup>[15]</sup> 使用关系的双线性变换，刻画实体和关系之间的二阶联系；ComplEx<sup>[16]</sup> 为了建模非对称关系将表示扩展到复数向量空间；ANALOGY<sup>[17]</sup> 将知识图谱中的类比关系进行了建模；SLM<sup>[18]</sup> 开始引入神经网络，其利用标准非线性单层神经网络来连接实体；NTN<sup>[18]</sup> 使用张量网络捕获头尾实体间的语义关联；ConvE<sup>[19]</sup> 为了捕获实体间的语义关系，使用了卷积网络。融入多源信息的模型也有很多，SSE<sup>[20]</sup> 和 TKRL<sup>[21]</sup> 将实体的类别信息引入了知识表示学习中；文献 [22] 为了引入关系路径将路径上所有的关系向量进行了组合以表示路径向量；为了引入文本描述，文献 [18] 将语料库训练的词向量累加平均作为实体的初始向量；文献 [21] 给每个实体赋予了基于结构和基于文本描述的两个表示；文献 [23] 选择将实体与文本库词汇对齐；GAKE<sup>[24]</sup> 和 TCE<sup>[25]</sup> 则引入了实体的上下文信息，包括邻居上下文、边上下文、路径上下文等。文献 [26] 引入图卷积网络的同时区分了不同关系对节点的影响，利用图信息传播机制编码知识图谱以获取实体表示。

虽然知识表示学习方面已经存在了很多不同角度展开的工作，但都只能得到实体、关系的静态表示向量。在云计算场景下的组件-事件知识图谱中，实体会随着上下文场景变化有不同的含义，需要动态地表示。

### 1.2.3 故障预测研究现状

故障预测是一种主动预防故障的方法，可以在故障发生前进行预测，从而告知运维人员采取措施防止故障发生，避免故障造成的损失。

文献 [27] 已经对在线故障预测技术进行了广泛地调研。其将现有的故障预测方法宽泛地分为两类：分类方法和函数逼近方法。基于分类的故障预测方法<sup>[28-31]</sup> 首先构建了容易发生故障和不容易发生故障的数据样本，随后在这些数据上训练分类器。分类器的决策边界通常来自于参考数据集，这个数据集每个数据点的决策边界都是已知的，即明确地知道它指示的是容易发生故障还是不容易发生故障。在进行在线故障预测时，只需要检查当前监视值位于决

策边界的哪一侧即可。函数逼近是一个广泛应用于各种科学领域的术语，可以用在故障预测任务的原因在于该任务可以被视作发掘一种函数映射关系，即被监视的系统状态（函数的输入）到系统未来是否会有故障（函数的输出）之间的未知函数关系。逻辑回归技术就是为目标性能值建立曲线拟合函数，并调整参数以最佳拟合训练数据集，其输入数据一般是性能指标时序数据<sup>[27]</sup>。文献[32]提出了一种基于统计模型的流量预测方法，它将滑动窗口内的观测值用泊松分布加权。文献[33]提出了一种事件感知策略，通过利用与计划事件相关的先验知识，可以更有效地预测系统工作时的负载突发。事件感知预测方法可以非常有效地预测故障，但无法给出故障出现的触发过程。文献[34]使用了基于隐马尔可夫模型的方法来推断受监控组件的状态是否健康。虽然作者文中未提及该方法可用于故障预测，但可以通过以下方式进行故障预测：假设系统中存在导致未来故障的错误状态，而其他错误状态不会导致未来的故障，所提出的隐马尔可夫模型方法就可以用来确定（分类）故障是否即将发生。文献[35]指出，可以通过识别系统正在经历的状态来预测系统出现故障的概率，但作者并没有给出详细的算法和实验结果。文献[36-41]开始使用深度学习方法，如循环神经网络（RNN, Recurrent Neural Networks）和长短期记忆神经网络（LSTM, Long Short-Term Memory）进行故障预测。文献[42]使用双向的长短期记忆神经网络（Bi-directional Long Short-Term Memory, BiLSTM）编码时序信息，并对数据赋予了不同权重，在预测是否会发生故障时取得了最佳效果。

从以上调研结果可见，目前的故障预测工作专注于预测是否会有故障发生，不能预测到具体会有何种故障。另外，由于预测过程均没有引入运维知识，预测结果的可解释性不高。

### 1.3 论文研究内容

基于上文对已有相关工作的分析，本文主要开展了以下工作：

- (1) 设计了横跨硬件、软件、异常数据的组件-事件知识图谱自动构建方案。一方面该知识图谱包含了高细粒度的多种信息，如集群中的服务器、负载均衡器、Pod、微服务等软硬组件关系，还有集群运行时产生的指标时序数据、日志信息所转化生成的事件。另一方面本文设计了共计6种事件特征，使用机器学习模型自动判别事件间是否存在因果关系，最终从历史数据中沉淀出了故障类型对应的组件-事件知识图谱。
- (2) 设计了针对组件-事件知识图谱的知识表示学习模型。为了表示学习(1)构建的组件-事件知识图谱，考虑了异常会沿着组件拓扑关系传递，且事件间有因果触发关系，另外实体在不同的上下文中有不同的含义。本文将实体表示分为语义表示和结构表示两部分，实现了实体随上下文变化的动态表示。
- (3) 设计了基于组件-事件知识图谱的故障预测模型。本文使用双向记忆网络编码实时产生的事件序列，并与组件-事件知识库中每个故障类型对应的组件-事件知识图谱做注意力机制匹配度评分，并按照评分由高至低的顺序输出，作为故障预测结果。通过引入含有运维知识的知识图谱，本文的故障预测结果具备了可解释性。IT 运维人员也可以根据知识图谱中的关键结点，更细粒度的分析系统状态。
- (4) 设计了面向 IT 运维的可视化查询分析及故障预测系统。基于以上的研究，本文设计了基于知识图谱的 IT 运维辅助系统，并完成了该系统的开发。最终，运维人员可以通过

该系统实时查看系统运行状态（包括属性信息、指标时序信息、事件信息和拓扑结构），并由故障预测结果及时采取防范措施，降低了运维难度，提高了运维效率。

## 1.4 论文组织架构

本文篇幅结构共有七章，每一章内容大体如下：

第一章为绪论，首先梳理了相关研究背景和研究现状，然后分点描述了本文研究要点。

第二章为组件-事件知识图谱构建，主要介绍了系统运行状态信息获取处理流程，事件因果关系挖掘和历史数据沉淀生成组件-事件知识图谱的过程。

第三章为组件-事件知识图谱的知识表示学习，主要分析场景特性，介绍随上下文变化的动态表示学习方案。

第四章为引入组件-事件知识图谱的故障预测，主要介绍引入组件-事件知识图谱，结合实时事件序列做出可解释故障预测的方法。

第五章为实验与评估，依次介绍了针对上述三章工作开展的实验，包括实验数据构建、实验评测指标和实验结果分析。

第六章为系统设计，主要介绍基于本文方法实现了一个基于知识图谱的IT运维辅助系统，并详细介绍该系统的设计和实现细节。

第七章为总结与展望，对本文完成的工作进行了总结归纳，也对未来的工作进行了规划。

## 第二章 组件-事件知识图谱构建

本章主要介绍了组件-事件知识图谱的构建流程。目前已有的分布式集群运行状态监测模型都不能涵盖所有种类的信息。文献 [1] 只构建了组件拓扑图而忽略了组件上数据之间的因果关系；文献 [2] 将运行时信息转为事件数据后，只关注了事件间的因果图而忽略事件所在组件之间的拓扑关系；文献 [5] 尝试构建运维知识图谱，但所构建的知识图谱只包含组件间的访问与部署关系、指标与组件间的来源产生关系，对于异常信息间的触发关系，只能精确到指标时序类型之间的因果关系，如“Container 1 CPU usage” 导致了“Microservice 1 Response Time”，不能进一步确定是什么样的 CPU 变化导致了什么样的响应时间变化。本文在构建组件-事件知识图谱时，首先通过公开 API（Application Programming Interface）、调用链追踪系统获取组件唯一标识符、属性和关联关系等信息；其次将日志、指标时序数据等运行时信息转为了事件类型数据，然后设计了共计 6 种事件特征用于发掘事件间因果关系；最后将同故障类型的组件-事件拓扑图整合起来，自动沉淀出了标有对应故障类型的组件-事件知识图谱。本章分别介绍了组件拓扑关系获取方式，事件因果关系获取方式和组件-事件知识图谱生成方式。

### 2.1 组件拓扑关系获取

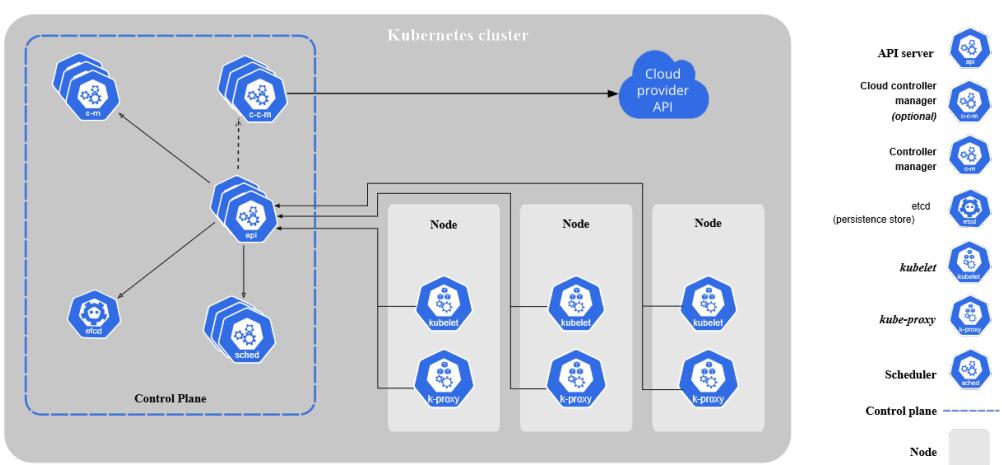


图 2-1 Kubernetes 集群结构示意图

Kubernetes 集群<sup>[43]</sup>在云计算行业中已经被广泛使用，它是一个具有扩展性、移植性的开源平台，同时具有着清晰的声明式配置和自动化特性，被用于管理各种容器化工作负载和服务。Kubernetes 集群由一组工作机器（称为节点）组成，这些节点上运行着微服务，但为微服务并不是直接运行在这些节点上，而是由节点中承载的 Pod 来负载这些微服务。在生产环境中，集群通常运行着多个节点，从而提供容错性和高可用性。图 2-1 为一个 Kubernetes 集群示意图，可见所有组件都被绑定在一起，其间存在着复杂的负载、交互等关系。

本文所使用的 Kubernetes 集群是阿里云针对企业级应用所优化的容器服务版 Kubernetes

集群，其管理容器化应用的能力具有伸缩性，性能较高。阿里云对其提供的 Kubernetes 集群中各类组件都进行了清晰定义，具体上各类组件类型及含义如表2-1所示。

表 2-1 集群组件信息

组件名称	含义
VPC	专有网络（Virtual Private Cloud, VPC）是自定义私有网络。不同的专有网络之间存在二层逻辑隔离。用户可以在自己创建的专有网络内创建和管理各个云产品实例，比如 ECS、SLB、RDS 等。
SLB	负载均衡器（Server Load Balancer, SLB）是负责对多台应用实例进行流量分发的负载均衡服务。用户可以通过分流提高系统的服务能力，也可以通过消除单点故障使其具有可用性。
ECS	云服务器（Elastic Compute Service, ECS）是一种简单高效、具有弹性处理能力的计算服务。云服务器可以辅助用户高效率构建安全性、稳定性高的应用。
Pod	Pod 是 Kubernetes 中最小的部署单元和计费单位，根据应用场景，可以由一个或多个容器组成。当一个 Pod 中有多个容器时，这些容器会共享 Pod 的计算资源、存储空间、IP 和端口。对于计算资源，Pod 还可以限制各个容器使用的比例。
Container	Container 即为容器，它包含在 Pod 中。一个 Pod 中有一个 Pause 容器和若干个业务容器。
Service	微服务是一种云原生架构方法，其中单个应用程序由许多松散耦合且可独立部署的较小 Service 组成。

为了获取处于运行状态的 Kubernetes 集群中各个组件实例的唯一标识、属性和其间的关联关系，本文使用了云服务商提供的公开 API 进行数据的请求，如获取某区域 ECS 信息的 API 为 `DescribeInstances()` 方法接口，输入地理区域 “cn-hangzhou”，就会返回该区域的 ECS 信息。图2-2为返回的杭州区域的一台 ECS 的信息列表，包含该 ECS 的资源组、内存、唯一标识符、创建事件等属性信息，以及所属 VPC 的 Id 信息。除了 Service 类型组件，其它集群组件均可通过 API 获取其属性信息及与其他组件的关联信息。

Service 类型组件对应着部署在 Kubernetes 集群中的分布式应用的各个微服务。微服务之间的拓扑关系与开发人员编写的应用架构结构有关，无法通过云服务商提供的公开 API 直接获取。分布式应用代码是其所属公司的保密性数据，不能被外部获取或改写，所以通过侵入代码获取微服务结构的方法是不可行的。目前开放式的微服务追踪系统（如 Jaeger<sup>[44]</sup>），可以跟踪微服务之间的请求数据，然后分析汇总这些请求链路就可以构建出微服务之间的访问拓扑关系。追踪请求数据生成微服务拓扑关系的方法不必侵入代码，也不需要运维人员阅读分布式应用的开发文档，只需要将追踪系统部署在分布式应用所在的集群中跟踪分布式应用即可。因此，本文通过云服务商提供的公开 API 和分布式请求跟踪系统，获取了组件部分的拓扑结构及实体属性。

```

{
    "Instances": [
        "Instance": [
            {
                "ResourceGroupId": "rg-bp67acfmxazb4p****",
                "Memory": 16384,
                "OSName": "CentOS 7.4 64 位",
                "InstanceNetworkType": "vpc",
                "InstanceId": "i-bp67acfmxazb4p****",
                "StoppedMode": "KeepCharging",
                "CpuOptions": {
                    "ThreadsPerCore": 2,
                    "Numa": "2",
                    "CoreCount": 4
                },
                "StartTime": "2017-12-10T04:04Z",
                "DeletionProtection": false,
                "VpcAttributes": {
                    "PrivateIpAddress": {
                        "IpAddress": [
                            "172.17.*.*"
                        ]
                    },
                    "VpcId": "vpc-2zeuphj08tt7q3brd****",
                    "VSwitchId": "vsw-2zeh0rlpabwtg6wcs****",
                    "NatIpAddress": "172.17.*.*"
                }
            }
        ],
        "TotalCount": 1,
        "RequestId": "473469C7-AA6F-4DC5-B3DB-A3DC0DE3C83E",
        "PageSize": 10,
        "PageNumber": 1,
        "NextToken": "caeba0bbb2be03f84eb48b699f0a4883"
    }
}

```

图 2-2 阿里云公开 API 返回 ECS 数据部分示例

## 2.2 事件因果关系获取

本小结旨在将集群运行状态数据经过清洗整理后转为事件信息，并进一步经过关系分类器挖掘事件间因果关系。在事件生成部分，使用了聚类、异常识别、模板匹配结合的方式，将实时产生的日志、指标时序数据转为了事件类数据。在事件因果关系挖掘部分，为了克服过往工作中只存在基于概率的事件特征，导致领域专家需要反复标注海量数据训练模型才能达到良好效果的问题，本文设计了 6 种事件特征，最终不需要反复标注海量数据就可以达到优质的效果。

### 2.2.1 事件生成

**Definition 1 (事件).** 事件。事件是对分布式系统中组件运行状态的描述。主要包括五个部分：

*Id*: 事件的唯一标识；

*Time*: 事件发生的时间戳；

*Location*: 事件发生所在的组件；

*EventType*: 事件类型描述；

*Detail*: 事件的具体描述

**Definition 2 (完全周期性事件).** 完全周期性事件是严格按照一定时间间隔出现的事件，如每隔 3s 必然出现一次。

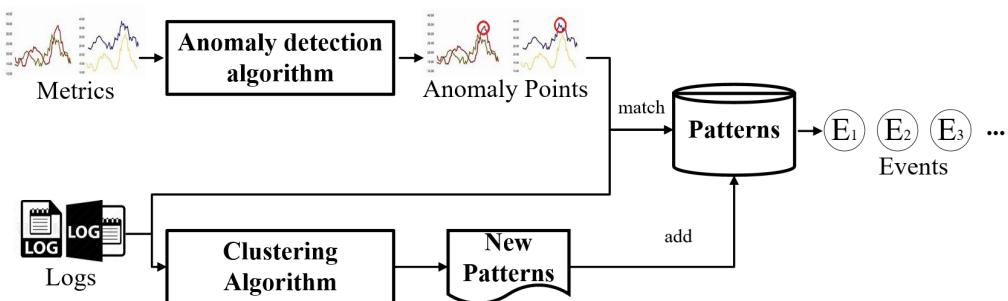


图 2-3 事件生成流程

事件定义如 Definition 1 所示。在生成事件之前，需要先收集到指标时序数据、日志数据，本文部署了云计算相关工作常用的两个开源应用 train-ticket<sup>[45]</sup>、sock-shop<sup>[46]</sup>于阿里云平台上，随后分别多次模拟了常见的故障，并收集了两个应用正常数据以及各个故障场景的异常数据。具体数据收集情况会在实验部分详细描述。

在收集完相关原始数据后，事件生成模块的主要功能是将系统的异常信息转为统一规范的事件数据，整体流程如图2-3所示。系统中的数据主要有两种，一种是指标时序数据，记录着系统各个组件的状态曲线变换，如内存、读写、CPU 等随时间的变化曲线；另一种是日志数据，记录着微服务在具体时间戳打印的日志内容，如在某个时间微服务打印“xxx service strats success”日志。对于指标时序数据，本文首先使用异常检测算法检测到异常点，比如某些指标开始很平滑，然后经过某个点后开始抖动，那么这个转折点就是异常点，异常点一般是系统发生异常的一种表现。在找到异常点后，经过模板匹配就可以得到异常事件。比如某 Container 的 CPU 使用率曲线突然上升到 100%，那么根据模板可以得到这个异常点事件类型是“container.cpu.usage.seconds.total”。对于日志数据，一方面会使用聚类算法发掘日志中的通用模板，如果有新模板产生，那么就会添加到模板库中；另一方面会使用模板库中的模板来匹配日志生成相应事件。图2-4中是一些日志数据，这些日志由于格式比较类似，相同的关键词也比较多，因此会被聚类算法分为一类，这一类的模板就会写为“\*\*\*\*\_\*\*-\* \*.\*.\*.\*.\*”，pod \*\*\* occurs Back-off restarting failed”，随后这个模板会被添加到模板库中，之后产生的日志匹配到此类模板会生成相应的事件。上文提到的异常检测算法<sup>[47]</sup>和聚类算法<sup>[48]</sup>目前业界已有很多，不是本文主要研究内容，所以这里就不展开对这方面的细致介绍。

```

1 2020-02-19 10:15:01, pod ts-user-service-7687c964f-dhwln  occurs Back-off restarting
failed container , event level Warning
2 2020-02-19 15:25:22, pod ts-user-service-7687c964f-dhwln  occurs Back-off restarting
failed container , event level Warning
3 2020-02-20 13:22:21, pod ts-food-map-service-b7977c997-42nmd  occurs Back-off restarting
failed container , event level Warning
4 ...

```

图 2-4 日志数据样例

本文生成的事件按照严重等级可以划分如表2-2所示，包括正常事件、异常事件和故障事件。正常事件是指系统内组件在正常业务逻辑中产生的事件，注意正常事件也有可能引发故障，比如事件“服务器正在重启”，虽然此事件是服务器正常重启产生的事件，但是如果在此服务器重启时，其他服务器对其访问，也会产生“访问失败”事件，触发系统故障。异常事件是指系统组件发生了正常执行业务时不会出现的事件，如异常检测算法检测到的系统异于

常态的事件，但异常事件发生时组件仍可工作，不会发生系统不可用。故障事件是描述组件故障的事件，故障事件发生时系统部分服务已无法正常工作。

表 2-2 事件严重程度划分结果

初级划分	次级划分	示例
正常事件	硬件正常事件	container memory 50%、container cpu 50%、...
	软件正常事件	数据库连接成功、...
异常事件	硬件异常事件	container memory 100%、container cpu 100%、...
	软件异常事件	数据库连接失败、...
故障事件	硬件故障事件	服务器宕机、...
	软件故障事件	HttpServerErrorException 503、 HttpServerErrorException 500、...

