软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(3):845−858 [doi: 10.13328/j.cnki.jos.006177]

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

基于人工智能方法的数据库智能诊断*

金连源, 李国良

(清华大学 计算机科学与技术系,北京 100084)

通讯作者: 李国良, E-mail: liguoliang@tsinghua.edu.cn

摘 要: 数据库是一种非常重要和基础的计算机系统软件,随着数据库在各行各业的广泛应用,越来越多的人开始关注数据库运行的稳定性.由于各种各样内部或是外部作用的影响,数据库在实际运行的过程中会出现性能异常,而这可能会带来巨大的经济损失.人们大多通过观察监控指标信息来进行数据库异常诊断,但是关于数据库监控指标有数百个,普通的数据库使用者根本无法提取出有价值的信息.一些传统的公司会聘用专业的人员管理数据库,而这种成本会是很多公司难以接受的.因此,如何用较低的成本完成对数据库的自动监控和诊断是具有挑战性的问题.现有的 OLTP 数据库自动异常诊断方法往往存在着监控信息收集成本过高、适用范围小抑或是稳定性较差等问题.提出了一种智能的数据库异常诊断框架 AutoMonitor,提供了数据库异常监测、异常指标提取和根因分析这 3 个模块,这 3 个模块分别使用了基于 LSTM 的时间序列异常诊断模型、Kolmogorov-Smirnov 检验、和优化的 K 近邻算法.整个框架分成离线训练和在线诊断这两个阶段.将提出的系统部署在 PostgreSQL 数据库,通过实验表明该框架对于异常诊断具有较高的精确程度,并且不会对系统性能造成太大的影响.

关键词: OLTP 型数据库;异常诊断;人工智能

中图法分类号: TP311

中文引用格式: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

英文引用格式: Jin LY, Li GL. AI-based database performance diagnosis. Ruan Jian Xue Bao/Journal of Software, 2021,32(3):845−858 (in Chinese). http://www.jos.org.cn/1000-9825/6177.htm

AI-based Database Performance Diagnosis

JIN Lian-Yuan, LI Guo-Liang

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: Database is a kind of important and fundamental computer system software. With the development of database application in all walks of life, a growing number of people begin to concern the stability of the database. Because of the numerous internal of external effect, performance anomaly may emerge when the Database running and it may cause huge economic loss. People usually diagnose database anomaly by analyzing monitoring metrics. However, there are hundreds of metrics in the system and ordinary database users are unable to extract valuable information from them. Some major companies employ DBA to manage the databases but the cost is unacceptable for many other companies. Achieving automatic database monitor and diagnose with low cost is a challenging problem. Current methods have many limitations, including high cost of metrics information collection, narrow range of application or poor stability. This study proposes an anomaly diagnose framework AutoMonitor which is deployed on the PostgreSQL database. The framework contains LSTM-based anomaly detection module and modified K nearest-neighbor algorithm-based root cause diagnose

* 基金项目: 国家自然科学基金(61925205, 61632016)
Foundation item: National Natural Science Foundation of China (61925205, 61632016)
本文由“支撑人工智能的数据管理与分析技术”专刊特约编辑陈雷教授、王宏志教授、童咏昕教授、高宏教授推荐.
收稿时间: 2020-07-19; 修改时间: 2020-09-03; 采用时间: 2020-11-06; jos 在线出版时间: 2021-01-21

module. Framework consists of an offline training and an online diagnose stage. The evaluations on the datasets show that the proposed framework has high diagnose accuracy with minor overload to system performance.

Key words: OLTP database; anomaly diagnosis; artificial intelligence

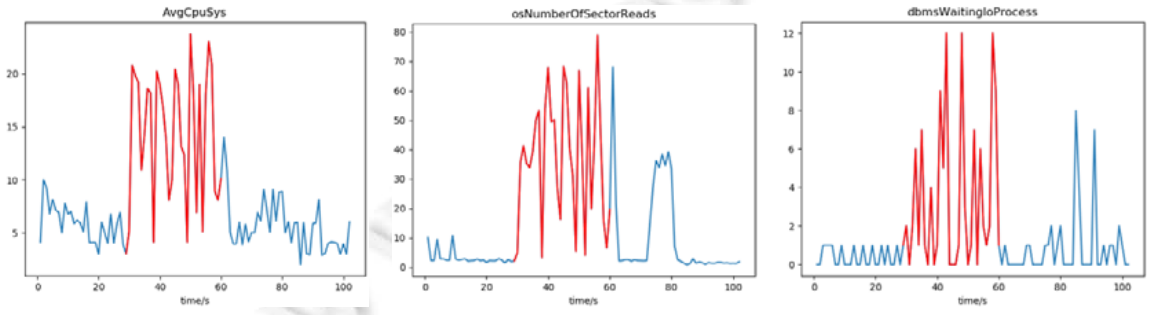
IT 运维,指的是和 IT 服务管理相关的人员及管理过程,IT 运维可以让公司提供的服务保持良好的质量,并且使服务达到用户付款后的预期.运维在人类未来的生产生活中的作用会越来越重要.2017 年,清华大学裴丹提出“预计到 2020 年,全球将有 500 亿到 1 000 亿的 IT 设备,这些设备会承载无数的服务,覆盖互联网、金融、物联网、智能制造、电信、电力网络、政府等等的生产生活的方方面面”(https://yq.aliyun.com/articles/272155).在运维发展的过程中,最先出现的是依靠手工的运维;后来,人们把重复的手工操作代码化,基于大量脚本的自动化运维就开始流行了起来;其后又出现了 DevOps 和智能运维[1].智能运维,指的是使用大数据、机器学习技术来支持 IT 运维.机器学习可以处理海量的监控数据并且提供强大的推断能力.目前已经有许多公司和研究机构使用智能运维技术在许多方向取得了非常显著的进展,包括云数据库服务质量分析[2]、磁盘故障的预测[3]、微服务故障的定位[4]等等.

数据库是各种企业常用的系统软件,根据所执行任务的不同,可以将数据库分为执行大量简单事务的 OLTP 型数据库、执行复杂分析任务 OLAP 型数据库和兼顾执行上述两种任务的 TATP 型数据库.作为一种传统的系统软件,近年来出现了许多利用人工智能技术优化数据库部件的工作[5],可以发现,人工智能技术赋能的数据库具有更加强大的综合能力,智能运维技术在数据库领域也有着广泛的应用场景.

本研究聚焦于智能运维中的 OLTP 环境数据库性能异常监测诊断,在数据库性能下降时发现问题并诊断问题发生的根因.该问题有如下几个难点.

- 监控指标和数据库性能异常之间的对应关系比较复杂,一种根因所引发的异常往往会导致多个监控指标出现问题,而单个监控指标的异常可能由不同的根因所导致的.此外,系统中会存在数百个指标,因此,采用简单的规则来进行数据库异常诊断精确度不高;
- OLTP 环境下的数据库执行的是大量简单的事务,如果监控指标数据成本较大,可能会使执行时间较短的事务变慢,带来数据库性能下降的问题;
- 由于 OLTP 任务请求的不稳定性以及操作系统、数据库系统的复杂性,监控指标中会有许多噪音,在这类监控指标中提取有用信息难度较大.

