申请上海交通大学硕士学位论文

基于日志监控的异常检测技术与异常监控系统

学校： 上海交通大学

院系： 电子信息与电气工程学院

班级： B1303391

学号： 1130339010

硕士生： 杜思忠

专业： 计算机科学与技术

导师： 曹健

上海交通大学电子信息与电气工程学院

2016 年 1 月

A Dissertation Submitted to Shanghai Jiao Tong University for the Degree of Master

Anomaly Detection Technique and Monitoring System Based on Log Monitoring

Author: Du Sizhong

Specialty: Computer Science

Advisor: Cao Jian

School of Electronic Information and Electrical Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

January, 2016

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在年解密后适用本授权书。

本学位论文属于

不保密□√。

（请在以上方框内打"√"）

学位论文作者签名：杜思忠　　指导教师签名：

日期：2016年 1月11日　　日期：　年　月　日

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文《基于磁盘失效预测的云数据恢复调度研究》，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：杜思宏

日期：2016年 1月 11日

基于日志监控的异常检测技术与异常监控系统

摘要

随着计算机科学和互联网技术的不断进步，信息技术的发展迈入了一个新的高度。支撑社会运转的软、硬件系统，不管从规模大小或是复杂程度，都到达了前所未有的水平。如今政府大力推进“互联网+”计划，互联网与传统行业结合得更加紧密，相互促进彼此的发展。在这个云计算和大数据大行其道的年代，维护互联网及其软硬件系统的稳定运行、检测和修复潜在的异常就变得更加重要。

日志监控是软件监控系统中的重要组成部分。通过对日志内容和日志输出特征的分析，可以判断当前系统的运行状态，预测可能发生的异常。目前已经有很多关于日志分析的研究，已有的监控系统也取得了很好的效果。但当前日志分析技术仍然面对着一些问题，主要表现在如下方面：1）日志数量巨大。大型软件系统每天可以产生 TB 数量级大小的日志，逐条分析日志内容会对日志分析系统造成巨大压力。2）日志结构不固定，并且可能不完整。日志的有效内容（payload）并不遵循特定的格式，在系统高负载的情况下日志内容可能出现丢失，给日志的自动化处理带来困难。3）不同系统产生的日志内容差异巨大，通用的日志监控系统存在检测准确率不高的问题。

我们针对以上这些问题进行了研究，经过实验，提出了基于日志监控的行为异常检测技术，并且实现了相应的日志监控系统。本文的研究工作主要体现在以下几个方面：

1. 提出了基于日志标准化和层次聚类的日志预处理方法。首先对日志信息进行正规化处理，然后采用自底向上层次聚类算法，基于日志有效内容的相似度对日志进行聚类。

2. 提出了基于行为异常的通用日志异常检测算法。从聚类后的日志中提取出特有行为模式，基于行为模式的异常指数和相似度等特征对日志序列进行异常检测，预测系统状态。

3. 设计实现了基于行为异常检测的日志监控系统。通过训练得到的行为模型，对日志流进行实时高效地分析，帮助管理人员有效监控和维护生产环境。

关键词：日志监控、异常检测、层次聚类、异常连续子序列

Anomaly Detection Technique and Monitoring System Based on Log Monitoring

ABSTRACT

With the continuous advance of computer science and internet technology, the information industry has evolved into a new stage. Both the scale and complexity of software and hardware that support the operation of our society, have reached a level never seen before. Governments are now carrying forward the plan of "Internet+", which consolidates the link between the Internet and traditional industry and accelerates the evolution of each other. In the era of cloud computing and big data, it becomes more and more important to maintain the runtime stability and detect potential anomalies in the large-scale software & hardware systems.

Log monitoring is one of the most important part in software monitoring systems. People could judge the runtime status of the system and predict possible anomalies by the analysis of log contents and the feature of log outputs. There have been lots of researches on log analysis, existing log monitoring systems have also obtained good results. But current monitoring techniques are still facing some problems. 1) The volume of data generated from log files can be really large. A large-scale software system could generate terabytes of logs per day. To analyze the log contents line by line would put big pressure on the system. 2) Log messages are unstructured and may be incomplete. The payload of log data does not follow a specific format, and logs may get lost when the system is under a heavy load or error, which makes it inconvenient for machine process. 3) Logs generated from different systems differ a lot in the format and payload, a general monitoring solution may not achieve a high accuracy in detecting anomalies.

In order to solve these problems, we did lots of researches and experiments, then put forward a behavioral anomaly detection approach based on log monitoring, and further implemented a general log monitoring system. Our work in this paper involved three contributions:

1. A log preprocessing method based on log normalization and message clustering. We first normalize the log data, then categorize logs into clusters by the similarity of the payload, with an effective hierarchical clustering algorithm.

2. A general log anomaly detection method based on behavioral anomalies. We extract specific behavioral patterns from clustered logs, detect anomalies in log sequences based on the anomaly score and similarity of those extracted patterns, so as to predict the

system status.

3. A log monitoring system based on the technique of behavioral anomaly detection. The behavior model via training analyzes log streams in an instant and efficient way, which helps the administrator monitor and maintain the production environment.

Keywords: Log monitoring, Anomaly detection, Hierarchical clustering, Anomalous contiguous subsequence

目录

第一章 绪论

1.1 研究背景及意义

信息技术自其诞生到现在的几十年间，其发展速度和重要性都是前人未曾想到的。随着计算机科学和互联网技术的不断进步，信息产业的发展也迈入了一个新的阶段，而支撑信息产业的互联网产品及其软硬件系统，无论是规模大小或者复杂程度，都达到了前所未有的水平。快报数据显示在 2014 年，我国的软件和信息技术服务业就实现了 3.7 万亿元的软件业务收入，同比增长 20.2%[1]，软件行业在国民经济增长中的重要性可见一斑。如今正是云计算和大数据大行其道的年代，大型分布式系统以云计算平台为依托，处理着海量的、来自各行各业的数据信息。我国政府大力推行的“互联网+”计划，使互联网与传统行业结合得更加紧密，相互促进彼此的发展。围绕着互联网的信息产业越来越成为支撑社会运转和前进的支柱动力，其重要性毋庸置疑。

虽然各行各业的软件系统肩负着重要的使命，互联网和软件系统本身却是脆弱的。如今的软件系统规模越来越庞大，结构越来越复杂，异常和错误的发生变得难以避免。软件异常是软件产品在使用过程中发生的实际运行结果和期望得到的结果不一致的问题。软件异常包括故障、失效和缺陷等等。可以说软件异常充斥在软件开发的每个阶段，并且包含在最终交付的软件产品中。调查表明生产环境的大型软件系统中，发生各类异常的频率超过每年 1000 次，平均维修时间从几小时到接近 100 小时不等[2]，不仅降低生产效率，造成经济财产损失，某些情况下还有可能危及生命安全。因此，尽量避免异常的发生是必须面对和重视的问题。目前一般有两种办法避免软件异常：一是在开发过程中做好软件异常的预防，包括良好的编程习惯，结构化、科学的软件开发流程，以及大量必不可少的单元测试、端到端测试等等；二是在系统运行过程中使用监控系统对软件状态进行实时监控，及时检测到已经发生或即将发生的异常行为，为采取对应措施留出宝贵时间，这就是我们所研究的监控系统中的异常检测。

异常检测（anomaly detection, outlier detection）一直是软件可靠性和数据挖掘中的热点问题，它是指在数据及行为中发现有别于期望行为的数据模式，这些异常模式包括点异常（point anomaly）、上下文异常（contextual anomaly）、集合异常（collective

anomaly）等。关于异常检测的研究已经进行了很多年，不同的异常检测技术也应用到了社会各个领域，例如网络入侵检测、欺诈检测、医疗和公共卫生异常检测、工业损伤检测、图像处理、文本异常检测、传感器网络等等[3]。异常检测最常见的用例之一就是监控计算机系统的运行状态，包括系统运行时消耗的硬件和网络资源及运行过程中发生的事件，并在导致系统崩溃的错误真正发生之前及时捕捉异常。日志监控在其中扮演了重要的角色。

日志监控是对软件系统运行中产生的各类日志进行采集、过滤、存储、分析、检测异常的一系列过程，是软件监控系统中的重要组成部分。日志广泛存在于各类系统中，记录系统运行时发生的事件，产生的数据，程序运行时间戳和上下文等等。日志记录较为详细地反映了当前系统状态和系统开发者想要表达的有意义的或异常的事件，为系统管理员提供了管理系统的重要参考。通过对日志内容和日志输出特征的分析，可以判断当前系统的运行状态，预测可能发生的异常。关于日志分析的研究已经持续了很长的一段时间，中间产生了不少研究成果，根据这些研究成果开发的产品也取得了不错的效果。但当前日志分析技术仍然面对着一些问题，主要表现在：

1. 日志数量巨大

如今的大型网站和软件系统，其规模和复杂度都达到了前所未有的程度。这些系统每天可以产生 TB 数量级大小的日志。如果使用传统的基于语义的日志分析系统，逐条对日志内容进行分析判断，势必对分析系统造成巨大压力。考虑到日志产生的速率、处理每条日志所需要的时间以及硬件开销，现有分析技术很难实现对日志内容的实时有效分析。

2. 结构不固定

虽然现在各个平台都有了格式化生成日志记录的库或插件，但日志内容中最重要的有效负载（payload）部分仍然是由程序员手动编写的，并不遵循特定的格式，其格式和内容由程序员自己决定。而且当所监控的客户系统处于高负载或发生错误的情况时，日志信息可能出现丢失或者不完整的情况。这些都给日志的自动化处理带来了不便。

3. 日志系统间的差异

由于不同行业、不同领域使用的软件系统千差万别，产生的日志记录不管是从格式或者内容上都有巨大的差异。很多日志分析技术如传统的基于语义的分析系统，只能针对特定软件产生的日志进行分析，而通用的日志监控系统存在异常检测精确度不高的问题。

根据对以上问题的分析，我们所需要的是一个实时高效的、能处理不同日志结构的通用日志监控系统。我们针对这些问题进行研究，经过实验，提出了基于日志监控的行为异常检测技术。检测方法分为两步：日志预处理和日志异常检测。本文的意义表现在：

1. 基于日志正规化和层次聚类的日志预处理方法。首先通过一系列正规化方法对日志数据进行正规化处理；然后基于日志有效内容的相似度对日志进行层次聚类。文中提出的聚类相似度定义和剪枝策略提高了聚类准确率，优化了聚类粒度。通过预处理后的日志信息具有良好结构，方便后续行为模式的提取和异常行为的判断，也为日志预处理技术进一步研究提供了参考。

2. 基于行为异常的通用日志异常检测方法。把聚类后的日志转化为频率序列，并从中提取出特有行为模式。基于行为模式的异常指数和相似度等特征对日志序列进行异常行为检测，预测系统状态。有别于传统的点异常检测，行为异常检测更好的利用了日志流的行为特征，为日志监控的研究探索了新的方向。

3. 基于行为异常检测的通用日志监控系统。我们设计并实现了通用的日志监控系统，验证了算法的有效性。通过训练得到的行为模型，可以对日志流进行实时高效地分析，帮助管理人员有效监控和维护生产环境。

1.2 国内外研究现状

1.2.1 关于日志预分析处理的研究

关于日志预分析处理的研究已经进行了很多年。早在上世纪 80 年代，Anderson 等就提出了基于日志分析的安全审计思想，使用数据挖掘技术和机器学习工具进行日志分析挖掘。90 年代我国自然科学基金会也开始发起对日志安全审计研究的支持，众多大学学者对日志的采集、存储和分析进行研究，取得了很多成果。Adam 和 Jon[4] 通过对五台巨型计算机系统日志的研究，对日志的一般结构和特征作了详细的描述，并给出了过滤有效数据的基本方法；V.Chandola 等人[3, 5]对异常检测这一学科内容作了结构化的总结，把异常大体分为三类：点异常、上下文异常和集合异常；Felix 和 Steffen[6]基于日志内容的 Levenshtein 对日志进行归类，并通过隐马尔科夫模型和数学统计算法对日志进行聚类和过滤；Zheng 等人[2]提出了日志预处理的三步算法，通过 Apriori 关联规则挖掘进行事件归类、事件过滤和因果关系过滤。Risto[8]通过频繁项集挖掘对日志进行聚类，以便检测频繁模式和异常日志记录。

1.2.2 关于日志异常检测的研究

已有的日志异常检测方法通常根据日志数据本身的特征或生成日志的客户系统来进行归类。当日志训练数据是已分类的，即单条日志是被标记为正常日志或错误日志时，一般使用监督学习或者半监督学习的机器学习方法对数据进行学习，得出分类模型。关于这方面的研究有很多，例如 Kenji 和 Yuko[9]通过隐马尔科夫模型和 online discounting 算法分析网络日志，检测网络错误；R.K.Sanhoo 等人[10]通过时间序列算法、基于规则的分类算法和贝叶斯网络模型，构建了一个主动预测和控制系统，预测将要发生的日志事件；Fronza 等人[11]使用随机映射和 SVM（支持向量机）对日志数据进行分类；Antti 和 Timo[12]采用随机映射和维数约减等方法，提出了自己的异常检测框架，改善了系统状态预测算法的时间复杂度。然而事实上，大多数的日志数据由于数量巨大，人工地逐条给这些日志进行标记极其消耗时间甚至是不可能完成的。因此实际应用更多的异常检测技术是基于非监督学习或其他统计学习方法。例如在[13, 14]中，Lim 和 Chen 通过日志有效载荷的相似度对日志进行聚类，然后基于对固定时间窗口中不同日志类产生的频率进行统计，通过频率特征对系统状态进行预测；Yen 等人[15]对网络日志数据中的一系列特征进行提取，如基于目标位置的特征，基于主机信息的特征等，通过这些特征检测可疑的用户行为；Xu 等人[16]则是通过源代码分析获取日志模式，然后使用基于主成分分析的算法把离群点标记为系统异常，获得了很好的效果，但缺陷是必须获取客户程序源码。

已有的研究更多的是把注意力放在点异常（point anomaly）上，对日志分析来说即是指单独的一条或一段日志记录出现异常。为了进一步提高预测的准确度，在这篇文章中，我们引入了行为异常（behavior anomaly）这个概念。行为异常是系统发生错误或工作负载发生变化时表现出的重要特征，可以通过检测日志流中发生的行为异常来判断系统状态。日志记录中的行为异常指的是由连续的日志记录组成的日志子序列在生成过程中表现出的异常行为。本文阐述的两步检测方法首先对日志进行预处理，以合适的粒度进行聚类，然后分析提取每一类日志的行为模式，在监控日志流时，计算新的日志行为模式与正常行为模式的匹配度，以此判断系统是否出现异常。

1.2.3 现有的日志监控解决方案

如今业界涌现出不少日志监控解决方案，不少已经发展得相当成熟。这些日志分析和存储方案涵盖了商用或开源的本地解决方案，以及允许将生产环境的日志外包给第三方进行托管和分析的服务。

1. Splunk①

提到日志管理软件就不得不提 Splunk。Splunk 是业界顶级的日志管理与分析软件，与传统的文本分析工具如 grep、awk、sed 等工具相比，Splunk 是质的飞跃，可以大大提高工作效率。使用 Splunk 只需要对应用服务器进行简单配置，通过 rsyslog、SFTP 或 NFS 等方式向 Splunk 推送提交日志数据。Splunk 允许集中存储日志，进行取证、审计和关联，也能根据需要和配置信息对日志进行过滤操作，判断异常并生成实时警报。

Splunk 的主要功能包括：1) 日志聚合。Splunk 可以把分散在主机和随机的源中的数据进行聚合和存储，方便进行搜索和其他分析操作；2) Splunk 有极为强大的搜索功能，通过字符串或正则表达式就能精确检索到结果；3) 抽取数据，对数据进行过滤、分组、联合、拆分和格式化；4) 强大的可视化功能，通过图、表展现结果等等。

Splunk 包含众多特性，支持多个平台和超大规模日志处理。但作为一个商业软件，Splunk 并不开源，研究者无法学习其功能背后使用的技术。虽然有限制功能的免费版本，但若要使用其完整功能并应用到大型系统中的话，还是要支付不菲的费用，增加了企业负担。

2. Logstash②

与 Splunk 不同，Logstash 是一款开源日志监控工具。Logstash 是 Elastic 解决方案的一部分，通常与该公司另外两个数据处理系统联合使用：ElasticSearch 和 Kibana。其中 ElasticSearch 是基于 Lucene 的搜索服务器，Kibana 则为 Logstash 和 ElasticSearch 提供日志分析和数据展示的 web 接口。这三个产品组成了完整的日志管理方案。

Logstash 的主要功能包括：1) 汇集不同类型的日志信息。作为数据输送管道，Logstash 可以帮助开发者处理来自不同系统的日志或其他事件数据。它拥有的众多插件覆盖了大多数日志类型，通过这些插件，Logstash 可以连接各种数据源和流数据，并把这些数据集中存储到中央数据分析系统；2) 日志信息标准化。Logstash 可以收集不同系统中不同类型的日志，通过解析日志内容，把其中一些公共信息如时间戳等进行标准化处理，统一格式以便后续分析；3) 插件的个性化定制。Logstash 具有良好的可扩展性，使用 Logstash 提供的 API，开发者可以开发适合自己系统日志的日志收集插件，还可以发布自己编写的插件供其他开发者使用。

Logstash 起初的设计目的只是为了进行日志收集、把分布在各处的日志汇集到一

① Splunk, http://www.splunk.com/

② Logstash, https://www.elastic.co/products/logstash

处以供后续处理。但后来其功能慢慢丰富，添加了数据的过滤和正则化功能，并且能与 ElasticSearch 及 Kibana 很好的集成在一起使用，足以替代商业软件。其开源的特性也吸引很多人参与到其中进行学习和开发，为日志分析这项研究提供了更多动力。

1.3 本文研究内容和结构安排

1.3.1 研究内容

本文主要研究的内容有：

1. 基于日志标准化和层次聚类的日志预处理方法。

首先进行标准化处理，对日志格式进行正规化、去参数化等操作，清洗数据以便于后续的日志分析；然后采用层次聚类算法，基于日志有效内容的相似度对日志进行聚类。把相同类型或相似的日志进行聚类的目的是为了对同一类日志的生成速率进行统计，基于不同日志的生成特征对系统状态进行判断。

2. 基于日志行为模式的通用异常检测算法。

算法首先把聚类后的日志记录以固定的时间间隔转化为频率序列；然后基于频率序列提取出特有行为模式，其中每一个类日志对应一个行为模式集；最后基于行为模式的异常指数和相似度等特征对日志流进行异常检测，预测系统状态。算法引入了时间序列不和（time series discord）理论，通过不同数据集的仿真实验对比证明其良好的预测准确率。

3. 基于行为异常检测的日志监控系统。

根据日志数据的异常检测算法框架设计并实现基于行为异常检测的日志监控系统。系统通过对已有日志数据训练得到行为模型，然后根据得到的模型对日志流进行实时分析。通过运行真实日志数据，验证了系统的有效性和预测准确率，证明系统可以在真实环境下帮助管理人员有效监控和维护生产系统。

1.3.2 文章结构安排

本文共有六个章节，各个章节的组织结构如下：

第一章：绪论。介绍了本文的研究背景和意义、国内外研究现状、本文研究内容和论文结构安排。在研究背景和意义中我们阐述了在信息技术高速发展的背景下互联网及其软硬件系统的脆弱性及进行异常检测的必要性，说明了当前日志监控研究面临的问题和针对这些问题进行研究的意义；在国内外研究现状中我们介绍了当

前日志预分析处理和日志异常检测的研究现状，以及现有的日志监控解决方案；最后介绍了本文主要研究内容和文章结构安排。

第二章：相关技术概念。介绍了日志基本概念和日志监控的相关技术基础。在日志概述中，介绍了日志的定义、功能和特点。在日志监控相关技术的介绍中，首先定义了日志监控的概念。随后介绍了日志的收集方式、存储格式和存储位置。接着列举了三种开源的日志收集、存储系统。最后介绍了日志的人工分析方法自动化分析技术，列举了四种常用的基于机器学习和数据挖掘的日志分析方法。

第三章：日志数据预处理。阐述了两步日志异常检测方法的第一步：日志数据预处理。首先介绍了日志处理的必要性，日志由于数据缺失、结构不固定、数量巨大等问题需要进行预处理操作。接着介绍了本文主要采用的两个数据集：Hadoop 事件记录日志和 LANL 数据集。然后分别对预处理中的日志正规化方法和日志信息聚类进行了比较详细的介绍，主要介绍了基于 complete linkage 的、以编辑距离为日志数据相似度的凝聚聚类方法。最后将预处理方法进行了实验验证，证明预处理方法的有效性。

第四章：基于行为模式的日志异常检测。介绍了两步日志异常检测方法的第二步：日志流异常检测。首先描述了异常检测算法中引入的连续子序列理论，讨论了异常子序列问题和滑动窗口技术，简要介绍了常用的几种异常序列检测算法，这些算法是日志异常检测技术的理论基础。接着详细描述了基于行为模式的异常检测算法。算法包括三个步骤：把日志流转换为行为序列、在行为序列集中训练行为模式、通过行为模式计算实时日志流的异常指数。最后对实际数据集进行了仿真实验，并对算法检测效果做了对比和分析。通过与传统频率算法在两个实验数据集上的测试结果对比，证明了本文提出的日志检测方法明显提高了异常检测准确率，具有很好的效果。