此外，在生成的事件中存在大量的完全周期性事件，其定义如 Definition 2 所示。完全周期性事件的存在，一方面使得事件数量级抖升而增加了时空复杂度，一方面会干扰后续事件因果分析、故障预测的效果。完全周期性事件不管集群有无异常都会保持绝对的周期出现，与集群状态无关。因此，在生成事件后，需要初步筛选去除具有完全周期性的事件，本文使用 DTW (Dynamic Time Warping) 算法<sup>[49]</sup> 判断事件是否为完全周期性事件。采用 DTW 算法的原因在于该算法已经在语音匹配任务中取得了优秀的效果，它不需要曲线精确的对位分析，能适应时间错位性。利用该算法进行去除完全周期性噪声事件的具体做法为：

(1) 统计某类事件  $e$  在异常注入前  $t_1$  时间段、异常到故障产生时间段  $t_2$  和故障发生后  $t_3$  时间段这三段时间内，发生的频次曲线。其中  $t_1 = t_2 = t_3$ 。

(2) 对比三段时间  $t_1$ 、 $t_2$  和  $t_3$  所对应的三条频次曲线。如果 DTW 算法判别三者等同，则认为该类事件是完全周期性事件。

(3) 对异常注入到故障产生时间段内的所有事件均采取上述判别过程。最终将认定有完全周期性的事件删除，只保留非完全周期性事件。该过程可以删去总事件量中约 15% 的完全周期性事件，如在总共收集到的 3738 条事件中，删去了 537 条完全周期性事件。

## 2.2.2 事件因果关系挖掘

关系分类器用于判别事件间关系类型，本文中为有无因果关系。输入关系分类器的是 A、B 两事件特征向量，输出为 A 到 B 是否有因果关系，即  $A \rightarrow B$  是否存在。若要判断有无 B 到 A (即  $B \rightarrow A$ ) 的因果关系，需要按照顺序输入 B、A 两事件特征向量。本文共设计了以下 6 种事件特征：

(1) 皮尔逊相关系数：该特征用来计算两个事件之间的相关性，即两个事件之间的皮尔逊相关系数越高，那么这两个事件越相关，越有可能是因果关系。皮尔逊相关系数计算方法如公式 2.1 所示。其中  $X, Y$  是事件  $A, B$  分别在时间维度上展开的向量。当事件发生时为 1，未发生时为 0，事件按照在每个时间戳是否发生，可以表示为 0、1 组成的数字向量。

$$p_{x,y} = \frac{\sum(X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum(X - \bar{X})^2(Y - \bar{Y})^2}} \quad (2.1)$$

(2) 关系置信度：该特征用来统计历史数据中， $A$  事件出现后  $B$  出现的概率。若  $A$  出现后  $B$  出现，则计  $AB$  共同出现一次。计算方式如公式2.2所示。

$$\text{relevant\_confidence}(A, B) = P(B/A) = P(\text{count}(AB)/ \text{count}(A)) \quad (2.2)$$

(3) 事件发生所在设备的距离度量：根据异常沿着组件拓扑传播的特性，两事件发生所在组件位置越近，越可能是因果关系。计算方式如公式2.3所示。其中  $\text{device}(\cdot)$  为找到事件所在设备， $\text{Distance}(\cdot)$  输出两设备距离， $\log(1 + \cdot)$  是为了让取值均大于 0。

$$d(A, B) = \log(1 + \text{Distance}(\text{device}(A), \text{device}(B))) \quad (2.3)$$

(4) 平均时间差：事件发生时间越近，越可能是因果关系。计算事件  $A$  和事件  $B$  时间差时， $A, B$  均可能出现多次，这时需要计算事件  $A$  与  $B$  的平均时间差。具体计算方式如公式2.4所示：统计  $A$  出现的时间戳序列  $A_{\text{stamps}}$ ，统计  $B$  出现的时间戳序列  $B_{\text{stamps}}$ 。统计每当  $A$  出现后  $B$  出现的时间差序列  $\text{intervals}$ 。最终求均值  $\text{mean}(\text{intervals})$  再将取值调整为均大于 0，即可得到  $A, B$  间平均时间差。

$$\text{time\_interval}(A, B) = \log(1 + \text{mean}(B_{\text{stamps}} - A_{\text{stamps}})) \quad (2.4)$$

(5) 事件周期性度量：虽然在事件生成后，本文已经初筛去掉了完全周期性事件，但仍保留了周期性不完美的事件，如每隔 4s 左右会出现大约 2 次。由于周期性事件在故障场景与正常场景中都会以一定的周期性出现，所以具有周期性的事件一般不会引发故障。可见事件周期性越高，它与其他事件的因果关系越弱。计算方式如公式2.5所示：统计事件  $A$  出现的时间戳序列  $A_{\text{stamps}}$ ，随后对事件戳序列差分求标准差即可。计算结果值越小，则事件周期性越强。将  $A, B$  两事件周期性值较小值输入分类器即可。其中  $\text{diff}(\cdot)$  表示求差分， $\text{std}(\cdot)$  表示求标准差。

$$\text{period}(A) = \log(1 + \text{std}(\text{diff}(A_{\text{stamps}}))) \quad (2.5)$$

(6) 事件关键性度量：如果事件常发生在故障时间段内，在系统正常时很少发生，这意味着此事件很可能与故障有关并且是关键事件；反之，更可能为白噪声事件。事件关键性值可以用于衡量事件关键性，计算方式如2.6所示。

$$c_{e_i} = 1 - \left( \frac{\text{count}_{\text{anomal}_{e_i}}}{\text{count}_{\text{anomal}}} \cdot \frac{\text{count}_{\text{normal}_{e_i}}}{\text{count}_{\text{normal}}} \right) \quad (2.6)$$

其中  $c_{e_i}$  表示事件  $e_i$  对故障的关键性， $\text{count}_{\text{anomal}_{e_i}}$  表示事件  $e_i$  在故障发生时不发生的频数， $\text{count}_{\text{anomal}}$  表示故障出现的频数， $\text{count}_{\text{normal}_{e_i}}$  表示不发生故障的情况下事件  $e_i$  发生的频数， $\text{count}_{\text{normal}}$  表示故障不发生的频数。将  $A, B$  两事件关键性值较小值输入分类器即可。

在确定了以上 6 种特征后，本文在实验部分对比了多种分类模型，最终选择了 SVM 分类模型判别事件之间是否有因果关系，具体选用原因在实验部分作详细描述。

## 2.3 组件-事件知识图谱生成

**Definition 3** (抽象事件). 抽象事件是同类型事件规约后的形式。只包含 2 个部分：

*TypeId*: 抽象事件唯一标识；

*EventType*: 抽象事件类型名；

**Definition 4 (组件-事件知识图谱).** 组件-事件知识图谱由各种实体和关系组成，并有对应的故障类型，可表示为  $G = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mathcal{R})$ 。 $\mathcal{F}$  为组件-事件知识图谱所在的故障场景名称。实体集合  $\mathcal{V}$  包含着每个服务器、容器、微服务、负载均衡器等组件实体和抽象事件实体。关系集合  $\mathcal{E}$  包含每条组件实体之间的交互关系、组件实体和抽象事件实体间的产生关系、抽象事件实体之间的因果关系。 $\mathcal{R}$  则表示关系类别集合。

前两小节描述了组件信息、事件信息获取的方式，本小节主要介绍从历史数据中沉淀生成组件-事件知识图谱的过程。首先本文给出了抽象事件的定义3和组件-事件知识图谱的定义4。

在定义4中组件-事件知识图谱的事件层结点是抽象事件，目的在于使得该知识图谱能够与具体时间戳、具体设备无关，而变成一种静态的通用型知识。在借助知识图谱进行运维时，知识图谱作为运维经验知识，只需要提供类似“存储票务信息的 mysql 容器的 cpu 爆满”会导致“订票服务找不到票务信息”的先验知识，而不是“id 为 75945d946-x6hw1 的 mysql 容器 ts-voucher-mysql 在 2019-12-09 22:08:30/1575900510 cpu 爆满”后导致了“id 为 f9d5c946f-4bs5j 的订票服务 ts-order-service 在 22:08:57/1575900537 找不到票务信息”。因为集群中的组件 id 会随着应用状态演变而不断变换，后者只能视作一条历史数据记录，只有前者才能被看作可复用的先验知识。

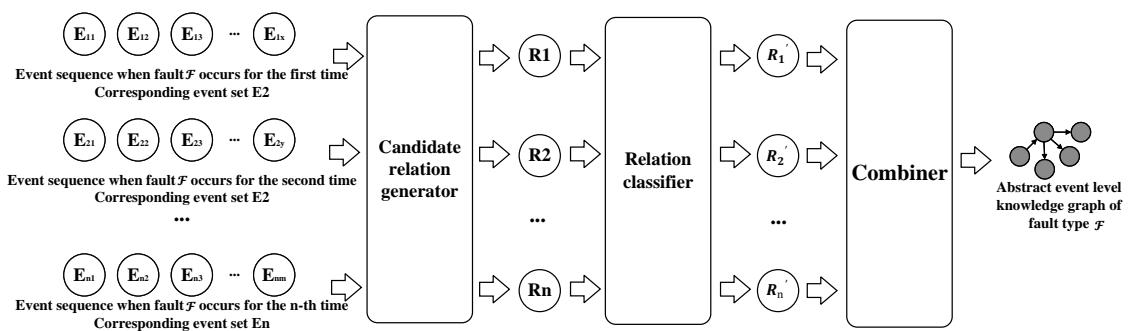


图 2-5 对应故障类型 A 的组件-事件知识图谱生成流程

图2-5是故障类型  $\mathcal{F}$  对应的抽象事件层知识图谱构建流程。下面是故障类型  $\mathcal{F}$  对应组件-事件知识图谱的具体构建步骤描述：

- (1) 生成候选关系集合。获取故障类型  $\mathcal{F}$  发生  $n$  次  $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n$  分别对应时间段内的事件集合  $E_1, E_2, \dots, E_n$ 。针对每个时间段内的事件集合，首先基于 TFIDF<sup>[50]</sup> 筛去部分白噪声事件后将事件按时间顺序前后相连，得到候选关系集合  $R_1, R_2, \dots, R_n$ 。
- (2) 生成因果关系集合。对候选关系集合  $R_1, R_2, \dots, R_n$  分别使用训练好的关系分类器判别该集合中每条候选关系是否为因果关系，仅保留判别为“存在”的因果关系，得到对应关系集合  $R_1', R_2', \dots, R_n'$ 。
- (3) 规约生成抽象事件因果拓扑。对分类器保留后的关系集合  $R_1', R_2', \dots, R_n'$  取并集，融合  $\mathcal{F}$  故障类型各个时间段内的事件因果关系对，其中每个事件都会按照类型归约到抽象事件，从而获得  $\mathcal{F}$  故障的抽象事件因果关系集合  $\mathcal{E}_{cause} = R_1' \cup R_2' \cup \dots \cup R_n'$ 。
- (4) 规约生成组件层拓扑。由于在相同应用上触发的故障，其组件层有着相同的拓扑逻辑结构。只需要将每个故障时间段组件图整合起来即可，具体整合方式为把具体 id 去除，只保

留组件类型。如位于两个时间段的“ts-voucher-mysql-75945d946-x6hw1”和“ts-voucher-mysql-5sdf45sdd-xz45z”可以归约到组件实体“ts-voucher-mysql”。此外，针对“ts-voucher-mysql-75945d946-x6hw1”上发生“22:08:57/1575900537 找不到票务信息”这种组件到事件的关系，在组件-事件知识图谱中该关系会被存为“ts-voucher-mysql”上发生“找不到票务信息”。最后为了区分负载“ts-train-service”的Container，和负载“ts-ticketinfo-service”的Container，本文选择如文献[5]对同类型组件进行序号标注的方式，将上述两个Container分别标注为“container1”，“container2”。

- (5) 生成故障类型  $\mathcal{F}$  对应的组件-事件知识图谱。融合生成的关系对集合  $\mathcal{E}_{cause}$ ，得到对应的有向图即为故障类型  $\mathcal{F}$  对应的抽象事件层知识图谱。再将组件实体与抽象事件实体按产生关系进行连接，就可以得到故障类型  $\mathcal{F}$  对应的组件-事件知识图谱。

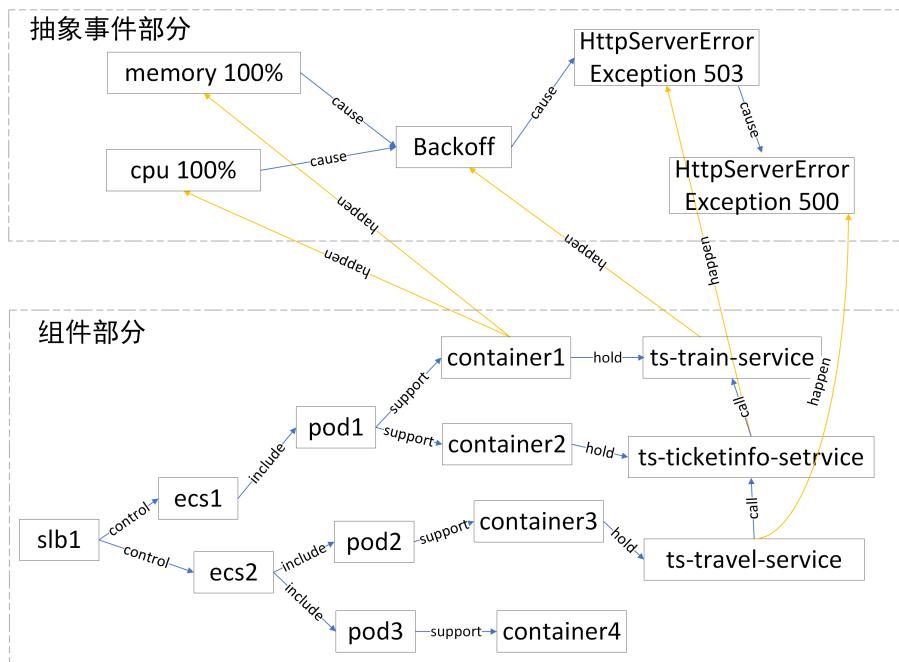


图 2-6 数据库不可用无法查票故障对应知识图谱部分示例

图2-6是故障类型“数据库不可用无法查票”对应的组件-事件知识图谱的一部分。该知识图谱中，组件部分清晰地展示了 ts-train-service, ts-ticketinfo-service, ts-travel-service 分别部署在 3 个 Container 容器上，而这些容器托付在多个 Pod 中。同样的，这些 Pod 被负载在 2 台云服务器 ECS 中，并且这些 ECS 又被一台负载均衡器 SLB 所调控。在抽象事件部分，抽象事件实体被因果关系连接构成了抽象事件图，该图沉淀了 ts-travel-service 发生 HttpServerError Exception 500 故障的触发链知识。另外，抽象事件部分与组件部分被 happen 关系连接，本质上描述了抽象事件在哪类组件上产生。

## 2.4 本章小结

本章主要介绍了组件-事件知识图谱的构建方式。对于组件部分数据，云服务商会提供开放的 API 返回分布式集群含有的组件属性信息、组件数量及每个组件间的关联关系，微服务访问追踪系统可以获取微服务间访问拓扑关系。在事件数据层面，通过聚类、异常检测算法

和模板匹配将多源异构数据统一为了事件类型数据，随后训练关系分类器挖掘事件间因果关系，最后将历史数据按照故障类型分组再沉淀生成对应故障类型的组件-事件知识图谱。下一章会在此基础上，研究组件-事件知识图谱的表示学习工作。

### 第三章 组件-事件知识图谱表示学习

本章主要设计了针对组件-事件知识图谱的知识表示学习模型。在已有的知识库中（如FB15K<sup>[8]</sup> 和 YAGO37<sup>[51]</sup>），由于每个三元组 (*head, relation, tail*) 都可以视作一条知识而单独存在，知识图谱可以用多个三元组组成的集合表示，即  $\mathcal{KG} = \{(e_i, r_k, e_j)\}$ （其中 *i, j* 为实体索引，*k* 为关系索引）。因此，已有的大多数知识表示模型都是输入三元组 (*head, relation, tail*)，然后着重于缩短这三者嵌入表示之间的语义距离。

在本文的组件-事件知识图谱  $G = (\mathcal{F}, \mathcal{V}, \mathcal{E}, \mathcal{R})$  中，每个组件-事件知识图谱都对应着一个故障类型  $\mathcal{F}$ ，这使得每个三元组的成立与其上下文背景信息有关，且每个实体在不同的上下文中应有不同的嵌入表示，即应随着上下文的变化而动态地表示。详细分析如章节3.1所示。虽然文献 [24, 25] 尝试将图结构上下文引入嵌入表示，但是它们都将原本的图结构转为邻居上下文、路径上下文再进行嵌入表示学习。这种方式不足以获取全面的图结构上下文信息，且不能满足动态的嵌入表示学习（不能根据具体上下文给同一实体不同的嵌入表示），所以本文利用图神经网络获取组件-事件知识图谱中的上下文结构信息，再结合实体的语义信息，获取动态的实体嵌入表示。

## 3.1 场景分析

本节主要分析了组件-事件知识图谱的特性，解释了为何本文的知识表示学习与上下文背景信息相关，且需要动态的嵌入表示。

### 3.1.1 上下文背景信息

在组件-事件知识图谱中，一个三元组的成立与其上下文拓扑关系有关，这使得一个三元组不能被视为一条可以单独存在的知识。下面是以图2-6中信息为例，对组件-事件知识图谱中三大类关系的分析：

(1) 抽象事件间的因果关系。以三元组 (*Backoff, cause, HttpServerErrorException503*) 为例，该三元组中 *Backoff*（微服务重启失败）和 *HttpServerErrorException503*（微服务服务不可用）之间不一定有触发关系。该三元组的成立，需要一定的背景信息如式3.1所示，即需要这两个事件发生在同一个微服务组件、或者有交互依赖关系的两个微服务组件。

$$\begin{aligned} & (ts - train - service, happen, Backoff) \\ & \wedge (ts - ticketinfo - service, happen, HttpServerErrorException503) \\ & \wedge (ts - ticketinfo - service, call, ts - train - service) \\ & \Rightarrow (Backoff, \mathbf{cause}, HttpServerErrorException503) \end{aligned} \tag{3.1}$$

(2) 组件到抽象事件的发生关系。以三元组  $(ts - travel - service, happen, HttpServerErrorException500)$  为例，该三元组中  $ts - travel - service$  是否会发生事件 *HttpServerErrorException500* 同样与其周边设备的状态有关。该三元组成立

依据于式3.2中所示的条件。

$$\begin{aligned}
 & (ts - ticketinfo - service, happen, HttpServerErrorException503) \\
 & \wedge (ts - travel - service, call, ts - ticketinfo - service) \\
 & \wedge (HttpServerErrorException503, cause, HttpServerErrorException500) \\
 \Rightarrow & (ts - travel - service, \textbf{happen}, HttpServerErrorException500)
 \end{aligned} \tag{3.2}$$

(3) 组件到组件的交互关系。以三元组  $(container1, hold, ts - train - service)$  为例, 该三元组中  $container1$  负载着  $ts - train - service$  而不是其他微服务, 同样需要借助背景信息才能推导出来。背景信息推导出该三元组成立如式3.3所示。

$$\begin{aligned}
 & (container1, happen, cpu100\%) \\
 & \wedge (container1, happen, memory100\%) \\
 & \wedge (ts - train - service, happen, Backoff) \\
 & \wedge (cpu100\%, cause, Backoff) \\
 & \wedge (memory100\%, cause, Backoff) \\
 \Rightarrow & (container1, \textbf{hold}, ts - train - service)
 \end{aligned} \tag{3.3}$$