下面是关于第一个难点的一个具体案例:当我们执行数据库备份操作的时候,我们观察到了很多指标均产生了异常,其中包括了 CPU、I/O、中断计数,数据库等待进程等等,如图 1 所示.



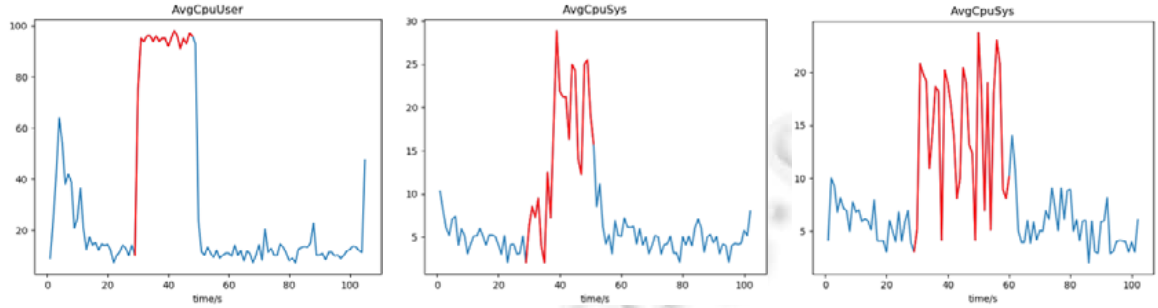
(a) CPU 异常 (b) I/O 异常 (c) 数据库进程异常

Fig.1 Anomaly metrics triggered by database backup

图 1 数据库备份引发的监控指标异常

同一个指标的异常可能是由不同的根因所造成的,我们以 CPU 统计指标举例,可以看到:在外部进程抢占 CPU、数据库备份和数据库外部文件导入这 3 个根因下,CPU 的指标均产生了异常.

图 2 展示了在 3 种异常触发的时候,CPU 指标变化的示意图.这是因为以上 3 个行为均需要计算机提供较高的计算资源.通过可视化我们发现,其他很多的指标均存在着相似的性质.因此,我们很难从单个异常的指标去推定数据库性能异常问题的根因.



(a) CPU 抢占 (b) 数据库备份 (c) 数据库还原

Fig.2 CPU metrics under different anomaly cases

图 2 不同异常下,CPU 监控指标

已有的数据库异常诊断方法并不完全适用于我们的问题,这些方法中有的过于依赖专家知识,需要依靠有经验 DBA 设计模型[6],并且这些模型只适用于一种数据库,不具备迁移性;还有的需要通过修改数据库系统源代码的方式获取细粒度监控信息[7],虽然提高了诊断的精度,但是这样既增加了开销,也带来更大的系统维护成本.

针对以上的问题,本文作者提出了一种自动、轻量级的监控诊断框架 AutoMonitor.通过模拟数据库故障分析挖掘数据库性能故障下的一系列监控指标所表现出来的特征,使用基于深度循环神经网络的模型完成对数据库性能异常的监控.在这里,循环神经网络能以较低的成本和较快的速度完成对时序数据进行建模,并捕捉不同监控指标之间的相关性,进而实现高质量的异常监测.针对监控指标的数据特点,提出了基于 Kolmogorov-Smirnov 检验的异常指标提取算法.该方法利用异常时序数据统计分布的差异来判断是否为异常监控指标,模拟人眼判断异常的过程,并且有一定的鲁棒性.最后,我们使用优化的 K 近邻算法挖掘了异常监控指标和异常根因之间的对应关系,实现了数据库异常根因诊断.相比于其他方法,我们这种基于特征距离比较的方法不仅利用了所有监控指标信息,而且还考虑到了不同监控指标的重要性,因而具有更高的诊断精度.

本文第 1 节是相关工作的介绍.第 2 节是论文总体研究方案的介绍.第 3 节是论文所使用到的算法的具体描述.第 4 节是关于实验结果展示和分析.第 5 节是对于本文的总结.

1 相关工作

国内外有许多关于系统性能诊断的研究工作,针对系统面向不同的任务,我们将这些工作分成面向 OLTP 环境和面向 OLAP 环境这两个大类上面.OLTP 环境下,单条 SQL 比较简单,执行的时间也比较短.因此,面向 OLTP 环境的自诊断方法考虑整体工作负载的性能问题.OLAP 环境下,SQL 语句比较复杂,因此诊断时主要考虑单条 SQL 语句的执行效率问题.

根据自诊断方法的不同,我们可以将自诊断方法分成基于专家结构模型的和基于机器学习的方法:前者利用专家知识构建的结构模型、决策树模型或者知识库进行诊断,后者则是利用历史运行数据结合已经标注的数据进行诊断.在有些文献中,这两种方法都会使用.

1.1 面向OLTP环境

Yoon 等人[8]提出了数据库性能辅助诊断框架 DBSherlock,该框架可以监控数据库运行期间的若干运行指标,当出现性能问题的时候,用户就可以划出性能异常的区域,让 DBSherlock 找出可能出现异常的指标.这些异常的指标具有较强的划分能力,即给定一个关于指标的断言(形如 $Attr>k$),使用这个断言可以把落在正常区域内的点和落在异常区域内的点分开.根据这些异常的指标,用户就可以找出问题发生的原因.DBSherlock 可以将用

户的反馈整合成因果模型,因果模型的原因是数据库性能问题的根因,结果是一些关于异常指标的断言,这些模型会用到未来的推断里去.DBSherlock 还在语义层面对因果模型进行合并,实验表明,这种方法大大提高了模型的诊断能力.

Benoit 等人[6]提出了数据库性能自动诊断框架,和之前工作不同的是,该框架不仅可以对数据库性能问题进行诊断,还将诊断和调优这两个部分利用决策树的模型统一在一起,决策树的非叶节点是条件判断,而决策树的叶节点是需要调优的资源.对于一次关于数据库的诊断,系统会根据数据库当前的状态从根节点出发,根据非叶节点的条件选择子节点进行访问.系统重复以上的步骤,直至到达叶节点,叶节点的资源即是我们需要调优的资源.此外,框架对于数据库的负载和资源分别进行了建模,利用数据库的结构信息提升了准确性,作者还会使用不同资源配置下数据库运行的性能结果调优决策树.不过,该决策树模型的模型需要依赖特定的数据库文档以及数据库专家的领域知识,因此通用性不是很强.

