Computer Science and Application 计算机科学与应用, 2019, 9(12), 2266-2276
Published Online December 2019 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2019.912252

Hans汉斯

Anomaly Detection of Large Scale Microservice Architecture Software System Based on Log Parsing

Liyuan Tai[1], Chunqi Tian[1], Wei Wang[2]

[1]Department of Computer Science and Engineering, Tongji University, Shanghai
[2]School of Data Science, East China Normal University, Shanghai
Email: tianchunqi@tongji.edu.cn, lytai2017@tongji.edu.cn

Received: Nov. 14th, 2019; accepted: Nov. 29th, 2019; published: Dec. 5th, 2019

Abstract

In recent years, with the rise of microservice architecture, the scale of the system is becoming larger and larger. The traditional manual positioning problems and anomaly methods are inefficient and time and energy are consumed. How to carry out automatic anomaly detection has attracted extensive attention of researchers. It is an effective means to carry out anomaly detection through logs. Due to the complexity of microservice architecture software system business, the amount of log data generated is huge, and these logs are unstructured logs from different cluster nodes and different user requests, with various types and complex formats, so it is difficult to extract useful log information for anomaly detection. This paper proposes an anomaly detection method that analyzes log source code through an abstract syntax tree, converts unstructured log data into structured data, and then groups the structured logs according to time windows and event identifiers. Long and short term memory networks are modeled to detect abnormal execution paths in the system. The experiment shows that it can effectively detect the anomalies in the microservice architecture software system, and the accuracy of the model is improved by about 10% compared with the traditional statistical method. At the same time, we also study the effect of the length of the log key sequence and the size of the training data set on the anomaly detection model.

Keywords

Log Parsing, Exception Detection, Microservice, Abstract Syntax Tree, Long Short Term Memory Network

基于日志解析的大规模微服务架构软件系统异常检测

邰丽媛[1]，田春岐[1]，王　伟[2]

文章引用: 邰丽媛, 田春岐, 王伟. 基于日志解析的大规模微服务架构软件系统异常检测[J]. 计算机科学与应用, 2019, 9(12): 2266-2276. DOI: 10.12677/csa.2019.912252

[1]同济大学计算机科学与技术系，上海
[2]华东师范大学数据科学学院，上海
Email: tianchunqi@tongji.edu.cn, lytai2017@tongji.edu.cn

收稿日期：2019年11月14日；录用日期：2019年11月29日；发布日期：2019年12月5日

摘　要

近几年随着微服务架构的兴起，系统规模越来越庞大，传统的人工定位问题和异常的方法效率低，耗费时间和精力，如何进行自动化的异常检测引起了科研人员的广泛关注，通过日志进行异常检测不失为一种有效的手段。由于微服务架构的软件系统业务复杂，产生的日志数据量庞大，而且这些日志是来自不同集群节点、不同用户请求的非结构化日志，类型多样，格式复杂，因此难以提取有用的日志信息进行异常检测。作者提出了一种通过抽象语法树进行日志源码解析，将非结构化的日志数据转为结构化的数据，再按照时间窗口和事件标识符对结构化的日志进行分组，最后通过长短期记忆网络建立模型以检测系统中的执行路径异常，实验表明，可以有效地检测微服务架构软件系统中的异常，相较于传统的基于统计方法的模型准确率提高了约10%，同时还研究了日志键序列长度和训练数据集大小对异常检测模型效果的影响。

关键词

日志解析，异常检测，微服务，抽象语法树，长短期记忆网络

1. 引言

近几年随着互联网用户量的剧增，一台机器已经远远满足不了系统的负载，不得不考虑系统的水平扩展，开发模式也逐渐从传统的单体应用模式转变成现代的敏捷式开发、微服务架构模式。微服务是一种架构思想，微服务倡导在传统的单体应用架构的基础上，将一个大型的软件系统按照功能拆分为更加细粒度的服务，每个小服务独立运行和部署，各个服务之间是松耦合的[1] [2] [3]，服务间采用轻量级的通信机制，每个服务可以由不同的团队开发和维护，采用不同的语言和存储。然而，微服务架构的软件系统通常规模庞大，在设计、实现和部署方面都较为复杂，并且行为上受到运行时所处的网络环境的影响，微服务应用之间互相通信，导致开发和维护人员很难准确地掌握整个系统的运行和各个组件之间的关联，难以进行故障诊断和修复。

当系统运行出现问题的时候，可以通过日志推断问题出现的原因和位置，分析应用产生的日志通常能有效地解决问题[4] [5] [6]。微服务架构的软件系统的日志包含了丰富的信息，不仅可以辅助开发人员和测试人员理解软件系统复杂的执行行为、区分用户需求，而且可以帮助他们进行异常检测和故障诊断。但是，目前利用日志进行微服务软件系统的异常检测的效率并不高，主要面临以下三个问题：1) 微服务架构软件系统每天需要处理大量的用户需求，产生的日志数量庞大；2) 这些日志来自于不同的业务功能模块，类型多样，格式复杂，大部分是非结构化日志，如图 1 所示，分别是来自微服务系统中 Nginx 服

