

学校代码: 10286

分 类 号: TP311.5

密 级: 公开

U D C: 004.41

学 号: 184607



东南大学

SOUTHEAST UNIVERSITY

工程硕士学位论文 基于知识图谱的 IT 运维辅助系统设计 与实现

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

工程领域名称 软件工程 论文答辩日期 2021年6月12日

研究方向 软件工程 学位授予日期

答辩委员会主席 何洁月教授 评阅人 盲审

2021年6月12日

東南大學

工程硕士学位论文

基于知识图谱的 IT 运维辅助系统设计 与实现

专业名称: 软件工程

研究生姓名: 方苏东

导师姓名: 漆桂林 教授

丁岩 高工

DESIGN AND IMPLEMENTATION 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

College of Software Engineering
Southeast University

2021/6/12

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、准确的运维决策来应对各种故障。近年来，智能运维（Artificial Intelligence for IT Operations, AIOps）通过引入人工智能技术，提升了IT运维效率。

然而，在实际场景中，IT运维依然面临着三个问题：多源异构数据难以整合、运维知识表示不足和故障难以准确预知。在多元异构数据整合方面，已有的运行状态监测模型都只能整合片面的数据，导致沉淀的运维知识也是片面的。在运维知识表示方面，传统的表示方法局限于知识的显示结构，忽略了运维知识的深层含义。在故障预测方面，现有的故障预测方法没有引入运维知识，预测结果缺乏可解释性，可靠性低。

基于此，本文提出了有效解决上述问题的IT运维辅助技术，主要包括：

(1) 提出了自动化构建组件-事件知识图谱的方法，整合了横跨硬件、软件、日志和运行指标的所有数据，使用了机器学习模型生成组件-事件知识图谱，减少了知识图谱构建的人工消耗，解决了多源异构数据难以整合的问题。

(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变化的动态表示，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，解决了运维知识表示不足的问题。

(3) 提出了引入组件-事件知识图谱的故障预测模型，利用知识图谱识别事件序列中的关键信息，预测出最匹配的故障类型，提高了预测结果的细粒度，增强了可解释性，解决了故障难以准确预知的问题。

(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。

综上所述，本文利用历史数据自动化构建了知识图谱，提出了针对该类知识图谱的表示学习模型，并将知识图谱引入到了故障预测中，最终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 failure. 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 deal with various failures. In recent years, Artificial Intelligence for IT Operations (AIOps) has improved the efficiency of IT operation and maintenance by introducing artificial intelligence technology.

However, in the actual scene, IT operation and maintenance still faces three problems: it is difficult to integrate multi-source heterogeneous data, insufficient knowledge of operation and maintenance and it is difficult to accurately predict faults. In the aspect of integration of multiple heterogeneous data, the existing operational status monitoring models can only integrate one-sided data, which leads to one-sided operation and maintenance knowledge. In the aspect of operation and maintenance knowledge representation, the traditional representation method is limited to the display structure of knowledge, ignoring the deep meaning of operation and maintenance knowledge. In the aspect of fault prediction, the existing fault prediction methods do not introduce operation and maintenance knowledge, and the prediction results are lack of interpretability and low reliability.

Based on this, this paper puts forward an IT operation and maintenance assistant technology to effectively solve the above problems, which mainly includes:

(1) An automatic method of building component-event knowledge graph is proposed, which integrates all data spanning hardware, software, logs and operational indicators, and uses machine learning model to generate component-event knowledge graph, thus reducing the labor consumption of building knowledge graph and solving the problem that multi-source heterogeneous data is difficult to integrate.

(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 changing with context. It achieves the best results in the task of component-event knowledge graph triple classification and link prediction, and solves the problem of insufficient operation and maintenance knowledge representation.

(3) A fault prediction model based on component-event knowledge graph is proposed. The key information in event sequence is identified by knowledge graph, and the best matching fault type is predicted. The fine granularity of prediction result is improved, the interpretability is enhanced, and the problem that the fault is difficult to predict accurately is solved.

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

To sum up, this paper uses historical data to build knowledge graph automatically, puts forward

a representation learning model for this kind of knowledge graph, introduces knowledge graph into fault prediction, and finally designs and implements an IT operation and maintenance assistant system based on knowledge graph.

Keywords: Failure Prediction, IT Operation and Maintenance, Knowledge Graph, Knowledge Graph Embedding

目录

插图目录

表格目录

第一章 绪论

1.1 研究背景与意义

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查询分析分布式集群数据，进而导致故障未被及时发现而进一步扩散，最终导致应用不可用，带来巨大的经济损失。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。近些年被提出的智能运维 (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，用于完成异常检测、异常分类和故障预测等任务，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。由于缺少对运维知识的研究，AIOps 仍然无法解决 IT 运维所面临的诸多问题。

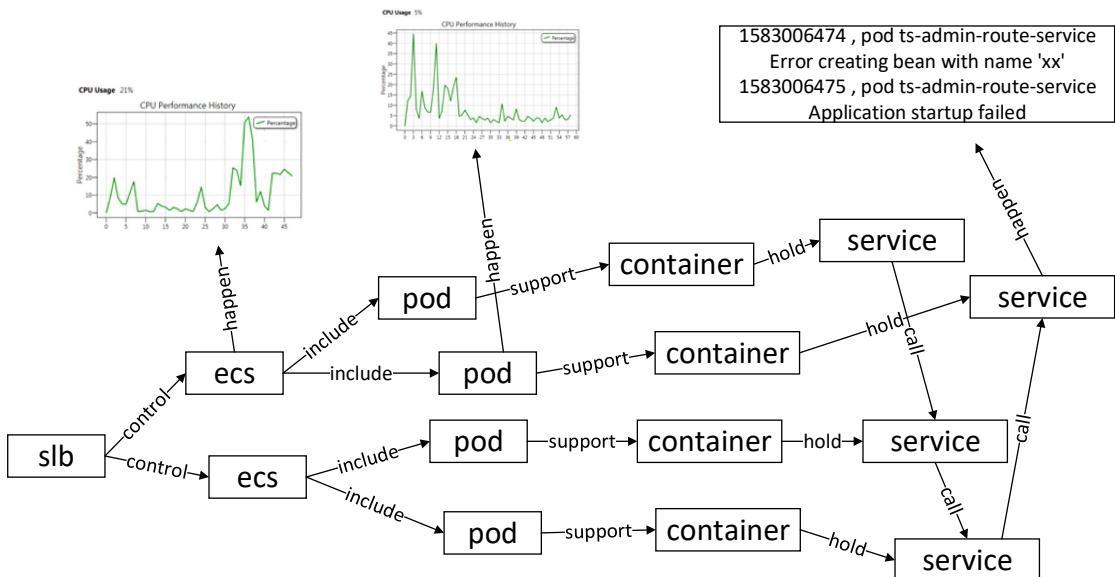


图 1-1 分布式集群信息关系示意图

首先，如何全面地整合多源异构数据，来获取运维知识始终是 IT 运维面临的一个问题。分布式集群中的数据不仅形式复杂多样，而且数量庞大。如图??所示，分布式集群中可用的数据包括组件间的拓扑关系、硬件的指标时序数据、软件的运行日志数据。组件间的拓扑关

系指明了组件间的交互、依赖等关系，如负载均衡器（Server Load Balancer, SLB）会调控多台云服务器（Elastic Compute Service, 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文本。随着分布式集群的实时运转，平均每秒钟产生的各类数据可达上万条。

已有的运行状态监测模型wang2019grano, nie2016mining-causality-graph, qiu2020causality-mining-knowledge-graph 可以实时监测、收集分布式集群中的数据，并进一步沉淀运维知识。但其始终无法整合全部数据，都只监测到了片面的数据，导致获取到的运维知识也是片面的。现有运行状态监测模型的覆盖率、细粒度都有待提高，这样才能提升运维知识的全面性和通用性。

另外，如何有效地表示运维知识，也是 IT 运维一个亟待解决的问题。文献nie2016mining-causality-graph, qiu2020causality-mining-knowledge-graph, tan2012prepare 将运维知识拓扑视作贝叶斯网，随后利用贝叶斯推理辅助运维工作。但贝叶斯网建立于条件独立性假设上，其认为给定父节点后，每个子节点与其非后代节点条件独立。事实上，在 IT 运维场景中，条件独立性假设是不成立的，比如故障产生与已发生的多条异常信息、系统整体状态都息息相关。这种运用贝叶斯网建模表示运维知识的方式，对 IT 运维辅助工作有一定提升，但局限于知识的显示结构，忽略了运维知识的深层逻辑。运维知识的表示，仍具有巨大提升空间。

IT 运维还面临着另一个问题是如何准确预知未来可能会出现的故障。故障预测是辅助 IT 运维的代表性工作，其通过预测可能会发生的故障，提醒运维人员及时采取预防措施。当前，基于深度学习的故障预测方法xu2016health, cheng2018machine, du2017deeplog, das2018desh, islam2017predicting, li2020predicting, gao2020task 是研究的热点，其可以在故障预测任务上达到很好的预测效果，如文献gao2020task可以达到 87% 的准确度和 86% 的 F1 值。但是该类方法只能预测到是否会有故障发生，不能具体到会出现何种故障。另外，由于没引入运维知识，也不能给出故障触发过程，缺乏可解释性，可靠性低。在故障预测中引入运维知识，提高预测结果细粒度，增强预测可解释性，是目前亟待展开的工作。

综上所述，目前在 IT 运维领域存在着以下挑战：

- (1) 在沉淀运维知识时，存在多源异构数据难以整合的挑战，已有的运行状态监测模型从不同方面获取集群运行状态信息，但都具有片面性，存在着巨大的提升空间。
- (2) 在表示运维知识时，存在运维知识表示不足的挑战，基于贝叶斯网建模表示的方式局限于显示的知识拓扑结构，需要进一步展开针对运维知识的表示学习研究。
- (3) 在故障预测时，存在着故障难以准确预知的挑战，只依靠深度学习的故障预测方法预测结果细粒度低，缺乏可解释性，需要展开引入运维知识进行故障预测的研究。
- (4) 针对上述的挑战，需要实现一种满足运维人员需求的 IT 运维辅助系统。该系统需要能够监测多源异构数据，有效地沉淀、表示运维知识，并用于实时的故障预测。

基于以上背景信息，本文收集了横跨软硬件的所有信息，构建了组件-事件知识图谱。其次，本文提出了组件-事件知识图谱实体的动态表示方案。随后，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，进行故障预测。最后，本文设计并实现了一个基于知识图谱的 IT 运维辅助系统。

1.2 国内外研究现状

为了解决 IT 运维实际中面临的问题，包括多源异构数据难以整合、运维知识表示不足和故障难以准确预知，国内外已经展开了大量研究。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，并进行了分析。

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^{wang2019grano} 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nie2016mining-causality-graph首先利用异常检测方法nguyen2011pal 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breiman2001randomforest 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qiu2020causality-mining-knowledge-graph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”qiu2020causality-mining-knowledge-graph，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bordes2012joint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bordes2011learning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^{bordes2013translatingE} 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^{wang2014knowledge} 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^{lin2015learning} 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^{ji2015knowledge} 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^{ji2016knowledge}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^{feng2016knowledge}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^{ou2016asymmetric}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^{jenatton2012latent}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^{trouillon2016complex}为了建模非对称关系将表示扩展到复数向量空间；ANALOGY^{liu2017analogue}将知识图谱中的类比关系进行了建模；SLM^{socher2013reasoning}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^{socher2013reasoning}使用张量网络捕获头尾实体间的语义关联；ConvE^{dettmers2018convolutional}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^{guo2015semantically}和TKRL^{xie2016representation}将实体的类别信息引入了知识表示学习中；文献lin2015modeling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socher2013reasoning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xie2016representation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wang2016text选择将实体与文本库词汇对齐；GAKE^{feng2016gake}和TCE^{shi2017knowledge}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schlichtkrull2018modeling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salfner2010survey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tan2012prepare, pitakrat2018hora, zhang2018prefix, baldoni2015line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salfner2010survey。文献dalmazo2013predicting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sladescu2012event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purushotham2005multi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boutros2011detection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xu2016health, cheng2018machine, du2017deeplog, das2018desh, islam2017predicting, li2020predicting开始使用深度学习方法，如循环神经网络（Recurrent Neural Networks, RNN）和长短期记忆神经网络（Long Short-Term Memory, LSTM）进行故障预测。文献gao2020task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 论文研究目标与内容

1.3.1 研究目标

本文的研究目标是设计并实现一个满足实际运维需求的 IT 运维辅助系统。为了达到研究目标，本文分析了目前的研究难点，并逐一展开了调研，总结了现有工作存在的问题。在此基础上，本文提出了一个基于知识图谱的 IT 运维辅助系统的设计方案。该方案基于本文提出的算法模型，将知识图谱构建、知识图谱表示学习、故障预测封装为多个功能模块，以应对多源异构数据难以整合、运维知识表示不足和故障难以准确预知等问题。

1.3.2 研究内容

本文主要开展了以下研究工作：

(1) 研究横跨硬件、软件、异常数据的组件-事件知识图谱自动构建方案。一方面研究如何让知识图谱整合高细粒度的多种信息，如软硬组件间关系、指标时序数据、日志数据。另一方面研究如何设计事件特征，使用机器学习模型挖掘事件间因果关系。最终实现从历史数据中沉淀出故障类型对应的组件-事件知识图谱。

(2) 研究针对组件-事件知识图谱的知识表示学习模型。在组件-事件知识图谱中，异常会沿着组件拓扑关系传递，事件间有因果触发关系，实体在不同的上下文中也有不同的含义。针对组件-事件知识图谱的这些特性，需要重点研究如何让实体随上下文变化动态地表示。

(3) 研究引入组件-事件知识图谱的故障预测模型。为了解决深度学习模型在故障预测时，可解释性低、预测结果细粒度低的问题，需要研究如何结合知识图谱与实时数据进行故障预测。最终使得预测结果可以具体到会出现何种故障，并能结合知识图谱得到触发逻辑。

(4) 研究基于知识图谱的 IT 运维辅助系统的设计与实现。为了满足运维人员使用需求，需要研究如何整合业务流程。重点研究如何实现分布式集群不同信息的实时快速获取，以及如何向运维人员展示运行状态信息和故障预测结果。

1.4 论文结构安排

本文篇幅结构共有七章，各个章节间的关系如图??所示，每一章内容如下：

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内容。

第二章为相关技术，描述了本文会涉及到的相关概念及技术，为下文各个方法模型的提出做好铺垫。

第三章为组件-事件知识图谱的构建流程，主要介绍了系统运行状态信息获取，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，在模拟构建的数据集上进行了实验，并分析了实验结果。