由上述分析可见, 组件-事件知识图谱中每条三元组都不能作为一条知识单独存在, 而是整个知识图谱中三元组之间相互佐证使得整个拓扑图成为了蕴含知识的知识图谱。由此启发, 组件-事件知识图谱的知识表示学习需要将实体所在上下文背景信息考虑进去, 才能学习得到有效的实体与关系嵌入表示。

### 3.1.2 动态嵌入表示

根据3.1.1节分析, 本文知识图谱表示学习需要引入上下文背景信息。在引入上下文背景信息方面, 文献 [24, 25] 将实体所在的图结构转为邻居上下文、路径上下文, 通过最大化实体在其上下文的条件概率进行嵌入表示学习。但这种最大化概率的方式, 对实体、关系的嵌入表示是静态的, 即一旦训练完成嵌入表示是不会再随着上下文改变的。

这种获取静态嵌入表示的方法不适用于组件-事件知识图谱的表示学习。因为同一抽象事件由于其所在上下文不同、所在设备不同应当有着不同的嵌入表示。如图3-1所示是“订票服务不可用”故障对应的组件-事件知识图谱部分示意图, 在该图中同样存在  $Backoff$  抽象事件和  $(ts - travel - service, happen, HttpServerErrorException500)$  三元组, 但该图中  $ts - travel - service$  发生  $HttpServerErrorException500$  是由于订票微服务  $ts - order - service$  上发生的抽象事件  $Backoff$  导致的。相对应的图2-6中  $ts - travel - service$  发生  $HttpServerErrorException500$  是微服务  $ts - train - service$  上的  $Backoff$  间接导致的, 对应着“数据库不可用无法查票”故障。在这两个故障场景中, 抽象事件  $HttpServerErrorException500$  是由不同原因导致的, 所以应有不同的嵌入表示。抽象事件  $Backoff$  发生在不同的微服务上, 也应有着不同的嵌入表示。所以, 在组件-事件知识图谱中, 实体嵌入表示需要随上下文信息动态变化。

根据上下文进行动态表示在词嵌入领域已经有大量的相关工作。在最初的工作中, word2vec<sup>[52]</sup> 最大化单词在上下文出现的概率, 将每个单词表示为低维稠密的静态向量, 却

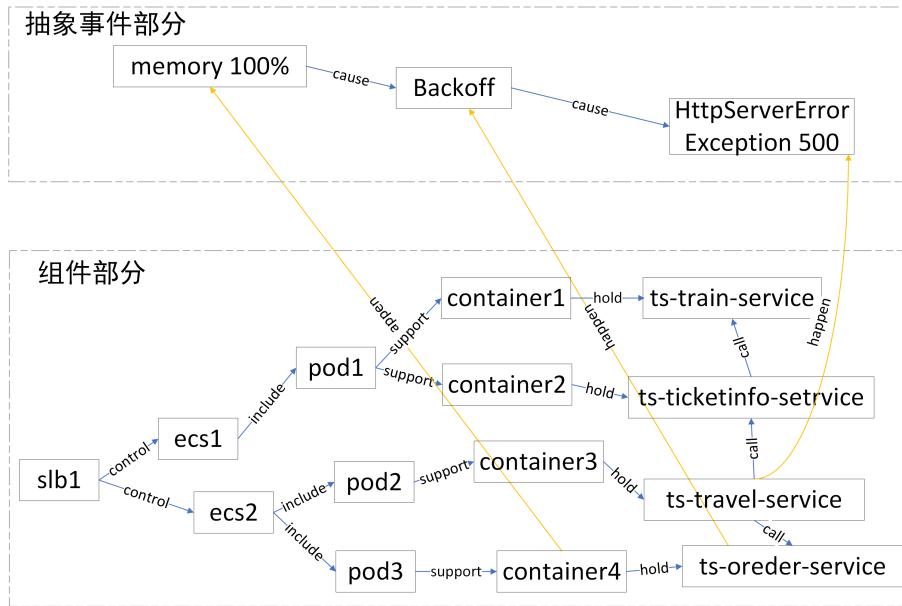


图 3-1 订票服务不可用故障对应知识图谱部分示例

忽视了单词在不同上下文含义不同。为了识别单词在不同上下文中具体的语义，ELMo<sup>[53]</sup> 提出使用双层记忆网络，Bert<sup>[54]</sup> 使用基于自注意力机制的 Transformer<sup>[55]</sup> 编码层以获取单词随上下文变动的动态词向量。其中 Bert 所使用的 Transformer 结构其实是将一段文本看做单词为结点的全连接图。受此启发，本文将知识图谱看作稀疏图，使用图卷积网络作为编码器，以引入三元组上下文的图结构信息。本文进一步将图卷积网络获取的上下文图结构信息与三元组实体的文本语义信息融合，从而获取了三元组在不同上下文的动态表示。

## 3.2 基于图卷积网络的上下文图结构嵌入算法

本小节介绍文献 [26] 中基于图卷积网络的 RGCN (Relational Graph Convolutional Networks) 算法。该算法区分了不同关系对节点的影响，利用图信息传播机制获取了实体的上下文信息。本文后续章节在该算法基础上，设计了组件-事件知识图谱表示学习模型。

### 3.2.1 模型结构

本小节介绍 RGCN 的模型结构，包括嵌入层和图卷积层。

嵌入层主要为了引入实体多维度的信息，如实体包含着的各种语义信息。对于每个实体，该文献将其不同的文本属性连接起来，并使用预先训练的 Bert 模型来获得它们的嵌入表示。这些嵌入表示形成了实体的初始特征向量。对于实体没有属性的数据集，嵌入层则是随机初始化的。

图卷积层使用图卷积网络将局部的信息通过关系传递到整个图中，因此可以将其理解为一个可微分的消息传递框架，可以用式子3.4表示。

$$h_i^{(l+1)} = \sigma \left( \sum_{m \in \mathcal{M}_i} g_m(h_i^{(l)}, h_j^{(l)}) \right) \quad (3.4)$$

$h_i^{(l)} \in \mathbb{R}^{d^{(l)}}$  是结点  $v_i$  在  $l$ -th 层神经网络的隐状态， $d^{(l)}$  是该层隐状态向量的维度。 $g_m(\cdot, \cdot)$

函数计算了  $v_j$  传入结点  $v_i$  的信息。随后累加  $v_i$  每个邻居结点传来的信息，并将累计结果传入激活函数  $\sigma(\cdot)$  就可以得到  $v_i$  在下一层神经网络的隐状态向量  $h_i^{(l+1)}$ ，其中激活函数可以使用  $\text{ReLU}(\cdot) = \max(0, \cdot)$  等常见的激活函数。 $\mathcal{M}_i$  表示每一条传入结点  $v_i$  的信息汇总集，其中每一条传入信息都与一条边对应。 $g_m(\cdot, \cdot)$  通常为信息计算函数，具体实现上可以是某种神经网络也可以是某种线性转变，比如  $g_m(h_i, h_j) = Wh_j$ ， $W$  为参数张量（如文献 [56] 所示）。这样的信息计算函数有效地累计并编码了来自局部的结构化邻域特征，并在诸如图分类<sup>[57]</sup>和图的半监督学习等领域取得了显著的改进效果。

基于上文的消息传递架构，一个知识图谱可以表示为  $\text{Graph} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ ，其中  $v_i \in \mathcal{V}$  表示结点， $(v_i, r, v_j) \in \mathcal{E}$  表示被标注的边， $r \in \mathcal{R}$  表示关系类型。其信息传播过程如式3.5所示。该式用于计算有向图中由  $v_i$  表示的实体或节点的隐状态随信息传递的变化。

$$h_i^{(l+1)} = \sigma \left( \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \quad (3.5)$$

其中， $h_i^{(l)} \in R^{d^{(l)}}$  为结点  $v_i$  在第  $l$  层神经网络的隐状态， $d^{(l)}$  是该层网络表示空间的维度。而  $\mathcal{N}_i^r$  是索引为  $i$  的结点在关系  $r \in \mathcal{R}$  下的所有邻接结点索引值集合。 $c_{i,r}$  是特定于问题的规范化常量，可以被学习或预先指定（例如  $c_{i,r} = |\mathcal{N}_i^r|$ ）。该方法通过归一化和来计算相邻节点传播来的信息，并且只选择相邻结点进行线性变换  $Wh_j$  具有重要的计算优势，既不需要使用大量内存保存关系向量，也通过使用稀疏矩阵乘法减少了资源消耗。

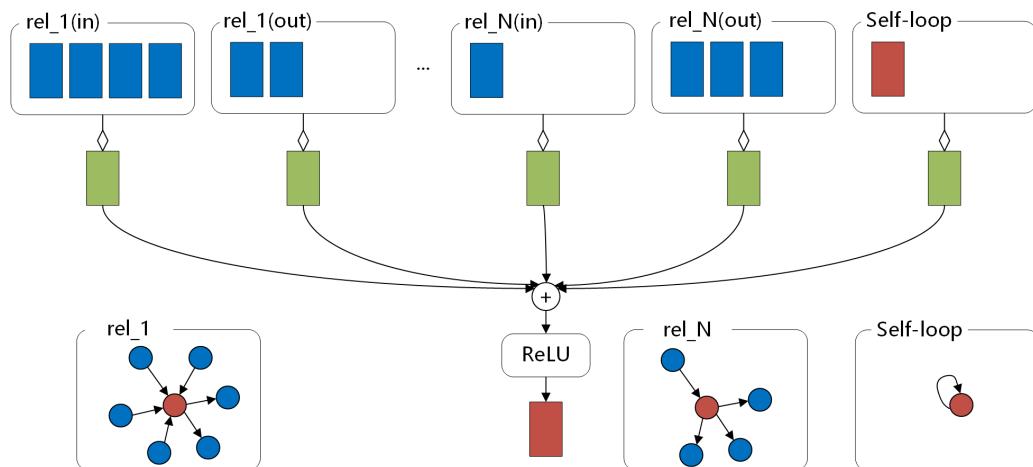


图 3-2 每层图网络隐状态更新过程

每一层图网络的更新都需要对有向图中每个节点按照公式3.5进行并行计算。在模型实现中，可以堆叠多个图网络层以获取跨多个边的依赖关系。其中单个节点更新的计算图如图3-2所示。待更新隐状态的结点或实体为红色，其所有的一跳邻居结点都为蓝色，用  $d$  维度的向量表示。然后针对每个 relation 类型分别进行信息传播计算，信息计算结果表示以标准化和的形式累加（绿色），最后经过激活函数作为新的隐状态向量，激活函数可以选择 ReLU 等。

### 3.2.2 模型训练

模型训练方面主要使用了来自实体分类和链接预测两方面的损失函数。

实体分类：将如式子3.5的每一层图神经网络堆叠起来，并在最后一层节点的输出隐状态向量上使用  $\text{softmax}(\cdot)$  激活函数。最终在标注的结点上最小化式3.6表示的交叉熵损失函数即可。

$$\mathcal{L} = - \sum_{i \in \mathcal{Y}} \sum_{k=1}^K t_{ik} \ln h_{ik}^{(L)} \quad (3.6)$$

其中  $\mathcal{Y}$  是有标签的结点索引集， $h_{ik}^{(L)}$  是索引为  $i$  的带标签结点对应的网络输出的第  $k$  维的实数值。 $t_{ik}$  表示该实体的标注向量第  $k$  维的实数值。最后使用梯度下降算法训练模型参数即可。

链接预测：预测事实存在还是不存在（即三元组  $(\text{head}, \text{relation}, \text{tail})$  是否存在）。有向、有标注的知识图谱可以使用符号式表示，即  $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ 。训练过程中没有使用完整的边集  $\mathcal{E}$ ，而是使用了其子集  $\hat{\mathcal{E}}$ 。模型会给候选边  $(h, r, t)$  一个分数  $f(h, r, t)$ ，用以确定该边属于  $\mathcal{E}$  的可能性有多大。为了进行链接预测，该文献引入了图编码-解码模型，编码部分使用上述图神经网络编码实体，解码部分对应着一个打分函数。编码器将每一个实体  $v_i \in \mathcal{V}$  映射到一个实数向量  $e_i \in \mathbb{R}^d$ 。解码器会根据编码器输出的实体表示预测链接，具体实现上就是通过形如  $s : \mathbb{R}^d \times \mathcal{R} \times \mathbb{R}^d \rightarrow \mathbb{R}$  的函数给三元组  $(\text{head}, \text{relation}, \text{tail})$  评分。编码器选用了上述的图卷积网络，解码器选用了文献 [58] 中的 DistMult 模型。DistMult 模型中每一个关系  $r$  都与对角矩阵  $R_r \in \mathbb{R}^{d \times d}$  相关，候选三元组可以用式3.7计算得到评分。

$$f(h, r, t) = e_h^T R_r e_t \quad (3.7)$$

训练时，使用负采样为每个三元组负采样  $\omega$  个负样本。负样本来自于将正确的三元组的 head 或 tail 随机地替换掉。最终以正确的三元组评分要高于负样本为目的，使用了以下式3.8为损失函数。

$$\begin{aligned} \mathcal{L} = & -\frac{1}{(1 + \omega)|\hat{\mathcal{E}}|} \sum_{(h, r, t, y) \in \mathcal{T}} y \log l(f(h, r, t)) + \\ & (1 - y) \log(1 - l(f(h, r, t))) \end{aligned} \quad (3.8)$$

其中  $\mathcal{T}$  是包含了正确和错误三元组的集合，而  $l$  是 sigmoid 函数。 $y$  是三元组的标签， $y = 1$  表示该三元组是正确的， $y = 0$  表示该三元组是错误的。

### 3.3 组件-事件知识图谱表示学习模型

本节主要介绍了针对组件-事件知识图谱的表示学习方法。上文介绍到实体会出现在不同的上下文中，所以需要随着上下文的变化拥有动态的向量表示。另外，由于图卷积网络可以基于图信息传播机制充分地提取图的拓扑结构信息，所以本文使用图卷积网络作为编码器获取上下文图结构信息，再与实体本身的语义信息融合，得到了实体随着上下文变化的动态嵌入表示。在图卷积网络部分，为了让实体对不同的邻居结点有不同的侧重，本文又引入了注意力机制给邻居结点赋予了不同的权重。

### 3.3.1 模型结构

整个模型结构如图3-3所示，可见主要包括嵌入层、Attention-RGCN层、三元组分类层、实体分类层和关系分类层。下面将会分别对每一层展开介绍。

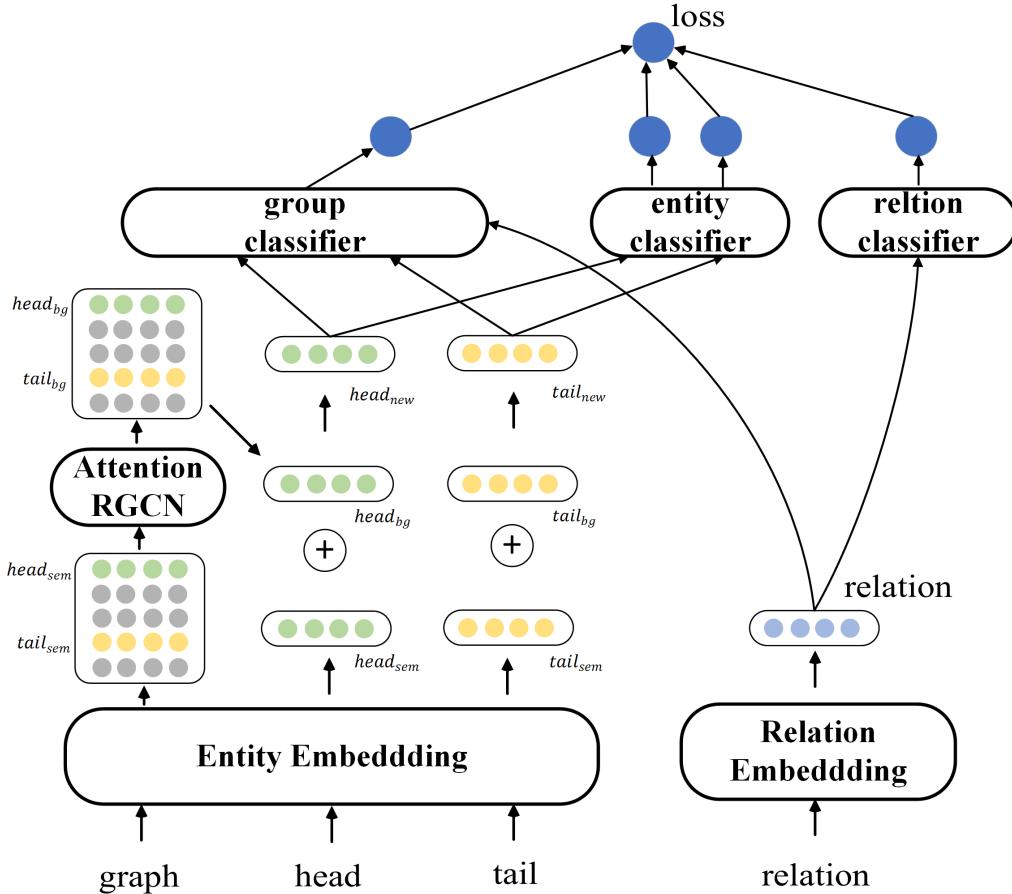


图 3-3 组件-事件知识图谱表示学习模型

在嵌入层，模型的输入除了三元组 ( $head, relation, tail$ ) 还有该三元组所在的上下文图  $graph$ ，即  $(graph, head, relation, tail)$ 。在输入数据前，若三元组为  $graph$  中存在的，会将  $graph$  中  $head$  与  $tail$  之间的关系  $relation$  去掉，防止引入的图结构信息  $head_{bg}$  与  $tail_{bg}$  包含了  $relation$  信息；若三元组中  $head$  或  $tail$  不存在于  $graph$  中，则对应  $head_{bg}$  或  $tail_{bg}$  设置为全零。在输入数据后，数据被嵌入层转化为初步向量表示， $graph$  被转为了张量矩阵， $head$ 、 $relation$ 、 $tail$  则都被转为了单维的向量。初始的嵌入层向量来自于实体、关系的语义信息。具体实现上，使用文本类的日志数据对 Bert<sup>[54]</sup> 进行预训练，然后每个实体或关系的初始特征向量使用其分词后多个单词的向量均值。

Attention-RGCN 层主要在 RGCN 模型中引入了注意力机制。章节3.2已经对 RGCN 进行了介绍，在进行特征传播时，实体对其上下文实体都赋予了相同的权重。但在组件-事件知识图谱中，实体对其上下文邻居实体是有不同侧重的，如图2-6中  $cpu100\%$  和  $memory100\%$  导致了  $Backoff$  的发生，但  $memory100\%$  对  $Backoff$  的发生更具关键性，它意味着  $container1$  中没有足够的内存满足  $ts-train-service$  配置文件里规定的启动需求。因此，本文将注意

力机制引入图卷积网络的传播过程，如式3.9所示。

$$h_i^{(l+1)} = \sigma \left( \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \alpha_{(j,r,i)} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \quad (3.9)$$

整体结构与式3.5一致，其中新引入的  $\alpha_{(j,r,i)}$  目的在于计算第  $i$  个结点对其在关系  $r$  下的所有邻居结点  $j \in \mathcal{N}_i^r$  的注意力权重。其计算公式如下：

$$v_{(j,r,i)} = \mathbf{W}_a^T (\mathbf{W}_e h_j + \mathbf{W}_r h_r + \mathbf{W}_e h_i) \quad (3.10)$$

$$\alpha_{(j,r,i)} = \text{softmax}(v_{(j,r,i)}) = \frac{\exp(v_{(j,r,i)})}{\sum_{j' \in \mathcal{N}_i^r} \exp(v_{(j',r,i)})} \quad (3.11)$$

式3.10中  $h_i, h_j, h_r \in \mathbb{R}^d$ ,  $\mathbf{W}_a \in \mathbb{R}^d$ ,  $\mathbf{W}_e, \mathbf{W}_r \in \mathbb{R}^{d \times d}$  是注意力参数矩阵。最终得到的  $v_{(j,r,i)} \in \mathbb{R}$  即为索引为  $i$  的结点对关系  $r$  下索引为  $j$  的结点的注意力权重值。式3.11使用  $\text{softmax}(\cdot)$  函数进一步规范化注意力权重值，其中  $\mathcal{N}_i^r$  为索引为  $i$  的结点在关系  $r$  下对应的实体索引值集合。经过 Attention-RGCN 层后可以得到  $head$ 、 $tail$  对应的上下文图结构嵌入表示  $head_{bg}$ 、 $tail_{bg}$ ，再与通过初步嵌入层的语义向量  $head_{sem}$ 、 $tail_{sem}$  累加即可得到同时融入了语义信息和上下文图结构信息的嵌入表示  $head_{new}$ 、 $tail_{new}$ 。

