**Computer Science and Application** 计算机科学与应用, 2020, 10(1), 98-111
Published Online January 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.101011

Hans汉斯

# A System Log Parsing Algorithm Based on Cluster Algorithm

**Wenjun Huo**

Department of Computer Science and Technology, Tongji University, Shanghai
Email: huowj@tongji.edu.cn

Received: Dec. 26th, 2019; accepted: Jan. 10th, 2020; published: Jan. 17th, 2020

## Abstract

**System log is an important source of checking system status in software system. The runtime status report and error information contained in system log are widely used in system operation and maintenance. With the current software system becoming increasingly large and complex, large-scale software systems usually use log analysis mining technology to automatically mine key system information from the system log. Log data is used in anomaly detection, root analysis, behavior analysis and other applications. Log data is usually unstructured text data. Before using data mining algorithm to train the log data, we need to use log parsing algorithm to process the original log data structurally. According to the characteristics and requirements of log data analysis and mining technology, this paper proposes a log parsing algorithm based on clustering algorithm, which can analyze the original log data from the number of original logs. A fixed depth tree structure model is also proposed to store the message template to realize the fast structure of new log messages and the detection of abnormal log messages. Through the experiments on the specific system log data, it is proved that the log parsing algorithm in this paper has high accuracy and generality.**

## Keywords

**System Log, Log Parsing, Anomaly Detection**

# 一种基于聚类的系统日志解析算法

**霍文君**

同济大学计算机科学与技术系，上海
Email: huowj@tongji.edu.cn

收稿日期：2019年12月26日；录用日期：2020年1月10日；发布日期：2020年1月17日

文章引用：霍文君. 一种基于聚类的系统日志解析算法[J]. 计算机科学与应用, 2020, 10(1): 98-111.
DOI: 10.12677/csa.2020.101011

## 摘　要

**系统日志是软件系统中检查系统状态的重要来源，系统日志中包含的运行时状态报告以及错误信息被广泛地用于系统运维中。随着现阶段软件系统变得日益庞大和复杂，大型软件系统通常会使用日志分析挖掘技术来自动地从系统日志中发掘系统关键信息，日志数据被用于异常检测、根因分析、行为分析等等应用中。日志数据通常是无结构化的文本数据，在使用数据挖掘算法对日志数据进行训练之前，需要使用日志解析算法对原始日志数据进行结构化处理，本文根据日志数据分析挖掘技术的特点和需求，提出一种基于聚类算法的日志解析算法，可以从原始日志数据中提取消息模板和事件序列，同时提出了一种固定深度的树结构模型对消息模板进行存储以实现新日志消息的快速结构化和异常日志消息的检测。通过在特定系统日志数据上的实验证明本文的日志解析算法具有较高的准确性和通用性。**

## 关键词

**系统日志，日志解析，聚类，异常检测**



## 1. 引言

系统日志是在系统实时运行时记录软硬件问题的文本信息，系统日志可以被用来监视系统运行时发生的事件信息，比方说事件的时间戳、一个请求的 IP 地址或者一个异常任务的状态[1]。系统日志蕴含的丰富信息可以帮助系统开发人员和维护人员更好地理解系统行为并在生产过程中检测和定位系统异常[2]。除此以外，对系统日志的分析和挖掘技术也被应用于金融和政府机构中，比方说，交易系统日志被用来进行异常交易的检测，电力系统日志被用来进行安全隐患的查找。

随着现阶段软件系统规模和复杂度的日益增加，系统日志规模也变得更加庞大，大型软件系统会按照每小时 10 gb 的速度产生日志。传统的人工方式对这些日志信息进行分析是非常不切实际的，因此，全自动的基于日志数据的分析挖掘技术是迫切需要的。比较典型的日志挖掘技术包括异常检测[3] [4] [5]、行为分析、程序验证[6]、根因分析[7] [8]等等。另外，由于现阶段算法和算力的发展，越来越多的开发人员选择使用机器学习和深度学习技术进行日志挖掘[9] [10]。然而，由于开发人员在记录系统日志为了简单和方便会使用简单文本，因此系统日志通常是无结构的文本数据，在使用数据挖掘算法对系统日志建立模型训练之前，需要使用日志解析算法将原始的日志数据结构化[11]。