第四章为组件-事件知识图谱的知识表示学习，主要分析了场景特性，介绍了随上下文变化的动态表示学习方案，在三元组分类和链接预测任务上进行了对比实验，并分析了实验结果。

第五章为引入组件-事件知识图谱的故障预测，主要介绍了引入组件-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，在事件序列故障预测任务上进行了对比实验，并分析了实验结果。

第六章为系统设计与实现，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。

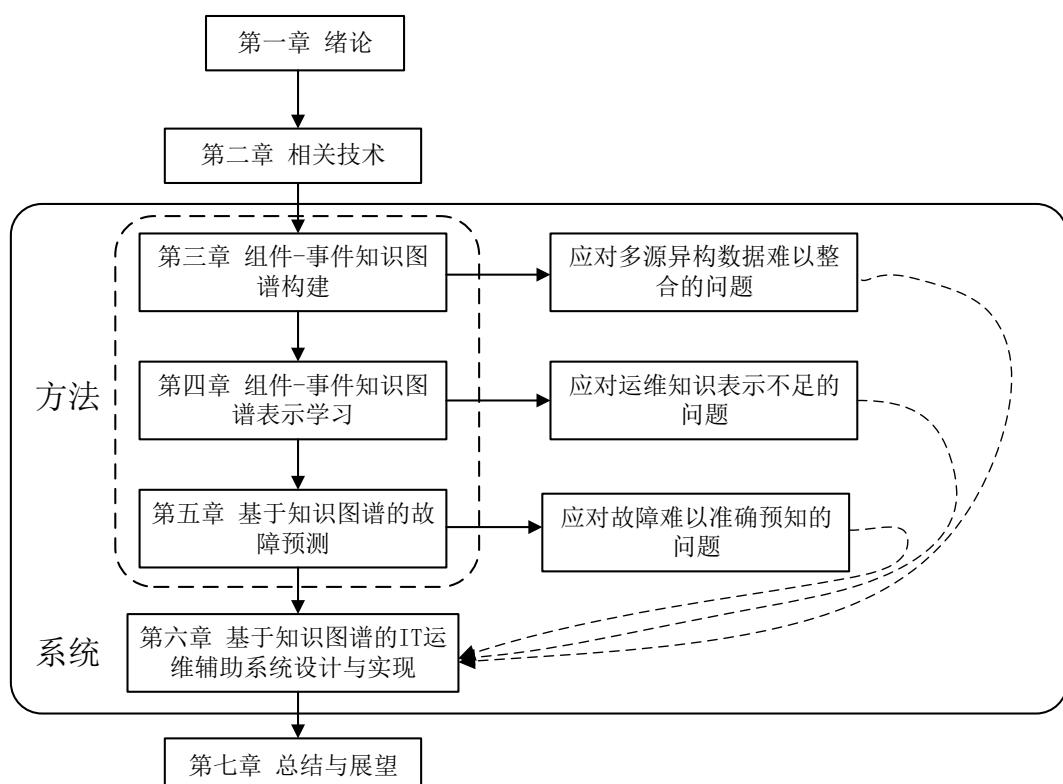


图 1-2 论文章节关系示意图

第二章 组件-事件知识图谱构建

目前已有的分布式集群运行状态监测模型都不能覆盖整合所有种类的信息。文献wang2019grano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nie2016mining-causality-graph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qiu2020causality-mining-knowledge-graph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“Container1 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。

为了解决上述多源异构数据难以覆盖整合的问题，本章提出了横跨硬件、软件、异常数据的组件-事件知识图谱构建方案。本章共分为5个小节展开介绍：首先，描述了组件-事件知识图谱构建的总体框架；其次，分3个小节分别介绍组件层信息获取，事件层信息获取和组件-事件知识图谱生成；最后模拟构建数据集，并对事件因果关系判别模型进行了实验分析。

2.1 总体框架

组件-事件知识图谱的总体构建框架如图??所示。一方面，通过公开 API (Application Programming Interface) 和微服务追踪系统，获取组件唯一标识符、属性和关联关系，得到了组件层图谱；另一方面，将日志、指标时序数据等运行时信息转为了事件类型数据，然后引入了新的事件特征用于发掘事件间因果关系，得到了事件层图谱。最后，将同故障类型的组件层图谱、事件层图谱连接起来，自动沉淀出了标有对应故障类型的组件-事件知识图谱。

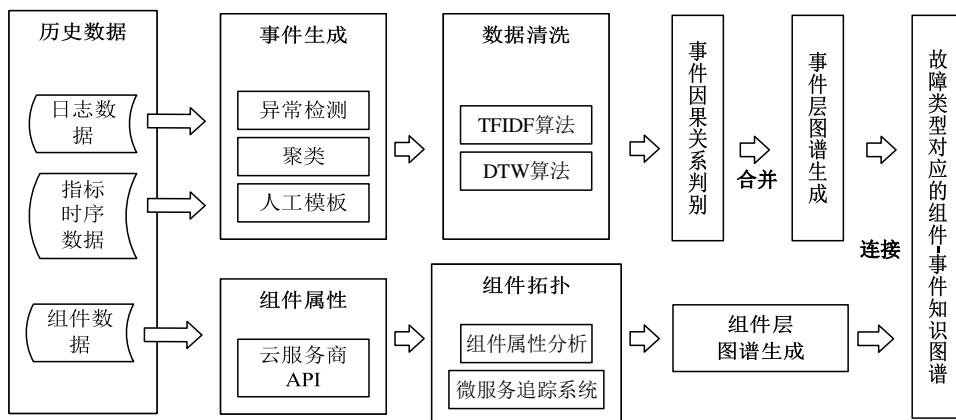


图 2-1 组件-事件知识图谱构建总体框架图

2.2 组件层信息获取

Kubernetes (k8s) 集群berNSTEIN2014containers 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。Kubernetes 中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??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）负责提供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。

对于组件属性信息，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 属性信息的 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??为调用 API 后返回的杭州区域一台 ECS 的信息列表，包含该 ECS 的资源组、内存、唯一标识符、创建时间等属性信息。此外，VPC、SLB、Pod 和 Container 都有着对应的 API。在获取组件属性时，本文分别调用不同的 API 设定不同参数，最终可以获取到同一集群内所有组件的属性信息。

对于组件间拓扑关系，VPC、SLB、ECS、Pod 和 Container 间的拓扑关系可以通过分析属性信息获取，而 Service 间的拓扑则需要借助微服务追踪系统获取。同样以图??为例，返回信息中 `VpcId` 字段就表明了该 ECS 位于 id 为 “vpc—2zeupjh 08tt7q3brd****” 的虚拟私有网络下。除了 Service 类型组件，所有集群组件均可通过分析 API 返回的属性信息，获取到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有关，无法通过云服务商提供的

公开 API 直接获取。另外，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^{mengistu2020distributed}），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。

