软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(3):1022−1039 [doi: 10.13328/j.cnki.jos.007282] [CSTR: 32375.14.jos.007282]

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

面向复合异常的分布式数据库异常诊断方法*

向清风[1], 邵鋆侠[1], 徐泉清[2], 杨传辉[2]

[1](北京邮电大学 计算机学院 (国家示范性软件学院), 北京 100876)

[2](蚂蚁集团, 浙江 杭州 310013)

通信作者: 邵鋆侠, E-mail: shaoyx@bupt.edu.cn

摘　要: 数据库是计算机服务中的重要基础组件, 然而其运行中可能出现性能异常, 影响业务服务质量. 如何对数据库产生的性能异常进行诊断成为工业界与学术界的热点问题. 近年来, 一系列自动化的数据库异常诊断方法被相继提出, 它们通过分析数据库运行状态, 对数据库整体的异常类型进行判断. 但随着数据规模的不断扩大, 分布式数据库正成为在业界中愈受欢迎的重要解决方案. 在分布式数据库中, 数据库整体由多个服务器节点共同组成. 现有的异常诊断方法难以有效地定位节点异常, 无法识别在多节点上发生的复合异常, 不能感知节点间复杂的性能影响关系, 欠缺有效的诊断能力. 针对上述问题, 提出了一种面向分布式数据库的复合异常诊断的方法: DistDiagnosis. 该方法采用复合异常图对分布式数据库的异常状态进行建模, 在表示各节点异常的同时有效地捕获节点间的相关性. DistDiagnosis 提出了节点相关性感知的根因异常排序方法, 根据节点对数据库整体的影响力有效地定位根因异常. 在国产分布式数据库 OceanBase 上构建了不同场景的异常测试案例. 实验结果表明, 该方法优于其他先进的对比方法, 异常诊断的 *AC*@1、*AC*@3、*AC*@5 最高达到 0.97、0.98 和 0.98, 在各诊断场景中相较于次优方法最多提升了 5.20%、5.45% 和 4.46%.

关键词: 分布式数据库; 异常诊断; 根因分析

中图法分类号: TP311

中文引用格式: 向清风, 邵鋆侠, 徐泉清, 杨传辉. 面向复合异常的分布式数据库异常诊断方法. 软件学报, 2025, 36(3): 1022–1039. http://www.jos.org.cn/1000-9825/7282.htm

英文引用格式: Xiang QF, Shao YX, Xu QQ, Yang CH. Distributed Database Diagnosis Method for Compound Anomalies. Ruan Jian Xue Bao/Journal of Software, 2025, 36(3): 1022–1039 (in Chinese). http://www.jos.org.cn/1000-9825/7282.htm

Distributed Database Diagnosis Method for Compound Anomalies

XIANG Qing-Feng[1], SHAO Ying-Xia[1], XU Quan-Qing[2], YANG Chuan-Hui[2]

[1](School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China)

[2](Ant Group, Hangzhou 310013, China)

Abstract: Databases are important foundational components in computer services. However, performance anomalies may occur during their operation, affecting business service quality. How to diagnose performance anomalies in databases has become a hot issue in industry and academia. Recently, a series of automated database anomaly diagnosis methods have been successively proposed. They analyze the runtime status of the database and determine the overall database anomaly types. However, with the continuous expansion of data scale, distributed databases are becoming an increasingly popular solution in the industry. In a distributed database, which is composed of multiple nodes,

* 基金项目: 国家自然科学基金 (62272054, 62192784); 新一代人工智能国家科技重大专项 (2022ZD0116315); 北京市科技新星计划 (20230484319); 小米青年学者项目
本文由“云原生数据库技术与系统”专题特约编辑杜小勇教授、杨晓春教授、祝园园教授推荐.
收稿时间: 2024-05-27; 修改时间: 2024-07-16, 2024-08-19; 采用时间: 2024-08-29; jos 在线出版时间: 2024-09-13
CNKI 网络首发时间: 2024-12-10

existing anomaly diagnosis methods struggle to effectively locate node anomalies, fail to identify compound anomalies across multiple nodes, and are unable to perceive the complex performance influence relationships between nodes, lacking effective diagnostic capabilities. To address these challenges, this study proposes a distributed database diagnosis method for compound anomalies, named DistDiagnosis. It models the anomalous state of distributed databases using a Compound Anomaly Graph, which not only represents anomalies at each node but also effectively captures the correlations between nodes. DistDiagnosis introduces a node correlation-aware root cause anomaly ranking method, effectively locating root cause anomalies according to the influence of nodes on the database. In this study, anomaly testing cases for various scenarios are constructed on OceanBase, a domestically developed distributed database. Experimental results show that DistDiagnosis outperforms other advanced baselines, achieving the *AC*@1, *AC*@3, and *AC*@5 values of 0.97, 0.98, and 0.98. Compared to the second-best method, DistDiagnosis improves accuracy by up to 5.20%, 5.45%, and 4.46% in each diagnostic scenario.

Key words: distributed database; anomaly diagnosis; root cause analysis

数据库管理系统是计算机服务的重要基础组件, 在互联网、金融、政府单位、科研院校等社会各界中发挥着关键的作用. 在数据库的实际使用中, 由于业务的请求变动、表结构的不合理设计等复杂因素, 可能发生突发的性能下降, 产生性能异常. 此时, 数据库系统提供的服务质量会受到严重的损害, 对下游业务产生不良影响. 这就要求当数据库产生性能异常时能够对当前异常根因进行及时、有效的诊断. 传统的数据库异常诊断流程严重依赖于数据库管理员的专家知识, 效果难以得到保证.

近年来, 为了避免繁琐的人工诊断过程, 一系列自动化的数据库诊断方法被相继提出[1]. 借助机器学习的强大能力, 这些诊断方法能够自动分析数据库监控指标, 对数据库当前运行状态进行判断, 并检测此时产生的异常种类. 如 DBSherlock[2]、AutoMonitor[3]等诊断方法通过分析集中式数据库的监控指标进行异常诊断, 给出数据库整体的运行状态并定位异常类型.

然而, 随着软件系统的复杂化与数据的大规模化, 单机数据库逐渐不能满足日益增长的业务需求. 分布式数据库通过同时管理多个服务器节点, 提供远超单机的数据处理能力, 并兼顾性能与容错的平衡, 正成为受业界青睐的选择[4]. 但由于集群管理存在着复杂性, 在分布式数据库中, 性能异常的定位与诊断变得愈发复杂, 传统单机数据库的异常诊断方法在分布式场景中面临如下问题.

(1) 如何在分布式数据库中准确地定位节点层次的异常. 分布式数据库由多个节点共同组成, 用户查询语句的执行会由各节点协同完成. 当数据库整体出现性能异常时, 可能是节点上产生的单节点异常或多节点异常所导致的[5]. 现有的异常诊断技术主要在数据库整体层次上进行分析, 关注如何判断数据库整体的运行状态与异常类型, 无法有效地定位节点的性能异常. 如 AutoMonitor 通过当前监控指标的变化特征确定异常类型, 但其仅能给出数据库整体的异常类型. 同时, 根据用户查询语句以及对应分布式执行计划的不同, 各数据库节点在运行时存在着复杂的相关性. 现有的诊断方法不能感知节点间的相互影响, 缺乏有效的节点层次异常定位手段.

(2) 如何对分布式数据库中的复合异常进行根因分析. 当性能异常发生时, 产生异常的原因可能不是单一的, 而是由多个具体异常类型共同组成[6]. 例如当数据库吞吐量降低时, 可能同时出现了 CPU 瓶颈与 I/O 瓶颈. 现有数据库诊断技术对复合异常诊断的支持不足, 准确性较低. 同时, 在分布式数据库中, 复合异常的表现则更为复杂, 可能出现多个节点产生多种异常的复合异常的情况, 具体定义与例子见第 2.2 节. 现有方法在处理复合异常时没有考虑如何对节点层级的复合异常进行诊断. 如 DBSherlock 在诊断复合异常时不考虑具体异常在数据库节点上的分布情况, 无法有效地处理分布式环境中的复合异常.

针对上述问题, 本文提出了一种面向复合异常的分布式数据库异常诊断方法 DistDiagnosis, 以实现节点层级的根因异常分析. 首先, 本文提出了基于图的分布式数据库异常建模方法, 构建复合异常图, 对分布式数据库运行状态进行抽象, 有效地捕获分布式数据库节点间关系. 对于各分布式节点, 采用多标签分类技术识别单节点的异常状态. 其次, 本文提出了节点相关性感知的根因异常排序技术, 综合利用各节点异常类型以及该节点与其他节点的相关性对各种可能发生的异常进行排序, 生成由<节点, 异常>对构成的根因异常序列, 实现对节点上复合异常的有效诊断. 综上, DistDiagnosis 在进行异常诊断时能够考虑节点本身异常及节点间关系, 相较于传统诊断方法具有更高的诊断准确性.

本文的主要贡献总结如下.

(1) 本文提出了基于图的分布式数据库异常建模方法, 根据分布式数据库各节点运行时指标构建复合异常图, 并结合节点层级异常诊断器, 实现对分布式数据库的运行状态的抽象, 同时捕获数据库中的节点关系与各单独节点异常情况.

(2) 本文设计了节点相关性感知的根因异常排序技术, 采用加权 PageRank 方法评估复合异常图节点的影响力, 识别对数据库影响较大的节点. 综合利用节点影响力与异常概率计算当前节点在系统整体异常中的重要性, 生成最终的根因异常序列.

(3) 基于上述技术, 本文实现了面向分布式数据库的智能异常诊断方法: DistDiagnosis. 实验结果表明, DistDiagnosis 方案在各类异常场景中有明显的优势, 在分布式异常诊断中效果超过了对比方法, 其 *AC*@1、*AC*@3、*AC*@5 最高达到 0.82、0.86 和 0.92.

