**摘要**

在现代计算机软件系统中，系统日志一直是检查系统状态的首要来源，随着计算机软件系统变得越来越复杂，软件和硬件之间的交互也变得越来越频繁。系统软件和硬件会生成大量的日志信息，这些日志信息包括运行时状态报告以及错误信息。使用纯粹人工方式对如此庞大的数据进行分析是不切实际的。

本文根据大型分布式软件的特点及需求，提出了一种基于日志数据的异常检测算法，通过日志解析技术从原始的日志数据中提取日志模板并将日志进行结构化得到事件序列，利用窗口化特征提取技术从结构化日志数据中提取关键特征，由于日志特征服从时间上的先后顺序，我们将基于日志数据的异常检测技术抽象为时间序列的异常检测方法，并提出一种基于序列相似性比较的时间序列异常检测算法，通过检测时间序列中的异常点或异常序列进行对系统异常的检测。通过在线实训平台kfcoding日志的实验证实本文提出的基于日志数据的异常检测算法具有较高的准确性和通用性。

**关键词**：日志解析，特征提取，时间序列，异常检测

**ABSTRACT**

In modern computer software system, system log is always the primary source of checking system status. With the computer software system becoming more and more complex, the interaction between software and hardware is becoming more and more frequent. System software and hardware will generate a lot of log information, which includes runtime status report and error information. It is impractical to analyze such a large amount of data in a purely human way.

According to the characteristics and requirements of large-scale distributed software, this paper proposes an anomaly detection algorithm based on log data, which extracts log template from original log data by log parsing technology and obtains event sequence by structured log, and extracts key features from structured log data by windowed feature extraction technology, because log features follow the time first. After that, we abstract the anomaly detection technology based on log data into the anomaly detection method of time series, and propose an anomaly detection algorithm based on sequence similarity comparison, which detects the system anomaly by detecting the outliers or abnormal sequences in time series. The experiment of online training platform kf-coding log proves that the anomaly detection algorithm based on log data proposed in this paper has high accuracy and generality.

**Key Words:** log parsing, feature extraction, time series, exception detection

**目录**

[**第1章 绪论** 6](#_Toc28034263)

[1.1 研究背景与意义 6](#_Toc28034264)

[1.2 日志解析方法综述 7](#_Toc28034265)

[1.3 基于日志的异常检测研究现状 8](#_Toc28034266)

[1.4 本文研究内容 9](#_Toc28034267)

[1.4.1 基于日志的异常检测算法面临的挑战 9](#_Toc28034268)

[1.4.2 本文研究内容 9](#_Toc28034269)

[1.5 本文组织结构 10](#_Toc28034270)

[**第2章 基于日志数据的异常检测一般流程和评价机制** 12](#_Toc28034271)

[2.1 基于日志数据的异常检测工作流 12](#_Toc28034272)

[2.2 日志解析方法的一般原理 13](#_Toc28034273)

[2.2.1 日志解析的定义 13](#_Toc28034274)

[2.2.1 日志解析的一般方法 14](#_Toc28034275)

[2.2.1 日志解析算法的评价准则 15](#_Toc28034276)

[2.3 基于日志数据的异常检测 16](#_Toc28034277)

[2.3.1 研究现状和主要挑战 16](#_Toc28034278)

[2.3.2特征提取 17](#_Toc28034279)

[2.3.3异常检测 17](#_Toc28034280)

[2.3.3评价准则 19](#_Toc28034281)

[2.4 本章小结 20](#_Toc28034282)

[**第3章 基于加权编辑距离聚类的日志解析算法** 21](#_Toc28034283)

[3.1 问题描述与符号定义 21](#_Toc28034284)

[3.1.1 问题描述 21](#_Toc28034285)

[3.2.1 符号定义 22](#_Toc28034286)

[3.2 基于LKE的日志解析算法 24](#_Toc28034287)

[3.2.1 算法概述 24](#_Toc28034288)

[3.2.2 步骤一 事件分组和标记变量替换 24](#_Toc28034289)

[3.2.2 步骤二 聚类 26](#_Toc28034290)

[3.2.3 步骤三 类中分组 27](#_Toc28034291)

[3.2.4 步骤四 消息模板生成 28](#_Toc28034292)

[3.3 实验结果与分析 29](#_Toc28034293)

[3.4 本章小结 31](#_Toc28034294)

[**第4章 基于固定窗口的日志特征提取算法** 33](#_Toc28034295)

[4.1 问题描述与符号定义 33](#_Toc28034296)

[4.1.1 问题描述 33](#_Toc28034297)

[4.1.2 符号定义 34](#_Toc28034298)

[4.2 一种基于固定窗口的日志特征提取算法 35](#_Toc28034299)

[4.2.1 算法概述 35](#_Toc28034300)

[4.2.2 特征提取 35](#_Toc28034301)

[4.3 实验结果与分析 36](#_Toc28034302)

[4.3.1 数据准备 36](#_Toc28034303)

[4.3.1 实验结果 36](#_Toc28034304)

[4.3 本章小结 37](#_Toc28034305)

[**第5章 基于序列相似性比较的异常检测算法** 39](#_Toc28034306)

[5.1 问题描述与符号定义 39](#_Toc28034307)

[5.1.1 问题描述 39](#_Toc28034308)

[5.1.2 符号定义 40](#_Toc28034309)

[5.2 一种基于序列相似度比较的异常检测算法 40](#_Toc28034310)

[5.3 实验结果与分析 44](#_Toc28034311)

[5.3.1 数据准备 44](#_Toc28034312)

[5.3.2 评价准则 45](#_Toc28034313)

[5.3.3 实验结果 45](#_Toc28034314)

[5.3.4 结果分析 47](#_Toc28034315)

[5.4 本章小结 48](#_Toc28034316)

[**第6章 总结与展望** 49](#_Toc28034317)

[6.1 总结 49](#_Toc28034318)

[6.2 进一步工作的方向 49](#_Toc28034319)

[**致谢** 50](#_Toc28034320)

[**参考文献** 51](#_Toc28034321)

**第1章 绪论**

1.1 研究背景与意义

系统日志是记录现代系统中硬件、软件和系统问题的信息，同时还可以监视系统中发生的事件。日志经常被广泛用来记录软件系统的实时信息，比方说一件事件的发生时间戳，一个用户请求的用户ID，或者任务执行的状态。系统日志所携带的丰富信息可以帮助系统开发者和系统维护人员对系统进行实时监控，同时对日志信息的挖掘可以追溯系统问题。不仅如此，系统日志分析技术也应用在其他许多领域，交易系统日志帮助维护者查找安全隐患，电力系统利用日志信息发现供电异常，数据中心系统利用日志信息挖掘技术监测系统问题。对于一个软件系统的稳健运行来说，能够快速地分析日志信息并在安全问题发生之前准确地诊断是至关重要的。

当今时代，我们面临着大规模互联网服务的爆炸式增长，这些服务的背后由一系列大型服务集群支撑着。云计算的发展同时也驱动着大型数据中心的建设和部署。大型的计算服务系统包含着成千上万的分布式单元，这些分布式单元包括服务器，网络设备，分布式计算软件以及操作系统。随着现代软件系统规模和复杂度的增长，系统日志的规模也在继续增长，传统的依靠人工观察对日志进行分析的方法不仅耗费大量的人力物力同时也是非常容易出错的。为了应对这一挑战，针对分布式系统的全自动的日志解析和日志挖掘技术不仅是至关重要的也是非常迫切的。

系统日志，作为一个广大的信息源，携带了大量重要的系统信息，对日志信息的挖掘可以帮助维护一个软件系统的稳健性和持久性。许多研究者对自动化日志分析和异常检测系统的设计和发展进行了大量的研究。系统日志可以被用来检测系统异常，监测网络故障，甚至查找系统ｂｕｇ。包含着系统内在基础信息的日志可以帮助开发人员和运维人员更好地理解系统行为。

现阶段，由于系统开发者会使用简单的文本记录日志信息，因此我们得到的日志信息通常是非结构化的。识别日志信息中的结构化信息可以帮助对特定系统信息的提取，这些结构化信息包括事件类型，事件发生的时间戳，一件事件的发生源，一件事件的目的地，关键的性能度量信息等等。因此为了实现自动化地对非结构化日志信息进行挖掘分析，第一步即是使用日志解析工具将非结构化的原始日志信息转变为结构化事件序列。

传统的基于日志的异常检测方法，使用基于规则的算法，规则通常是由富有经验的运维人员来制订的，当某一条日志信息违反了先前制订的规则将被视为异常。另外，一些基于传统统计学习方法的日志分析技术通过对系统日志的分析度量来进行系统异常检测。这种方法从日志信息中提取信息，使用统计学习技术建立模型，并通过学习的模型自动地识别系统错误和系统问题。这些统计学习方法通常包括子空间分析，无监督聚类分析以及有监督分类算法。现今随着数据和算力的增长，深度学习、强化学习、迁移学习等算法的大力发展，这些算法也大量被应用在日志信息的挖掘上。

综上所述，针对特定分布式系统的准确的，高效的日志解析和日志挖掘系统对分布式系统的稳健运营以及系统开发人员和运维人员都至关重要。本文针对分布式软件系统，提出了一种创新的日志解析和日志挖掘算法，该算法在分布式系统日志上表现出极好的准确度和高效性，同时本文的实验结果也证明该算法在其他分布式系统也有极强的通用性。

1.2 日志解析方法综述

为了实现自动化的非结构化日志挖掘，首先是使用日志解析技术将非结构化的日志数据进行转换，将其转变为结构化数据，对日志挖掘系统来说，一个高效的日志解析算法非常重要，日志解析技术通常可以提取系统中关键信息。

在日志解析方法上已有大量的研究，比方说正则表达式，利用源代码，或者利用数据挖掘算法基于系统日志特征提取，这类数据挖掘算法包括聚类或者迭代分割等等。

传统的日志解析算法大多依赖正则表达式来提取特定的日志事件(SEC[1])，这种算法需要特定领域专家的专业知识，并且针对特定系统制订的规则很难适用于其他系统，因此通用性大打折扣。不仅如此，随着现代软件系统规模和复杂度的不断增加，包含多种日志事件的日志规模体积也在不断增大，显然人工方式对这些正则表达式规则的定义是不切实际的。尤其是，当一个系统持续地更新迭代时，则日志解析的规则也将会过时，因此现在的软件系统需要自动化的日志解析方法，可以随着系统的迭代而更新。同样的，利用源代码的日志解析算法也缺乏通用性。包括基于正则表达式和基于源代码方法的日志解析算法都无法真正地实现在线解析。

现今同样存在着对合适地系统日志管理和存储系统，因此日志管理系统被大量地部署在大规模软件平台上，一个典型的日志管理平台架构如图１所示，在每个节点中，一个ｌｏｇ　ｓｈｉｐｐｅｒ通常包含一个日志解析工具，一个日志索引，一个存储引擎和一个用户界面。在类似这样地系统中，默认的日志解析工具通常只会解析简单的语法信息，比方说时间戳和ｈｏｓｔｎａｍｅ。经过这样处理后的日志信息本身仍是无结构的。

另外，为了可以实现自动化的日志解析系统，许多研究者提出了大量的基于数据方法的自动日志解析系统。数据驱动的日志解析系统使用以往的日志信息进行统计模型的训练来抽取日式事件，这些典型的工具包括有SLCT[25], IPLoM [47], LKE[3], LogSig[48]。

1.3 基于日志的异常检测研究现状

传统的自动日志分析工具是通过将日志和正常的日志进行一系列的规则对比，如果当前日志违反了某些规则，则可以从中发现异常行为信息，通常来说，规则是由富有经验的系统专家根据系统的设计和实现而制定的，这种方法通常被称为简单的基于规则的挖掘技术。

另外，一些基于传统统计学习方法的日志分析技术通过对系统日志的分析度量来进行系统异常检测。这种方法从日志信息中提取信息，使用统计学习技术建立模型，并通过学习的模型自动地识别系统错误和系统问题。这些统计学习方法通常包括子空间分析，无监督聚类分析以及有监督分类算法。现阶段，在处理大规模系统中复杂的监控和诊断问题上，统计机器学习和数据挖掘技术显示出极大的潜力。通过统计性地分析系统日志，一些基于统计学习的方法被用来监测系统问题。比方说，Dickenson[55]等使用基于字符串距离度量地分类方法将相似地日志序列分为一类，分析师通过检查其中的一类或者几类与其他类的相似性来决定其是否为异常的。Mirgorodskiy[56]等人同样也使用字符串距离度量的方法将函数水平上的记录进行分类，并使用某一类与其他类的差异性来决定该类别是否异常。Yuan[57]等人首先使用 n-grams算法从系统调用序列中提取特征，接着根据已知问题序列的相似性使用支持向量的算法来对序列进行分类。Xu[58]等人首先对日志序列进行预处理，提取其中的消息数量作为日志特征，接着使用主成分分析方法来检测异常。