综上，本文通过云服务商提供的公开 API 和微服务追踪系统，获取了组件部分的实体属性及拓扑结构。

```
{
  "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": [
              "172.17.*.*"
            ]
          },
          "VpcId": "vpc-2zeuphj08tt7q3brd****",
          "VSwitchId": "vsw-2zeh0r1pabwtg6wcs****",
          "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

2.3 事件层信息获取

本小节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。本小节分为两部分展开介绍，分别为事件生成和事件因果关系挖掘。

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，过往工作大量依赖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。为了克服该问题，本文设计并引入了新的事件特征，最终不需要反复标注海量数据就可以达到优质的效果。

2.3.1 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。

事件定义如 Definition ??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^{zhou2018poster}、sock-shop^{rahman2019predicting}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。

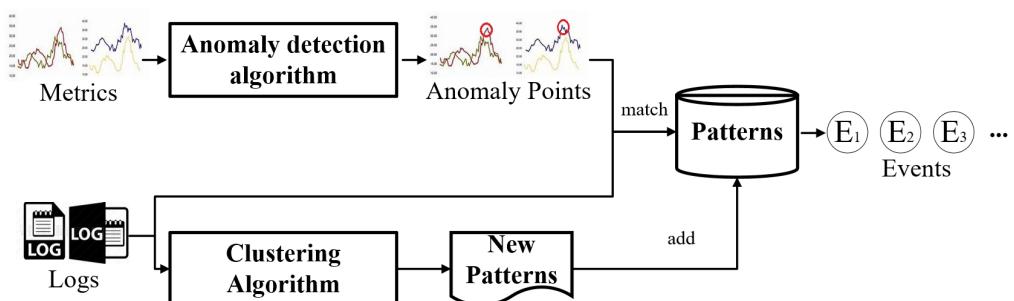


图 2-3 事件生成流程

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??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??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 yang2019integrated 和聚类算法 landauer2020system 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。

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

图 2-4 日志数据样例

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 ??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^{mueen2016extracting} 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。

本文生成的事件数据按照来源划分，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表??所示。

表 2-2 事件类型层次关系

事件大类别	细分	类别数目	
		(train-ticket)	(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

其中，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，如表??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。

表 2-3 异常事件监控项列表

监控项名称
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??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。

表 2-4 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%

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，表??展示了部分 k8s 日志事件类型。

表 2-5 部分 k8s 日志事件类型

k8s 日志事件类型名
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^{boettiger2015introduction} 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??中列出了部分 DockerIO 日志事件类型名称。

2.3.2 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。本文共使用了以下 6 种事件特征，其中（1）（2）源自相关工作^{nie2016mining-causality-graph}，（3）-（6）为本文新设计的事件特征。

表 2-6 部分 DockerIO 日志事件类型

DockerIO 日志事件类型
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

(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??所示。其中 X, Y 是事件 A, B 分别在时间维度上展开的向量， \bar{X}, \bar{Y} 是对应的样本平均值。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。用 $\text{count}(AB)$ 表示 AB 共同出现的次数， $\text{count}(A)$ 表示 A 出现的次数。计算方式如公式??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??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??所示：统计 A 出现的时间戳序列 A_{stamps} ，统计 B 出现的时间戳序列 B_{stamps} 。统计每当 A 出现后 B 出现的时间差序列 $intervals$ 。最终求均值 $mean(intervals)$ 再将取值调整为均大于 0，即可得到 A, B 间平均时间差。

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

(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??所示：统计事件 A 出现的时间戳序列 A_{stamps} ，随后对事件戳序列差分 $diff(\cdot)$ 求标准差 $std(\cdot)$ 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 $diff(\cdot)$ 表示求差分， $std(\cdot)$ 表示求标准差。

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

(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??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。

2.4 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：

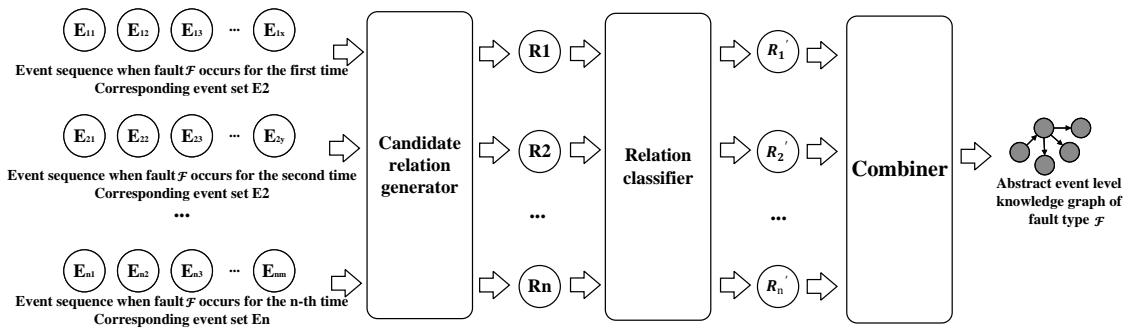
TypeId: 抽象事件唯一标识；

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-事件知识图谱对应的故障类型名。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 Definition?? 和组件-事件知识图谱的定义 Definition??。

在 Definition?? 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-x6hwl 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 F 的组件-事件知识图谱生成流程

图??是故障类型 F 对应的抽象事件层知识图谱构建流程示意图。图中构建流程可以使用算法??表示。

算法 2.1 组件-事件知识图谱构建算法

输入: 故障类型 F 发生 n 次分别对应的事件集合 E_1, E_2, \dots, E_n , 分别对应的组件拓扑图 D_1, D_2, \dots, D_n

输出: 故障类型 F 对应的组件-事件知识图谱 kg_F

- 1: $i \leftarrow 0$
- 2: **while** $i < n$ **do**
- 3: $R_i \leftarrow$ 生成候选关系集合 (E_i)
- 4: $\mathbf{R}_i' \leftarrow$ 事件因果关系判别模型 (R_i)
- 5: **end while**
- 6: $\mathcal{E}_{cause} = \mathbf{R}_1' \cup \mathbf{R}_2' \cup \dots \cup \mathbf{R}_n'$
- 7: $\mathcal{D}_{graph} = D_1 \cup D_2 \cup \dots \cup D_n$
- 8: $\mathcal{E}_{graph} = \text{Combiner}(\mathcal{E}_{cause})$
- 9: $kg_F \leftarrow link(\mathcal{D}_{graph}, \mathcal{E}_{graph})$
- 10: **return** kg_F

具体上，故障类型 F 对应组件-事件知识图谱的构建步骤描述如下：

(1) 生成候选关系集合：获取故障类型 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^{joachims1996probabilistic} 和 DTW 算法^{mueen2016extracting} 筛去部分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 $\mathbf{R}_1', \mathbf{R}_2', \dots, \mathbf{R}_n'$ 。

(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 $\mathbf{R}_1', \mathbf{R}_2', \dots, \mathbf{R}_n'$ 取并集，融合 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 F 故障的抽象事件因果关系集合 $\mathcal{E}_{cause} = \mathbf{R}_1' \cup \mathbf{R}_2' \cup \dots \cup \mathbf{R}_n'$ 。

(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**qiu2020causality-mining-knowledge-graph**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??是上述构建过程生成的故障类型“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。

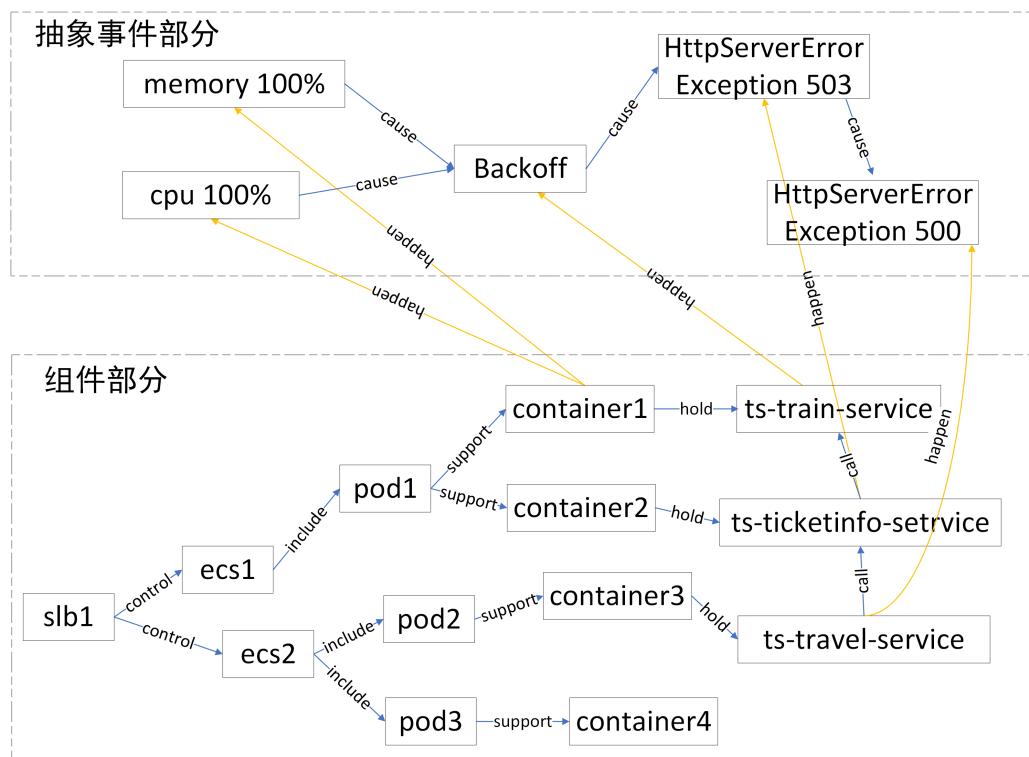


图 2-6 数据库不可用无法查票故障对应知识图谱部分示例

2.5 实验与分析

由于组件间拓扑关系、组件到抽象事件的产生关系都可以准确获取，影响组件-事件知识图谱质量的只有事件间因果关系。因此，本节重点关注事件因果关系判别的效果，进行了相关实验的设计与实施。本节实验目的在于验证使用的 6 种事件特征，可以提升事件因果判别

模型效果。本节共分为 4 小节介绍实验，包括数据集构建、评测指标、实验方法和实验结果分析。

2.5.1 数据集构建

实验采用了阿里云 kubernetes 集群，集群各个组件数量如表??所示。在该集群上部署了文献zhou2018poster, rahman2019predicting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??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。

表 2-7 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

本实验对数据集中的关系种类进行了标注，标注数据覆盖了每一种故障类型。其中对于每种故障类型，随机抽取了该类故障类型 2 个具体时间段的数据进行了标注，只对认为有因果关系的事件对标注为 1，其余事件对默认无因果关系标注为 0。

表 2-8 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	491	573
故障种类	10	17
事件类型种类	204	211

由于事件因果判别模型是根据事件对统计特征发掘事件间因果关系，与事件来自何种应用无关，所以本块实验不区分事件来自于 train-ticket 还是 sock-shop，直接把两个应用来源的标注事件对整合起来。最终标注为 1 的事件对共计 286 条，其余默认无关系的共计 8,250,290 条。数据集 train-ticket、sock-shop 的标注信息如下表??所示。由于正负样本数量级差距较大，

后续选择了按照正负样本比 1:10 随机抽取了一批负样本数据用于训练。抽取出用于训练的数据共包含正样本 286 条，负样本 2860 条。

表 2-9 事件对因果关系标注数据集

分布式应用	正例	负例	总计
train-ticket	121	3610827	3610948
sock-shop	165	4639463	4639628
总计	286	8250290	8250576

2.5.2 评测指标

由于事件因果关系判别模型目的在于正确判别事件对之间的关系，所以本块实验以被标注的事件对关系被识别的精确率 (*precision*)、召回率 (*recall*) 和 *F1* 值作为评测标准。计算公式分别如式??、??和??所示。

$$precision = \frac{|R_{correct}|}{|R_{predict}|} \quad (2.7)$$

$$recall = \frac{|R_{correct}|}{|R_{label}|} \quad (2.8)$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \quad (2.9)$$

其中， $|R_{correct}|$ 为同时被标注存在因果关系且被判别为存在因果关系的事件对数， $|R_{predict}|$ 为模型判别为有因果关系的事件对数， $|R_{label}|$ 为被标注存在因果关系的事件对数。

2.5.3 实验方法

表 2-10 对比特征集

特征名	计算方式
共现次数 ^{liu1998integrating}	A, B 共现次数
关系置信度 ^{liu1998integrating}	$P(B A)$
逆关系置信度 ^{liu1998integrating}	$P(A B)$
Lift ^{liu1998integrating}	$P(AB)/((P(A) \cdot P(B)))$
KULC ^{liu1998integrating}	$(P(A B) + P(B A))/2$
IR ^{liu1998integrating}	$P(A)/(B)$
Pearson ^{mahimkar2008troubleshooting}	皮尔逊相关系数
Location ^{nie2016mining-causality-graph}	事件是否发生在同一个设备

对于标注数据中的正样本 286 条、负样本 2860 条，随机抽取平分为 5 份，以便采用 5 折交叉验证方法。其中每份数据集中包含正样本数 57，负样本数 572。本块实验选用了 SVM、XGBoost、随机森林、逻辑回归和多层感知机共计五种经典模型进行了对比实验，以选出最优表现的分类模型。

另外，为了验证本文选用的 6 种事件特征对因果关系判别的提升（其中只有“关系置信度”为基于概率的特征），实验总结了已有工作中使用的特征进行对比（其中前 6 项均是基于

概率的特征), 如表??所示。随后, 用“multi”表示本文使用的事件特征集, “base”表示对比特征集。

2.5.4 实验结果与分析

实验统计了各个模型在进行 5 折交叉验证时, 所计算得到的精确率、召回率和 F1 值, 结果统计如表??所示。对于每个模型, 都尝试了多种超参数组合, 最终通过网格搜索选择可使其达到最佳效果的超参数组合。

表 2-11 各分类模型事件因果关系判别结果

模型	选用数据特征	精确率	召回率	F1
SVM	multi	0.951	0.961	0.956
XGBoost	multi	0.911	0.895	0.903
随机森林	multi	0.950	0.792	0.864
逻辑回归	multi	0.913	0.845	0.878
多层感知机	multi	0.892	1.0	0.943
SVM	base	0.822	0.769	0.795
XGBoost	base	0.779	0.756	0.767
随机森林	base	0.831	0.770	0.799
逻辑回归	base	0.782	0.754	0.768
多层感知机	base	0.739	0.816	0.776

在表??统计的结果中, SVM 模型的精确率、召回率和 F1 值均达到了 0.95 以上, 而随机森林、逻辑回归和 XGBoost 效果较差。多层感知机召回率达到了 1.0, 但精确率只有 0.892, 可见其偏向于将数据都判别为正例, 判别结果会出现过多的假阳。在本实验中, SVM 模型取得最佳效果时所设置的超参数组合为高斯核函数、 $sigma = 1$ 、松弛变量 =5。另外, 对比选用的数据特征, 发现使用“multi”比只使用“base”的特征在所有模型上都取得了更好的效果, 证明了本文设计并引入新的事件特征对因果关系判别效果有显著提升。

2.6 本章小结

本章主要介绍了组件-事件知识图谱的构建方式, 该图谱整合了所有种类的数据。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。在实验部分, 实验结果验证了本文新设计引入的事件特征, 对事件因果关系判别的有效性。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

在已有的知识库中（如 FB15K^{bordes2013translatingE} 和 YAGO3^{guo2018knowledge}），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??所示。

虽然文献feng2016gake, shi2017knowledge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，即不能根据具体上下文给同一实体不同的嵌入表示。

为了满足实体随上下文动态表示，本章设计了针对组件-事件知识图谱的知识表示学习方案（Dynamic Knowledge Graph Embedding, DKGE）。本章共分为 4 节展开介绍：首先，本章介绍了所提表示学习方案的总体框架；随后，本章进行了场景分析，罗列了组件-事件知识图谱进行表示学习时要解决的问题；然后详细介绍了针对组件-事件知识图谱的表示学习模型；最后，本章设计了详细的实验验证本文模型的有效性。

3.1 总体框架

本章提出的针对组件-事件知识图谱的表示学习方案的总体框架图如??所示。由图可见，输入为三元组及其对应的上下文图结构。实体表示被分为了语义表示和结构表示两部分，语义表示通过实体文本属性训练的预训练模型获取，结构表示通过引入注意力机制的关系图卷积网络获取。关系表示则只有语义表示。整个表示学习模型，通过训练模块计算损失函数并更新模型参数，通过预测模块检测其效果。

3.2 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。

3.2.1 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??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）

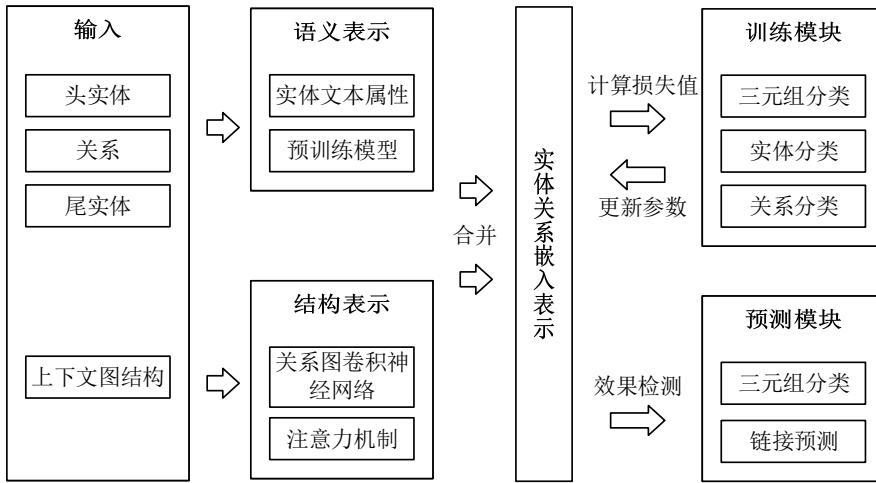


图 3-1 组件-事件知识图谱动态表示学习总体框架图

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??所示。

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

(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??中所示的条件。

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

(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的过程，如式??所示。

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

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。

3.2.2 动态嵌入表示

根据??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feng2016gake, shi2017knowledge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??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, HttpServiceErrorException500*) 三元组，但该图中 *ts - travel - service* 发生 *HttpServiceErrorException500* 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??中 *ts - travel - service* 发生 *HttpServiceErrorException500* 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 *HttpServiceErrorException500* 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。

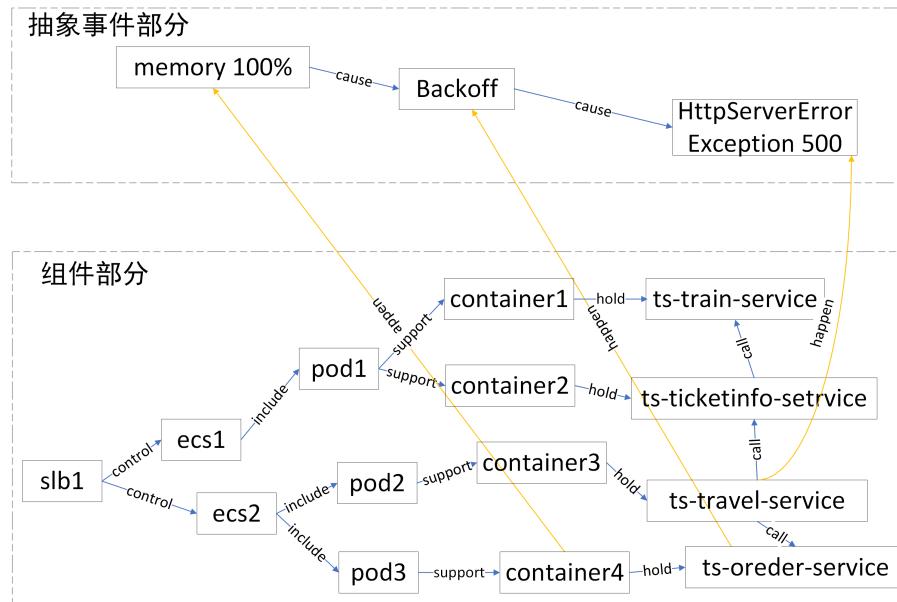


图 3-2 订票服务不可用故障对应知识图谱部分示例

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^{mikolov2013efficient} 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^{peters2018deep} 提出使用双层记忆网络，Bert^{devlin2018bert} 使用基于自注意力机制的 Trans-

former^{vaswani2017attention} 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，本文可以将知识图谱看作稀疏有向图，使用关系图卷积神经网络作为编码器，获取实体的上下文图结构信息。

3.3 组件-事件知识图谱动态表示学习模型

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。由于 RGCN 可以充分提取图的拓扑结构信息，所以本文使用 RGCN 作为编码器，获取实体的上下文图结构信息。在 RGCN 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，即 Attention-RGCN。

本节详细介绍了针对组件-事件知识图谱的表示学习方法，分为模型结构、模型训练和预测 3 小节。

3.3.1 模型结构

整个模型结构如图??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。

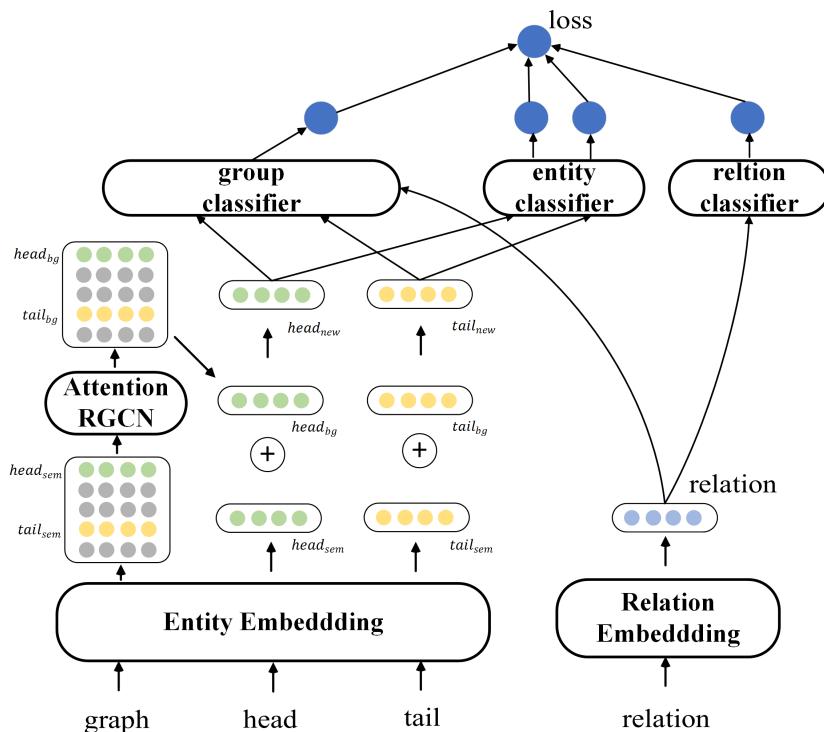


图 3-3 组件-事件知识图谱表示学习模型

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 *relaion* 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，使用实体的属性信息对 Bert^{devlin2018bert} 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??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??中 *cpu*100% 和 *memory*100% 导致了 *Backoff* 的发生，但 *memory*100% 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??所示。

$$h_i^{(l+1)} = \sigma \left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \alpha_{(j,r,i)} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \quad (3.4)$$

本式整体与式??一致，各个符号含义此处不进行赘述。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：

$$v_{(j,r,i)} = \mathbf{W}_a^T (\mathbf{W}_e h_j + \mathbf{W}_r h_r + \mathbf{W}_e h_i) \quad (3.5)$$

$$\alpha_{(j,r,i)} = \text{softmax}(v_{(j,r,i)}) = \frac{\exp(v_{(j,r,i)})}{\sum_{j' \in \mathcal{N}_i^r} \exp(v_{(j',r,i)})} \quad (3.6)$$

式??中 $h_i, h_j, h_r \in \mathbb{R}^d$ 分别对应索引为 i, j 两实体和关系 r 的嵌入向量， $\mathbf{W}_a \in \mathbb{R}^d$, $\mathbf{W}_e, \mathbf{W}_r \in \mathbb{R}^{d \times d}$ 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??使用 *softmax(.)* 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 下关联的实体索引值集合。经过 Attention-RGCN 层后可以得到对应索引 i 的实体 *head*、对应索引 j 的实体 *tail* 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 *contanier*、*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 $d \times h \times c_n$ 的多层次感知机和 *softmax(.)* 层，即可得到实体对各类分类情况。损失函数使用交叉熵，如式??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。

$$\mathcal{L}_{entity} = - \sum_{k=1}^K y_{ik} \ln h_{ik} - \sum_{k=1}^K y_{jk} \ln h_{jk} \quad (3.7)$$

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，使用多层次感知机、*softmax(.)* 层分类，然后损失函数使用交叉熵即可，如式??所示。其中， h_r 为关系 *relation* 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。

$$\mathcal{L}_{relation} = - \sum_{k=1}^K y_{rk} \ln h_{rk} \quad (3.8)$$

三元组分类器用于判别三元组是否存在，可以使用文献trouillon2016complex中形如式??的评分函数。其中 *diag(.)* 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??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， g 为三元组 (i, r, j) 所在的上下文图结构。

$$f(h_j, h_r, h_i) = h_j^T \text{diag}(h_r) h_i \quad (3.9)$$

$$\mathcal{L}_{group} = -\frac{1}{|\mathcal{T}|} \sum_{(g, i, r, j) \in \mathcal{T}} y \log l(f(h_i, h_r, h_j)) + (1 - y) \log(1 - l(f(h_i, h_r, h_j))) \quad (3.10)$$

3.3.2 模型训练

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??所示，各个符号含义已在上文进行了描述：

$$\begin{aligned} \mathcal{L} = & -\frac{1}{|\mathcal{T}|} \sum_{(g, i, r, j) \in \mathcal{T}} y \log l(f(h_i, h_r, h_j)) + (1 - y) \log(1 - l(f(h_i, h_r, h_j))) \\ & - \sum_{k=1}^K y_{ik} \ln h_{ik} - \sum_{k=1}^K y_{jk} \ln h_{jk} \\ & - \sum_{k=1}^K y_{rk} \ln h_{rk} \end{aligned} \quad (3.11)$$

根据以上损失函数计算的损失值，就可以利用梯度下降算法更新模型参数，直到模型训练完成。模型训练算法如??所示。

3.3.3 模型预测

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）。本质上，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??所示，其中 $events$ 为候选抽象事件索引集，其它符号含义已在上文进行了描述。

$$e = \arg \max_{j \in events} (f(h_i, h_r, h_j)) \quad (3.12)$$

3.4 实验与分析

在本节中，对本章提出的组件-事件知识图谱表示学习模型进行了实验与分析。

3.4.1 数据集构建

如实际运维场景一样，本文将模拟收集的数据划分成了历史数据和实时数据。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，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，如表??所示。

根据章节??事件因果关系判别实验结果，本处选用 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??所示步骤沉淀生成每类故障的组件-事件知识图谱。

算法 3.1 组件-事件知识图谱动态表示学习模型的训练算法

输入: 四元组 $(graph, head, relation, tail)$ 集合 D , 学习率 α , 训练代数 $epoch$

输出: 更新后的模型参数 θ