Dec 27, 2019 @ 15:19:08.000 GET /api/saved_objects/_find?type=index-pattern&per_page=10000&page=1&default_search_operator=OR 200 6ms - 9.0B src: 172.23.2.176 dest: 111.187.17.115

**Figure 1.** A typical log
**图 1.** 一条典型的日志

一条典型的日志，如图 1 所示，记录了一件特定的系统事件，包括多个部分：时间戳(记录了事件发生的具体时间)，源 IP 地址和目的 IP 地址，以及原始的消息内容[4]。日志解析技术一般是针对日志事件中的消息部分，即图 1 中的“GET/api/saved_objects/…200 6 ms -9.0B src: 172.23.2.176 dest: 111.187.17.115”。

消息部分通常可以分为常量部分和变量部分，常量部分是固定不变的文本内容，通常定义了一个消息模板的事件类型，变量部分显示了系统的运行时信息，比方说状态的值和参数(IP 地址、持续时间、文件路径等等)，变量部分在不同的日志中会有不同的值。日志解析算法的目标是自动化地从日志消息部分中提取常量部分和变量部分，并将每一条日志转化为一个特定的事件，比方说图 1 所示例子中的常量部分为“GET * * -9.0B src: * dest: *”，变量部分我们用通配符替代，常量部分就代表了一个特定的事件。在本文中我们使用消息模板/事件类型来指消息部分中的常量部分。

日志解析技术是日志挖掘的关键一环。传统的日志解析算法包括正则表达式匹配[12]、基于源代码的方式，或者基于规则式的方式[13]。现代软件系统的迭代和发展速度过于迅速，一般系统中定义日志消息的源代码很难获得，并且日志的规则需要对系统非常熟悉的专业人士来制订，当系统发生变化时，规则也需要同时发生改变，这是比较费时费力的，因此传统的日志解析算法并不能适用于现在的软件日志系统。

一般来说，日志消息的常量部分是由系统中定义日志输出的代码决定的，通常同一条消息模板产生的日志消息都非常相似。聚类算法的思想即是通过比较实例之间的距离，并将相似的实例聚为一类，本文利用这种思想，提出了一种基于聚类的日志解析算法，并使用一种固定深度的树结构模型来存储消息模板，以实现对新的日志消息快速解析和异常日志的检测。

本文的组织如下：第二部分本文将会给出相关学术工作以及符号定义，第三部分介绍一种基于聚类的日志提取算法，第四部分给出本文实验在特定系统日志数据上的实验结果，第五部分为总结与展望。

## 2. 问题描述与相关工作

### 2.1. 问题描述

系统日志一般是独立的文本数据，文本内容记录了系统中发生的事件，包括消息部分和一些事件属性，事件属性包括时间戳、消息级别等等。消息部分时由源代码定义的，比方说下面的一行代码：

$$printf\left(message, Connection\ from\ \%s\ port\ \%d, ipaddress, portnumber\right);$$

可以产生下面的日志消息：

*Connection from* 192.168.10.6 *port* 25.

*Connection from* 192.168.10.6 *port* 80.

*Connection from* 192.168.10.7 *port* 25.

*Connection from* 192.168.10.8 *port* 21.

这些消息可以形成一个消息模板/事件类型，将其中的变量部分用通配符表示，则该条消息模板可以表示为：

*Connection from* * *port* *.