本文第 1 节介绍数据库异常诊断的相关工作和研究现状. 第 2 节介绍基础知识, 包括分布式数据库以及本文所研究的分布式数据库异常诊断问题. 第 3 节是本文所提方法 DistDiagnosis 的概述. 第 4 节描述本文所提出的基于图的分布式数据库异常建模方法. 第 5 节描述本文设计的节点相关性感知的根因异常排序技术. 第 6 节通过实验验证本文所提出 DistDiagnosis 方案的有效性. 最后在第 7 节总结全文.

1 数据库异常诊断相关工作

目前国内外有一系列针对数据库异常诊断的相关研究工作, 根据它们所采用的诊断判断依据, 主要可以分为 3 类[1], 包括基于时间分析、基于日志分析、基于监控指标. 其中, 基于时间分析的方法专注于分析数据库中的执行时间指标, 它不能有效地利用数据库的监控能力, 并且难以迁移到不同的数据库系统. 基于日志分析的方法则通过记录日志中的事件记录判断异常的发生, 其异常检测效果依赖于预定义的检测规则与日志粒度. 基于监控指标的异常诊断技术不需要对数据库系统内部进行任何改动, 能够便利地接入数据库本身提供的指标接口, 额外开销小, 是较为通用与主流的方案. 本文主要讨论基于监控指标的数据库诊断工作, 同时本文所设计的 DistDiagnosis 也是基于监控指标的诊断方法.

现有的基于监控指标的数据库异常诊断方法根据技术路线的区别, 主要可以分为两类, 基于规则的诊断技术与基于机器学习的诊断技术.

● 基于规则的诊断技术

Yoon 等人提出了数据库性能诊断框架 DBSherlock[2], 能够帮助用户进行异常分析. DBSherlock 在使用时, 首先需要由人工选定指标时间序列中的异常区域. 之后, DBSherlock 根据正常区域与异常区域中各指标的取值差异, 通过因果模型生成一系列用于判断数据库状态的谓词. 在进行根因分析时, DBSherlock 通过判断当前谓词是否满足来确定当前的数据库运行状态. 最终, DBSherlock 把数据库当前状态及指标满足情况作为根因分析报告, 辅助用户定位异常根因.

ISQUAD[7]则专注于诊断系统中的慢查询, 并通过监测指标的变化模式进行根因分析. 首先, 它通过鲁棒阈值[8]与 T 检验[9]方法识别各指标的尖峰与突变, 从而查找出具有异常变化模式的指标. 之后, ISQUAD 通过聚类方法识别出具备高相似度的指标, 并进一步采用贝叶斯实例模型进行分析. 在实际进行诊断时, ISQUAD 通过指标的异常分数判断当前数据库的异常类型, 并向用户呈现出异常的指标作为根因.

Dundjerski 等人提出了一种面向 Azure SQL 的自动化异常诊断系统[10]. 该系统由 3 种不同的模型共同组成: 通用模型、分类模型和基于规则的专家系统. 其中, 通用模型用于处理数据库的意外行为, 包括查询超时、数据库超时、执行异常等. 通用模型根据监控数据计算异常分数, 并把异常分数与预定义的基准值进行比较. 分类模型旨在通过监控数据的特定模式为用户提供异常解释. 专家系统则通过预定义的判断语句对具体异常进行精确区分. 通过这些不同的诊断模型间的配合, 该异常诊断系统最终能够对不同的真实异常根因进行定位与分析. 但这些模型难以被迁移到其他数据库系统, 通用性相对较弱.

● 基于机器学习的诊断技术

金连源等人提出了 AutoMonitor[3]. 它是一个轻量化的自动诊断框架. 首先, AutoMonitor 采用基于 LSTM 的自编码器[11]检测当前数据库中是否产生了性能异常. 在发现当前数据库异常后, AutoMonitor 采用基于滑动窗口的 Kolmogorov-Smirnov 检验方法为所有监控数据计算异常得分, 最终形成当前数据库的异常向量. 最后通过计算当前异常向量与训练得到的异常向量间的 K 近邻, 确定当前数据库的异常类型. 实验展示出基于学习的 AutoMonitor 方法的根因分析效果能够超过 DBSherlock.

Huang 等人设计了一个异常诊断的基准测试套件: DBPA[6]. 针对目前数据库诊断领域欠缺评测标准的问题, 它们设计了可迁移的异常数据收集框架, 并且开源了 MySQL 与 PostgreSQL 上的根因分析数据集. 在 DBPA 中, 他们采用 XGBoost[12]、随机森林[13]等多种机器学习模型进行根因分析, 并对比了不同方法的诊断效果. 然而, DBPA 提供的数据集及其构造的测试实验主要在单机场景进行, 没有考虑到分布式场景中更复杂的异常情况.

Zhou 等人提出了基于大语言模型的数据库诊断方法 D-Bot[14]. 他们设计了基于树的大模型诊断推理流程, 并采取提示工程引导模型对当前数据库指标进行逐步分析. 同时, D-Bot 提出协同诊断流程, 利用不同的大模型专家诊断数据库产生的不同异常, 通过各专家之间的协同配合推动整体诊断流程, 并借助大模型具备的分析能力生成对数据库当前性能异常的总结报告. D-Bot 生成的报告中将描述当前发生的性能异常以及对应解释, 但其分析流程不考虑到分布式数据库多节点特点, 最终报告中也不能有效地描述分布式数据库异常.

综上, 现有大多数异常诊断工作主要关注单机数据库上的诊断, 忽略了分布式数据库的诊断场景. 同时, 现有方法对复合异常的支持不足, 在复合异常诊断准确性上有所欠缺.

除了上述的异常诊断工作外, 目前还有一些面向分布式环境的异常检测方法被提出, 如 DBCatcher[5]、AutoMAP[15] 、Perseus[16]等. 然而, 这些工作主要关注如何检测当前系统中是否发生异常, 而不关注诊断具体的异常类型. 例如: DBCatcher 提出数据库节点间具有相关性, 并采用相关性变化选出最可能的异常节点. AutoMAP 根据分布式的微服务节点建立服务行为图, 并采用随机游走技术. 之后 AutoMAP 会选出被访问频率更高、更倾向于发生异常的节点. Perseus 则针对大型分布式存储系统中的“慢失败”现象而设计, 通过轻量的回归模型与阈值的动态调整技术发现异常现象. 这些异常检测工作与本文的主要研究目标不同, 本文主要关注分析当前数据库中的异常根因, 并给出相应节点的异常类型等具体解释.

2 基础知识

2.1 分布式数据库

分布式数据库能够管理大规模的服务器集群, 并提供数据的分布式存储与处理能力. 如在 OceanBase[17]分布式数据库中, 其底层由多个对等的 OBServer 服务器节点组成, 用户表能够以分区表的形式存储在各 OBServer 节点上. 对这类分布式存储的数据表的查询语句, 会生成对应的分布式执行计划, 并分发至各节点执行. 由于分布式数据库中各节点需要承担数据处理任务, 任何一个节点的异常都可能对数据库的整体执行性能产生负面影响. 相较于传统集中式数据库, 分布式数据库中的性能异常场景更为复杂多样.

同时, 根据分布式数据库集群节点的性质, 目前分布式数据库可以分为由对等节点组成和由非对等节点组成两类. 如基于 SharedNothing 架构的 OceanBase 由完全对等的 Observer 节点组成; Spanner[18]集群每个分区通过对等的 Span Server 处理用户请求; TiDB[19]集群则可以由行存节点与列存节点等不对等节点组成. 本文工作主要讨论由对等节点组成的分布式数据库.

2.2 分布式数据库性能异常

在现有的数据库异常诊断工作中, 一般将数据库性能异常场景分为普通异常和复合异常[6].

普通异常: 当前数据库整体出现某一种异常, 如单机数据库整体产生 CPU 瓶颈、I/O 瓶颈等.

复合异常: 当前数据库性能异常是由 2 种或 2 种以上的异常同时引起的, 例如同时发生 CPU 瓶颈与负载超常.

在对单机数据库进行诊断时, 现有方案设计的异常分类器输入为数据库整体的监控指标, 对应输出为数据库整体的异常类型. 例如当一个集中式的数据库由于 CPU 计算资源不足产生性能下降时, 现有异常分类方法会判断该数据库处于 CPU 瓶颈状态.

在分布式数据库中, 性能异常场景变得更加复杂. 由于分布式数据库由多个节点组成, 根据这些节点之间的影响关系与逻辑关联, 性能异常可能会产生在分布式数据库的多个节点中. 图 1 描述了分布式数据库中可能存在的复杂异常情况, 包括单节点异常、多节点异常和复合异常.

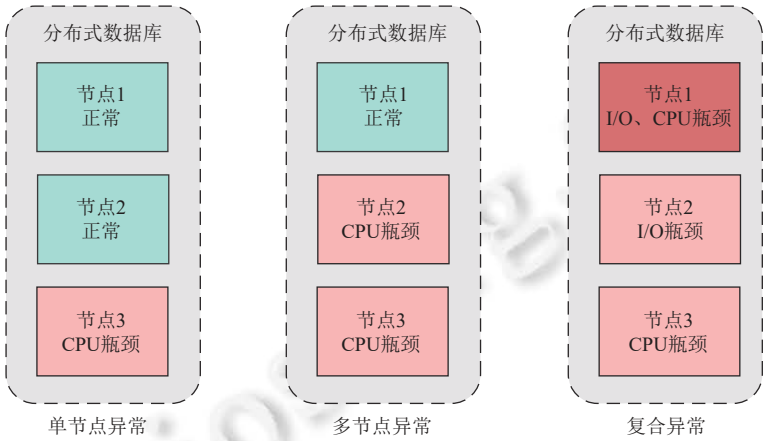

图 1　分布式数据库中复杂的各种异常场景

单节点异常: 分布式数据库某个节点上产生一种特定异常. 如图 1 中的节点 3 单独产生 CPU 瓶颈.

多节点异常: 分布式数据库多个节点上产生一种特定异常. 如图 1 中的节点 2 与节点 3 同时产生 CPU 瓶颈.