```

1: 随机初始化模型参数  $\theta$ ,  $e \leftarrow 0$ 
2: while  $e < epoch$  do
3:   for all  $(graph, head, relation, tail) \in D$  do
4:     if  $(head, relation, tail) \in graph$  then
5:       将  $(head, relation, tail)$  从  $graph$  中去除
6:     end if
7:      $graph_{sem}, head_{sem}, tail_{sem}, relation_{sem} \leftarrow Embedding(graph, head, relation, tail)$ 
8:      $head_{bg} \leftarrow 0, tail_{bg} \leftarrow 0$ 
9:     if  $head, tail \in graph$  then
10:     $head_{bg}, tail_{bg} \leftarrow Attention-RGCN(graph_{sem})$ 
11:  end if
12:   $head_{new} \leftarrow head_{bg} + head_{sem}, tail_{new} \leftarrow tail_{bg} + tail_{sem}$ 
13:  如式??计算损失值  $\mathcal{L}_\theta(graph, head, relation, tail)$ 
14:   $\theta \leftarrow \theta - \alpha \nabla \mathcal{L}_\theta(graph, head, relation, tail)$ 
15: end for
16: end while
17: return  $\theta$ 

```

表 3-1 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

由模型初步生成的知识图谱虽然满足了实际使用需求，但不能保证没有任何瑕疵。因此，后续运维专家参与了知识图谱调优，保证了知识图谱绝对无瑕。在调优过程中，因为本文方法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??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。

表 3-2 知识库三元组数目及划分

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

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被重复使用，防止数据不均衡。表??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。

3.4.2 评测指标

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**bordes2011learning, bordes2013translatingE**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**bordes2013translatingE**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**bordes2013translatingE**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**bordes2013translatingE, wang2014knowledge, lin2015learning** 中展开过，也被广泛的应用

表 3-3 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

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。

3.4.3 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：

[itemsep=0 pt,topsep = 0 pt,parsep =0pt,partopsep=0pt]**TransE^{bordes2013translatingE}**: TransE 将每个三元组 (head,relation,tail) 中的 relation 视作从 head 到 tail 的翻译过程。**TransH^{wang2014knowledge}**: 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。**TransR^{lin2015learning}**: TransR 将关系也嵌入为矩阵，可以区分具有不同语义的关系。**GAKE^{feng2016gake}**: 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。**TCE^{shi2017knowledge}**: 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。**DKGE**: 本文在章节??所提出的组件-事件知识图谱表示学习模型。**DKGE-1**: 相比本文表示学习方法，去除了 Attention-RGCN 层的注意力机制。**DKGE-2**: 相比本文表示学习方法，去除实体分类模块的损失。**DKGE-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结构信息对表示学习的提升；GAKE、TCE 引入上下文结构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；DKGE-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；DKGE-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；DKGE-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。

3.4.4 实验结果与分析

在链接预测任务上，表??列出了各个方法的链路预测结果。DKGE 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??列出来各个方法的精确率（precision）、召回率（recall）和 F1 值。DKGE 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 DKGE-1、DKGE-2、DKGE-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重要性更高。

另外，DKGE 在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 ω 个“不成立”的受损三元组（本实验中 ω 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数为：嵌入层维度为 100；Attention-RGCN 为 4 层、输入窗口大小为 128 节点，中间隐状态维度为 50、输出隐状态维度为 100；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 3-4 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
DKGE	9	85.4	13	82.9
DKGE-1	26	79.5	34	67.6
DKGE-2	19	82.2	30	78.4
DKGE-3	16	83.6	24	81.2

表 3-5 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
DKGE	0.864	0.837	0.850	0.836	0.814	0.825
DKGE-1	0.763	0.816	0.788	0.776	0.746	0.761
DKGE-2	0.826	0.801	0.813	0.783	0.808	0.795
DKGE-3	0.845	0.822	0.834	0.788	0.810	0.799

3.5 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 Attention-RGCN 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。同时也介绍了优化目标和预测事件的公式。实验结果验证了本章表示学习模型的有效性。

第四章 基于知识图谱的故障预测

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。根据这些实时事件序列，准确地预测未来会发生的故障始终面临着巨大挑战。已有的方法^{pitakrat2018hora, zhang2018prefix, baldoni2015line, xu2016health, cheng2018machine, du2017deeplog, das2018desh, islam2017predicting, cheng2018mac}均集中在基于时序数据的故障预测方法，只能预测到是否会发生故障，缺乏可解释性，详细分析如??所述。

为了解决故障难以准确预知的问题，本章提出了引入知识图谱的故障预测方案。本文的故障预测模型可以预测到会产生何种故障。另外，知识图谱增强了预测结果的可解释性，有助于运维人员分析、调整系统，防止故障真正发生带来损失。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。

本章节共分为 4 小节介绍基于知识图谱的故障预测模型（Fault Prediction with Knowledge Graph, FPKG）。首先，本章介绍了所提故障预测模型的总体框架图。随后，本章进行了场景分析，列出了本文场景中故障预测要解决的问题。然后详细介绍了引入组件-事件知识图谱的故障预测模型。最后，本章在数据集上进行了充分的实验，并对结果进行了详尽分析。

4.1 总体框架

本章所提故障预测方案的总体框架图如??所示。由图可见，故障知识库中存储着多个故障类型对应的组件-事件知识图谱，这些知识图谱均是历史数据经过上文所述图谱构建方法得到的。实时数据（日志和指标时序数据）会经过事件生成模块，转化为实时事件序列。实时事件序列输入故障预测模块后，经 BiLSTM 编码，会与知识库中每个经 Attention-RGCN 编码的组件-事件知识图谱计算匹配度。训练模块会最大化实时事件序列与对应组件-事件知识图谱的匹配度，并更新模型参数。预测模块会输出匹配度最高的知识图谱作为预测结果，用于检测故障预测模型效果，同时会根据匹配到的知识图谱与实时事件，标出实时事件间的触发关系，解释故障触发过程。

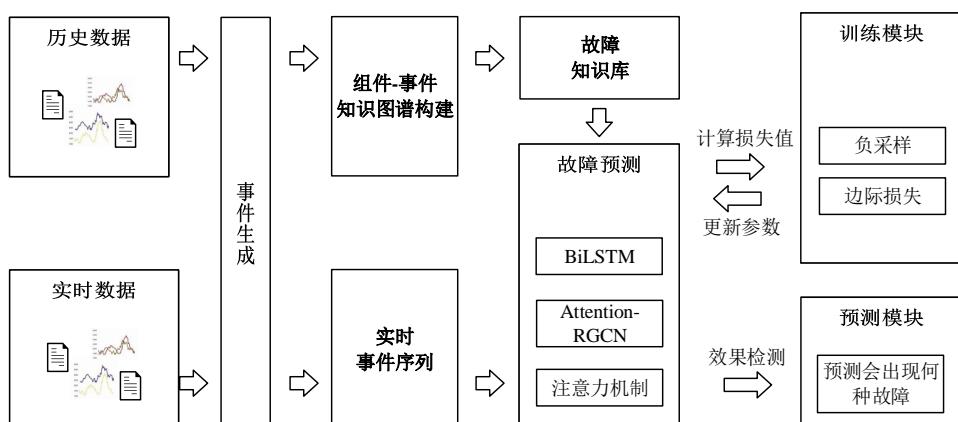


图 4-1 基于事件序列和知识图谱的故障预测总体架构图

4.2 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pitakrat2018hora, zhang2018prefix, baldoni2015line使用了统计学习和机器学习方法，如隐半马尔可夫模型（Hidden Semi-Markov Model, HSMM）、支持向量机（SVM）。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xu2016health, cheng2018machine, du2017deeplog, das2018desh, islam2017predicting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cheng2018machine, du2017deeplog, das2018desh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gao2020task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，除了要分析时序数据外，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。

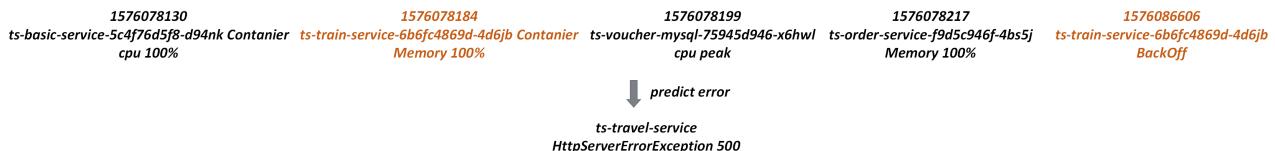


图 4-2 事件序列故障预测

(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??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??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gao2020task，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自适应注意力权重，即由组件-事件知识图谱选取其所关注的信息。

4.3 基于事件序列和知识图谱的故障预测模型

本小节主要介绍基于时序数据和组件-事件知识图谱的故障预测模型。在章节??已经介绍了编码时序数据的 BiLSTM 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

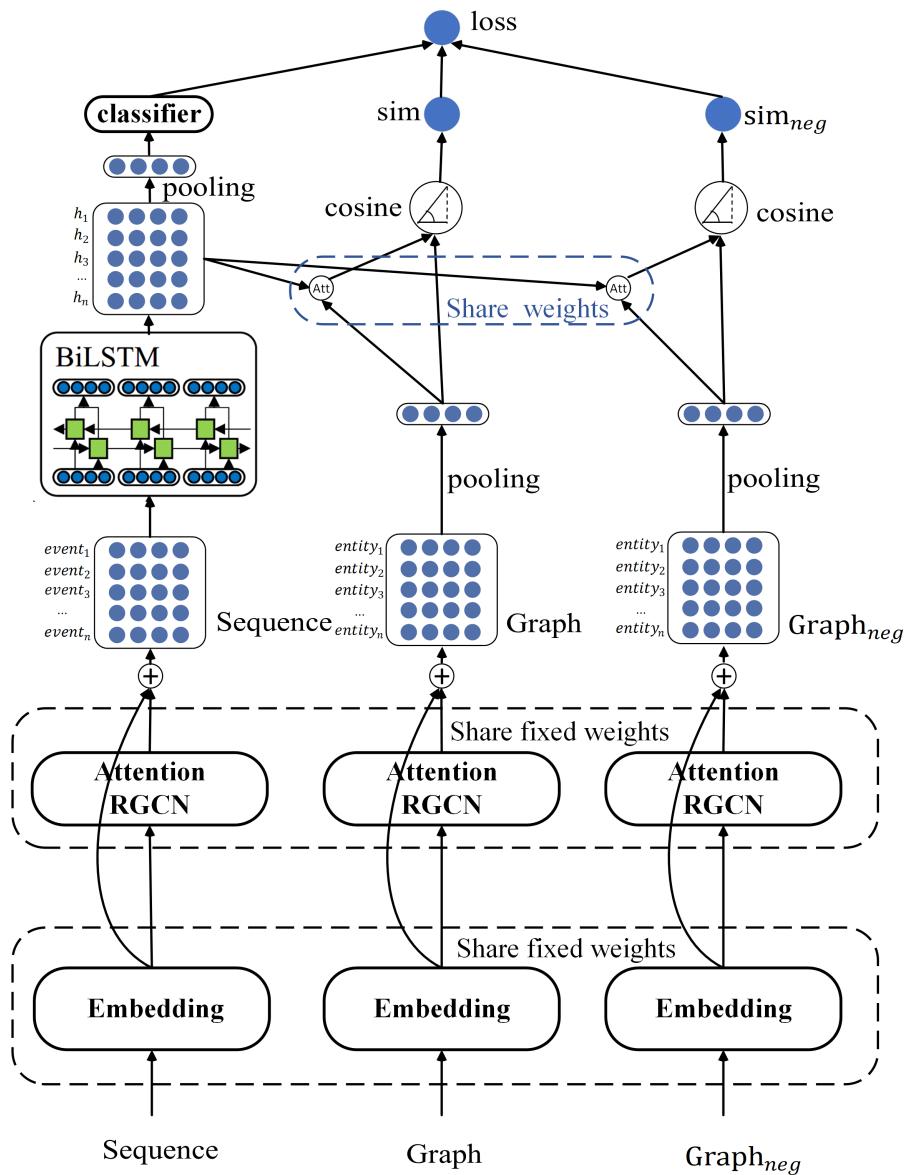


图 4-3 故障预测模型

图??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。BiLSTM 层、注意力层和分类层的参数会在后续进行学习。

嵌入层规定了实时事件序列和知识图谱的嵌入方式。图??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??中的表示学习模型，获取每个实时事件的嵌入表示向量。输入的组件-事件知识图谱则同图??所示。输入层中的 embedding 和 Attention-RGCN 均使用章节??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。

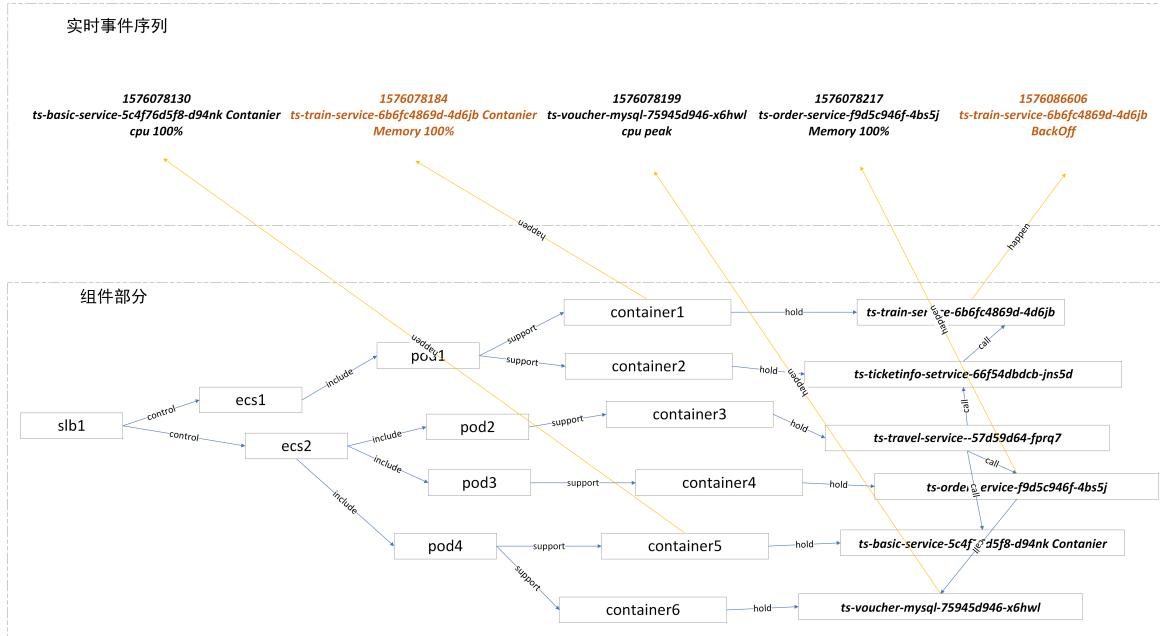


图 4-4 输入实时事件序列样例

BiLSTM 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。

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 BiLSTM 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 BiLSTM 网络后会得到隐向量序列 $\{h_1, h_2 \dots h_m\}$ 。

$$\begin{aligned} e_{(h_i, g)} &= h_i \mathbf{W} g \\ \alpha_{(h_i, g)} &= \text{softmax}(e_{(h_i, g)}) = \frac{\exp(e_{(h_i, g)})}{\sum_{j=0}^m \exp(e_{(h_j, g)})} \\ h_{sequence} &= \sum_{i=0}^m \alpha_{(h_i, g)} \cdot h_i \end{aligned} \quad (4.1)$$

如式??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 $h_{sequence}$ 。其中， h_i 为事件 x_i 的嵌入表示， g 为组件-事件知识图谱经过嵌入表示层后的表示向量， $\mathbf{W} \in \mathbb{R}^{d \times d}$ 为参数矩阵。 m 为事件序列长度， $e_{(h_i, g)}$ 为 g 对 h_i 的注意力值。

输出层分为两方面，一方面会将 BiLSTM 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 模型训练

模型的损失函数如式??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。

$$\begin{aligned} \mathcal{L} = & -\sum_{i=1}^n [Y_i \log(f(X_i)) + (1 - Y_i) \log(1 - f(X_i))] \\ & + Y_i \cdot \max(0, b + \cos(h_i, g_{neg}) - \cos(h_i, g))] \end{aligned} \quad (4.2)$$

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 事件序列经过如式??注意力机制计算得到的隐状态嵌入表示， 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。

根据以上损失函数计算的损失值，就可以利用梯度下降算法更新模型参数，直到模型训练完成。模型训练算法如??所示。

算法 4.1 基于事件序列和知识图谱的故障预测模型的训练算法

输入：带有无故障标签 Y 和具体故障类型标签的事件序列集合 X ，学习率 α ，训练代数 $epoch$

输出：更新后的模型参数 θ