第五章，基于行为异常检测的日志监控系统。介绍了日志监控系统的原型实现。首先对系统需求进行了分析；接着对系统的整体框架、框架中各个模块以及模块之间的结构关系、数据流动关系做了描述；最后依次介绍了监控系统的日志采集模块、存储分析模块和前端展示模块，并用功能框架图、流程图、类图和功能列表进行辅助说明。

第六章，总结与展望。对本文提出的理论和完成的工作进行总结，并对未来的研究工作进行展望和规划。

第二章 相关技术概念

2.1 日志概述

日志存在于软件系统的各个角落，详细描述了当前系统状态，清楚地反映了系统开发者对值得注意的或不寻常事件的记录。已有的研究表明，系统在真正失效乃至崩溃之前，会经历一系列非致命的异常和错误，而这些异常和错误可能作为发生的事件实时记录在日志流中。因此对日志流进行监控和分析是检测系统异常行为的有效手段，通过异常检测可以对系统失效做出提前预警，为系统管理人员及时采取对应措施节省出宝贵时间。

2.1.1 日志定义

我们所研究的日志（Log）是对系统行为进行描述的记录，通常是指用来描述应用软件、操作系统、服务器设备、网络设备以及安全设备等特定对象中产生的某些操作和操作的结果按时间有序排列的集合[17]。日志一般以日志文件的方式保存在存储设备中，日志文件可以是可直接阅读的文本文件或是机器可读的二进制文件。每个日志文件由一行一行的日志记录组成，一条或连续的几条记录描述一次独立的事件。事件是在某个环境下发生的事情，通常涉及改变状态的尝试。事件通常包括发生时间、事件内容、以及与时间或者环境相关的任何可能有助于解释事件原因及效果的细节。日志即是事件记录的集合。虽然日志记录有各种各样的格式，但一条日志记录通常包含一个事件发生的时间戳、日志的具体内容、日志类型、日志级别，有的日志还包括节点 ID、源地址、目标地址、进程 ID 等等。这些内容大体可以划分为三类组成部分：主体（Subject）、客体（Object）和行为（Action）[19]，一条日志表现为主体对客体进行的某种行为及其产生的结果。在一些入侵检测模型中，主体代表用户或者由用户发起的某种操作，客体代表各类资源如软硬件设施或网络环境等，行为则代表系统服务或者客户软件系统的某种应用行为。

2.1.2 日志的功能

日志由开发人员编写，存在于程序的各个角落。日志什么时候、从什么地方、通过什么条件以什么样的方式产生，都是由开发人员决定的，其目的除了对程序逻辑进行调试，更多的是在生产环境中对程序运行状态和用户行为进行记录和监控，以便对系统进行维护、对用户进行审计等等。具体来说，日志在辅助各类软件系统

的管理中有如下作用：

1. 资源管理

软件系统的平稳运行离不开各种系统资源的支持。为了充分并且高效的利用系统资源，可以通过系统日志对进程、线程、CPU、内存、外部存储以及网络资源的使用情况和实时状态进行记录，根据记录对资源的硬件错误进行检测、对负载均衡进行调优等。

2. 入侵检测

用户行为包含在系统日志中，当一个用户登录、注销或者从某处进入等情况发生时，某些日志（如进程账户日志）会告诉你一些关于用户行为的信息。作为网络安全屏障的防火墙，其日志文件中就记录了大量的有用信息[17]，通过对日志记录的分析可以检测出隐藏在其中的入侵行为、重现入侵过程、修复入侵漏洞，甚至能够追踪犯罪入侵者。

3. 异常检测

异常检测指在围绕着数据的行为中发现有别于期望结果的模式。开发者在编写程序时，把程序逻辑和可能发生的错误通过日志反映出来，包括 syslog 在内的多种日志就是为了这个目的而设计的。通过对日志内容和日志输出特征的分析，可以判断当前系统的运行状态，预测可能发生的异常。

4. 事务回滚和重建

很多时候我们运行的一个业务流程包含一系列对数据库操作的事务，当其中一个事务发生错误时，需要对已经完成的事务进行回滚。数据库日志记录了用户对数据的每一步操作，这些日志可以用来使数据回滚到操作之前的状态，或者对保存的数据执行日志中的操作，使其状态更新到最新。

5. 取证

取证是指事件发生后重建情景的过程。日志一经记录就不会由于系统的正常使用而被修改，它们可以为系统中其他容易被更改或破坏的数据提供准确的补充。由于每条日志通常都包含时间戳，不仅可以告诉我们发生了什么事件，还告诉我们这些事件发生的时间和顺序。日志记录有助于揭示到底发生了什么并指出正确的方向。

6. 审计

审计是验证系统是否如预期运行的过程。日志是审计过程和形成审计跟踪的一部分。例如通过查看用户信用卡交易日志、持卡人访问日志等用户账户使用行为，可以实现对用户操作的有效监控、进行权限管理、防止行为滥用、进行欺诈检测等。

2.1.3 日志的特点

计算机的日志系统发展至今，不仅类型繁多、格式多样、数量巨大，并且还产生了众多成熟的、可配置、或商用或开源的日志组件，如 Log4j、Sl4j、Apache Commons Logging、Logback 等等。日志系统的共同特点总结如下：

1. 多样性

日志的多样性体现在日志的传输方式、日志的语法与格式、日志的存储方式等。首先日志有许多种传输协议[20]，例如 syslog、SOAP over HTTP、SNMP、WS-Management 和许多专有的产品特定的日志传输协议，以及本地存储方式。其次日志有不同的语法和格式[20]，知名的日志格式就有 W3C 扩展日志文件格式、Apache 访问日志、Cisco SDEE/CIDEE、ArcSight 公共事件格式、syslog、IDMEF 等，但很多日志仍然没有遵循任何通用格式，以自由格式的文本存在，这也给日志分析工作带来了困难。最后，日志的存储方式包括人类可读的文本文件或机器可读的二进制格式。

2. 日志数量巨大

以前日志的分析工作一般是由开发管理人员借助简单的文本处理工具进行手工分析，例如查找、过滤等等，这主要是因为当时软件系统规模还不是很大，日志规模还在人工可控的范围之内。如今大型网站及背后的业务系统，其规模和复杂度都是空前的，每天产生的日志大小能达到 TB 数量级，给存储和分析带来很大的困难，通过人工逐条分析日志内容以检测系统问题更是不可能完成的任务，也因此才需要日志监控系统对海量的日志数据自动进行快速分析和检测。

3. 可读性较差

由前述可知日志记录没有固定的格式结构，并且有时不一致、不完整或充满误导。很多日志文件是以二进制形式存储，只能通过专用软件打开，这些都给日志阅读带来困难。很多日志内容只记录一些关键数据，要读懂这些数据需要借助相应的帮助文档才能明白。良好的日志记录至少应该包含发生了什么事件、事件发生的时间、位置、源头和目标等。

4. 脆弱性

虽然日志作为软件系统的一部分，为软件系统的运行状态做了重要的记录，但日志系统本身的优先级却是很低的，当系统遇到高负载或发生异常时，日志可能出现不一致、不完整的情况。日志文件一般也没有加密措施或其他防止篡改的保护机制，还可以通过修改配置和杀死进程的方式关闭日志记录。因此说日志是脆弱的、

不可靠的。

2.2 日志监控

日志监控是对软件系统运行过程中产生的各类日志流进行跟踪、汇集、过滤、存储、分析、最后汇报结果的一系列过程，是软件监控系统的重要组成部分。通过对日志进行不同角度的分析，可以达到管理资源、检测入侵行为和软件异常、控制事务状态、进行事件取证和日志审计等目的。本文所涉及的研究是通过对日志内容和日志输出特征的分析，判断当前系统的运行状态和预测可能发生的异常。

2.2.1 日志收集

进行日志采集首先要了解数据的来源。日志数据来源总的分为两类：基于拉取（pull）的方式和基于推送（push）的方式 [20]。

基于 pull 的方式是指应用程序从来源拉取日志信息，使用这种方式通常要建立一个客户/服务器模型。以这种方式运行的系统通常使用自己设计的方式保存日志数据，通过安装在日志来源系统的客户端或代理，经由 TCP socket 或 HTTP 报文等方式把日志汇集到服务器。很多第三方的日志管理系统就通过这种方式对日志进行收集，我们的日志监控系统也是基于 C/S 模型，通过配置文件收集日志数据。

基于 push 的方式是设备或者应用程序主动向本地存储或者网络端口发出消息，如果通过网络则必须配备一个对应的日志收集器来接收消息。通过推送的日志及协议有 syslog、SNMP 和 Windows 事件日志等。

1. syslog

syslog，即系统日志，是在 IP 网络中转发日志信息的标准，syslog 被 Unix 内核、Linux 内核以及许多应用程序用来记录日志信息，是基于 Unix 的系统中进行日志记录最常使用的协议，目前已成为业界工业标准，由 RFC3194 进行规定，而 RFC5424 提出了新的 syslog 协议标准草案，进一步完善 syslog 协议。syslog 包含 syslogd 守护进程，应用程序通过 syslog 程序库调用来和 syslogd 进行通信。syslogd 通过 socket 从内核或应用程序接收日志记录，并且可以通过 UDP 514 端口从远程主机接收日志数据，而后起之秀 rsyslog 和 syslog-ng 还能使用 TCP 协议。syslogd 可以通过配置文件对其行为进行设置，配置文件通常位于/etc/syslog.conf。syslog 可以包含多种消息类型，如来自内核的消息、来自设备和应用程序的消息、身份日志消息等。syslog 输入的消息格式是 ASCII 文本格式，因此查看 syslog 日志文件不需要任何特殊的软件，普通的文本阅读工具就能满足要求。

2. SNMP

SNMP，即简单网络管理协议，是基于 TCP/IP 协议栈的网络管理标准，用于在 IP 层网络中管理网络节点设备（如服务器、路由和交换机）的标准协议，可以对设备进行查询和配置、及时发现和解决网络问题、帮助优化网络结构。SNMP 是 IETF 即互联网工程工作小组定义的 internet 协议族的一部分，从上世纪 90 年代初开始，SNMP 就已经集成到几乎所有网络系统中，包括安全相关的网络设备和系统。SNMP 协议整体来说并不是专用于日志记录，但网络设备在特定事件发生时会生成特殊的 SNMP 消息，称作 SNMP 陷阱和通知，这些消息可以看作日志的消息类型。SNMP 管理的网络由被管理的硬件、SNMP 代理和网络管理系统 NMS 组成，被管理的硬件通过 SNMP 代理把管理数据经由 SNMP 协议发送给 NMS，由 NMS 运行管理程序实现对被管理设备的监控。

3. Windows 事件日志

Windows 操作系统从早期版本开始就有了比较完善的日志生成和收集系统，称为事件日志（Event Log），通过内建的事件查看器就可以进行查看。Windows 事件日志系统主要收集和查看两类日志：Windows 日志和应用程序日志。Windows 日志包括系统日志和安全日志，系统日志记录了系统启动时加载的驱动程序和用户组件信息，以及运行过程中可能发生的故障等，安全日志记录了用户登录、注销、对资源的访问等安全信息；应用程序日志则主要记录应用程序的运行记录、个人设置、程序运行中产生的各种错误等等。

2.2.2 日志存储

日志数据根据不同的类型、规模、格式、来源等，在不同的位置以不同的格式进行存储。存储的格式有文本格式、二进制格式和压缩文件格式，存储的位置有普通文件系统、数据库、Hadoop HDFS 等。

1. 存储格式

基于文本文件的存储格式是目前最常见的格式类型，很多计算机语言都集成了能够轻松生成文本日志的框架。这种格式的主要优点是记录成本较低，以文本格式顺序存储日志数据不需要耗费太多 CPU 或 I/O 资源；另外一个优点是便于阅读，无需专用工具，只需要普通的文本工具就可以进行查阅和处理。缺点则是文件体积过大、查询效率不高等。

二进制日志文件是机器可读的日志文件，有特定的格式，需要借助专用的软件才能进行阅读和处理，典型的二进制日志文件有 Windows 事件日志，只能通过事件

查看器进行查阅。相比文本文件，二进制日志文件较节省空间，存储效率更高，缺点则是不方便阅读。

大部分日志系统会周期性对日志文件进行删除和压缩工作，其中超过保存期限的文件会被删除，非最近生成的日志则会被压缩存储以节省空间。可以通过通用或专用的文件压缩工具对日志进行压缩，但当对文件数据进行查询分析的时候可能会增加一次解压缩的操作。

2. 存储位置

日志数据最常见的存储方式还是直接以文件的方式存储在文件系统中。如果软件系统本身规模不是很大、对日志数据的处理也没什么特殊要求的话，把日志文件简单保存在磁盘是最好的选择。

许多系统将日志数据写入关系数据库中，根据键值关系对日志数据进行搜索和查询。这样做的优点是显而易见的，可以使用 SQL 语句方便快捷地对日志数据进行 CURD 操作，有利于日志审计、取证和一般的检测分析。缺点则是写入数据和维护数据库的开销较大，且不能很好支持对特大规模日志数据的操作。

随着日志规模的不断扩大，使用 Hadoop 存储日志开始成为新的解决方案。Hadoop 专为大数据而生，其中的 Hadoop 分布式文件系统 HDFS 有效解决了海量数据存储的问题。通过把日志数据储存在 HDFS，借助 Hive 和 HBase 等工具可以对日志进行快速高效的检索和分析，同时解决了传统数据库的容量瓶颈，目前包括 Splunk 和 Logstash 在内的很多日志分析都提供了对 Hadoop 的支持。使用 Hadoop 的缺点可能就是对 Hadoop 进行维护的成本和一些多余的硬件开销。

2.2.3 几种开源日志系统

这些日志系统主要用于收集和存储日志，并在业界得到了广泛应用[23]。

1. Scribe①

Scribe 隶属于 Facebook，是由 Facebook 开发的开源日志收集系统，应用于 Facebook 内部，为其每天产生的大量网站日志提供存储支撑。它从各类源头对日志数据进行收集并把数据存储到 NFS 网络文件系统或其他分布式文件系统上面，以便于后续的统一分析处理。Scribe 主要的特点是可扩展性和高容错性。

图 2-1 是 Scribe 的整体架构，主要包括三部分：Scribe 代理、Scribe 服务器和存储系统。Scribe 代理是一个 thrift 客户端，通过它把应用程序的日志数据经由 thrift 接口发送给 Scribe 服务器；Scribe 根据配置信息，把不同 topic 的数据发送到不同的

① Scribe, https://github.com/facebookarchive/scribe

存储对象；存储系统支持很多存储对象，如文件、数据库、网络等等。

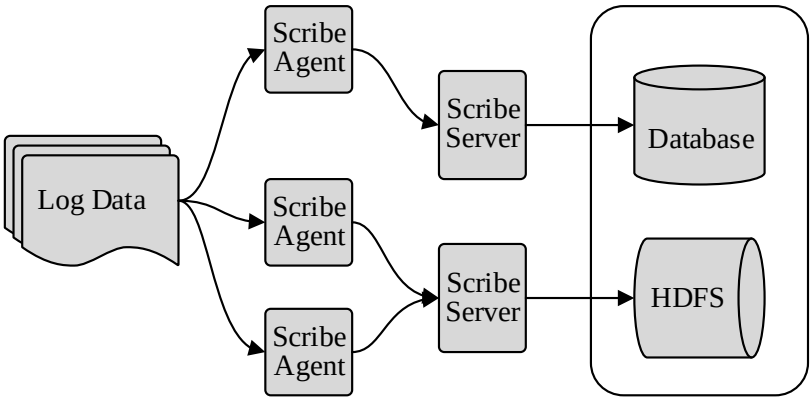


图2-1 Scribe 基本架构

Figure 2-1 Framework of Scribe

2. Chukwa①

Chukwa 是 Apache 基金会下的开源项目，是 Hadoop 家族产品的一部分，用于对大型分布式系统日志进行监控和收集。Chukwa 构建在 Hadoop 的 HDFS 和 MapReduce 编程框架上，因此对 Hadoop 有天然的支持，并且继承了 Hadoop 伸缩性和鲁棒性。Chukwa 能够动态收集不同来源的数据，具有高性能的存储系统，集成了数据分析和展示功能。

图 2-2 是 Chukwa 的基本架构，主要包含四个组件：agents、collectors、MapReduce jobs 和 HICC。其中 agents 运行在每台客户机器上，把每台机器上的日志数据收集和传输到 collectors；collectors 接收汇集多个 agents 传来的数据并把数据写入暂存设备；MapReduce jobs 对暂存设备中的数据进行解析和过滤，并把最终结果写入到存储系统；HICC 是前端接口，展示检索和分析的结果。

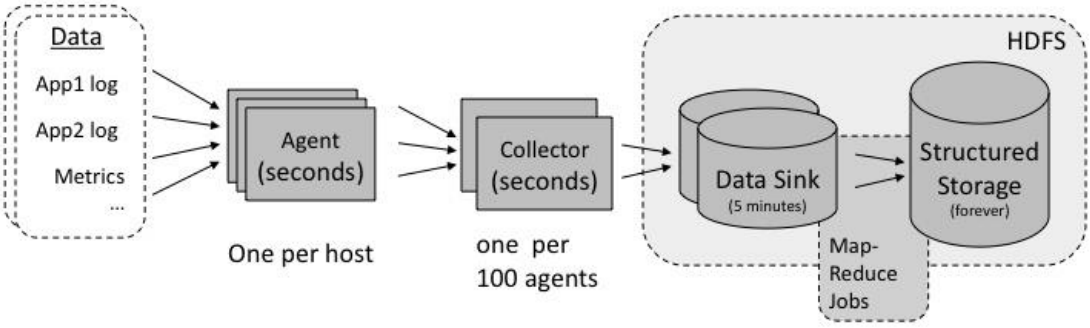


图2-2 Chukwa 基本架构[24]

① Chukwa, https://chukwa.apache.org/

Figure 2-2 Framework of Chukwa [24]

3. Flume①

Flume 最早是 Cloudera 开发的日志收集系统，于 2009 年开源，目前也是隶属 Apache 的开源项目。Flume 功能比较完善，同样提供了从多种来源收集并存储日志数据的功能，并且能对数据进行简单处理。Flume 的特点是高可靠、可扩展和易管理。

图 2-3 展示了 Flume 的基本架构。Flume 采用三层架构，包括 agent、collector 和 storage，其中 agent 和 collector 均由 source 和 sink 两部分组成，分别处理数据来源和去向。agent 把日志数据发送到 collector，collector 将数据汇集后传输到 storage 进行存储。

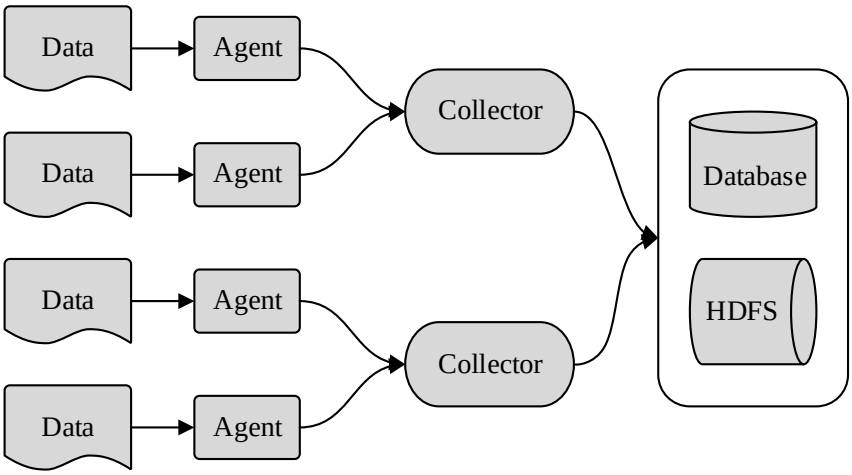


图2-3 Flume 基本架构

Figure 2-3 Framework of Flume

2.2.4 常用日志分析方法

早期的日志分析工作由于日志规模不是很大、对日志信息挖掘的要求不高，分析工作主要由开发管理人员手工进行。这是一个单调乏味的过程，分析人员逐条查看日志数据，期望发现一些关键内容或日志信息之间的关联。为了提高效率会借助一些文本分析工具如 Linux 平台上的 grep、tail、awk 等。grep 是 Linux 平台上最常用的文本过滤工具，能够根据关键字或正则表达式对日志进行搜索和过滤，列出包含目标模式的每一行；tail 命令主要用来查看文件的最末几行，由于文本文件类型的日志都是按顺序每次在文件末添加一行，通过 tail 命令就可以查看最近生成的日志，有实时查看的效果。想要查看文件的其他部分可以用 head、cat、sed 等命令；awk

① Flume, https://flume.apache.org/

是 Linux 平台上另一个强大工具，有着很高的处理效率。awk 有自己语法，通过编写 awk 程序能够实现非常强大的功能，因此可以用作日志分析工具，从日志文件提取或隔离信息。在 Windows 平台上除了一般的文本查看工具外，还有事件查看器、Log Parser 等，Linux 平台上的 grep、awk 等工具在 Windows 上也有对应的移植版本。其中的 Log Parser 日志解析器提供了对文本数据的查询访问，以及 Windows 事件日志等二进制日志记录的查询访问。