复合异常: 分布式数据库多个节点上产生多种异常. 如图 1 中所展示的, 节点 1 同时产生 I/O 与 CPU 瓶颈这 2 种异常, 节点 2 产生 I/O 瓶颈, 节点 3 则产生 CPU 瓶颈. 这是分布式数据库中最复杂的异常情况, 在示例中, 异常种类共有 I/O 瓶颈与 CPU 瓶颈这 2 种, 同时这 2 种异常产生在当前数据库的不同节点上.

2.3　分布式数据库异常诊断问题

为了应对复杂的复合异常场景, 本文将分布式数据库异常诊断定义为一个排序问题. 给定由 N 个节点 v_i 组成的分布式数据库 $DB=\{v_i\}_{i=1}^{N}$, 已知其中每个数据库节点 v_i 对应的监控指标 $M_i=\{m_i^1,m_i^2,\ldots,m_i^d\}$, 其中, d 为监控指标个数, m_i^k 代表节点 v_i 上第 k 种监控指标组成的时间序列, $1\leqslant k\leqslant d$. 同时, 给定每个节点上可能发生的异常类型 L 种, 即异常类型集合 $A=\{ab_1,ab_2,\ldots,ab_L\}$.

这些总数为 $M=N\times L$ 的<节点, 异常>对构成数据库中所有可能产生的异常集合 $R=\{(v_i,ab_j)_m\}_{m=1}^{M}$. 当该分布式数据库产生性能异常时, 各节点上实际产生的异常将共同组成 R 的一个子集 $R^C\subseteq R$, 其代表实际的根因异常集合. 诊断方法需要根据分布式数据库监控指标生成由<节点, 异常>对构成的 Top-k 序列 $D=((v_i,ab_j)_n)_{n=1}^{k}$ 作为诊断结果. 在该序列 D 中, 需要尽可能包含集合 R^C 中的元素, 并尽量排在靠前位置.

该诊断问题定义具有如下优势.

(1) 能够应对分布式数据库中多节点的诊断场景. 在传统问题定义下, 虽然现有工作能够通过增加多标签分类的方法支持分布式节点诊断, 然而所需要的总分类标签个数会成倍增加, 导致难以对异常类型进行观测与分析.

(2) 能够诊断分布式场景中复杂异常情况, 如分布式节点异常、复合异常等. 最终输出结果为<节点, 异常>所组成的序列, 能够支持不同节点异常、不同种类异常及复合异常的结果展示. 在 DistDiagnosis 输出的结果中, 被判断位于较重要地位的异常会排在结果前列. 数据库管理员在分析与利用 DistDiagnosis 时, 可以通过<节点, 异常>对的排序顺序, 逐个进行检查与恢复.

3 方法概览

DistDiagnosis 是一种面向分布式数据库的异常诊断方法, 其概览如图 2 所示.

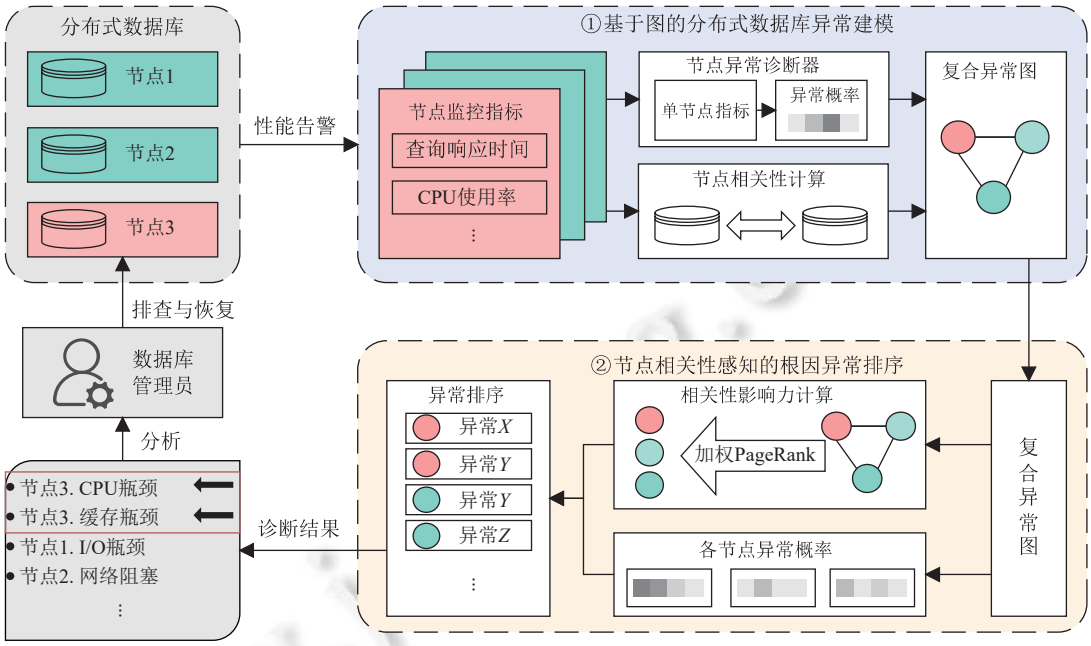

图 2 DistDiagnosis 方法结构示意图

在数据库的实际线上运行中, 存在大量依赖经验或业务要求设定的异常探测阈值, 当发现数据库中存在性能异常时, 会产生数据库告警. 在检测到告警后, 首先, DistDiagnosis 对数据库各节点的监控指标进行分析, 利用基于图的分布式数据库异常建模技术构建描述当前数据库异常状态的复合异常图, 概括各节点异常概率与节点间的相关关系. 接着, 在复合异常图基础上, DistDiagnosis 进行节点相关性感知的异常根因排序, 生成<节点, 异常>形式的根因异常序列作为诊断结果. 最后, 数据库管理员根据 DistDiagnosis 提供的诊断结果进行对数据库节点的异常排查, 修复问题并恢复数据库正常运行.

在总体诊断流程中, DistDiagnosis 主要包含 2 个关键技术: 基于图的分布式数据库异常建模与节点相关性感知的根因异常排序. 具体描述如下.

● 基于图的分布式数据库异常建模

DistDiagnosis 针对分布式数据库构建复合异常图. 该图是对数据库节点关系的抽象, 能够捕获分布式数据库节点间的关联性, 有效地描述数据库当前运行状态. 在构建复合异常图时, DistDiagnosis 令数据库各个节点作为复合异常图中的图节点. 对于每一个节点, DistDiagnosis 进行单节点层级的异常诊断, 判断单个节点上发生各类异常的概率, 并把该异常诊断结果作为复合异常图的图节点属性. DistDiagnosis 进一步利用节点上的监控指标计算节点的相关性, 并以节点相关性作为边权. 在最终生成的复合异常图中, 图节点表示该数据库节点的异常状态, 图节点间的边权表示数据库节点间的影响关系.

● 节点相关性感知的根因异常排序

在这一步中, DistDiagnosis 通过节点间的相关性评估每个节点影响力, 识别影响数据库性能的重要节点. 本方法基于已生成的复合异常图, 采用基于加权 PageRank[20]的无监督图节点排序方法为每个数据库节点计算相应的影响力分数. 其后, DistDiagnosis 根据各节点的影响力分数与每个节点上的异常发生概率对<节点, 异常>进行排

序. 在分布式数据库的运行中, 与其他节点相关性越强的指标, 代表该节点在查询语句执行过程中的重要性相应越高, 其越有可能成为性能异常产生时的根因节点. 节点相关性感知的根因异常排序方法能够有效地识别此类节点, 并根据其影响力获取最终的根因异常诊断序列.

4 基于图的分布式数据库异常建模

在传统单机数据库中, 由于仅需要考虑整体系统级的性能异常, 其异常表示方式相对简单. 往往利用一维的异常向量即可代表数据库当前的复合异常, 其中每个值对应一种异常类型的发生概率. 然而与单机场景不同, 当分布式数据库整体表现出性能异常时, 其原因可以为多个节点上各自产生的异常, 异常诊断方法需要能够感知与定位节点层级的异常. 同时, 在分布式数据库中, 节点间具有复杂的关系, 异常诊断方法需要在进行诊断时能够考虑节点间的影响. 为了有效地描述分布式场景中的节点异常情况, 捕获节点间相互作用关系, DistDiagnosis 提出复合异常图技术对分布式数据库异常状态进行建模.

4.1 复合异常图定义

在 DistDiagnosis 中, 复合异常图被定义为一张有权的无向图: $G(V,E,A,W)$. 其中, $V=\{v_i\}_{i=1}^{N}$, 代表复合异常图中的图节点的集合. 对于含有 N 个节点的分布式数据库而言, 复合异常图中的图节点数也为 N, 图节点 v_i 与数据库各节点相对应. 为了充分表示所有节点间的关系, G 是一张完全图, E 中包含 V 中两两节点间所有的边. $A=\{a_i\}_{i=1}^{N}$ 代表图节点的节点属性集合, 其中每个节点 v_i 的属性 a_i 是一维的异常向量. 该异常向量表示当前节点运行状态, 其中各值对应不同类型异常的发生概率. W 为边权集合, 其元素 w_{ij} 代表节点 v_i 与 v_j 之间的相关性. w_{ij} 越大, 对应节点 v_i 与 v_j 间的相关性也越高. 在复合异常图构建时, 需要通过节点层级异常诊断器计算各节点属性, 利用节点监控指标计算各边上的权重.

在现有的异常诊断工作中, 图已经被用于捕获指标间的相关性关系, 从系统提供的大量监控指标中发掘对性能异常最大的指标. 例如 FluxInfer[21]、MicroCause[22]和 MonitorRank[23]把单项指标抽象为图节点, 并通过图节点排序定位重要指标. DistDiagnosis 则关注数据库节点异常类型的诊断. 给定分布式数据库, 本文中设计的复合异常图 G 可以有效地表示该数据库所有节点的异常状态与各节点间的相互联系, 实现对分布式数据库运行状态的有效建模.