另外一类算法[59,60,61]使用有限状态自动机模型来表示日志序列，这种方法对系统开发和运维人员来说比较好理解。比方说，SALSA [61]使用Hadoop中的Datanode和TaskTracker日志信息来建立FSA模型。Cotroneo等人[59]使用ｊａｖａ虚拟机中的日志记录来创建FSA模型，一个新的记录将会被拿来与建立的FSA模型进行比较来确定其是否为异常的。同样的，文献［60］中的作者使用FSA模型来表示正常系统日志的行为，一个新的日志序列如果不符合FSA将会被视为异常的。

1.4 本文研究内容

1.4.1 基于日志的异常检测算法面临的挑战

由上文可知，系统日志蕴涵着大量的系统信息，对日志信息的分析挖掘有利于开发人员和运维人员更好地理解和维护系统，逐渐成为现今大规模软件系统所需要的关键技术。基于日志信息的异常检测技术可以利用日志解析技术自动化地从原始的系统日志中提取关键信息，并通过日志挖掘技术检测系统的异常行为。一个全自动的、高效的基于日志的异常检测算法需要日志解析和日志挖掘技术的共同合作来运行。

传统的基于正则表达式的日志解析技术通常需要特定领域的专家来制订针对特定系统的解析规则，一旦制订很难适用于其他的系统，因此通用性不高。另外基于源代码的日志解析技术也是针对某一类源代码，同样缺乏通用性。另外，随着系统的持续运行，对于流式的日志数据，现在很少有同时具有非常高效性和通用性的在线日志解析算法。许多“在线”的日志解析技术需要搜集一段时间的日志后对搜集的日志进行解析，这样即使出现了异常，也错过了最佳的诊断期。不仅如此，一些在线的基于日志信息的异常检测算法需要对日志信息进行实时地提取特征和分析，因此一种有效的在线日志解析技术是非常迫切的。

基于日志的异常检测算法也同样面临着一般异常检测算法所面临的挑战。传统的一些基于规则的异常检测算法，规则的制订在使用统计机器学习方法进行异常检测时，不仅需要从大量的非结构化日志信息中提取重要的特征信息，同时需要大量的训练数据建立统计模型，一方面，在训练集的构造时，有标注的真实数据集很难获得，另一方面，即使得到了有标注的训练数据集，真实的异常数据也是比例很少的，这是典型的不平衡数据分析问题。

1.4.2 本文研究内容

本文在对大量的日志解析技术和异常检测算法进行研究之后，提炼出了对针对分布式系统的在线的基于日志的异常检测算法的需求，提出了一种具有较高通用性，准确性的基于日志的异常检测算法。主要研究内容包括一下三个部分：

⑴ 日志解析。搜集到的原始日志数据包括时间戳、源、目的、消息等等部分，消息部分通常是由一条打印语句定义的字符串，描述了该条日志事件的具体信息，日志提取主要是从大量的杂乱无章的原始日志数据中提取到消息模板/事件类型和事件序列这些结构化的数据。通常基于日志数据进行的挖掘算法都会将日志解析作为算法的第一步。一般来说，一个消息模板对应源代码中定义的一条打印语句，我们根据日志数据的主要特点，对相关日志提取算法进行了研究并介绍了一种结合聚类和生成式算法的日志提取方法。

⑵ 特征提取。根据日志解析出的结构数据中提取重要的特征是我们研究的第二个方向，本文首先研究了从结构化日志数据中提取特征的主要方法，并介绍了一种基于固定窗口的特征提取算法，从结构化日志数据中提取重要的特征信息作为异常检测算法的输入，特征提取算法利用了日志数据中时间戳的信息，提取的特征也遵从于时间上的先后顺序关系。

⑶ 异常检测。根据提取的特征，由于特征按时间顺序排列即是时间序列，因此我们利用时间序列的异常检测算法，通过检测异常的序列达到对异常事件的检测，我们对时间序列异常检测算法进行了大量的研究并介绍一种基于序列相似度比较的时间序列异常检测算法。

1.5 本文组织结构

本文由六个章节组成，各个章节的内容安排如下：

第1章 绪论。介绍日志解析和日志挖掘技术的研究背景和研究意义，现有的日志解析技术和日志挖掘算法和这些技术方法现阶段所面临的挑战，以及介绍了本文的主要研究内容。

第2章 基于日志的异常检测技术的一般原理和评价标准。介绍基于日志的异常检测技术的基本工作流，以及本文介绍的基于日志的主要工作机制，主要包括日志提取、特征提取和异常检测，分别从这三个部分介绍主流的研究方法、面临的挑战以及评价机制。

第3章 日志提取。根据搜集日志的主要特征，介绍一种结合聚类算法和生成式算法的日志提取方法，算法通过四个步骤从原始的日志数据中提取消息模板/事件类型，并生成事件序列，对新的一条日志消息，我们引入了一种具有固定高度的树结构模型进行消息模板匹配以及异常日志诊断。

第4章 特征提取。根据第三章介绍的日志提取方法可以从原始的日志数据中提取事件序列，介绍了一种基于固定窗口的特征提取方法，可以从事件序列中提取事件特征形成时间序列，将基于日志数据的异常检测算法抽象为时间序列的异常检测算法。

第5章 异常检测。根据第四章提取的特征生成时间序列数据，介绍一种基于序列相似度比较的时间序列异常检测算法，可以高效准确地检测时间序列数据中的异常序列。

**第2章 基于日志数据的异常检测一般流程和评价机制**

2.1 基于日志数据的异常检测工作流

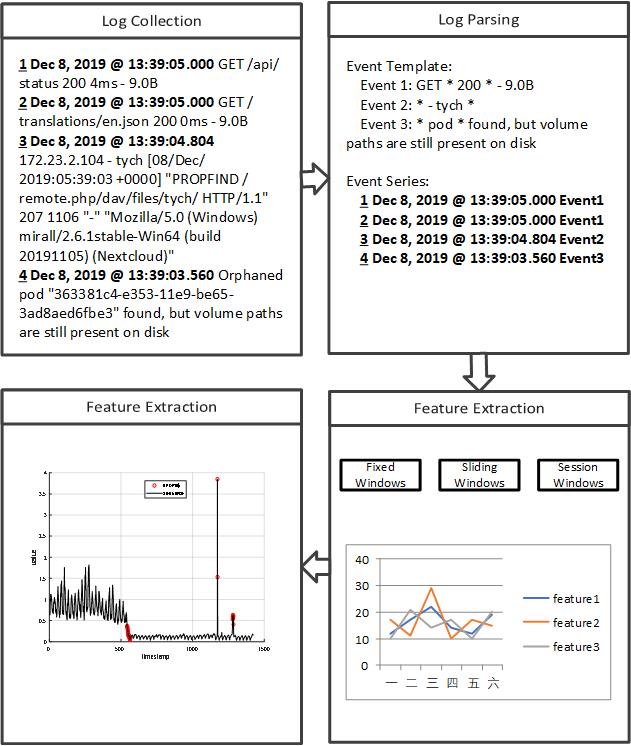
一般基于日志的异常检测算法主要需要以下四个步骤：日志搜集、日志解析、特征提取和异常检测，如图2.1所示。

图2.1 基于日志数据的异常检测算法一般流程

日志搜集：大规模软件系统通常会生成记录了系统状态和运行信息的系统日志，每一条日志包括一个时间戳和一条标示发生时间的日志消息。这些富含价值的日志消息具有多种用途，比方说异常检测，因此这些日志通常被搜集起来为了之后的使用。

日志解析：日志是包含纯文本的无结构化数据，日志解析的作用是从原始日志数据中提取一组事件模板，将无结构化的日志数据转变为结构化数据。每一条日志消息可以被解析为一件事件模板（这里被称为常量部分）和一些特定的参数（变量部分）。

特征提取：经过将日志解析为分别的事件后，我们需要将结构化的数据转变为数值型的特征向量，接着可以使用机器学习方法建立模型。一般我们会使用不同的分组技术将原始的日志分为一系列的日志序列，这些分组技术包括固定窗口，滑动窗口等等。接着我们会从每一日志序列中提取特征向量形成特征矩阵。

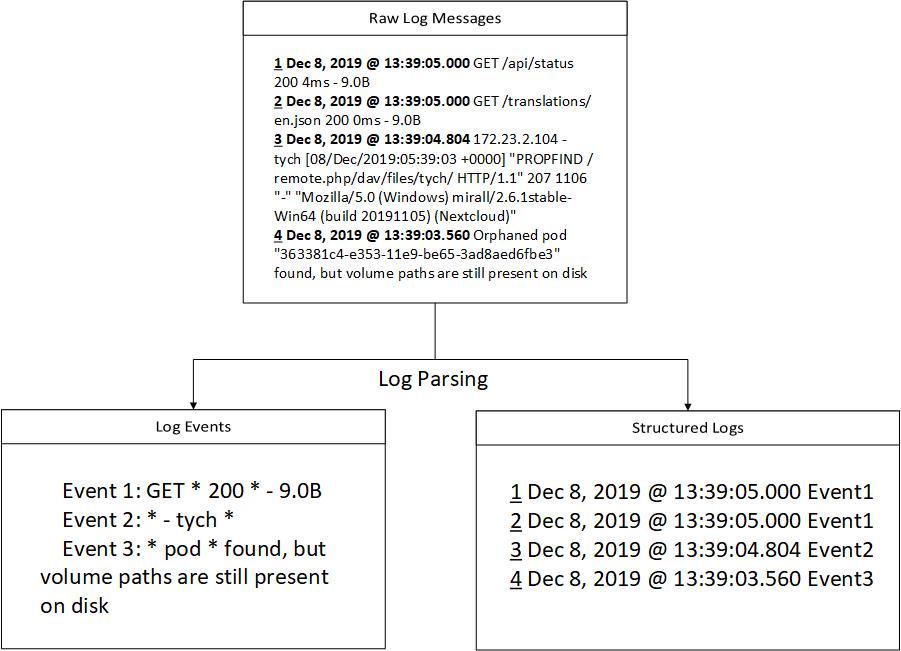
异常检测：最后我们将得到的特征矩阵送入机器学习模型训练生成适用于异常检测的模型。这个模型可以被用来判定一条新的日志序列是否异常。

2.2 日志解析方法的一般原理

2.2.1 日志解析的定义

日志是由常量部分和变量部分组成的纯文本，这些部分在不同的情况下可能有所不同。比方说，对于这样两条日志：“Connection from 10.10.34.12 closed”和“Connection from 10.10.34.13 closed”，单词“Connection”，“from”和“closed”被认为是常量部分，因为这些单词通常是不变的，剩余的部分为变量部分因为他们是不固定的。常量部分是由开发者在源代码中定义好的，而变量部分是动态生成的，比方说端口号，IP地址等等。一个日志解析工具的主要功能是将日志信息中的常量部分和变量部分区分开并形成日志事件，比方之前给出的例子中，日志解析工具生成的典型的日志事件为“Connection from \* closed”。

图2.2显示了一个日志解析工具的主要功能，原始的日志数据来自于在线实训平台kfcoding。原始的日志数据是非结构数据，包括时间戳和原始信息内容（为了简单表示一些内容有所省略）。在真实的情形下，一个日志文件中包含成千上万条这样的日志信息。日志解析工具在将原始的日志消息划分为常量部分和变量部分后，所有的常量消息模板会形成一系列的日志事件，每条原始日志消息生成的结构化日志部分对应着特定的事件。一个日志解析工具的输出是两个文件：日志事件和结构化数据。日志事件记录提取的日志消息模板，结构化数据包括一系列事件和事件的发生事件。最后，解析后的结构化日志可以非常方便地被用来进行日志挖掘，包括异常检测和部署验证。

图2.2 日志解析工具的主要功能

2.2.1 日志解析的一般方法

日志解析对日志挖掘非常重要，传统的日志解析技术使用正则表达式匹配来提取日志事件(SEC[1])。但是现代的软件系统规模和复杂性都越来越大，生成的日志也越来越多，人工制订正则表达式的规则是不切实际的。针对这种现象，现代的研究者提出了基于数据的自动化日志解析算法，这些算法使用历史日志数据建立统计模型来进行事件提取。

比较典型的基于数据的日志解析方法可以大致分为两类：基于聚类方法的和基于启发式方法的。对于基于聚类方法的日志解析算法，首先会计算日志间的距离，接着使用聚类方法将日志聚成不同的簇，最后从这些簇中生成事件模板。对于基于启发式的方法，首先会统计每个单词在每个日志位置上出现的次数，接着，频繁出现的单词被选为事件候选，从这些候选中选择单词作为日志事件。我们在这里介绍四种典型的基于数据的自动化的日志解析方法。