```

1: 随机初始化模型参数  $\theta$ ,  $i \leftarrow 0$ 
2: while  $i < epoch$  do
3:   for all  $X_i \in X$  do
4:     if  $Y_i == 1$  then
5:        $g \leftarrow$  Attention-RGCN( $X_i$  对应故障类型的组件-事件知识图谱)
6:        $g_{neg} \leftarrow$  Attention-RGCN(知识库中随机一个不与  $X_i$  匹配的知识图谱)
7:     end if
8:     如式??计算损失值  $\mathcal{L}_\theta(X_i)$ 
9:      $\theta \leftarrow \theta - \alpha \nabla \mathcal{L}_\theta(X_i)$ 
10:   end for
11: end while
12: return  $\theta$ 

```

4.3.3 模型预测

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 $model(.)$ 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??所示。

$$failure = \arg \max_{f \in failures} (model(X, g_f)) \quad (4.3)$$

4.4 实验与分析

4.4.1 数据集构建

在构建组件-事件知识图谱时，表??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 BiLSTM 的输入长度需要是确定的。所以，若 BiLSTM 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??所示。

表 4-1 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

4.4.2 评测指标

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。

4.4.3 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：

[itemsep=0 pt,topsep = 0 pt,parsep =0pt,partopsep=0pt]**PREPARE^{tan2012prepare}**: 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。**RNN^{xu2016health}**: 使用循环神经网络编码事件序列，再多分类到故障类型。**LSTM^{cheng2018machine, du2017deeplog, das2018desh, islam2017predicting, li2020predicting}**: 使用长短期记忆网络编码事件序列，再多分类到故障类型。**Bi-LSTM^{gao2020task}**: 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。**FPKG**: 本文在章节??所提出的基于时序数据和知识图谱的故障预测模型。**FPKG-1**: 相比本文方法，去除了知识图谱对事件序列的 attention 机制。**FPKG-2**: 相比本文方法，使用 RNN 替换了 BiLSTM。**FPKG-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 都后接了多层次感知机和 $softmax(.)$ 层进行多分类。FPKG-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；FPKG-2 用于验证在编码事件序列时 BiLSTM 比 RNN 效果更好；FPKG-3 用于验证在编码事件序列时 BiLSTM 比 LSTM 效果更好。

4.4.4 实验结果与分析

表 4-2 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
FPKG	0.939	0.904	0.918	0.887	0.848	0.861
FPKG-1	0.904	0.892	0.896	0.842	0.831	0.832
FPKG-2	0.841	0.815	0.823	0.787	0.776	0.782
FPKG-3	0.879	0.887	0.883	0.825	0.813	0.814

实验记录了各个方法的精确率、召回率和 F1 值，结果如表??所示。可见 FPKG 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。FPKG-1 不如 FPKG 的效果，证明了注意力机制提升了故障预测效果。FPKG-3 比 FPKG 效果差，说明了编码事件序列特征方面，LSTM 不如 BiLSTM。FPKG-2 比 FPKG-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 层设置为章节??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；损失函数的偏置超参 b 为 0.7。

4.5 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 BiLSTM 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。另外，在数据集上的实验结果，证明了本章引入知识图谱的故障预测模型具有优越性。

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??所示。在该集群上部署了文献zhou2018poster, rahman2019predicting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??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??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??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。