随着日志数据的容量和类型不断增长，分析人员早已无法跟上软件日志信息的生成速度，日志内容的人工实时分析变得不再可能，自动化的日志分析技术和工具成了日志分析的必然发展趋势。自动化能够确保与数据流一样的速度对数据进行收集和分析，并对分析得到的结果进行快速响应。其中机器学习算法和数据挖掘技术是分析自动化和提升分析效果的关键，作为处理海量数据的有效方法和计算机智能化的有效手段，通过日志分析，可以很好的完成知识发现、检测系统异常、预测系统状态等任务。主要的方法有关联规则挖掘、序列模式挖掘、分类和聚类等。

1. 关联规则挖掘

关联规则挖掘（Association rule mining）是很常用的数据挖掘方法，用来发现事物之间的联系。关联规则指的是如同$A \rightarrow B$的蕴含式，A 和 B 分别称作关联规则的前导和后继，由 A 推导出 B 的可能性称作置信度和支持度，用来度量关联规则的确定性和有用性。两个事物存在的关联性可以通过蕴含规则进行预测。关联规则挖掘包括两个阶段，首先按照预定义的最小支持数要求，从项目集合中找出所有频繁项集，然后在这些频繁项集中产生满足最小支持度和最小置信度的强关联规则[25]。常见的关联规则挖掘算法有 Apriori 算法、基于划分的算法、FP-growth 算法等。

关联规则挖掘在日志分析中主要用来发掘不同日志条目之间的关系。假设两个日志集合 A 和 B 分别代表两个不同类型，集合 A 中的日志是软件系统在正常状况下产生的，而集合 B 包含的是警告或错误日志。如果存在关联规则$A \rightarrow B$，则可以通过 A 类型日志的出现预测到 B 类型日志的出现，即预测系统发生了异常。

2. 序列模式挖掘

序列模式挖掘指挖掘相对时间关系或其它模式的频繁模式，与关联规则挖掘相比，序列模式挖掘强调规则中事物间的前后序列关系，例如事件 X 和 Y，事件 X 发生一段时间后事件 Y 很可能会发生。序列模式挖掘同样要指定一个频繁子序列的最小支持度。常见序列模式算法主要 Apriori 算法的变种，包括 AprioriAll 算法、AprioriSome 算法、GSP 算法等。

序列模式可以在日志分析中挖掘不同类型日志之间的序列关系。由于日志流本

身就是天然的时间序列数据，因此相较关联规则挖掘，序列模式挖掘更适合通过预测将要出现的日志类型来对系统状态进行判断。通过对网络日志进行序列模式挖掘，还能发现和建立网络领域中的前后依赖关系模型，进行行为建模，优化用户体验。

3. 分类

分类是机器学习中最常见的一个问题，属于预测问题的一部分，当预测目标 y 为有限几个离散值时，预测问题就成为了分类问题。分类学习是监督学习的一种，需要一个训练数据集，集合中的每个样本包含特征值 x 和输出值 y（标签），然后从这些数据中学习出一个分类模型。在对新的数据点x_{new}进行分类时，通过训练得到的分类器对数据类别 y 进行预测。很多机器学习算法可以用于分类，例如 k 近邻法、朴素贝叶斯法、决策树、支持向量机、神经网络等。

可以通过分类算法对日志类型进行分类操作，如果当前日志被分类为代表系统警告或错误的日志，则可以据此发出系统异常警告。进行这样的分类需要预先定义好日志类型以及一个训练集，训练集中包含了已经归类的日志样本。

4. 聚类

上面所说的分类问题属于监督学习，在实际操作中需要准备好训练数据集和日志数据模式。然而我们知道如今软件系统产生的不仅数量巨大，而且类型复杂，要为这样规模的日志定义类型，并为样本中的每条数据贴上标记，不仅需要相关领域的专家，还需要耗费大量的时间。因此无须人工提供训练集的非监督学习更适于解决这类问题。

聚类是典型的非监督学习，不需要预定义的类和标记好的训练数据。根据对象之间的相似度，聚类将对象的集合划分成多个组，每个组中的对象彼此相似，不同组中的对象相似度较低。传统聚类分析可以分为划分方法（K-means、K-medoids 等）、层次方法（BIRTH、CURE 等）、密度方法（DBSCAN 等）、基于网格的方法和基于模型的方法等。

用聚类替代分类操作可能降低了归类精度，却有效避免了标记海量训练数据的过程。通过聚类操作，我们可以把日志数据划分为不同类型，并通过实验对关键类型进行标记。层次聚类是比较适于进行日志聚类操作的方法。在聚类之前可能并不清楚日志类型的数目，通过层次聚类可以控制聚类粒度，达到比较好的聚类效果。在日志预处理一章，我们便提出了基于日志有效负载相似度的层次聚类方法。

2.3 本章小结

本章介绍了日志基本概念和日志监控的相关技术基础。首先在日志概述中，介

绍了日志的定义和功能，以及日志的特点，日志的特点决定了日志自动化分析处理的必要性。随后在日志监控相关理论技术的介绍中，首先定义了日志监控是对日志流进行跟踪、汇集、过滤、存储、分析结果的一系列过程。随后介绍了日志基于 push 和 pull 的两种收集方式，以及日志存储的存储格式和存储位置。接着列举了三种开源的日志收集、存储系统：Scribe、Chukwa 和 Flume。最后介绍了日志的人工分析方法和自动分析技术，列举了关联规则挖掘、序列模式挖掘、分类和聚类四种常用的基于机器学习和数据挖掘的分析方法。

第三章 日志数据预处理

本文提出的日志分析技术主要分为日志数据的预处理和行为异常的检测两个部分，本章介绍预处理部分。我们设计的预处理技术包括日志正规化和日志信息聚类两个步骤。首先通过日志正规化统一日志格式并对日志去参数化，经过去参数化后，拥有不同内容的日志种类数量大大约简；然后通过层次聚类算法把日志划分为不同类型。传统的日志聚类是为了发现聚类结果中的离群点，据此进行异常检测，而本文是为了挖掘不同类型日志的行为模式。文中提出的聚类相似度定义和剪枝策略提高了聚类准确率，优化了聚类粒度。

一般的数据挖掘和分析可分为四个阶段[26]：问题定义、数据预处理、知识发现以及结果的解释和评估。在进行数据分析和结果评估之前，首先要对数据进行预处理操作。这主要是因为现实世界中的数据很容易受到噪声、数据丢失和数据冗余的侵扰，这样的数据会导致低质的数据挖掘和分析结果。数据的质量包括很多因素，例如准确性、完整性、数据一致性、时效性、可信性和可解释性等[25]。针对这些因素提出的数据预处理技术包括数据清理、数据集成、数据规约和数据变换。

3.1 日志预处理的必要性

我们所分析的软件系统日志数据是由开发者编写，本意也是由人工阅读查询，目的是在系统运行中发生错误或崩溃之后，对系统进行功能错误的诊断和错误原因的分析。为了对日志数据进行自动化的分析挖掘，同样需要预处理操作。日志数据预处理的必要性体现在：

1. 日志可能出现数据缺失。当所监控的客户系统处于高负载或发生错误的情况时，日志信息可能出现丢失或者不完整的情况。这是不可避免的，与系统中运行的其他功能模块相比，日志系统通常只有较低的优先级，为了保证系统系能和稳定性，日志系统通常是最先被舍弃的部分。但日志缺失的情况并不会经常发生，已有的研究也表明较少的数据缺失几乎不会对最终分析结果产生影响，因此通常的解决方式是直接忽略掉这部分数据。

2. 日志结构不固定。虽然现在各个平台都有了格式化生成日志记录的插件，但不同的插件格式并没有遵循一个公共的约定。而且日志内容中最重要的有效负载（payload）部分仍然是由程序员以自然语义手动编写的，其格式和内容都由程序员自己决定。这个问题需要通过正规化方式得到解决，本章第三节将详细讨论这个问题。

3. 日志数量巨大。如今的日志规模已经升级到海量级别，传统的基于语义逐条对日志内容进行分析判断的日志分析系统已经很难承受如此规模的日志流。本文提出的分析技术从另一个角度出发，分析每一类日志数据的生成规律，并以此为基础进行异常检测。面对海量且无类型标记的数据，首先要做的就是对日志进行聚类操作，使每条日志归属于特定的类型。本章第四节提出的层次聚类方法将用于解决这个问题。

3.2 实验数据介绍

为了对本文提出的数据预处理方法和后续的异常检测方法进行实验测试和结果评估，本文主要使用了两个真实数据集：Hadoop 集群事件日志和 LANL 数据集[28]。

Hadoop 集群事件日志收集于某高性能计算中心，数据集包含了 Hadoop 集群 NameNode 节点从 2015 年 2 月到 5 月发生的所有事件记录。其日志样本如图 3-1 所示：

```
2015-01-19 22:50:14,427 INFO org.apache.hadoop.hdfs.server.blockmanagement.
CacheReplicationMonitor: Scanned 0 directive(s) and 0 block(s) in 1
millisecond(s).

2015-01-19 22:50:25,988 INFO org.apache.hadoop.hdfs.server.namenode.
FSNamesystem: Roll Edit Log from 192.168.11.97

2015-01-19 22:51:21,258 INFO BlockStateChange: BLOCK* addToInvalidates:
blk_1073792881_52057 192.168.12.174:50010 192.168.12.18:50010
192.168.12.178:50010

2015-01-19 22:51:22,467 INFO org.apache.hadoop.hdfs.StateChange: BLOCK*
BlockManager: ask 192.168.12.174:50010 to delete [blk_1073792880_52056,
blk_1073792881_52057, blk_1073792877_52053, blk_1073792878_52054]

2015-01-19 22:51:44,426 INFO org.apache.hadoop.hdfs.server.blockmanagement.
CacheReplicationMonitor: Rescanning after 30000 milliseconds
```

图3-1 Hadoop 集群事件数据样本

Figure 3-1 Log sample of the Hadoop event data

LANL（Los Alamos National Laboratory）数据集是由 Los Alamos 国家实验室公开的日志数据集，包含了数个高性能计算系统的事件及错误日志。其日志样本如图 3-2 所示：

```
465980,node-129,unix.hw,state_change.unavailable,1145652795,1,Component
State Change: Component \042alt0\042 is in the unavailable state (HWID=4709)
2558399,node-127,action,start,1073118369,1,wait  (command 1894)
2568662,node-137,action,start,1074119886,1,clusterAddMember  (command 1899)
160862,2390,boot_cmd,new,1077875740,1,Targeting domains:node-D0 and
nodes:node-[0-31] child of command 2370
2595701,node-31,node,status,1074288817,1,not responding
```

图3-2 LANL 数据集样本

Figure 3-2 Log sample of LANL data set

表 3-1 对两个数据集信息作了总结：

表3-1 实验数据集信息

Table 3-1 Information of experiment data set

Logs	Records	Size	Log Fields
Hadoop Logs	945818	131.2MB	3
LANL Data	433490	31.5MB	7

3.3 数据正规化

虽然不同日志信息有不同的结构格式，但除了有效负载部分之外，其他域的内容是大同小异的。大多数日志消息由 3 到 10 个域组成，包括时间戳、有效载荷、日志级别、节点 ID、进程 ID、源地址、目标地址、错误代码等等。这些域由空格、逗号或其他符号进行分隔，行与行之间一般由换行符分隔。过去的研究曾讨论过一些正规化方法[6]，例如删除冗余字符、识别日志边界、统一时间戳格式等。为了使日志便于自动化处理，我们总结了已有的研究并提出了自己的正规化转换方法。

1. 重排不规范日志记录

重排不规范记录包括去处冗余字符，把跨越多行的记录调整为一行。很典型的一个情况是当客户系统在运行中发生错误并抛出一个异常时，日志不仅记录错误发生的时间和位置，还会打印抛出异常的函数及其调用栈，调用栈的每个函数占用一行，如图 3-3 所示：

```
2015-03-30 14:55:52,592 FATAL org.apache.hadoop.hdfs.server.namenode.
NameNode: Exception in namenode join
java.io.IOException: Failed on local exception: java.net.SocketException:
Unresolved address; Host Details : local host is: "cluster1"; destination host
is: (unknown):0;
    at org.apache.hadoop.net.NetUtils.wrapException(NetUtils.java:764)
    at org.apache.hadoop.ipc.Server.bind(Server.java:419)
    at org.apache.hadoop.ipc.Server$Listener.<init>(Server.java:561)
    ...
```

图3-3 系统抛出异常时的日志记录

Figure 3-3 Log records when exceptions are thrown

针对这种情况有两种解决方法，一是直接删除调用栈记录所占的多余几行，二是把这几行记录统一合并到一行，即删除行与行之间的换行符，并用其他分隔符代替。我们更倾向于选择第二种方法，以便于为后面的分析操作保留足够多的有用信息。

2. 把日志级别转化为数字表示

很多日志格式都包含日志级别，日志级别反应了日志事件的严重性或值得重视的程度。日志级别主要包含如下几项，其级别由高到低：

- FATAL – 十分严重的、造成服务器中断或系统崩溃的错误；
- ERROR – 一般运行错误，可以允许应用继续运行；
- WARN – 表明系统存在异常，可能导致潜在的错误；
- INFO – 打印应用运行过程中的一般事件；
- DEBUG – 细粒度的有助于应用调试的事件；

为了方便日志聚类时候的相似度测量，我们为每个日志级别赋了一个数字值，例如 DEBUG 等于 1 一直到 FATAL 等于 5，相邻的日志级别有相近的值，表明其严重程度相近。

3. 日志内容去参数化

Vaarandi 等人[8]在对日志文本出现单词的统计中发现，大多数单词在日志样本中只出现一到两次，而少部分单词在文件样本中则出现得十分频繁，并且这些频繁出现的单词间还存在着关联。出现这种情况主要是由于开发人员在编写日志内容时使用参数模板进行了格式化输出，例如：

printf(“Connection established from %s.”, parameters);

这些参数可能是 IP 地址、进程 ID、简单的数字或内存地址等等。日志分析系统因为日志消息中不同的参数值可能把相同类型的日志误判为不同类型。为了解决这个问题，我们使用了去参数化的方法，把每条日志中出现的数字参数用关键字占位符替代，例如把 IP 地址的值替换为占位符“ip”，把进程 ID 的值替换为“processid”。图 3-4 是去参数化的一个例子，通过去参数化，同一类型的不同日志记录被转换成了相同的文本。

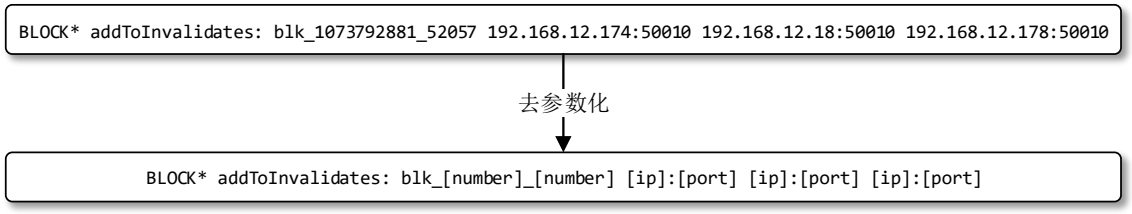


图3-4 去参数化示例

Figure 3-4 De-parameterize a log record

3.4 日志信息聚类

本文提出的异常检测技术需要分析每类日志的生成特征并从中提取出行为模式。如果数据集中的每一条日志记录都表示一个类型，那我们的检测算法可能需要处理包含上百万个特征变量的特征空间。而且，由于每条记录都表示一个独特的类型，我们将很难获得任何行为模式，因为一个类型在一段时间内可能只会出现一两次，无法总结出任何规律。只有存在足够的数据样本，才能发现隐藏在特定日志类型背后的行为特征。这就是我们进行日志聚类的目的。

3.4.1 聚类方法分析

在上一章的介绍中我们说过，聚类是典型的非监督学习，根据对象之间的相似度，聚类将对象的集合划分成多个组或簇，每个组中的对象彼此相似，不同组中的对象相似度较低。典型的聚类步骤包括特征选择、特征提取、相似度选择、分组、聚类结果评估[30]。

- 特征选择：对数据进行特征准化和降维，并从中选中最有效的特征。
- 特征提取：对选择的特征进行数据转换，形成新的突出特征。其结果可表示为一个矩阵，其中行代表样本，列代表特征变量。
- 相似度选择：对相似度标准进行选择，找出最适合特征类型的距离函数或构造新的距离函数。

- 分组：执行聚类算法，对数据进行分组。算法的输入是样本矩阵，输出可以是树形图或具体分类方案，以不同的粒度反映分类情况。
- 聚类结果评估：凭借经验或领域知识划定分类阈值，得到最终聚类结果，评估聚类有效性。

由于不同的数据集有不同的类型、结构和表示方法，因此没有任何普适的聚类算法可以对所有数据集进行聚类并取得好的分类结果。应针对不同数据集的特征选择最合适的聚类方法。好的聚类方法应该满足这些要求[25]：

- 可伸缩性：在包含几百个对象的小数据集和在包含几百万甚至上亿对象的海量数据集上都有良好的聚类表现。
- 处理不同属性类型：可以处理数值类型或正/负的、分类的、文本等不同数据类型。
- 发现任意形状的簇：一个组或簇可能是任何形状，好的聚类算法能够发现任何形状的簇。
- 不要求过多的领域知识：聚类算法可能要求用户为算法提供领域知识，而聚类结果会依赖这些参数，使聚类结果不够客观。
- 处理噪声数据：大部分数据集可能包含噪声，而噪声数据可能对聚类结果产生很大影响，因此需要降低算法对噪声的敏感性。
- 对输入方式不敏感：数据输入过程可能包括数据的增量更新或次序的改变，聚类算法不能因为这些变化影响聚类效果。
- 聚类高维数据：许多算法只擅长处理低维度数据，而高维的数据包含了更多细节特征，算法应提高对高维数据的聚类能力。
- 基于约束聚类：现实中的数据聚类可能需要满足不同约束条件。聚类算法应在满足诸多约束的条件下保持聚类准确率。
- 可解释性和可用性：聚类的输出结果要能向最终用户解释并且能够使用。

按照不同的数据凝聚方式和实现这些方式的不同方法，主要的聚类算法可以划分成如下几类[25]：

1. 划分方法

在划分方法中，大小为 n 的数据集被划分为 k 个组，$k \leq n$，其中每个组包含至少一个对象。在基本划分中，每个对象也只对应一个组，但在模糊划分中限制不那么严格。基于距离的划分是主要的划分方法。好的划分满足同一个组中的对象距离尽量相互接近，而不同组中的对象尽量相互远离。常用的划分方法有 K-means 和 K-medoids，两者的核心思想相同，都是找出 k 个聚类中心$t_1, t_2, \ldots t_k$，并使数据集中

每一个点x_i和离它最近的聚类中心t_x的距离之和最小化。两者的区别在于中心点的选取，在 K-means 算法中，中心点取每个组中所有数据点的坐标平均值，而在 K-medoids 算法中，中心点是每个组中的某一个数据点，这个点到该组中其他店的距离之和为所有点中最小。

2. 层次方法

层次聚类根据组间距离度量标准，使用数据的联接和层次的划分方式来获得满足给定终结条件的簇。根据层次划分的形成方式，层次聚类方法可以分为两类：基于凝聚的方法和基于分裂的方法。凝聚的方法也称为自底向上的方法，首先视每个数据点为一个簇，选择相邻的两个簇进行合并，接着更新簇与簇之间的距离，然后重复此过程，直到满足聚类的终止条件；分裂的方法也称为自顶向下的方法，首先把所有的数据点视为一个簇，然后对每个簇进行划分，形成更小的簇，直到满足聚类终止条件。层次聚类的优点是不必预先指定数据要聚类为多少簇，只需要给定聚类的终止条件，增大了聚类的灵活性。

3. 基于密度的方法

大部分聚类方法主要发现球形簇，但簇的形状不止球形，可以是任何形状。基于密度的方法可以通过数据密度发现任意形状的簇，其主要思想是给定一个簇，如果这个簇相邻空间数据点的密度大于某个阈值，则继续增大这个簇。这种方法可以过滤噪声数据或离群点，并能发现非球状的簇。典型的算法包括 DBSCAN、OPTICS 和 DENCLUE。

4. 基于网格的方法

基于网格的方法由空间驱动，使用一个网格结构，把整个数据空间划分为矩形块网格单元，聚类过程就在该网格结构上进行。基于网格的方法常与密度或层次方法相结合，根据这些规则对网格进行聚类。这种方法的优点是聚类速度快，聚类时间只依赖于网格数目，独立于数据点个数。典型算法有 STING 和 CLIQUE。

3.4.2 基于层次方法的日志聚类

通过上一节对聚类概念的理解和几种主要聚类算法特点的分析，结合日志数据的类型、结构和格式特点，我们决定使用层次方法对日志进行聚类。使用层次聚类主要基于如下原因：

1. 日志数据由文本组成。与数值型数据不同，基本上大部分软件系统日志都是由普通文本组成。如果要使用如 K-means 等划分方法，必须基于某种策略把日志文本转换成为数值向量才行，而层次聚类就没有对数据类型有任何要求，只要给出数