Risto Vaarandi等在2003年提出了SLCT(Simple Log Clustering Tool)，一种自动的日志解析技术，同时开源了相应的日志解析工具。SLCT之后被广泛地用来进行日志挖掘任务，比方说事件日志挖掘，系统问题根因分析和网络预警分类。受关联规则挖掘算法的启发，SLCT会对日志消息进行两次扫描，并且包含三个步骤：⑴ 建立单词字典，第一次对日志进行扫描并建立包含每个单词频率和坐标的字典；⑵ 建立日志簇，在第一步建立的字典上第二次扫描日志并建立日志簇；⑶ 生成日志模板，从第二步中建立的日志簇选取包含有足够日志信息的日志簇，每一个簇可以生成一个日志模板，余下的日志簇被视为异常簇。

IPLoM(Iterative Partitioning Log Mining)是一种基于系统消息特点设计的生成式方法，也被用于很多的日志挖掘方法，比方说预警检测，事件日志分析和事件摘要。在生成事件模板之前，IPLoM会首先进行三次启发式划分过程：⑴ 通过日志消息的不同长度进行划分；⑵ 基于标记位置进行划分于每一次划分，不同位置的单词都会统计次数，接着拥有最少变化单词的位置将用来对日志消息进行划分；⑶ 基于映射的划分，通过单独的标记集合在两个标记位置之间的映射关系对簇进行划分；⑷ 日志模板生成，类似于SLCT，最后一步是从每个簇中生成日志模板。

微软提出了一种日志解析方法LKE(Log Key Extraction)，该方法也被用于一系列的非结构化日志分析任务中。LKE同时使用了聚类算法和启发式规则来进行日志解析：⑴ 日志聚类：首先使用层次聚类算法和自定义的加权编辑距离度量对原始日志消息进行聚类；⑵ 对簇进行划分，接着使用基于启发式规则的划分方法对第一步得到的簇；⑶ 生成日志模板，最后一步是从每一个簇中生成日志模板，使用的方法类似于SLCT和IPLoM。

LogSig是一种比较新的日志解析方法，包含三个过程：⑴ 生成单词对，每一个日志消息会被转化为一系列的单词对，单词对包括单词本身和它的位置信息；⑵ 日志聚类；根据单词对，对每一个日志消息计算一个度量值来决定该条日志消息属于哪一个簇，经过若干轮的迭代，日志消息就会被聚成几类；⑶ 日志模板生成，在每一个簇中，通过对日志消息的提取生成日志模板。

2.2.1 日志解析算法的评价准则

一般来说，日志解析算法需要从非结构化的原始日志数据中提取日志模板和结构化的日志信息，并为日志挖掘做数据准备。一个好的日志解析工具除了需要实现上述的主要功能外，还应满足以下几项标准：

⑴ 日志解析工具解析出的日志模板和结构化日志数据需要具有极高的准确性。由上文所知，日志解析工具是进行日志挖掘必经的关键一步，日志解析工具生成的结构化日志数据将会被用来进行异常检测、系统错误根因分析、系统性能分析等等，这些日志挖掘任务本身就需要极高的准确性，更不用说用来生成分析数据集的日志解析算法了。

⑵ 日志解析算法是否可以对大规模的原始数据也能高效地处理。现代的软件系统大多复杂，生成的日志不仅数量大并且形式也很多变，因此，对于这种形式复杂的大型日志数据，良好的日志解析算法也应表现出准确和高效的解析效果。

⑶ 现今的日志挖掘技术大多是异常检测，基于日志的异常检测算法中，日志解析算法是为异常检测算法做服务的，因此，不同的日志解析算法是否会对异常检测算法的效果造成影响，影响是否是积极的，这些因素在选择日志解析算法也非常重要。

2.3 基于日志数据的异常检测

2.3.1 研究现状和主要挑战

系统异常检测是指在一系列时序行为中发现系统异常行为，在现在大规模软件系统的事件管理中体现着重要的作用。实时的系统异常检测需要开发者或者运维人员可以及时发现问题并立即解决。在系统运行过程中，系统通常会实时地生成记录系统信息的日志消息。这种包含大量系统信息的日志是系统异常检测的重要数据来源。不管在工业界还是学术界，基于日志的异常检测方法已经成为了一种非常具有实际意义的研究方向。

对于传统的独立系统，开发人员通常人工地检查系统日志或者根据领域知识制订某些规则来进行异常检测，使用的方法包括关键字检索或者正则表达式匹配。但是对于现在的大规模软件系统，这些依靠人工检查日志的异常检测方法是远远不够的，主要原因有以下几点：

⑴ 现今的系统通常规模很大并且并行运行以至于系统行为非常复杂，因此这些系统的开发运维任务通常由很多开发人员协作完成，每个运维人员只会负责自己相应的部分，所以一个开发人员可以完全理解整个系统的运作是根本不可能的，这也说明从巨大的系统日志中提炼异常的规则是巨大的挑战。

⑵ 现代系统以每小时50千兆字节的速率生成庞大的系统日志，对于如此庞大的数量，人工地进行噪声过滤、提取关键信息和异常检测是非常困难的。

⑶ 大规模软件系统通常具有不同的错误容忍机制，系统有时会冗余同一任务，甚至会为了提升系统性能主动终止某一风险任务。在这种设定下，使用关键字搜索或者提取敏感日志信息的传统异常检测方法会产生非常多的错误分类。

因此，自动化的基于日志分析的异常检测技术是非常迫切的，已有很对对基于日志的异常检测算法的研究。这些的算法包括有监督的统计学习方法，无监督的统计学习方法，深度学习等等。由上文可知，由于基于日志的异常检测算法的特殊性，在进行模型训练和模型建立前，会首先将无结构的日志数据转变为结构书，接着使用特征提取的方法从结构化的日志数据中提取特征，接着使用数据挖掘的一些算法进行模型建立，因此，一个好的基于日志的异常检测算法，大多是由日志解析、特征提取、异常检测等方法共同作用的，下面两部分主要介绍特征提取和异常检测算法等方法。

2.3.2特征提取

由1.2可知，在进行异常检测算法之前，会首先从结构化的日志中提取特征，再使用这些特征进行异常检测模型的建立。为了进行特征提取，我们首先需要将日志数据分为多组，每一组表示一条日志序列，窗口方法通常被用来将原始日志数据集划分为不同的有限的帧，这里主要介绍三种窗口技术：

固定窗口：固定窗口和滑动窗口都基于时间戳，时间戳是每条日志中用来记录事件发生事件的部分。每一个固定窗口有固定的长度，意味着有固定的时间域，一般指一个小时或者一天。因此，固定窗口的个数依赖于事先定义的固定窗口的长度，发生在同一个固定窗口的日志被看成一个日志序列。

滑动窗口：和固定窗口不同，滑动窗口包含两个属性，窗口大小和步长，比方说，每五分钟滑动的小时滑窗。通常情况下，步长小于窗口大小，步长是指滑动窗口前进的距离，因此不同的滑动窗口之间往往存在重复的时间段。对于同样的日志数据和同样的窗口长度，滑动窗口的个数往往多于固定窗口，滑动窗口的个数同时取决于窗口大小和步长。在同一个滑动窗口发生的日志被视为一个日志序列，由于重复时间段的存在，不同的滑动窗口会有重复的日志。

会话窗口：和固定窗口和滑动窗口不同，会话窗口基于标示而不是时间戳。在一些日志数据中，标示被用来标记不同的异常路径。比方说，HDFS日志使用block\_id来记录某一个block的分配、写入、复制和删除。因此，我们可以根据标示将日志进行分组，每一组有自己的标示。

2.3.3异常检测

根据使用的数据集类型和采用的机器学习方法，异常检测方法通常可以分为两大类：有监督的异常检测算法和无监督的异常检测算法。有监督的方法需要明确标注正常实例和异常实例的训练数据集，接着使用分类方法建立模型，模型的建立以最大化区分正常数据和异常数据为目标。无监督的方法不需要数据标注，这种方法通常将和其他实例相比非常不同的实例作为异常实例，聚类算法通常作为典型的无监督的算法应用在异常检测中。

有监督的学习是指衍生于有标注数据的的机器学习任务。对于有监督的异常检测算法，标注了异常和正常的日志数据是必需的，一般情况下，越多标注的训练数据，检测效果就会越准确。这里介绍一些被用来用于异常检测的有监督算法：

逻辑回归：逻辑回归算法是分类模型的一种。在异常检测中，为了判断一个实例是否异常，逻辑回归算法会使用逻辑函数分别计算该实例属于正常数据和异常数据的概率，逻辑函数是通过训练数据建立的。

决策树：决策树是使用分支预测的方法来确定一个实例是否异常的方法，决策树的建立是通过训练数据自顶向下建立的，每一个树节点是使用当前最好的属性建立。决策树首先在文献[62]中被用于在网络请求日志中进行异常诊断。

支持向量机：支持向量机是通过在异常数据和正常数据之间建立最大分割超平面来进行分类的，超平面的建立是通过凸优化的方法。Liang等人[63]使用支持向量机技术来进行异常检测，并将该方法于其他方法进行了比较。

LSTM：长短期记忆网络(LSTM, Long Short-Term Memory)是一种时间循环网络，是为了解决一般递归神经网络中存在的长期依赖问题而专门设计出来的。文献[15]使用lstm构建了一种实时的异常检测和特征操作算法。

无监督学习是在没有标注的数据集上建立的，真实的情形是，我们的生产环境一般无法产生有标注的数据，因此无监督算法也被大量使用于异常检测算法，这些算法包括聚类分析，关联规则挖掘，主成分分析等等。

日志聚类：Lin等人在文献[5]中设计了一种基于聚类的异常检测算法LogCluster来在线地对系统问题进行检测。LogCluster包括两个训练过程：知识库初始化阶段和在线学习，因此，训练数据集也被分为相应地两部分。包括三步：日志向量化，日志聚类，向量提取。首先对日志序列进行向量化，接着LogCluster使用层次聚类将日志向量分别聚成两类：正常和异常，最后选择每个类的类中心作为代表的日志向量。在线的学习过程将会对每个类进行评估，经过在线的学习过程，算法LogCluster将会进行异常检测的步骤。

主成分分析：主成分分析是一种被广泛用于数据降维的统计学习方法。主成分分析的主要思想是将高位数据转换为新坐标系下的K个主成分(K小于原先的维度)。Xu等人在文献[16]中首先使用主成分分析在基于日志的异常检测中，在他们提出的算法中，每一个日志序列作为一件事件技术向量被向量化，之后，使用主成分分析生成两个子空间，分别为正常子空间和异常子空间，正常子空间是由前面k个主成分组成，异常子空间是由剩余的(n-k)个主成分组成，其中n是原始维度。对于一条日志，计算该日志的计数向量在异常子空间上的投影，如果该投影长度超过一定阈值，对应的事件技术向量会被认为是异常的。

不变性挖掘，程序不变性是指在系统运行中不随输入和负载改变而改变的线性关系，某文献 首先使用不变性挖掘技术在基于日志的异常检测中。具有相同会话id的日志(比方说HDFS中的block id)通常表示一个会话的运行流，直观地，不变性挖掘可以从多种日志事件中发现线性关系，线性关系在现今系统事件中经常出现，比方说，正常情形下，在一个文件打开后要被关闭，因此打开文件的日志和关闭日志的日志经常会成对出现，如果打开文件的日志数量和关闭文件的日志数量不等，则是违背了线性关系的异常。为了找到不变性(线性关系)，不变性挖掘通常包含三步。不变性挖掘的输入是由日志序列生成的事件计数向量，其中每一行都是某一事件的计数向量。首先，使用奇异值分解生成不变子空间，不变子空间决定了不变性的数量r，第二部通过暴力搜索查找不变性，最后，通过比较在第二部中找到的候选不变性支持值与某个阈值的大小来决定是否是真的不变性。这一步将会一直持续知道r个不变性全部找到。在基于不变性挖掘的异常检测中，对于一个新的日志序列，我们通过检查其是否尊沙鸥不变性关系来决定其是否异常，如果违反了不变性规则，则我们视其为异常。

对于有监督方法，异常检测需要有标注的训练数据，相对于其他的有监督算法，决策树的解释性更强，逻辑回归可以解决线性不可分的问题，同时支持向量机在使用核函数的情况下也可以解决线性不可分，但是支持向量机的超参数更多，需要人工调整。无监督算法由于其不需要有标注的数据而更为具有实用性，日志聚类利用了在线学习的思想，因此更适用于处理大型日志数据集，不变性挖掘技术不仅可以准确地进行异常检测，同时也可以为异常检测提供简明易懂的解释，然而不变性过于耗时。主成分分析比较难以解释且对数据集较敏感，因此其异常检测的准确性随数据集的变化而变化。