4.2 节点层级异常诊断器

在复合异常图中, 图上节点属性 a_i 代表节点 v_i 自身的异常诊断结果. 为了支持诊断节点上的复合异常, DistDiagnosis 采用多标签分类算法作为节点层级异常诊断器.

该异常诊断器的输入为 OceanBase 节点 v_i 上的监控指标构成的多维时间序列 $M_i=\{m_i^1,m_i^2,\ldots,m_i^d\}$. DistDiagnosis 基于专家经验选出 40 项 OceanBase 所提供的监控指标, 包括 SQL 统计、性能统计、缓存等, 以及 dstat 工具所监控的 20 项主机物理资源指标, 包括 CPU 统计、内存统计、I/O 统计与网络统计等, 并以 5 s 为间隔收集监控指标, 取时间窗口 30 s 内的指标时间序列以供诊断. 综上, 对于每一指标时间序列 m_i^k, $1\leqslant k\leqslant d$, 其采样间隔为 5 s, 总长度为 30 s, $d=60$.

对于 L 种可能的异常类型 $\{ab_1,ab_2,\ldots,ab_L\}$, 节点层级异常诊断器的输出结果为一个 L 维的向量 $a_i=(score_i^1,score_i^2,\ldots,score_i^L)$, 其中每一维 $score_i^l$ 代表当前节点 v_i 产生第 l 种异常 ab_l 的概率. DistDiagnosis 采用 XGBoost 作为节点层级异常诊断器的具体模型, 并训练 L 个预测概率的二分类模型实现对多种类异常的同时诊断. 为了满足 XGBoost 的输入要求, DistDiagnosis 会把 M_i 展平为一维向量.

● 诊断器的训练

在训练阶段中, 需要由数据库管理员辅助进行异常诊断与训练数据标注. 当数据库产生性能异常时, 数据库管理员首先人工进行异常节点定义, 并标注各异常节点上产生的具体异常类型. 在完成人工的异常分析后, 各节点的诊断结果即被加入节点层级异常诊断器的训练数据中. 随着数据的增加, 节点层级异常诊断器将逐渐生成更准确

的预测结果, 并辅助数据库管理员进行诊断.

● 图节点属性的预测

在利用节点层级异常诊断器进行复合异常图构建时, 对于每一图节点, 都利用该诊断器生成其单节点异常向量, 作为其复合异常图中的节点属性. 采用单节点层级的诊断器优势在于其避免了高维输入带来的准确性下降. 在利用复合异常图技术时, DistDiagnosis 把复杂的多节点分类任务拆解为多个单节点分类任务, 不需要训练同时预测所有节点的复杂诊断模型, 从而取得更优秀的诊断效果. 同时, 分布式数据库往往具备节点上的扩、缩容能力, 能够基于容器化环境动态增减运行节点, 采用节点层级异常诊断器可以更好地适应集群中节点数目的变化, 提升 DistDiagnosis 的可扩展性.

4.3 节点相关性计算

在复合异常图中, DistDiagnosis 通过皮尔逊相关系数[24]来度量节点间的相关性, 能够有效地计算时间序列间的影响关系. 对于两个时间序列 x,y, 它们的相关系数计算为:

$$r(x,y)=\frac{\sum((x_i-\bar{x})(y_i-\bar{y}))}{\sqrt{\sum(x_i-\bar{x})^2\sum(y_i-\bar{y})^2}} \tag{1}$$

给定节点 v_i 与 v_j, 其对应的监控指标分别为 $M_i=\{m_i^1,m_i^2,\ldots,m_i^d\}$, $M_j=\{m_j^1,m_j^2,\ldots,m_j^d\}$. 这些监控指标充分反映了节点的当前状态, 能够用来度量节点运行性能. DistDiagnosis 通过节点间监控指标的相关性代表节点间执行性能与处理状态的相关性. 同时, 为了全面考虑数据库节点上的不同指标, DistDiagnosis 通过计算各指标相关系数绝对值的平均值作为两节点间的边权, 其计算方式如下:

$$w_{ij}=\frac{1}{d}\sum_{k=1}^{d}|r(m_i^k,m_j^k)| \tag{2}$$

在最终形成的复合异常图中, 图节点间边的权重代表分布式数据库节点间的影响. 两个节点的权重取值越高, 它们运行时监控指标间的相关性水平就越高, 对应反映出其数据库运行时性能表现的相互关联性也就越强.

5 节点相关性感知的根因异常排序

在进行异常的根因分析时, 一种简单的做法是直接将<节点, 异常>对根据每一异常的预测概率进行排序. 然而这种做法有一定的缺陷, 它把所有节点均等对待, 完全依赖于单节点层级的异常检测能力, 无法考虑分布式数据库节点间存在的复杂关系. 在用户查询语句的分布式执行中, 由于数据倾斜[25]、分区键划分不均[26]、执行计划算子放置[27]等原因, 会导致某些节点对数据库整体执行性能有更大的影响. 为了有效地考虑节点间的相关关系, 本文设计了节点相关性感知的根因异常排序技术. DistDiagnosis 根据节点间的指标的相关程度, 判断该节点在整体数据库中的影响力, 并结合各节点影响力与异常概率进行最终的根因异常排序.

5.1 节点相关性影响力计算

首先计算各分布式数据库节点所对应的影响力分数. 在复合异常图中, 各图节点对应分布式数据库节点, 节点间的边权则反映了两节点间性能表现的相关性. 通过分析复合异常图中各图节点在图中的影响力, 即可获取数据库节点在数据库整体中的影响力与重要程度.

为了在计算图节点影响力时有效地考虑整张复合异常图, DistDiagnosis 采用加权 PageRank 算法对各节点的影响力进行计算. 该算法是一种在带权图上的无监督节点排序方法. 与普通的 PageRank 方法不同, 它能够在进行基于随机游走的图节点评估的同时, 考虑各边上不同权重带来的差异. 对于相邻边上权重更高的节点, 加权 PageRank 为其计算出相应更大的影响力并分配更高的排名. 对于每个节点 v_i 而言, 其对应的加权 PageRank 影响力计算为:

$$PR(v_i)=(1-d)+d\times\sum_{v_j\in N(v_i)}PR(v_j)\times w_{ij}\times w_{ji} \tag{3}$$

其中, $N(v_i)$ 代表 v_i 的所有邻居节点, d 为超参数, w_{ij} 为 v_i 的出边权重, w_{ji} 为入边权重.

由于复合异常图是一张无向图, 公式 (3) 即等价于:

$$PR(v_i)=(1-d)+d\times\sum_{v_j\in N(v_i)}PR(v_j)\times w_{ij}^2 \tag{4}$$

在该影响力分数 $PR(v_i)$ 的每轮计算中, 节点 v_i 与其所有邻居的边权会被不断聚合, 并反映到影响力的最终取值中. 对于与其他数据库节点相关性更高的节点而言, 其在复合异常图中对应的邻边权重也会更高. 通过加权 PageRank 的计算后, 这类重要节点也将获得较高的影响力分数. 在对复合异常图进行分析时, 在图中影响力越高的节点对数据库性能的影响相对就越大, 也就越趋于成为真正的根因异常节点.

5.2 根因异常排序

在进行最终的根因异常排序时, DistDiagnosis 同时考虑每个节点的影响力与每个节点对应的异常发生概率. 在排序中, 每个<节点, 异常>对 (v_i,ab_l) 的重要性计算为:

$$Importance_{il}=PR(v_i)\times score_i^l \tag{5}$$

它是各节点影响力与各类异常发生概率的乘积.

根据该重要性得分, DistDiagnosis 对所有可能存在的<节点, 异常>对进行排序, 形成 Top-k 根因异常诊断序列 $D=((v_i,ab_j)_n)_{n=1}^k$, 作为输出的根因分析结果. 对于由 N 个节点组成的分布式数据库 $DB=\{v_i\}_{i=1}^N$, 每个节点可能产生的异常种类为 L 种的情况下, k 取值最大为 $N\times L$.

与直接根据<节点, 异常>对 (v_i,ab_l) 的异常得分 $score_i^l$ 排序相比, 采用节点相关性感知的根因异常排序有效地利用了复合异常图, 并通过加权 PageRank 算法引入节点间的相关关系. 在分布式数据库的执行中, 与其他节点相关性越强的指标, 代表该节点在查询执行过程中的重要性相应越高, 就越有可能成为性能异常产生时的根因节点, 其对应发生的异常应该处在最终结果的靠前位置. 节点相关性感知的根因异常排序方法能够有效地识别此类节点, 并根据其重要性修正根因诊断序列. 算法 1 描述了 DistDiagnosis 的整体执行流程.

算法 1. DistDiagnosis 异常诊断算法.

输入: 当前分布式数据库各节点 $DB=\{v_i\}_{i=1}^N$, 各节点监控指标序列 $M=\{M_i\}_{i=1}^N$;
输出: 由<节点, 异常>对 (v_i,ab_l) 组成的根因诊断序列 D.

1. //基于图的分布式数据库异常建模
2. **for** $v_i\in DB$ **do**
3. 图节点属性计算 $a_i\leftarrow XGBoostClassifier(M_i)$, 其中, $a_i=(score_i^1,score_i^2,\ldots,score_i^L)$;
4. **for** $v_j\in DB$ and $v_j\neq v_i$ **do**
5. 节点邻边权重计算 $w_{ij}\leftarrow(1/d)\times\sum_{k=1}^d|r(m_i^k,m_j^k)|$, 这里, $m_i^k\in M_i$, $m_j^k\in M_j$, r 为皮尔逊相关系数;
6. **end**
7. **end**
8. //节点相关性感知的根因异常排序
9. **while** $PR(v_i)$ 每轮改变量大于一定的阈值 **do**
10. **for** $v_i\in DB$ **do**
11. $PR(v_i)=(1-d)+d\times\sum_{v_j\in N(v_i)}PR(v_j)\times w_{ij}^2$;
12. **end**
13. **end**
14. **for** $v_i\in DB$ **do**
15. $Importance_{il}\leftarrow PR(v_i)\times score_i^l$;
16. **end**
17. $D\leftarrow$ 根据 $Importance_{il}$ 对所有 (v_i,ab_l) 进行排序;

6 实验分析

6.1 实验设置

为了验证 DistDiagnosis 在分布式数据库诊断时的有效性, 本文在多种异常场景中进行了对比实验. 本节介绍相应的实验设置.