据点之间的距离或相似度定义就能进行聚类。

2. 日志没有维数概念。与普通的如对 p 维多变量数据矩阵进行聚类不同，大部分日志类型并没有维数的概念。日志的绝大部分信息都集中在有效载荷部分，并且很难从这部分内容中提取出多维特征，因此 K-means 等方法同样不适用。层次聚类没有这样的顾虑，它只需要定义一个相似度矩阵。

3. 日志类型数目未知。日志包含哪些类型通常是未知的，而通过专家系统或领域知识同样很难对日志类型数据进行预估。层次聚类不依赖预定义的类别数目等参数，且可以通过设置终止条件灵活确定最终的聚类数目。

上一节提到层次聚类分为基于凝聚的方法和基于分裂的方法。以凝聚算法 AGNES 和分裂算法 DIANA 为例，图 3-5 表示了两类方法对包含五个对象的数据集的层次处理过程。AGNES 方法首先把每个对象各作为一个簇，然后根据某种规则逐步进行合并。合并过程不断进行，最终整个数据集形成一个簇；DIANA 方法刚好与之相反，一开始所有对象为一个初始簇，之后根据规则反复分裂，直到每个簇只包含一个数据点。

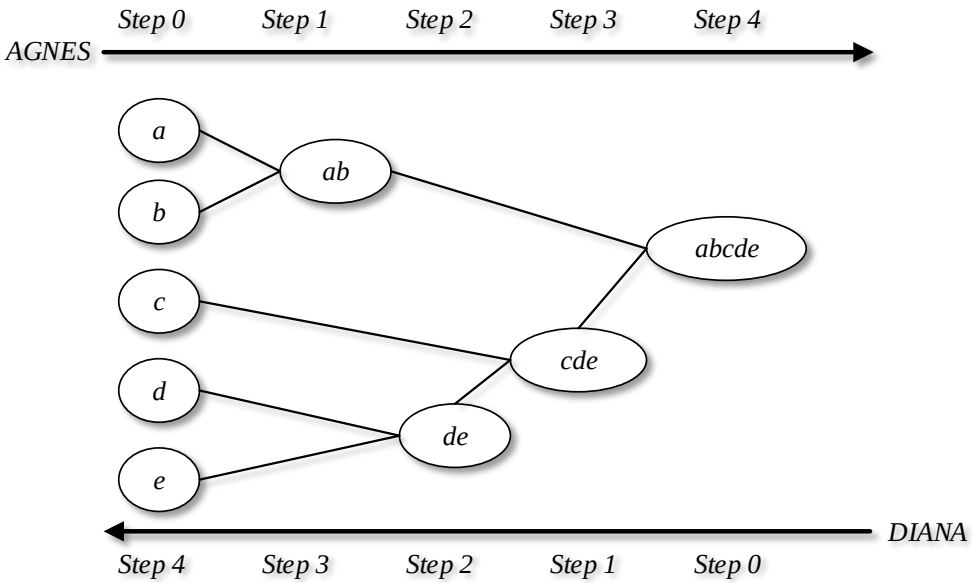


图3-5 凝聚和分裂聚类示例[25]

Figure 3-5 Sample of agglomerative and divisive clustering [25]

然而大多数的聚类操作采用的都是自底向上的凝聚算法，这主要是由于相比凝聚算法，分裂算法的平均时间复杂度都比较高[33]，实际上并没有得到广泛使用。因此本文提出的日志聚类算法也是采用自底向上的凝聚方式。凝聚方式的聚类具体包

含如下步骤：

- 把数据集划分为 n 个簇，每个簇包含一个数据点。计算所有簇之间的距离，得到$n \times n$的相似性矩阵；
- 找到矩阵中距离最小的两个簇：$D(c_i, c_j) = \min\limits_{\substack{1 \le m,l \le n \\ m \ne l}} D(c_m, c_l)$，合并$c_i, c_j$，形成新的簇；
- 更新相似性矩阵；
- 重复第二、第三步，直到满足终止条件或所有数据点都在一个簇。

确定了聚类方式后，接下来要选择采用何种距离度量方法。使用聚类算法的一个核心问题就是如何度量不同簇之间的距离。设 p 和 p'是数据集中的两个数据点，$|p - p'|$代表两个点之间的距离或相异程度，m_i代表簇c_i中数据点坐标的平均值，n_i是簇c_i中数据点的数目。通常有四种簇间距离度量方法及对应的算法：

1. 单连接（single linkage）

单连接聚类也称作最近邻聚类算法，它使用最小距离$dist_{min}(c_i, c_j)$作为距离度量标准：

$$dist_{min}(c_i, c_j) = \min_{p \in c_i, p' \in c_j} \{|p - p'|\} \tag{3-1}$$

最小距离是两个簇中任意两点之间距离的最小值。当两个簇之间的最小距离超过给定的阈值时，聚类过程就会停止。单连接算法善于聚类非椭圆结构的簇。

2. 全连接（complete linkage）

全连接聚类也称作最远邻聚类算法，它使用最大距离$dist_{max}(c_i, c_j)$作为距离标准：

$$dist_{max}(c_i, c_j) = \max_{p \in c_i, p' \in c_j} \{|p - p'|\} \tag{3-2}$$

最大距离是两个簇中任意两点之间距离的最大值。当任意两个簇之间的最大距离超过距离阈值时聚类过程便停止。全连接算法使用于大小近似且分布较为紧凑的簇。

3. 重心连接（centroid linkage）

重心连接使用均值距离$dist_{mean}(c_i, c_j)$作为距离标准：

$$dist_{mean}(c_i, c_j) = |m_i - m_j| \tag{3-3}$$

均值距离是两个簇的中心之间的距离，簇的中心是簇中所有点坐标的均值。

4. 类平均（group average linkage）

类平均连接使用类平均距离$dist_{avg}(c_i, c_j)$作为距离标准：

$$dist_{avg}(c_i, c_j) = \frac{1}{n_i n_j} \sum_{p \in c_i, p' \in c_j} |p - p'| \quad (3\text{-}4)$$

类平均距离是两个簇中任意两个点间距离的平均值。使用平均距离的优势是既可以处理数值型数据，也可以处理分类数据。

通过对不同距离标准的分析，结合日志数据的特点，我们最终决定采用全连接 complete linkage 作为度量标准，这主要是由于全连接可以产生十分紧凑、直径较小的簇，计算并行度高，并且满足相同簇中每条日志数据都与其他数据有较高相似度的要求。

3.4.3 相似度计算

在确定了簇间距离度量标准后，这一节我们将讨论数据点间距离本身的定义。设 p 和 p' 是数据集中的两个数据点，$|p - p'|$代表这两个点的距离或相异程度。一般的距离定义包括欧几里得距离，闵可夫斯基距离等等。日志信息主要由文本字符串组成，而度量两个字符串间相似度或距离的标准之一就是他们的编辑距离[34]。

编辑距离，也称为 Levenshtein 距离，是指在两个字符串之间通过字符的插入、删除或替换等编辑操作，把一个字符串转换成另外一个字符串所需的最少编辑次数。编辑距离越大，两个字符串之间的相似度就越小，字符串之间的相似度可以定义成编辑距离的倒数。通过观察日志内容可以看到，拥有相同或相似类型的两条日志具有相同的语义特征，尽管可能有少许差异，但其有效载荷的绝大部分是由相同的词语按相同的顺序构成的。也因此我们可以使用编辑距离定义两条日志的相似度，已有的研究[13]也证明了使用编辑距离计算日志之间相似度具有很好的效果。

计算编辑距离可以有很多方法，我们给出了一个基于动态规划的算法。动态规划是多阶段的决策最优化算法，通过把要求解的问题分解成子问题，完成了求解子问题的各个子阶段后，原问题也得到了解答。动态规划避免了不必要的重复计算，有效地节省了时间。而通过延迟验证等方法，动态规划求解编辑距离的速度可以得到进一步提高。下面说明算法的状态转移关系。

设两个字符串$S_1[1 \dots m]$和$S_2[1 \dots n]$。S_1长度为 m，S_2长度为 n，分别代表两条日志的有效载荷部分，$ED(i, j)$表示S_1的前缀$S_1[1 \dots i]$和S_2的前缀 $S_2[1 \dots j]$之间的编辑距离。

初始状态：

$$ED(0, 0) = 0 \quad (3\text{-}5)$$

$$ED(i, 0) = i, 1 \le i \le m \quad (3\text{-}6)$$

$$ED(0, j) = j, 1 \le j \le n \quad (3\text{-}7)$$

状态转移方程，其中$S(i)$表示字符串 S 的第 i 个字符：

$$ED(i,j)=\begin{cases} ED(i-1,j-1), if\ S_1(i)=S_2(j) \\ min\big(ED(i-1,j),ED(i,j-1),ED(i-1,j-1)\big)+1, if\ S_1(i)\neq S_2(j) \\ 1\le i\le m, 1\ \le j\le n \end{cases} \quad (3\text{-}8)$$

其中初始状态显然成立。然后在状态转移方程中，$S_1[1\dots i]$和 $S_2[1\dots j]$的编辑距离$ED(i,j)$取决于两种情况：当$S_1(i)=S_2(j)$时，表示两个字符相等，不需要任何编辑，因此编辑距离等于前缀$S_1[1\dots i-1]$和 $S_2[1\dots j-1]$的编辑距离；当$S_1(i)\neq S_2(j)$时，可以对其中一个前缀添加或删除一个字符，或替换$S_1(i)$或$S_2(j)$为对方字符，这几种操作都能使前缀重新相等，但都增加了一次编辑操作，最后得到的编辑距离需要取其中较小的一个。

通过 DP 算法我们得到了S_1和S_2的编辑距离$ED(m,n)$。为了便于后续计算，我们进一步把$ED(m,n)$正规化至$[0,1]$范围，我们用编辑距离除以两个字符串中较长字符串的长度，得到的结果记为 *Edit Ratio (ER)*：

$$ER(i,j)=\frac{ED(i,j)}{max(Length(S_i),Length(S_j))} \quad (3\text{-}9)$$

如果日志记录包含日志级别，那还可以加上日志级别的相似度。在日志正规化步骤中，日志级别按其严重性依次替换为 1 开始的自然数。例如为 TRACE 级别的日志赋值 1，为 ERROR 级别的错误日志赋值 5，日志级别的相似度可以由其数值间的差反应出来。给定两条日志 i 和 j，日志级别的距离定义为

$$LD(i,j)=|T_i-T_j| \quad (3\text{-}10)$$

其中T_i和T_j表示 i 和 j 的日志级别。

对$LD(i,j)$除以日志级别数目后，得到正规化结果，记为$LR(i,j)$：

$$LR(i,j)=\frac{LD(i,j)}{NumOf(T)} \quad (3\text{-}11)$$

最后结合两个公式，我们得到了两条日志数据的距离定义，即相异度定义。相异度记为 *Dissimilarity Ratio (DR)*：

$$DR(i,j)=\frac{ED(i,j)}{max(Length(S_i),Length(S_j))}+\alpha\frac{LD(i,j)}{NumOf(T)} \quad (3\text{-}12)$$

其中的系数α用于平衡编辑距离和日志级别距离的权重。基本上不同级别的日志信息其类型也一般不同，因此通常有$\alpha>1$。

定义了数据点相似度和簇间距离后，便可以定义相似性矩阵，使用基于 complete linkage 的凝聚方法对日志数据进行聚类。

3.4.4 最优聚类粒度

凝聚方式的层次聚类最后会终止于预定义的终止条件或使所有数据点聚到同一

个簇，我们需要确定终止条件，使同类型的日志尽量聚到一起，同时区分不同的类，保留数据本来的信息。

层次聚类的结果可以表示成一个二叉树类型的树状图（dendrogram），如图 3-6 所示。

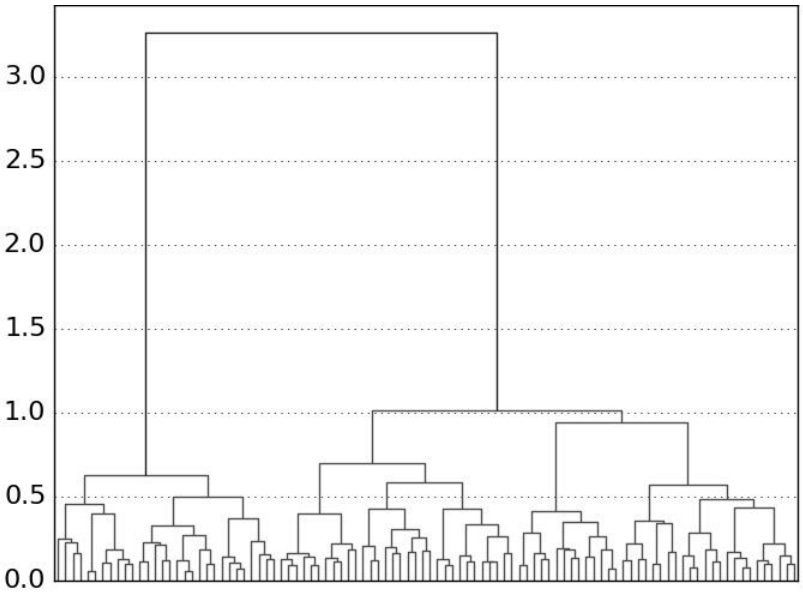


图3-6 层次聚类的树行图表示

Figure 3-6 Dendrogram of hierarchical clustering result

其中图的根节点表示整个数据集，叶节点表示单个数据点。聚类过程从叶节点出发，一步步不断合并，直至根节点为止。从图中可以看到，数据集的最小簇间距离也随着这个过程不断增大。我们的目的是提出一个合适的剪枝策略，按这个策略对整棵树进行剪枝操作，得到的最终结果是每个叶节点都代表一个簇。具体说来就是给出一个簇间距离的阈值，当数据集的最小簇间距离大于这个阈值的时候即终止聚类操作。图中可以看到当阈值等于 1.0 时，聚类结果为 3 个簇，当阈值升至 1.5 时则减少到了两个。

聚类终止条件，即簇间距离阈值，是决定最终聚类效果的一个关键因素。如果距离取得过小，那么将会得到很多个簇，也就是日志类型，这会给后续的异常检测带来计算上的压力。而且若单个簇包含的数据过少，可能无法从中学习到任何行为模式；反之如果距离阈值取得过大，生成日志类型数量过少，会导致最后异常检测精度的退化，因为某些比较细微而重要的行为特征可能由于被合并到一个大的簇中，其行为模式也遭到了隐藏。

图 3-7 是我们对 LANL 数据集进行聚类测试得到的实验结果。其中横坐标代表不同聚类粒度，即簇间距离的阈值；纵坐标则代表簇的个数。通过上一步的正规化操作，我们得到了 675 个初始的簇，接着随着簇间距离阈值的不断增长，簇的数目也跟着减少，最终减少至 1。

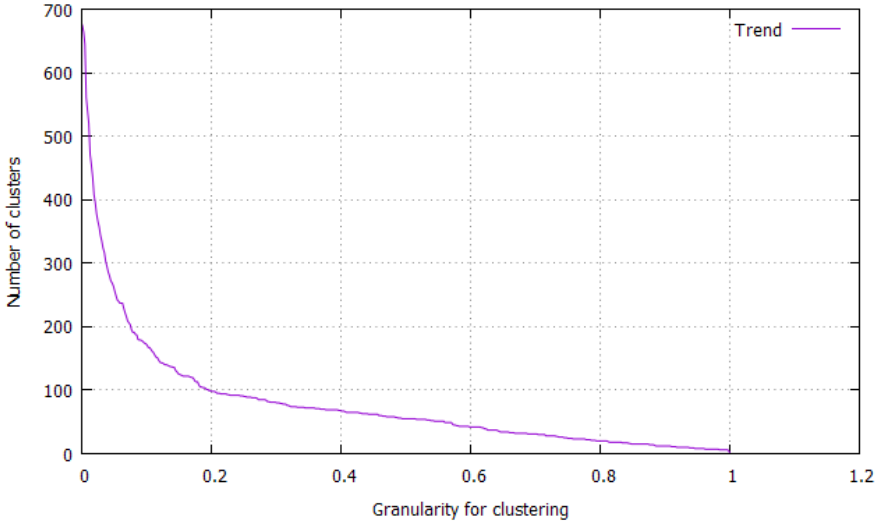


图3-7 不同距离阈值得到的聚类结果

Figure 3-7 Number of clusters with different distance values

通过假设和实验验证，我们持有这样的意见：在聚类过程中，簇的数目随着簇间距离阈值的增大而减少，当阈值到达最优后，生成的簇数目的下降率将迅速减小，簇的数目的变化趋向于平缓。从图 3-7 我们也可以看到，当聚类粒度到达 0.16 后，随粒度下降的曲线开始变的平缓。我们选择这个距离阈值作为聚类的终止条件并进行了实验测试。通过对实验结果的观察，我们的观点得到了很好的验证。例如在对 LANL 数据集进行聚类操作之前，初始簇的数目为 675，通过聚类，最终的日志类型被归为 112 类。在每组结果的观察中我们看到，相同或相似的日志数据被归类到同一个簇，不同组之间的数据在内容和格式上都有较大差别。但是仍然有某些日志记录被错误的归类到了一起，毕竟我们采用的聚类算法属于非监督学习，无法达到万无一失的效果。在实验中我们手动对一些系数和阈值进行了微调，进一步提高了日志数据聚类的准确度（精确率和召回率）。

3.5 实验结果分析

本章介绍的日志预处理方法是两步日志异常检测方法的第一步。我们对之前介绍的 Hadoop 事件记录和 LANL 事件日志两个数据集做了预处理实验。

首先在日志正规化中，日志数据进行了去参数化操作，数值型数据被替换为占位符，接着重新调整了日志结构，删除了冗余数据。正规化后的日志被储存在关系数据库，元组的每一个属性代表了日志记录中的一项。通过对不同日志记录的总数进行统计，可以看到经过了日志正规化的简单操作后，数据类型得到了极为显著的压缩，两个数据集的压缩比都超过了 99%。表 3-2 列出了日志正规化处理后的结果。

表3-2 日志正规化结果

Table 3-2 Result of log normalization

Logs	Original Records	Normalization	Compression Rate
Hadoop Logs	945818	1541	99.94%
LANL Data	433490	675	99.84%

接着进行日志聚类操作，通过凝聚式层次聚类，把日志进一步归类为类簇。在这一步主要对两个参数进行调试，分别是聚类公式中的平衡因子α和簇间距离阈值，两个参数对于最终的聚类效果是至关重要的。表 3-3 列出了日志聚类后的结果，经过聚类后日志类型数目进一步降低。可以看到 Hadoop 数据集在聚类之前有比较多的类型，聚类之后数目反而少于 LANL 数据集，这表明 Hadoop 数据集实际类型较少。

表3-3 日志聚类结果

Table 3-3 Result of log clustering

Logs	Normalization	Clustering
Hadoop Logs	1541	61
LANL Data	675	112

3.6 本章小结

本章阐述了两步日志异常检测方法的第一步：日志数据预处理。首先介绍了日志处理的必要性，日志由于数据缺失、结构不固定、数量巨大等问题需要进行预处理操作。接着介绍了本文主要采用的两个数据集：Hadoop 事件记录日志和 LANL 数据集。然后分别对预处理中的日志正规化方法和日志信息聚类进行了比较详细的介绍，主要介绍了基于 complete linkage 的、以编辑距离为日志数据相似度的凝聚聚类

方法。文中提出的聚类相似度定义和剪枝策略提高了聚类准确率，优化了聚类粒度。最后将预处理方法进行了实验验证，实验结果证明了预处理方法的有效性。

第四章 基于行为模式的日志异常检测

本文提出的日志分析技术分为日志数据预处理和行为异常检测两个部分，上一章介绍预处理阶段所用到的理论和技术，本章继续介绍第二部分。

这一部分我们把重点放在从不同类型的日志数据集中提取出行为序列，通过检测日志流的行为模式是否异常来对系统状态进行判断。文献[14]的研究证明系统在真正发生错误和崩溃之前会经历一系列非致命的异常，这些非致命的异常会通过事件的方式反映在日志记录上，因而不同类型的日志流的频率变化能客观地反映当前的系统状态。以往的研究主要把注意力放在频率本身，通过频率的期望和方差等统计规律做出点异常（point anomaly）判断，而我们侧重于从连续的日志频率子序列中发现行为模式，进行行为异常（behavioral anomaly）判断。本文提出的异常检测算法其核心在于研究给定时间间隔里日志数据的发生频率以及由连续时间间隔组成的滑动时间窗口中频率的变化特征，即日志行为模式。

4.1 连续子序列问题

4.1.1 异常连续子序列

我们给出的解决方案的关键在于把日志异常检测转化成为多变量连续子序列[5]的异常检测问题。基于连续子序列的异常检测是指在一个长序列 L 中检测出包含异常的一段连续子序列 t，判断的标准是基于长序列 L 除 t 以外的部分表现出的序列特征，这里的 t 即是所谓的异常连续子序列。

异常连续子序列[35]（anomalous contiguous subsequence）也称为时间序列不和（time series discords），指的是在一段较长的序列数据或其他序列数据中的这样一段子序列，他的序列形状等特征和长序列中的其他数据相似度最小、区别最大。图 4-1[35]给出了时间序列不和的示例。