2.3.3评价准则

基于日志数据的异常检测算法通常评价指标有以下几项：

⑴ 准确度。异常检测其实可以归结于一个二分类问题，对于任意一个时间点，预测错误都可能会造成一些难以弥补的后果，因此异常检测算法的准确性时首要的。二分类算法准确性的评价指标本身有很多，通常会有精确率(precision)、召回率(recall)、F1-score、ROC和AUC等等。

⑵ 时间复杂度。很多异常检测算法在设计时有时会过于考虑准确度，因此会用非常复杂的模型来建模，这导致了一个比较严重的问题就是算法的时间复杂度很高。可是真实的运维场景对算法结果的时效性有很高要求，另一方面，任何算法都要有时间效率的考量，尤其在基于日志数据的异常检测算法中，由于日志数据量较大，在对日志的结构化处理中，需要进行大量的字符操作，本身已经是一个比较耗时的算法，因此时间复杂度应该成为衡量一个基于日志数据的异常检测算法的重要标准。

⑶ 算法超参数。通常的回归和分类算法都需要设置某些参数，比方说K近邻算法中的K值，参数取值通常会对算法效果产生决定性的影响，异常检测算法也不例外，一个模型如果受影响的参数个数较少，且模型效果受参数大小设置影响较小，那无疑时一个更稳定的模型算法。

2.4 本章小结

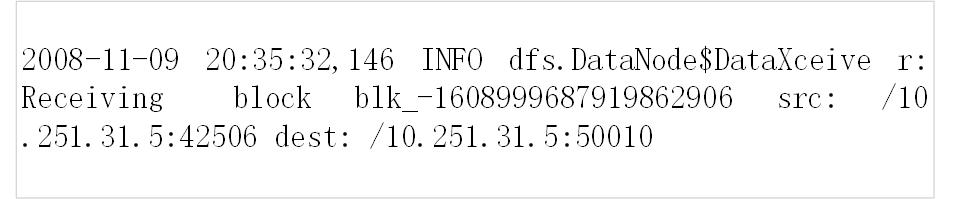
本章首先介绍了基于日志数据的系统异常检测方法的一般流程，具体包括日志搜集、日志解析、特征提取和异常检测，本章接着对各个步骤使用的具体方法进行了介绍，最后给出了每个步骤的大致评价机制。

**第3章 基于加权编辑距离聚类的日志解析算法**

3.1 问题描述与符号定义

3.1.1 问题描述

从系统中安装的应用里产生的日志是独立的文本数据，这些文本数据中包含了发生在系统中的事件。一般来说，一个日志消息，图3.1中显示的一条日志，记录了一个特定的事件，同时包含一些属性：时间戳（记录事件的发生时间），信息级别（表示事件的严重程度，比方说INFO），原始信息内容（记录系统运维期间具体发生的事件）。

图3.1 包含时间戳、信息级别、原始信息内容的日志

一般原始的信息内容格式由源码决定，比方说对于以下的一行C语言代码：

可以产生如下的日志信息：

这四条日志消息可以形成一个簇，或者在系统日志中可以称为一个消息模板，这个消息模板可以由下面的信息类型所表示：

其中通配符“\*”表示变量。

由例子可知，原始消息内容可以被分为两部分：常量部分和变量部分，但是考虑下面的一个例子“Link 1 is up,” “Link 1 is down,” “Link 3 is down,” “Link 4 is up.”。逻辑上的事件模板应该是“Link \* is \*,”但是从分析的角度上看，“Link \* is up” and “Link \* is down”更应该作为消息模板。还有很多其他的日志例子，这些例子从不同的角度看或许会产生不同的消息模板，但是，大部分来说，消息模板应该对应同一条print语句。

从上文中的描述可以得出，日志解析的主要作用就是从大量的日志消息中提取消息模板，一条消息模板对应于一个事件类型，因此，事件日志中的每一条事件都对应于一个事件类型，按照时间顺序排列即可以得到一个事件序列。

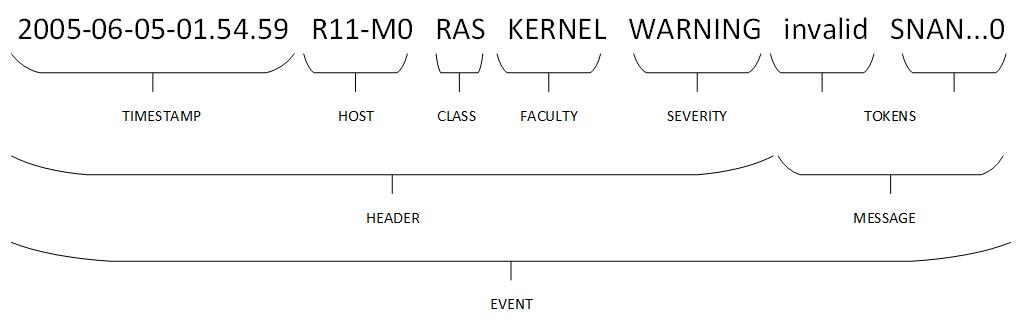
一般来说，提取事件模板所面临的主要挑战一是数据量大，二是定义事件的源代码不好获得，三是对消息模板/事件类型的定义比较主观，并没有比较官方的对事件类型的定义。

本文为简单起见，定义消息模板/事件类型即为源代码中定义的一条打印语句。同一条打印语句打印的事件具有高度的相似性，因此日志提取的方法大多集中于聚类和生成式算法，我们根据搜集到的日志数据类型，选择一种结合聚类算法和生成式算法的日志提取算法LKE(Log Key Extraction)。

3.2.1 符号定义

我们在这个小节给出LKE日志解析算法的相关符号定义。

**事件日志** 定义事件日志为对系统内发生的事件或计算机系统上的应用程序跟踪记录的文本。

图3.2 日志标注

**事件** 事件为事件日志中一行独立的文本，详细地说明了系统或计算机系统上地应用程序发生的一次事件。

**消息** 消息是指事件中除去时间戳、标签、源等信息，只记录发生的具体事件信息的文本序列。

一个事件通常不仅仅包含一个消息，还有其他信息包括日期，源和标签等。对于消息模板/事件类型提取，我们只关心事件中的消息部分，如图3.2所示

**标记** 在一个事件中消息部分被空格划分的独立的单词定义为标记。例如，在图几

**消息长度** 在一个事件中消息部分独立的标记的个数。

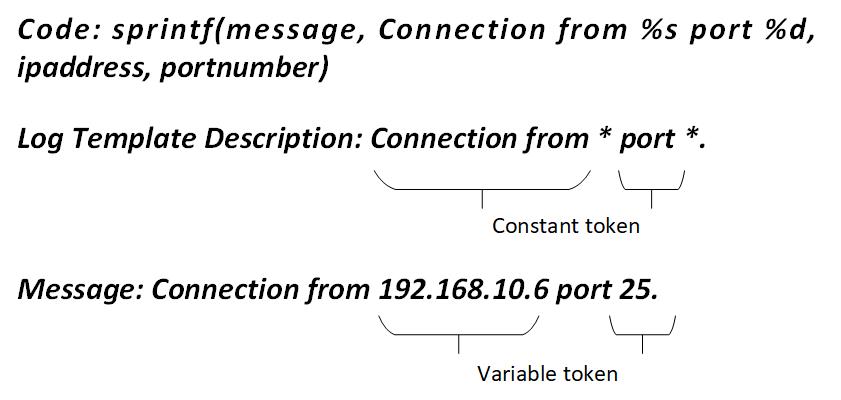
**消息模板/事件类型** 消息模板是指由同一条print语句生成的事件日志中的消息字段。由于确定消息模板具有一定的主观性，人类可能会将一个消息模板生成的事件视为由不同的消息模板产生，或者将由不同的打印语句产生的事件视为由同一个消息模板产生。同样的print语句也可能出现在代码的不同部分，从而产生具有相同消息模板的不同事件。但是，在这里我们认为这些情况相对较少，因此为了简单起见，我们就使用这样的定义。

图3.3 消息模板描述示例

**消息模板描述** 我们定义消息模板描述为可以表示消息模板所有成员的包含通配符的文本模板。如图3.3所示，一般一条消息模板描述对应着一条打印日志语句。

**标记常量** 标记常量为出现在一个事件中的消息部分但是在消息模板中并没有被通配符表示的标记。如图3.3中所示，标记常量为Connection, from和port。

**标记变量** 标记变量为出现在一个事件中的消息部分并且在消息模板中被通配符表示的标记。如图3.3中所示，标记变量为192.168.10.6 和25。

**事件序列** 事件日志中的每一条事件按照事件戳顺序排列即得到事件序列

3.2 基于LKE的日志解析算法

3.2.1 算法概述

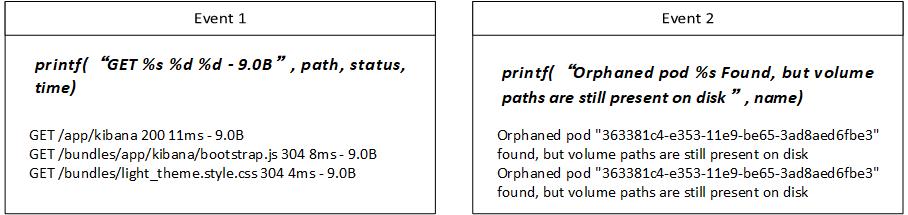
日志解析的主要困难一个是当我们拿到了很多的事件日志，我们并不知道其中有哪些是由同一条打印语句打印的，并且我们也并不知道事件中的标记哪些是常量，哪些是变量。但是，由同一条打印代码的事件之间高度相似，而由不同打印语句打印的事件却很不相同。根据这个结论，我们可以使用聚类的方法来对所有事件进行分组，接着通过查找这些事件中的公共部分进行事件模板的提取。

对标记序列的聚类算法，通常会大量重复地进行距离相似性的计算，一是为了算法的运行效率，再是为了可以将日志解析的结果适用于之后的异常检测，我们在聚类算法的前一步，首先进行对事件的预处理，该处理包括计算每个事件的长度，使用正则表达式将每个事件中的某些明显的变量标记替换掉，之后将相同长度下的事件分为一组。接着在每一组中使用聚类算法，将相似的消息分成一个类，由于此时同一个类中的消息可能还是由不同的打印语句生成的，在聚类之后需要在每一个类中进行再一次分组使得在这一步操作后，每一个组对应一条独立的打印语句，并在最后生成消息模板/事件类型。

3.2.2 步骤一 事件分组和标记变量替换

我们在这一步中首先将事件按照事件长度分为若干组，并使用正则表达式将事件中的变量参数进行替换。我们首先介绍这一部分算法的大致思想，并将该思想使用伪代码进行描述，具体如表3.1。

事件长度定义为事件中独立的标记的数量，我们在这里认为，同一条打印代码打印出来的消息应该是相同长度的，这与代码的性质有关，如图3.4所示，Event1和Event2的打印代码定义的消息长度是不同的。另外由于我们在提取事件模板时是使用聚类的方法，聚类中进行的大量重复的操作是计算两个标记序列的距离相似度，我们认为具有相同长度的事件相似度更大，另一方面，在聚类之间进行分组是对数据规模的缩小，可以大量减少聚类算法的计算量，有利于提升整体算法的运行效率。

图3.4 消息模板打印代码示例

通过对大量的日志事件进行观察发现，一些标记变量通常都是以数字，URI，IP地址等等的形式出现，或者这些标记中包含了大括号，中括号，圆括号，再或者标记中有下划线，斜线，反斜线等等。这些标记在事件消息中非常容易辨认，可以使用规则定义的方法进行识别。因此，在这一步中，除了计算事件长度进行分组之外，我们通过定义显式的规则表达来描述这些典型的标记变量，并将这些标记变量使用空来表示。这一步骤之后，我们认为剩下的标记为候选的标记常量。如图几所示，事件日志中的IP地址，数字，或者文件目录等等，都从事件中识别出来并标记为空。

我们将这一步的算法称为MessagePreprocess，将得到的相同长度的消息分组称为MessageMaps，算法MessagePreprocess每读取一条消息，记录消息中的token总数量作为消息长度，并将其中的每一个token与预先定义的规则作比较，满足规则的token用通配符替代，具体的伪代码如表3.1所示。

表3.1 算法MessagePreprocess

|  |
| --- |
|  |
|  |

3.2.2 步骤二 聚类

每一条消息都可以看成是标记的有序序列，在第一步中，我们得到的MessageMap中每一个组的消息长度都是相同的，在这一步我们使用聚类算法将每一组中的消息进行聚类。

我们首先使用加权的字符串编辑距离来计算两个消息和之间的相似度，加权的字符串编辑距离定义如下：

 (4)

其中EO为到的操作次数，是第次操作时单词的索引，是控制权重的超参数。

我们将相似的消息聚为一类，对于两个不同的消息，如果它们之间的加权编辑距离小于阈值ζ，我们认为它们之间具有联系，通过计算每两两之间的距离，所有之间具有联系的消息成为一个组group。