(1) 实验环境

本实验在由 4 台 Linux Server 组成的分布式集群中进行, 操作系统为 CentOS 7.9. 每台服务器配置为 32 核 AMD EPYC Milan@2.55 GHz 处理器, 64 GB 内存以及 500 GB SSD. 实验中采用的国产分布式数据库为 OceanBase 社区版 4.3.0. 其中, 3 台服务器作为 OBServer 节点部署 OceanBase 分布式集群, 1 台服务器用于进行压测.

(2) 异常类型

表 1 展示了本实验中采用的异常类型, 具体类型选定参考文献 [2,3,6] 以及 OceanBase DBA 的建议, 总共包括 6 种不同的异常类型.

表 1 OceanBase 数据库中常见异常类型

异常类型	具体描述
CPU瓶颈	数据库外部进程抢占该节点CPU计算资源
I/O瓶颈	数据库外部进程抢占该节点I/O资源
节点网络瓶颈	数据库外部进程抢占该节点网络资源
缓存瓶颈	OceanBase数据库节点缓存大小不足
负载超常	用户和事务的并发请求过大
索引过多	数据表中非必要的索引过多, 影响性能

在这 6 种异常类型中, 对于 CPU 瓶颈、I/O 瓶颈、阶段网络瓶颈与缓存瓶颈而言, 它们可以在数据库底层的 1 个或多个节点上触发. 对于负载超常与索引过多而言, 它们在触发时会同时反映在各个节点上. 比如由于数据表将存储于数据库的各个节点中, 当数据表上建立的索引过多时, 所有保存该数据表的节点上都会产生索引过多异常.

在收集异常数据时, 首先数据库运行 TPC-C 工作负载, 然后再进行各类异常的触发. 对于 CPU 瓶颈与 I/O 瓶颈, 本实验中采用 stress-ng 工具在对应节点主机上抢占对应资源. 对于缓存瓶颈, 实验中通过调节 OceanBase 中的单个节点的 memstore_limit_percentage 参数触发. 负载超常异常通过发送大量的并发请求触发. 节点网络瓶颈通过 Linux 内置流量控制工具 tc 实现. 当触发索引过多异常时, 我们在进行压测前手动在数据表上建立索引.

(3) 异常场景

根据表 1 中异常类型, 本实验构造了 4 种异常诊断的测试场景, 包括单节点异常场景、多节点异常场景、单节点复合异常场景、多节点复合异常场景.

单节点异常场景: 分布式数据库 3 个节点中的某 1 个节点产生表 1 中的异常.

多节点异常场景: 分布式数据库 3 个节点中的 2 个节点产生一种异常.

单节点复合异常场景: 分布式数据库的某 1 个节点上产生不同种类的异常.

多节点复合异常场景: 分布式数据库的多个节点上产生异常, 并且包含不同种类的异常.

表 2 详细展示了测试中采用的所有异常场景.

(4) 指标收集

现有的主流分布式数据库都提供了完善的监控系统, 能够对系统整体以及底层的分布式节点上的多种指标进行检测, 包括了操作系统、事务、磁盘、CPU 等多个方面的详尽数据. 在 OceanBase 分布式数据库中, 各节点上都有超过 400 个监控指标可供使用. 然而, 这些指标中有一些与数据库运行性能关联较弱, 对进行数据库诊断帮助

较小. 本实验中通过人工挑选的方式选择了 40 个关键监控指标, 同时, 本实验中也额外采用 Linux 工具 dstat 提供的 CPU、内存、磁盘、网络统计的 20 个主机监控指标. 在收集实验数据时, 首先令数据库进行压测, 模拟其正常运行场景, 之后每 5 s 检测一次 OceanBase 各节点指标, 并以时间序列的形式保存. 在收集异常数据时, 对于每一种具体异常场景通过改变异常节点位置或改变异常触发程度的方式收集 10 组数据, 总共形成 160 组异常数据, 每组数据长 180 s.

表 2　异常场景描述

异常场景	编号	异常类型	场景描述
单节点异常场景	A1	CPU瓶颈	单个节点触发CPU瓶颈
	A2	I/O瓶颈	单个节点触发CPU瓶颈
	A3	节点网络瓶颈	单个节点触发节点网络瓶颈
	A4	缓存瓶颈	单个节点触发缓存瓶颈
多节点异常场景	B1	CPU瓶颈	2个节点触发CPU瓶颈
	B2	I/O瓶颈	2个节点触发I/O瓶颈
	B3	节点网络瓶颈	2个节点触发节点网络瓶颈
	B4	缓存瓶颈	2个节点触发缓存瓶颈
单节点复合异常场景	C1	CPU瓶颈+I/O瓶颈	单个节点触发CPU瓶颈与I/O瓶颈
	C2	CPU瓶颈+节点网络瓶颈	单个节点触发CPU瓶颈与节点网络瓶颈
	C3	I/O瓶颈+节点网络瓶颈	单个节点触发I/O瓶颈与节点网络瓶颈
	C4	缓存瓶颈+I/O瓶颈	单个节点触发缓存瓶颈与节点网络瓶颈
多节点复合异常场景	W1	CPU瓶颈+负载超常	1个节点CPU瓶颈, 数据库触发负载超常
	W2	I/O瓶颈+负载超常	1个节点I/O瓶颈, 数据库触发负载超常
	W3	节点网络瓶颈+索引过多	1个节点网络瓶颈, 数据库触发索引过多
	W4	缓存瓶颈+索引过多	1个节点缓存瓶颈, 数据库触发索引过多

(5) 对比方法

在实验中, 我们采用的对比方法如下.

DBSherlock: 在进行异常诊断时, 根据用户指定的异常区域形成关于各关键指标的异常过滤谓词, 并根据当前指标的变化状态判定异常状态, 具有较强的可解释能力.

AutoMonitor: 基于改进的 KNN 算法进行异常诊断, 能够有效地识别当前数据库指标变化, 分析当前数据库运行状态最佳匹配的异常类型.

XGBoost 和 RandomForest: 参考 DBPA 中设计的对比方法, 本实验中还采用了其他基于机器学习的分类器进行实验, 包括 XGBoost 和 RandomForest.

本文中所研究的分布式异常诊断问题的诊断结果为一个<节点, 异常>对组成的序列, 但现有方法并非为分布式异常诊断而设计的. 为了将对比方法迁移到分布式诊断场景中, 对于 DBSherlock、XGBoost 和 RandomForest, 我们为其训练多个二分类器来实现对每个<节点, 异常>的预测, 把各方法预测的<节点, 异常>得分按从高到低次序形成诊断序列, 预测得分相同则随机排序. 对于 AutoMonitor, 其设计的基于 KNN 的异常诊断方法能够支持对不同异常类型的打分, 从而实现异常诊断序列的生成.

(6) 评估指标

本实验中采用 *Precision@k* 与 *AC@k* 指标对各方法的诊断效果进行评估. 它们被广泛用于评价根因分析结果[21,22,28]. *Precision@k* 代表前 k 个诊断结果中包含准确根因异常的精度, *AC@k* 代表在多个异常案例中的平均精度. 给定异常场景集合 C, 对于其中一个案例 c, 其诊断精度为:

$$Precision@k = \frac{\sum_{i<k} D^c[i] \in R^c}{\min(k, |R^c|)} \tag{6}$$

对于整个异常场景集合 C, 有:

$$AC@k = \frac{1}{|C|}\sum_{c\in C} Precision@k \tag{7}$$

其中, R^c 为该异常案例的实际产生的根因异常; $|R^c|$ 和 $|C|$ 分别为 R^c 和 C 的元素个数; $D^c[i]$ 代表根因分析所得到结果序列 D^c 中的第 i 个根因.

6.2 效果对比与分析

(1) 单节点异常场景

表 3 展示在 4 种单节点异常诊断场景中, 各方法的 *Precision@k* 指标. 图 3 展示在单节点异常场景中各方法的 *AC@k* 指标.

表 3 单节点异常诊断场景中各方法 *Precision@k*

异常场景	A1			A2			A3			A4		
	k=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5
DBSherlock	0.83	0.86	0.86	0.77	0.79	0.80	0.68	0.73	0.79	0.43	0.55	0.63
AutoMonitor	0.83	0.90	0.95	0.83	0.88	0.89	0.62	0.77	0.84	0.48	0.57	0.71
RandomForest	0.84	0.84	0.84	0.94	0.95	0.95	0.77	0.85	0.92	**0.88**	**0.91**	**0.93**
XGBoost	0.80	0.80	0.80	0.95	0.95	0.95	0.79	0.88	0.90	0.74	0.80	0.87
DistDiagnosis	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.81**	**0.94**	**0.96**	0.82	0.82	0.86

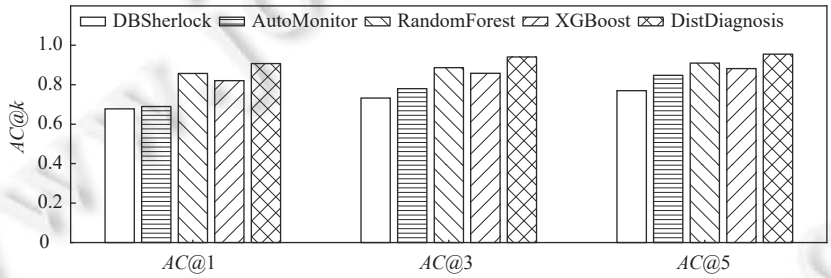

图 3 单节点异常诊断场景中各方法 *AC@k*

在单节点异常诊断场景中, 实验数据展示 DistDiagnosis 的诊断效果总体优于其他对比方法, DistDiagnosis 在为分布式数据库底层的单个节点检测异常时具有一定的优势, 能够有效地在多个节点中定位到异常节点. 在 CPU 瓶颈 (A1) 与 I/O 瓶颈 (A2) 的诊断中, DistDiagnosis 能够有效地利用节点层级的系统监控指标, 其诊断效果明显优于其他方法. 与次优的 RandomForest 相比, 在单节点诊断场景中, DistDiagnosis 分别在 *AC@1*、*AC@3* 和 *AC@5* 上提升了 0.05、0.05 和 0.04.