实体分类器用于对实体进行分类，类别标签包括 *containier*、*event*、*service* 等共计  $c_n$  个。输入为经过 Attention-RGCN 得到的融合了语义及上下文图结构信息的嵌入表示  $head_{new}$ 、 $tail_{new}$ 。随后将  $head_{new}$ 、 $tail_{new}$  输入  $d \times h \times c_n$  的多层感知机和  $\text{softmax}(\cdot)$  层即可，即可得到实体对各类分类情况。损失函数使用交叉熵，如式3.12所示。其中， $i$ 、 $j$  对应着  $head$ 、 $tail$  两实体的索引号， $h_i$ 、 $h_j$  为两实体在网络模型中的输出， $y_i$ 、 $y_j$  为两实体的标注向量， $k$  表示向量第  $k$  维度。

$$\mathcal{L}_{entity} = - \sum_{k=1}^K y_{ik} \ln h_{ik} - \sum_{k=1}^K y_{jk} \ln h_{jk} \quad (3.12)$$

关系分类器用于对关系向量分类，类别包括 *cause*、*happen*、*hold* 等。与实体分类一样，使用多层感知机、 $\text{softmax}(\cdot)$  层分类，然后损失函数使用交叉熵即可，如式3.13所示。其中， $h_r$  为关系对应的网络输出向量， $y_r$  为关系的标注向量， $k$  表示向量第  $k$  维度。

$$\mathcal{L}_{relation} = - \sum_{k=1}^K y_{rk} \ln h_{rk} \quad (3.13)$$

三元组分类器用于判别三元组是否存在，可以使用文献 [16] 中形如式3.14的评分函数。其中  $\text{diag}(\cdot)$  将  $h_r \in \mathbb{R}^d$  中每一个元素当作二维矩阵的对角线元素，从而得到  $\mathbb{R}^d \times \mathbb{R}^d$  的二维矩阵。损失函数使用交叉熵，如式3.15所示，其中  $\mathcal{T}$  表示所有的输入  $(g, i, r, j)$  集合， $l(\cdot)$  为激活函数， $y$  为三元组是否成立的标签。

$$f(h_j, h_r, h_i) = h_j^T \text{diag}(h_r) h_i \quad (3.14)$$

$$\mathcal{L}_{group} = - \frac{1}{|\mathcal{T}|} \sum_{(g,i,r,j) \in \mathcal{T}} y \log l(f(h_i, h_r, h_j)) + (1 - y) \log(1 - l(f(h_i, h_r, h_j))) \quad (3.15)$$

### 3.3.2 参数学习

根据上一小节所述，本模型的损失函数为实体分类器、关系分类器、三元组分类器三部分损失函数之和。最终本模型的损失函数如式3.16所示：

$$\begin{aligned}\mathcal{L} = & -\frac{1}{|\mathcal{T}|} \sum_{(g,i,r,j) \in \mathcal{T}} y \log l(f(h_i, h_r, h_j)) + (1 - y) \log(1 - l(f(h_i, h_r, h_j))) \\ & - \sum_{k=1}^K y_{ik} \ln h_{ik} - \sum_{k=1}^K y_{jk} \ln h_{jk} \\ & - \sum_{k=1}^K y_{rk} \ln h_{rk}\end{aligned}\quad (3.16)$$

### 3.3.3 预测

对组件-事件知识图谱进行表示学习后，可以进行抽象事件预测（预测在已发生抽象事件情况下是否会在某组件上发生某种事件）。根本上，就是在给定上下文图结构  $g$  下，三元组  $(i, r, j)$  是否成立。

将上下文图结构、头实体、关系和候选尾实体  $g$ 、 $(i, r, j)$  输入表示学习模型中，会得到该三元组成立的评分，将评分最高的三元组对应的尾实体作为预测抽象事件即可，如式3.17所示。

$$e = \arg \max_{j \in events} (f(h_i, h_r, h_j)) \quad (3.17)$$

## 3.4 本章小结

本章主要介绍了组件-事件知识图谱的表示学习模型，通过将实体表示分为结构特征和语义特征，其中拓扑结构特征通过图卷积网络获取，语义特征使用实体分词对应词向量均值表示，从而实现了实体在不同拓扑上下文的动态表示，另外也介绍了参数更新和预测事件的公式。下一章将在本章提出的表示学习模型基础上进行故障预测。

## 第四章 基于知识图谱的故障预测

本章节主要介绍基于知识图谱的故障预测模型。在云计算场景中，分布式集群会随时间产生大量的事件序列。本文提出的故障预测模型可以利用已发生的事件序列，并结合知识图谱，预测是否会有故障产生以及具体产生何种故障，有助于运维人员提前检测、调整系统，防止故障真正发生而带来损失。具体实现上，该模型通过双向记忆网络获取实时事件序列编码表示，再使用上一章的表示学习模型获取知识图谱嵌入表示，最后将两者通过注意力权重计算相似度值，相似度值最高的知识图谱对应的故障信息即为所预测的故障。本模型在进行故障预测时取得了最高的准确率，且具有更高的细粒度与更好的可解释性。

### 4.1 场景分析

目前已经存在着较多相关的工作，均集中在基于数据的时序性展开研究。文献 [29-31] 使用了统计学习和机器学习方法，如隐半马尔可夫模型（HSMM，Hidden Semi-Markov Model）、支持向量机（SVM，Support Vector Machine）。然而，HSMM 和 SVM 都建立于所有输入都是平稳且相互独立的假设之上，这种假设在云计算场景中是不成立的，因为云计算场景中不同组件产生的事件信息之间会存在一定的关联，比如先后时间顺序。为了处理时间序列数据进行故障预测，文献 [36-40] 使用深度学习方法如循环神经网络（RNN）和长短期记忆神经网络（LSTM）进行预测。但 RNN 存在着难以克服的缺点，它会遗忘掉较远距离的数据信息。LSTM 则克服了这个缺点，可以通过记忆门依然保留长距离数据的信息。文献 [37-39] 等工作已经使用 LSTM 在故障预测任务上比 HSMM、SVM 和 RNN 取得了更高的准确率。上述的方法都使用 CPU 使用率、内存使用率、未映射页缓存、平均磁盘 I/O 时间和磁盘使用率作为输入，把系统任务是否会失败作为输出。随后，文献 [42] 在 LSTM 的基础上使用了多层的 Bi-LSTM、引入了任务优先级、提交时间、提交次数等更多的特征，还根据不同数据对故障的重要性赋予其不同的权重。

在本文场景中进行故障预测时，还需要考虑以下场景特性：

1) 数据形式为事件序列。本文所有的数据，除了包括上文提到的 CPU 使用率、内存使用率、未映射页缓存、平均磁盘 I/O 时间和磁盘使用率，另外还有新引入的微服务启动、重启、程序运行日志等数据，都已经被转为统一规范化的事件数据，所以要处理的数据为事件序列。

2) 需引入知识图谱增强可解释性。本文在进行故障预测时，已经存在了前文所构建的组件-事件知识图谱。引入知识图谱会使故障预测结果更具可解释性，还能具体到会出现何种故障以及导致故障发生的触发链。

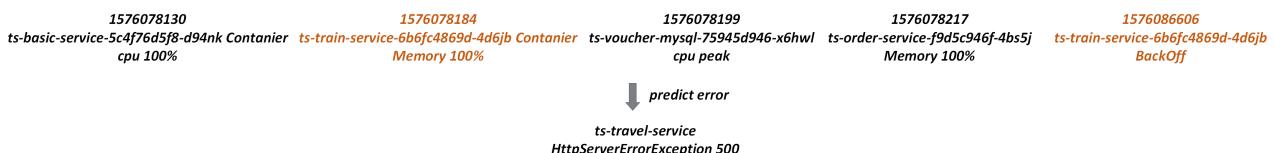


图 4-1 事件序列故障预测

3) 事件权重分配需要考虑序列上下文,不能仅根据数据对故障的重要性赋予权重。如图4-1所示是一个实时事件序列案例。在进行故障预测时,并不是每个异常事件都是同等重要的。 $ts-train-service-6b6fc4869d-4d6jb$ 上的 $ContainierMemory100\%$ 和 $ts-train-service-6b6fc4869d-4d6jb$ 上的 $BackOff$ 事件应当比其他事件的重要性更高,因为这两个事件的发生意味着 $ts-train-service$ 所在的 $Container$ 已经内存不足,使得 $ts-train-service$ 微服务无法正常启动,后续导致 $ts-travel-service$ 发生 $HttpServerErrorException500$ 的可能性极高。而其它的事件如 $ts-order-service-f9d5c946f-4bs5j$ 上的 $Memory100\%$ 虽然对图3-1的“订票服务不可用”故障较为重要,但是并没有发生后续的关键事件 $ts-order-service$ 上的 $BackOff$ ,所以其重要性较低。

基于以上的分析,本文在编码事件序列时,依然选择基于LSTM的模型,但不同于之前根据每个事件元素对故障重要程度赋予权重<sup>[42]</sup>,本文选择使用组件-事件知识图谱的嵌入向量来对各个时间单元的数据计算注意力权重,即由组件-事件知识图谱选取其所关注的信息。

## 4.2 基于记忆网络的故障预测算法

本小节介绍了编码时序数据的双向记忆网络模型,文献[42]使用该模型在预测是否会有故障时达到了最好的效果。下一节4.3中,本文在此模型基础上引入了组件-事件知识图谱进行故障预测。

### 4.2.1 模型结构

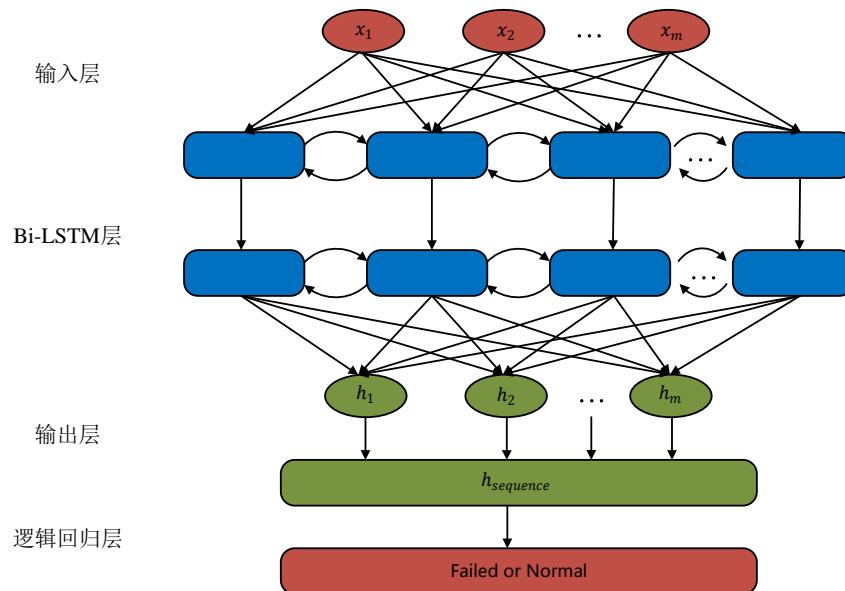


图 4-2 双向记忆网络

图4-2所示为双向记忆神经网络,可见其包含输入层、Bi-LSTM层、输出层和逻辑回归层。

输入层主要是数据的嵌入层。文献[42]拼接了同一时间点的CPU使用率、内存使用率、未映射页缓存、平均磁盘I/O时间、磁盘使用率、任务优先级、提交时间和提交次数共计8个特征。这样之后每个时间点都是一个8维的嵌入向量。

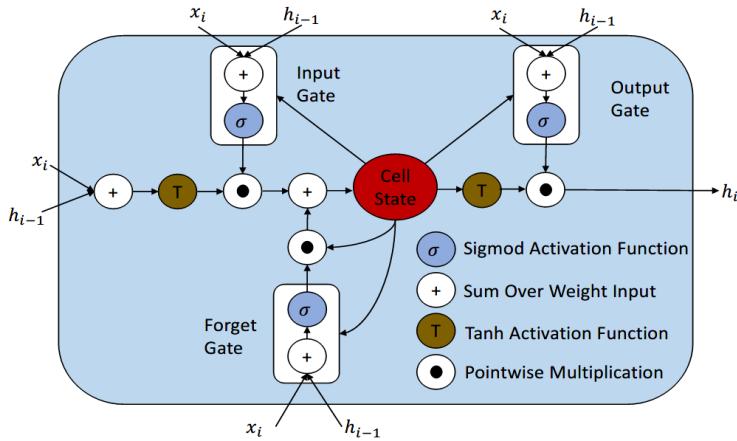


图 4-3 记忆网络单元

Bi-LSTM 层完成了对时序数据的特征编码。该层会将输入的时序数据分别进行前向编码和后向编码，再将得到的前向隐状态和后向隐状态拼接得到最终的隐状态。经过上述步骤后，每个时间点的隐状态都会得到正向反向的上下文信息。图4-3展示了 LSTM 神经元内部的结构，可见其主要利用了非线性函数选择数据保存还是舍弃。具体实现上，每个神经元内部共有三个门控制其状态，包括输入门、遗忘门和输出门。输入门决定了应该更新哪个神经元状态，遗忘门决定了应该忽视掉什么信息，输出门决定输出隐状态的哪部分信息。该过程可以用公式4.1表示：

$$\begin{aligned}
 g_i &= \varphi(w_{gx}x_i + w_{gh}h_{i-1} + b_g) \\
 n_i &= \sigma(w_{nx}x_i + w_{nh}h_{i-1} + b_n) \\
 f_i &= \sigma(w_{fx}x_i + w_{fh}h_{i-1} + b_f) \\
 o_i &= \sigma(w_{ox}x_i + w_{oh}h_{i-1} + b_o) \\
 s_i &= g_i \odot n_i + s_{i-1} \odot f_i \\
 h_i &= \varphi(s_i) \odot o_i
 \end{aligned} \tag{4.1}$$

其中  $w_{gx}, w_{nx}, w_{fx}$  和  $w_{ox}$  是记忆单元输入  $x_i$  的权重系数。 $w_{gh}, w_{nh}, w_{fh}$  和  $w_{oh}$  是记忆单元上一步输出  $h_{i-1}$  的权重系数。 $b_g, b_n, b_f$ , 和  $b_o$  分别为输入信息  $g_i$ 、输入门  $n_i$ 、遗忘门  $f_i$  和输出门  $o_i$  对应的偏置系数。而  $s_i$  和  $s_{i-1}$  则分别为时间  $i$  和  $i-1$  的神经元状态。另外， $\odot$  代表点乘， $\sigma$  代表 sigmoid 激活函数， $\varphi$  代表 tanh 激活函数。

输出层目的在于将输入序列中每个记忆单元隐状态整合起来。具体方式是将输入  $X = \{x_1, x_2 \dots x_m\}$  传入 Bi-LSTM 得到的表示序列  $\{h_1, h_2 \dots h_m\}$  进行式4.2中所示的平均池化操作，得到序列表示向量  $\hat{h}$ 。

$$\hat{h} = \frac{1}{m} \sum_{i=1}^m h_i \tag{4.2}$$

逻辑回归层是预测当前数据序列将来是否会有故障的二分类器。具体实现上，该层将输出层得到的向量表示  $\hat{h}$  输入 logistic(.) 函数，计算会出现故障的概率。当概率大于阈值时，则预测会出现故障；反之，则预测不会出现故障。

## 4.2.2 模型训练

基于记忆网络的故障预测模型的最终训练目标为正确分类每种数据序列。对应的损失函数为式4.3中所示。

$$\mathcal{L} = - \sum_{i=1}^n [Y_i \log(f(X_i)) + (1 - Y_i) \log(1 - f(X_i))] \quad (4.3)$$

其中  $X_i$  表示第  $i$  条输入序列,  $Y_i$  为  $X_i$  对应的标签, 即下一个时间点是否会有故障发生 (1 代表有故障发生, 0 代表不会有故障发生)。 $f(\cdot)$  表示上文的网络模型。

## 4.3 基于记忆网路和知识图谱的故障预测模型

本小节主要介绍基于双向记忆网络和组件-事件知识图谱的故障预测模型。在上节已经介绍了编码时序数据的双向记忆网络, 本节会进一步引入组件-事件知识图谱辅助故障预测, 旨在提高故障预测的细粒度 (预测到会出现何种故障) 和预测结果可解释性。

### 4.3.1 模型结构

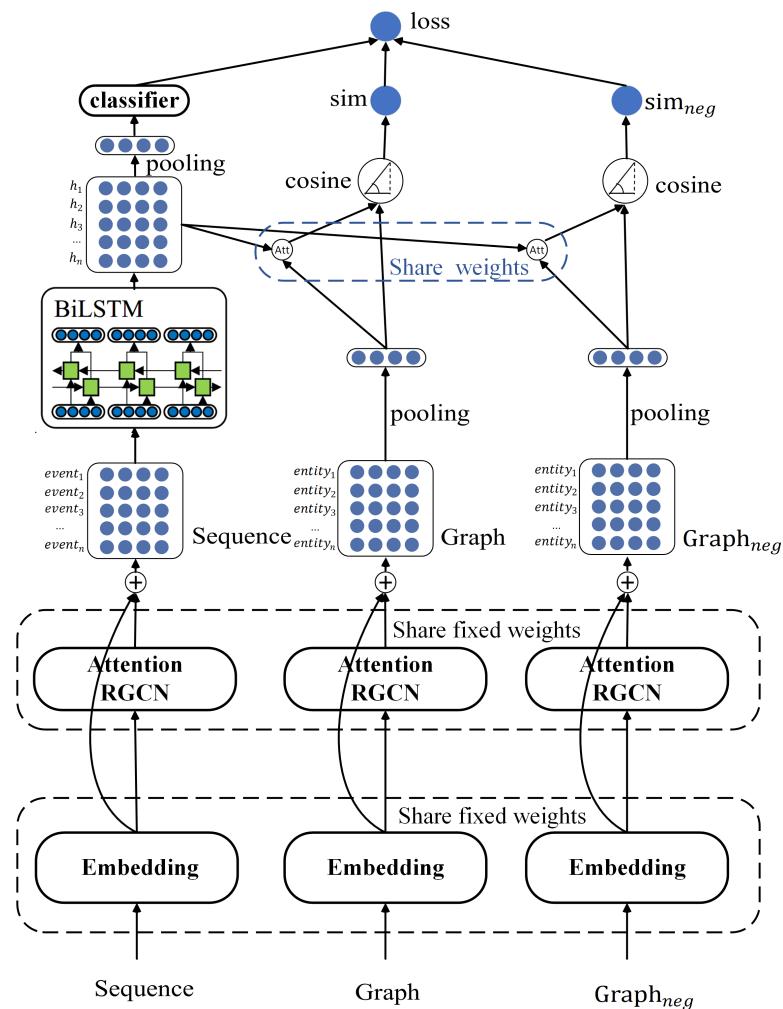


图 4-4 故障预测模型

图4-4是引入组件-事件知识图谱的故障预测模型。其中嵌入层、Attention-RGCN 使用组件-事件知识图谱表示学习模型的结构与参数，并固定不变。双向记忆网络、注意力层和分类层的参数会在后续进行学习。