大部分情况下，一条消息模板对应一条打印语句，日志解析就是从大量的原始日志数据中提取消息模板，这样一条日志就对应一个事件类型，按照时间顺序排列即得到事件序列。

图 2 展示了一个日志解析工具的主要功能，其中原始日志数据来源与在线实训平台功夫编程 kf-coding，由图 2 可知，原始的系统日志是非结构的，每一条日志都包含时间戳、类别、消息内容，真实情形下，系统在一天中会产生成千上万条类似这样的日志信息，日志解析将会读取原始日志消息并生成消息模板和事件序列。

### 2.2. 相关工作

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 [14])。

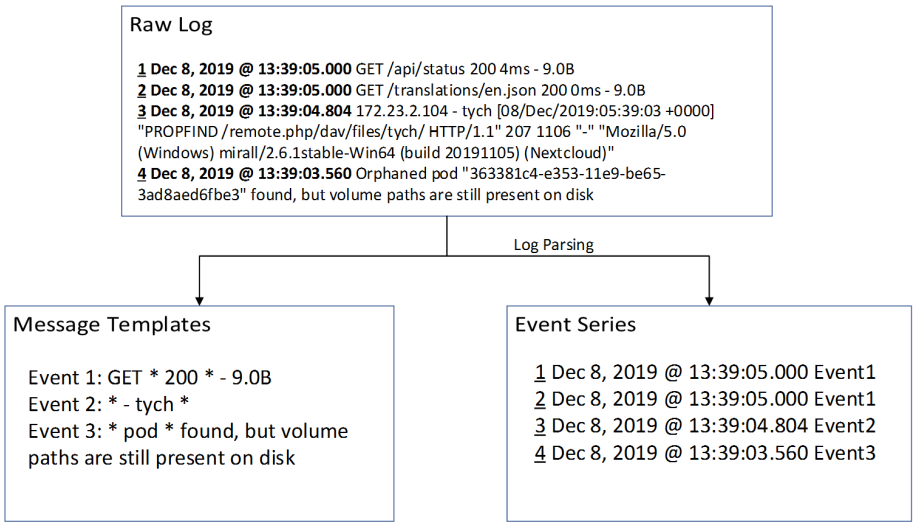


**Figure 2.** Main functions of a log parsing tool
**图 2.** 日志解析工具的主要功能

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。我们在这里介绍三种典型的基于数据的自动化的日志解析方法。

Risto Vaarandi 等[15] [16]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：1) 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；2) 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；3) 生成日志模板，从第二步中建立的日志簇中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) [17]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：1) 通过日志消息的不同长度进行划分；2) 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；3) 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；4) 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。

LogSig [18]是一种比较新的日志解析方法，包含三个过程：1) 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；2) 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；3) 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。

## 2.3. 符号定义

我们在这个小节给出基于聚类的日志解析算法的相关符号定义。

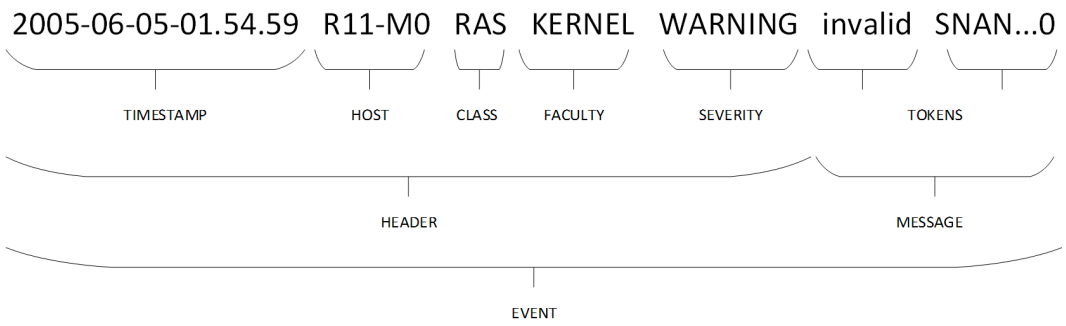


**Figure 3.** Log tagging
**图 3.** 日志标注

定义 1：事件日志：定义事件日志为记录系统内发生的事件或对应用程序跟踪记录的文本。

定义 2：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应用程序发生的一次事件。

定义 3：消息：消息是指事件中除去时间戳、类型、标签等信息，只记录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 所示的事件文本中，invalid SNAN..0 为消息部分。