对于上述的聚类算法，阈值的确定是较为关键的一步，我们在这里给出一种阈值的确定方法。通过计算消息两两之间的加权编辑距离，我们可以得到每对消息的距离，将这些距离使用kmeans算法分成两类，选择其中类中心较小的类中的最大距离作为阈值ζ。

第二部完整的算法MessageCluster，对于MessageMap中的每一个value，计算其中每两两消息之间的距离并使用k-means算法将距离分类，拿其中较小的类中心代表的类中最大的距离作为阈值，所有距离小于阈值的消息之间视为有联系的，所有有联系的消息为一个组。MessageCluster算法的伪代码如表3.2。

表3.2 算法MessageCluster

|  |
| --- |
|  |
|  |

3.2.3 步骤三 类中分组

经过上两步的两次分组，理想的情况下，每一个类别会对应一个事件模板，在这种理想情况下，我们可以通过从这些事件中提取公共部分作为事件模板，其余的非公共部分作为标记变量用通配符代替。但是，在一个类中的事件消息并不一定对应着同一条打印代码，因此，我们在这一步进行类中分组将同一类中的事件再重新进行分组。

对于一个MessageClusters，设这一个cluster中消息的消息长度都为N，其中这些消息的公共标记有M个，这M个公共标记将每一条消息划分为M+1个位置，对其中的一个位置，我们可以得到一个MessageCluster中每个消息在这个位置的标记序列，空用0表示，统计在这个位置上标记序列一共有多少不同的值V，一般来说，V的值越多表示这个位置越不可能是标记常量，我们在这里同样引入阈值，当V超过阈值时，我们认为这个位置的标记是标记变量，并且不再进行划分组的操作，当V不超过阈值时，我们认为这个位置的标记都是标记常量，并将这个MessageCluster划分为V+1各组。

3.2.4 步骤四 消息模板生成

经过上面的几个步骤，我们认为一个Message Group中可以由一个消息模板描述来表示，通过将一个Message Group中所有消息的公共部分作为标记常量，其余的标记为标记变量，统一用通配符表示，我们可以得到一个Message Group的消息模板描述。

通常来说，一条日志对应一条消息，一条消息对应一个消息模板/事件类型，因此，如果我们搜集道德日志数据较为完备，通过日志提取算法可以得到所有的消息模板/事件类型，任意一条日志都对应一个事件类型，我们就可以得到事件类型的序列。

3.2.5 新消息的消息模板

步骤1到4是我们在日志训练集上提取消息模板/事件类型的步骤，当我们得到所有的消息模板之后，对于新的一条消息，首先使用正则表达式将其中符合规则的标记替换掉并统计消息长度，将其和该长度下的消息模板描述一一匹配，对于匹配成功的消息模板描述，我们将其对应的事件类型归为该消息的事件类型。如果没有匹配的消息模板描述，我们认为该条日志消息为异常的消息。

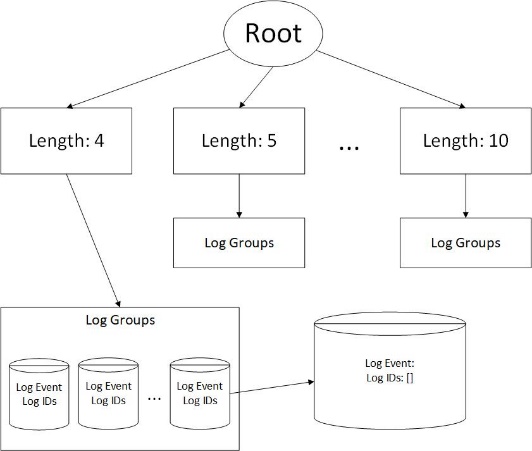
为了简化这一步的查找效率，我们引入一个深度为3的树结构模型来存储之前在日志训练集中提取的消息模板/事件类型，如图3.5，我们称这个树结构为消息模板树，根节点的子节点为消息长度，消息长度的子节点也是树的孩子节点为消息模板/事件类型，对于一条新的消息，我们会计算它的消息长度，再根据正则表达式替换其中的符合规则的标记，与消息模板树孩子节点中的消息模板一一比较，直到找到匹配的消息模板，如果找不到，则视其为异常日志消息。

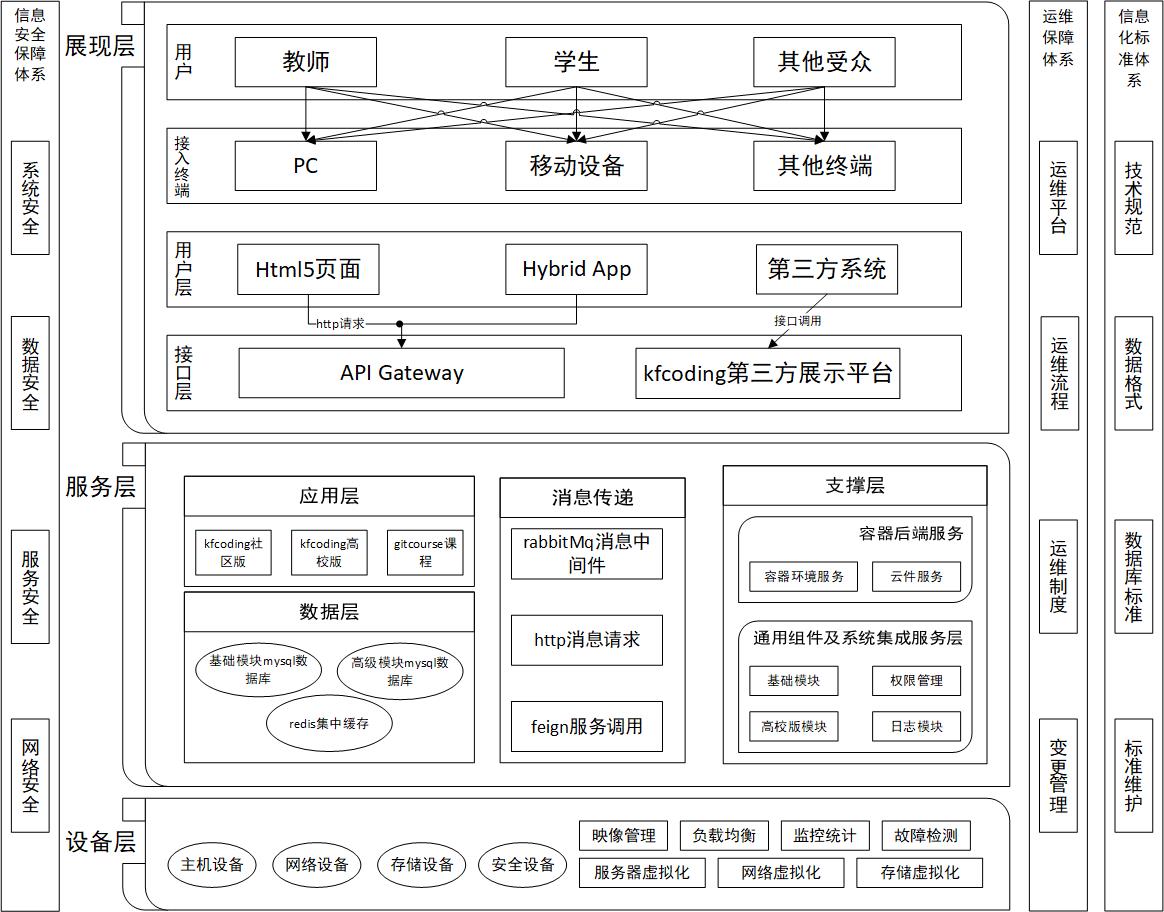
图3.5 存储消息模板和事件类型的树结构模型

3.3 实验结果与分析

3.3.1 数据准备

本文的最原始日志数据来源于在线实训平台功夫变成kfcoding，kfcoding是一个一站式IT实训服务平台，网址为<http://kfcoding.com/>，包括开源大学、内容创作、极速体验、学校教学、实训行、少儿编程六个子模块，提供了包括git、python、数据科学、云计算等等多种课程的在线实训教程。

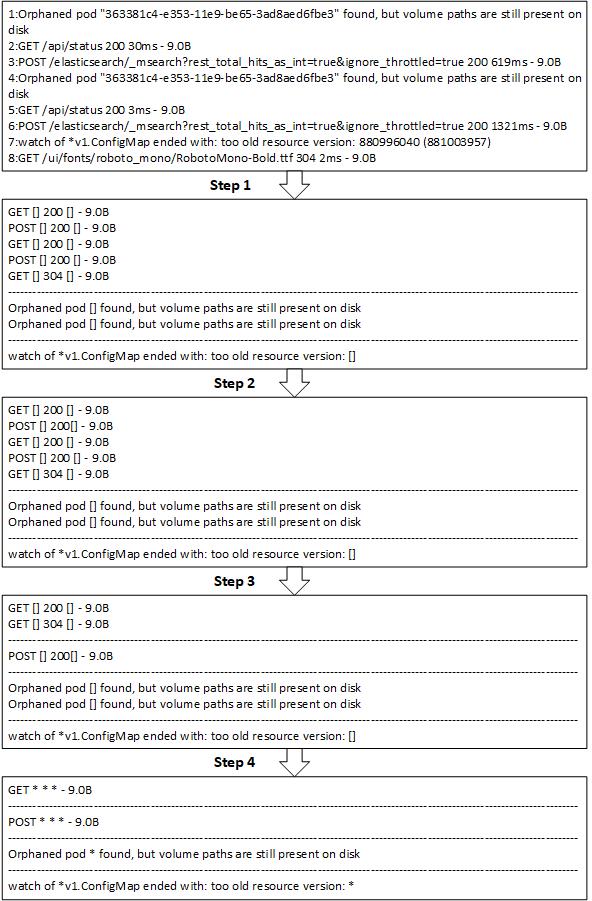
Kfcoding采用微服务架构思想构建，整体架构设计图如图3.6,我们在此对kfcoding的整体架构做简单介绍。该平台可以分为以下三个层面，第一层为展现层，即前端，为教师、学生或者其他用户展现的个人学习界面、学校教学界面、实训活动界面等；第二层为服务层，这一层包括应用层、数据层、消息传递和支撑层四大模块，展现层通过API Gateway或者kfcoding第三方展示平台调用服务层的服务，第三层为设备层，即各种硬件设备和虚拟化等服务。

图3.6 功夫编程kfcoding架构设计图

3.3.2 实验结果

我们按照时间顺序从kfcoding系统中搜集系统日志并使用3.2节中描述的方法进行消息模板的提取并构建消息模板树，其中生成消息模板的过程本文用比较典型的日志事件并将具体的过程用图3.7表示。

Kfcoding以大约平均每小时3600条日志的速率产生日志事件，我们通过拿出一个月的日志量进行日志提取，整体的日志数据经过步骤一到步骤四得到的输出结果为消息模板和事件序列，其中消息模板会被用来生成消息模板树，以对新的日志消息进行异常日志的检测和新的事件序列生成。

图3.7 日志解析结果示例

日志解析的输出之一事件序列会在第四章中特征提取章节用来生成特征，由于事件在时间上具有先后关系，我们也会使用具有时间先后关系的窗口技术进行特征提取，将基于日志数据的异常检测问题抽象为时间序列的异常检测。

3.4 本章小结

本章研究基于日志数据的系统异常检测算法，原始的系统日志是无结构化的文本数据，每一条打印的日志文本，内容由系统中的日志打印代码决定，一般来说，日志挖掘技术的第一步是将无结构化的日志数据结构化，及将原始的日志数据转化为消息模板和事件序列，一般来说，一个消息模板对应着一条打印代码，事件序列可以用来提取重要的特征，为之后的数据分析和数据挖掘提供数据支持。

本文结合分布式系统日志数据的主要特点，结合具体系统的需求，并对原始的日志数据进行分析之后，给出了一种基于聚类算法的日志提取方法，可以有效得解析出日志数据中消息模板和事件序列，另外，本章给出了一种固定深度得树结构模型，用来储存生成的消息模板，这种数据结构可以用来检测异常日志消息并快速地对新的日志消息进行结构化。本章给出的日志解析算法被用于特定分布式系统的日志数据，生成的结构化日志被用于之后的章节进行特征提取和系统异常检测。