务器和 Redis 缓存服务的日志，格式不同，难以提取有效的信息；3) 这些日志分散在不同的机器节点上的文件中，微服务进程执行的日志和不同用户请求产生的日志相互交错，难以区分和提取有用的日志[7]。

Figure 1. Heterogeneous log of microservice architecture software system
图 1. 微服务架构软件系统的异构日志

为了解决以上三个问题，本文针对大规模微服务架构软件系统中的异常检测做了如下工作：首先对日志数据集进行预处理，通过抽象语法树进行源代码解析，将非结构化的数据转成结构化的数据，在进行日志划分之后通过长短期记忆网络(Long Short Term Memory network, LSTM)建立异常检测模型，并在微服务架构中的系统日志中得到了良好的检测效果，整体流程图如图 2 所示。

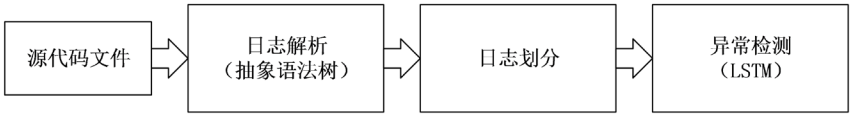

Figure 2. Overall flow chart of exception detection based on log analysis
图 2. 基于日志分析的异常检测整体流程图

本文文章结构如下：第二章对以往基于日志进行异常检测的文献进行概述，第三章详细介绍我们的异常检测模型，包括对日志数据的预处理：日志解析和日志划分方法，第四章介绍我们的实验数据和实验结果，并对实验结果进行评估，第五章总结我们提出的方法并提出未来可持续研究的方向。

2. 相关工作

基于日志的异常检测，由于其能够适应更复杂的环境，更好地追踪定位故障，近年来受到越来越广泛的关注。

基于日志的异常检测方法，以对原始日志文件中的非结构化数据的解析为基础。日志文件记录了系统的运行实况，在进行异常检测之前，我们首先要从日志中提取出事件，形成一个时间序列，传统上有三类日志解析的方法：基于聚类算法的日志解析方法，基于启发式算法的日志解析方法，基于二进制代码或源代码分析的日志解析方法。

文献[8]提出了 LogSig 算法，通过生成消息签名从文本日志消息中生成系统事件，他们将日志语句中的子序列作为事件类型的签名，LogSig 算法通过搜索最有代表性的消息签名，将日志消息按照事件类型分成不同的组。Fu.等人[9]设计了一种非结构化的日志解析算法，将日志转化为特定的关键字格式，然后从这些日志关键字序列中简历有限状态自动机模型来表示正常的工作流，通过定义执行模式的概念来反映系统的运行行为和执行路径，为异常诊断提供帮助。传统的基于监督的日志解析局限于用户所寻求并且关注的已知错误，并且不能容易地适应新的数据源和格式，此外，需要大量的计算。

LangHead [10]通过更高级的聚类算法和关联规则，从信息检索中提取评分，来提取日志的消息模版。IPLOM [11]通过一系列的启发式方法来捕获相似日志消息的差异，从而确定日志的类型。启发式算法比较简单，易于实现，但是无法处理格式复杂的日志。

Zhao X，Kc K [12] [13]等人基于源代码分析日志的文法结构，形成正则表达式。文献[14]通过分析原始日志并结构化来发现应用程序日志中的模式，它从一组表示系统正常运行的日志中发现一组 GROK 模

式，然后使用这些 GROK 模式来解析日志。

此外，还可以利用特定的领域知识进行日志解析[15] [16] [17]，这种方法结果通常比较准确，但是需要分析人员对系统或者代码有充分的理解，才能写出有效的脚本，而且，手工分析和训练日志所需要的人力资源也比较大，效率低。

异常检测通常分为监督方法和无监督方法两大类，常见的监督方法有逻辑回归、决策树[18]和支持向量机[19]。标记训练数据是监督异常检测的前提，它通过标记来指示正常或异常状态。训练数据的标签越多，模型就越精确，建立模型需要大量的人工努力。有相当一部分文献针对非监督的异常检测做出了贡献。