Belknap 等人在文献[7]中介绍了 Oracle 中 SQL 诊断工具 Automatic Database Diagnostic Monitor(ADDM),在这项工作中,作者引入了数据库性能的衡量通货“数据库时间(database time)”,即数据库各个资源模块、数据库语句执行的各个阶段的运行时间,并且在这个基础上建立了图模型 DBTime graph,通过对图的搜索,我们就可以定位问题.针对统计信息收集成本开销过大的问题,文献里提出一种基于采样的方法(active session history),即每隔一段时间观察用户的行为,Database Time 大的模块更加容易被采样到,诊断系统也会相应地重点分析.

Ma 等人[9]针对 OLTP 数据库中间歇性的慢查询,提出了诊断框架 iSQUAD,包含了异常抽取、相关性清楚、基于类型的模式集成聚类以及贝叶斯样例模型这 4 个部分.作者将该方法应用于阿里云数据库的实际数据中,取得了较好的实验效果.

1.2 面向OLAP环境

Borisov 等人[10]提出了诊断工具 DIADS,它适用在以 SAN 为底层存储结构的 DBMS 上面,注释计划图(annotated plan graph)是文献[10]中主要用到的模型,该图分为 Query Layer,Database Layer 和 SAN Layer:Query Layer 的内容是数据库的查询计划(因为是 OLAP 场景,计划较为复杂),Database Layer 则是数据库相关性能运行指标,SAN Layer 则是存储的物理体系结构.通过将查询计划里的操作符和相关的物理、逻辑硬件资源通过图的方式联系在一起,就能在查询出现性能问题时,使用该模型结合历史数据进行诊断.当查询执行时间和预期不符的时候,系统会根据查询计划提取每一个操作符所消耗的时间,将异常的操作符使用机器学习算法提取出来,再根据注释计划图模型找出异常的硬件资源完成诊断.

Kalmegh 等人[11]提出了框架 iQCAR,该框架适用于大数据分析处理系统 Apache Spark 上,主要的功能是分析一个查询在运行的时候受到系统中并发执行的查询的影响.该研究的主要贡献在于定性地分析查询间资源抢占的影响,并且提供了多粒度的分析.该研究首先聚焦于不同的任务关于单个资源的抢占问题,作者在这里提出了阻塞时间(blocked time)这一个概念,即单个工作获取一个特定的资源时的等待时间;此外,作者还提出了资源获取时间惩罚(resource acquire time penalty)来衡量一个任务获取特定的资源的效率.在建立相关概念以后,作者定量地分析了一个查询在获取资源时,其他并发的查询对其阻塞造成的贡献.接着,作者构建了 iQC-graph,这是一个层级结构的图模型,每一层代表着查询执行过程不同的粒度,根据该图分析查询不同执行过程的性能情况.

2 总体研究方案概览

为了实现对数据库运行状况进行监控和诊断,AutoMonitor 需要实时地收集数据库执行任务时相关的统计信息,并分析这些数据的变化情况.我们设计的异常诊断系统总体框架如图 3 所示,设计的框架分为两个主要阶段:第 1 阶段是离线的数据分析,第 2 阶段是在线实时的异常监测和根因诊断.

(1) 关于离线的部分.

系统的主要任务是训练在线阶段所使用的模型,共有两部分训练数据.

- 一部分是异常时间序列数据,这部分数据通过模拟数据库异常收集而来,它是高维的时序数据,每一维

对应一种监控指标的结果,它们被用来确定每一种根因触发的异常下监控指标的特征,系统使用 Kolmogorov-Smirnov 检验给出每一维指标的异常程度,挖掘有异常的监控指标,然后将这些异常指标和正常指标拼接在一起,生成维数固定的异常特征向量,该向量将用于优化的 K 近邻算法中.此外,对于每一种异常的诊断,我们需要确定对应的重要监控指标,每个监控指标的重要性由数值 α_i 决定,越重要的指标,对应的 α_i 越大.相比于普通 K 近邻算法直接使用欧几里得距离进行相似度的计算,优化的 K 近邻算法在相似度计算时会涉及 α_i,使得系统对于根因不同但是表现相似的异常具有更强的判别能力,α_i 的具体计算方法将在第 3.3 节描述;

- 另一部分是数据库正常运行状态下的数据,该部分数据的结构和异常数据相似,区别在于我们没有手动注入异常,它们被用以 LSTM 循环神经网络的训练,训练以后的模型对于正常监控时序数据具有较强的重构能力,即:将一段时序数据依次输入到模型,经过处理保存中间结果在一个隐状态里,模型再从该状态开始逐步重建还原输入的时序数据.系统的最终目标是让输入和输出的时序数据尽可能地相似.当异常时序数据进入模型时,由于正常和异常时序数据在结构上的差异,重构的结果会带来较大的误差,如此,系统便获得异常监测的能力.

(2) 关于在线的部分.

当用户执行工作负载的时候,系统将实时地监控信息输入到数据库中.模型实时从数据库读取这部分时间序列数据并进行异常监测,当监测异常值高于警报阈值(该阈值在离线阶段由历史异常数据和正常数据共同确定,目标是最大化诊断准确度)的时候,便认为此时出现了异常,然后启动根因分析模块进行诊断分析.系统先提取未知异常的异常特征向量,然后使用优化的 K 近邻算法将未知根因的异常特征向量和已知根因的异常特征向量进行相似度比较,从而让用户可以获取异常监控指标和该问题可能的根因.系统会将诊断的结果汇总成报告反馈给用户.

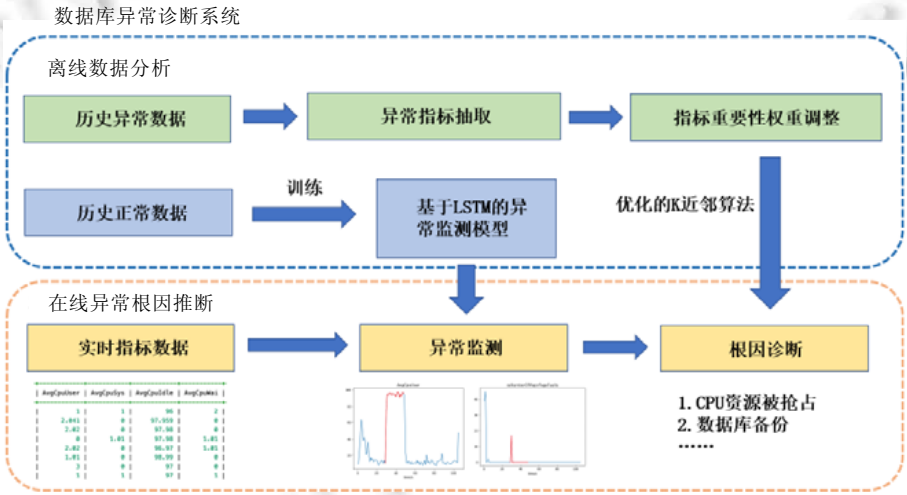


Fig.3 Anomaly diagnosis system architecture

图 3 异常诊断系统结构示意图

AutoMonitor 系统收集的统计信息主要包含了两个部分,分别来自于操作系统和数据库系统,我们使用了不同的方法收集这两部分数据.