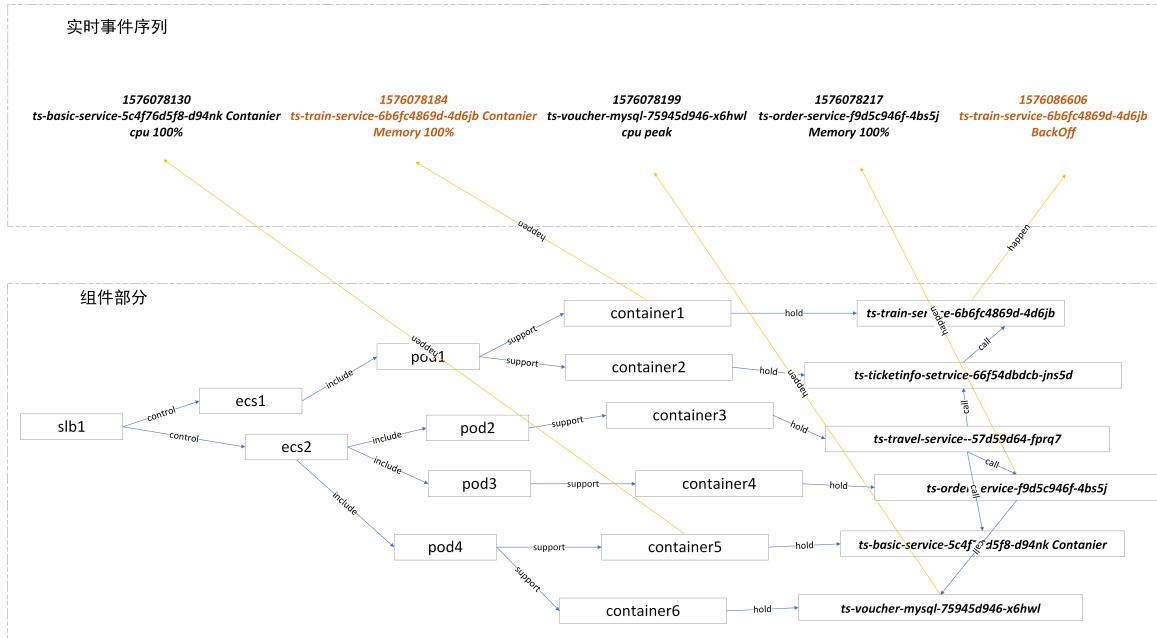


图 4-5 输入实时事件序列样例

嵌入层规定了实时事件序列和知识图谱的嵌入方式。图4-5为一个实时事件序列案例，可见实时事件序列中每个事件也都对应着所在的组件，且组件之间存在着拓扑关系，只是每个组件都有其唯一标识，但对应的组件类型未发生改变，所以同样可以使用图3-3中的表示学习模型，获取每个实时事件的嵌入表示向量。组件-事件知识图谱则同图2-6所示。输入层中的 embedding 和 Attention-RGCN 均使用章节3.3.2所学到的参数并固定，实时事件序列和组件-事件知识图谱都会共享这些参数用于嵌入表示。

双向记忆网络层用来获取实时事件序列之间的时序特征。在实时事件序列中，虽然存在着组件之间的拓扑关系，但不存在事件之间的因果关系，事件之间只存在着前后的时序信息。记忆网络层使用上节4.2中的双向记忆网络 BiLSTM 模型。

在注意力层中，特定故障类型的组件-事件知识图谱会对经过双向记忆网络得到的隐状态序列做注意力权重累加。这样的注意力计算方式可以让知识图谱选择其所关注的事件子序列，并减少了噪声事件的干扰。输入事件序列  $X = \{x_1, x_2 \dots x_m\}$  在经过嵌入层和双向记忆网络后会得到隐向量序列  $\{h_1, h_2 \dots h_m\}$ 。如式4.4所示，注意层会首先计算知识图谱对隐状态序列的权重  $\alpha_{(.,g)}$ ，然后进行权重累加获取事件序列嵌入表示  $h_{sequence}$ 。

$$\begin{aligned} e_{(h_i,g)} &= h_i \mathbf{W} g \\ \alpha_{(h_i,g)} &= \text{softmax}(e_{(h_i,g)}) = \frac{\exp(e_{(h_i,g)})}{\sum_{j=0}^m \exp(e_{(h_j,g)})} \\ h_{sequence} &= \sum_{i=0}^m \alpha_{(h_i,g)} \cdot h_i \end{aligned} \quad (4.4)$$

输出层分为两方面，一方面会将双向记忆网络的输出平均池化后得到的向量直接进行预测分类（有无故障），另一方面将组件-事件知识图谱的嵌入表示与注意力层权重累加得到的

序列嵌入表示  $h_{sequence}$  做相似度计算，输出每张故障类型对应组件-事件知识图谱的匹配度评分，按得分从高至低排列，即可得到预测故障类型排列结果。

### 4.3.2 参数学习

模型的损失函数如式4.5所示，分为两部分。在预测判别有无故障时，使用基于交叉熵的损失函数。在知识图谱相似匹配时，通过负采样用边际损失作为损失函数。

$$\begin{aligned} \mathcal{L} = & - \sum_{i=1}^n [Y_i \log(f(X_i)) + (1 - Y_i) \log(1 - f(X_i))] \\ & + Y_i \cdot \max(0, b + \cos(h_i, g_{neg}) - \cos(h_i, g))] \end{aligned} \quad (4.5)$$

其中， $X_i$  为第  $i$  个输入事件序列  $\{x_1, x_2 \dots x_m\}$ ， $Y_i$  为第  $i$  个输入事件序列有无故障的标签（0 表示输入事件序列无故障，1 表示输入事件序列有故障）。 $h_i$  为  $X_i$  事件序列的隐状态嵌入表示， $g$  为  $X_i$  对应故障类型的组件-事件知识图谱的嵌入表示， $g_{neg}$  为负采样任意一个与  $X_i$  不匹配的组件-事件知识图谱的嵌入表示向量， $b$  为损失函数的偏置超参。

### 4.3.3 预测

在训练完上述基于知识图谱的故障预测模型后，该模型可用于实时的故障预测，即输入实时的异常事件序列  $X$ ，模型会计算发生各种故障的概率，IT 运维人员可根据故障预测结果及时采取防范措施。

具体计算方式为遍历故障库  $failures$  中每个故障  $f$  对应的组件-事件知识图谱  $g_f$ ，分别与实时事件序列  $X$  成对输入模型  $model(\cdot)$  计算匹配度评分。最后，取评分最高的故障类型  $failure$  作为预测结果即可，如式4.6所示。

$$failure = \arg \max_{f \in failures} (model(X, g_f)) \quad (4.6)$$

## 4.4 本章小结

本章主要介绍了基于实时事件序列和组件-事件知识图谱的故障预测。通过上一章的组件-事件知识图谱表示学习模型获取每个故障类型对应的组件-事件知识图谱的嵌入表示向量，然后对双向记忆网络编码的实时事件序列隐状态进行注意力权重累加，获取事件序列嵌入表示。最后计算两个嵌入表示向量的相似度，按照相似度从高至低排序各个知识图谱即可得到预测的故障类型列表。本文的故障预测模型引入了组件-事件知识图谱，预测结果可以具体到故障类型，且具有了可解释性。

## 第五章 实验与评估

本章主要介绍了针对上文所提模型进行的实验。首先，本章描述了模拟故障、构建数据集的过程。随后，分为三块分别介绍了与上文各章节对应的实验，包括事件因果关系判别实验、组件-事件知识图谱表示学习实验、故障预测实验。

### 5.1 数据集构建

表 5-1 集群各类组件数量统计

组件名称	train-ticket	sock-shop
VPC	34	28
SLB	30	36
ECS	30	30
Pod	215	181
Container	720	621
Service	172	132
共计	1201	1028

表 5-2 故障模拟信息

分布式应用	train-ticket	sock-shop
时间跨度	25 天	14 天
平均故障时间段跨度	303s	310s
单个故障时间段含有事件数目	4823	3352
故障次数	473	541
故障种类	10	17
事件类型种类	204	211

实验采用了阿里云 kubernetes 集群，集群各个组件数量如表5-1所示。在该集群上部署了文献 [45, 46] 实验时分别使用的两个分布式应用，即train-ticket和sock-shop。随后使用kube-monkey和ChaosBlade向应用注入异常以模拟故障。在异常注入到故障产生的时间段中，阿里云的 SLS 平台负责收集实时产生的日志数据和指标时序数据。所模拟故障的数据统计信息如表5-2所示。这些数据后续经过事件生成模块，均转化为了事件类型数据。

事件类型数据按照来源划分可以分成两类：指标时序事件和日志事件。指标时序事件对应着曲线变化的异常点，而日志事件对应着集群中组件实时产生的日志。指标时序事件和日志事件可以继续往下划分，所划分生成的类别与每类别的数目如下表5-3所示。

表 5-3 事件类型层次关系

事件大类别	细分	类别数目 (train-ticket)	类别数目 (sock-shop)
指标时序事件	异常点事件	64	64
	告警事件	21	21
日志事件	k8s 日志事件	50	50
	dockerIO 日志事件	69	76
共计		204	211

异常点事件名称由监控项拼接异常点类型构成，表示了在组件的某个监控项曲线出现了某种异常点。监控项共有 16 种，如式 5.1 所示。异常点类型共有 4 种：突然上升、突然下降、尖峰、低谷。每个监控项都会出现这四种异常点，比如 *system.mem.util* 突然上升 *system.mem.util* 突然下降、*system.mem.util* 出现尖峰、*system.mem.util* 出现低谷。因此，将每个监控项与异常点类型两两排列组合，可生成共计 64 种异常点事件。

(5.1)

```

container.cpu.load.average.10s,
container.cpu.usage.seconds.total,
container.fs.reads.bytes.total,
container.fs.writes.bytes.total,
container.memory.usage.bytes,
container.network.receive.bytes.total,
container.network.receive.packets.dropped.total,
container.network.transmit.bytes.total,
container.network.transmit.packets.dropped.total,
system.net.drop.util,
system.net.in,
system.net.out,
system.io.disk.rbps,
system.io.disk.wbps,
system.mem.util,
system.cpu.util

```

告警事件名称是由监控项和阈值组成的，表明了在设备的某个监控项上出现了达到阈值的告警信息。会发生告警事件的监控项共 7 种，每个监控项目都有 3 种告警阈值。监控项和相应阈值如表 5-4 所示。可见每个监控项都会生成 3 种告警事件，比如 *pod.cpu.util* 达到 100%、*pod.cpu.util* 达到 90%、*pod.cpu.util* 达到 95%。因此，将每个监控项与对应阈值组合，可生成告警事件共计 21 种。

k8s (kubernetes) 是分布式集群容器的管理系统，k8s 日志事件就是 k8s 管理系统产生的事件，包含 pod 镜像拉取、pod 健康状况等事件。这类事件共有 50 种细分类型，式 5.2 展示了

表 5-4 告警事件监控项及相应阈值

监控项	阈值
pod.cpu.util	100% 95% 90%
pod.mem.util	100% 95% 90%
system.cpu.util	100% 95% 90%
system.mem.util	100% 95% 90%
system.net.drop.util	20% 10% 5%
system.net.err.util	20% 10% 5%
system.partition.space.usage	100% 95% 90%

部分 k8s 日志事件类型。

(5.2)

*Unhealthy.Warning,*  
*Killing.Normal,*  
*Scheduled.Normal,*  
*Pulled.Normal,*  
*Created.Normal,*  
*Started.Normal,*  
*Pulling.Normal,*  
*BackOff.Warning,*  
*FailedSync.Warning,*  
*Failed.Warning,*  
*FailedCreatePodSandBox.Warning,*  
*SandboxChanged.Normal,*  
*FailedKillPod.Warning,*  
*FailedScheduling.Warning,*  
*BackOff.Normal,*  
*FailedMount.Warning,*  
*Scheduled.Normal,*  
*Pulled.Normal,*

DockerIO 日志事件指的是 Docker 中的输入输出日志所转化生成的事件。该类事件记录了 Docker<sup>[59]</sup> 中的各种操作信息和容器中微服务应用运行时日志信息。在将这些日志聚类并编写人工模板后，人工模板被用来匹配日志信息以收集统计 DockerIO 日志事件类型。其中，因为不同分布式应用的微服务结构不同，且输出日志直接由开发人员编写的代码所决定，所以两应用中的 DockerIO 事件类型有所不同。最终 train-ticket 应用中收集到 69 种 DockerIO 日志事件，而 sock-shop 应用中共收集到 76 种 DockerIO 日志事件。式 5.3 中列出了部分 DockerIO

日志事件类型名称。

```
java.lang.AbstractMethodError,
java.lang.AssertionError,
java.lang.ClassCircularityError,
java.lang.ClassFormatError,
java.lang.Error,
spring.server.connect.failed,
warn.Omitting,
warning.stderr,
spring.http.server.error,
error.mcp.Failed.to.create.a.new.MCP.sink.stream,
trying.to.establish.new.MCP.sink.stream,
socket.go.Successful.write.metrics.to.monitor.server,
operator.go.sync.prometheus,
event.go.reason.UPDATE.Ingress,
```

(5.3)

## 5.2 事件因果关系判别实验

## 5.3 组件-事件知识图谱表示学习实验

### 5.3.1 数据标注

在事件因果关系判别实验结束后，为了如现实场景一样存在历史数据和实时数据，本文将模拟收集的数据进行了划分。具体上，模拟收集的所有数据会首先按照故障类型分组。然后，如表5-5所示，每种故障类型下的时间段集合会被划分成历史故障数据和实时故障数据。根据上小结实验结果，本块实验选用 SVM 模型从每类故障对应的历史故障数据中挖掘事件因果对，再按照章节2.3所示步骤沉淀生成每类故障的组件-事件知识图谱。由模型初步生成的知识图谱仍会有一些噪声数据，后续运维专家参与了知识图谱调优，保证了知识图谱质量。因为初步生成的知识图谱含有实体、关系量级低，所以知识图谱调优过程消耗人工较少。

表 5-5 模拟数据划分

分布式应用	模拟故障	历史故障	实时故障
	时间段总数	时间段数	时间段数
train-ticket	471	325	146
sock-shop	539	371	168

表 5-6 各类故障对应知识图谱信息

分布式应用	故障代号	抽象事件节点数目	抽象事件因果关系数目
train-ticket	f1	41	285
	f2	39	297
	f3	6	8
	f5	62	81
	f6	34	233
	f10	15	51
	f13	15	15
	f16	35	59
	f17	38	65
	f18	54	88
sock-shop	db_goods_disappeared	20	86
	db_cart_disappeared	70	638
	user_unable_log_in	52	180
	db_order_disappeared	20	114
	order_cart_500	9	8
	order_count_500	55	536
	order_payment_500	13	30
	user_register_500	7	6
	cart_disappeared	4	5
	catalogue_goods_disappeared	6	7
	add_cart_delay	30	57
	user_register_and_log_in_delay	17	43
	check_order_delay	102	598
	net_loss_check_order	92	614
	net_loss_add_cart	34	79
	net_loss_user_register_and_log_in	17	55
	net_loss_goods_appear_delay	15	33

表 5-7 知识库三元组数目及划分

分布式应用	组件层 三元组数	组件-事件间 三元组数	事件层 三元组数	总 三元组数	训练集	验证集	测试集
train-ticket	1469	339	1182	2990	1794	598	598
sock-shop	1043	563	3089	4695	2817	939	939

表5-6统计了两个分布式应用每类故障类型所对应组件-事件知识图谱的抽象事件实体数目及抽象事件因果关系对数。在每张组件-事件知识图谱中，由于每个抽象事件实体都对应着一个组件实体，所以组件到抽象事件的发生关系数量与抽象事件节点数相等；由于组件层实体数量和关系数量只与分布式应用有关，所以每个组件-事件知识图谱的组件层关系数是恒定的。*train-ticket* 组件层关系数为 1469，*sock-shop* 组件层关系数为 1043。

在知识图谱中，一个关系对应着一个三元组，所以直接将各个组件-事件知识图谱中的每个关系对应的三元组视作正样本，即将其标注为“成立”。其中组件层的关系不会重复使用，防止数据不均衡。表5-7为本块实验使用的正样本三元组数量，和按照 6: 2: 2 划分数据集的结果。另外，对于每一个正样本三元组，可以将其首实体或尾实体随机替换为不匹配的实体，构成负样本三元组，即“不成立”的三元组。

### 5.3.2 评测标准

在知识图谱表示学习模型评测方式中有两个被广泛使用的任务。

**任务 1：**链接预测，即预测受损三元组中缺少的头或尾实体。文献 [7, 8] 定义链接预测为给定  $(head, relation)$  或  $(relation, tail)$  预测对应丢失的  $tail$  或  $head$ 。在具体进行链接预测任务时，实验会排序候选实体集合，而不是直接只输出一个最佳匹配实体。

具体上，依据文献 [8] 的做法，对于每一个测试三元组  $(head, relation, tail)$ ，实验会使用实体集合中任意不匹配  $(relation, tail)$ （或  $(head, relation)$ ）的实体来替换  $head$ （或  $tail$ ）生成受损三元组，然后根据模型输出的三元组成立概率分数降序排列这些三元组。在获得所有三元组排名后，实验使用了两个衡量指标：正确实体的平均排名（即 MR，mean rank）；正确实体在前 N 名的占比（即 Hits at n, Hits@n）。MR 越小，Hits@N 越大，都意味着表示模型效果越好。另外，文献 [8] 发现生成的受损三元组可能是正确三元组，所以需要将该三元组筛选过滤掉。而本文中，由于知识图谱已被调优过，每个正确三元组都在对应 *graph* 中完备地存在着，当实体与  $(relation, tail)$ （或  $(head, relation)$ ）不匹配时（不存在于组件-事件知识图谱中）意味着其生成的受损三元组一定是错误的，所以本块实验不需进行过滤操作。

**任务 2：**三元组分类，即判断给定的三元组是否是知识图谱中真实存在的。在三元组分类任务中，给定知识图谱 *graph* 和三元组  $(head, relation, tail)$ ，需要判断它是否是正确的。该任务已经在已有的工作 [8-10] 中展开过，也被广泛的应用在很多自然语言处理场景，比如问答。

链接预测时，输入候选实体和上下文信息，输出三元组成立的概率，成立概率值越大排名越高，再计算对应的 MR 和 Hits@N 即可。三元组分类时，输入待测三元组及其上下文，输出该三元组是否成立，再计算精确率（precision）、召回率（recall）和 F1 值即可。

### 5.3.3 实验方法

本块实验选取了多个对比方法以验证本文模型的优越性。实验记录了选取的每个方法在上小节两个评测任务上的表现指标。所选取的各个方法如下所示：

- (1) **TransE<sup>[8]</sup>**: TransE 将每个三元组  $(head, relation, tail)$  中的 *relation* 视作从 *head* 到 *tail* 的翻译过程。

- (2) **TransH<sup>[9]</sup>**: TransH 将头尾实体都投影到关系所在的张量空间中，可以解决复杂的一对多、多对多关系。
- (3) **TransR<sup>[10]</sup>**: TransR 将关系也嵌入为矩阵，可以区分具有不同语义的关系。
- (4) **GAKE<sup>[24]</sup>**: GAKE 引入邻居上下文、边上下文、路径上下文信息，可以捕获图结构的语义。
- (5) **TCE<sup>[25]</sup>**: TCE 引入了两种结构信息，一种是实体的相邻实体，另一种是一对实体之间的关系路径。
- (6) **本文表示学习方法**: 本文在章节三所提出的组件-事件知识图谱表示学习模型。
- (7) **本文表示学习方法-1**: 相比本文表示学习方法，去除了 Attention-RGCN 层的注意力机制。
- (8) **本文表示学习方法-2**: 相比本文表示学习方法，去除实体分类模块的损失。
- (9) **本文表示学习方法-3**: 相比本文表示学习方法，去除关系分类模块的损失。

其中，TransE、TransH 和 TransR 将每个三元组视作一条独立的知识，用于对比验证引入上下文对表示学习的提升；GAKE、TCE 引入上下文信息获取静态表示向量，用于对比验证动态表示学习的有效性；**本文表示学习方法-1** 使用无 attention 机制的 RGCN，用于对比验证本文在 RGCN 中引入注意力机制的提升；**本文表示学习方法-2** 去除实体分类模块，用于对比验证引入实体类别信息对模型的提升；**本文表示学习方法-3** 去除关系分类模块，用于对比验证关系类别信息对模型的提升。

### 5.3.4 实验结果与分析

在链接预测任务上，表5-8列出了各个方法的链路预测结果。**本文表示学习方法**在 MR 和 Hit@5 指标上都得到了比其他方法更好的实验结果。在三元组分类任务上，表5-9列出来各个方法的精确率（precision）、召回率（recall）和 F1 值。**本文表示学习方法**相较基线方法获得了 5% 以上的 F1 值提升。实验结果证明了本文利用注意力机制的 RGCN 获取图结构上下文信息，再结合语义信息的动态表示学习方法，可以更好地嵌入表示组件-事件知识图谱。