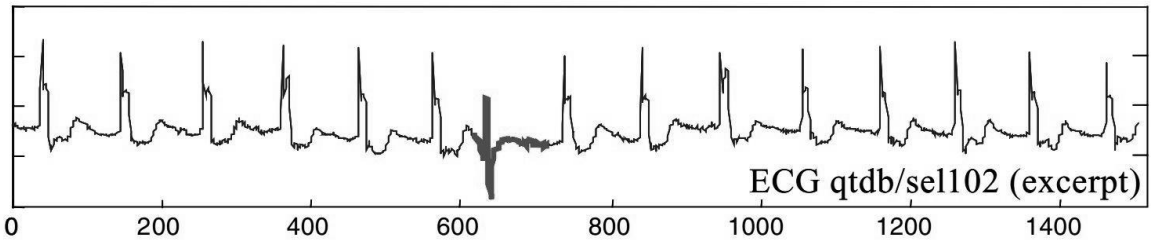


图4-1 异常连续子序列示例

Figure 4-1 Sample of anomalous contiguous subsequence

图中显示的是一段心电图序列，其中加粗的子序列部分与其他子序列相比有明显的波动异常，这也是子序列异常检测的典型应用之一。除了用于心电图检查，子序列异常检测还能用于病情监控、电力需求分析、图像识别、模式识别、以及系统监控等领域。

这里给出一些概念定义：

- 时间序列

一段时间序列$T = t_1, \dots, t_m$是由一系列有序排列的实值变量组成的序列，其中$C = t_a, \dots, t_b$表示一段连续子序列，其中$1 \le a \le b \le m$。时间序列 T 中的任何一段子序列都可能是异常序列，因此在进行异常检测时，一般使用一个滑动窗口从 T 中提取出所有的连续子序列。

- 滑动窗口

给定一个长度为 m 的时间序列 T、一个用户定义的连续子序列长度 $n, 0 < n < m$。所有的子序列可以通过一个类似滑动着的长度为 n 的窗口的方式，对 T 从头到尾，每次滑动一格数据而获得，子序列集合定义为 $C, C = \{C_1, C_2, \dots C_{m-n+1}\}$，共$m - n + 1$个元素。

- 子序列距离

子序列距离的度量可以表示为 *Dist(M, C)*，其中 M 和 C 代表两个输入子序列，通过计算得到的非负结果 R 代表 M 到 C 的距离。距离函数须有交换性，即 $Dist(M, C) = Dist(C, M)$，且两个子序列长度应该相等。

- 非自我匹配

当对一个时间序列 T 的所有子序列进行距离度量时，需要排除临近匹配。这是由于相邻的两个子序列只有一个元素不同，在序列形状上有很高的相似度，而这会影响最终结果的判断。非自我匹配（non-self match）定义如下：给定时间序列 T，要

进行距离度量的两个子序列 C 和 M，其长度均为 n，C 的起始下标为 p，M 的起始下标为 q。如果有$|p-q| \geq n$，则认为 M 是 C 的非自我匹配。

- 时间序列不和

这里给出异常连续子序列，即时间序列不和的公式定义。给定时间序列 T，子序列 D 长度为 n，起始下标为 l，如果序列 D 与其他符合 non-self 匹配条件的子序列间距离是所有子序列中最大的，或超过预定义的阈值，则认为该子序列是异常的，也表示时间序列 T 包含了异常连续子序列。

4.1.2 滑动窗口模型

我们提出的异常检测需要通过滑动窗口提取数据流的子序列。与一般的静态数据集不同，数据流是一种实时连续的、随着时间不断产生和变化的数据序列。由于数据产生和到达的速度不固定，因此单位时间内的数据产生频率也是不断变化的。数据流表示为数列$\{\dots, a_{t-1}, a_t, a_{t+1}, \dots\}$。数据流可以用多种逻辑模型进行抽象表示，例如滑动窗口模型、界标模型、快照窗口模型等。其中滑动窗口是最常用的模型之一，可以降低算法数据规模，降低复杂度，十分适用于进行流数据的分析挖掘，并且常用于描述计算机网络中数据链路层和传输层的流量控制技术。

图 4-2 是一个滑动窗口模型实例。窗口大小为 7，初始时刻位于下标 1 至 7，经过 1 个单位时间间隔后窗口整体向右滑动一格，包含的数据变为 2 至 8。之后同样以相同的方式向右滑动，直至数据流结束。滑动窗口模型以先进先出的形式对数据进行处理，非常适合以线性队列的方式进行实现。

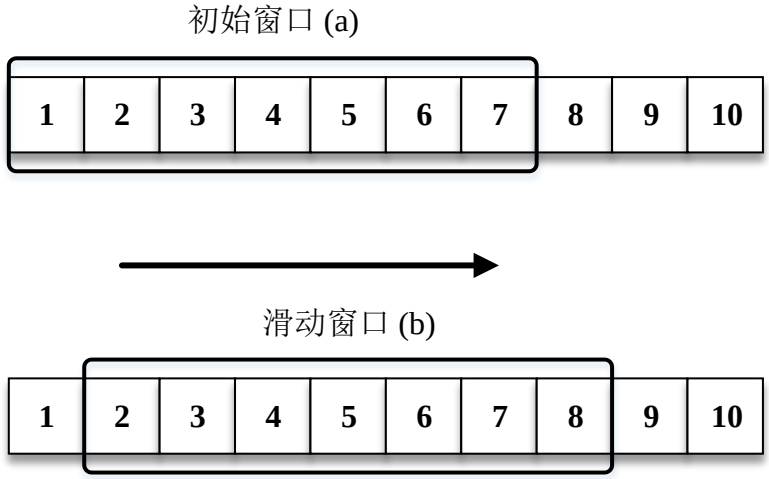


图4-2 滑动窗口实例

Figure 4-2 Sample of sliding window

4.1.3 常用检测方法

把日志数据或其他序列数据用一个多变量时间序列建模后，就可以用连续异常子序列的方法进行异常检测。基本的异常检测算法描述如下：首先定义一个长度为 n 的滑动时间窗口，通过滑动窗口提取出长度 m 的时间序列 T 中所有的连续子序列，子序列的集合记为 C，$C = \{C_1, C_2, \dots C_{m-n+1}\}$，共$m-n+1$个元素，子序列可以存储在文本文件或数据库中。接着要为每个子序列C_k计算异常指数（anomaly score），记为AS_k，AS_k可以是子序列C_k到所有其他子序列的距离的最小值或平均值等，其中要注意避免临近匹配的问题。最后根据专家系统或领域知识定义异常阈值，若子序列C_k的异常指数大于异常阈值，该子序列即被标记为异常子序列。

关于子序列异常指数（anomaly score）的计算方法，Keogh 等人提出了一些改进策略或新的方法。

其中一种策略同样是通过子序列间的距离进行度量，但不是取所有距离中的最小值，而是取与它第 k 近的子序列C_k之间的距离作为异常值，这种方法被称为 HOTSAX[36]。当$k = 1$时，这种算法等同于最小距离算法。

另一个改进算法称为 WCAD（Window Comparison Anomaly Detection）。为了计算时间序列 T 中子序列 W 的异常值，使用一种基于信息论和压缩率的相异度测量方式（Compression Based Dissimilarity, CDM），比较 W 和 T 中除 W 外的时间序列 T'的相异度，CDM 定义为

$$CDM(W, T') = \frac{C(W,T')}{C(W)+C(T')} \tag{4-1}$$

其中$C(W)$是对字符串 W 使用标准压缩算法进行压缩得到的结果。算法的思想在于，如果 W 是正常子序列，那么它会和 T 中剩余部分 T'良好匹配，压缩率会比较高，CDM 值就会比较低；反之若 W 是异常序列 discord，那匹配度和压缩率都会比较低，CDM 值会升高，表明有较高的相异度，即异常指数。

最后一种方法则简单直接。序列数据被分成训练集和测试集，首先按之前提到的步骤，从训练集中获得所有的长度固定的子序列集，然后对子序列集中不同子序列的出现次数进行统计，有相同或相似序列形状的子序列被看做同一类型的子序列。统计结果组成一个键值类型的字典 *Dic*，其键为子序列本身，值是这个序列的出现次数，也即频率。对测试集中每一子序列的异常检测就基于这个字典。令要测试的子序列为 W，$Dic(W)$代表子序列在训练集中出现的频率。给定一个频率阈值λ，若$Dic(W) < \lambda$则直接判断 W 为异常序列。W 的异常指数可以通过其出现频率的倒数来计算：

$$AnomalyScore(W) = \frac{\alpha}{Dic(W)} \quad (4\text{-}2)$$

其中α为正规化因子。这种方法简单快速，同样能得到很好的结果，并且十分适用于流数据的异常检测，本文的日志检测方法也是基于此算法思想。

最后需要注意的问题是滑动窗口大小 n 的选取。确定 n 的最优值是具有挑战性的。若 n 值取得过小，导致子序列过短，所有子序列的出现次数可能都会很高，因此可能找不到异常序列，导致高 false negative 率；反之如果 n 值取得过大，每条子序列都很长，不仅会增大计算压力，并且可能由于相同序列出现频率低导致高 false positive 率。解决这个问题需要取不同值进行多次测试或定义多个不同长度的滑动窗口。

4.2 基于行为模式的异常检测算法

在上一节我们介绍了连续子序列问题，包括异常连续子序列的概念以及通过子序列距离和频率等特性计算异常指数的方法。这一节将介绍的日志异常检测算法就是基于异常连续子序列检测的思想，通过把日志流转化为行为模式序列，用子序列异常检测的方法计算实时日志数据的异常指数，最后据此进行系统监控，预测系统状态。

检测过程大体上分为三步。首先，把训练数据转换为行为序列。在这一步中，根据日志聚类后得到的类型信息和给定的时间间隔，把不同类型日志的产生频率作为多条行为序列进行记录。第二步，使用一个滑动窗口分别提取出每条行为序列的连续子序列，组成行为子序列的集合，并进行聚类等归类操作，对行为模式进行统计。最后，对日志流进行实时监控，截取子序列，利用训练得到的行为模式集计算子序列的异常指数，并对检测结果进行判断评估。

4.2.1 日志流到行为序列的转换

通过之前的日志预处理操作，我们使用层次聚类算法把规范化的日志归类为不同类型。接下来，我们要根据这些类型把训练数据集划分为不同类型的行为序列。

第一步，为每个日志类型赋一个类型号。

以 Hadoop 实验数据集为例，在上一步聚类中，正规化后的 1541 条日志数据被归为61个簇，每个簇代表一个类型。我们用0到60的序号为每个簇赋一个cluster ID，每一个 cluster ID 代表一个类。

第二步，构造一个字典结构，把正规化的日志数据映射到对应的 cluster ID。

例如正规化后的 Hadoop 日志包含 1541 条数据，生成字典就包含 1541 个键值对，键是日志文本，值是日志对应的类。构造这个字典的目的是为了方便后续的查找工作。

第三步，把训练日志序列转化为由 cluster ID 组成的序列。

以训练集为输入，依次读入每条日志记录、对日志记录进行规范化、把规范化的日志映射到对应的类，最后输出类型对应的 cluster ID。其中的映射工作可以借助上一步生成的字典进行快速查找。可以通过构造字典树（Trie tree），加快字符串查找速度，进一步提高查找效率。整个转化过程如图 4-3 表示：

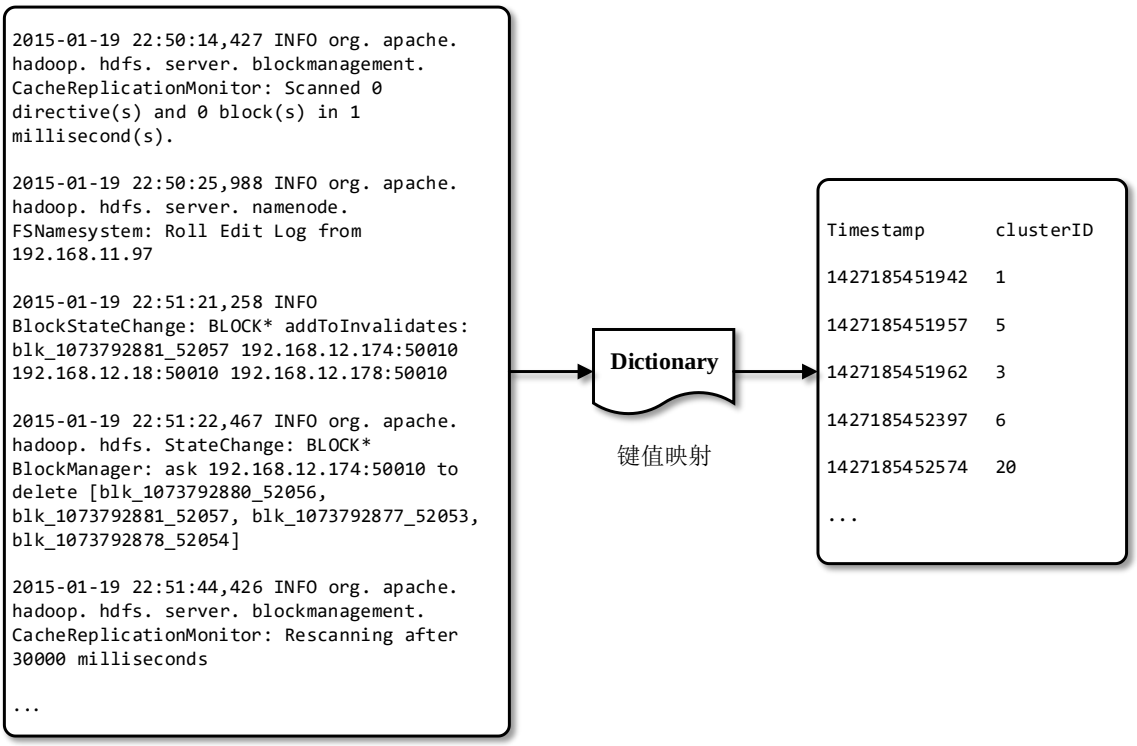


图4-3 日志流到 cluster ID 序列的转换

Figure 4-3 Transformation from log stream to sequence of cluster IDs

最终的结果序列包含了一个日志时间戳和对应的 cluster ID，加上时间戳是为了提取频率序列做准备。

第四步，把各个类型的日志序列转化为频率序列。

在这之前需要定义一个单位时间间隔，把单位时间内产生的日志数据作为此时的日志流频率。为达到较好的统计效果需要对不同长度的时间间隔进行测试，例如在 Hadoop 数据集的试验中，我们分别采用了 30 秒、1 分钟、1.5 分钟等长度做了实

验，最终决定使用 1 分钟作为时间间隔。

统计过程如图 4-4 所示。首先顺序读入由时间戳和 cluster ID 组成的序列，按定义的时间间隔统计单位时间内各个 ID 出现的次数并分别存入代表各个日志类型频率序列的文件中。

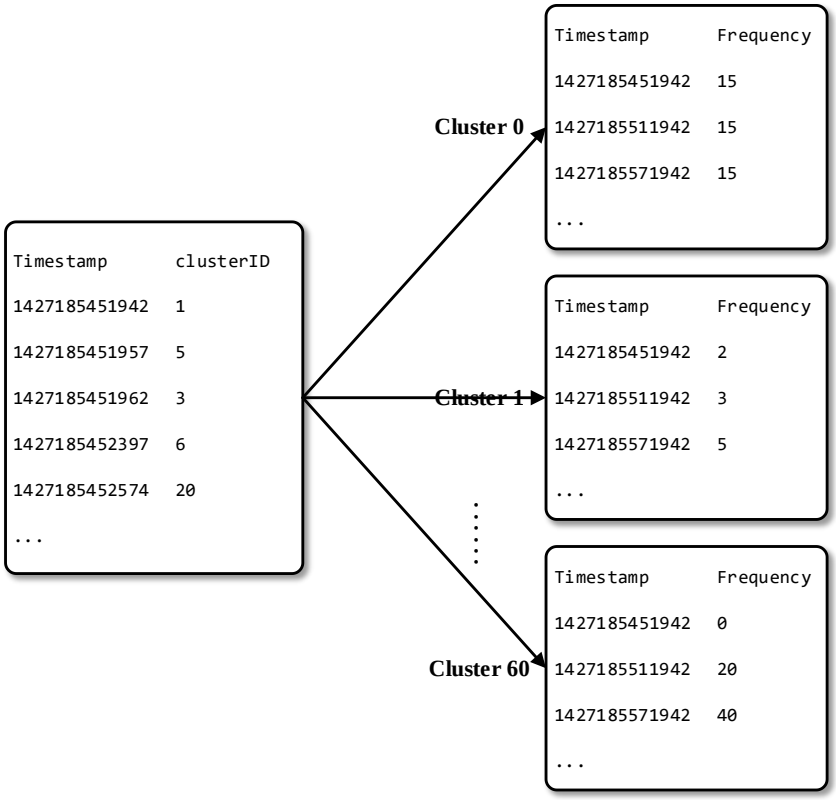


图4-4 频率序列统计流程

Figure 4-4 Generation of frequency sequence

最终得到的是每个日志类型的行为序列文件，包含时间戳及该时刻日志的频率。主要统计过程可以用下面的伪代码表示：

```
BufferedReader br = new BufferedReader(new FileReader(input)); // 读数据流
FileWriter fw = new FileWriter(new File(output)); // 写数据流

final long INTERVAL = 1000 * 60 * 1; // 时间间隔
long timeMark = getTime() + INTERVAL; // 初始化时间戳
String msg = null; // 日志记录
int[] frequency = new int[numberOfCluster]; // 频率计数器

while ((msg = br.readLine()) != null) {
```

```
    long timestamp = getTime(msg); // 获得事件戳
    int clusterID = getID(msg); // 获得 clusterID
    ++frequency[clusterID];
    while (timestamp > timeMark) { // 每单位时间间隔进行一次统计
        storeFrequency(frequency, fw);
        reset(frequency);
        timeMark += INTERVAL;
    }
}

br.close();
fw.close();
```

图 4-5 是对 Hadoop 数据集进行实验得到的频率数据样本，显示的是 cluster ID 为 10 的日志频率序列。图中我们看到大部分时间内频率曲线都比较稳定，但位于时间轴 1000 和 3000 左右的位置有两个刺突，其频率远高于其余部分，表明在这两个时刻软件系统很可能发生了异常。

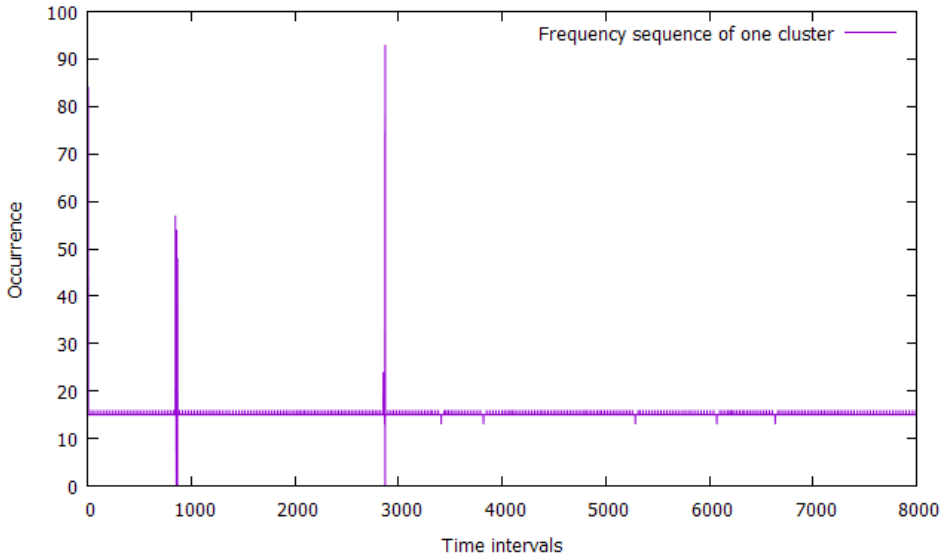


图4-5 Hadoop 数据集频率序列样本

Figure 4-5 Sample of frequency sequence of Hadoop data set

在对统计结果的观察中我们发现某些序列包含的数据过少，以至于无法从中发掘任何有用模式，我们的策略是删除这部分序列信息。还有一些序列有着极为一致的频率特征，包括相位和振幅。对这些序列我们采取合并措施，以减少计算复杂度。经过这些操作后，行为序列的数目进一步降低，例如在 Hadoop 数据集中频率序列就

被减少到了 42 个，提高了后续处理效率。

4.2.2 生成行为模式

在 4.1 节我们讨论了异常连续子序列问题，即时间序列不和（time series discords）。时间序列不和是极为有效的异常检测器，借助时间序列不和理论，只需考虑序列本身的形状等模式特征便能检测整条序列中的细微异常。基于这种思想，我们提出了基于频率序列的日志检测方法，通过转换模型把实时日志流转化成实时频率子序列，然后计算实时子序列与由日志训练集生成的行为序列集的相似度，据此进行日志异常判断。