- 出于统计信息的可获得性和数据库部署的实际性考虑,我们选定 Linux 作为研究的操作系统.Linux 系统会自动维护系统运行的相关统计数据并存放在特定的目录下,系统使用开源工具 dstat 收集整合这个一部分的信息,其中的监控指标涉及到 CPU、I/O、网络、中断等部件;
- PostgreSQL 内置了多个统计信息视图,记录了数据库、数据表、索引和连接用户等模块的统计信息.

通过对视图构造查询,就可以定时地获取 DBMS 相关的监控数据.我们将构造的监控数据查询写成插件集成到 dstat 的工具里,在 dstat 运行的时候,它会定时读取这一部分的数据并和操作系统的数据相对齐.考虑到 PostgreSQL 内置统计信息的更新频率以及查询开销,在默认情况下,系统选择 1 秒作为信息收集的间隔.

需要注意的是:以上两部分数据是操作系统和数据库自动维护生成的,使用该系统并不需要修改操作系统和数据库,这也保证在添加监控以后数据库的性能不会受太大的影响.

当用户需要使用 AutoMonitor 时,系统首先会启动 dstat,开始监控指标数据的收集.dstat 从系统目录以及 PostgreSQL 视图中读取这部分数据,并将收集到的数据定时写入数据库中,然后自动监测程序定时从数据库读取最新的时序数据,并将这一段时间的数据输入到 LSTM 模型进行异常监测.如果 AutoMonitor 认为此时出现了异常,就调用根因诊断程序,并将这段时序数据传入进行根因诊断.最终的结果会以报告的形式反馈给网页端的用户.

3 算法描述

3.1 异常监测

我们使用基于 LSTM 的编码器-解码器模型来进行针对高维时间序列异常监测[12].给定一段时间监控指标数据的时间序列,系统可以判断它是正常还是异常的.该算法的主要思想是:构造一个编码器-解码器,如图 4 所示,它的输入是一段时间的高维监控指标数据,输出是同样长度、同样维度的时序数据.即:该模型对于一段时间序列按逆序进行重建,根据重建结果的好坏来判断时间序列的异常性.在训练的阶段,我们只使用正常的数据来训练 LSTM 神经网络,网络的损失函数是序列重构结果的误差.如此训练下,LSTM 网络就获得了对于正常序列较好的建模能力,当异常时间序列输入的时候,模型就会产生较大的重构误差.在网络结构部分采用了深层的长短期记忆网络和 Attention 机制[13],以提高模型拟合数据的能力.

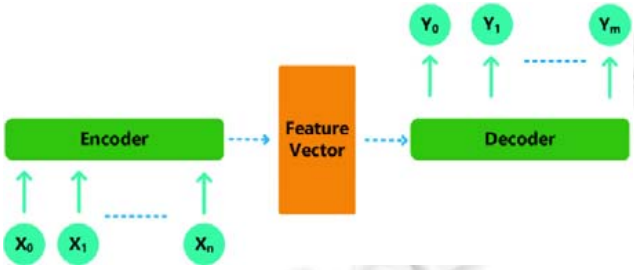


Fig.4 Encoder-decoder structure diagram

图 4 编码器-解码器结构示意图

下面是具体算法的描述.

算法 1. 异常时序数据监测算法.

输入:正常的时间序列 s_N,v_{N1},总训练轮数 *Epoch*;

输出:误差协方差矩阵 Σ,误差向量 μ.

1. 初始化 Initialization;
2. **While** 训练轮数<*Epoch* **do**
3. 使用 s_N,训练 LSTM 编码-解码模型;
4. 使用模型在验证集,v_{N1} 上进行测试;
5. **If** 模型精度不再提升 **then**
6. **break**,结束训练;
7. **else**

8. 训练轮数+=1;
9. **end**
10. **end**
11. 重构误差向量的列表 *error_list*←[·];
12. **for** *sequence*∈v_{N1} **do**
13. 重构序列 *reconstructed_sequence*←*LSTMmodel*(*sequence*);
14. 误差序列 *error_sequence*←|(*reconstructed_sequence*$\tilde{}$*sequence*)|;
15. *error_list.extend*(*error_sequence*);
16. **end**
17. 误差向量μ←*average*(*error_list*);
18. 误差协方差矩阵 $\sum=\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^T(x_i-\mu)$,在这里,*N*=*length*(*error_list*),x_i∈*error_list*;

由于在实际应用的过程中,异常数据的收集较为困难,因此在设定算法的异常阈值中,我们使用了一种基于极值理论(extreme value theory)[14]的方法,该方法大致原理如下:假定异常数据在一个数据集中出现的概率很小,我们设定一个较小的异常概率;然后,结合数据集求解对应的异常阈值.首先,在数据中选取一个分位数(例如0.95);然后选取大于分位数的小部分数据,根据极值理论,这一部分数据经过线性变换以后满足帕累托分布.首先使用极大似然估计的方法求解该帕累托分布的参数,然后根据概率求解异常阈值.具体的公式如下:

$$z_q \simeq t+\frac{\hat{\sigma}}{\hat{\gamma}}\left(\left(\frac{qn}{N_t}\right)^{-\hat{\gamma}}-1\right).$$

在这里,z_q是最后求解的阈值,t是分位数的值,$\hat{\sigma}$和$\hat{\gamma}$是帕累托分布的参数估计结果,q是预先设定异常阈值的概率,n是总的数据集样本数,N_t是分位数下的样本数.由于异常监测涉及到流式的数据,因此在实际应用的过程中,系统会实时的更新帕累托分布的相关参数和异常阈值,进而保证异常监测的实时精确性.

选用该算法有如下的优势.

- LSTM是一种用于有效时间序列分析的神经网络,监控数据可能存在相互关联的情况,而LSTM网络可以较好地捕捉高维数据的关联性;
- 相比于一些针对时间序列异常点的算法,在数据库监控的时候,更加关注区域的异常而不是单点的异常.这种考虑一段时间数据异常性的策略,让监控鲁棒性更强;
- 在设定异常阈值的时候,不需要事先获得异常数据的样本,而是通过设定一个异常发生的概率来进行异常阈值的求解.在实际应用的过程中,用户可以根据业务的实际需求来调整这个概率.另外,这种异常监测的方法可以在流数据的环境下动态地调整阈值,可以适应实际场景下数据的动态偏移.

3.2 监控指标提取

在找到异常区域以后,AutoMonitor需要找出该异常下具有较强特征的指标,即:系统需要模仿DBA自动收集在异常区域下形状特异的监控指标,并且能定量地描述这种异常的程度.

我们设计了基于滑动窗口的Kolmogorov-Smirnov检验方法来解决此问题.Kolmogorov-Smirnov检验是一种统计检验方法,可以用来比较两组数据分布的相似程度.它的一个优点就是:它是一种非参数的检验方法,相比于T检验,不需要提前假设数据的分布,即使少量的点出现偏差,也不会对整体结果产生大的影响.因此,在小样本、噪音较大的数据场景下具有更高的稳定性.