对比**本文表示学习方法-1**、**本文表示学习方法-2**、**本文表示学习方法-3**，可发现图神经网络的注意力机制对模型的提升效果最大，其可以强化重要信息的传播、挖掘重要的图结构信息；实体、关系类别信息的引入也可以改善知识表示学习效果，其中实体类别信息比关系类别信息提升效果更高。

另外，三元组分类任务上取得的精确率要比链接预测上 Hits@5 要高，原因在于三元组分类任务中每个“成立”的三元组都只生成  $\omega$  个“不成立”的受损三元组（本实验中  $\omega$  取 100）；而链接预测任务中，每个正确三元组与平均约 1000 个受损三元组进行比较排序。

本块实验中，本文模型达到最佳效果时使用的超参为：嵌入层维度为 100；Attention-RGCN 为 2 层、中间隐状态维度为 50、输出隐状态维度为 100；图卷积网络输入窗口大小为 128 节点，隐状态维度为 100，共计 4 层；实体分类器结构为  $100 * 32 * 7$ ，7 为实体种类数；关系分类器结构为  $100 * 32 * 8$ ，8 为关系种类数。

表 5-8 链接预测结果

方法	train-ticket		sock-shop	
	MR	Hits@5	MR	Hits@5
TransE	59	48.3	63	42.7
TransH	42	57.3	56	45.2
TransR	37	66.2	43	59.8
GAKE	33	62.8	39	53.2
TCE	23	81.9	31	78.1
本文表示学习方法	<b>9</b>	<b>85.4</b>	<b>13</b>	<b>82.9</b>
本文表示学习方法-1	26	79.5	34	67.6
本文表示学习方法-2	19	82.2	30	78.4
本文表示学习方法-3	16	83.6	24	81.2

表 5-9 三元组分类结果

方法	train-ticket			sock-shop		
	precision	recall	F1	precision	recall	F1
TransE	0.650	0.692	0.670	0.653	0.626	0.639
TransH	0.703	0.730	0.716	0.707	0.675	0.691
TransR	0.743	0.716	0.730	0.680	0.701	0.690
GAKE	0.781	0.749	0.765	0.722	0.742	0.732
TCE	0.804	0.783	0.793	0.736	0.769	0.752
本文表示学习方法	<b>0.864</b>	<b>0.837</b>	<b>0.850</b>	<b>0.836</b>	<b>0.814</b>	<b>0.825</b>
本文表示学习方法-1	0.763	0.816	0.788	0.776	0.746	0.761
本文表示学习方法-2	0.826	0.801	0.813	0.783	0.808	0.795
本文表示学习方法-3	0.845	0.822	0.834	0.788	0.810	0.799

## 5.4 基于知识图谱与实时事件序列的故障预测实验

### 5.4.1 数据标注

在构建组件-事件知识图谱时，表5-5中的历史故障时间段数据已经被使用。本处的故障预测实验选择使用实时故障时间段数据。由于每一个故障时间段都有对应的故障类型，所以此处不需要人工标注，直接将同一故障类型的组件-事件知识图谱与事件序列视为相似，不同故障类型的知识图谱与事件序列视为不相似即可。

由于故障预测任务是根据已发生的事件序列，预测可能会出现何种故障，所以对每一个实例时间段的事件序列选择去除其最后的故障事件，只保留故障发生前的异常事件序列。另外，每个时间段的事件序列长度不一致，而双向记忆网络模型的输入长度需要是确定的。所以，若记忆网路的输入长度为  $L$ ，对于长度小于  $L$  的事件序列填零补全至  $L$ ，对于长度长于  $L$  的事件序列只保留前  $L$  个事件。

本文用于故障预测的事件序列数较少，两应用中分别有 146 和 168 条。为了扩增数据量，首先，实验将每类故障下的事件序列数按照 6: 2: 2 进行划分得到训练集、验证集、测试集；随后，遍历事件序列中每个事件，选择随机保留或去除（事件关键性越高，被保留的概率越高；反之，越低），这样遍历一次保留下来的事件序列视作随机采样得到的一条新数据；最后，对每一条事件序列随机采样 9 次，完成数据增强。这样随机采样生成新事件序列的方式，不仅扩充了数据，也使得模型具有了鲁棒性。模型即使利用实时产生的部分事件序列信息，也可以正确预测其可能发生的故障。最终，用于故障预测的实时事件序列数目如表5-10所示。

表 5-10 实时事件序列数

分布式应用	初始	随机采样扩展后	随机采样扩展后	随机采样扩展后
	事件序列数	训练集	验证集	测试集
train-ticket	146	870	290	300
sock-shop	168	1000	330	350

### 5.4.2 评测标准

本文将实时事件序列与组件-事件知识图谱进行匹配，等同于将实时事件序列按故障类型做多分类。因此，最终的评测标准以测试集中事件序列分类到组件-事件知识图谱的精确率 (precision)、召回率 (recall) 和 F1 值为指标。本处多分类的精确率、召回率和 F1 值的计算方式为：分别计算各类别的 precision、recall 和 F1，再取均值。

### 5.4.3 实验方法

在模型训练过程中，每一个事件序列除了有其对应的相似组件-事件知识图谱  $G_{pos}$ ，还需负采样任选一个不相似的组件-事件知识图谱  $G_{neg}$ 。模型训练目标函数为最大化两者差距，即与相似组件-事件知识图谱的匹配度评分  $Sim_{pos}$  要远大于与不相似的组件-事件知识图谱的匹配度评分  $Sim_{neg}$ 。在计算测量指标时，首先将事件序列与组件-事件知识库中每个图谱都做匹配度评分计算，并按照匹配度评分降序排列。由于实验数据中类别都只有十余种，所以只在

对应组件-事件知识图谱匹配度评分  $Sim_{pos}$  排序第一时视作分类正确。为了验证本文基于记忆网络和知识图谱的故障预测模型的性能表现，实验选用了以下几种方法对比：

- (1) **PREPARE**<sup>[28]</sup>: PREPARE 结合了属性值预测模型和多变量分类模型进行故障预测。属性值预测模型可以估计属性在未来时间的值分布。多变量分类模型可以计算一组估计的未来属性值映射成“正常”或“异常”状态的概率。具体实现上，该方法使用了马尔可夫模型进行属性值预测，使用了贝叶斯网络进行分类。本块实验将贝叶斯网改为多分类，以和其他方法进行对比。
- (2) **RNN**<sup>[36]</sup>: 使用循环神经网络编码事件序列，再多分类到故障类型。
- (3) **LSTM**<sup>[37-41]</sup>: 使用长短期记忆网络编码事件序列，再多分类到故障类型。
- (4) **Bi-LSTM**<sup>[42]</sup>: 使用双向长短期记忆网络编码事件序列，再多分类到故障类型。
- (5) **本文故障预测方法**: 本文在章节四所提出的基于双向长短期记忆网络和知识图谱的故障预测模型。
- (6) **本文故障预测方法-1**: 相比本文方法，去除了知识图谱对事件序列的 attention 机制。
- (7) **本文故障预测方法-2**: 相比本文方法，使用 RNN 替换了 BiLSTM。
- (8) **本文故障预测方法-3**: 相比本文方法，使用 LSTM 替换了 BiLSTM。

其中，PREPARE、RNN、LSTM 和 BiLSTM 为已有代表性工作所使用的方法，在本实验中作为基线模型。为了对比各个模型，本块实验将 RNN、LSTM 和 BiLSTM 都后接了多层感知机和  $softmax()$  层进行多分类。**本文故障预测方法-1** 用于验证知识图通过注意力机制可以选取有效信息，提升预测效果；**本文故障预测方法-2** 用于验证在编码事件序列时 BiLSTM 比 RNN 效果更好；**本文故障预测方法-3** 用于验证在编码事件序列时 BiLSTM 比 LSTM 效果更好。

#### 5.4.4 实验结果与分析

实验记录了各个方法的精确率、召回率和 F1 值，结果如表5-11所示。可见**本文故障预测方法**取得了最佳的精确率、召回率和 F1 值，证明了本文模型的优质性。PREPARE、RNN、LSTM 和 BiLSTM 相比其他方法效果都较差，证明了引入知识图谱会显著提升故障预测效果。**本文故障预测方法-1** 不如**本文故障预测方法**的效果，证明了注意力机制提升了故障预测效果。**本文故障预测方法-3** 比**本文故障预测方法**效果差，说明了编码事件序列特征方面，LSTM 不如 BiLSTM。**本文故障预测方法-2** 比**本文故障预测方法-3** 效果更差，表明了 RNN 比 LSTM、BiLSTM 在编码事件序列时效果更差。

本块实验中，输入嵌入层和 Attention-RGCN 层设置为上一章四中组件-事件知识图谱表示学习模型的结构与参数，并固定不变。本块实验效果达到最佳时，模型选用的超参数为 BiLSTM 层为 2 层、输入序列长度为 1024、中间隐状态为 50；序列分类器层为  $100 * 32 * 2$  的神经网络加  $softmax()$  层。

表 5-11 故障预测结果

方法	train-ticket			sock-shop		
	precision	recall	F1	precision	recall	F1
PREPARE	0.732	0.727	0.728	0.689	0.677	0.679
RNN	0.765	0.757	0.759	0.703	0.688	0.691
LSTM	0.789	0.761	0.772	0.719	0.691	0.701
BiLSTM	0.807	0.792	0.796	0.747	0.780	0.759
本文故障预测方法	<b>0.939</b>	<b>0.904</b>	<b>0.918</b>	<b>0.887</b>	<b>0.848</b>	<b>0.861</b>
本文故障预测方法-1	0.904	0.892	0.896	0.842	0.831	0.832
本文故障预测方法-2	0.841	0.815	0.823	0.787	0.776	0.782
本文故障预测方法-3	0.879	0.887	0.878	0.825	0.813	0.814

## 5.5 本章小结

本章主要介绍了针对前面各章所述模型的实验实施。在事件因果关系挖掘模块，实验对比了多种机器学习模型，并对比了只依据概率特征和依据本文 6 种特征的表现，最终取 F1 值最高的 SVM 作为最终的因果判别模型；在组件-事件知识图谱表示学习模块，实验对比了已有工作中的模型、本文提出的模型和基于本文模型的退化版，实验效果表明本文引入图传播模型作为编码器生成事件动态嵌入表示的方法在链接预测和三元组分类任务上都取得了最佳效果；在故障预测部分，实验对比了相关工作的模型、本文模型和基于本文模型的退化版，实验效果表明在本文表示学习模型基础上再引入知识图谱会取得最佳的故障预测效果。下一章将介绍基于本文已提出模型所设计的 IT 运维辅助系统。

## 第六章 系统设计与实现

本章介绍基于上述提出的各个模型所设计实现的一个基于知识图谱的 IT 运维辅助系统。实时运行的集群中存在着大量的组件与实时发生的事件，IT 运维人员面对海量的信息需要消耗大量精力和时间逐步查询分析多来源的繁杂数据。一个基于知识图谱的 IT 运维辅助系统可以有效地辅助运维人员，帮助其及时发现集群异常并预测可能出现的故障，从而大大降低了运维难度。本文设计实现的系统通过 jaeger<sup>[44]</sup>、prometheus<sup>[60]</sup>、阿里云公开 API 等方式获取集群拓扑结构、微服务间拓扑结构、日志数据和指标时序数据，将集群实时运行状态可视化。随后，历史监测数据被按故障类型分组，经过事件因果关系挖掘模型构建了每种故障对应的组件-事件知识图谱。基于构建的组件-事件知识图谱，利用上文提出的表示学习模型和故障预测模型，本系统可以根据实时发生的事件序列预测可能会发生的故障，提醒运维人员及时采取应对措施。本章共分 4 小结来介绍系统，介绍的内容包括系统整体架构设计、各类实时数据获取方式、系统各模块实现和前端界面功能展示。

### 6.1 系统框架设计

#### 6.1.1 系统框架

本系统使用 B/S 架构，即用户使用浏览器请求服务，后台服务器响应返回结果的工作模式。整体架构分为三个层次，包括模型、视图和控制器。其中，模型负责实现业务功能逻辑，即按照功能需求整合转换数据形式；视图则将数据可视化，即把模型返回的数据直观地易于理解地展示出来；控制器则是从视图中接收用户输入的数据，并将数据传送给模型。另外，按照系统功能又可以把系统结构划分为数据层、业务逻辑层和交互层。图6-1展示了这三层架构。

- (1) 数据层。本系统的数据层主要包含了日志数据、指标时序数据、集群组件拓扑和微服务间拓扑。这四类数据有不同的获取方式，具体获取方式在小节6.2进行了具体描述。
- (2) 业务逻辑层。业务逻辑层连接了数据层与交互层，包含了算法模型和业务逻辑。该层有多个模块协同实现系统功能，分为事件生成模块、组件拓扑生成、集群状态检测模块、事件因果判别模型、组件-事件知识图谱生成、故障预测模型。各个模块的协同工作实现了系统的核心功能，每个模块的具体功能在6.1.2小节展开具体介绍。
- (3) 交互层。交互层将业务逻辑层数据可视化地直观展示在前端页面，并支持用户操作数据持久化。本系统交互层主要分为三部分：组件-事件知识图谱查询及更正、实时集群状态查询和故障预测结果展示。

#### 6.1.2 系统功能

系统功能已在图6-1中业务逻辑层展示，分为：a. 事件生成模块、b. 组件拓扑生成模块、c. 集群状态监测模块、d. 事件因果判别模块、e. 组件-事件知识图谱生成模块、f. 故障预测模块。

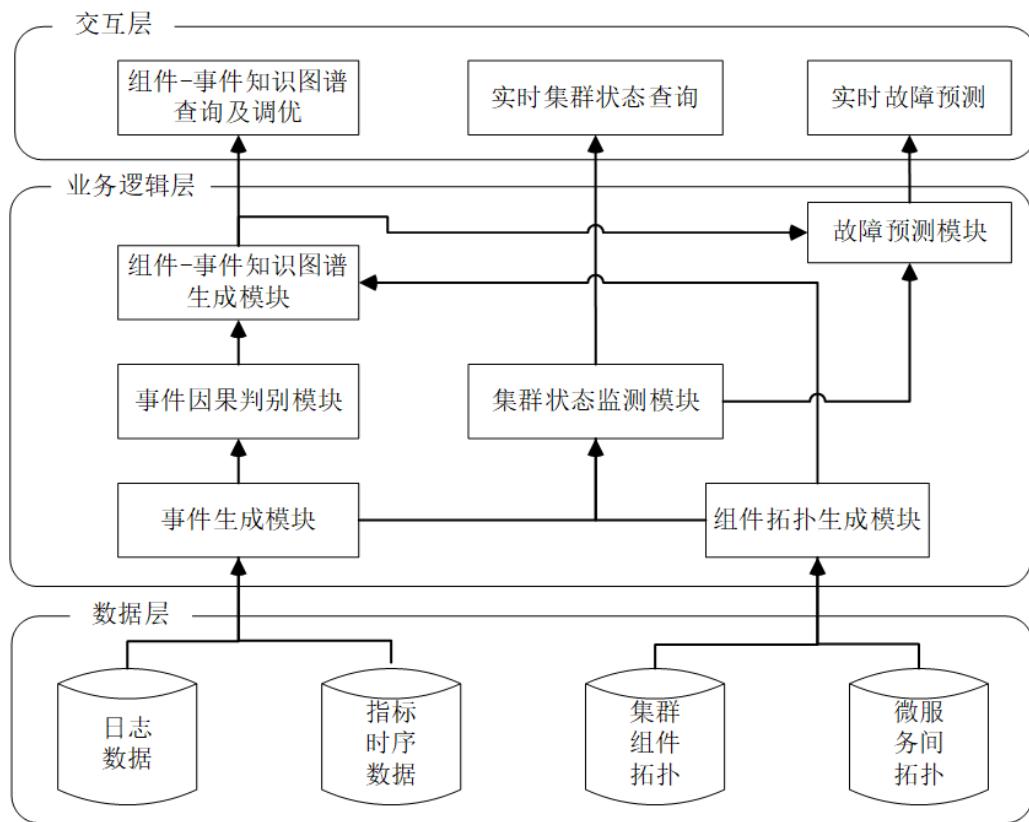


图 6-1 IT 运维辅助系统框架图

事件生成模块将多源异构数据统一转化为规范的事件类型数据。组件拓扑生成模块通过分析集群组件属性信息列表，利用其中表明了集群组件关系的字段生成了集群组件拓扑，而微服务间的拓扑则利用其实际工作时的访问关系生成。集群状态监测模块，则是将前两个模块数据，利用事件发生所在的组件位置连接起来，得到了可视化的集群运行状态图。事件因果关系判别模块会分析事件生成模块所产生事件间的因果关系。组件-事件知识图谱生成模块是从某类故障的历史数据中沉淀出包含组件依赖关系、组件事件产生关系和抽象事件因果关系的组件-事件知识图谱。故障预测模块基于集群状态监测模块所收集的事件序列，联合知识图谱预测未来可能发生的故障。下面具体介绍这些模块的功能：

- (1) 事件生成模块：该模块使用多线程调用阿里云公开 API 读取日志数据、指标时序数据。同样用多线程调用人工模板将数据转为事件类型数据。
- (2) 组件拓扑生成模块：内部使用数据库查询语句，快速获取集群组件与微服务连接关系，从而整合两部分拓扑构建组件拓扑图。
- (3) 集群状态监测模块：该模块负责汇总组件拓扑和事件信息，同时在交互页面点击组件时，该模块会返回组件属性信息和指标时序信息。
- (4) 事件因果判别模块：该模块使用判别模型对事件候选关系二分类，类别为有无因果关系。在判别关系时，输入为两个事件特征向量，输出为有因果关系的判别结果。事件因果关系判别模块将事件集合转化为因果事件对集合，再输入组件-事件知识图谱生成模块。
- (5) 组件-事件知识图谱生成模块：知识图谱的生成即是从历史数据中沉淀知识的过程。本

模块收集历史数据，包括事件因果关系对和相应组件拓扑图，然后按故障类型分组历史数据，依次合并每组数据，即可得到对应故障类型的组件-事件知识图谱。

- (6) 故障预测模块：该模块中，输入为集群状态监测模块所收集的发生在组件拓扑上以时间顺序排列的事件序列，另外组件-事件知识图谱也会被输入到该模块中，最后按照匹配度降序输出故障预测结果即可。

## 6.2 数据获取

系统需要从多个源头获取多种类别的数据。系统需要获取的数据共有 4 种，包括集群拓扑结构、微服务拓扑结构、日志数据、指标时序数据。下面分为四部分分别介绍这四种数据具体的获取方式。

### 6.2.1 集群拓扑结构

第一部分为集群拓扑结构数据的获取，分为以下步骤：

- (1) 系统开发时，引入阿里云提供的 SDK 核心库和组件的 Maven 项目依赖。
- (2) 编写多线程代码调用不同组件的数据请求方法并设置参数，再保存方法返回的 Json 数据。如调用方法 `DescribeImagesRequest()`，设置要查询的镜像类型 `setImageOwnerAlias()`，就会返回对应镜像的 Json 格式的基本信息。
- (3) 分析返回的 Json 数据，读取指定字段值。返回的 Json 数据不同字段包含着不同类型的信息，如字段 `ImageId : freebsd_11_02_64_30G_alibase_20190722.vhd` 表明了镜像的唯一标识符；`OSNameEn : FreeBSD11.264bit` 表明了系统名称；`VpcId : vpc—2zeuphj08tt7q3brd****` 表明了 ECS 所在 VPC 的唯一标识符。
- (4) 按获取的字段信息构建实体属性列表，和组件间关联拓扑。

### 6.2.2 微服务拓扑结构

第二部分为微服务拓扑结构的获取。微服务依赖关系虽然在分布式应用的代码逻辑中已清晰规定，但应用的代码属于机密信息且阅读代码时耗过大。为了无需了解应用代码逻辑就可以获取微服务间访问拓扑关系，本系统选用开源 Jaeger 实时监控并收集微服务间的访问信息。