通过上一步的计算我们得到了由不同日志类型组成的频率序列集，即行为序列。令频率时间序列集为$T_i, i \in [1, N]$，N 为序列的总数，即日志类型的数目。这一步将对每一个频率序列进行提取行为模式的操作，通过滑动窗口技术对序列$T_i = \{t_{i_1}, t_{i_2} \dots t_{i_m}\}$（$m$ 为序列长度）进行遍历，提取出所有的频率子序列并生成行为模式。

首先，定义长度为 k 的滑动窗口，提取长度 m 的时间序列T_i中所有的频率子序列。我们称所有的频率子序列s_{i_j}为行为子序列：

$$s_{i_j} = \left\{t_{i_j}, t_{i_{j+1}} \dots t_{i_{j+k-1}}\right\}, j \in [1, m-k+1] \tag{4-3}$$

每个长度为 m 的频率序列T_i包含 $m\text{-}k\text{+}1$ 个行为子序列，相邻子序列有长为 $k\text{-}1$ 的部分相互重叠。现在我们得到了 N 个行为序列集S_i, $i \in [1, N]$，每个行为序列集表示为

$$S_i = \{s_{i_1}, s_{i_2} \dots s_{i_{m-k+1}}\} \tag{4-4}$$

图 4-6 是为一个行为序列生成子序列的示例。其中滑动窗口的大小$k = 6$，通过窗口的顺序滑动，把包含在其中的子序列依次记录在另一个文件中，每行代表一个行为子序列。

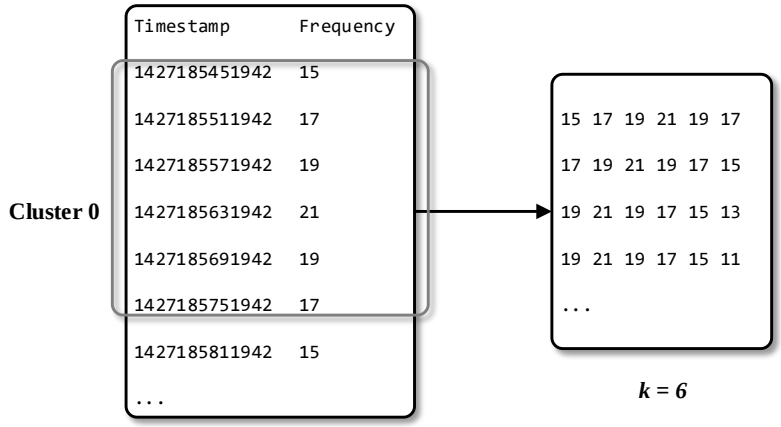


图4-6 行为子序列的生成示例

Figure 4-6 Sample of generation of behavior subsequence

接着需要为行为子序列定义一个相似性度量标准，并对相似行为子序列进行统计操作，从行为序列中提取行为模式。

与定义日志文本的相似度不同，子序列由整数数值组成，不同子序列可以看作相同长度的数值向量。而向量间的距离或差可以有多种度量方式，如曼哈顿距离、欧几里得距离、切比雪夫距离等等。我们取常用的欧氏距离作为子序列的距离度量标准。令$s_{i_a} = \{t_{i_a}, t_{i_{a+1}} \dots t_{i_{a+k-1}}\}$, $s_{i_b} = \{t_{i_b}, t_{i_{b+1}} \dots t_{i_{b+k-1}}\}$为两条长为 k 的子序列，两者间的距离定义为

$$Dist(s_{i_a}, s_{i_b}) = \sqrt{\sum_{n=0}^{k-1}(t_{i_{a+n}} - t_{i_{b+n}})^2} \tag{4-5}$$

定义了距离标准后，我们对相同及相似的子序列数目进行统计，把不同类型子序列的形状特征及出现频率作为该类型序列的行为模式。对相似子序列的归类操作可以使用日志预处理中的层次聚类方法。由于之前所做的预处理操作使得行为子序列的周期性和波形都相当规律，因此可以采取更简单的方式，如定义一个相似度阈值进行简单聚类，或者把子序列向量作为键值，统计其出现次数。

我们把由子序列集S_i得到的行为模式集记为P_i：

$$P_i = \{p_{i_1}, p_{i_2} \dots p_{i_x}\} \tag{4-6}$$

x 为聚类数目。统计过程可以由图 4-7 表示：

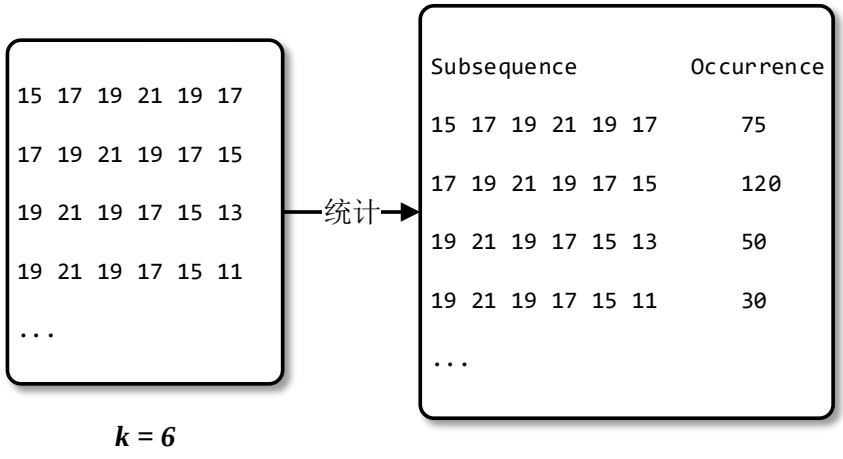


图4-7 生成行为模式

Figure 4-7 Generating behavior patterns

最后，为行为模式集中的每一个行为模式定义一个异常值。我们很自然地使用每个行为子序列的出现频率作为行为模式参数，出现次数少的行为模式必然具有更高的异常值。因此我们把子序列出现频率的倒数作为行为模式p_{i_j}的异常值，记作

$$\frac{1}{Occur(p_{i_j})} \tag{4-7}$$

图 4-8 展示了对 Hadoop 数据集进行实验时得到的行为子序列样本，两条子序列都属于 cluster ID 为 10 的行为序列。

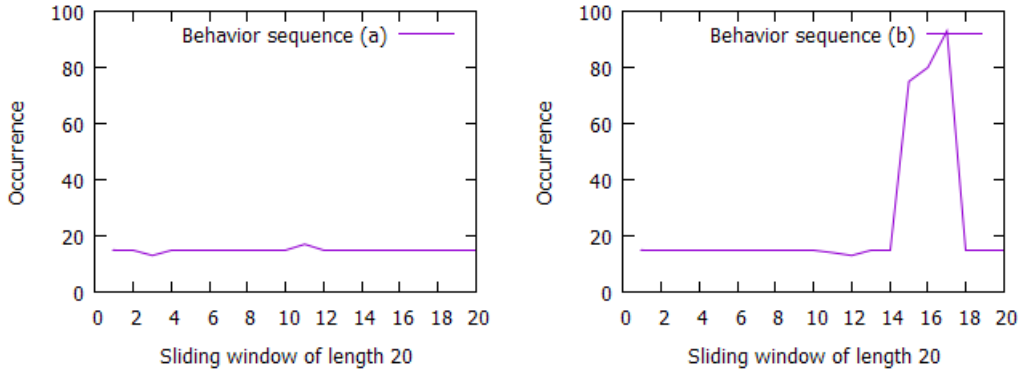


图4-8 行为模式样本

Figure 4-8 Sample of behavior patterns

其中序列（a）的出现次数为 290 次，而序列（b）仅出现了两次，这表明序列（b）代表的行为模式有更高的异常值。通过对实验数据的观察，我们的推论也得到了证实，在序列（b）出现的时间段中软件系统确实出现了崩溃异常。

4.2.3 计算异常指数

这是异常检测过程的最后一步，通过计算实时日志流的异常指数对系统状态进行判断。经过之前对训练数据集的预处理和模型训练，我们得到了描述日志特征的行为模式集$\{P_1, P_2, \dots, P_N\}$，代表 N 个类型日志的行为模式，接下来我们用这个模式集作为参数计算日志序列的异常值。

首先根据预先定义的单位时间间隔和长度 k 的滑动时间窗口，实时截取最近 k 个单位时间的日志序列 L，并根据不同日志类型把日志序列拆分成 N 个日志子序列，记作

$$L = \{l_1, l_2, \dots, l_N\} \tag{4-8}$$

接着使用本节提出的转换算法，把当前日志序列转换成包含 N 个元素的频率序列集 C，C 表示为

$$C = \{c_1, c_2 \dots c_N\} \tag{4-9}$$

现在对异常指数如何计算作定义。当前序列集 C 包含 N 个行为序列，每个行为序列c_i对应一个行为模式集P_i，P_i由$p_{i_1}, p_{i_2} \dots p_{i_x}$等 x 个行为模式组成，其中与c_i相似度最高的行为模式记为p_{i_j}，日志子序列l_i的异常指数由c_i和p_{i_j}共同决定。我们定义l_i的异常指数等于c_i间p_{i_j}的相异度同行为模式p_{i_j}本身的异常值之和，记为

$$AnomalyScore(l_i) = Dist\left(c_i, p_{i_j}\right) + \frac{\beta}{Occur(p_{i_j})} \tag{4-10}$$

而日志序列 L 的异常指数定义为所有子序列异常指数的和，即

$$\sum_{i=1}^{N} AnomalyScore(l_i) \tag{4-11}$$

最后的异常指数计算公式记为

$$AnomalyScore(L) = \sum_{i=1}^{N} \min_{j}(Dist(c_i, p_{i_j})) + \frac{\beta}{Occur(p_{i_j})} \tag{4-12}$$

当c_i与距其最近的行为模式p_{i_j}间距离较大，即$Dist(c_i, p_{i_j})$值较大时，表明日志子序列l_i与任何行为模式都不相似，因此其出现异常的可能性较大。l_i的异常性还与p_{i_j}本身的异常性有关，因为最邻近行为模式的特性很大程度代表了序列的特性。β是平衡因子，可以在实验过程中进行调试以得最佳结果。

4.3 实验结果对比与分析

本文之前对 Hadoop 事件日志和 LANL 数据集的正规化实验做了介绍，本节将对两组实验数据最终的异常检测结果进行对比与评估。实验数据分为训练集和测试集两部分，其中训练集占 80%，剩余的 20%为测试数据。

4.3.1 实验评估标准

首先定义成功的检测：一次成功的异常检测是指成功预测到 5 分钟内发生的系统异常事件。其次是准确率（accuracy）的度量标准，由精确率（precision）、召回率（recall）和 F1 值（F-score/F-measure）组成。在统计学习的分类预测领域中，数据的分类或预测结果通常由 true positives（TP）、true negatives（TN）、false positives（FP）及 false negatives（FN）组成，具体评估意义如表 4-1 所示：

表4-1 分类评估标准

Table 4-1 Evaluation criteria of classification result

		Actual class (observation)	
		Actual positive	**Actual negative**
Predicted class (expectation)	**Predicted positive**	True positive (TP)	False positive (FP)
	Predicted negative	False negative (FN)	True negative (TN)

- 精确率：$Precision = \frac{TP}{TP+FP}$;
- 召回率：$Recall = \frac{TP}{TP+FN}$;
- F1 值：$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$，F1 值同时考虑了精确率和召回率两个因素。

4.3.2 结果对比与分析

首先通过训练集训练数据模型、得到行为模式。4.2 节曾经讨论过滑动窗口大小 k 取值的重要性，不同的 k 值会导致不同的准确率和计算复杂度，可能还会产生过度拟合或拟合不足等后果。通过多次实验，我们选择把$k = 20$作为滑动窗口大小。经过这步后，Hadoop 日志数据产生了 42 个行为模式集、LANL 数据集产生了 72 个行为模式集，每个行为模式集对应一类日志序列。

然后对测试数据集进行异常值计算并据此预测系统异常和错误。由于要根据异常指数的大小判断是否产生异常，需要定义一个异常指数阈值 y，异常数值高于 y 时即预测系统即将出现错误。阈值的大小会影响最终预测的精确率（precision）和召回率（recall），当 y 值取得很大时，虽然能提高精确率，但可能会漏掉一些异常即 false negative，导致召回率降低；反之则可能出现误判，导致召回率升高而精确率降低。阈值大小需要根据使用者的要求进行合适调整。

为了更好地验证算法性能，我们引入了同样基于频率分析的传统检测方法，用来进行横向比对。传统日志频率分析方法把分析重点放在日志流单一时间间隔的频率特征上面，通过主成分分析（Principal Component Analysis，PCA）和期望、方差等统计规律分析对离群点进行检测，当这段时间的日志流频率属于离群点或小概率事

件时即判断出现了异常。

我们分别对两组日志数据集进行了基于频率的方法和基于行为异常的方法的模型训练及预测。为了使每个算法在每个数据集上都取得最好的效果，我们针对每次实验设置了不同的参数，如异常阈值、平衡因子等等。为了综合评价预测结果的精确率和召回率，我们使用 F1 值作为预测准确率的评价标准。

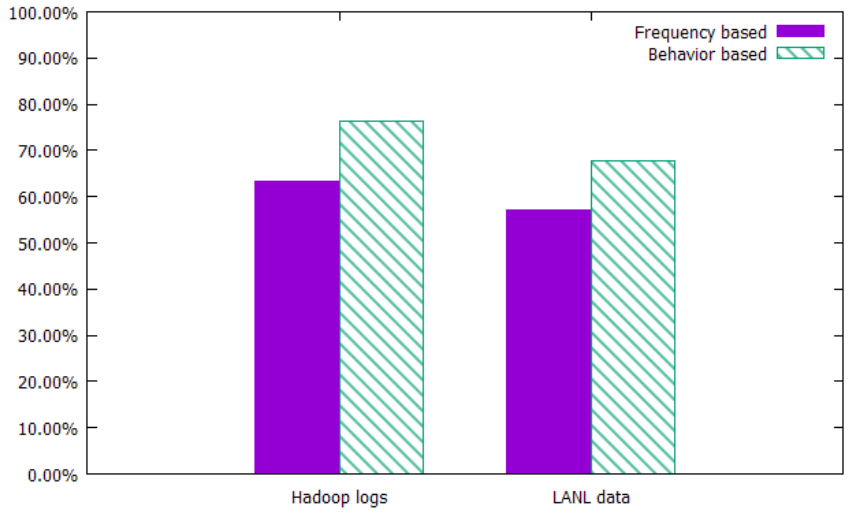
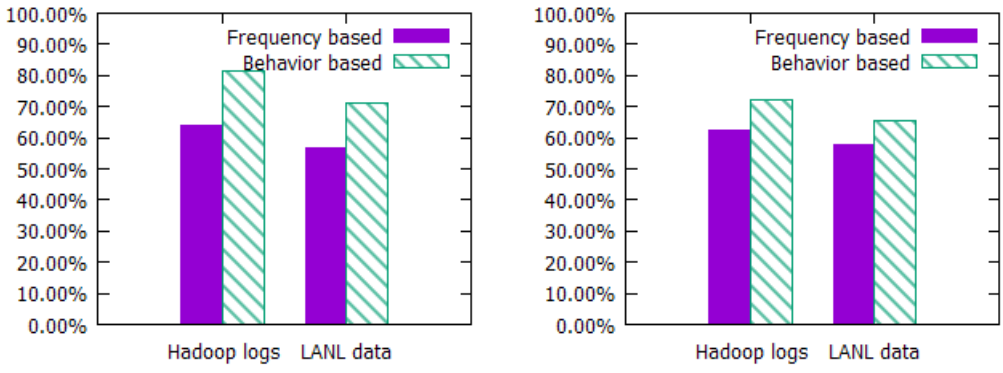


图4-9 预测准确率对比

Figure 4-9 Comparison of prediction accuracy

图 4-9 展示了两个数据集上共计四次实验的准确率对比，其中 frequency based 是传统频率检测方法，behavior based 是本文提出的行为异常检测方法。通过直方图可以看到我们的检测方法在检测准确率上有了明显的提高，其中 Hadoop 数据集测试的 F1 值达到了 76.4%，比传统方法提高了 13%，LANL 数据集测试的 F1 值也达到了 67.9%，提高了 10%左右。



Precision Recall

图4-10 预测精确率和召回率对比

Figure 4-10 Comparison of prediction precision and recall rate

图 4-10 分别展示了实验结果的准确率和召回率对比。从图中可以看到我们的检测方法在准确率的改善程度上要大于召回率，两个数据集皆是如此。而两种方法对 Hadoop 数据集的检测准确率都要高于 LANL 数据集，这是由于不同日志数据的数据特征不同。Hadoop 事件日志在有效载荷部分传达了更多信息，更易于提取类型特征，LANL 数据集中事件信息则是分布在不同的日志项中。

4.4 本章小结

本章首先描述了异常检测算法中引入的连续子序列理论，讨论了异常子序列问题和滑动窗口技术，简要介绍了常用的几种异常序列检测算法，把这些算法作为日志异常检测技术的理论基础。接着详细描述了基于行为模式的异常检测算法。算法把日志监控问题抽象为异常连续子序列的检测问题，包括三个步骤：把日志流转换为行为序列、在行为序列集中训练行为模式、通过行为模式计算实时日志流的异常指数。最后对实际数据集进行了仿真实验，并对算法检测效果做了对比和分析。通过与传统频率算法在两个实验数据集上的测试结果对比，证明了本文提出的日志检测方法明显提高了异常检测准确率，具有很好的效果。

第五章 基于行为异常检测的日志监控系统

本文在前几个章节对日志监控的基本理论和现有方案做了简单介绍，然后较为完整的阐述了自己提出的、包括日志数据预处理和日志异常检测两个步骤的、基于行为异常检测的日志监控技术。这一章将对日志监控方案的系统原型实现进行介绍。

5.1 需求分析

首先，为了实现一个完整的日志监控方案，所设计的日志监控系统需要包含如下功能：

- 日志采集：对客户系统各台服务器上生成的日志数据进行实时监控，将产生的日志流集中缓存并统一传输；
- 日志存储：把采集的日志数据集中保存在永久存储设备上，存储方式可以是文件或数据库表；
- 模型训练：根据前述理论对存储的日志数据进行训练，生成行为模式集，同时周期性对行为模式进行更新；
- 日志流分析：根据行为模式对日志流进行实时分析，计算异常值，通过与异常阈值的比较判断系统状态；
- 前端面板：包括展示分析结果、对异常进行告警、以及在前端对监控系统进行控制。

其次，监控系统还需满足一定的非功能性需求，包括：

- 性能需求：能及时并正确地对系统可能发生的异常做出准确预测，并且在不同负载情况下均能稳定运行；
- 可扩展：当被监控系统进行软、硬件方面的调整时，监控系统能很好地适应这种调整而不影响功能和性能；
- 可维护：监控系统应该是易于维护的，特别是在进行调整或升级时不能影响被监控系统的运行；
- 可用性：由于要对客户系统连续监控，因此要满足 7×24 不间断正常运行的要求；
- 易使用：监控系统应该是容易安装的，一般管理人员要能够理解系统原理和一般操作；

- 低消耗：监控系统应该保持高工作效率，在低功耗的一般硬件设施上就能稳定运行。

系统的基本网络结构如图 5-1 所示。图中日志存储服务器、日志分析服务器与被监控系统硬件设备处于同一个局域网，对日志结果进行本地存储和分析。前端展示和控制面板可以在同一个局域网，也可以通过连接互联网在外网进行查看和控制。大体流程为监控系统通过安装在 client 中的 log agent 对日志流进行监控和采集；采集得到的数据分两份进行传输：一份传入存储服务器进行永久存储，另一份传入日志分析服务器进行实时分析；日志分析服务器利用存储服务器中的日志数据进行模型训练，根据得到的模型对日志流进行异常检测；最后的检测结果通过前端平台向用户展示。