该方法的有效性基于以下两个假设.

- 在发生异常以后,一些特征明显的指标的数据分布会发生较大的改变;
- 在数据库正常运行的状态下,指标点的分布在一段时间里是稳定的,即满足一个未知的分布.

我们在实践中发现,大多数监控指标都满足以上的假设.对于那些呈线性单调递增的指标,我们使用差分的

策略来进行预处理.考虑到根因发生具体触发的时间难以定位,一些指标的记录也会有滞后性,只能在一个时间窗口内处理数据.设想时间窗口内的数据被分成了 3 段:前两段是正常的数据,第三段是异常的数据.系统会对前两段数据进行 Kolmogorov-Smirnov 检验,然后对后两段数据进行 Kolmogorov-Smirnov 检验.检验后,统计量结果的差值即作为该指标异常的评价依据.因为具体的异常发生点是未知的,系统会在一个时间窗口内进行多次计算整合结果.在监控指标提取以后,每一个的数据库异常实例被转化成了一个异常向量 V,向量中的每一维对应了一种指标的异常程度,取值范围是[0,1],如果该值接近为 0,则代表该监控指标没有出现异常.

3.3 根因诊断算法

AutoMonitor 使用最近邻算法作为根因诊断算法,虽然 K 近邻算法比较简单,但是在这个问题下有较好的性能.因为同一种问题发生的时候,异常指标的集合也是比较接近的.为了进一步提高性能,我们设计了基于全局信息异常指标权重调整的策略.

该策略的思路来源于文档搜索,对于文档搜索问题,简单的布尔检索的策略不能取得很好的效果,因为布尔检索将文档里的词语看作一样的.作为布尔检索改进,基于 tf-idf 文档向量模型[15]有着更高的搜索准确度,因为每个词的重要性不一样,一些出现频率较高的词被忽略了.

普通的 K 近邻算法对于每一维的指标不做特殊的处理,作为这个方法的改进,优化后的 K 近邻算法会重新计算每一种标签的异常向量每一维的指标权重.

假设原来使用欧几里得距离 $L=\sqrt{(x_1-x_1')^2+(x_2-x_2')^2+...+(x_n-x_n')^2}$ 计算向量之间的差异,新方法对其中的每一项赋予了特定的权重,在这里,距离的计算变成了:

$$New\_L=\sqrt{\alpha_1(x_1-x_1')^2+\alpha_2(x_2-x_2')^2+...+\alpha_n(x_n-x_n')^2},\sum\alpha_i=1.$$

这里,α的计算是通过分析已知样本所得的.我们希望以下两类指标的权重尽可能大,因为它们在诊断的场景下具有比较重要的价值.

- 该指标在其他根因中出现的较少,但是在本类根因下出现的比较频繁;
- 该指标在其他根因的异常中出现的比较频繁,但是在本类根因下出现的较少.

以上两点是基于数据库运维人员对根因诊断的经验所得出来的,第 1 点的思路是和文档搜索 tf-idf 的思路相似,找出根因中最显著的指标;第 2 点的思路类似于医生对病例的诊断,比方说一个病人的验血结果白细胞正常,我们就不应该认为该病人出现了炎症.

为了使系统根因推断算法可以辨别异常向量相近的不同根因,我们设计了如下的权重调整算法:首先,将每一种的异常根因对应的一系列异常向量聚合成一个向量,代表这种异常根因的特征;然后使用欧几里得距离计算向量两两之间的差异,得到距离矩阵矩阵 ***Dist***.如果两个向量之间的距离较小,将它们之间差异较大的指标提取出来并且放大相应的权重;否则就认为它们之间的距离不会对诊断结果造成影响,进而缩小它们之间差异较大指标的权重.下面是算法的具体伪代码描述.

算法 2. 每类异常指标权值计算策略.

输入:每种异常根因对应的特征向量 $V_1,V_2,\ldots,V_n$,特征向量的维度 k;

输出:每种异常根因对应的权重向量 $A_1,A_2,\ldots,A_n$.

1. 计算特征向量两两之间的距离,得到距离矩阵 ***Dist***;
2. **for** $i\leftarrow 1$ to n **do**
3. 全局加权向量 $G_i\leftarrow\frac{1}{N}\sum_{j=1,j!=i}^{n}\beta_j\times V_j$;
4. 在这里, $\beta_j=\frac{1}{Dist[i,j]^2}, N=\sum_{j=1,j!=i}^{n}\beta_j$;
5. 计算 G_i 和 V_i 的差绝对值 $Residual\_Vec\leftarrow|G_i-V_i|$;

6. $A_i \leftarrow \frac{1}{N} Residual\_Vec, N = \| Residual\_Vec \|_1$;

7. **end**;

值得注意的是:AutoMonitor 使用的是基于最近邻的方法,因此当我们引入新的异常类型以后,我们只需要将这些新类型的异常向量添加进训练集中,然后重新的计算指标权重.由于总的数据集不大,重新计算只需要很短的时间,模型的推断速度也比较快.由此我们认为:在添加新的数据或者新类型的异常,AutoMonitor 仍然可以有效地工作.

4 实验结果分析

在实验部分,我们需要验证以下 3 个问题.

- AutoMonitor 是否会对数据库实际运行造成较大的影响?
- AutoMonitor 的异常监测模块是否能快速部署并达到较高的监测精度?
- AutoMonitor 的根因分析模块相较于同类方法是否能有较高的诊断准确率?

4.1 环境配置

我们使用两台配置相同的阿里云服务器作为实验的环境,服务器具体性能参数如下.

- 操作系统:Ubuntu 16.04 64 位;
- CPU:单核 Intel(R) Xeno(R) Platinum 8163 CPU@2.5GHz;
- 内存:2G.

服务器安装的 PostgreSQL 是 12.0 版本,使用了 python2.7/python3.6 两种解释器.实验使用 TPC-C[16]作为实验的 Benchmark.表 1 是在和华为高斯数据库 DBA 交流讨论后,经过归纳整理的几种常见的数据库故障的问题根因,其中的设计参考了文献[8].这里的问题根因涉及到了数据库性能异常的各个方面,既包括了数据库外部进程对 CPU、网络等资源的抢占,也涵盖了数据库日常运维的一系列事件,还有一部分访问数据库的错误行为.