定义 4：标记：消息部分中被分隔符划分的独立的单词为标记。一般情况下，分隔符为空格。

定义 5：消息长度：消息部分中标记的个数。

定义 6：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。

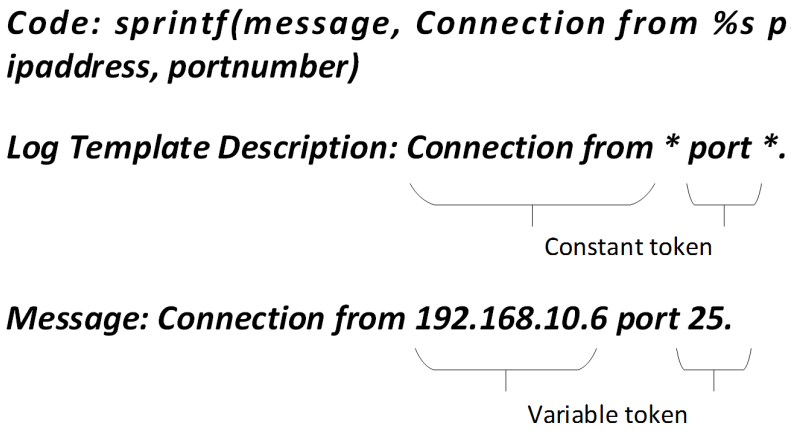


**Figure 4.** Message template description example
**图 4.** 消息模板描述示例

定义 7：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 4 所示，一般一条消息模板描述对应着一条打印日志语句。

定义 8：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 4 中所示，标记变量为 192.168.10.6 和 25。

定义 9：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 4 中所示，标记变量为 192.168.10.6 和 25。

定义 10：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. 基于聚类的日志解析算法

本节具体介绍基于聚类的日志解析算法的主要思想和具体步骤。

### 3.1. 算法概述

通过对大量的日志数据进行分析发现，由相同的消息模板产生的日志通常都比较相似，这与它们都有相同的常量标记有关，聚类算法的主要思想是通过比较实例之间的相似性，将相似的实例聚为一类，这与日志解析的目的类似，日志解析是将相似的日志消息(由同样的消息模板产生的日志消息)聚为一类并从中提取公共部分作为消息模板。另外由于聚类算法会涉及到大量的相似度计算，因此我们在聚类算法的前一步会首先对日志数据进行处理以减少时间复杂度。

根据上文的分析，本文基于聚类算法的日志解析算法通过四个步骤实现日志解析的功能，预处理会统计每个消息的消息长度，并使用简单的正则表达式替换消息中明显的标记变量，将相同长度的消息分为一组，在同一组中使用聚类算法将同组中相似的消息聚为一类，由于经过聚类后，相同类别的消息也许还是由不同的消息模板产生的，在聚类之后会在同类中进行分组，使得每一个组都对应一条独立的打印语句，在每一组中生成消息模板/事件类型。

### 3.2. 事件分组和标记变量替换

消息长度定义为消息中独立的标记的数量，我们认为，由相同的消息模板产生的日志消息具有相同的消息长度[19]，如图 5 中所示，事件 1 和事件 2 消息长度是不同的，将相同的消息长度分为一组的另一个主要目的是，聚类算法会进行大量的距离计算，预处理可以减少聚类的时间复杂度，并且有利于提升整体算法的运行效率。