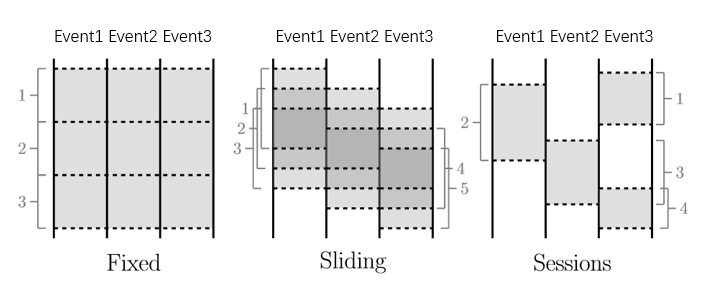
**第4章 基于固定窗口的日志特征提取算法**

4.1 问题描述与符号定义

4.1.1 问题描述

我们在本小节主要介绍从解析后的日志中提取特征的技术。在基于日志的异常检测工作流中，提取特征通常采用窗口技术。

窗口可以将一个数据集分为有限块，每个块作为一个组进行处理。窗口技术实际上总是基于时间的，其中窗口中按顺序排列的元素具有连续增加的逻辑时间戳。窗口可以是对齐的，即应用于有关时间窗口的所有数据，也可以是未对齐的，即仅仅应用于给定时间窗口的特定数据子集。我们这里给出有关处理带有时间戳的日志数据的三种窗口技术，如图4.1所示。

图4.1

固定窗口（也称翻滚窗口）的窗口大小是固定的，通常是一个小时或者是一天，固定窗口一般都是对齐的，即每个窗口在相应的时间段内应用于所有数据。为了使窗口负载在时间上均匀分布，有时会通过将每个关键点的窗口相移一些随机值而不对齐。

滑动窗口由一个窗口大小和滑动期共同定义，比方说每分钟开始的小时窗口，一分钟是滑动期，一小时是窗口大小。一般滑动期比窗口大小要小，滑动期指窗口间的重叠部分，滑动窗口通常也是对齐的，虽然图4.1中的滑动窗口给人一种滑动的感觉，其中所有的窗口都将应用于图中的所有三个键，固定窗口实际上是滑动窗口的一种特殊情况，固定窗口的窗口大小等于滑动期。

会话窗口是在数据集中捕获某个活动周期的窗口。在基于日志数据的会话窗口中，一个活动周期是指包含日志事件的一个时间段，通常会话窗口是由超时间隔来定义的，在一段时间内发生的任何小于超时时间的事件都会被分组为一个会话，会话是未对齐的窗口。如图4.1，窗口2仅适用于事件1，窗口3仅适用于事件2，窗口1和4仅适用于事件3。

在定义了窗口技术之后，我们可以得到在窗口大小中发生的事件序列，并可以从中提取事件日志特征，作为之后的异常检测算法使用。文献[4]通过搜集包含异常情形的日志进行分析以检测甚至预测异常日志，搜集的包含异常行为的日志时间跨度为四小时，其中三小时为在异常发生之前，一小时在异常发生之后，其中对这些日志进行分析的主要特征有在日志中发生的事件类型，在日志中事件的发生频率，以及事件在这一段时间内的分布情况。文献[23]中忽略了日志中的事件戳，只保存了事件的序列，在对系统日志进行建模时，通过将系统日志划分为许多会话来得到一个会话序列，其中每个会话是一系列的事件并使用隐马尔可夫HMM模型进行异常检测。文献[16]通过建立基于时间的窗口，并关注于状态比例向量和时间计数向量两个特征，并使用主成分分析的方法进行异常检测并取得了很好的检测效果。

本文根据日志来源系统的主要特征，考虑到日志搜集的效率和主要功能，采用固定窗口的技术，并在每个固定窗口中主要采集固定窗口中的总体事件频数、事件频数等几种特征作为第六部分异常检测算法分析使用。

4.1.2 符号定义

我们在这个小节给出特征提取算法的相关符号定义。

**事件窗口** 事件窗口定义为一段固定时间长度中的按照时间顺序排列的一条事件序列。

**事件窗口大小** 事件窗口的固定时间长度称为事件窗口的大小。

**总体事件频数** 给定事件窗口大小，一个事件窗口中事件发生总数为总体事件频数。

**事件频数** 给定事件窗口大小，一个事件窗口中某一事件类型发生的数量为该事件类型的事件频数。

4.2 一种基于固定窗口的日志特征提取算法

4.2.1 算法概述

我们在特征提取时，主要关注事件发生的频次，和所有事件发生的总频次，这些特征可以能反映出系统的性能，对于事件之间的相关关系，我们这里较少考虑。固定窗口技术相较于滑动窗口和会话窗口这两种技术来说，一方面可以更为全面地记录到发生的事件，另一方面可以提升特征提取算法的性能。

我们根据经验在这里选取固定窗口的大小为一小时，根据第三章的日志提取方法，每得到一条日志消息，我们可以得到它对应的消息模板/事件类型，并由此得到一个事件序列，通过计算固定窗口中的各种特征，这些特征遵从时间上的先后关系，每个窗口中某一特征的值按时间顺序排列便可得到一条时间序列，我们便可以将日志数据的异常检测问题抽象为时间序列的异常检测问题。

4.2.2 特征提取

考虑到本文的整体目标，即从日志数据中挖掘到可以反应系统性能的关键指标，从而检测到系统异常，我们需要可以显示系统异常的较为通用且比较普遍的异常，因此只针对某一特定系统的特定异常的特征我们并没有考虑。本文主要研究了以下几个比较普遍的特征：整体的事件频次和单独的事件频次。

整体的事件频次是指每单位时间日志事件的总数，这个特征可以大致反应系统的某些现象，提取该特征是出于以下考虑：一般情况下，系统在发生故障之前进程都会处于非常频繁的状态。

单独的事件频次是指某一个消息模板/事件类型在单位时间内发生的次数。一般来说，并不是所有的事件频次都会在系统异常发生时产生明显的波动异常，鉴于这种情况，我们需要那些会在系统异常出现时波动发生明显变化的事件类型作为我们主要研究的对象，筛选的方法有很多，我们在这里主要介绍坡度过滤、窗口最大值过滤和窗口平均值过滤这三种方法。

坡度过滤是比较自然的一种过滤方法，它通过在时间和单独的事件频次之间建立回归方程来确定候选的研究事件，一般来说，当系统异常发生时，事件频次与回归方程预测的值相差较大，我们则认为这个事件类型是可以被用来提取频次来检测系统异常的。

窗口最大值过滤和窗口平均值过滤方法都是基于这样的一种思想，即系统异常发生时的一段时间内，事件频次会表现得比较异常。因此通过提取系统异常发生时事件频次的最大值和平均值，与系统正常运行时事件频次的最大值和平均值，当系统异常时的取值大于正常运行时的取值时，我们认为该事件类型是我们可以拿来提取特征来检测系统异常的。

4.3 实验结果与分析

4.3.1 数据准备

我们利用第三章描述的日志解析方法对原始的日志数据进行解析可以得到消息模板/事件类型和事件序列，每一个事件类型都对应一条事件序列以及发生的时间点，利用基于固定窗口的特征提取技术，我们这里选定固定窗口的长度为30分钟，可以得到总体的事件频次特征序列以及单独的事件频次特征序列，对于所有的单独事件频次特征序列，我们使用坡度过滤、窗口最大值过滤以及窗口平均值过滤的方法进行特征筛选，我们这里认为，凡是通过其中一项筛选的特征都被作为第五章的时间序列异常检测的研究对象。

一般来说，我们将系统异常发生之前一段时间的数据也视为异常数据，在提取特征之后，我们将系统异常发生时以及发生前十分钟的数据标记为异常。

4.3.1 实验结果

图4.2是总体的事件频次随时间的变化情况，其中时间跨度大约为，单位时间为，由图4.2可以看出，总体的事件频次还是具有一定的周期性，这与本文基于的系统特点有关，由于kfcoding的主要受众是老师和学生，并且主要用于学校教学使用，因此系统总体的事件频次是具有周期性的。异常数据我们用红色点进行标注，对于系统的整体事件频次，异常情形在时间序列中一般表现为序列的忽然升高，这种情形大多发生于系统负荷过高时。

由图4.2可以看出，在系统运行正常时，事件的总体频次会处于一种较为有规律的波动状态，而当异常情形发生时，在时间序列波动的情形中可以看出，异常的时间点或时间序列会表现出在取值上的突然升高，或者波动得更为陡峭，这与系统在负荷过大或者请求过多时特征的表现较为相似。

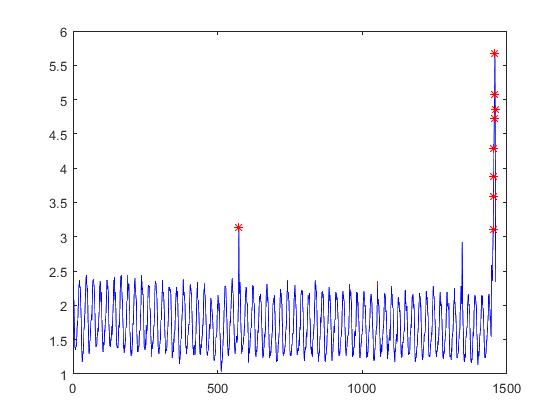
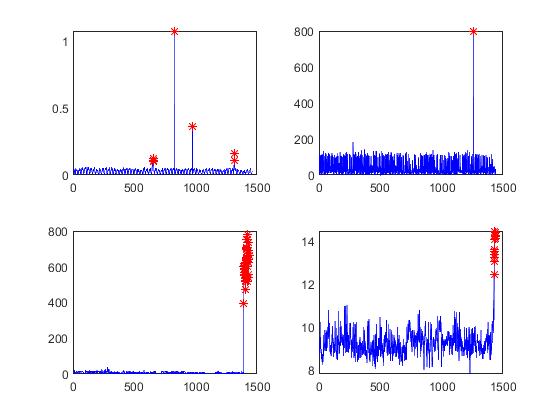
图4.2 事件总体频次的时间序列图

图4.3是比较典型的四个特征的时序图，其中异常点用红色标注，这些特征的异常点或异常序列与正常点相比波动发生了明显的改变，说明我们在本章描述的特征筛选方法是比较有效的，这些特征将会被第五章用来进行异常检测。

图4.3 单独事件频次的时间序列图

4.3 本章小结

本章在前面章节对原始日志数据进行结构化之后的日志数据基础上，进行对日志数据特征的提取，特征提取主要是为了能够从日志数据中抽取重要信息，并为之后的异常检测算法进行数据储备。本文首先介绍了一般基于日志数据的特征提取方法——窗口化技术，接着基于第三章得到的数据类型和特点，并结合系统的具体要求和特点，本文采用固定窗口技术的特征提取方法。由于异常检测需要能具体反应系统运行状态的特征，本文给出了能够提升异常检测算法效果和性能的特征选择算法，本章节的特征提取算法应用在第二章所得到的结构化日志数据中，得到的特征以时间序列的形式供第五章的时间序列异常检测算法使用。

**第5章 基于序列相似性比较的异常检测算法**

5.1 问题描述与符号定义

5.1.1 问题描述

在对原始的日志进行日志模板提取，并采用窗口技术对结构化的日志信息提取信息后，我们可以得到按照时间顺序排列的一系列特征序列，如果我们限定一次只考虑一种指标的变化以及异常情形，我们可以将这种问题抽象为单变量时间序列的异常检测问题。

时间序列即是按照时间顺序排列的一组随机变量，我们在日志分析工作流的日志解析和特征提取部分，提取到的一系列特征即是我们需要分析的随机变量，窗口本身是由时间顺序的，因此将每个特征按照窗口本身的时间顺序排列即可以得到每个特征的时间序列，通过统计机器学习和深度学习等等方法，我们可以发现时间序列中的异常序列，并及时给出警告。

时间序列中的异常检测是指在时间序列数据中找到与其周围点表现异常的一个点或一段序列。在真实的情况下，异常点或异常序列一般代表着异常情况。智能运维背景下的基于日志数据的异常检测主要涉及到的真实异常情况主要由网络恶意攻击，网络或内存负载过大等等。针对时间序列异常检测的算法主要是通过对正常情形下的时间序列数据进行建模，若当前观测点与正常数据差异过多时将被视为异常点。基于这一思想，数据挖掘中的许多模型算法被应用在检测时间序列异常中，比方说线性回归、支持向量机、神经网络等等。这些算法通常将时间序列数据集分为训练集和测试集，在训练集中训练得到模型并在测试集上进行检测。

本文的异常检测针对日志数据中提取到的各种特征，在第三章中的特征提取章节，我们给出了事件窗口和各种特征的定义以及计算方法，由于事件窗口在时间上具有先后顺序，并且相邻的事件窗口之间没有缝隙，每个特征是从每个事件窗口中提取到的，将这些特征的值按照时间顺序进行排列便可得到针对某个特征的时间序列。我们将基于日志数据的异常检测方法抽象为从日志数据中提取特征并生成时间序列，从时间序列中检测异常序列的问题。

结合日志数据衍生的时间序列特点，我们选择一种基于序列比较相似度的异常检测算法，应用滑动窗口机制，通过计算时间序列中当前时间子序列的最小非几距离，从而衡量当前时间子序列的异常性，当异常性超过给定阈值则判为异常。