Table 1 PostgreSQL common anomaly root causes

表 1 PostgreSQL 数据库常见的问题根因

编号	异常类型	具体描述
1	CPU 瓶颈	数据库外部的进程占据了较多的 CPU 计算资源
2	I/O 瓶颈	数据库外部的进程占据了较多的 I/O 资源
3	数据表备份	启动 PG_DUMP 备份数据库的部分表
4	数据表还原	启动 PGSQL 将外部的数据表导入到数据库
5	错误的物理设计	在一些数据列上构造不必要的索引,当 INSERT 操作过多时,会造成不必要的开销
6	开销巨大的查询	在多张数据表之间 JOIN 的查询,一般这种查询会被 OLTP DBA 所禁止
7	负载超常	用户和事务请求突然变大,这会造成数据库的阻塞效应
8	网络阻塞	数据库所在服务器的网络出现了故障
9	锁竞争	由于各个用户频繁地访问相同的数据库区域,导致锁的竞争
10	统计信息更新	使用 VACUUMANALYZE 更新 PostgreSQL 数据库中的统计信息

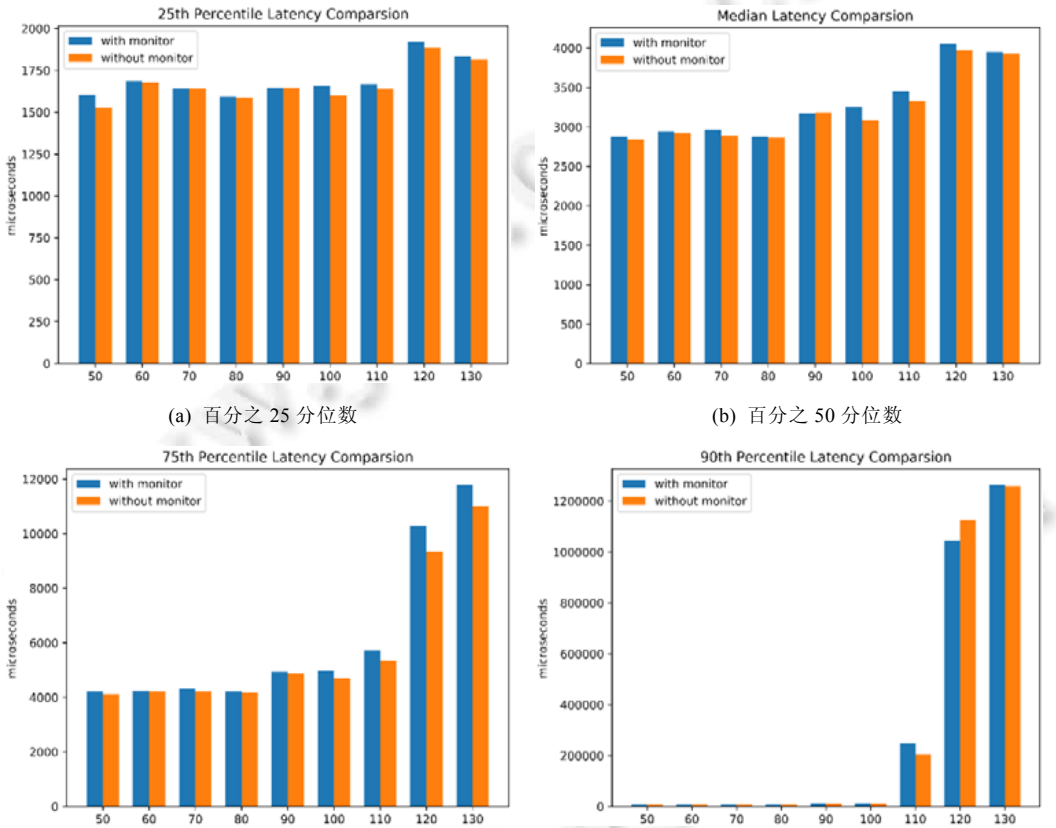
实验中使用 OLTP-Bench[16]工具实现了 TPC-C 基准,该工具可以生成不同请求频率和不同持续时间的 TPC-C 负载,还能生成导入不同规模的表数据.由于真实环境下数据库的运行数据难以获得,以上这些异常案例数据我们通过手动模拟触发异常的方法来产生,即两台服务器,一台作为数据库服务器,另一台作为用户.每一次实验里,用户先启动 OLTP-Bench 执行数据库负载,在实验中途启动服务器中的异常程序触发数据库性能异常,等到异常程序结束再让数据库正常运行一段时间,然后结束单次实验.下面介绍实验的具体细节:对于 CPU、I/O 瓶颈,实验使用 Linux 环境下开源工具 stress-ng 来抢占 CPU 以及 I/O 的资源;对于数据表备份以及数据表还原,实验直接使用 PostgreSQL 中关于数据导入/导出的命令(pg_dump,psql);对于错误的物理设计、开销巨大的查询、负载超常和统计信息更新这 4 类异常,实验直接使用描述中的方法来产生;对于网络故障,我们使用 Linux 系统内置命令 tc 来进行流量控制,手动让网络通信速度变慢;对于锁竞争,我们使用 OLTP-Bench 对 TPC-C 数据

表中的一小部分数据进行短时间大量的访问;考虑到数据库访问请求的多样性,对于每一组生成的数据,我们设定了不一样的访问频率(每秒事务请求的数目从 30~150).

4.2 监控工具性能开销测试

本实验旨在探究数据库根因诊断系统监控工具对 OLTP 数据库性能的影响.

我们选取了不同的请求频率,并且考虑到单次实验的不稳定性,在每一种请求频率下重复进行了 20 次实验.考虑到实验中平均数的不稳定性,本实验使用分位数作为分析性能差异的评价指标,图 5 是该实验的结果.



(a) 百分之 25 分位数 (b) 百分之 50 分位数

(c) 百分之 75 分位数 (d) 百分之 90 分位数

Fig.5 Database performance comparsion with/without monitor tool

图 5 有无监控工具下数据库性能比较结果

通过观察可以发现:开启了监控工具性能的损失相对而言是非常小的,分位数的平均延迟增加一般不会超过 5%.原因主要有以下两点.

(1) 当前监控数据请求的频率为 1 秒 1 次,这个用户对数据库的访问频率而言是比较短的.在实践中,1 秒 1 次的数据请求频率也足以保证推断的精确程度,如果根据实际需要减少访问频率,性能的损失可以进一步降低;

(2) 我们收集的数据包含了数据库和操作系统,以上两部分的数据均会被操作系统和数据库相关的进程定期生成.换句话说,系统并没有增加数据收集上的开销,增加的开销仅仅来源于访问获取数据,CPU 和 I/O 增加的额外开销都比较小.

综上所述,我们可以确信:当前的数据库监控诊断框架是轻量的,具有较强的实用价值.