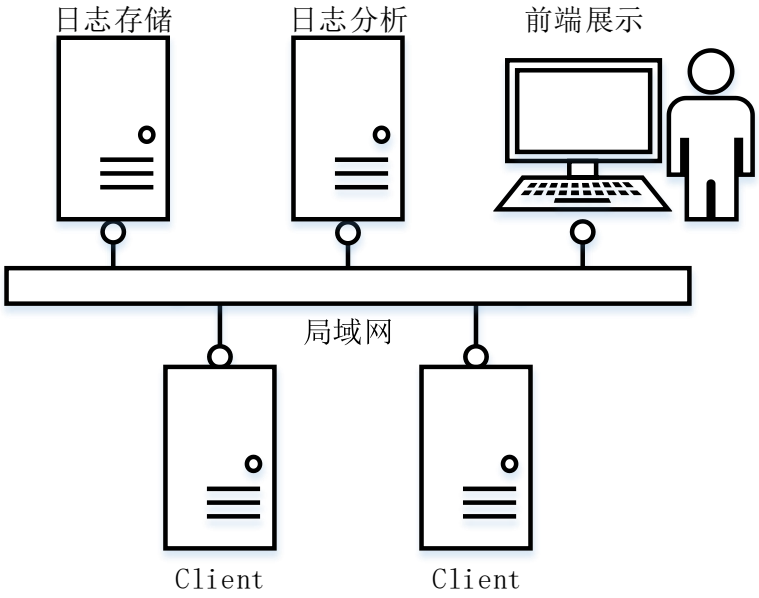


图5-1 系统基本网络结构

Figure 5-1 Basic system architecture

5.2 系统框架

根据前述功能需求和非功能需求分析，我们设计了日志监控系统的系统框架，其框架结构如图 5-2 所示：

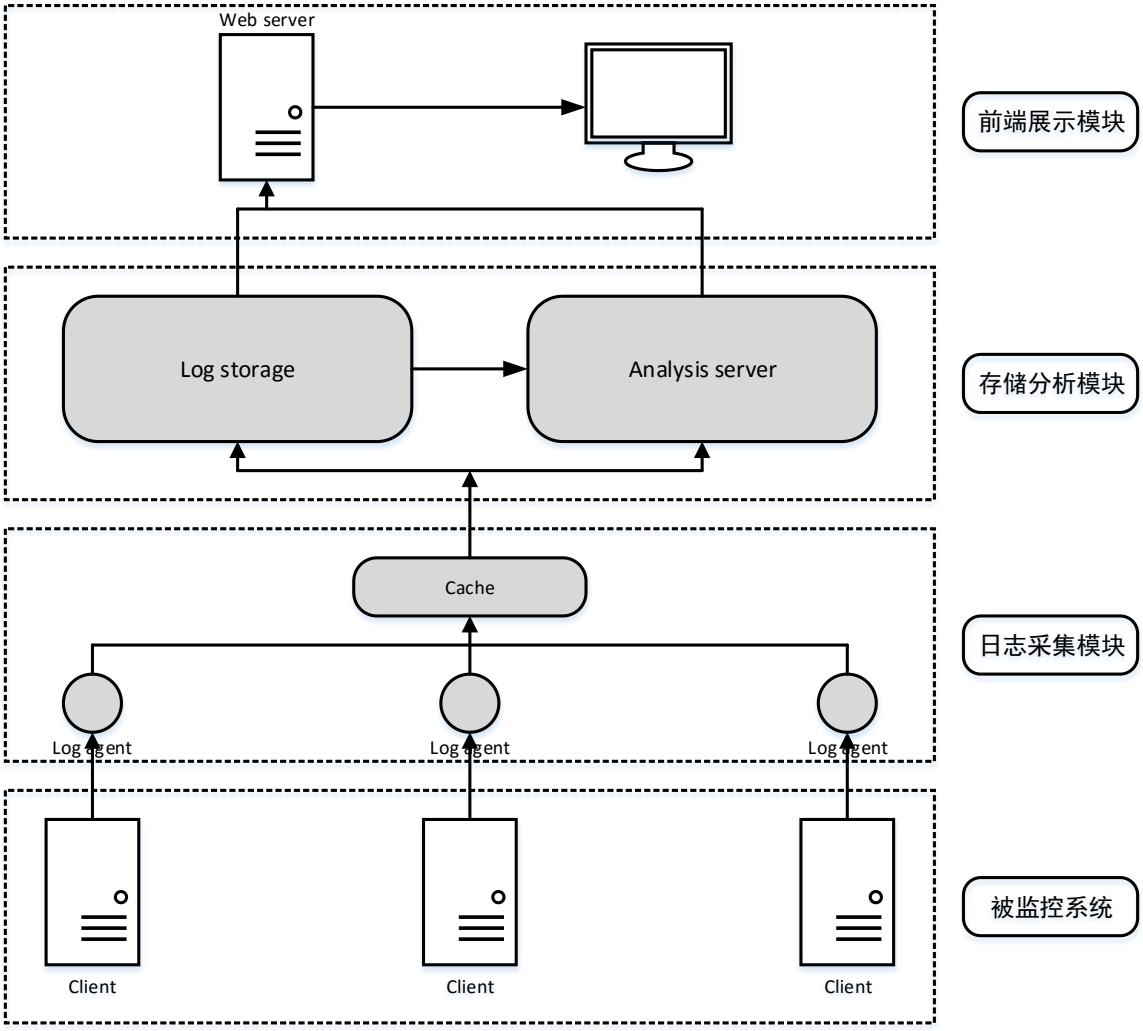


图5-2 系统框架结构

Figure 5-2 System architecture

连同受监控的系统在内，整个监控方案从下到上共分为四个部分，分别是被监控系统、日志采集模块、日志分析模块和前端展示模块。

- 被监控系统

被监控系统可以是任何大型分布式业务系统，包括电信管理系统、电力监控系统、医疗卫生系统、生产管理系统等等。其产生的日志可能是保存在本地的文件或通过网络传输的日志流。

- 日志采集模块

日志采集模块提供了对被监控系统产生的日志进行实时监控和收集的功能。日志采集模块的设计原则是在不干扰被监控系统运行的条件下收集日志，被监控系统甚

至感觉不到日志采集模块的存在。我们的方法是在每台应用服务器上安装独立的监控探针（log agent），把产生的日志数据传输到中央缓存。

- 存储分析模块

此部分是整个监控系统的核心，对异常检测算法进行了实现。主要包括存储和分析两个部分，其中存储部分通过文件或数据库表的方式对收集到的日志数据进行永久存储，分析部分对存储模块中的数据进行模型训练，生成行为模式，并根据行为模式对实时日志流进行分析，生成检测结果。

- 前端展示模块

前端展示模块用于查看日志信息，展示系统状态的检测结果，对异常进行告警，同时还控制监控系统的开启、关闭和设置等等。存储分析模块的检测结果以 xml 或 JSON 等协议数据格式，通过 TCP socket 或 HTTP 报文等方式发送到 web 服务器，以 web 界面的方式展示给管理人员或用户。

整体系统结构中数据的流动和转换如图 5-3 所示：

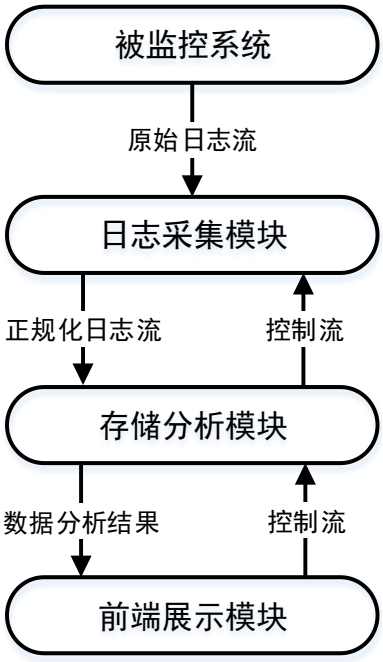


图5-3 数据流动和转换

Figure 5-3 Data streaming and transformation

接下来依次对日志监控系统的日志采集模块、存储分析模块和前端展示模块进行介绍。

5.3 日志采集模块

本文第二章介绍过两类日志收集方式，分别是基于客户系统主动推送的 push 方式和基于监控系统主动拉取的 pull 方式。其中基于 push 的方式主要为客户系统内部使用，通过 syslog 或 SNMP 等协议，把生成的日志数据主动发送到网络端口，接收方对端口进行监听，把接收到的数据进行存储或进一步分析。第三方的日志管理系统则主要使用基于 pull 的方式，通过在客户系统服务器上安装监控代理，对日志数据变化进行监控，把新产生的数据通过网络协议发送到日志服务器进行缓存和存储。我们的监控系统同样使用基于 pull 的收集方式，其中监控代理使用了开源日志收集系统 Logstash 的代理组件。

之前我们曾对 Logstash 进行过简要介绍。作为优秀的开源日志管理方案，Logstash 常与 ElasticSearch 和 Kibana 一起，组成完整的日志收集、查询和展示系统。整体架构如图 5-4 所示：

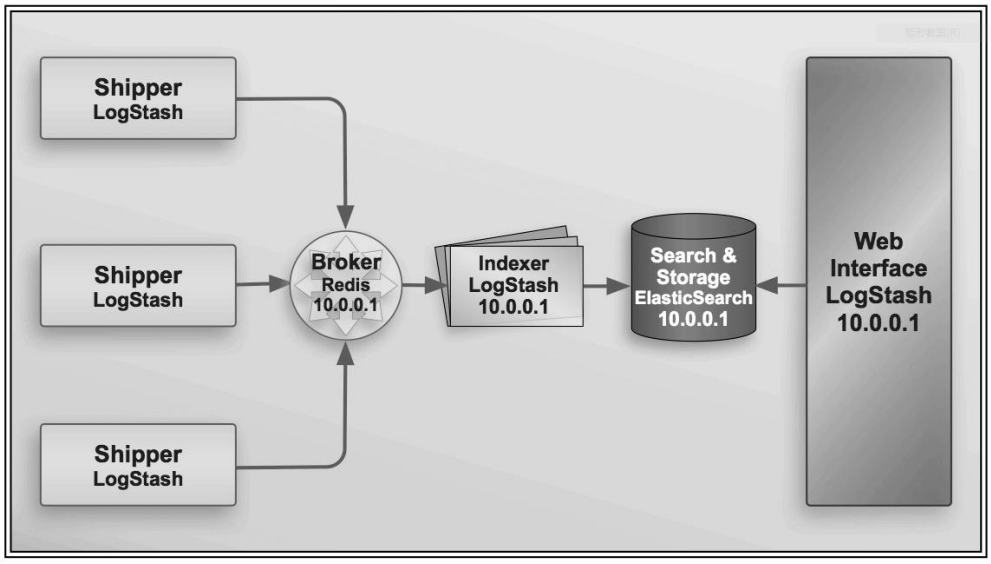


图5-4 Logstash 整体架构

Figure 5-4 Architecture of Logstash

其中 Shipper Logstash 作为监控代理安装在每台应用服务器上，对日志目录或日志流进行监控。收集的日志通过 Redis 数据库汇总并缓存。接着日志数据被发送到日志存储服务器上，这台服务器通过另一个 Logstash 对日志数据进行接收和存储，经由 ElasticSearch 进行索引和提供查询功能，最后由 Kibana 提供的 web 前端对进行展示，并向用户提供操作接口。

选择 Logstash 作为日志监控代理主要基于如下原因：

- Logstash 开源。

Logstash 的开源特性使它能够集成在自己开发的日志处理方案中，并且可以进行定制和修改。其社区支持也使其越发成熟，并且具有强大的生命力。

- 可扩展性。

当客户系统进行调整或修改，例如增加应用服务器时，可以动态地添加监控代理，只需简单配置就能同时完成监控系统的扩展。

- 灵活性。

Logstash 拥有众多插件，通过配置文件的简单设置，就能利用不同插件对几乎任何格式的日志进行收集，并且以用户希望的方式进行数据传输。

Logstash 根据配置文件确定监控位置和传输方式，并能根据正则表达式对日志进行初步过滤。配置文件的简单示例如下：

```
input {
  file {
    path => "/etc/httpd/*/*log"
  }
}

output {
  #stdout {}
  #stdout { codec => json }
  tcp {
    host => "192.168.1.101"
    port => "8000"
    codec => line
  }
}
```

图5-5 Logstash 配置文件示例

Figure 5-5 Sample of Logstash configuration file

其中监控位置为通配符表示的目录，传输方式为 TCP socket 传输，同时可以为传输数据指定格式、进行过滤等等。

最终的日志采集模块结构定义如图 5-6 所示。其中的监控代理由 Logstash 及其配置文件组成。Logstash 对日志源进行周期性扫描，检测到变化后触发日志传输事件。通过 Logstash 把采集的日志汇集到一个 Redis 中央缓存系统进行临时存储。缓存中的

数据最后通过通信协议传入存储分析模块。

实验证明日志采集模块对日志的收集及时有效（触发时延在 0.1s 左右），并且由于独立于被监控系统，对被监控系统的性能几乎没有影响。

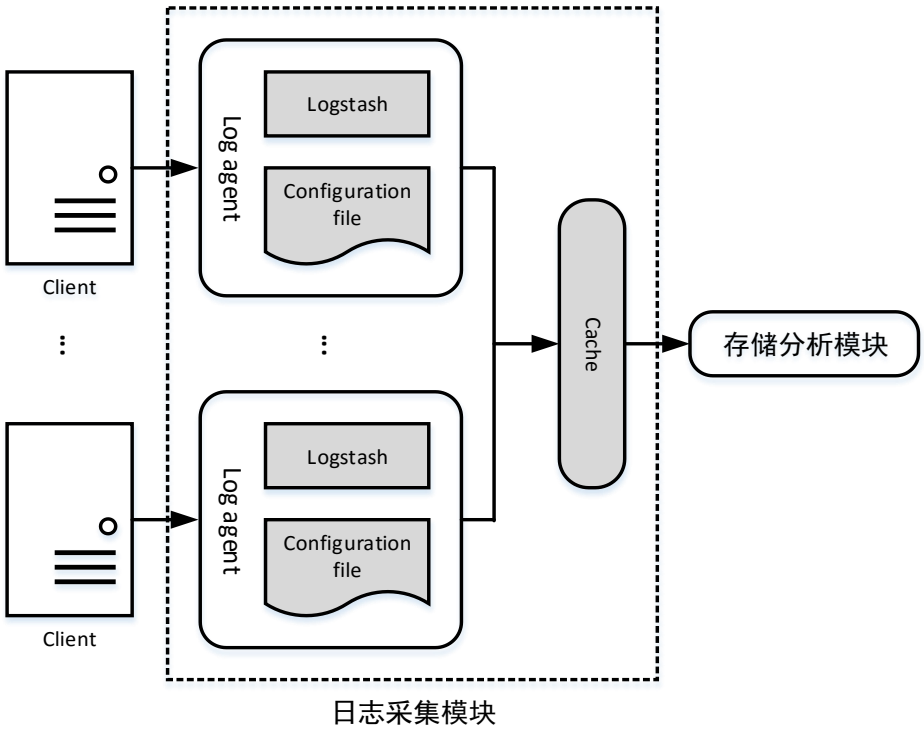


图5-6 日志采集模块

Figure 5-6 Log collection module

5.4 存储分析模块

存储分析模块是整个日志监控系统的核心部分，日志数据的存储、转换和分析都在这部分完成。按照功能划分可分为存储模块和分析模块两部分，存储模块主要接收来自日志采集模块的日志流，分析模块除了接收实时数据流外，还把存储模块的日志内容作为输入，用以训练和更新异常检测模型。

5.4.1 日志存储模块

日志存储模块主要负责日志数据的存储，功能十分简单却很重要。存储的日志要作为日志分析的训练集，因此要满足方便查看和快速存取的要求。日志存储模块通常包括一个或几个存储服务器，对存储日志的文件系统进行维护。

本文在第二章对日志存储问题进行过讨论，包括日志存储格式和存储位置。存

储格式包括普通文本格式、二进制格式和压缩格式，为了方便查看和保持存取过程的效率，我们选择把日志存储为普通文本格式；而存储位置包括普通文件系统、数据库和 Hadoop HDFS，考虑到对原始日志数据的操作只限于顺序读取、并且当前的数据规模不是很大，我们选择把日志数据以普通文件的方式存储在一般文件系统中。

对于文件大小和文件存储周期，我们选择以 24 小时为一个周期，即每天新增一个日志文件，顺序存储当天发生的所有日志，同时以“年-月-日”的方式为文件命名。

5.4.2 日志分析模块

日志分析模块实现了本文提出的日志异常检测方法，包括日志预处理、检测模型训练和最后的异常检测。从图 5-2 中可以看到，日志分析模块接收两个不同的数据输入，分别来自日志采集模块和日志存储模块。来自日志采集模块的数据流用于进行实时异常检测，并把检测结果送到前端展示模块；来自日志存储模块的是永久存储的日志数据，用于对异常检测模型进行训练。由于存储的日志数据不断变化，因此还需要对检测模型进行周期性更新。

下面按照日志处理流程对日志分析模块的功能进行逐项介绍。

1. 行为模式的训练和更新

这部分承担了整个日志分析工作的主要部分，通过对存储模块的数据进行训练，生成日志检测所依赖的行为模式。然后由于日志是源源不断生成的，存储模块的内容不断变化，因此模式训练工作也需要周期性地重复进行，不断对行为模式进行更新，以保持对日志特征的准确描述。更新周期可能是一天、一周，或者由用户自己制定。每次训练所花的时间也随数据规模变化，从几分钟到几小时不等。训练过程对 CPU 压力较大，因此建议在空闲时间训练或使用单独的服务器进行训练。

行为模式的训练或更新流程如图 5-7 所示。首先以行为单位，顺序读取日志存储模块的日志记录；接着对日志进行正规化操作，包括合并多余的行、去处冗余字符、日志级别转换和日志内容去参数化；然后调用凝聚方式的层次聚类算法，按合适的粒度将日志归类为不同类型；接下来依次调用模型训练算法，把日志数据转换为不同类型的行为序列，为不同行为子序列生成行为模式集，最终得到数据的行为模式。当上次训练时间超过更新周期长度或收到更新命令时，重新执行整个训练流程。

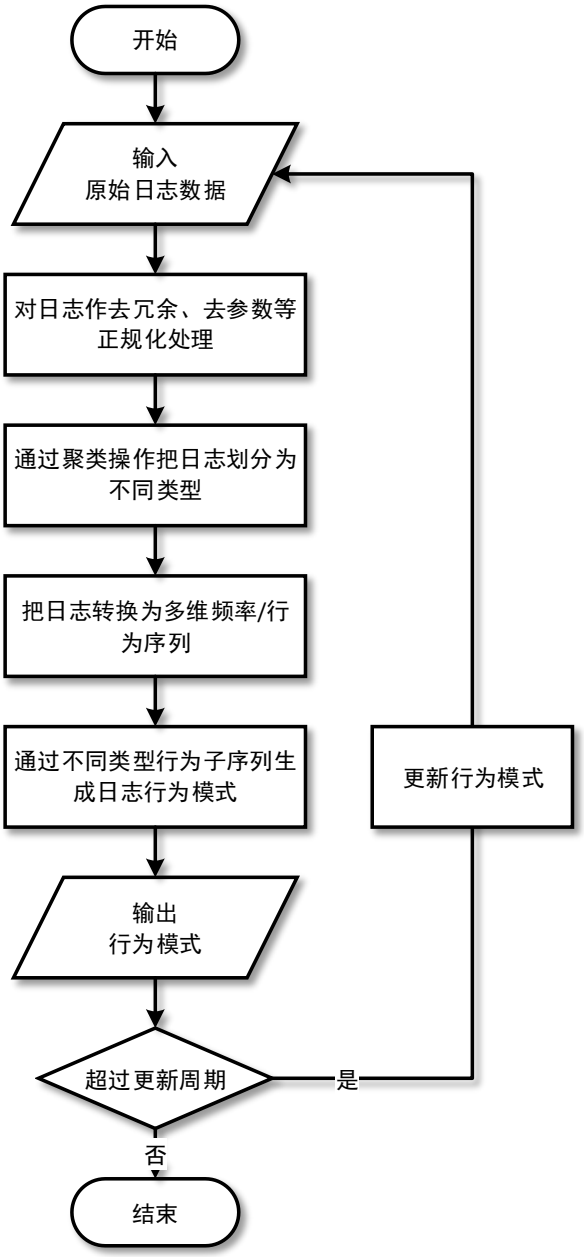


图5-7 模式训练流程图

Figure 5-7 Flow chart of pattern training

训练过程中用到的主要几个类如图 5-8 所示。其中 PayloadPreprocessor 对日志进行正规化处理，ClusteringAlgorithm 实现主要层次聚类算法，LogClustering 对日志进行聚类操作，DataGenerator 把日志转化为行为序列。