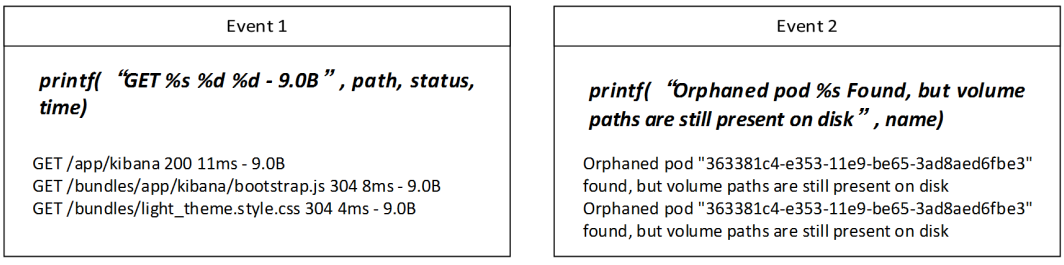


**Figure 5.** Message template print code example
**图 5.** 消息模板打印代码示例

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 5 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 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 1 所示。

**Table 1.** Algorithm MessagePreprocess
**表 1.** 算法 MessagePreprocess

```
Algorithm MessagePreprocess
Input: Message Series S, Empirical Rules R
Output: Message Maps
Function Message Maps = MessagePreprocess (S, R)
Message Maps = dict
For message in M:
    message_length = len(message.split(' '))
    For token in message.split(' '):
        IF token obey R
            token = '*'
    End
    IF message_length in dict:
        dict[message_length].append(message)
    Else:
        dict[message_length] = [message]
End
```

### 3.3. 聚类

我们在上一步得到的每一个 MessageMap 中使用聚类算法将相似的消息聚为一类，聚类中比较重要的是距离的度量，我们在这里使用加权的字符串编辑距离[3]，具体定义如下：

$$WED(S_1, S_2) = \sum_{i=1}^{EO} \frac{1}{1+e^{(x_i - v)}} \tag{1}$$

其中 EO 为 $S_1$ 到 $S_2$ 的操作次数，$x_i$ 是第 $i$ 次操作时单词的索引，$v$ 是控制权重的超参数。

本文对于两个不同的消息，会首先计算它们的加权编辑距离，如果小于阈值，则它们之间具有联系，我们将所有两两之间具有联系的消息分为一个组 MessageGroup。

本文在这里给出一种阈值的确定方法，通过计算每对消息之间的加权编辑距离，我们可以得到训练集中所有对的加权编辑距离的集合，对它们进行 k-means 聚类，选定 k 为 2，选定其中类中心值比较小的类中最大加权距离作为阈值。

表 2 给出了聚类的完整算法 MessageCluster，对于 MessageMap 中的每一个消息，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。算法的伪代码如表 2。

**Table 2.** Algorithm MessageCluster
**表 2.** 算法 MessageCluster

```
Algorithm MessageCluster
Input: Message Maps M
Output: Message Clusters
Function Message Clusters = MessageCluster (M)
Message Clusters = M.copy()
For key, value in M.items():
```

**Continued**

```
4:     distances = WED(i, j) for i, j in value
5:  cluster1, cluster2 = kmeans(distances, k=2) //
6:  sigma = max(cluster1)
7:  for message in value:
8:     link(message, i) if WED(message, i) < sigma for i in value
9:  Do:
10:    put i, j in a cluster for i, j in value
11: While value is not empty
12: Update Message Clusters
```

### 3.4. 类中分组

经过预处理和聚类两个步骤，在一个 MessageCluster 中的消息也可能出自不同的消息模板，因此我们在这一步进行对 MessageCluster 的分组。

这一步的主要思想为，对于一个 MessageCluster，N 为其中消息的消息长度，设其中消息的公共标记有 M 个，将每条消息划分为 M + 1 个位置，在每一个位置上，我们统计共有多少种不同的标记序列，用 V 表示，一般来说，V 的值越大说明该位置越不可能式标记常量，通过引入阈值，当 V 超过阈值，我们认为这个位置为标记变量，并且不再进行划分操作，当 V 不超过阈值时，则该位置为标记常量，我们将这个 MessageCluster 划分为 V + 1 组。