4.3 异常检测模块测试

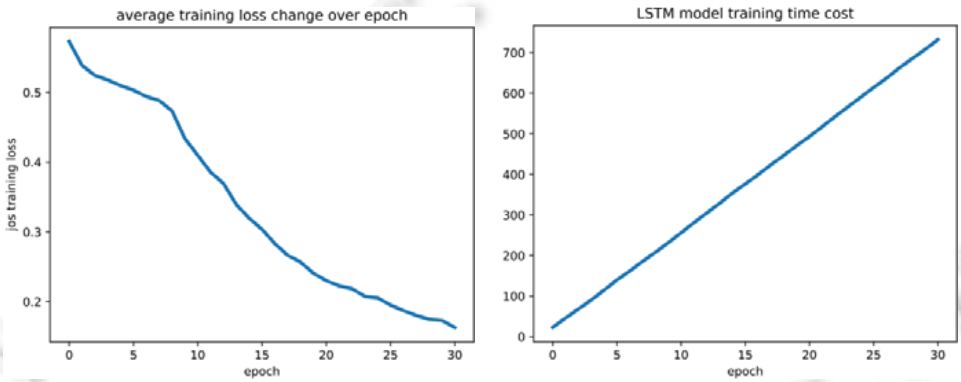
本实验旨在验证数据库根因诊断系统异常监测模块的有效性,总共选取了 8 000 条长度为 32 的正常时间序列作为模型的训练数据.关于神经网络的一些超参数见表 2.

Table 2 RNN model hyper parameters

表 2 循环神经网络模型超参数

超参数类型	数值
RNN 堆叠层数	2
学习率	0.01
隐状态维数	80
批训练大小	32

自编码器模型于 GPU 服务器训练,图 6 是神经网络训练结果.可以观察到:该神经网络训练的开销是比较小的,网络也较快地达到收敛的状态.在实际部署的时候,系统会实时调用最近的历史数据来更新神经网络的模型参数,较小的训练开销保证了模型的可用性.



(a) 神经网络训练中损失变化 (b) 神经网络训练耗时

Fig.6 Neural network training result

图 6 神经网络训练结果

实验选取了基于 VAE[17]和基于 GAN[18]的时序数据异常监测方法和 AutoMonitor 方法进行了对比.在测试阶段,我们选取了 200 组异常序列数据和 500 组正常序列数据,图 7 和表 3 是异常监测模块灵敏度的实验结果.

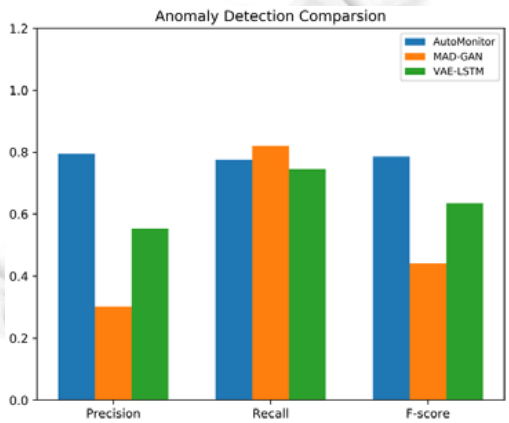


Fig.7 Anomaly detection module experimental results

图 7 异常监测模块实验结果

Table 3 Anomaly detection performance comparison

表 3 各算法性能的对比结果

评价指标\使用方法	AutoMonitor	MAD-GAN	VAE-LSTM
Precision	**0.795**	0.302	0.554
Recall	0.776	**0.821**	0.745
F-score	**0.785**	0.442	0.635

可以看到:我们的监测模型同时达到较高的精度和召回率.对比结果表明,AutoMonitor 方法是最优的.我们认为,原因有如下几点.

(1) 已有的基于生成模型的方法虽然可以在一些时序数据异常检测的任务中取得比较好的效果,但是因为时序数据的模式各异,数据库的监控数据又和负载相关,因此,生成模型很难去学习到隐变量参数;

(2) 复杂的深度生成模型往往有较多的参数,如果未经充分的调优会难以学习,从而造成性能的下降;

(3) 对于一个偏向于工程的任务,我们认为:在选择模型算法的时候不应该只追求精度,同样要考虑到模型的稳定性和训练速度.

4.4 根因分析模块测试

本实验旨在验证 AutoMonitor 中根因分析模块的有效性.对于每一种异常设置,在不同的用户请求频率下,各生成 25 组数据(总共为 250 组数据).这 250 组实验数据被随机划分成了训练集(80 组)和测试集(170 组),我们的方法将会和 DBSherlock[8]、普通的 K 近邻算法和决策树算法进行对比.

图 8 是实验总体的精度对比,可以看到:AutoMonitor 的表现比其他 3 种算法要出色;权重调整的 K 近邻算法要比普通 K 近邻算法精度提升 10 个百分点,比决策树算法精度高近 5 个百分点.可以看到:DBSherlock 在实验数据集上的表现并不出色,只有不到 60%的诊断精度.这说明在比较复杂的环境下(每组数据用户的请求频率不同),DBSherlock 并不能起到很好的效果.

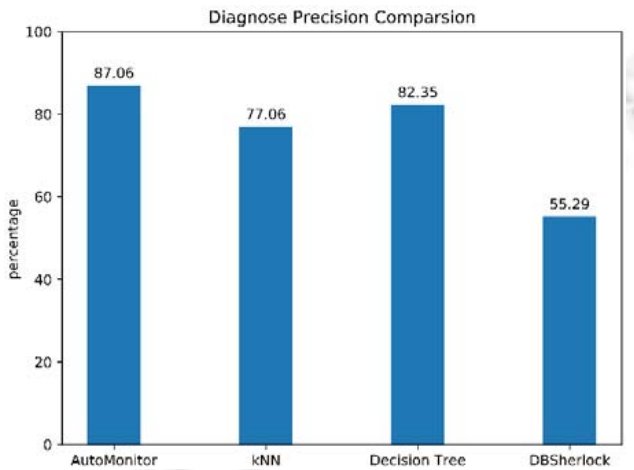


Fig.8 Comparison on diagnosis precision of different diagnosis methods

图 8 根因分析各方法精度对比

图 9 是关于 3 种分类器对于每一种异常的识别能力比较,我们使用 F-Score 作为评价标准.从实验结果图中可以看到:AutoMonitor 在各类异常的根因诊断上均有较为出色的表现,并不存在对于某种异常根因诊断能力较弱的情况.而 DBSherlock 方法对于某几类异常的诊断精度较高,但是对于另外几类异常的诊断精度确不尽如人意.主要的问题在于:DBSherlock 生成的断言对于数据的敏感度较高,并且合并因果模型的策略也存在着较大的不确定性.表 4 是对于每一种根因诊断的具体指标结果,包含了精确度(第 1 行)和召回率(第 2 行).