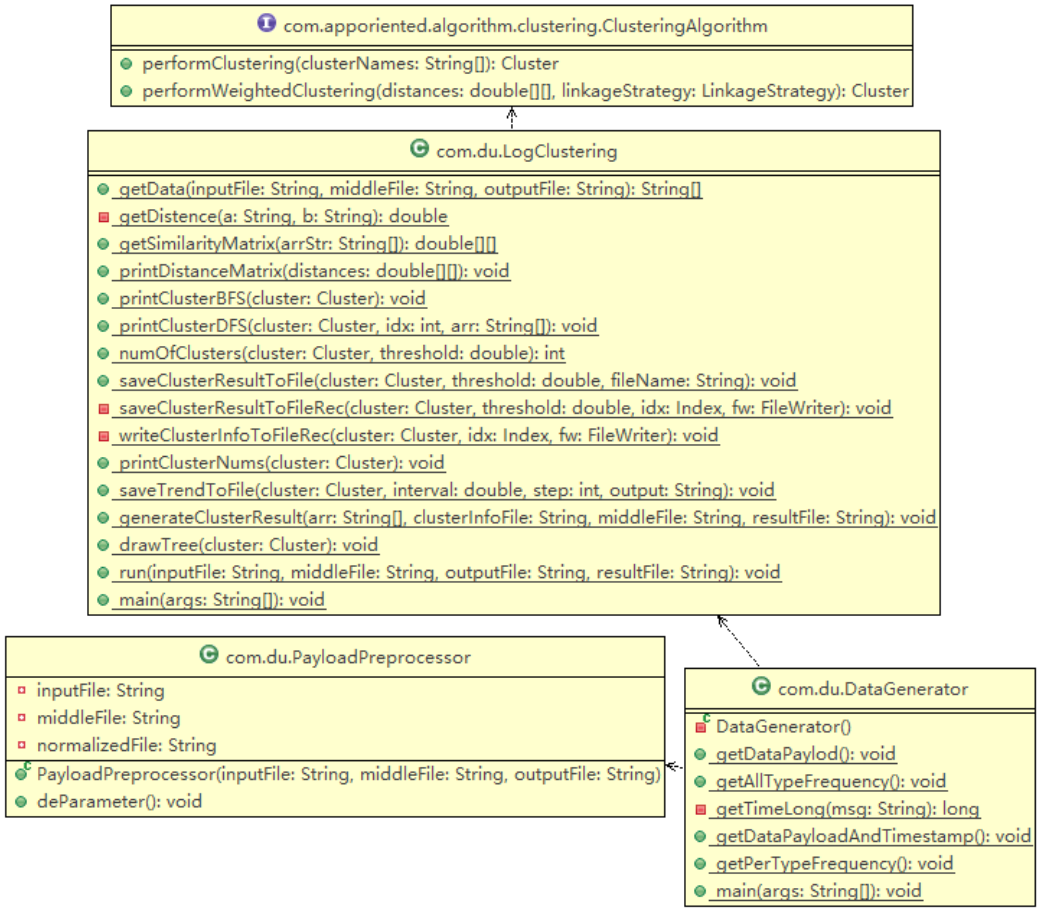


图5-8 训练流程主要类图

Figure 5-8 Class diagram of pattern training

2. 实时日志流的异常检测

这部分负责对实时日志流进行异常行为检测，同时向外输出分析结果。整个分析流程如图 5-9 所示。首先按照预定义时间间隔和时间窗口对日志流分片，根据聚类结果把日志转化为不同类型的频率子序列；然后把日志流的行为序列和日志行为模式作为参数，根据异常检测计算公式计算日志异常值。其间如果行为模式到达更新时间，要执行行为模式的更新操作；接着比较异常值和异常阈值的大小，若大于阈值则创建异常告警；最后把分析结果传输到前端平台进行展示。在第四章我们曾对检测效果做了实验对比，结果显示我们的方法在检测准确率上有了明显提高（超过10%）。

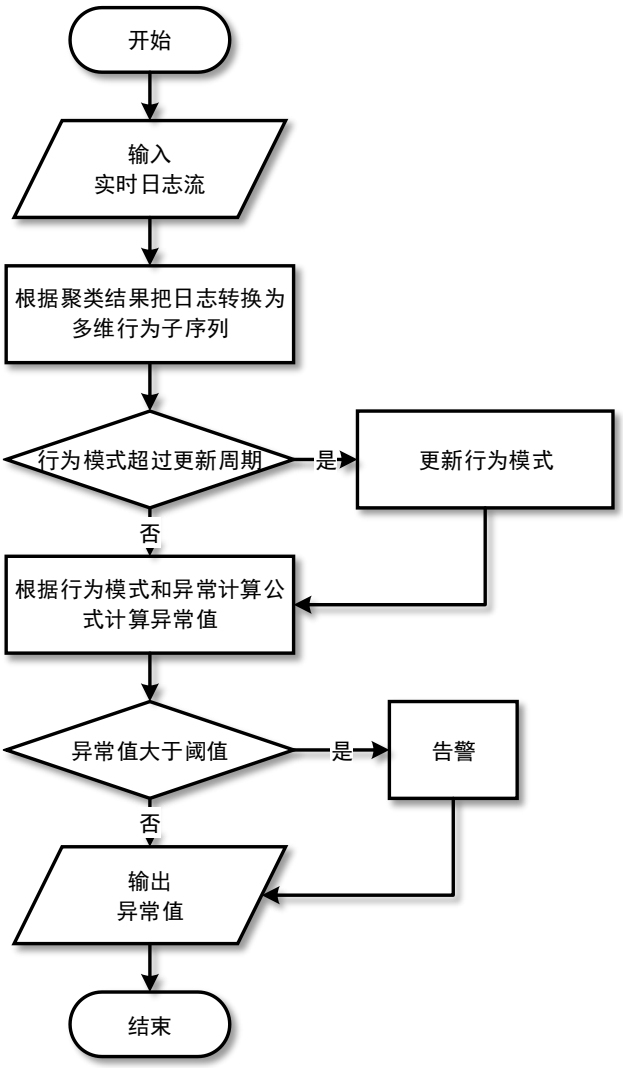


图5-9 异常检测流程图

Figure 5-9 Flow chart of anomaly detection

主要类间关系如图 5-10 所示：

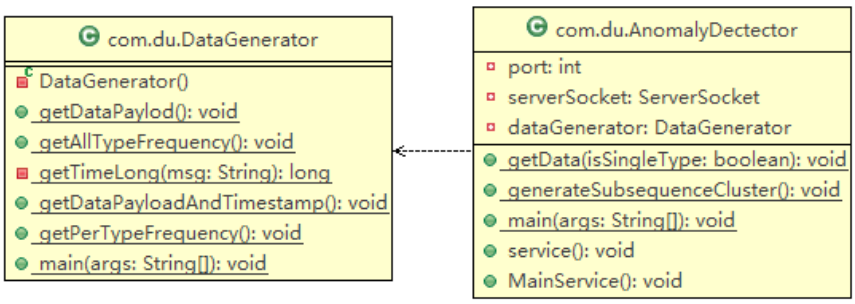


图5-10 异常检测类图

Figure 5-10 Class diagram of anomaly detection

5.5 前端展示模块

整个日志监控系统基于前后端分离的方式进行开发。前后端分离的思想是从物理层区分，把客户端作为前端，服务器端作为后端。具体来说，前端只包含 View 展示层和少许 Controller 层，后端负责 Model 层和业务数据的处理等。在我们的系统框架中，后端完成了绝大部分的数据处理和分析工作，前端只负责最终结果的展示和一些控制功能。存储分析模块完成对日志数据的分析工作后，得到的分析结果传入前端展示模块进行展示。前后端通过基于 HTTP 协议的 Web API 进行数据交互。

前端展示模块的功能及对应 API 示例如表 5-1 所示，前端通过不同 HTTP 请求执行不同功能，后端在收到请求后对 URL 进行解析并执行不同的操作，如运行开启或关闭系统的脚本、返回当前系统状态等，并把执行结果通过 HTTP 回复传回前端。前端收到 HTTP 报文后，解析 JSON payload 的内容，得到所要的数据或命令的执行结果后向终端用户进行展示。

表5-1 前端模块功能

Table 5-1 Functions of front-end module

功能	HTTP Request	HTTP Response

获取当前系统状态	GET
	http://serverhost/DetectionSystem/SystemStatus
		HTTP/1.1 200 OK
		Content-Type: application/json
		{
		"AnomalyScore": "30%",
		"IsAnomalous": "false"
		}

获取当前存储信息	GET
	http://serverhost/DetectionSystem/LogStatus
		HTTP/1.1 200 OK
		Content-Type: application/json
		{
		"TotalSize": "100G",
		"FileNumber": "2000"
		}

启动监控	GET
	http://serverhost/DetectionSystem/StartMonitoring
		HTTP/1.1 200 OK
		Content-Type: application/json
		{
		"Status": "true",
		}

停止监控	GET
	http://serverhost/DetectionSystem/StopMonitoring
		HTTP/1.1 200 OK
		Content-Type: application/json
		{
		"Status": "true",
		}

图 5-11 是前端展示的流程示例。前端周期性发送请求到后台，希望获取当前异常状态；后台收到请求后返回状态响应；前端对响应报文进行解析，展示状态图并根据结果决定是否进行异常告警。状态图中横坐标为时间轴，纵坐标为异常值，根据时间进行周期性更新。

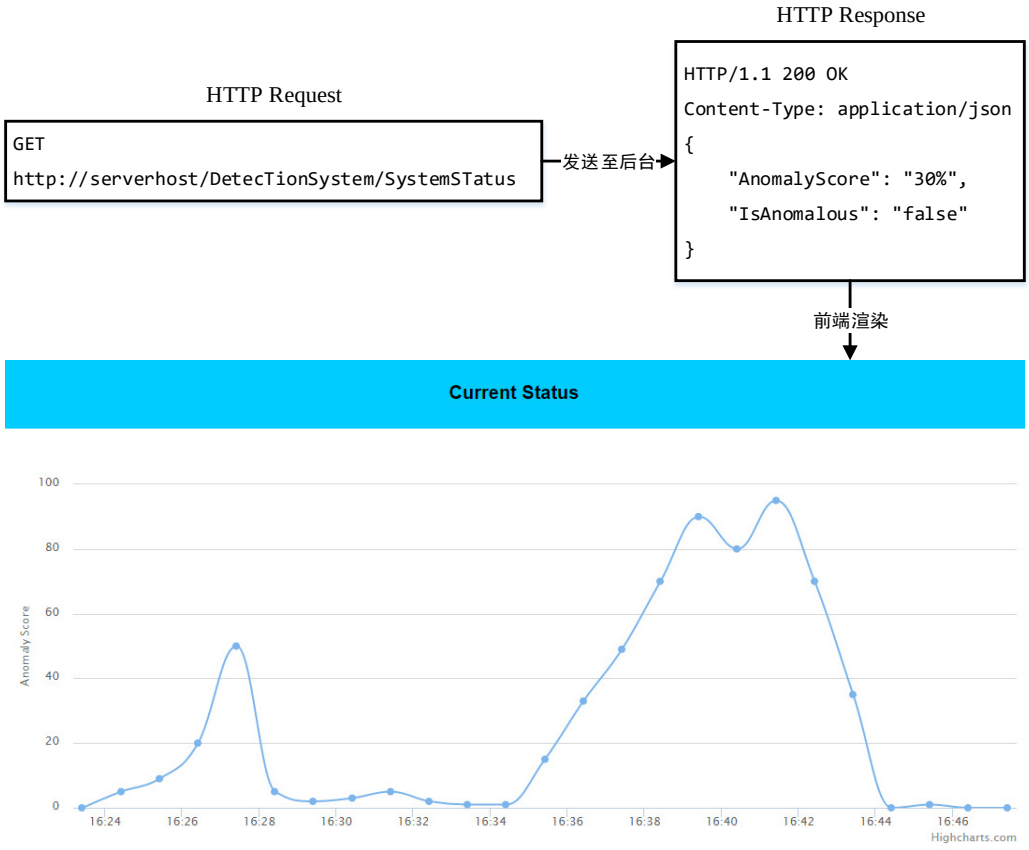


图5-11 前端展示流程

Figure 5-11 Flow chart of front-end display

本章小结

本章详细介绍了基于行为异常检测的日志监控系统的原型实现。首先对系统需求进行了分析；接着对系统的整体框架、框架中各个模块以及模块之间的结构关系、数据流动关系做了描述；最后依次介绍了监控系统的日志采集模块、存储分析模块和前端展示模块，并用功能框架图、流程图、类图和功能列表进行辅助说明。

第六章 全文总结与展望

6.1 本文工作总结

日志监控一直是软件监控的有效手段之一，通过对系统产生的日志数据进行采集和分析，可以判断当前系统状态，并对可能发生的异常事件进行预测。通过对当前研究现状和所面临问题的分析，本文提出基于行为异常检测的日志监控方法。本文提出的方法和完成的工作体现在以下几个方面：

1. 提出了基于日志标准化和层次聚类的日志预处理方法。首先通过一系列正规化方法对日志数据进行正规化处理；然后利用基于 complete linkage 的、以编辑距离为日志数据相似度的凝聚聚类方法对日志进行聚类。文中提出的聚类相似度定义和剪枝策略提高了聚类准确率，优化了聚类粒度。经过预处理后的日志数据被归类为不同类型，具有良好结构，方便后续行为模式的提取和异常行为的判断。

2. 提出了基于行为异常的通用日志异常检测模型。算法引入了连续子序列理论中对时间序列不和的检测原理，首先把日志流转换为行为序列，然后在行为序列集中训练行为模式，最后通过行为模式计算实时日志流的异常指数。通过异常值的大小对系统状态进行判断，预测可能发生的系统异常。通过与传统算法在实验数据集上的测试结果对比，证明了本文提出的日志检测方法的有效性。

3. 利用提出的检测算法实现了通用日志监控系统。结合已有的日志管理方案和本文提出的行为异常检测算法，设计实现了包含日志采集模块、存储分析模块和前端展示模块的日志监控系统。

6.2 未来工作展望

本文对基于日志监控的异常检测技术进行了理论研究和系统实现，通过对真实数据的实验证明了论文提出的异常检测方法不仅提高了检测准确率，同时一定程度上解决了处理大量数据、日志结构不统一的问题，是一个通用的日志检测算法。然而由于时间有限，现有研究仍有所不足。在已经完成工作的基础上，对于下一步研究我们有如下展望和规划：

1. 进一步降低模型训练和异常检测算法运行时间复杂度。虽然选择了优化过的算法策略，但当前算法仍然有一定延迟，特别是模型训练阶段的运算带来了很大的计算压力。下一步将寻找更高效的算法或对算法进一步优化，以提高计算效率。

2. 提高异常检测技术的通用性。目前的检测方案对某些类型的日志检测效果并

不是很好，为了提高检测方法的通用性，下一步将尝试从其他角度提取日志类型等特征。

3. 训练算法并行化。目前的模型训练方法并没有有效利用并行计算、特别是大数据计算的优势，下一步是使训练算法并行化，使其运行在 Hadoop MapReduce 等离线计算平台，提高训练效率。

4. 新的日志检测方法。关于日志异常检测的研究从未停止过，不少日志监控技术已经得到了广泛应用。我们也不会止步于现在的研究内容，而是会探索新的方法，继续完善日志监控技术。

参考文献

[1] 中国报告大厅. 2014 年 1-12 月我国软件行业发展情况解析[EB/OL]. [2015-1-27]. http://www.chinabgao.com/stat/stats/40324.html.

[2] Zheng, Z. et al. System log pre-processing to improve failure prediction[C]. Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on. IEEE, 2009.

[3] Chandola, V., A. Banerjee and V. Kumar. Anomaly detection: A survey[J]. ACM computing surveys (CSUR) 41.3 (2009): 15.

[4] Oliner, A. and J. Stearley. What supercomputers say: A study of five system logs[C]. Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on. IEEE, 2007.

[5] Chandola, V., A. Banerjee and V. Kumar. Anomaly detection for discrete sequences: A survey[C]. Knowledge and Data Engineering, IEEE Transactions on 24.5 (2012): 823-839.

[6] Salfner, F. and S. Tschirpke. Error Log Processing for Accurate Failure Prediction[C]. WASL. 2008.

[7] Shen, Q., J. Cao and H. Gu. A Similarity Network Based Behavior Anomaly Detection Model for Computer Systems[C]. Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on. IEEE, 2014.

[8] Vaarandi, R. A data clustering algorithm for mining patterns from event logs[C]. Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM). 2003.

[9] Yamanishi, K. and Y. Maruyama. Dynamic syslog mining for network failure monitoring[C]. Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005.

[10] Sahoo, R. K. et al. Critical event prediction for proactive management in large-scale computer clusters[C]. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.

[11] Fronza, I. et al. Failure prediction based on log files using Random Indexing and Support Vector Machines[J]. Journal of Systems and Software 86.1 (2013): 2-11.

[12] Juvonen, A. and T. Hamalainen. An Efficient Network Log Anomaly Detection

System Using Random Projection Dimensionality Reduction[C]. New Technologies, Mobility and Security (NTMS), 2014 6th International Conference on. IEEE, 2014.

[13]Lim, C., N. Singh and S. Yajnik. A log mining approach to failure analysis of enterprise telephony systems[C]. Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, 2008.

[14]Chen, C., N. Singh and S. Yajnik. Log analytics for dependable enterprise telephony[C]. Dependable Computing Conference (EDCC), 2012 Ninth European. IEEE, 2012.

[15]Yen, T. et al. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks[C]. Proceedings of the 29th Annual Computer Security Applications Conference. ACM, 2013.

[16]Xu, W. et al. Detecting large-scale system problems by mining console logs[C]. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

[17]黄锦,李家滨. 基于防火墙日志信息的入侵检测研究[J]. 计算机工程,2001,09:115-117.

[18]李勋章. 网络日志监控及安全审计系统的设计与实现[D].电子科技大学,2012.

[19]石彪. 网络环境下的日志监控与安全审计系统研究与实现[D].国防科学技术大学,2004.

[20]Chuvakin A. A., K. J. Schmidt and C. Phillips. 日志管理与分析权威指南[M]. 机械工业出版社, 2014: 25-39

[21]薛文娟. 基于层次聚类的日志分析技术研究[D].山东师范大学,2013.

[22]朱宝金. 面向云计算系统的日志过滤系统的设计与实现[D].杭州电子科技大学,2014.

[23]开源日志系统比较. [2011]. http://dongxicheng.org/search-engine/log- systems.

[24]Chukwa: Architecture and Design. [2010]. https://chukwa.apache.org/docs/r0.4.0/design.html.

[25]Han, J. and Micheline Kamber. 数据挖掘概念与技术（原书第 3 版）[M]. 机械工业出版社, 2012.

[26]陈文臣. Web 日志挖掘技术的研究与应用[D].中国科学院研究生院（计算技术研

究所）,2005.

[27]李航. 统计学习方法[M]. 清华大学出版社, 2012.

[28]CFDR Data. [2006]. The LANL data. https://www.usenix.org/cfdr-data.

[29]Schroeder, B. and G. Gibson. A large-scale study of failures in high-performance computing systems[C]. Dependable and Secure Computing, IEEE Transactions on 7.4 (2010): 337-350.

[30]孙吉贵,刘杰,赵连宇. 聚类算法研究[J]. 软件学报,2008,01:48-61.

[31]Xu, R. and D. Wunsch. Survey of clustering algorithms[C]. Neural Networks, IEEE Transactions on 16.3 (2005): 645-678.

[32]Fu, X. et al. LogMaster: mining event correlations in logs of large-scale cluster systems[C]. Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on. IEEE, 2012.

[33]Everitt, B., S. Landau, and M. Leese, Cluster Analysis[M]. London: Arnold, 2001.

[34]Gusfield, D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology[M]. Cambridge University Press, 1997.

[35]Keogh, E. et al. Finding the most unusual time series subsequence: algorithms and applications[J]. Knowledge and Information Systems 11.1 (2007): 1-27.

[36]Keogh, E., J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence[C]. Data mining, fifth IEEE international conference on. IEEE, 2005.

[37]Peters, T. A. The history and development of transaction log analysis[J]. Library hi tech 11.2 (1993): 41-66.

[38]Johnson, S. C. Hierarchical clustering schemes[J]. Psychometrika 32.3 (1967): 241-254.

[39]Berkhin, P. A survey of clustering data mining techniques[J]. Grouping multidimensional data. Springer Berlin Heidelberg, 2006. 25-71.

[40]Scikit, http://scikit-learn.org/stable/

[41]hierarchical-clustering-java, https://github.com/lbehnke/hierarchical-clustering-java/

[42]Bhuyan, M, H., D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: methods, systems and tools[J]. Communications Surveys & Tutorials, IEEE 16.1 (2014): 303-336.

[43] Jiang, Y. et al. Real time contextual collective anomaly detection over multiple data streams[C]. Proceedings of the ODD (2014): 23-30.

[44] Kittler, J. et al. Domain anomaly detection in machine perception: A system architecture and taxonomy[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on 36.5 (2014): 845-859.

[45] Hodge, V. J. and Jim Austin. A survey of outlier detection methodologies[J]. Artificial Intelligence Review 22.2 (2004): 85-126.

[46] Chandola, V., V. Mithal and V. Kumar. Comparative evaluation of anomaly detection techniques for sequence data[C]. 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008.

[47] Oliner, A., A. Ganapathi and W. Xu. Advances and challenges in log analysis[J]. Communications of the ACM 55.2 (2012): 55-61.

致谢

在两年半的研究生生涯中，我要对所有在学习和生活上支持和帮助过我的人们表示感谢。

首先感谢我的导师曹健教授。在攻读硕士期间，曹老师在我的课程学习、学术研究和毕业设计上始终给予了认真负责的指导和无微不至的关怀。在曹老师的指导和帮助下，我完成了学术方向的研究、实验室项目和本文的研究工作；曹老师渊博的专业知识、严谨的科研态度、细致的治学作风和亲切朴实的为人都深深感染了我，是我学习的榜样；曹老师对我在学习态度、科研方法和未来工作和生活上的指导是我的宝贵财富，将使我获益终身。

然后我要感谢和我一起学习和生活的同学们。在云计算项目组期间，我和陈昌源学长一起进行了云计算和虚拟化的研究，共同完成了实验室项目，在相互帮助和学习的过程中结下了友谊；之后在监控项目组，我和顾骅同学、于晨同学以及钱诗友博士合作参与软件监控项目研究，他们的学习和科研态度值得我学习。还有王烺、蒋雨生等同学，大家在 CIT 一起生活成长的时光将是我永远的美好回忆。

最后，感谢我的父母、哥哥和姐姐。在我成长的岁月里他们一直支持着我、给我鼓励，有这样的家人让我觉得无比幸福，他们的关爱是我努力前进的不竭动力。

攻读硕士学位期间已发表的论文

一 已发表、录用的论文：

[1] Sizhong Du and Jian Cao. Anomalous Behavior Detection Approach Based on Log Monitoring[C]. International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC), 2015.（第一作者）