5.1.2 符号定义

我们在这个小节给出异常检测算法的相关符号定义。

**时间序列** 时间序列定义为按照时间顺序排列的一组随机变量。

**时间序列点异常** 时间序列中的点异常是指时间序列中与其周围点差异较大的数据点。

**时间序列序列异常** 时间序列中的序列异常是指一段时间内的观察值与其周围序列差异较大。

**子序列** 给定长度为n时间序列，定义时间序列中的一段子序列为以某点向前展开长度为l的一段连续点的集合。

**滑动窗口** 给定长度为n的时间序列以及以点x向前展开长度为l的子序列，定义滑动窗口为

**非几配对** 给定长度为n的时间序列以及时间子序列和滑动窗口，定义滑动窗口中与给定时间子序列没有公共点的时间子序列为给定时间子序列的非几配对。

**距离** 给定非空集合S，对于S中任意的两个元素x与y，按某一法则都对应唯一的实数d(x, y)，而且满足非负性、对称性和三角不等式三条公理。

**最相似非几配对** 给定时间子序列以及滑动窗口，与给定时间子序列距离最小的非几配对为给定时间子序列的最相似非几配对。

**异常性** 给定时间子序列以及滑动窗口，给定时间子序列与其最相似非几配对之间的距离为给定时间子序列起始点的异常性。

**异常点** 给定时间子序列和阈值K，定义时间子序列的起始点为异常点当且仅当该起始点的异常性大于阈值K。

5.2 一种基于序列相似度比较的异常检测算法

5.2.1 算法概述

通过对日志数据生成的时间序列观察发现，这些特征序列波动较为频繁，有些事件特征波动具有比较明显的周期性，而有些特征波动非常不具有规律性，另外当异常行为出现时，数据表现出的异常形态也有所不同，时间序列中的异常点通常相较于其周围点表现出取值的突然增大或突然减小，而异常序列通常会表现出一段点的走势相较其周围点（尤其是前面的点）有所不同。

根据上述异常数据的特点，我们选择一种基于序列相似度比较的时间序列异常检测算法，主要思想是对给定的时间序列中的没一点，计算由该点向前展开的子序列在滑动窗口中的异常性。一般来说，异常数据的异常性相较于正常数据会明显增大，我们的算法利用某一特定的统计分布来拟合时间序列中各点向前展开的时间子序列在整条时间序列中的异常性，当异常性偏离该分布中心较大时，该点被视为异常点。

5.2.2 滑动窗口中的异常性度量

算法1给出了利用滑动窗口机制计算时间序列中各点异常性的计算方法，伪代码如下.

表5.1 算法AnomalyCalculate

|  |
| --- |
|  |
|  |

AnomalyCalculate算法计算时间序列中各点异常性的具体方法是，通过给定长度为n的时间序列、滑动窗口长度、子序列长度，那么对时间序列中所有下标在区间的点即，有以点向前展开长度为的子序列以及以点向前展开长度为的滑动窗口，其中，在中找到与欧氏距离最小的非几配对子序列，即最相似非几配对，将与之间的欧式距离大小即作为点的异常性，同定义8的描述，在这里，为了防止当前点与之前的异常点做比较，算法1限定最相似非几配对均为正常点。

通过算法1计算给定时间序列中各点的异常性后发现，当异常点或者异常序列出现时，异常性会明显增大。

5.2.3 阈值选择机制

由前面分析可知，基于序列相似度比较的异常点或者异常序列出现时，异常性会明显增大。实际情形下得到的运维数据是流式数据，仅仅知道异常性增大是不够的，增大到多少才算异常，或者增大到多少需要报警，这需要对时间序列中的每个点的异常性确定阈值，本文提出了两种阈值确定方式。

假定一段时间序列中每个点的异常性服从一定分布，异常点或者异常序列的异常性取值较大因此会落在该分布的分位点之外，当异常性落在该部分时将被视为异常点或异常序列。可以使用正态分布和对数正态分布来分别拟合时间序列中各时间点的异常性分布情况。

本文使用均值和方差的递归计算方法[14]对分布的均值和方差进行估计，计算如下：

 (3)

对于新元素的加入，本文在递归的均值和方差计算公式中引入遗忘因子，则引入遗忘因子后，均值和方差的计算如下：

 (4)

引入遗忘因子后的递归计算方法如下：

 (5)

5.2.4 完整算法

5.2.1小节中给出的算法1为按照时间顺序依次计算给定时间序列中下标在区间内的各点异常性，利用2.3小节中给出的两种拟合分布，以及均值方差的递归计算方法，可以在计算时间序列中各点异常性的同时，对两种拟合分布的参数进行在线估计以及更新。由于最先开始递归计算的均值和方差波动较大，因而可能导致拟合分布的不稳定性，影响检测效果。因此，完整的算法引入了一段过渡期，在过渡期内不进行均值和方差的更新。均值和方差的初始值是由在过渡期内点的异常性计算的，过渡期之后的均值和方差由2.3节中的公式(3)或公式(5)更新。

算法2（AnomalyDetect）给出了在计算时间序列中各点异常性的同时，对异常性分布进行拟合并估计参数，以及划定阈值并判定给定点是否异常的具体步骤，具体下所示。

表5.2 算法AnomalyDetect

|  |
| --- |
|  |
|  |

算法2是在算法1的基础上加入了异常性分布拟合、参数估计和划定阈值，进而进行异常检测的完整过程。图2展示了算法2的具体实现过程，如图2所示，对于由当前点向前展开的时间子序列，算法2在由当前点向前展开的滑动窗口中寻找它的最相似非几配对，计算两子序列的欧式距离作为当前点的异常性，并计算该异常性在拟合的异常性分布中偏离中心的程度，当偏离过多时，当前点被认为是异常点，同样，为了防止与异常点比较相似性，这里限定最相似非几配对均为正常点。

5.3 实验结果与分析

5.3.1 数据准备

由第四章，我们从搜集到的日志中抽取到整体事件频数、事件频数、事件分布等等特征，这些特征的值按照时间顺序即形成时间序列，根据第四章描述的特征筛选方法进行特征筛选得到一系列时间序列，其中系统异常时期的数据会被标记为异常。

5.3.2 评价准则

时间序列中的异常检测问题可以看为一个典型的二分类问题，时间序列中的每一点，再真实情况下被标记为正常或者异常，同时也可以被预测为正常或者异常，根据真实情况和预测情况可以被标记为TP、FN、FP、TN，准确率precision和召回率recall的计算方法如下式：



F1 Score是另一种衡量二分类模型精确度的指标，同时兼顾了准确率和召回率，计算方法如下：



由于时间序列中异常数据相对较少，时间序列异常检测为样本不平衡问题，在正负样本比例悬殊时，PRC(precision recall curve)更能反应分类器的性能，结合本文的实际需求，我们使用precision、recall、F1和PRC这四种评价指标对我们在5.2小节中介绍的异常检测方法进行评价。

5.3.3 实验结果

我们使用AnomalyDetect在第四章得到的时间序列中进行异常检测，按照AnomalyDetect算法的步骤，对异常序列和异常点的检测在一个滑动窗口和过渡期之后进行，在检测的同时，随着均值和方差的波动，异常性的具体分布会有一些变化，在检测的最后，我们可以得到一条时间序列中个点的异常性，以及最终迭代估计出的异常性分布情况。

表5.1给出了利用AnomalyDetect算法在我们基于日志数据得到的四条时间序列数据上的检测准确性，其中每条时序数据均在固定了滑动窗口大小、时间子序列长度、过渡期以及遗忘因子后进行异常值的检测，每条时序数据中实际的检测点均在一个滑动窗口加一个过渡期之后，同时表5.1还给出了四条时间序列的准确率、召回率和F1-Score。由表5.1可以看出，本文的AnomalyDetect算法具有较高的检测准确性，尤其是在四条时序上的F1Score都达到了0.90以上，之所以出现如此高的准确性，是与算法思想有关，因为AnomalyDetect主要基于时间序列相似性比较的原理，并不仅仅利用相邻点之间的相关性，除此以外，AnomalyDetect算法可以实现对异常点和异常序列的通用检测。

表5.1 AnomalyDetect算法在四条时间序列上的检测效果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | Algorithm | TP | FP | FN | Precision | Recall | F1 Score |
| TimeSerie1 | norm | 1543 | 10 | 111 | 0.9329 | 0.9936 | 0.9623 |
| lognorm | 1543 | 10 | 111 | 0.9329 | 0.9936 | 0.9623 |
| TimeSerie2 | norm | 461 | 0 | 5 | 0.9893 | 1 | 0.9946 |
| lognorm | 461 | 1 | 5 | 0.9893 | 0.9978 | 0.9935 |
| TimeSerie3 | norm | 766 | 30 | 37 | 0.9539 | 0.9623 | 0.9581 |
| lognorm | 766 | 29 | 37 | 0.9539 | 0.9635 | 0.9587 |
| TimeSerie4 | norm | 572 | 66 | 315 | 0.6449 | 0.8966 | 0.7502 |
| lognorm | 577 | 68 | 281 | 0.6725 | 0.8946 | 0.7678 |

为了进行时间序列异常检测效果的比较，我们这里给出另外一种开源异常检测算法RPCA与AnomalyDetect算法的PRC（Precision Recall Curve）曲线作为参考，对于不平衡数据来说，PRC相较于ROC更能体现算法的效果，由图5.1可以看出，AnomalyDetect算法的检测效果明显优于RPCA。

图5.1 AnomalyDetect和RPCA算法的PRC

为了可以从多个角度反应本文AnomalyDetect算法的具体效果，由上一小节中对算法的具体描述我们知道，AnomalyDetect算法会具体涉及到几个超参数的设置，算法的具体检测效果与超参数的设置也比较相关，我们在这里给出超参数的大小对算法检测效果的影响。

图5.2显示的是对于同一条时间序列数据，不同滑动窗口大小下的FN、FP、TP，可以看出，各指标随窗口大小的增大会出现不同程度的波动，但是都维持在比较小的范围之内，并且，我们通过对大量时间序列数据进行实验发现，对于周期性比较明显的时间序列来说，滑动窗口的大小至少应该大于一个周期长度，这样可以保证算法的检测效果。



图5.２AnomalyDetect在不同滑动窗口下的检测效果

　　图５．３给出了不同遗忘因子下的检测效果，遗忘因子是在跟新异常性分布均值和方差时使用到的，由图５．３可以看出，遗忘因子对检测效果的影响较大，并且检测效果随遗忘因子的上升而变好，总体来说，遗忘因子取值在０．９５和１之间检测效果较好。



图５.３AnomalyDetect在不同遗忘因子下的检测效果

5.3.4 结果分析

由实验结果可知，本文使用的时间序列异常检测算法具有以下比较明显的优势：

⑴ 准确性，通过F1-score和PRC曲线可以看出本文的算法准确度比较高。

⑵ 通用性，通常来说，一个异常检测算法并不能适用于所有的数据集，但是本文介绍的算法通过比较子序列之间相似度可以同时检测异常点和异常序列，在本文的数据集上表现较好。

⑶ 在线性，AnomalyDetect算法利用滑动窗口的技术，利用滑动窗口的思想可以将比较容易地将异常检测算法变为在线检测，对于当今系统的流式数据来说在线的异常检测更加具有实时性和准确性。

同样，AnomalyDetect算法也有弱点，一个是算法涉及的超参数比较多，比方说滑动窗口大小、子序列长度、初始参数大小等等，由于异常检测的过程中需要不断比较异常性和阈值的大小，并且模型本身也在不断更新，因此参数的设置起到了关键作用，模型的完全自动化还有很大的空间。

5.4 本章小结

本章节在前面两章（日志解析和特征提取）的基础上进行基于日志数据的系统异常检测，由于特征提取章节从结构化日志数据中提取除了可供异常检测使用的特征数据，日志数据本身是基于事件戳的，本文使用的特征提取算法是基于固定窗口的，保证了我们得到的特征数据同样也遵循时间的先后顺序，因此我们将基于日志数据的系统异常检测抽象为时间序列的异常检测算法，这与通常的基于日志数据的异常检测工作流大致一致。

本章在对时间序列数据进行分析后，给出了一种创新的基于序列相似性比较的时间序列异常检测算法，该算法不同于机器学习和深度学习的思想，使用滑动窗口和子序列比较，通过量化每个点或者时间子序列的异常性来检测异常。由于采用滑动窗口的技术，本文的检测算法可以适用于流式数据，同时算法思想的创新性使得该检测算法在时间序列数据上表现出了较高的准确度。