文献[20]提出了一种名为 LogCollect 的方法用于在线的异常检测。LogCluster 需要两个培训阶段，即知识库初始化阶段和在线学习阶段，在知识库初始化阶段将正常和异常事件计数向量分别聚类，然后分别生成正常聚类和异常聚类作为知识库，在线学习阶段进一步调整知识库初始化阶段构建的集群。检测异常时根据新日志序列到知识库中代表性向量的距离判断其是否为异常。日志聚类使用在线学习的思想，因此，它适合处理大量日志数据。Wei Xu [21]等人通过 PCA 算法进行异常检测，基本思想是将高维数据投影到由 k 个主分量组成的新坐标系中，PCA 通过寻找捕捉高维数据中最大方差的分量来计算 k 个主分量。在异常检测中，每个日志序列被矢量化为事件计数向量。之后，PCA 被用来寻找事件计数向量维度之间的模式。文献[22]从控制台日志中挖掘不变量关系进行异常检测，挖掘不变量可以揭示代表系统正常执行行为的多个日志事件之间的线性关系。不变量挖掘不仅可以高精度地检测异常，而且可以为每个检测到的异常提供有意义和直观的解释。然而，不变量挖掘过程非常耗时。B Debnath [14]等人提出了一种名为 LogLens 的实时日志分析系统，该日志分析系统自动从没有目标系统知识和用户规范的日志中检测异常。LogLens 是一个通用的日志分析系统，可以从任何软件系统日志中自动检测异常，它分为两个阶段，首先捕获事件的正常行为建立模型，然后再使用自动机模型来分析实时生产的日志从而检测异常。

3. 异常检测模型

3.1. 微服务系统日志数据预处理

微服务架构下的日志通常是由不同的应用程序和独立部署的服务生成的，日志数据来源广、数据量大，因此日志数据具有异构性，表现为类型多样，结构复杂，日志解析的工作就是将非结构化的日志转化为结构化的数据。日志解析的目的是从日志源文件中挖掘有用的信息，从而作为日志特征提取的输入。

基于聚类算法的日志解析的精度和效率受到日志相似度度量公式的限制，适用于离线的日志解析。本文基于源代码进行日志解析，对于源码可见或者二进制码未经加固的微服务系统，在解析精度上更具有优势，并且适用于在线的日志解析。

3.1.1. 微服务系统日志解析

微服务应用中集成了 Log4j 框架打印日志，通过配置文件，我们可以控制每一条日志的输出格式。如图 3 所示的日志，可以看到日志中包含时间戳、日志等级、微服务名称和主机号等，通过分析源代码信息可以更容易发现日志的模式，我们将常量部分称为日志消息类型，将变量部分称为日志消息变量。

```
2019-08-13 03:08:59.120  INFO [kfcoding-gateway,bcad7d62bc86d624,bcad7d62bc86d624,false]
1 --- [ctor-http-nio-4] c.c.k.gate.filter.AccessGatewayFilter    : check token and user permission....
2019-08-13 03:09:59.128  INFO [kfcoding-gateway,513fdb15dcb3100f,513fdb15dcb3100f,false]
1 --- [ctor-http-nio-1] c.c.k.gate.filter.AccessGatewayFilter    : check token and user permission....
```

Figure 3. Log of service gateway in the system
图 3. 系统服务网关的日志

日志消息变量包含了每条日志的重要信息，有大量的研究工作专注于研究消息变量中的数值变量，实际上日志消息中还包含了一些非数值变量的有效信息，比如日志的级别：FATAL、ERROR、WARN、INFO、DEBUG、TRACE，不同的级别具有不同的优先级，代表不同的意义。INFO 级别的日志可以帮助测试人员判断这是否是一个真正的 bug，而不是自己误操作造成的；ERROR 级别则是明确告诉开发人员系统出现了错误，需要处理。

微服务架构中有多个独立部署的应用，来自不同的系统和应用的日志都有各自定义的格式，可能对于某些日志消息类型处于缺省状态。在面向对象的语言中，无法通过一个正则表达式来对所有的日志消息进行模版化，因为并不是所有的日志都通过拼接字符串或者正则表达式来输出，如图 4 所示，打印日志的语句只有常量，日志是通过方法调用生成的。

```
public void A(){
   ...
   log.info(B.getInfo());
}

public class B{
  public String getInfo(){
    return "Date:"+this.date+"host:"+this.host+"message:"+this.message;
  }
}
```

Figure 4. Program fragment: method call
图 4. 程序片段：方法调用

针对字符串拼接和方法调用两种打印日志的方法，我们分别采用不同的方法生成日志模板。对于字符串拼接的方式，通过拆分加法表达式来获得日志模板。对于方法调用，采用程序的抽象语法树(Abstract Syntax Tree, AST)的方法生成模板。抽象语法树是常用来进行遍历和解析源码的数据结构。日志模版识别的输入为程序源代码，首先构造抽象语法树，然后对其进行遍历，寻找返回值类型为 String 的方法，并获得其返回值的正则表达式，从而产生对应的返回值模板。最终我们将产生程序的方法映射表，映射表的键是返回值类型为 String 的方法名，值为返回值模板。注意，由于不同类中可能包含同名方法，所以方法名必须使用方法的全限定名。最后将两种方法生成的日志模板做进一步整合。

3.1.2. 微服务系统日志划分

我们将日志信息转化为结构化的数据，可以发现日志消息之间具有稳定的相关性，比如，具有相同的服务名称的日志消息来自于同一个微服务实例的日志，而具有相同的请求 ID 的日志消息则是来自于系统中逻辑相关的执行步骤。在进行异常检测之前将日志按照相关的属性聚集到一起，能够提高异常检测模型的效率和准确度。