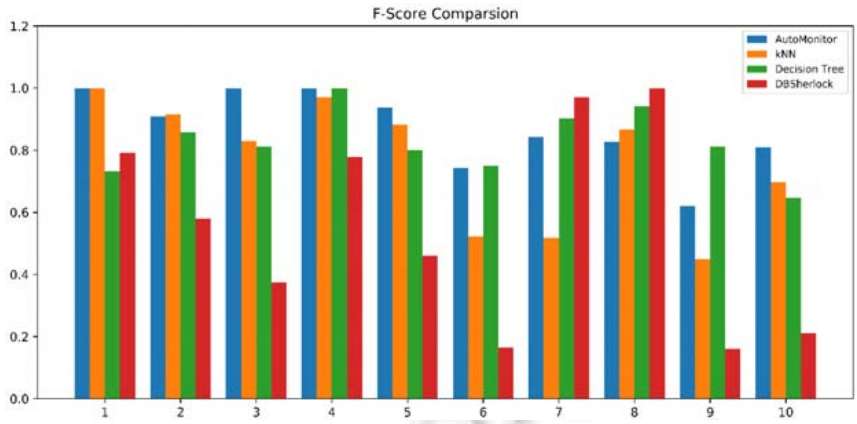


Fig.9 Comparisionon *F-score* of different diagnosis methods

图 9 不同算法 *F-score* 比较结果

Table 4 Precision/Recall comparison result

表 4 各算法精确度/召回率的对比结果

异常类型\使用方法	AutoMonitor	*k*NN	Decision tree	DBSherlock
CPU 瓶颈	1.0/1.0	1.0/1.0	0.625/0.88	0.65/1.0
I/O 瓶颈	0.94/0.88	0.89/0.94	0.83/0.88	0.64/0.53
数据表备份	1.0/1.0	0.71/1.0	0.87/0.76	0.40/0.35
数据表还原	1.0/1.0	1.0/0.94	1.0/1.0	0.74/0.82
错误的物理设计	1,0/0.88	0.88/0.88	0.78/0.82	0.67/0.35
开销巨大的查询	0.72/0.76	1.0/0.35	0.80/0.71	0.11/0.30
负载超常	0.76/0.94	0.7/0.41	1.0/0.82	1.0/0.94
网络阻塞	1.0/0.71	1.0/0.76	0.94/0.94	1.0/1.0
锁竞争	0.75/0.53	0.39/0.52	0.87/0.76	0.25/0.11
统计信息更新	0.68/1.0	0.58/0.88	0.65/0.65	1.0/0.11

5 总结与未来工作

本文研究了 OLTP 数据库在实际运行时可能遇到的异常,分析了这些异常和一系列监控指标之间的影响关系,以及对这些异常的监测和根因推断方法.

针对上面的问题,本文设计了一个部署在 PostgreSQL 数据库管理系统上的自动监控诊断框架,该框架造成的额外开销较小,能为经验较少的数据库使用者提供自动异常监测诊断的服务.在根因推断上面,本文使用了一种权重调整的 *K* 近邻算法.和普通的 *K* 近邻算法相比,这种 *K* 近邻算法可以更好地捕捉到不同根因之间的差异关系,并且可以提取关于某种根因相对重要的监控指标.

本文的实验部分探究了自动异常监测模块和根因推断模块的准确性,结果表明:上述的两个模块准确性都较高,经过权重调整的 *K* 近邻算法对于表现比较接近的问题根因具有更加好的辨别区分能力.总体来说,我们的系统具有实际的应用价值.

该系统目前的不足之处在于只适用于普通的单机 PostgreSQL 数据库,扩展性较差,并且诊断系统在离线的过程中需要比较多的训练数据.下一步计划将框架移植到 MySQL 等主流开源数据库上,并且考虑将框架扩展到分布式数据库、云数据库上.此外,目前的异常数据主要是人工合成模拟的,我们计划通过和数据库主流厂商合作的形式,研究更加实际的场景.

References:

[1] Dang Y, Lin Q, Huang P. AIOps: Real-world challenges and research innovations. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Companion (ICSE-Companion 2019). IEEE, 2019: 4–5.

[2] Cao W, Gao Y, Lin B, Feng X, 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 SIGMOD. 2018.

[3] Xu Y, Sui K, Yao R, Zhang H, Lin Q, Dang Y, Li P, Jiang K, Zhang W, Lou JG, Chintalapati M, Zhang D. Improving service availability of cloud systems by predicting disk error. In: Proc. of the UsenixAtc. 2018. 481−494.

[4] Zhou X, Peng X, Xie T, Sun J, Ji C, Liu D, Xiang Q, He C. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proc. of the 2019 27th ACM Joint Meeting European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE 2019). 2019. 683−694.

[5] Li GL, Zhou XH, Sun J, Yu X, Yuan HT, Liu JB, Han Y. A survey of machine-learning-based database techniques. Chinese Journal of Computers, 2019 (in Chinese with English abstract). http://kns.cnki.net/kcms/detail/11.1826.TP.20191104.1009.002.htm

[6] Benoit DG. Automatic diagnosis of performance problems in database management systems. In: Proc. of the 2nd Int'l Conf. on Autonomic Computing (ICAC). 2005. 326−327.

[7] Dias K, Ramacher M, Shaft U, Venkataramani V, Wood G. Automatic performance diagnosis and tuning in oracle. In: Proc. of the CIDR. 2005.

[8] Yoon DY, Niu N, Mozafari B. DBSherlock: A performance diagnostic tool for transactional databases. In: Proc. of the SIGMOD. 2016.

[9] Ma M, Yin Z, Zhang S, Wang S, Zheng C, Jiang X. Diagnosing root causes of intermittent slow queries in cloud databases. In: Proc. of the PVLDB Endowment. 2020. 1176−1189.

[10] Borisov N, Uttamchandani S, Routray R, Singh A. Why did my query slow down? In: Proc. of the CIDR. 2009.

[11] Kalmegh P, Babu S, Roy S. IQCar: Inter-query contention analyzer for data analytics frameworks. In: Proc. of the SIGMOD. 2019.

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

[13] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proc. of the Computer Ence. 2014.

[14] Siffer A, Fouque PA, Temier A. Anomaly detection in streams with extreme value theory. In: Proc. of the ACM SIGKDD. 2017.

[15] Ramos J. Using tf-idf to determine word relevance in document queries. In: Proc. of the 1st Int'l Conf. on Machine Learning, Vol.242. 2003. 133−142.

[16] Difallah DE, Pavlo A, Curino C, CudreMauroux P. OLTP-bench: An extensible testbed for benchmarking relational databases. Proc. of the VLDB Endowment, 2013,7(4):277−288.

[17] Lin S, Clark R, Birke R, Schonborn S, Roberts S. Anomaly detection for time series using VAE-LSTM hybrid model. In: Proc. of the 2020 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP 2020). IEEE, 2020.

[18] Li D, Chen D, Shi L, *et al*. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. arXiv: 1901.04997. 2019.

附中文参考文献:

[5] 李国良,周煊赫,孙佶,余翔,袁海涛,刘佳斌,韩越.基于机器学习的数据库技术综述.计算机学报,2019. http://kns.cnki.net/kcms/detail/11.1826.TP.20191104.1009.002.html

金连源(1997－),男,博士生,主要研究领域为数据库与机器学习的交叉技术.

李国良(1981－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为数据库,大数据分析和挖掘,群体计算.