(2) 多节点异常场景

表 4 和图 4 展示在 4 种多节点异常诊断场景中, 各方法的 *Precision@k* 和 *AC@k* 指标.

表 4 多节点异常诊断场景中各方法 *Precision@k* 对比

异常场景	B1			B2			B3			B4		
	k=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5
DBSherlock	0.82	0.83	0.84	0.84	0.88	0.92	0.73	0.75	0.80	0.65	0.73	0.77
AutoMonitor	0.84	0.93	0.93	0.83	0.85	0.90	0.85	0.84	0.94	0.85	0.90	0.92
RandomForest	0.98	0.98	1.00	0.96	0.95	1.00	0.95	0.97	0.97	0.87	0.88	0.88
XGBoost	1.00	0.98	**1.00**	0.95	0.97	1.00	0.96	0.97	0.97	0.88	0.90	0.90
DistDiagnosis	**1.00**	**1.00**	0.97	**0.97**	**1.00**	**1.00**	**0.96**	**0.98**	**0.98**	**0.93**	**0.92**	**0.94**

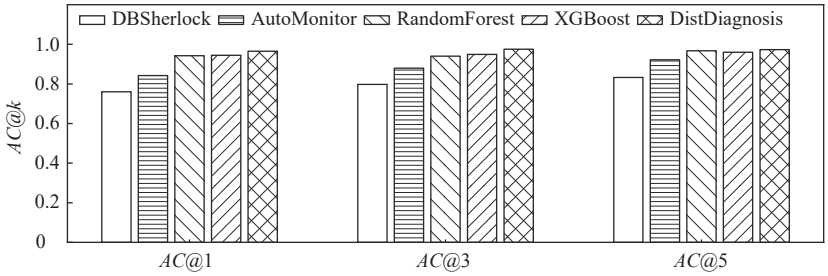

图 4 多节点异常诊断场景中各方法 *AC@k* 对比

由实验结果可见, 当分布式数据库的多个节点产生异常时, DistDiagnosis 依然能够对异常进行有效的诊断. 在各异常场景中, DistDiagnosis 基本都能取得最优或接近最优的效果, 表现出了较好的泛用性. 如图 4 所示, 在多节点异常场景的平均诊断效果中, DistDiagnosis 的 *AC*@1、*AC*@3 和 *AC*@5 指标能够达到 0.96、0.98 和 0.98, 总体好于其他所有对比方法. 在计算复合异常图时, DistDiagnosis 能够考虑节点间的性能关联, 并根据性能关联形成最终的排序结果. 同时, 在为每个节点诊断异常时, DistDiagnosis 采用节点层级异常诊断器, 能够有效地避免同时诊断多个节点时, 其他节点指标带来的噪声与干扰, 从而获取更准确的诊断结果.

(3) 单节点复合异常场景

表 5 与图 5 展示在单节点复合异常场景中各诊断方法的效果. 在单节点复合异常诊断场景, DistDiagnosis 的 *AC*@1、*AC*@3 和 *AC*@5 能够达到 0.97、0.98 和 0.98, 与该场景中次优的 RandomForest 相比提高了 0.03、0.05 和 0.04. 可见, 在与传统单机数据库复合异常所类似的场景中, DistDiagnosis 仍然具备一定的优势. 本方法在能够处理的诊断场景上拥有一定的泛用性, 能够应对不同诊断任务.

表 5 单节点复合异常诊断场景中各方法 *Precision@k* 对比

异常场景	C1			C2			C3			C4		
	k=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5
DBSherlock	0.83	0.85	0.84	0.68	0.71	0.74	0.62	0.64	0.68	0.55	0.57	0.67
AutoMonitor	0.81	0.86	0.88	0.70	0.76	0.77	0.65	0.70	0.78	0.50	0.58	0.70
RandomForest	0.82	0.89	0.89	1.00	0.96	0.96	0.93	0.95	0.97	0.94	0.87	0.95
XGBoost	0.90	0.92	0.93	0.92	0.92	0.94	0.97	0.97	0.97	0.93	0.93	0.94
DistDiagnosis	**0.93**	**0.98**	**0.98**	**1.00**	**1.00**	**1.00**	**1.00**	**0.98**	**0.98**	**0.94**	**0.97**	**0.97**

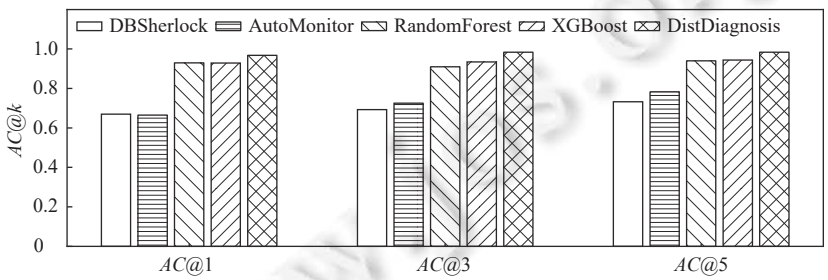

图 5 单节点复合异常场景中各方法 *AC@k* 对比

(4) 多节点复合异常场景

表 6 与图 6 展示在多节点复合异常场景中各诊断方法的效果. 在多节点复合异常诊断场景, DistDiagnosis 有着较为明显的优势. 如图 6 所示, DistDiagnosis 的 *AC*@1、*AC*@3、*AC*@5 能够达到 0.90、0.91 和 0.92, 相对于 XGBoost 提高了 5.20%、5.45% 和 4.46%. 在只取首个异常诊断项的 *AC*@1 指标上, DistDiagnosis 与次优方法

XGBoost 存在 0.05 的差距. 这体现出 DistDiagnosis 在多节点复合异常场景中, 仍然能够保持准确的识别异常能力, 并为用户选择最有可能的根因异常作为排序首位.

表 6 多节点复合异常诊断场景中各方法 *Precision@k* 对比

异常场景	W1			W2			W3			W4		
	k=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5	*k*=1	*k*=3	*k*=5
DBSherlock	0.64	0.69	0.75	0.83	0.83	0.92	0.53	0.53	0.61	0.57	0.61	0.73
AutoMonitor	0.72	0.73	0.76	0.69	0.73	0.89	0.72	0.82	0.87	0.55	0.53	0.74
RandomForest	0.88	0.90	0.90	0.86	0.91	0.91	0.81	0.81	0.88	0.73	0.78	0.82
XGBoost	**0.93**	**0.93**	**0.95**	0.88	0.88	0.88	0.90	0.90	0.91	0.75	0.75	0.80
DistDiagnosis	0.90	0.92	0.93	**0.95**	**0.95**	**0.95**	**0.92**	**0.96**	**0.97**	**0.81**	**0.84**	**0.84**

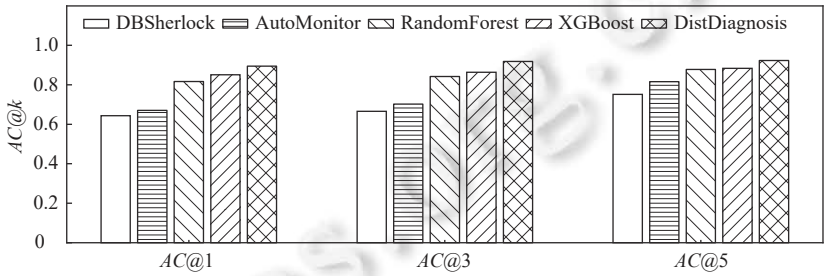

图 6 多节点复合异常场景中各方法 *AC@k* 对比

6.3 消融实验

表 7 集中展示消融实验效果, 其中, DistDiagnosis-w/o-CR 代表去除节点相关性感知的根因异常排序技术后的 DistDiagnosis 方法.

表 7 DistDiagnosis-w/o-CR 与 XGBoost 以及 DistDiagnosis 的对比效果集中展示

异常场景	对比方法	*AC@1*	*AC@3*	*AC@5*
单节点异常	XGBoost	0.82	0.86	0.88
	DistDiagnosis-w/o-CR	0.88	0.93	0.91
	DistDiagnosis	**0.91**	**0.94**	**0.95**
多节点异常	XGBoost	0.95	0.95	0.97
	DistDiagnosis-w/o-CR	0.95	0.96	0.96
	DistDiagnosis	**0.97**	**0.98**	**0.97**
单节点复合异常	XGBoost	0.93	0.93	0.94
	DistDiagnosis-w/o-CR	0.94	0.94	0.85
	DistDiagnosis	**0.97**	**0.98**	**0.98**
多节点复合异常	XGBoost	0.85	0.86	0.88
	DistDiagnosis-w/o-CR	**0.90**	0.90	0.91
	DistDiagnosis	0.89	**0.92**	**0.92**

从实验结果中有以下发现. 1) 虽然本方法在单节点诊断器上采用了 XGBoost, 但得益于所提出的异常建模技术, DistDiagnosis 与 DistDiagnosis-w/o-CR 通过单节点指标对各节点异常类型进行诊断, 避免了同时处理数据库所有指标时的高维输入, 获得了超过 XGBoost 的效果. 2) 在大部分场景中, 本文提出的节点相关性感知的根因异常排序技术能够提高诊断效果; 在单节点异常、多节点异常与复合异常场景中, DistDiagnosis 的诊断效果能够全方面超越 DistDiagnosis-w/o-CR.