系统的运行具有时间局部性，出现在一段时间内的日志之前存在的关联性的概率比这段时间之外的日志的关联性要高，因此我们按照时间窗口划分日志进行分析。由于时间窗的大小决定了日志集合的规模和日志分析的效率和精度，因此采用滑动窗口[23]可以优化对日志的划分结果，尽可能保证更高的准确性。滑动窗口由两个属性组成：窗口大小和步长。每完成对一个时间窗口内的日志分析之后，将时间窗口向前滑动一个步长，通常步长小于窗口大小，因此会导致不同窗口的重叠，这样就可能识别出一些之前没有被完全覆盖的日志序列之间的关联。

如果仅通过时间窗口对日志进行划分，异常检测的精度往往受到相关的噪声影响，通过相关的属性将一组日志消息关联起来，能够更准确地反映出一组消息序列中的异常情况。微服务架构软件系统产生

的日志来自不同的功能模块、不同的机器节点，对于同一个请求事件，日志中包含相同的请求 ID，我们根据事件请求 ID 划分日志就可以重构出用户请求的执行路径。

3.2. 建立模型

本文主要针对大规模微服务架构软件系统中请求事件的执行路径异常进行检测，首先提取日志键序列，进而建立异常检测模型。

系统中每个用户请求的事件通常有一个执行路径，通过一系列记录每个事件发生的日志序列可以构造出一条用户请求的执行路径，该路径指向系统某个事件的特定执行顺序。对于不同的用户请求可能对应于不同的执行路径，我们通过预测接下来的事件的概率来判断是否发生了异常。

3.2.1. 提取微服务系统日志键序列

我们将解析之后的日志模板中的有效信息提取出来作为日志键，由于我们需要每个日志模板记录的执行事件构造执行路径，因此将数据集按照时间先后排成时间序列，选择每个日志模板中的消息内容作为日志键，发生在滑动窗口中间的多个日志键序列代表程序的执行路径流。

3.2.2. LSTM 异常检测模型

大规模微服务架构软件系统具有一定的时序动态性，当前的状态往往取决于长期的历史趋势。传统的监督学习方法只能将输入的日志序列看成独立的个体，无法捕获输入的日志序列之间的依赖关系，递归神经网络(Recurrent Neural Network, RNN)可以记忆收到的输入信息，以便精准地预测接下来的输出，但是 RNN 不能存储很长时间的输入信息，本文采用的长期短期记忆网络通过引入内存单元，使传统的 RNN 能够长时间的记住他们的输入信息，可以有效利用到日志序列长期的依赖关系。此外，基于微服务架构的软件系统产生的日志数据量大，标注异常成本高，而基于 LSTM 的神经网络模型是无监督的异常检测方法，不需要进行数据标注。

图 5 是 LSTM 递归神经网络的单元，每个单元包含三个门，输入门、遗忘门和输出门，它们决定了如何对输入做出反应，根据每个节点收到的信息的强度，它将决定阻止还是传递。当信息通过这些单元的时候，我们通过与每个单元关联的权重对其进行筛选。来自上一段时间的内存单元的状态连同当前的数据输入一起进入了下一个输入当中，用以计算新的状态和输出，因此 LSTM 可以记忆历史信息。

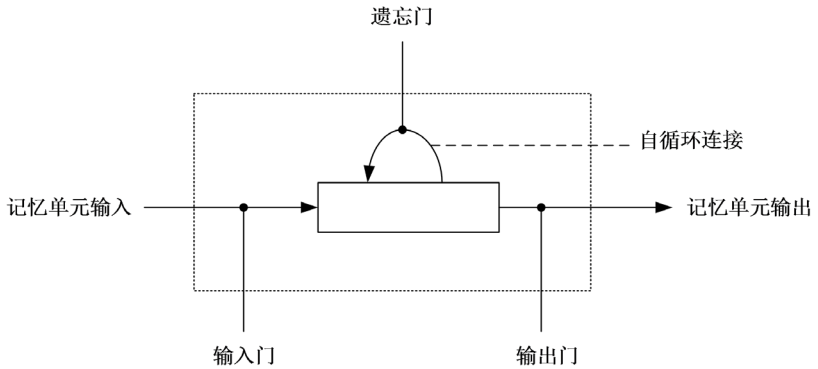

Figure 5. Unit of LSTM recurrent neural network
图 5. LSTM 递归神经网络的单元

我们将历史日志键序列按照一定的窗口大小进行输入，使用 LSTM 异常检测模型对当前日志键进行分析，输出发生异常的概率，从而将异常检测问题转变成一个多分类问题，类别的个数是日志键的个数，使用正常的数据作为训练集。

问题定义为，微服务系统的日志集合为 L (K)，预测在时间窗口 w 内发生异常的概率为 P (w)，输入是长度为 L 的历史特征序列，目标是二元向量 d_i，表示正常或异常。