```
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
```

(5.1)

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??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??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^{boettiger2015introduction} 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??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

5.2.1 数据标注

本实验对数据集中的关系种类进行了标注，标注数据覆盖了每一种故障类型。其中对于每种故障类型，随机抽取了该类故障类型两个具体时间段的数据进行了标注，只对认为有因果关系的事件对标注为 1，其余事件对默认无因果关系标注为 0。

由于事件因果判别模型是根据事件对统计特征发掘事件间因果关系，与事件来自何种应用无关，所以本块实验不区分事件来自于 train-ticket 还是 sock-shop，直接把两个应用来源的标注事件对整合起来。最终标注为 1 的事件对共计 286 条，其余默认无关系的共计 8,250,290 条。数据集 train-ticket、sock-shop 的标注信息如下表??所示。由于正负样本数量级差距较大，后续选择了按照正负样本比 1:10 随机抽取了一批负样本数据用于训练。抽取出用于训练的数据共包含正样本 286 条，负样本 2860 条。

表 5-5 事件对因果关系标注数据集

分布式应用	正例	负例	总计
train-ticket	121	3610827	3610948
sock-shop	165	4639463	4639628
总计	286	8250290	8250576

5.2.2 评测标准

由于事件因果关系判别模型目的在于正确判别事件对之间的关系，所以本块实验以被标注的事件对关系被识别的精确率 (*precision*)、召回率 (*recall*) 和 *F1* 值作为评测标准。计算

公式如式??所示。

$$\begin{aligned}precision &= \frac{|R_{correct}|}{|R_{predict}|} \\recall &= \frac{|R_{correct}|}{|R_{label}|} \\F1 &= \frac{2 \times precision \times recall}{precision + recall}\end{aligned}\quad (5.4)$$

其中 $precision$ 、 $recall$ 、 $F1$ 分别代表精确率、召回率和 F1 值。 $|R_{correct}|$ 为同时被标注存在因果关系且被判别为存在因果关系的事件对数， $|R_{predict}|$ 为模型判别为有因果关系的事件对数， $|R_{label}|$ 为被标注存在因果关系的事件对数。

5.2.3 实验方法

对于标注数据中的正样本 286 条、负样本 2860 条，随机抽取平分为 5 份，以便采用 5 折交叉验证方法。其中每份数据集中包含正样本数 57，负样本数 572。

本块实验根据文献**nie2016mining-causality-graph**, **qiu2020causality-mining-knowledge-graph**中的实验，选用了 SVM、XGBoost、随机森林、逻辑回归和多层感知机共计五种模型进行了对比实验，以选出最优表现的分类模型。

另外，为了验证本文设计的 6 种事件特征对因果关系判别的提升（其中只有“关系置信度”为基于条件概率的特征），实验选用文献**nie2016mining-causality-graph**中总结的共计 6 条基于条件概率的特征进行对比，如表??所示。“multi”表示使用了本文提出的 6 种特征，“probability based”表示只使用了基于概率的特征集。

表 5-6 基于条件概率的特征集

特征名	计算方式
共现次数	A, B 共现次数
关系置信度	$P(B A)$
逆关系置信度	$P(A B)$
Lift	$P(AB)/((P(A) \cdot P(B)))$
KULC	$(P(A B) + P(B A))/2$
IR	$P(A)/(B)$

5.2.4 实验结果与分析

实验统计了各个模型在进行 5 折交叉验证时，所计算得到的精确率、召回率和 F1 值，结果统计如表??所示。对于每个模型，都尝试了多种超参数组合，最终通过网格搜索选择可使其达到最佳效果的超参数组合。

在表??统计的结果中，SVM 模型的精确率、召回率和 F1 值均达到了 0.95 以上，而随机森林、逻辑回归和 XGBoost 效果较差。多层感知机召回率达到了 1.0，但精确率只有 0.892，可见其偏向于将数据都判别为正例，判别结果会出现过多的假阳。在本实验中，SVM 模型取得最佳效果时所设置的超参数组合为高斯核函数、 $sigma = 1$ 、松弛变量 =5。

另外，对比选用的数据特征，发现使用“multi”比只使用“probability based”的特征在所有模型上都取得了更好的效果，证明了本文所设计的事件特征对因果关系判别效果有显著

提升。

表 5-7 各分类模型事件因果关系判别结果

模型	选用数据特征	精确率	召回率	F1
SVM	multi	0.951	0.961	0.956
XGBoost	multi	0.911	0.895	0.903
随机森林	multi	0.950	0.792	0.864
逻辑回归	multi	0.913	0.845	0.878
多层感知机	multi	0.892	1.0	0.943
SVM	probability based	0.822	0.769	0.795
XGBoost	probability based	0.779	0.756	0.767
随机森林	probability based	0.831	0.770	0.799
逻辑回归	probability based	0.782	0.754	0.768
多层感知机	probability based	0.739	0.816	0.776

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??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??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 5-8 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 5-9 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 5-10 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??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??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**bordes2011learning, bordes2013translatingE**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**bordes2013translatingE**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**bordes2013translatingE**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**bordes2013translatingE, wang2014knowledge, lin2015learning**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**: 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^{wang2014knowledge}**: 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^{lin2015learning}**: 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^{feng2016gake}**: 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^{shi2017knowledge}**: 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??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??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??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 ω 个“不成立”的受损三元组（本实验中 ω 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 5-11 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	9	85.4	13	82.9
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 5-12 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	0.864	0.837	0.850	0.836	0.814	0.825
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??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??所示。

表 5-13 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^{tan2012prepare}**: 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^{xu2016health}**: 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^{cheng2018machine, du2017deeplog, das2018desh, islam2017predicting, li2020predicting}**: 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^{gao2020task}**: 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??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

表 5-14 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	0.939	0.904	0.918	0.887	0.848	0.861
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

实验记录了各个方法的精确率、召回率和 F1 值，结果如表??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??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。

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 基于知识图谱的 IT 运维辅助系统设计与实现

实时运行的集群中存在着大量的组件与实时数据，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。

本章基于第三章、第四章和第五章提出的各个方法模型，设计并实现了一个基于知识图谱的 IT 运维辅助系统，并对设计与实现过程展开了细致地描述。本章共分 4 小节来介绍系统，介绍的内容包括需求分析、系统总体设计、系统详细实现和系统测试。

6.1 需求分析

本节详细介绍基于知识图谱的 IT 运维辅助系统需要满足的需求，包括功能性需求和非功能性需求。功能性需求是指 IT 运维辅助系统要实现的功能，而非功能性需求是指该系统应当满足的除功能性需求之外的其他需求。

6.1.1 功能性需求

本系统主要目的是辅助 IT 运维人员，降低实际运维的难度。因此，在系统实现前，本文咨询了 IT 运维人员，进行了详细的需求调研，总结了 IT 运维辅助系统的三大功能需求：实时集群状态查询、组件-事件知识图谱查询及调优、实时故障预测。图??为本章 IT 运维辅助系统用例图。

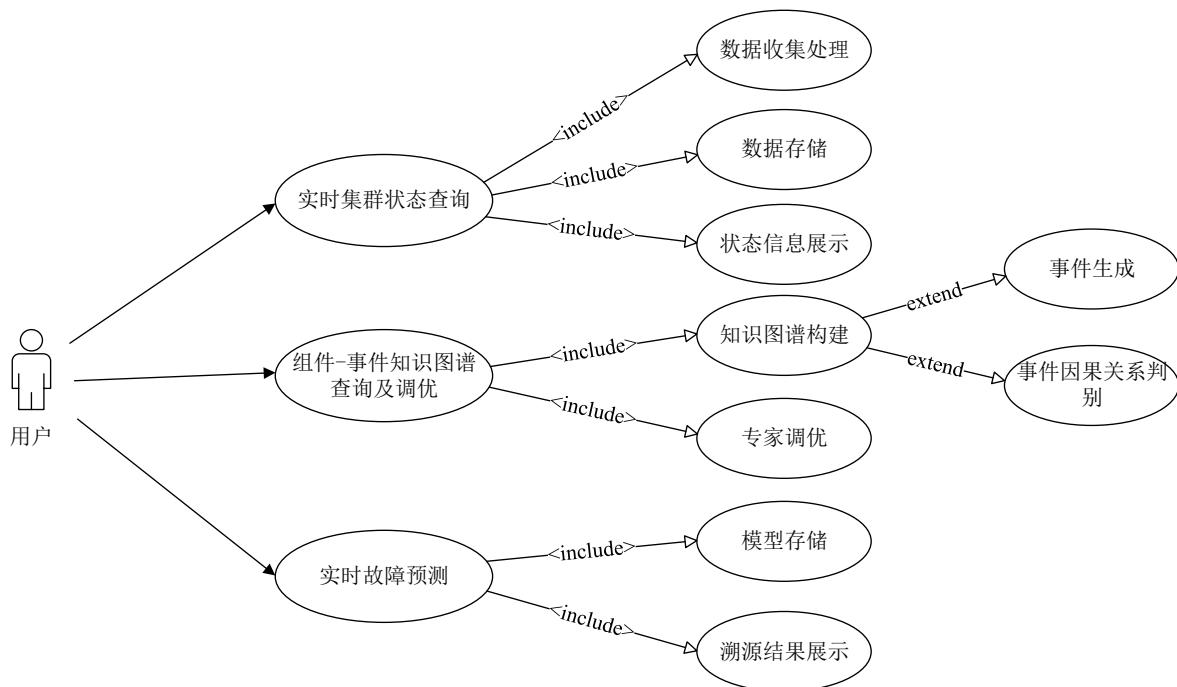


图 6-1 IT 运维辅助系统用例图

(1) 实时集群状态查询

分布式集群往往包含着上千个组件，每个组件有其特异的属性信息。另外，每个组件会实时产生不同种类的信息：硬件组件会产生内存、读写、CPU 等时序数据，软件组件会产生文本类的日志数据。在分布式集群实际运转时，每秒钟会产生万级的多源异构数据。

IT 运维人员在实际工作中，需要调用不同接口，使用不同的开源软件才能全方面的了解每个组件的状态。繁琐的集群状态查询过程，不仅消耗了 IT 运维人员大量的精力，还会导致运维人员错失最佳的故障解决时机。因此，IT 运维人员需要一个能够全面便捷地获取系统状态信息的查询系统。这个集群状态查询系统需要整合多个接口，收集各种状态数据，并将这些数据存储起来。另外，该系统需要提供简洁的查询界面，方便 IT 运维人员直观地操作。

(2) 组件-事件知识图谱查询及调优

组件-事件知识图谱直观地展示了故障触发过程，包含着丰富的运维知识。运维人员在实际工作时，需要查询这些知识图谱，获取运维知识，指导下一步运维工作。系统需要根据第三章的组件-事件知识图谱构建流程，在后台从历史数据沉淀出组件-事件知识图谱。在知识图谱构建时，需要将多源异构数据转为统一规范的事件类型数据，并调用分类器获取事件间因果关系。这种依靠机器学习构建组件-事件知识图谱的方式，虽然能够快速沉淀出可用的故障触发知识，但始终不能做到完全精准，需要运维人员介入，进行少量的人工调优。基于以上分析，本系统需要构建并展示组件-事件知识图谱，方便 IT 运维人员查询，也方便其对知识图谱进行验证与调优。

(3) 实时故障预测

在实际运维工作中，运维人员很难预知是否会有故障发生，往往是在故障发生后采取修复工作。这样解决已发生故障的工作模式，很难避免故障带来的损失。因此，本系统需要提供实时故障预测的功能，能够结合组件-事件知识图谱和实时的运行状态数据，给出准确具体的故障预测结果。为了实现实时故障预测，本系统需要存储多个模型，并给出故障预测结果的可视化展示。

6.1.2 非功能性需求

IT 运维辅助系统除了需要满足上节所提的功能，还要满足以下非功能需求：

- (1) 各个查询接口返回结果要迅速，需要在 2s 内给出查询结果。
- (2) 故障预测过程要迅速，系统需要根据实时产生的数据，在 5s 的响应时间内给出正确预测结果。

6.2 系统总体设计

本节进行了系统的总体设计，首先给出了系统的总体框架图，随后介绍了各个功能模块的设计。

6.2.1 系统总体框架图

本系统使用 B/S (Browser/Server) 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。B/S 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??展示了这三层结构。

(1) 数据层。本系统的数据层主要用于存储各种数据，使用了三种存储数据库，用于存储不同类型的数据。Kafka 用于缓存大量的微服务实时访问请求，这部分数据会被用于生成微服务间拓扑关系；Neo4j 用于存储生成的组件-事件知识图谱；阿里云 SLS 用于存储日志数据和指标时序数据。

(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。该层由多个模块协同工作实现系统功能，分为状态监测模块、组件-事件知识图谱构建模块和故障预测模块。每个模块的具体功能在下面几小节展开介绍。

(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。本系统交互层主要分为三部分：组件-事件知识图谱查询及调优页面、实时集群状态查询页面和故障预测页面。

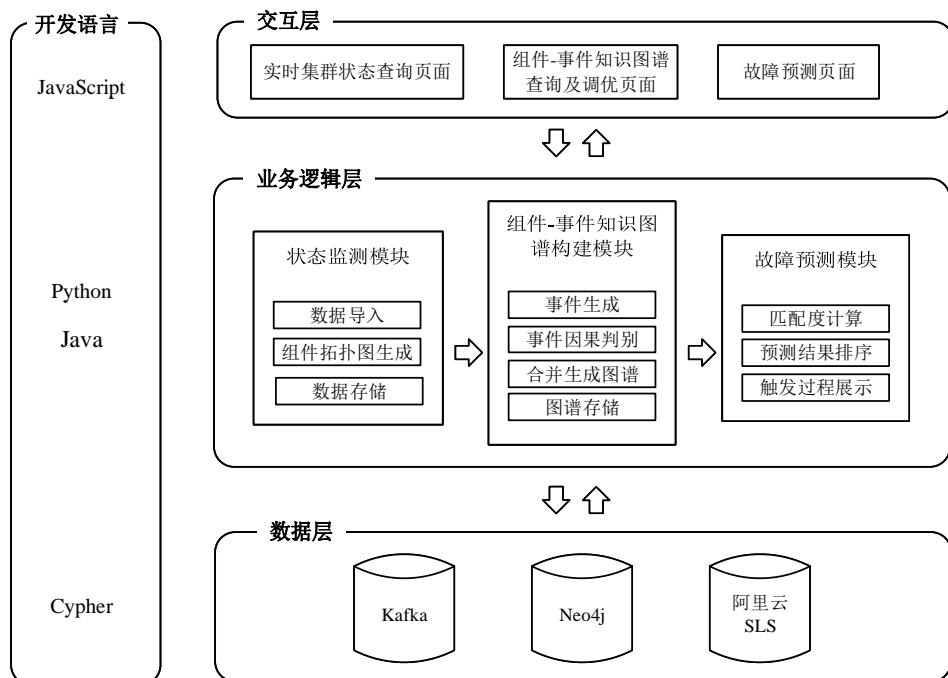


图 6-2 IT 运维辅助系统框架图

6.2.2 状态监测模块概要设计

本模块主要负责将不同来源、不同种类的数据整合起来，减少 IT 运维人员查询数据的难度。该模块需要设计并实现数据导入、组件拓扑图生成和数据存储三部分，分别如下：

- (1) 数据导入部分要使用不同的 API，不同的中间件，获取不同种类的数据，包括日志数据、指标时序数据、组件属性数据、微服务请求数据；
- (2) 组件拓扑图生成部分需要分析组件属性、微服务请求数据，生成组件间拓扑图；
- (3) 数据存储部分负责数据持久化，需要整理不同形式的数据，并根据数据特性存入不同的数据库中。

6.2.3 组件-事件知识图谱构建模块概要设计

本模块的主要功能是从历史数据中沉淀生成组件-事件知识图谱。本模块需要设计并实现 4 个子功能：

- (1) 首先，该模块需要整理日志数据、指标时序数据，转化为统一格式的事件类型数据；
- (2) 随后，该模块需要调用机器学习模型，判别事件间是否有因果关系；
- (3) 然后，该模块需要合并生成的事件因果关系，生成抽象事件层图谱，再结合组件拓扑图得到组件-事件知识图谱；
- (4) 最后，该模块要把生成的组件-事件知识图谱存入 Neo4j 中，以保证可以被其他模块调用。

6.2.4 故障预测模块概要设计

本模块的主要功能是实时预测集群是否会发生故障，并给出故障预测结果排名。本模块需要设计并实现 3 个部分，分别为匹配度计算，预测结果排序和触发过程展示，具体如下：

- (1) 在匹配度计算部分，每个故障类型的组件-事件知识图谱，会分别与实时事件序列进行匹配度计算，匹配度越高意味着越有可能发生对应类型的故障。
- (2) 在预测结果排序部分，本模块会根据匹配度降序排列候选故障类型，当所有匹配度均小于阈值时，会认为不会有故障发生。
- (3) 在触发过程展示部分，本模块会结合组件-事件知识图谱和实时事件序列，给出可能会发生的故障的触发过程。

6.3 系统详细实现

本节介绍系统的详细实现过程，分为 3 小节展开介绍，分别为状态监测模块详细实现、组件-事件知识图谱构建模块详细实现和故障预测模块详细实现。

6.3.1 状态监测模块详细实现

状态监测模块从多个源头获取多种类别的数据。本模块通过处理收集到的数据，生成共计 4 类数据，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 4 部分分别介绍这四种数据的实际收集、处理、存储流程。

- (1) 第一部分为集群拓扑结构数据。

首先，在系统开发时，引入阿里云提供的 SDK 核心库和组件的 Maven 项目依赖。其次，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。

随后，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。

最后，按获取的字段信息构建实体属性列表，和组件间关联拓扑。

(2) 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 实时监控微服务间的访问信息，并存储到 Kafka 中。

Jaeger 获取微服务拓扑结构流程如图??所示，其中 span 的形状代表了它对应何种微服务。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 对应着微服务间的一条请求链路。具体步骤如下：

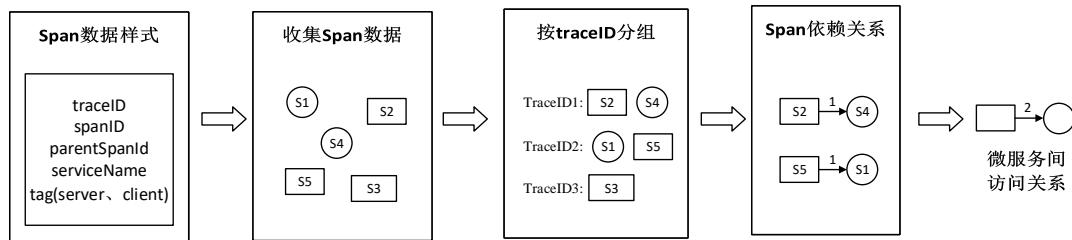


图 6-3 微服务间拓扑图生成流程

首先，获取 Span 数据，从 Kafka 中读取一个时间段内所有来源的 Span。随后，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。然后，分析同组 Span 间依赖关系，即遍历每个 Span 对应的 ParentSpanID，得到 Span 之间的父子依赖关系。其次，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。最后，生成微服务间拓扑图，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。

(3) 第三部分为日志数据的收集。