Jaeger 架构图如图6-2所示，主要包含以下几个部分：

- (1) **jaeger client libraries:** jaeger 客户端是基于 OpenTracing API 的特定语言实现的，可通过与 OpenTracing 集成的各种现有开源框架来检测应用程序，实现分布式跟踪。它会以较小的开销始终在后台开启，不断采样 Span 并异步传输到 Jaeger Agents。采样策略默认为随机采样 0.1%。
- (2) **jaeger agents:** Agent 是网络守护程序，负责侦听通过 UDP 传输的 Span，并分批次将 Span 发送给 Collector。另外，它作为一种基础结构组件会被部署到所有主机。

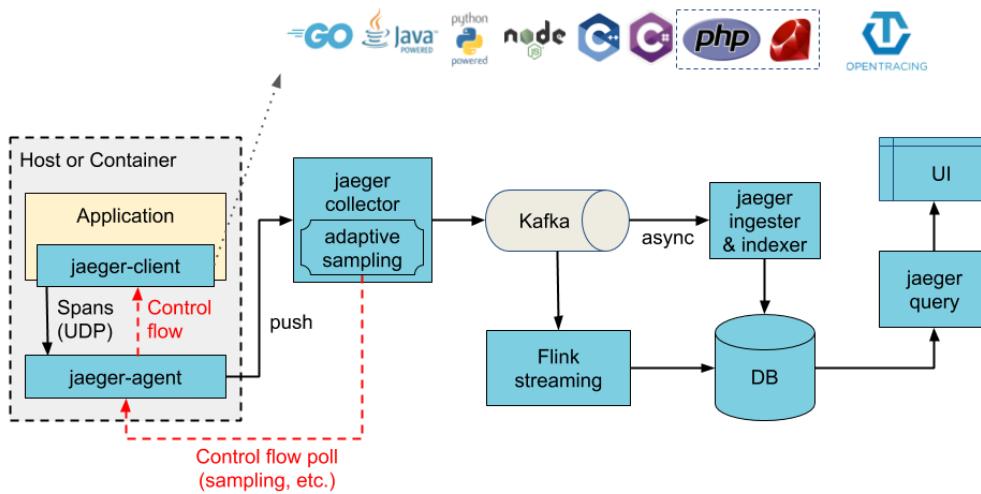


图 6-2 Jaeger 架构图

- (3) **Collector:** Collector 在接收到 Agent 传来的 Span 后，会验证跟踪并为这些 Span 建立索引，并最终存储起来。Jaeger 使用的存储设备是可插拔的，目前支持 Cassandra、Elasticsearch 和 Kafka。
- (4) **Query:** Query 是一项从存储中检索微服务依赖信息，并用 UI 显示的服务。
- (5) **Ingester:** Ingester 是负责提供从 Kafka 读取数据并写入其他存储后端的服务。

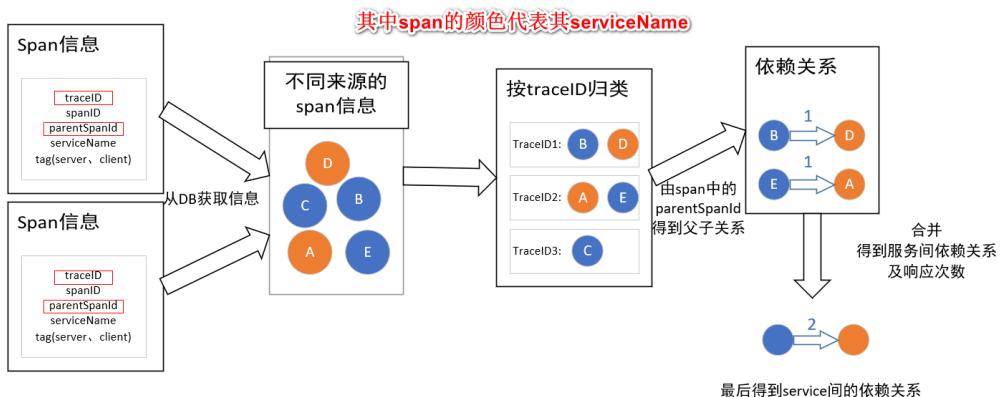


图 6-3 微服务间依赖图生成流程

其中，一个 Span 对应一条微服务被调用的记录，其记录了微服务被标识符为 parentSpan 中的微服务调用了一次。一个 Trace 对应了分布式应用完成一个业务请求时，微服务之间的一条请求链路。而微服务拓扑图的生成是由 Flink Streaming 块完成的。其整体流程如图6-3所示。分为以下步骤：

- (1) 获取 Span 数据。从 Kafka 中读取一个时间段内所有来源的 Span。
- (2) 分组 Span 数据。将所有 Span 按照 traceID 分组，这样就获得了 traceID 到 Span 集合的映射关系。

- (3) 分析同组 Span 间依赖关系。遍历每个 Span 对应的 ParentSpanID，得到 Span 之间的父子依赖关系。
  - (4) 生成微服务间依赖关系。根据 Span 中 serviceName 信息，将 Span 间依赖关系映射为微服务间依赖关系。
  - (5) 微服务间拓扑图生成。将不同 Trace 中微服务间依赖关系整合规约，生成微服务拓扑图结构。

### 6.2.3 目志数据

第三部分为日志数据的收集。日志数据的收集分为以下步骤：

- (1) 日志数据采集。阿里云日志服务 (Log Service, SLS) 提供了数据投递、查询、消费和采集等功能，可以高效的处理海量日志。在阿里云日志服务中创建 Logstore，选择 DockerIO、k8s 日志等作为数据源，日志服务就会实时将数据汇总收集起来。

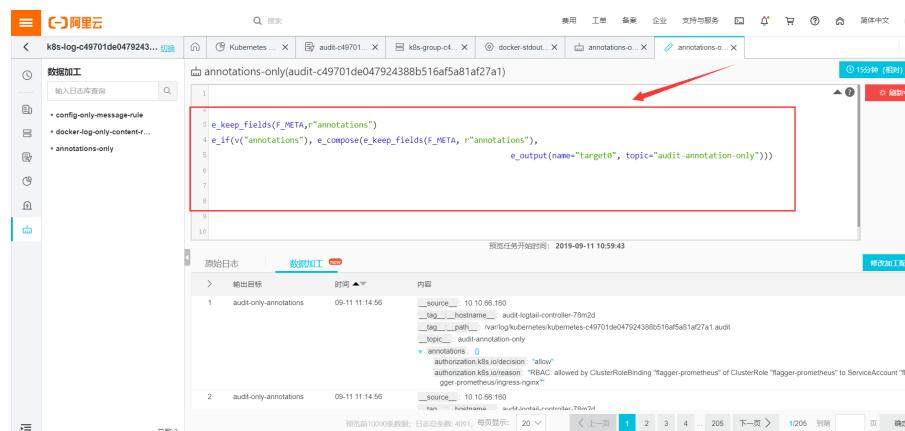


图 6-4 数据加工指令

- (2) 日志数据加工。由于原始日志数据包含了很多无效信息，上一步得到的原始日志需要进一步加工处理只保留有效字段信息。加工过程直接使用日志服务的数据加工功能，编写加工命令（如图6-4所示）选取保留的字段，并传输到新的 Logstore 中即可。图6-5为数据加工实例，该例中只保留了 audit 数据中的 annotation 字段。
  - (3) 公开 API 请求日志数据。经过上述步骤，日志数据被存储在日志服务各个 Logstore 中。当系统需要获取某段时间日志数据时，系统直接使用公开 API 并设置参数请求即可。如使用函数 `PullLogsRequest(project, logStore, shardId, logNum, cursor)` 并设置所要消费的 project、LogStore 和每次读取日志数量 logNum 等参数，就会返回符合参数约束的日志数据。

#### 6.2.4 指标时序数据

第四部分为获取指标时序数据。指标时序数据均为曲线类的数据，在存储形式上，均为一组时间戳到值的映射关系集合。其获取过程如下：

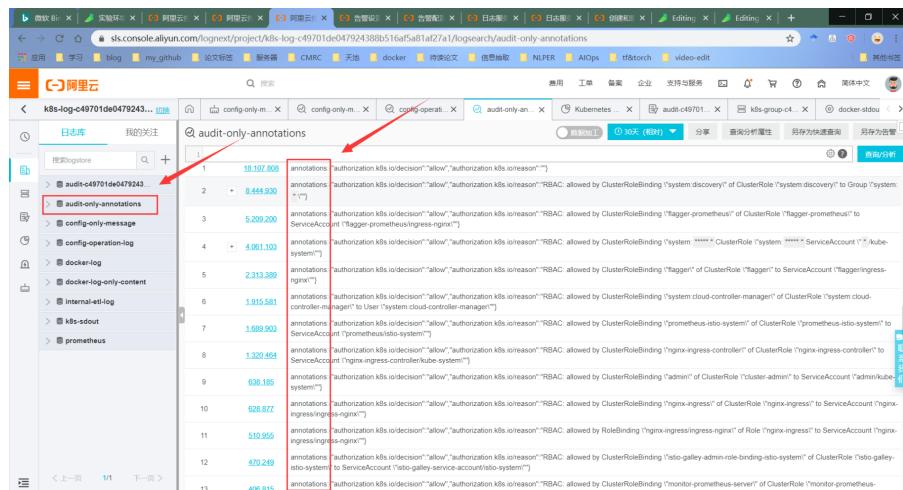


图 6-5 数据加工结果实例

- (1) 指标时序数据检测。在分布式集群中，多种组件都会有指标时序数据，且每个组件上会有不同种类的指标时序数据，如 CPU、内存、磁盘吞吐率等。系统选用了 Prometheus 进行实时的监测。
- (2) 指标时序数据长期存储。由于 Prometheus 监测得到的指标时序数据不能长期存储（一般只保留 14 天），系统为了长期存储选择将数据导入日志服务。图 6-6 展示了指标时序数据导入日志服务后的结果，可见每条数据都对应着一个时间戳到值的映射关系。

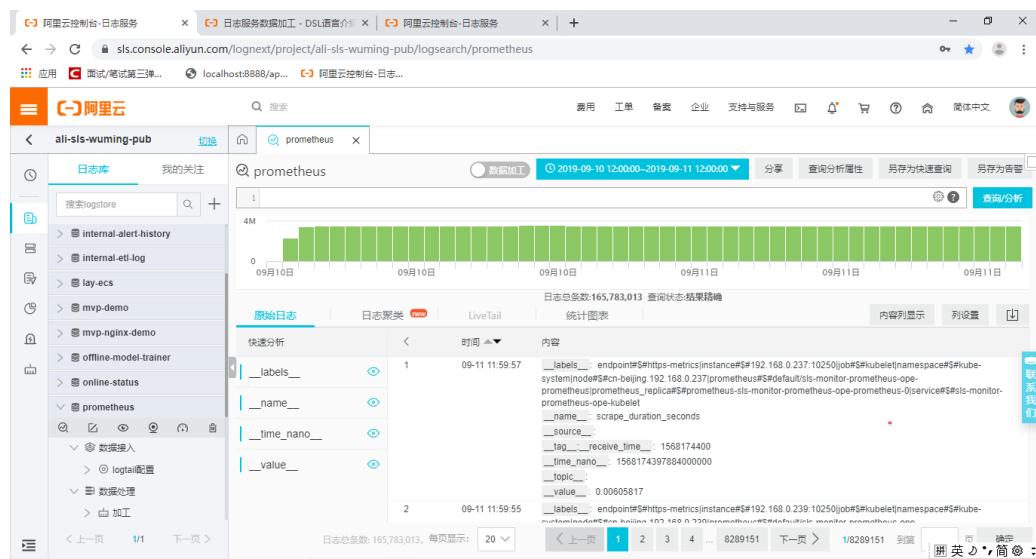


图 6-6 指标时序数据导入日志服务示例

- (3) 公开 API 获取指标时序数据。将数据长期存储入日志服务平台后，系统可采用请求日志的 API 来请求获取指标时序数据。

## 6.3 系统实现和功能展示

### 6.3.1 系统实现

本文所设计实现的 IT 运维辅助系统采用了 B/S 架构，用户可以通过 web 端访问系统。开发本系统的环境为：操作系统为 64 位 Windows 10，处理器为 11th Gen Intel(R) Core(TM) i5-11300H CPU 3.11GHz，内存为 16G；开发语言为 Java, jdk 版本为 1.8.0.60，集成开发环境使用了 IntelliJ IDEA。整个 IT 运维辅助系统使用 Spring Boot 开发后台，Vue.js 编写前端，Neo4j 存储图数据，从而实现了 MVC 框架。此外，在系统具体开发工程中使用了多种开源工具，包括 Jaeger、Flink、Kafka、阿里云日志服务和 Echarts 等，相关技术如图6-7所示。下面对所使用到的技术展开了具体介绍：

- (1) **Spring Boot:** Spring 框架提供了统一的编程、配置模型，便于 Java 语言实现的企业应用程序快速地健壮开发。它通过为应用程序提供基础的支持，使得团队可以关注应用程序层次的业务逻辑，而不必过于关注特定的部署环境。SpringBoot 为 Java 开发人员提供了一个很好的平台，便于开发人员开发一个独立的、生产级的 Spring 应用程序。其优势包括易于理解和开发 Spring 应用程序、提高生产力、缩短了开发时间。本系统使用 SpringBoot 完成了数据读取以及主要的业务逻辑功能模块。
- (2) **Neo4j:** Neo4j 与传统关系型数据库不同，是一个高性能的 NoSql 的图数据库，直接使用了图结构表示存储数据。图结构的存储形式便于数据属性关系的更改，而关系型数据库中的表结构是确定的，一旦更改会消耗大量成本。另外，图数据库实现了路径、拓扑查询等功能。在本系统中，需要处理大量的组件-事件拓扑关系，所以选择了使用 Neo4j 作为后台知识图谱数据库。
- (3) **Jaeger:** Jaeger 是一个分布式跟踪系统，由 Uber 技术公司作为开源项目发布。它被用于监视和排除基于微服务的分布式系统的故障，包括以下方式分布式上下文传播、分布式事务监控、根本原因分析、服务依赖性分析和性能优化。本系统使用 Jaeger 监测并获取了分布式应用各个微服务之间的依赖拓扑。
- (4) **Kafka:** Kafka 属于分布式系统，由通过高性能 TCP 网络协议进行通信的服务器和客户机组成。它可以部署在本地和云环境中的裸机硬件、虚拟机和容器上。它主要拥有三项功能，分别为 a. 发布（写入）和订阅（读取）事件流，包括从其他系统连续导入/导出数据；b. 持久可靠地存储事件流；c. 当事件发生或追溯时处理事件流。所有这些功能都是以分布式、高度可扩展、弹性、容错和安全的方式提供的。在本系统中，Kafka 被用于缓冲从集群各个组件收集来的实时数据。
- (5) **阿里云日志服务:** 日志服务（SLS）是云原生观测分析平台，为 Log/Metric/Trace 等数据提供了大规模、低成本、实时平台化服务。一站式提供数据采集、加工、分析、告警可视化与投递功能，全面提升研发、运维、运营和安全等场景数字化能力。阿里云日志服务是本系统收集、分析日志的重要工具。
- (6) **Prometheus:** Prometheus 是一个开源的系统监控和警报工具包。普罗米修斯使用度量名称和键/值对识别时间序列数据的多维数据模型。普罗米修斯可以很好地记录任何纯数

字时间序列。它既适用于以机器为中心的监视，也适用于高度动态的面向服务体系结构的监视。在微服务的世界中，它对多维数据收集和查询的支持是一个特别的优势。本系统即是使用普罗米修斯获取各个组件上各种时序数据。

- (7) **Vue.js:** Vue.js 是一个用于构建交互式 web 界面的库。它主要关注 MVVM 模式的 View-Model 层，并通过双向数据绑定连接视图和模型。它是一个用于构建用户界面的渐进式框架。与其他单片框架不同，Vue.js 从一开始就被设计成可增量的。核心库只关注于视图层，并且易于获取并与其他库或现有项目集成。另一方面，当与现代工具和支持库结合使用时，Vue.js 也完全能够为复杂的单页应用程序提供动力。本文使用 Vue.js 开发了系统前端架构。
- (8) **Echarts:** Echarts 是一个开源的 JavaScript 可视化工具，可以在 PC 和移动设备上流畅地运行。它与大多数现代网络浏览器兼容，例如 IE8/9/10/11、Chrome、Firefox、Safari 等等。ECharts 依靠 ZRender（一个图形渲染引擎）来创建直观、交互式和高度可定制的图表。本系统在向用户展示数据时，使用 Echarts 设计了各种图标。



图 6-7 IT 运维辅助系统相关技术

在模型方面，系统共使用了事件因果判别模型、组件-事件知识图谱表示学习模型、基于知识图谱和时序数据的故障预测模型。事件因果判别模型，输入事件对特征向量，输出事件对间是否存在因果关系判别结果。组件-事件知识图谱的表示学习模型在训练时，输入三元组及其背景拓扑结构，以三元组是否成立以及实体类别为损失函数训练模型。将训练完成的表示学习模型作为嵌入层，将故障类型对应的组件-事件知识图谱输入嵌入层，将事件序列输入双向记忆网络，最后以两者匹配度评分降序排列输出预测故障类型。基于以上模型，本系统可以实时展示集群各种维度信息状态、提前预测可能会出现的故障。

### 6.3.2 功能展示

下面分为各部分展示本系统所实现的功能：

- (1) 组件信息查看。图6-8为组件信息显示界面，在该界面中，组件类型、唯一标识和其间拓扑关系都会被直接展示出来。另外，为了显示组件的所有具体属性信息，在鼠标悬停于组件上时，右上方的灰色方框中会浮现该组件的属性信息列表。

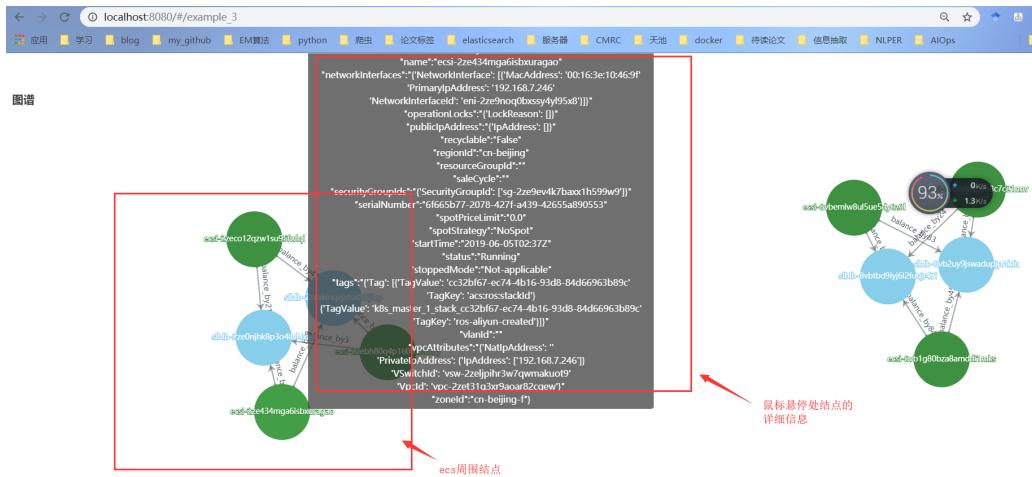


图 6-8 组件属性信息

对于组件的指标时序数据，可以通过鼠标左键点击组件，右侧抽屉会弹出该组件的各种指标时序数据。如图6-9所示，在指标时序数据折线图中，上方会显示该组件的唯一标识，中间有该组件多种指标时序数据类型可以选择，下方横坐标可以任意选择展示区间。



图 6-9 组件指标时序信息

- (2) 条件搜索。在进行运维工作时，运维人员会有不同的搜索需求。本系统实现了常用的搜索方式，图6-10显示了搜索符合指定路径类型关系（如 vpc-slb-ecs 路径类型关系）的所有拓扑路径。图6-11显示了模糊搜索符合某类型某 id 的组件（如 id 中包含字段 k1aexj 的 vpc 组件）及其周边结点。