训练阶段的数据为系统中正常执行所产生的日志键序列，输出每个日志键的条件概率分布模型。在对系统进行在线的异常检测时，模型的输入是 $w=\{m_{t-h},\cdots,m_{t-1}\}$，代表最近出现过的 h 个日志键，输出向量 y_t 表示系统的状态，通过网络输出层的 Softmax 函数来参数化：

$$\Pr\left(d_t=k\middle|y_t\right)=\frac{e^{y_t^k}}{\sum_{i=1}^{k}e^{y_t^k}} \tag{1}$$

对于目标函数，我们使用分类交叉熵损失函数去训练：

$$C=-\sum_{i=1}^{K}w_k\times\left[d_t^i\log\left(y_t^i\right)+\left(1-d_t^i\right)\log\left(1-y_t^i\right)\right] \tag{2}$$

其中，K 为 2，表示分类数，w_k 表示分类的权重，$w_{正}$ 为 0.99，$w_{负}$ 为 0.01，因为根据软件系统长期的运行情况统计，异常日志约占 1%。在异常检测中正样本(异常)相对于负样本(正常)来说数量是很少的，如果对正负样本的损失乘以相同的权重系数，那么就会导致网络把所有的日志都预测成负样本，所以需要给正样本的损失加上一定的权重，当正样本分类错误的时候，乘以一个很大的权重，造成的总的损失会很大，而当负样本分类错的时候，乘以一个很小的权重，但是由于负样本的总数比较多，所以总的损失也是符合实际情况的。d 为 1 或 0。

假如一个日志文件中日志模板为 $\{k_1,k_2,k_3,k_4,k_5,k_6\}$，其中 k_i 表示系统源代码产生的不同的日志键，固定一个滑动窗口大小为 3，那么用来训练异常检测模型的输入日志键序列和输出标签对为：$\{k_1,k_2,k_3\to k_4\}$，$\{k_2,k_3,k_4\to k_5\}$，$\{k_3,k_4,k_5\to k_6\}$，模型的输出是一个概率分布 $\Pr=\left[d_t\middle|y_i\right]=\{k_1:p_1,k_2:p_2,\cdots,k_n:p_n\}$，描述了给定历史的每个日志键会在下一个出现的概率。

由于在实际的日志键序列中，有多种日志键可能都对应着正常的系统事件，因此不能简单地把模型输出概率最大的日志键同当前系统产生的日志进行比较，比较合理的做法是将模型的输出按照概率值大小进行排序，取概率值最大的前 n 个日志键，只要当前检测的日志键在这 n 个日志键中，就认为其是正常的日志，否则是异常日志。

4. 实验及分析

4.1. 实验数据准备

本文实验所用的日志数据来自 KFCoding 功夫编程在线实训平台，该平台基于微服务架构设计如图 6 所示，微服务基于业务功能拆分，独立开发，相互通信。在该系统中，前端用户向后端服务发送 GET/POST 等 HTTP 请求时，请求会先经过服务网关 APIGateway，然后再由它转发到对应的微服务实例上。各个微服务应用中触发的事件都会以消息的方式写入消息中心，微服务实例运行时会产生大量的日志。

EFK 是由一套开源软件组成的日志解决方案，它包括三个组件：Elasticsearch, Fluentd, Kibana。Elasticsearch 是一个分布式的日志存储和日志搜索引擎，通过 Restful 方式进行交互，Fluentd 负责收集日志发送给 Elasticsearch, Kibana 可以将 Elasticsearch 中的数据通过友好的界面展示出来。我们通过 EFK 来收集在线实训平台系统产生的海量日志，当系统管理员发现问题时，会记录该异常数据。

4.2. 准确率

异常检测模型最终的结果要判断日志是正常还是异常，如果正常值标记为 0 (Negative)，异常标志为 1 (Positive)，那么有以下几种基础指标，即混淆矩阵，如表 1 所示。

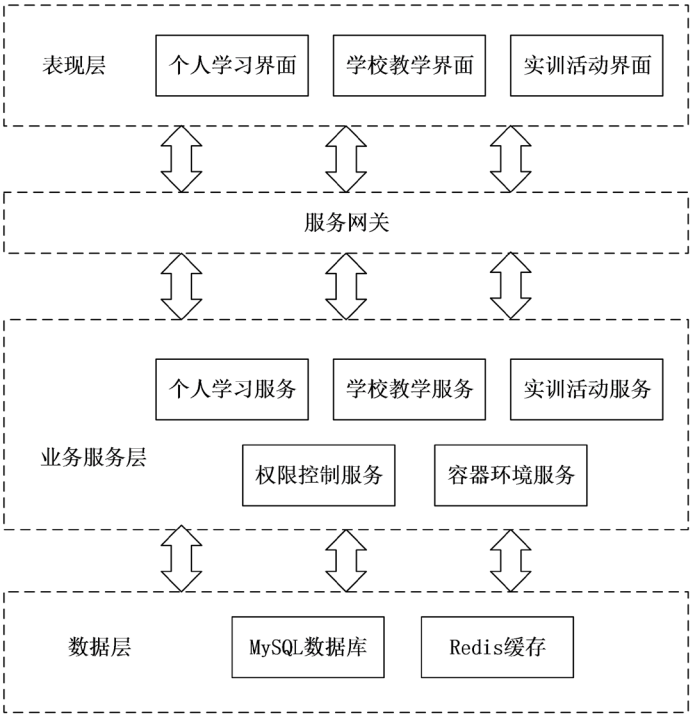

Figure 6. Microservice architecture software system architecture
图 6. 微服务架构软件系统架构图