```

// Data processing logic
1 e_if(v("annotations"), e_output(name="target0", topic="audit-annotation-only"))
2
3 e_if(v("annotations"), e_compose(e_if(v("annotations"), e_output(name="target0", topic="audit-annotation-only"))))
4
5
6
7
8
9
10

```

输出目标	时间	内容
audit-only-annotations	09-11 11:14:56	<code>_source: 10.10.66.160</code> <code>_tag__hostname: audit-logger-controller-78m2d</code> <code>_tag__path: /var/log/kubernetes/kube-apiserver-audit</code> <code>_topic: audit-annotation-only</code> <code>annotations: { authorization: k8s.io/decision: "allow"</code> <code>authorization: k8s.io/reason: "RBAC: allowed by ClusterRoleBinding "flagger-prometheus" of ClusterRole "flagger-prometheus-ingress-nginx"</code>
audit-only-annotations	09-11 11:14:56	<code>_source: 10.10.66.160</code> <code>_tag__hostname: audit-logger-controller-78m2d</code> <code>_tag__path: /var/log/kubernetes/kube-apiserver-audit</code> <code>_topic: audit-annotation-only</code> <code>annotations: { authorization: k8s.io/decision: "allow"</code> <code>authorization: k8s.io/reason: "RBAC: allowed by ClusterRoleBinding "flagger-prometheus" of ClusterRole "flagger-prometheus-ingress-nginx"</code>

图 6-4 数据加工指令示例

首先对日志数据进行采集。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。

然后，对日志数据进行加工。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，编写加工命令选取保留的字段，并传输到新的 Logstore 中即可。如图??所示图，该数据加工指令只保留了 audit 数据中的 annotation 字段。

最后，使用公开 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。

(4) 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：首先，监测获取指标时序数据。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。其次，对指标时序数据进行长期存储。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。最后，使用公开 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。

6.3.2 组件-事件知识图谱构建模块详细实现

本小节介绍构建组件-事件知识图谱的详细实现。在构建知识图谱之前，所有的历史故障数据都已按照故障类型进行了分组。在实际运行时，本模块接收用户的构建请求，根据对应的历史故障数据构建对应的知识图谱，并返回构建结果。

前端向后台发送的请求示例如下所示。请求 URI 路径为 /construct，请求方式为 POST，请求参数共包含两个字段。其中 “faultType” 表明了要构建哪个故障类型的知识图谱（当未设置时，默认构建所有故障类型的知识图谱），“showResult” 表明是否展示构建结果。

```
[xleftmargin=12em,label=code] "faultType": f3, "showResult": True
```

在后台接收到请求后，本模块开始构建指定故障类型的组件-事件知识图谱，整个构建流程如图??所示。针对读取到的数据，本模块根据章节??中所述方法，对指标时序数据使用异常检测算法，对日志数据使用聚类算法，后续使用人工模板匹配转化为事件类型数据。对于生成的事件集合，按照时间先后顺序两两相连，得到候选因果关系集。为了加快对候选因果关系的判别，实际开发中使用了多线程批量处理候选因果关系，直到候选因果关系集为空。每一个线程都调用事件因果关系判别模型，判别出真正存在的事件因果关系。所有被判别为存在的事件因果关系被收集了起来，规约生成抽象事件因果图。

最后，按照章节??中的方法，本模块合并组件拓扑图和抽象事件因果图，生成对应故障类型的组件-事件知识图谱，并存入到 Neo4j 中。

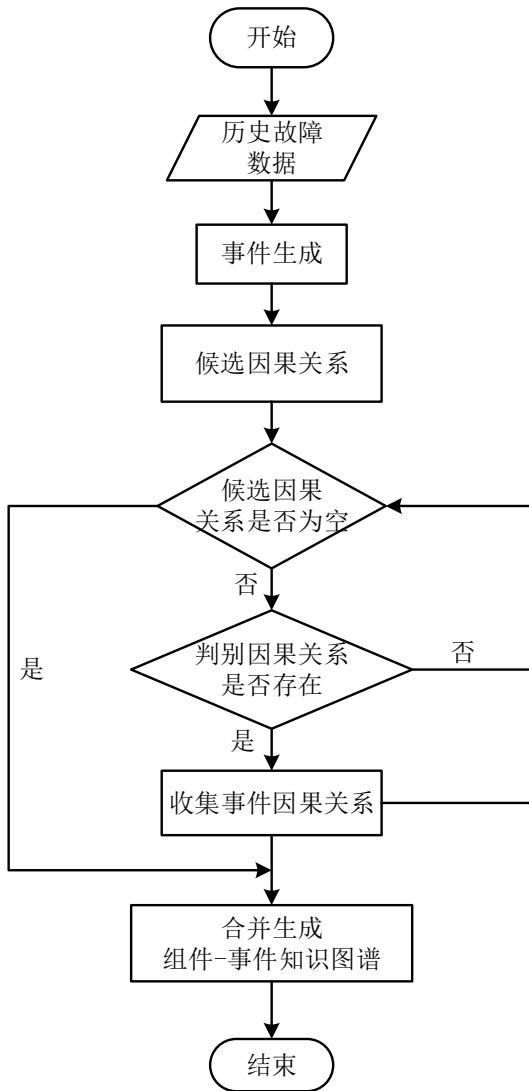


图 6-5 组件-事件知识图谱构建程序流程图

另外，后台会向前端返回响应结果，如下所示。其中，“status”返回请求响应状态，“result”返回了所构建图谱的结点、关系列表。列表中详细记录着每个结点、每条关系的属性信息。`[xleftmargin=8em] "status": 200, "result": {"nodes": [{"typeId": "appLog_INFO_1_tracing...", "eventType": "appLog_INFO_1", ...}], "links": [...]}`

6.3.3 故障预测模块详细实现

本节介绍故障预测模块的详细实现过程。故障预测模块被启动后，其会默认一直运行，不停地监测实时事件序列，并给出故障预测结果。运维人员在不需要故障预测时，将该功能手动关闭即可。

`[xleftmargin=2em] "eventSequence": [{"id": "container_cpu_usage_seconds_total_sudden...", "time": "1576077300", "location": "appLog_INFO_1_tracing...", "eventType": "appLog_INFO_1", ...}], "links": [...]`

每当状态监测模块监测到新事件后，就会向故障预测模块发送一个请求。请求 URI 路径为`/failurePredict`，请求方式为 POST，请求参数中包含着最新的 1024 个事件（少于 1024 时，故障预测模块会补零）。请求示例如上所示，“`eventSequence`”字段记录着事件序列中每个事

件的属性字典。

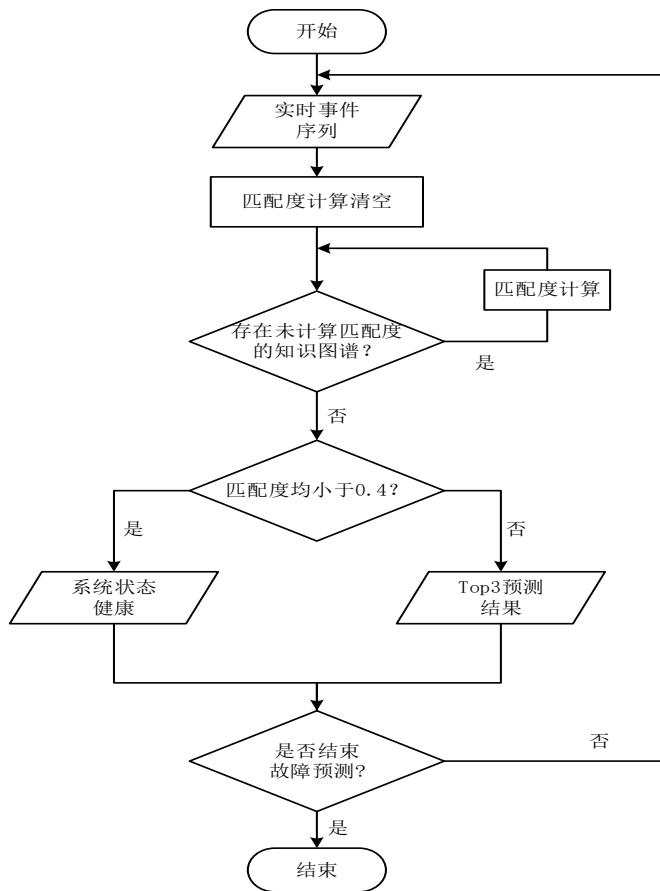


图 6-6 故障预测程序流程图

故障预测模块在接受请求后，会运行如图??所示的程序。首先，实时事件序列被双向长短句记忆神经网络编码为隐向量。随后，本程序遍历知识库中每种故障类型对应的组件-事件知识图谱，与输入的事件序列计算匹配度。在遍历知识库每张知识图谱计算匹配度的过程中，使用了多线程来加快故障预测速度。当匹配度均小于阈值 0.4 时，程序会输出集群状态健康的提示；当存在匹配度大于等于 0.4 的，程序会输出前三的故障预测结果，预测结果为每个匹配的知识图谱对应的故障类型以及其匹配度。本处的阈值 0.4 是运维人员根据实际运维经验调整设定的。

另外，本模块会对比最匹配的组件-事件知识图谱和实时事件序列，当知识图谱中某条因果关系对应的首尾事件同样在实时事件序列中，就认为该关系为可能的触发关系，合并所有的触发关系即可得到故障触发链。

最后，故障预测模块将预测结果返回到前端。预测结果包含在响应参数中，如以下示例所示。在“result”字段中，“failureType”和“value”依次记录着预测的 top3 结果及其匹配度值，“nodes”和“links”记录着可能的故障触发链。每一个 node 都对应着一个具体的事件，所以会有唯一标识、时间戳、位置、事件类型和具体信息 5 种属性。每个 link 对应着一条关系，下例中为 source 到 target 的一条 cause 关系。由于每条属性信息过长，不易于直接展示，所以此处省略了一部分。[xleftmargin=6em] "status": 200, "result": "failureType": ["f3", "f6", "f17",], "value": [0.821, 0.381, 0.217], "nodes": ["id": "container_cpu_usage...", "time": "1576077480", "location": "...