6.4 DistDiagnosis 诊断开销对数据库性能影响的分析

为了验证 DistDiagnosis 运行时的额外开销对 OceanBase 数据库性能产生的影响, 本节通过实验来对比采用

DistDiagnosis 前后的数据库事务处理性能. 图 7 展示了不同情况下, OceanBase 数据库在执行 TPC-C 时的每分钟总事务处理数 (transactions per minute, TPM). 其中, OceanBase 代表不使用 DistDiagnosis 时的数据库性能. OceanBase-w-Monitor 代表运行 DistDiagnosis 监控指标采集, 但不进行诊断运算时的数据库性能, 其模拟使用 DistDiagnosis 但不触发诊断时的应用场景. OceanBase-w-Diagnosis 则在每次收集监控指标时进行完整的 DistDiagnosis 诊断. 其中, 诊断运算由集群中的一个节点承担, 其模拟了运行较重诊断任务的场景.

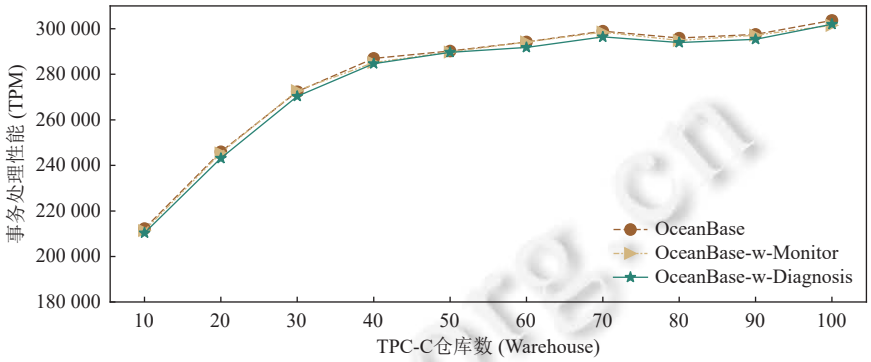

图 7 不同仓库数下 DistDiagnosis 对数据库的性能影响

由图 7 中结果可见, 在各种不同的 TPC-C 仓库 (Warehouse) 数量下, DistDiagnosis 收集监控的性能开销极小, 与 OceanBase 相比, OceanBase-w-Monitor 性能最多下降 0.72%, 几乎对性能无影响. 同时, 即使 DistDiagnosis 不断进行诊断, OceanBase-w-Diagnosis 与 OceanBase 相比也仅有少量的性能损失, 性能最多仅下降 1.17%. 这说明 DistDiagnosis 不会产生过大的额外性能开销, 能够被用于线上系统辅助诊断.

6.5 集群新增节点对诊断效果影响的分析

为了展示 DistDiagnosis 适应数据库节点规模变化的优势, 本节展示在数据库集群增加新节点时的 DistDiagnosis 诊断效果. 新加入的节点配置与集群中已有的节点相同. 表 8 展示了增加新节点后的诊断效果对比. 其中, DistDiagnosis-3 代表仅通过原本三节点集群上的异常数据进行训练, 并直接在新集群进行诊断测试; DistDiagnsis-4 则根据新集群环境下的异常数据重新对节点层级异常诊断器进行训练. 其他对比方法由于不能直接迁移, 必须在新集群中重新训练. 新集群中的异常数据采用与第 6.1 节 (4) 中相同的方式进行收集.

表 8 增加一个新节点后的 DistDiagnosis 效果展示

异常场景	A1			B1			C1			W1		
	k=1	k=3	k=5	k=1	k=3	k=5	k=1	k=3	k=5	k=1	k=3	k=5
RandomForest	0.75	0.75	0.75	0.78	0.78	1.00	0.62	0.62	0.65	0.70	0.98	1.00
XGBoost	0.75	0.75	0.75	0.80	1.00	0.80	0.69	0.69	0.69	0.71	1.00	1.00
DistDiagnosis-3	0.96	1.00	1.00	1.00	1.00	1.00	0.48	0.88	0.90	0.97	1.00	1.00
DistDiagnosis-4	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.83**	**0.88**	**0.88**	**1.00**	**1.00**	**1.00**

由表 8 中结果可见, 当集群节点数发生变化时, DistDiagnosis 仍然能够保持可观的诊断效果. 在 A1、B1 和 W1 场景中, 直接从三节点集群所迁移的 DistDiagnosis-3 效果能够超越对比方法. 虽然在 C1 场景上的效果略有不足, 然而针对全局所有节点指标设计的诊断方法在集群规模改变时必须再次收集新集群的异常数据并重新训练, 而得益于本方法采用的节点层级诊断器, DistDiagnosis 能够实现直接对新增节点进行诊断, 从而提高了对分布式数据库不同异常场景的泛用性. 同时, 当获取新集群中的异常数据后, 也可对 DistDiagnosis 重新训练以提高准确度. 表 8 中展示的 DistDiagnosis-4 在 4 个场景中都能够取得最好的效果.

6.6 不同规模集群中的诊断效果对比实验

本节展示 DistDiagnosis 在不同规模的数据库集群中的诊断效果, 集群中的节点配置与对比实验中保持一致,

节点数分别为 3、5、7、9. 该实验采用了与第 6.5 节相同的 4 种异常场景. 对于每一种规模的集群, 各诊断方法使用该集群下选定的异常场景数据进行训练与测试. 图 8 展示了各方法的 *AC@k* 指标的变化.

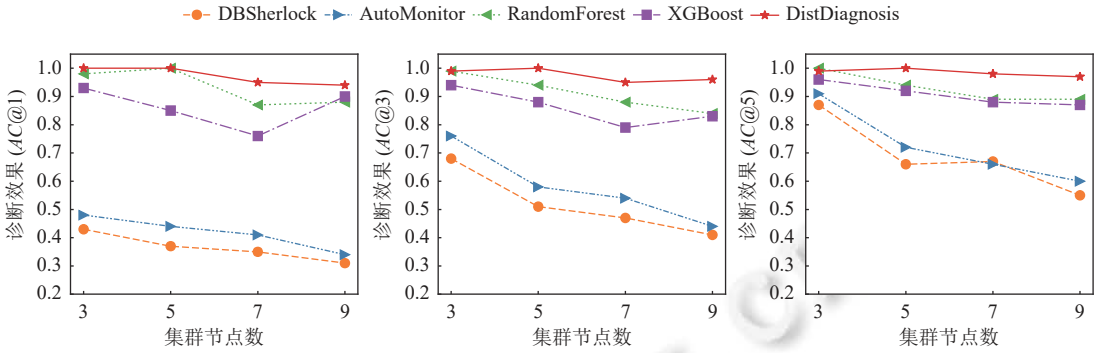

图 8 不同规模集群中的诊断效果对比

由图 8 中结果可见, 在对比方法中, AutoMonitor 与 DBSherlock 的效果最差, 且其诊断效果随着节点数的增加而明显下降. 这是由于随着相应指标个数的不断增加, 基于最近邻或基于空间划分的分类技术容易受输入变量高维的特点所影响, 导致分类效果变差. 而在集群中节点数逐渐增加时, DistDiagnosis 却仍然能够保持最优的诊断效果. 即使在 9 个节点的集群中, 其平均的 *AC@1*、*AC@3*、*AC@5* 能够达到 0.94、0.96、0.97, 与次优的 RandomForest 方法相比, 分别提高了 0.06、0.12 和 0.08. 这体现了 DistDiagnosis 方法在不同规模的数据库集群中都有较好的适应能力.

6.7 案例分析

图 9 展示了当数据库中产生 CPU 瓶颈+负载超常的多节点复合异常时, 各方法输出结果的一个样例. 左图各节点邻边上的权重即对应 DistDiagnosis 所生成的复合异常图边权.

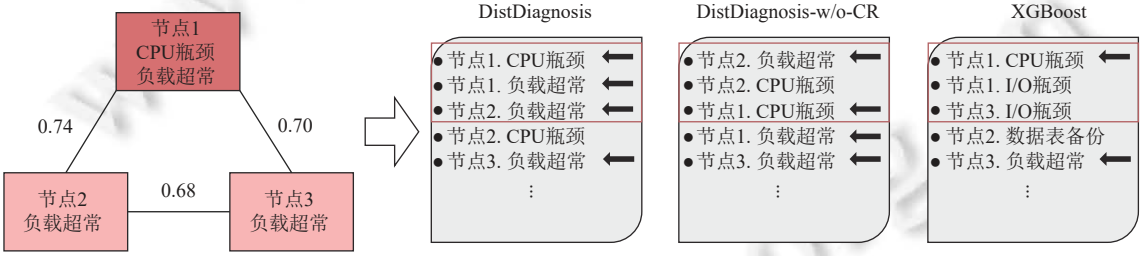

图 9 CPU 瓶颈+负载超常的多节点复合异常时各方法的输出结果

在该异常场景下, 数据库所有节点同时产生负载超常异常. 节点 1 则额外产生 CPU 瓶颈异常, 是在这次异常中较为重要的节点. 由图 9 可见, 此时 DistDiagnosis 为各节点间计算相关性作为边权时, 节点 1 的邻边的权重是图中最高的, 这与其在当前异常中的重要性相对应. 根据节点相关性感知的根因异常排序技术, DistDiagnosis 在进行最终的结果排序时, 能够根据该节点上重要性分数修正排序结果, 从而实现结果序列中前三者都是正确的根因异常. DistDiagnosis-w/o-CR 因为把节点 1 和节点 2 均等对待, 将<节点 2, CPU 瓶颈>这一错误的异常排在更靠前的位置. 在该案例中, XGBoost 则错误地识别了节点 1 的 I/O 瓶颈和节点 3 的 I/O 瓶颈. DistDiagnosis 取得了最好的结果.