Table 1. Confusion matrix
表 1. 混淆矩阵

真实值 \ 预测值	Positive (1)	Negative (0)
Positive (1)	True Positive (TP)	False Negative (FN)
Negative (0)	False Positive (FP)	True Negative (TN)

我们使用最常用的度量标准精确率(Precision)、召回率(Recall)、F 值(F-Measure)来评估异常检测方法的准确性，Precision 衡量报告的异常正确率，Recall 衡量检测到的实际异常率，F-Measure 表示精准率和召回率的调和平均值。公式如下：

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{5}$$

日志数据集按时间先后顺序分为训练集和测试集，选择其中的 20%的数据作为训练数据，80%作为测试数据，LSTM 网络设置 2 个隐藏层和 64 个内存单元，用于训练和检测的日志键序列长度为 10，取前 9 个候选日志键为正常日志。我们将本文的 LSTM 异常检测模型同传统的基于统计学习的模型 LogCollect [20]聚类和主成分分析(PCA) [21]方法的实验结果进行比较。实验结果如图 7 所示，实验表明，LSTM 异常检测模型表现出了最佳的整体性能，达到了约 90%的准确率，因为实际生产环境中大规模微服务架构

软件系统异常数据少，样本失衡，传统的基于统计学习的模型将离群点标记为异常，容易出现比较严重的过拟合，泛化性能不佳，具有较高的误报率，且 PCA 对数据集较敏感，聚类方法有多个阈值需要调整。

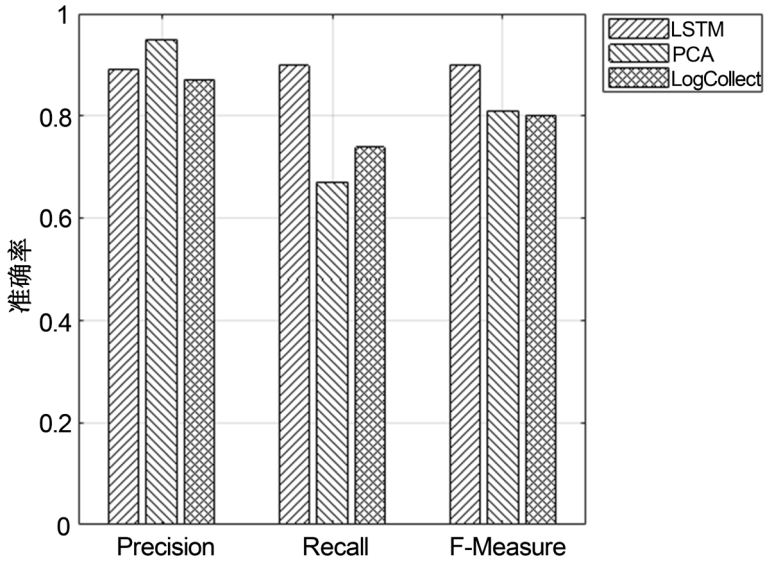

Figure 7. Comparison of the accuracy of LSTM anomaly detection model with PCA and Logcollect methods
图 7. LSTM 异常检测模型的准确率和 PCA、LogCollect 方法的比较

4.3. 日志键序列的长度

考虑 LSTM 异常检测模型的准确率与历史日志键序列的长度关系。如图 8 所示，随着日志键序列逐渐变长，LSTM 异常检测模型的准确率平稳上升，当长度达到 15 的时候准确率开始下降，实验表明，日志键序列的长度并不是越长越好，这是因为在微服务架构系统中，异常的产生可能与之前几个邻近阶段的事件相关性比较大。

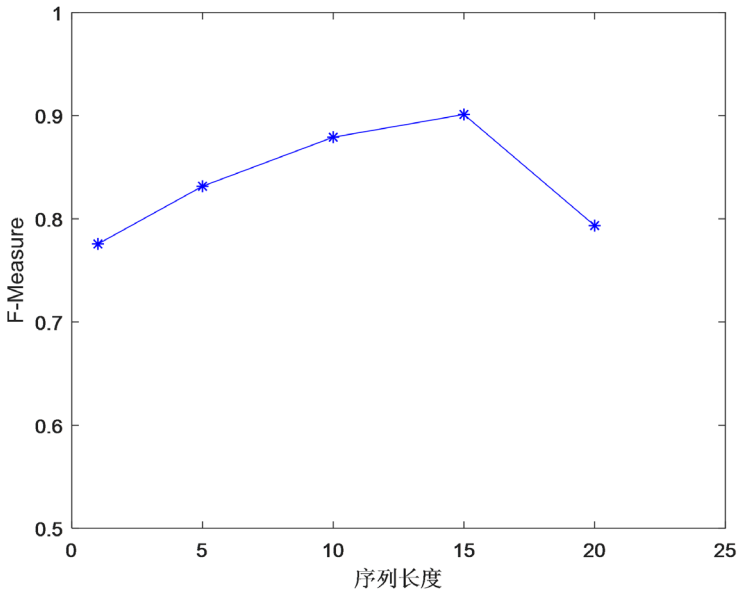

Figure 8. Effect of log key sequence length on the accuracy of LSTM anomaly detection model
图 8. 日志键序列长度对 LSTM 异常检测模型准确率的影响