```
"containertrain...","eventtype" : "metriccontainer...", "detail": "1576077480pod...", ...], "links" :  
["source" : "k8seventBackOffwarning...", "target": "containernetwork...", "name": "cause", ...]
```

6.4 系统测试

本章根据章节??中所述的功能性需求和非功能性需求，采用 JUnit 测试框架对系统进行了测试。本节分为 5 个小节，分别为系统测试环境、运行状态查询测试、组件事件知识图谱查询及调优测试、实时故障预测测试和系统性能测试。

6.4.1 系统测试环境

本章在进行测试前，记录测试过程所在的环境，包括集群、服务器和各种开发工具版本号，如表??所示。

表 6-1 系统测试环境

环境项	环境参数
集群	集群类型: Kubernetes 托管版
	版本: 1.12.6-aliyun.1
	云服务器个数: 30
	节点操作系统: CentOS 7.6 64 位
	节点 CPU: 8 核
服务器	节点内存: 16 GiB
	操作系统: Windows 10
	处理器: 11th Gen Intel(R) Core(TM) i5-11300H
	内存: 256GB
	显存: 11GB
开发工具	硬盘: 2TB
	Neo4j 版本号 4.2.6
	Java 版本号 1.8.0_281
	Python 版本号 3.6.8
	Kafka 版本号 2.2.1
	prometheus 版本号 2.17.2
	Vue.js 版本号 4.5.12
	Spring Boot 版本号 2.5.0
日志分析	Jaeger 版本号 1.22.0

6.4.2 运行状态查询测试

在进行运维工作时，运维人员会有不同的查询需求，本系统实现了常用的查询方式。本小节对实现的各个运行状态查询功能进行了测试，共设计了 4 条测试用例，分别对应 4 种查询功能需求，测试结果如表??所示。测试结果表明各个查询功能均已正确实现，下面展示各个测试用例在前端界面的返回结果。

表 6-2 运行状态查询测试用例

内容项	描述
测试目的	测试运行状态查询功能
前置条件	测试机联网、系统查询界面可访问
用例设计	<ol style="list-style-type: none"> 输入查询语句“vpc slb ecs”，查询符合特定路径类型的所有拓扑，点击查询按钮。 输入查询语句“vpc id k1aexj”，模糊查询 id 包含 k1aexj 的 vpc 节点，点击查询按钮。 鼠标悬停在任一组件上方。 鼠标左键点击任一组件。
预期结果	<ol style="list-style-type: none"> 返回查询结果，查询结果为符合“vpc slb ecs”路径类型的所有拓扑。 返回查询结果，查询结果是 id 包含“k1aexj”子序列的 vpc 节点。 组件右上方浮现该组件的属性信息列表。 查询页面右侧弹出该组件的所有指标时序数据。
测试结果	4 个用例经过测试与预期结果相符合
状态	通过

图??显示了搜索符合特定路径类型关系的所有拓扑。如图中所示，在搜索框输入“vpc-slb-ecs”，回车后，下方界面会显示所有由 vpc、slb 和 ecs 三种类型组件构成的拓扑结构。

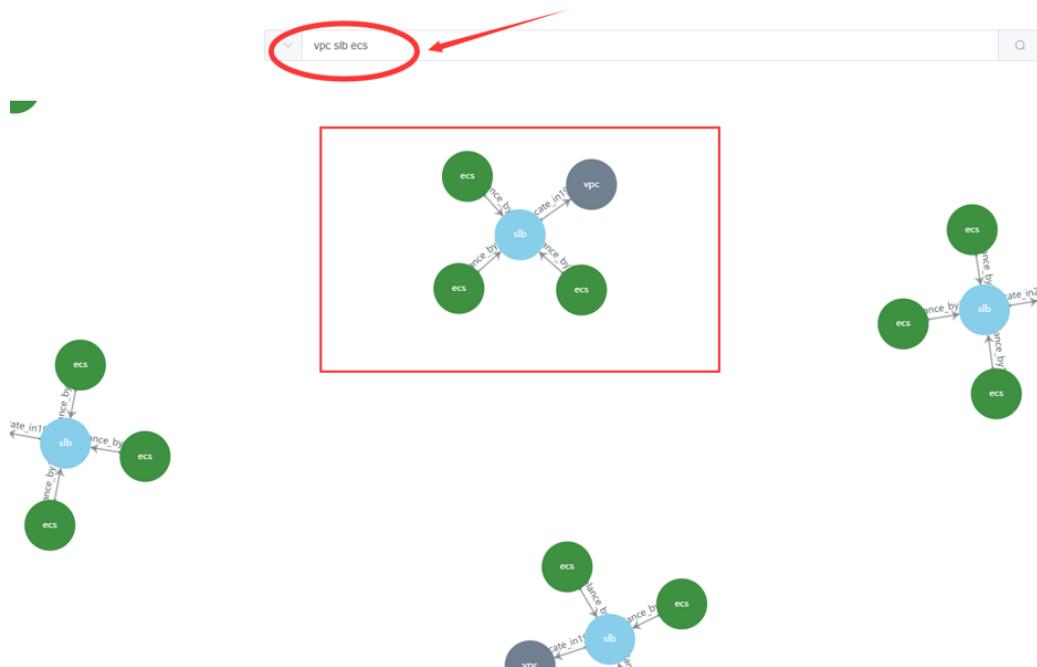


图 6-7 搜索指定路径结果

图??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)。返回结果除了包含符合条件的组件，还包含该组件的周边组件，便于运维人员查看其上下游拓扑。

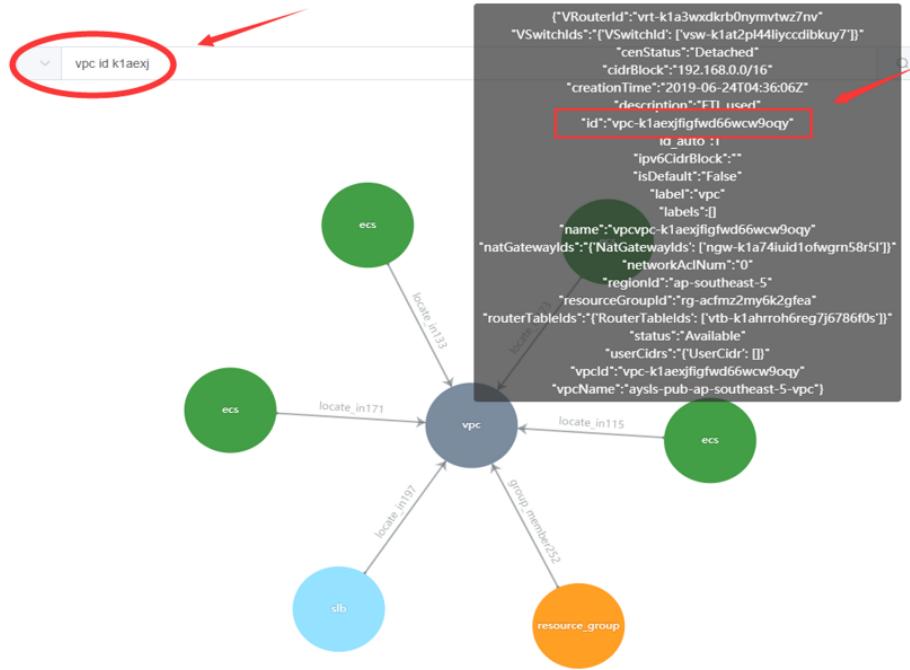


图 6-8 模糊搜索结点结果

图??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。为了显示组件的所有具体属性信息，将鼠标悬停于组件上，右上方的灰色方框中会浮现该组件的属性信息列表。

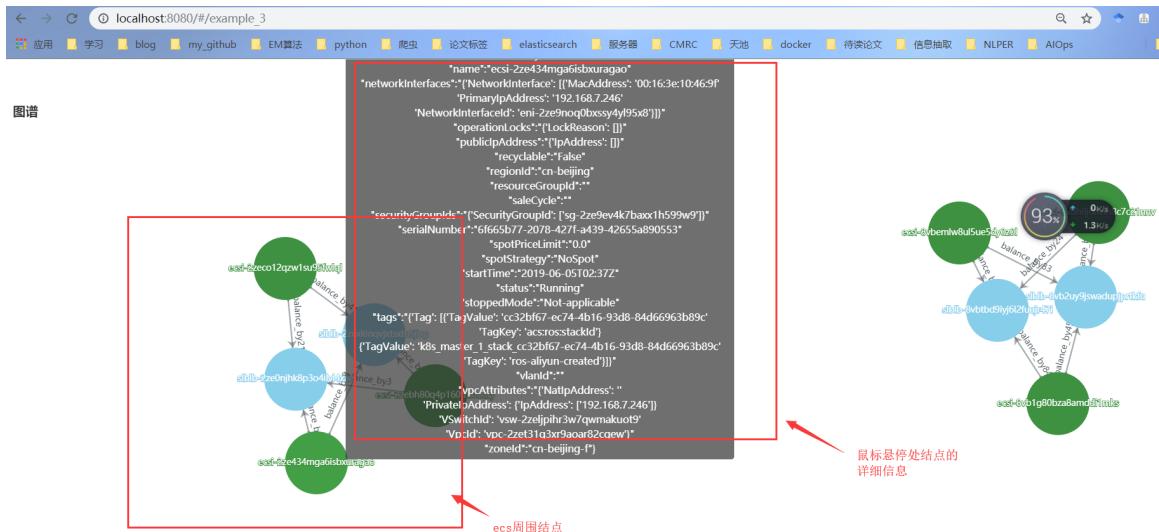


图 6-9 查询组件属性信息结果

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折线图。如图??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 6-10 查询组件指标时序信息结果

6.4.3 组件事件知识图谱查询及调优测试

本小节对组件-事件知识图谱查询及调优功能进行了测试，共设计了 3 条测试用例，分别对应查询及调优，测试结果如表??所示。

表 6-3 组件-事件知识图谱查询及调优测试用例

内容项	描述
测试目的	测试组件-事件知识图谱查询及调优功能
前置条件	系统已经存在故障代号为 f3 的组件-事件知识图谱
用例设计	<ol style="list-style-type: none"> 输入查询语句“f3”，查询故障类型代号为 f3 的组件-事件知识图谱，点击查询按钮。 左键任一条边，选择删除，并再次查询“f3”。 选择两个事件结点，单击右键添加一条因果关系，并再次查询“f3”。
预期结果	<ol style="list-style-type: none"> 返回查询结果，查询结果为故障类型代号为 f3 的组件-事件知识图谱。 返回查询结果，删除的边已经不存在于图中。 返回查询结果，添加因果关系存在于图中。
测试结果	3 个用例经过测试与预期结果相符合
状态	通过

图??为查询到的故障代号为 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；下层为部分集群组件层。

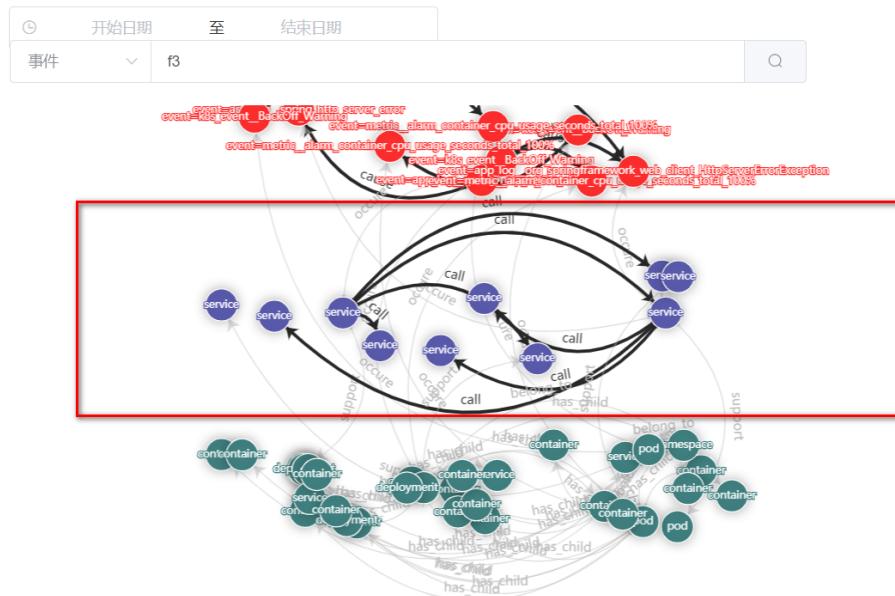


图 6-11 查看 f3 故障类型的组件-事件知识图谱

6.4.4 故障预测测试

本小节对故障预测功能进行了测试，共设计了 3 条测试用例，测试结果如表??所示。

表 6-4 故障预测测试用例

内容项	描述
测试目的	故障预测功能
前置条件	系统存在组件-事件知识图谱，开启了故障检测功能
用例设计	<ol style="list-style-type: none"> 开启故障预测功能，后台注入 f3 故障的根因异常。 关闭故障预测功能，后台注入 f3 故障的根因异常。 开启故障预测功能，后台不注入任何异常。
预期结果	<ol style="list-style-type: none"> 在故障产生前，页面弹出故障预测 top3 结果及其对应匹配度值。 故障预测页面未出现任何信息。 故障预测页面，一直显示“集群状态健康”。
测试结果	3 个用例经过测试与预期结果相符合
状态	通过

图??为实时故障预测界面。在该界面中，实时数据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。事件层会显示可能的故障触发链条。

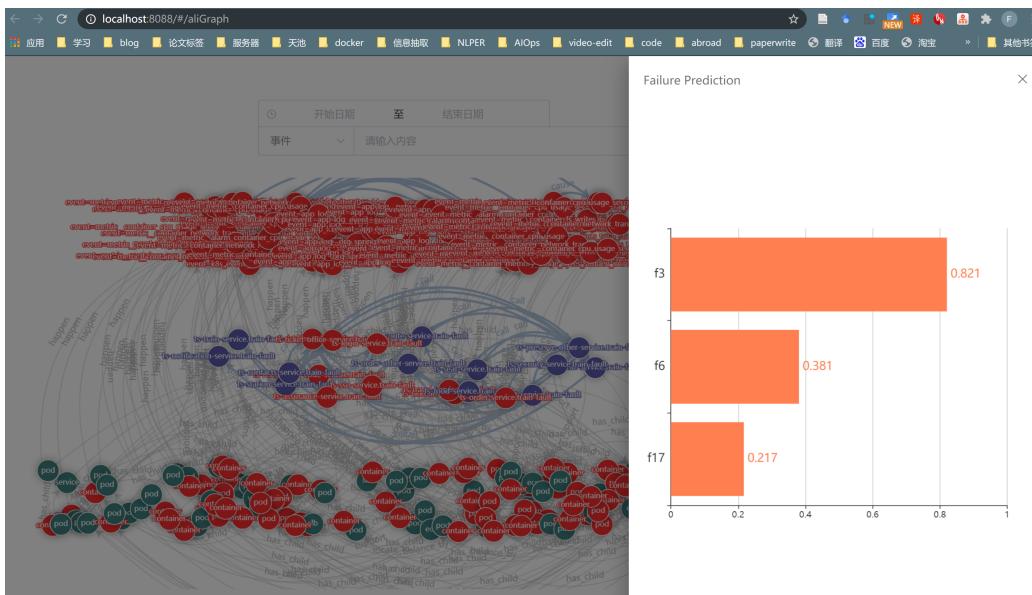


图 6-12 实时故障预测结果

6.4.5 系统性能测试

本节测试了系统的性能表现，主要包括各个查询功能和故障预测的时间消耗。本节对每一个测试项重复了 100 次，最终以平均耗时作为测试结果，如表??所示。由测试统计结果可见，本系统满足了秒级的系统性能需求。

表 6-5 系统性能测试

系统性能测试项	时间消耗
查询特定路径类型的拓扑	0.65s
模糊查询某个组件	0.59s
悬停显示组件信息	0.76s
左键获取组件 指标时序数据	1.71s
搜索某故障类型的 组件-事件知识图谱	0.63s
注入异常到故障预测 正确并弹出结果	2.38s

6.5 本章小结

本章分析了 IT 运维人员在实际工作中的需求，根据第三章、第四章和第五章所提出的各个算法模型，设计了一个基于知识图谱的 IT 运维辅助系统。随后，本章详细实现了多个功能模块，使得系统可以快速整合多源异构数据、自动沉淀生成组件-事件知识图谱，进行实时故障预测。最后，经过系统测试，证明了本系统可以满足 IT 运维人员的实际工作需求。

第七章 总结与展望

7.1 本文工作总结

本文首先描述了IT运维的研究背景及意义，分析了在实际运维场景中存在的问题：多源异构数据难以整合、运维知识表示不足和故障难以准确预知。随后，本文针对这些问题，对国内外研究现状展开了调研，并分析了已有方案的局限性。本文提出了自动构建组件-事件知识图谱、动态表示组件-事件知识图谱和结合知识图谱进行故障预测的一套算法模型。基于这些算法模型，本文设计并实现了基于知识图谱的IT运维辅助系统。本文完成的工作可分为以下4点：

(1) 本文提出了全面整合多源异构数据，自动构建组件-事件知识图谱的方案。在判别事件因果关系时，本文引入了新的事件特征，提升了事件因果关系判别模型的效果。构成的知识图谱包含了高细粒度的多种信息，如软硬组件间关系、指标时序数据和日志数据，解决了多源异构数据难以整合的问题。

(2) 本文提出了适配组件-事件知识图谱的动态表示学习模型。该模型将实体表示分为了语义表示和结构表示，语义表示通过实体文本信息获取，结构表示通过Attention-RGCN获取，实现了实体随上下文变化的动态表示，解决了运维知识表示不足的问题。

(3) 本文提出了引入知识图谱的故障预测模型。该故障预测模型，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，解决了故障难以准确预知的问题。

(4) 基于以上工作，本文设计并实现了基于知识图谱的IT运维辅助系统。该系统能够详细展示集群运行状态，并根据实时发生的事件序列和知识图谱预测故障，提醒运维人员及时采取应对措施，满足了实际的IT运维需求。

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，提出了结合知识图谱、场景实时特征的基于知识图谱的IT运维辅助方案。但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系统性能，并采取相对应的优化措施。

(2) 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通用性故障，能够解决冷启动问题，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。

(3) 添加持续学习功能模块。当故障预测结果与后续实际发生的故障不符合时，可以自动反馈给模型进行学习优化。

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。

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)