**第6章 总结与展望**

6.1 总结

本文首先对基于日志数据的异常检测的方法进行了简单的介绍，根据现阶段大型软件系统的特点，提出了一种基于系统日志数据的异常检测解决办法，该方法可以有效地从日志数据中检测异常日志并能检测系统异常，主要的流程包括日志解析，特征提取，异常检测三个具体任务。本文针对这三个任务给出了具体的算法流程，包括基于聚类算法和生成式算法的日志解析算法，基于固定窗口技术的特征提取算法，以及基于序列相似性比较的事件序列异常检测算法。

6.2 进一步工作的方向

根据本文实验结果显示，本文提出的基于日志数据的异常检测方法具有较高的准确性和通用性，但是仍然需要进一步的研究工作，简要讨论如下:

⑴ 日志解析方面，由于本文提出的基于聚类算法和生成式算法会涉及到一些人工规则的制订以及大量的相似性计算步骤，因此算法不可以更轻易地移植到其他系统，需要对系统较为熟悉地开发人员和运维人员制订一些简单的规则，另外算法的时间复杂度也可以得到进一步的提升。

⑵ 特征提取方面，本文给出了一些比较典型的特征以及它们的计算方法，但是对于不同的系统，特征的提取应该需要针对这些系统的具体特点以及异常检测的具体需求，本文给出的特征并不针对所有系统的异常检测情景。

⑶ 异常检测方面，针对日志数据的异常检测，本文抽象为时间序列的异常检测问题，这种方法并不适用于所有系统，比方说，对于一些系统，日志数据间会出现极高的相关性，对于这种日志数据，异常检测方法如果使用不变性挖掘也许会更有效，因此，针对日志数据的更为通用的异常检测算法需要进一步的研究工作。

**致谢**

时光荏苒，岁月如梭。将近三年的研究生生涯即将结束，自己在大学七年的学习时光也即将画上句点。感谢大学给我的学习成长的平台，让我褪去曾经的青涩与稚气，一步步变得成熟与稳重。

最要感谢的是我的导师，王伟老师和田春崎老师。这两位老师学识渊博，学术水平高超，为人也和蔼可亲。多年以来，老师们在学术上为我提供了良好的学习环境以及研究平台，每当我在学术上有所困惑时，老师们总能给与耐心指导。此外，老师们也让我知道，在学习工作中，严谨、认真的态度是必不可少的。最近在毕业论文撰写中，特别感谢王老师不厌其烦地指导与反馈。在此，我由衷地感谢这两位老师多年来对我的关心和帮助。

我要感谢张礼庆，沈旭东，廖正宇，李静，李文，邰丽媛，吴邵岭，潘塘，毕枫林等实验室的小伙伴们。正因为你们的陪伴，研究生的生活完整而美好。与你们一起学术，一起求职，一起娱乐的日子让我受益良多，你们的优秀是我的榜样，是我不断前进的勇气和动力。

感谢我的父母，感谢你们二十五年来的悉心教诲和一路上的支持，养育之恩，没齿难忘。我只有努力学习，努力工作，才能回报万一。

感谢在研究生阶段所有有缘遇到的老师们，同学们，正因为你们的存在，我的研究生生涯变得多姿多彩，充满意义。

2020年3月

**参考文献**

[1] D. Lang, “Using SEC,” USENIX; login: Magazine, vol. 38, no. 6, pp. 38–43, 2013.

[2] Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios,“A Lightweight Algorithm for Message Type Extraction in System Application Logs” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 11, NOVEMBER 2012.

[3] Q. Fu, J. Lou, Y. Wang, and J. Li, “Execution anomaly detection in distributed systems through unstructured log analysis,” in ICDM’09: Proc. of International Conference on Data Mining, 2009.

[4] Chinghway Lim, Navjot Singh, Shalini Yajnik, A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems, International Conference on Dependable Systems &Networks,2008.

[5] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, Xuewei Chen, Log Clustering based Problem Identification for Online Service Systems, 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion.

[6] Min Du, Feifei Li, Spell: Online Streaming Parsing of Large Unstructured System Logs, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 31, NO. 11, NOVEMBER 2019.

[7] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, Hua Cai, Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 24, NO. 6, JUNE 2013.

[8] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar, Nipun Arora, Cristian Lumezanu, Jianwu Xu, Bo Zong, Hui Zhang, Guofei Jiang, Latifur Khan, LogLens: A Real-time Log Analysis System, 2018 IEEE 38th International Conference on Distributed Computing Systems.

[9] Jian-Guang LOU, Qiang FU, Shengqi YANG, Ye XU, Jiang LI, Mining Invariants from Console Logs for System Problem Detection.

[10] Ding Yuan, Soyeon Park, Yuanyuan Zhou, Characterizing Logging Practices in Open-Source Software, ICSE, 2012.

[11] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu, Experience Report: System Log Analysis for Anomaly Detection, 2016 IEEE 27th International Symposium on Software Reliability Engineering.

[12] De-Qing Zou, Hao Qin, Hai Jin, UiLog: Improving Log-Based Fault Diagnosis by Log Analysis, JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(5): 1038–1052 Sept. 2016.

[13] Jon Stearley, Towards Informatic Analysis of Syslogs, CLUSTER 2004.

[14] Adam Oliner, Jon Stearley, What Supercomputers Say: A Study of Five System Logs.

[15] Lo¨ ıc Bontemps, Van Loi Cao, James McDermott, and Nhien-An Le-Khac, Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network.

[16] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I. Jordan, Detecting Large-Scale System Problems by Mining Console Logs.

[17] Victoria J. Hodge, Jim Austin, A Survey of Outlier Detection Methodologies, Artiﬁcial Intelligence Review 22: 85–126, 2004.

[18] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fern´andez-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle, The Dataﬂow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing.

[19] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu, Drain: An Online Log Parsing Approach with Fixed Depth Tree, 2017 IEEE 24th International Conference on Web Services.

[20] John P. Rouillard, Real-time log ﬁle analysis using the Simple Event Correlator (SEC), LISA 2004 Conference.

[21] Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu, An Evaluation Study on Log Parsing and Its Use in Log Mining, 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.

[22] Vinayakumar R, Soman KP, Prabaharan Poornachandran, Long Short-Term Memory based Operation Log Anomaly Detection, International Conference on Advances in Computing, Communications and Informatics (ICACCI-2017).

[23] Kenji Yamanishi, Yuko Maruyama, Dynamic Syslog Mining for Network Failure Monitoring, the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,2005.

[24] Risto Vaarandi, SEC–a Lightweight Event Correlation Tool, IEEE Workshop on IP Operations and Management,2002.

[25] Risto Vaarandi, A Data Clustering Algorithm for Mining Patterns From Event Logs, IP Operations and Management, 2003.

[26] Federico Michele Facca, Pier Luca Lanzi, Mining interesting knowledge from weblogs: a survey, Data & Knowledge Engineering 53 (2005) 225-241.

[27] Adam Oliner, Jon Stearley, What Supercomputers Say: A Study of Five System Logs, The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007.

[28] O. Vallis, J. Hochenbaum, A. Kejariwal, A Novel Technique for Long-Term Anomaly Detection in the Cloud, HotCloud, 2014.

[29] Yahoo, S5 - A Labeled Anomaly Detection Dataset, version 1.0, http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70, 2015.

[30] N. Laptev, S. Amizadeh, I. Flint, Generic and Scalable Framework for Automated Time-series Anomaly Detection, KDD, pp. 1939-1947, 2015.

[31] Chengqiang Huang, Geyong Min, Yulei Wu, Yiming Ying, Ke Pei, Zuochang Xiang , Time Series Anomaly Detection for Trustworthy Services in Cloud Computing Systems.

[32] Artit Sagoolmuang, Krung Sinapiromsaran, Median-Difference Window Subseries Score for Contextual Anomaly on Time Series, IC-ICTE.

[33] Markus Thill, Wolfgang Konen, Thomas Ba ̈ck, Online Anomaly Detection on the Webscope S5 Dataset: A Comparative Study, EAIS, 2017.

[34] Yanping Chen, Bing Hu, Eamonn Keogh , Gustavo Enrique Batista, DTW-D: time series semi-supervised learning from a single example.

[35] Suwon Suh, Daniel H. Chae, Hyon-Goo Kang, Seungjin Choi, Echo-State Conditional Variational Autoencoder for Anomaly Detection, 2016 InternationalJoint Conference on Neural Networks (IJCNN).

[36] Qin Yu, Lyu Jibin, Lirui Jiang, An Improved ARIMA-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks, International Journal of Distributed Sensor Networks.

[37] Rob J Hyndman, Earo Wang, Nikolay Laptev, Large-Scale Unusual Time Series Detection.

[38] Li Wei, Nitin Kumar, Venkata Lolla, Eamonn Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, Assumption-Free Anomaly Detection in Time Series, 17th International Conference on Scientific and Statistical.

[39] Donald J.Berndt, James Clifford, Using Dynamic Time Warping to Find Patterns in Time Series, AAAI Technical Report WS-94-03.

[40] Chandola, V., Cheboli, D., Kumar, V., 2009. Detecting anomalies in a time series database. … Department, University of Minnesota, Tech. Rep 12.

[41] B. P. Welford, “Note on a method for calculating corrected sums of squares and products,” Technometrics, vol. 4, no. 3, pp. 419–420, 1962.

[42] Watson, S.M., Tight, M., Clark, S., Redfern, E., Detection of Outliers in Time Series., Institute of Transport Studies, University of Leeds.

[43] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, Helga Van Herle, Finding the most unusual time series subsequence: algorithms and applications, Knowledge and Information Systems, January 2007, Volume 11, Issue 1, pp 1–27.

[44] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, Christine Largouët, Anomaly Detection in Streams with Extreme Value Theory, the 23rd ACM SIGKDD International Conference.

[45] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003) A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.

[46] D. Yuan, S. Park, and Y. Zhou, “Characterizing logging practices in open-source software,” in ICSE’12: Proc. of the 34th International Conference on Software Engineering, 2012, pp. 102–112.

[47] A. Makanju, A. Zincir-Heywood, and E. Milios, “Clustering event logs lusing iterative partitioning,” in KDD’09: Proc. of International Conference on Knowledge Discovery and Data Mining, 2009.

[48] L. Tang, T. Li, and C. Perng, “LogSig: generating system events from raw textual logs,” in CIKM’11: Proc. of ACM International Conference on Information and Knowledge Management, 2011, pp. 785–794.

[49] Yingying Chen, Ratul Mahajan, Baskar Sridharan, Zhi-Li Zhang, A Provider-side View of Web Search Response Time, Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, Volume: 43

[50] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Dan Massey, Christos Papadopoulos, Hiren Shah, Jennifer Yates, Argus: End-to-End Service Anomaly Detection and Localization From an ISP’s Point of View, IEEE INFOCOM

[51] Netflix, Surus, Available: <https://github.com/Netflix/Surus>

[52] Linkedin, luminol, Available: <https://github.com/linkedin/luminol>

[53] Emmanuel J. Cand`es1,2 , Xiaodong Li2 , Yi Ma3,4 , and John Wright4, Robust Principal Component Analysis

[54] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu Shimin Tao,, Zhi Zang, Rapid and Robust Impact Assessment ofSoftwareChanges in Large Internet-based Services, CoNEXT ’15, December 01-04, 2015, Heidelberg, Germany.

[55] W. Dickinson, D. Leon, and A. Podgurski, “Finding Failures by Cluster Analysis of Execution Profiles. In the proceeding of the 23rd International Conference on Software Engineering, May, 2001.

[56] A.V. Mirgorodskiy, N. Maruyama, and B.P. Miller, “Problem Diagnosis in Large-Scale Computing Environments”, In the Proceedings of the ACM/IEEE SC 2006 Conference, Nov. 2006.

[57] C. Yuan, N. Lao, J.R. Wen, J. Li, Z. Zhang, Y.M. Wang, and W. Y. Ma, “Automated Known Problem Diagnosis with Event Traces”, In the proceeding of EuroSys 2006, Apr. 2006.

[58] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, “Mining Console Logs for Large-Scale System Problem Detection”, In Workshop on Tackling Computer Problems with Machine Learning Techniques, Dec. 2008.

[59] D. Cotroneo, R. Pietrantuono, L. Mariani, and F. Pastore, “Investigation of Failure causes in work-load driven reliability testing”, In the proceeding of the 4th International Workshop on Software Quality Assurance, Sep. 2007.

[60] S. Orlando and S. Russo, “Java Virtual Machine Monitoring for Dependability Benchmarking”, In proceedings of the 9th IEEE International Symposium on Object and Component-oriented Real –time Distributed Computing, Apr. 2006.

[61] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, “SALSA: Analyzing Logs as State Machines”, In the proceeding of 1st USENIX Workshop on the Analysis of System Logs, Dec. 2008.

[62] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In ICAC’04: Proc. of the 1st International Conference on Autonomic Computing, pages 36–43. IEEE, 2004.

[63] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In ICDM’07: Proc. of the 7th International Conference on Data Mining, 2007.