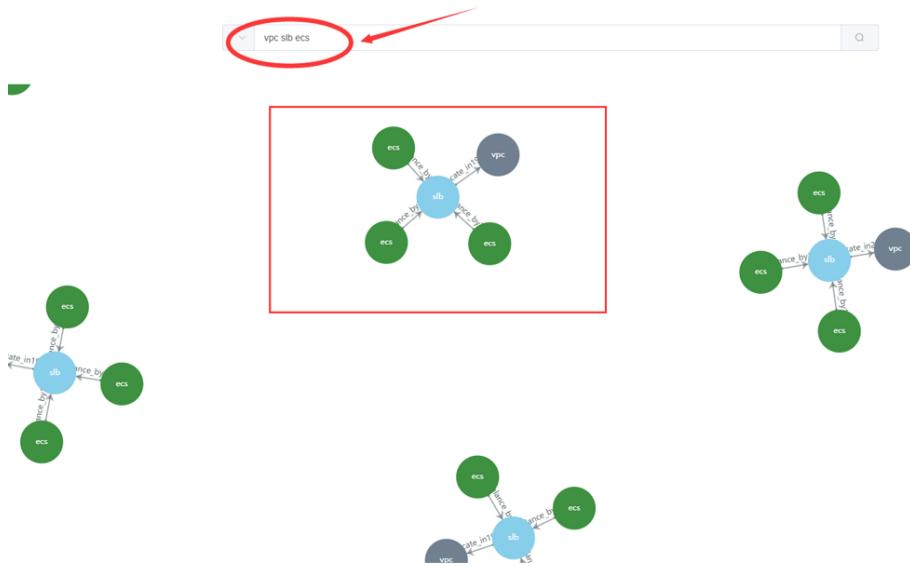


图 6-10 搜索指定路径

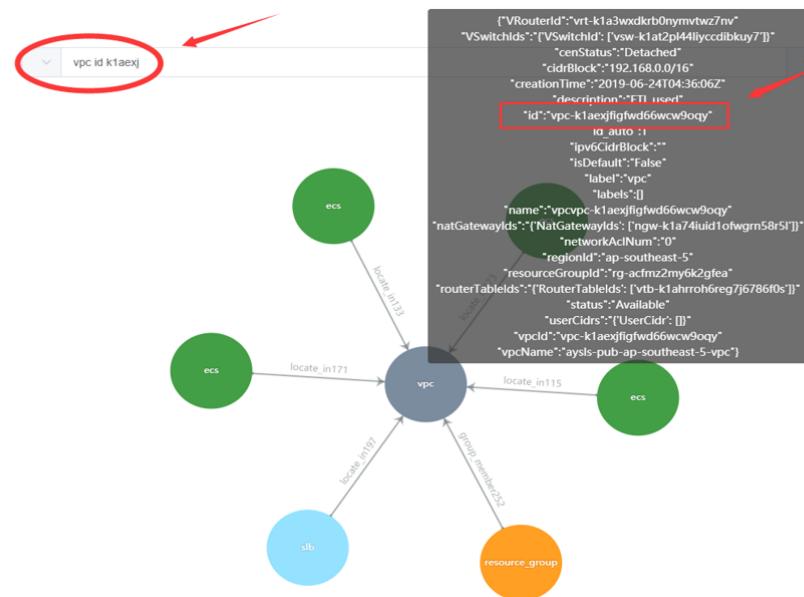


图 6-11 搜索指定结点

- (3) 查看指定故障类型的组件-事件知识图谱。图6-12展示了故障代号为f3的组件-事件知识图谱。为了便于查看，整个组件-事件知识图谱使用三层展示：最上层为事件层；中间层为微服务层；下层为集群组件层。



图 6-12 查看 f3 故障类型的组件-事件知识图谱

- (4) 故障预测。图6-13为实时故障预测界面。在该界面中，实时数据生成的组件-事件拓扑图同样以三层结构展示，但发生异常事件的微服务、系统组件节点会显示为红色。当实时数据经过后台故障预测模型判别会出现故障时，右边会弹出故障预测结果窗口，并按照故障类型匹配度由高至低排出前 3 个最匹配的结果。

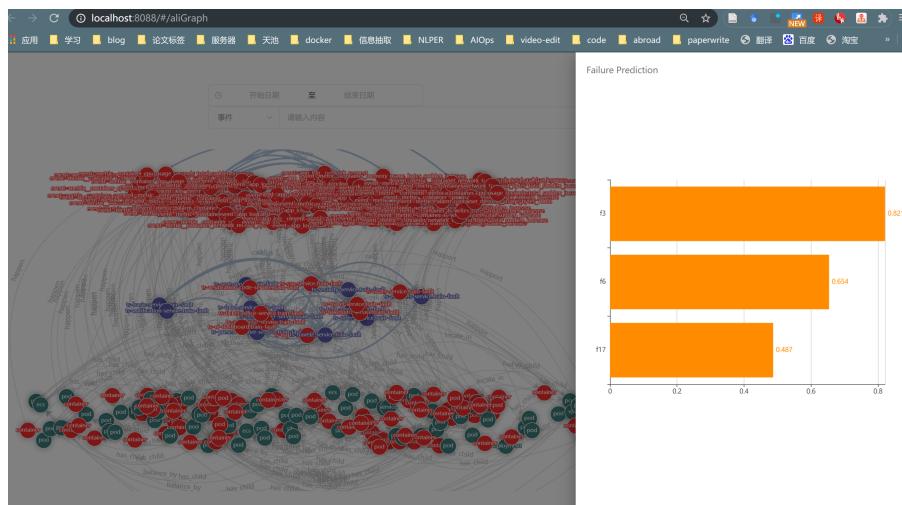


图 6-13 实时故障预测

## 6.4 本章小结

本章介绍了基于知识图谱的 IT 运维辅助系统，包括系统架构、系统功能和数据收集方式，并给出了系统功能的展示案例。

## 第七章 总结与展望

### 7.1 本文工作总结

本文首先分析了 IT 运维目前所遇到的难点、研究背景及研究意义，然后提出了基于知识图谱辅助 IT 运维的一套方案。首先，本文设计了事件特征，利用机器学习模型从硬件、软件、日志和指标时序数据中自动地构建组件-事件知识图谱。其次，本文设计了针对组件-事件知识图谱的动态表示学习模型，将实体表示分为语义表示和结构表示，实现了随着实体上下文变化动态地表示实体。最后，本文以上述表示学习模型作为嵌入层，利用双向记忆网络编码事件序列信息，再结合知识图谱进行了故障预测。本文不仅全面地获取并表示了系统多元异构信息，也结合了场景拓扑传播特征、先验知识、实时事件序列进行了表示学习和故障预测。本文完成的工作可分为以下 4 点：

- (1) 本文提出了从硬件、软件、日志、指标时序数据等多源异构数据中自动构建组件-事件知识图谱的方案。其中，在事件因果关系判别模块，本文设计了 6 种事件特征，有效解决了以往工作中需要人工标注海量数据的问题。
- (2) 本文针对组件-事件知识图谱设计了动态的知识表示学习模型。动态实体表示充分地适应了运维场景特性，即同一事件实体在不同上下文中出现，有着不同的触发逻辑，对应的故障类型也有所不同。
- (3) 本文引入知识图谱进行故障预测，提高了预测的准确率、细粒度和可解释性。本文的故障预测模型不仅可以预测是否会有故障产生，还会更细粒度的预测会有何种故障发生，并由知识图谱给出故障的触发逻辑。
- (4) 基于以上工作，设计了面向 IT 运维的可视化查询分析及故障预测系统。该系统有效地辅助运维人员实时查看系统运行状态，并由故障预测结果及时采取防范措施。

### 7.2 未来工作展望

本文深入调研 IT 运维难点及现有工作不足，提出了结合先验知识、场景实时特征的基于知识图谱的 IT 运维辅助方案。另外，本文所作工作仍有继续优化的空间。

- (1) 需要在工业界项目上进一步验证效果。目前本系统已在开源的分布式应用 train-ticket 和 sock-shop 上取得了良好效果。但工业界应用如淘宝、饿了吗、滴滴等，相较本文使用的两个开源应用，在客户请求量、系统复杂度上都要高出较多量级。因此，本文需要寻求在工业级应用中进一步验证，并采取相对应的优化措施。
- (3) 自动生成新的知识图谱，拓展知识库。本文虽然已模拟了常见的十余种故障，但仍难以涵盖实际运行中可能出现的新故障，需要添加业务逻辑自动收集过往未出现的新故障数据并沉淀生成对应的组件-事件知识图谱。

## 致谢

恍然间，三年硕士生活已然结束。在东南大学读研的日子里，指导老师、实验室同学和室友均给本人的学习和生活带来了良多帮助，诚心感谢硕士生涯所遇到的诸位良师益友。

科研是一种看似简单而又充满荆棘的认知探索。在科研中，研究生不仅需要有恒心毅力去坚持，还需要有分析问题、提出思路、验证思路的严谨方法论。本人的导师漆桂林教授在科研方面具有丰富的经验与严谨好学的求知态度，引领实验室营造了良好的科研氛围，激发了实验室同学们强烈的科研热情，并常常耐心地指导实验室同学们从事实际的科研工作。本人的硕士毕业论文从开题、模型设计、实验方法到撰写投递，漆老师都提供了细致深入的指导。此外，漆老师也经常在生活、职业规划、修身为人方面给予本人诸多教诲，珠玑之言帮助本人形成了正确的价值观与人生观，确立了持续学习、奉献祖国的人生态度。

本人也要感谢实验室的诸位同学。在初入实验室时，花云程博士分享了学校宿舍，使我得以在研究生入学前就开始参与实验室的科研项目。在工程项目中，李林、罗安源、吴畏、李震等同学帮助我补全科研知识、提高工程能力，一起顺利完成项目结题。实验室的师兄也经常提供科研与生活方面的建议，每当科研受阻、思路难寻、或生活受挫时，吴天星、花云程、谭亦鸣、吴桐桐、毕胜诸位博士都会提出宝贵的意见。在日常起居方面，本人真心感谢蒋志强、李同哲、丛宏宇、李蒙、支伯川和傅汉霖同学，不仅共同维护了良好的宿舍环境，也经常互相督促学习共同进步。

最后，本人真诚感谢父母、妹妹与弟弟，家人不仅给予了求学所需的物质基础，也在生活受挫时给予了我安慰与鼓励。幸福的家庭氛围给予了本人面对生活、工作和求学的本源信心与动力。

## 参考文献

- [1] Wang H, Nguyen P, Li J, et al. GRANO: Interactive graph-based root cause analysis for cloud-native distributed data platform[J]. Proceedings of the VLDB Endowment, 2019, 12(12): 1942-1945.
- [2] Nie X, Zhao Y, Sui K, et al. Mining causality graph for automatic web-based service diagnosis[C]. in: 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC). 2016: 1-8.
- [3] Nguyen H, Tan Y, Gu X. Propagation-aware anomaly localization for cloud hosted distributed applications[G]. in: Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques. 2011: 1-8.
- [4] Breiman L. Random forests[J]. Machine learning, 2001, 45(1): 5-32.
- [5] Qiu J, Du Q, Yin K, et al. A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications[J]. Applied Sciences, 2020, 10(6): 2166.
- [6] Bordes A, Glorot X, Weston J, et al. Joint learning of words and meaning representations for open-text semantic parsing[C]. in: Artificial Intelligence and Statistics. 2012: 127-135.
- [7] Bordes A, Weston J, Collobert R, et al. Learning structured embeddings of knowledge bases[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 25: 1. 2011.
- [8] Bordes A, Usunier N, Garcia-Duran A, et al. Translating embeddings for modeling multi-relational data[C]. in: Neural Information Processing Systems (NIPS). 2013: 1-9.
- [9] Wang Z, Zhang J, Feng J, et al. Knowledge graph embedding by translating on hyperplanes[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 28: 1. 2014.
- [10] Lin Y, Liu Z, Sun M, et al. Learning entity and relation embeddings for knowledge graph completion[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 29: 1. 2015.
- [11] Ji G, He S, Xu L, et al. Knowledge graph embedding via dynamic mapping matrix[C]. in: Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers). 2015: 687-696.
- [12] Ji G, Liu K, He S, et al. Knowledge graph completion with adaptive sparse transfer matrix[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 30: 1. 2016.
- [13] Feng J, Huang M, Wang M, et al. Knowledge graph embedding by flexible translation[C]. in: Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning. 2016: 557-560.
- [14] Ou M, Cui P, Pei J, et al. Asymmetric transitivity preserving graph embedding[C]. in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. 2016: 1105-1114.
- [15] Jenatton R, Le Roux N, Bordes A, et al. A latent factor model for highly multi-relational data[C]. in: Advances in Neural Information Processing Systems 25 (NIPS 2012). 2012: 3176-3184.
- [16] Trouillon T, Welbl J, Riedel S, et al. Complex embeddings for simple link prediction[C]. in: International Conference on Machine Learning. 2016: 2071-2080.
- [17] Liu H, Wu Y, Yang Y. Analogical inference for multi-relational embeddings[C]. in: International conference on machine learning. 2017: 2168-2178.

- [18] Socher R, Chen D, Manning C D, et al. Reasoning with neural tensor networks for knowledge base completion[C]. in: Advances in neural information processing systems. 2013: 926-934.
- [19] Dettmers T, Minervini P, Stenetorp P, et al. Convolutional 2d knowledge graph embeddings[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 32: 1. 2018.
- [20] Guo S, Wang Q, Wang B, et al. Semantically smooth knowledge graph embedding[C]. in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2015: 84-94.
- [21] Xie R, Liu Z, Sun M. Representation Learning of Knowledge Graphs with Hierarchical Types.[C]. in: Proceedings of International Joint Conference on Artificial Intelligent (IJCAI). 2016: 2965-2971.
- [22] Lin Y, Liu Z, Luan H, et al. Modeling relation paths for representation learning of knowledge bases[C]. in: Empirical Methods Natural Language Process. 2015: 705-714.
- [23] Wang Z, Li J, Liu Z, et al. Text-enhanced representation learning for knowledge graph[C]. in: Proceedings of International Joint Conference on Artificial Intelligent (IJCAI). 2016: 4-17.
- [24] Feng J, Huang M, Yang Y, et al. GAKE: Graph aware knowledge embedding[C]. in: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. 2016: 641-651.
- [25] Shi J, Gao H, Qi G, et al. Knowledge graph embedding with triple context[C]. in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. 2017: 2299-2302.
- [26] Schlichtkrull M, Kipf T N, Bloem P, et al. Modeling relational data with graph convolutional networks[C]. in: European semantic web conference. 2018: 593-607.
- [27] Salfner F, Lenk M, Malek M. A survey of online failure prediction methods[J]. ACM Computing Surveys (CSUR), 2010, 42(3): 1-42.
- [28] Tan Y, Nguyen H, Shen Z, et al. Prepare: Predictive performance anomaly prevention for virtualized cloud systems[C]. in: 2012 IEEE 32nd International Conference on Distributed Computing Systems. 2012: 285-294.
- [29] Pitakrat T, Okanović D, van Hoorn A, et al. Hora: Architecture-aware online failure prediction[J]. Journal of Systems and Software, 2018, 137: 669-685.
- [30] Zhang S, Liu Y, Meng W, et al. Prefix: Switch failure prediction in datacenter networks[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2018, 2(1): 1-29.
- [31] Baldoni R, Montanari L, Rizzuto M. On-line failure prediction in safety-critical systems[J]. Future Generation Computer Systems, 2015, 45: 123-132.
- [32] Dalmazo B L, Vilela J P, Curado M. Predicting traffic in the cloud: a statistical approach[C]. in: 2013 International Conference on Cloud and Green Computing. 2013: 121-126.
- [33] Sladescu M, Fekete A, Lee K, et al. Event aware workload prediction: A study using auction events[C]. in: International Conference on Web Information Systems Engineering. 2012: 368-381.
- [34] Purushotham V, Narayanan S, Prasad S A. Multi-fault diagnosis of rolling bearing elements using wavelet analysis and hidden Markov model based fault recognition[J]. Ndt & E International, 2005, 38(8): 654-664.
- [35] Boutros T, Liang M. Detection and diagnosis of bearing and cutting tool faults using hidden Markov models[J]. Mechanical Systems and Signal Processing, 2011, 25(6): 2102-2124.
- [36] Xu C, Wang G, Liu X, et al. Health status assessment and failure prediction for hard drives with recurrent neural networks[J]. IEEE Transactions on Computers, 2016, 65(11): 3502-3508.

- [37] Cheng Y, Zhu H, Wu J, et al. Machine health monitoring using adaptive kernel spectral clustering and deep long short-term memory recurrent neural networks[J]. IEEE Transactions on Industrial Informatics, 2018, 15(2): 987-997.
- [38] Du M, Li F, Zheng G, et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning[C]. in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 1285-1298.
- [39] Das A, Mueller F, Siegel C, et al. Desh: deep learning for system health prediction of lead times to failure in hpc[C]. in: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. 2018: 40-51.
- [40] Islam T, Manivannan D. Predicting application failure in cloud: A machine learning approach[C]. in: 2017 IEEE International Conference on Cognitive Computing (ICCC). 2017: 24-31.
- [41] Li Y, Jiang Z M, Li H, et al. Predicting Node Failures in an Ultra-large-scale Cloud Computing Platform: an AIOps Solution[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2020, 29(2): 1-24.
- [42] Gao J, Wang H, Shen H. Task failure prediction in cloud data centers using deep learning[J]. IEEE Transactions on Services Computing, 2020.
- [43] Bernstein D. Containers and cloud: From lxc to docker to kubernetes[J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
- [44] Mengistu D M. Distributed Microservice Tracing Systems: Open-source tracing implementation for distributed Microservices build in Spring framework[J], 2020.
- [45] Zhou X, Peng X, Xie T, et al. Poster: Benchmarking microservice systems for software engineering research[C]. in: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). 2018: 323-324.
- [46] Rahman J, Lama P. Predicting the end-to-end tail latency of containerized microservices in the cloud[C]. in: 2019 IEEE International Conference on Cloud Engineering (IC2E). 2019: 200-210.
- [47] Yang L, Yang N. An Integrated Event Summarization Approach for Complex System Management[J]. IEEE Transactions on Network and Service Management, 2019, 16(2): 550-562.
- [48] Landauer M, Skopik F, Wurzenberger M, et al. System log clustering approaches for cyber security applications: A survey[J]. Computers & Security, 2020, 92: 101739.
- [49] Mueen A, Keogh E. Extracting optimal performance from dynamic time warping[C]. in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016: 2129-2130.
- [50] Joachims T. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.[R]. Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.
- [51] Guo S, Wang Q, Wang L, et al. Knowledge graph embedding with iterative guidance from soft rules[C]. in: Proceedings of the AAAI Conference on Artificial Intelligence: vol. 32: 1. 2018.
- [52] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality[C]. in: Advances in Neural Information Processing Systems. 2013: 3111-3119.
- [53] Peters M E, Neumann M, Iyyer M, et al. Deep Contextualized Word Representations[C]. in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT. 2018: 2227-2237.

- [54] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]. in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 4171-4186.
- [55] Vaswani A, Shazeer N, Parmar N, et al. Attention is All you Need[C]. in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems. 2017: 5998-6008.
- [56] Kipf T N, Welling M. Semi-Supervised Classification with Graph Convolutional Networks[C]. in: 5th International Conference on Learning Representations. 2017.
- [57] Duvenaud D, Maclaurin D, Aguilera-Iparraguirre J, et al. Convolutional Networks on Graphs for Learning Molecular Fingerprints[C]. in: Advances in Neural Information Processing Systems. 2015: 2224-2232.
- [58] Yang B, Yih W, He X, et al. Embedding Entities and Relations for Learning and Inference in Knowledge Bases[C]. in: 3rd International Conference on Learning Representations (ICLR). 2015.
- [59] Boettiger C. An introduction to Docker for reproducible research[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1): 71-79.
- [60] Padgham L, Winikoff M. Prometheus: A methodology for developing intelligent agents[C]. in: International Workshop on Agent-Oriented Software Engineering. 2002: 174-185.

## 作者攻读硕士学位期间的研究成果

### 发表的论文

- [1] 智能运维中事件知识图谱的构建与事件表示学习 [J]. 东南大学计算机科学与工程学院学术论坛, 2018. (第一作者)