4.4. 训练数据集的大小

考虑 LSTM 异常检测模型的准确率和训练数据集大小的关系。如图 9 所示，随着训练数据量的增大，LSTM 异常检测模型的准确率越来越高，当训练的数据集达到 1G 的时候，模型性能趋于稳定，实验表明，应该采用适当大小的数据量进行模型训练，避免过拟合。

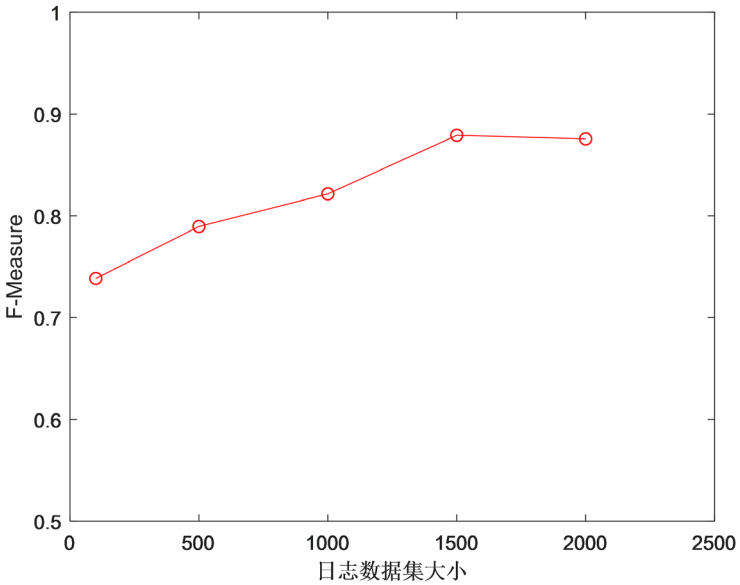

Figure 9. Effect of training data set size on the accuracy of LSTM anomaly detection model
图 9. 训练数据集大小对 LSTM 异常检测模型准确率的影响

5. 结论

本文研究了在大规模微服务架构软件系统中的异常检测，针对其中存在的问题，提出了一种异常检测模型。首先对日志数据进行处理，通过抽象语法树对源代码进行日志解析，再按照时间窗口和事件标识符对日志进行划分，最后通过 LSTM 模型进行异常检测，以检测系统在执行路径中的异常情况，该方法在大规模微服务架构的系统日志中得到了大约 90%的准确率；同时通过实验研究了日志键长度和训练数据集大小对异常检测模型的影响。未来我们将考虑进一步挖掘日志的时序特征，研究日志事件或状态之间的相关性，从而检测大规模微服务架构软件系统在性能方面的异常情况。

基金项目

国家自然科学基金(61772372，61672384)；上海市科学技术委员会科研项目(17JC1400603)。

参考文献

[1] Dragoni, N., Giallorenzo, S., Lafuente, A.L., *et al.* (2016) Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M. and Meyer, B., Eds., *Present and Ulterior Software Engineering*, Springer, Cham, 195-216. https://doi.org/10.1007/978-3-319-67425-4_12

[2] Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J. and Montesi, F. (2016) Self-Reconfiguring Microservices. In: Ábrahám, E., Bonsangue, M. and Johnsen, E., Eds., *Theory and Practice of Formal Methods. Lecture Notes in Computer Science*, Springer, Cham, 194-210. https://doi.org/10.1007/978-3-319-30734-3_14

[3] Thönes, J. (2015) Microservices. *IEEE Software*, **32**, 113-116. https://doi.org/10.1109/MS.2015.11

[4] 廖湘科, 李姗姗, 董威, 等. 大规模软件系统日志研究综述[J]. 软件学报, 2016, 27(8): 1934-1947.

[5] Alspaugh, S., Chen, B., Lin, J., Ganapathi, A., Hearst, M. and Katz, R. (2014) Analyzing Log Analysis: An Empirical Study of User Log Mining. LISA14, 62-77.

[6] Lee, G., Lin, J., Liu, C., Lorek, A. and Ryaboy, D. (2012) The Unified Logging Infrastructure for Data Analytics at Twitter. *Proceedings of the VLDB Endowment*, **5**, 1771-1780. https://doi.org/10.14778/2367502.2367516

[7] 陆杰, 李丰, 李炼. 分布式系统中的日志分析及应用[J]. 高技术通讯, 2019, 29(4): 303-320.