7 总结与未来的工作

本文研究了分布式数据库中的异常诊断问题, 提出一种能够对分布式数据库节点上产生的复合异常进行诊断的方法: DistDiagnosis. 该方法采用复合异常图对分布式数据库的异常状态进行建模, 在表示各节点异常的同时,

有效地捕获节点间的相互作用. DistDiagnosis 提出了节点相关性感知的根因异常排序方法, 根据节点对数据库整体的影响力有效地定位根因异常. 为了验证本方法的有效性, 本文在国产分布式数据库 OceanBase 上构建了不同场景的异常测试案例, 实验结果表明, DistDiagnosis 优于其他对比方法, 其异常诊断时, *AC*@1、*AC*@3、*AC*@5 最高达到 0.97、0.98 和 0.98, 相对于次优方法最多提升了 5.20%、5.45% 和 4.46%.

目前 DistDiagnosis 仍存在一定的改进空间. 首先, 本方法针对硬件规格相同的节点才具备最佳的诊断效果, 当各节点资源规格不同时, DistDiagnosis 需要针对不同规格节点训练不同的节点层级诊断器进行处理. 下一步计划设计更为通用的分布式数据库诊断方法, 使 DistDiagnosis 能够更好地支持异构节点及不同规格节点的诊断. 其次, 本方法在进行诊断时采用的监控指标依赖于数据库专家凭借人工经验挑选, 在对指标重要度的判别方面有所不足. DistDiagnosis 下一步考虑设计自动的监控指标选择技术, 并根据指标重要性动态调整其对诊断结果的影响.

References:

[1] Huang SY, Qin YZ, Zhang XY, Tu YF, Li ZL, Cui B. Survey on performance optimization for database systems. Science China Information Sciences, 2023, 66(2): 121102. [doi: 10.1007/s11432-021-3578-6]

[2] Yoon DY, Niu N, Mozafari B. DBSherlock: A performance diagnostic tool for transactI/Onal databases. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco: ACM, 2016. 1599–1614. [doi: 10.1145/2882903.2915218]

[3] Jin LY, Li GL. AI-based database performance diagnosis. Ruan Jian Xue Bao/Journal of Software, 2021, 32(3): 845–858 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6177.htm [doi: 10.13328/j.cnki.jos.006177]

[4] Wang J, Yang YQ, Wang T, Sherratt RS, Zhang JY. Big data service architecture: A survey. Journal of Internet Technology, 2020, 21(2): 393–405.

[5] Zhang GY, Li CH, Zhou K, Liu L, Zhang C, Chen WC, Fang HT, Cheng B, Yang J, Xing JS. DBCatcher: A cloud database online anomaly detection system based on indicator correlation. In: Proc. of the 2023 IEEE Int'l Conf. on Data Engineering (ICDE). Anaheim: IEEE, 2023. 1126–1139. [doi: 10.1109/ICDE55515.2023.00091]

[6] Huang SY, Wang ZW, Zhang XY, Tu YF, Li ZL, Cui B. DBPA: A benchmark for transactional database performance anomalies. Proc. of the ACM on Management of Data, 2023, 1(1): 72. [doi: 10.1145/3588926]

[7] Ma MH, Yin Z, Zhang SL, Wang S, Zheng C, Jiang XH, Hu HW, Luo C, Li YL, Qiu NJ, Li FF, Chen CC, Pei D. Diagnosing root causes of intermittent slow queries in cloud databases. Proc. of the VLDB Endowment, 2020, 13(8): 1176–1189. [doi: 10.14778/3389133.3389136]

[8] Cao W, Gao YS, Lin BC, Feng XJ, Xie Y, Lou X, Wang P. TcpRT: Instrument and diagnostic analysis system for service quality of cloud databases at massive scale in real-time. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 615–627. [doi: 10.1145/3183713.3190659]

[9] Pettitt AN. A non-parametric approach to the change-point problem. Journal of the Royal Statistical Society: Series C (Applied Statistics), 1979, 28(2): 126–135. [doi: 10.2307/2346729]

[10] Dundjerski D, Tomašević M. Automatic database troubleshooting of Azure SQL Databases. IEEE Trans. on Cloud Computing, 2022, 10(3): 1604–1619. [doi: 10.1109/TCC.2020.3007016]

[11] Malhotra P, Ramakrishnan A, Anand G, Vig L, Agarwal P, Shroff G. LSTM-based encoder-decoder for multi-sensor anomaly detection. arXiv:1607.00148, 2016.

[12] Chen TQ, Guestrin C. XGBoost: A scalable tree boosting system. In: Proc. of the 22nd Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 785–794. [doi: 10.1145/2939672.2939785]

[13] Breiman L. Random forests. Machine Learning, 2001, 45(1): 5–32. [doi: 10.1023/A:1010933404324]

[14] Zhou XH, Li GL, Sun ZY, Liu ZY, Chen WZ, Wu JM, Liu JS, Feng RH, Zeng GY. D-bot: Database diagnosis system using large language models. Proc. of the VLDB Endowment, 2024, 17(10): 2514–2527. [doi: 10.14778/3675034.3675043]

[15] Ma M, Xu JM, Wang Y, Chen PF, Zhang ZH, Wang P. AutoMAP: Diagnose your microservice-based Web applications automatically. In: Proc. of the Web Conf. 2020. Taipei: ACM, 2020. 246–258. [doi: 10.1145/3366423.3380111]

[16] Lu RM, Xu EC, Zhang YM, Zhu FY, Zhu ZS, Wang MT, Zhu ZP, Xue GT, Shu JW, Li ML, Wu JS. Perseus: A fail-slow detection framework for cloud storage systems. In: Proc. of the 21st USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2023. 49–63.

[17] Yang ZK, Yang CH, Han FS, Zhuang MQ, Yang B, Yang ZF, Cheng XJ, Zhao YZ, Shi WH, Xi HF, Yu H, Liu B, Pan Y, Yin BX, Chen

JQ, Xu QQ. OceanBase: A 707 million tpmC distributed relational database system. Proc. of the VLDB Endowment, 2022, 15(12): 3385–3397. [doi: 10.14778/3554821.3554830]

[18] Corbett JC, Dean J, Epstein M, *et al*. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems (TOCS), 2013, 31(3): 8. [doi: 10.1145/2491245]

[19] Huang DX, Liu Q, Cui Q, Fang ZH, Ma XY, Xu F, Shen L, Tang L, Zhou YX, Huang ML, Wei W, Liu C, Zhang J, Li JJ, Wu XL, Song LY, Sun RX, Yu SP, Zhao L, Cameron N, Pei LQ, Tang X. TiDB: A Raft-based HTAP database. Proc. of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

[20] Xing W, Ghorbani A. Weighted PageRank algorithm. In: Proc. of the 2nd Annual Conf. on Communication Networks and Services Research. Fredericton: IEEE, 2004. 305–314. [doi: 10.1109/DNSR.2004.1344743]

[21] Liu P, Zhang SL, Sun YQ, Meng Y, Yang JH, Pei D. FluxInfer: Automatic diagnosis of performance anomaly for online database system. In: Proc. of the 39th IEEE Int'l Performance Computing and Communications Conf. (IPCCC). Austin: IEEE, 2020. 1–8. [doi: 10.1109/IPCCC50635.2020.9391550]

[22] Meng Y, Zhang SL, Sun YQ, Zhang RR, Hu ZL, Zhang YY, Jia CY, Wang ZG, Pei D. Localizing failure root causes in a microservice through causality inference. In: Proc. of the 28th IEEE/ACM Int'l Symp. on Quality of Service (IWQoS). Hangzhou: IEEE, 2020. 1–10. [doi: 10.1109/IWQoS49365.2020.9213058]

[23] Kim M, Sumbaly R, Shah S. Root cause detection in a service-oriented architecture. ACM SIGMETRICS Performance Evaluation Review, 2013, 41(1): 93–104. [doi: 10.1145/2494232.2465753]

[24] Benesty J, Chen JD, Huang YT, Cohen I. Pearson correlation coefficient. In: Noise Reduction in Speech Processing. Berlin: Springer, 2009. 1–4. [doi: 10.1007/978-3-642-00296-0_5]

[25] Xu Y, Kostamaa P, Zhou X, Chen L. Handling data skew in parallel joins in shared-nothing systems. In: Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data. Vancouver: ACM, 2008. 1043–1052. [doi: 10.1145/1376616.1376720]

[26] Zilio DC, Jhingran A, Padmanabhan S. Partitioning key selection for a shared-nothing parallel database system. 1994. https://api.semanticscholar.org/CorpusID:14485315

[27] Kumar TVV, Kumar A, Singh R. Distributed query plan generation using particle swarm optimization. Int'l Journal of Swarm Intelligence Research (IJSIR), 2013, 4(3): 58–82. [doi: 10.4018/ijsir.2013070104]

[28] Wang P, Xu JM, Ma M, Lin WL, Pan DS, Wang Y, Chen P. CloudRanger: Root cause identification for cloud native systems. In: Proc. of the 18th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing (CCGRID). Washington: IEEE, 2018. 492–502. [doi: 10.1109/CCGRID.2018.00076]

附中文参考文献:

[3] 金连源, 李国良. 基于人工智能方法的数据库智能诊断. 软件学报, 2021, 32(3): 845–858. http://www.jos.org.cn/1000-9825/6177.htm [doi: 10.13328/j.cnki.jos.006177]

向清风(2000－), 男, 硕士生, CCF 学生会员, 主要研究领域为分布式系统、数据库系统与机器学习的交叉技术.

徐泉清(1980－), 男, 博士, 正高级工程师, CCF 杰出会员, 主要研究领域为数据库系统, 分布式系统.

邵荃侠(1988－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为大规模图数据高效计算与学习, 并行计算框架, 知识图谱分析.

杨传辉(1985－), 男, 硕士, CCF 专业会员, 主要研究领域为分布式系统, 数据库系统.