[8] Tang, L., Li, T. and Perng, C.S. (2011) LogSig: Generating System Events from Raw Textual Logs. In: *Proceedings of the* 20*th ACM International Conference on Information and Knowledge Management*, 785-794. https://doi.org/10.1145/2063576.2063690

[9] Fu, Q., Lou, J.G., Wang, Y. and Li, J. (2009) Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. 2009 Ninth IEEE International Conference on Data Mining, Miami, FL, 6-9 December 2009, 149-158. https://doi.org/10.1109/ICDM.2009.60

[10] Vaarandi, R. (2004) A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. In: Aagesen, F.A., Anutariya, C. and Wuwongse, V., Eds., *Intelligence in Communication Systems. INTELLCOMM* 2004. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 293-308. https://doi.org/10.1007/978-3-540-30179-0_27

[11] Yamanishi, K. and Maruyama, Y. (2005) Dynamic Syslog Mining for Network Failure Monitoring. *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, Chicago, IL, 21-24 August 2005, 499-508. https://doi.org/10.1145/1081870.1081927

[12] Zhao, X., Zhang, Y., Lion, D., *et al*. (2014) LPROF: A Non-Intrusive Request Flow Profiler for Distributed Systems. In: *Proceedings of the* 11*th USENIX Symposium on Operating Systems Design and Implementation*, Broomfield, CO, 629-644.

[13] Kc, K. and Gu, X. (2011) ELT: Efficient Log-Based Troubleshooting System for Cloud Computing Infrastructures. *Proceedings of the* 30*th IEEE Symposium on Reliable Distributed Systems*, Madrid, Spain, 4-7 October 2011, 11-20. https://doi.org/10.1109/SRDS.2011.11

[14] Debnath, B, Khan, L, Solaimani, M., *et al*. (2018) LogLens A Real-Time Log Analysis System. *IEEE* 38*th International Conference on Distributed Computing Systems*, Vienna, Austria, 2-6 July 2018, 1052-1062. https://doi.org/10.1109/ICDCS.2018.00105

[15] Beschastnikh, I., Brun, Y., Ernst, M.D. and Krishnamurthy, A. (2014) Inferring Models of Concurrent Systems from Logs of Their Behavior with CSight. *Proceedings of the* 36*th International Conference on Software Engineering*, Hyderabad, Italy, 31 May-7 June 2014, 468-479. https://doi.org/10.1145/2568225.2568246

[16] Beschastnikh, I., Brun, Y., Schneider, S., *et al*. (2011) Leveraging Existing Instrumentation to Automatically Infer Invariant-Constrained Models. *Proceedings of the* 19*th ACM SIGSOFT Symposium and the* 13*th European Conference on Foundations of Software Engineering*, Szeged, Hungary, 5-9 September 2011, 267-277. https://doi.org/10.1145/2025113.2025151

[17] Logstash (2018) Centralize, Transform & Stash Your Data. https://www.elastic.co/cn/products/logstash

[18] Chen, M., Zheng, A.X., Lloyd, J., Jordan, M.I. and Brewer, E. (2004) Failure Diagnosis Using Decision Trees. *Proceedings of the* 1*st International Conference on Autonomic Computing*, New York, 17-18 May 2004, 36-43.

[19] Liang, Y., Zhang, Y., Xiong, H. and Sahoo, R. (2007) Failure Prediction in IBM BlueGene/L Event Logs. *Seventh IEEE International Conference on Data Mining*, Omaha, NE, 28-31 October 2007, 583-588. https://doi.org/10.1109/ICDM.2007.46

[20] Lin, Q., Zhang, H., Lou, J.G., Zhang, Y. and Chen, X. (2016) Log Clustering Based Problem Identification for Online Service Systems. 201616 *Proceedings of the* 38*th International Conference on Software Engineering*, Austin, TX, 14-22 May 2016, 102-111. https://doi.org/10.1145/2889160.2889232

[21] Xu, W., Ling, H., Fox, A., Patterson, D. and Jordan, M.I. (2009) Detecting Large-Scale System Problems by Mining Console Logs. *Proceedings of the ACM SIGOPS* 22*nd Symposium on Operating Systems Principles*, Big Sky, MT, 11-14 October 2009, 117-132. https://doi.org/10.1145/1629575.1629587

[22] Lou, J.G., Fu, Q., Yang, S., Xu, Y. and Li, J. (2010) Mining Invariants from Console Logs for System Problem Detection. *Proceedings of the* 2010 *USENIX Conference on USENIX Annual Technical Conference*, Boston, MA, 23-25 June 2010, 24.

[23] Fu, X., Ren, R., Zhan, J., *et al*. (2012) LogMaster: Mining Event Correlations in Logs of Large-Scale Cluster Systems. Proceedings of the 31st Symposium on Reliable Distributed Systems, Irvine, CA, 8-11 October 2012, 71-80. https://doi.org/10.1109/SRDS.2012.40