第三步完整的算法，对于每一个 Message Cluster，统计每一个非公共序列的部分共有多少个不同的标记，并与阈值作比较，小于阈值就做一次划分，算法的伪代码如表 3。

**Table 3.** Algorithm SplitGroup
**表 3.** 算法 SplitGroup

**Algorithm** SplitGroup
**Input**: *Message Clusters C*, threshold $\varsigma$
**Output**: *Message Groups*

```
1: Function Message Groups = SplitGroup (C)
2: For cluster in C:
3:    For i from 1 to M+1:
4:       let V be the number of different tokens in position i
5:       if V < ς:
6:          split C into V+1 groups
```

### 3.5. 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。

### 3.6. 消息模板树

前面的步骤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 6，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。

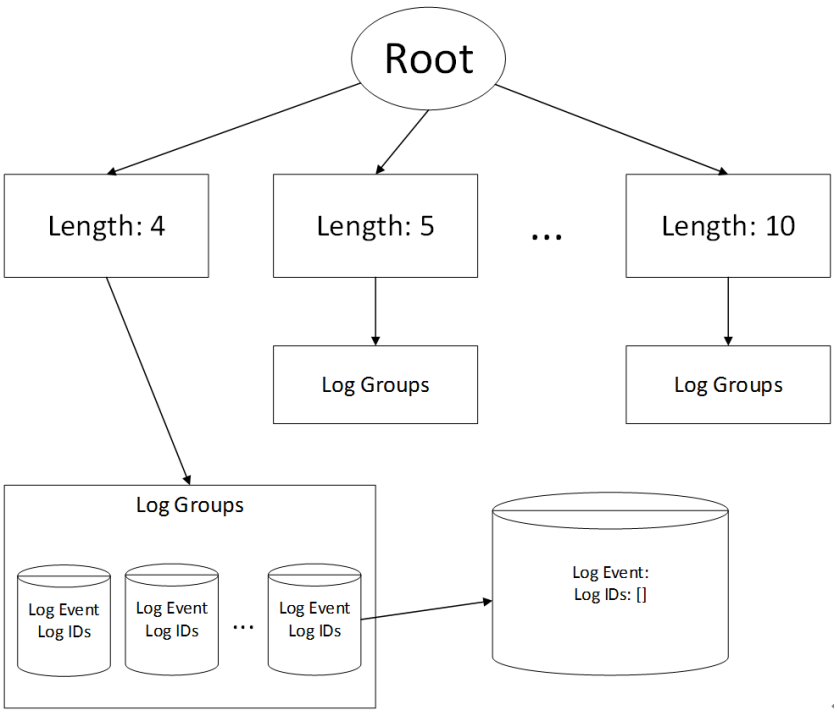


**Figure 6.** Message template tree
**图 6.** 消息模板树

## 4. 实验结果与分析

### 4.1. 数据来源

本文的日志数据来源于在线实训平台功夫编程 kfcoding，功夫编程是采用微服务架构思想构建的，旨在满足三种不同的业务场景下不同用户的需求，第一中业务场景为个人学习场景，在该场景下，个人用户可以通过该平台学习和发现最新的前沿 IT 技术，如云计算、大数据、人工智能、机器学习、容器技术等等。用户不仅可以看到教程书籍，还能够在平台上进行实践。与此同时，用户还可以自己创建/编辑/删除书籍，即具有书籍管理功能。

第二种业务场景为学校教学场景。在该场景下，学校教师可以通过该平台开设课程，上传课件等资源，并布置作业。相应的，选修该课程的学生可以在平台上进行课程学习，边看课件边动手操作，并可以通过该平台提交作业。

第三种业务场景为企业实训活动场景。在该场景下，实训活动举办方(企业/组织)可以定制自己的交互式产品文档，采用平台提供的定制化的统一配置环境，举办实训活动或团队培训。而用户通过参与活

动即可以在真实的使用环境中快速学习或体验相关产品。

功夫编程的整个系统架构如图 7 所示：

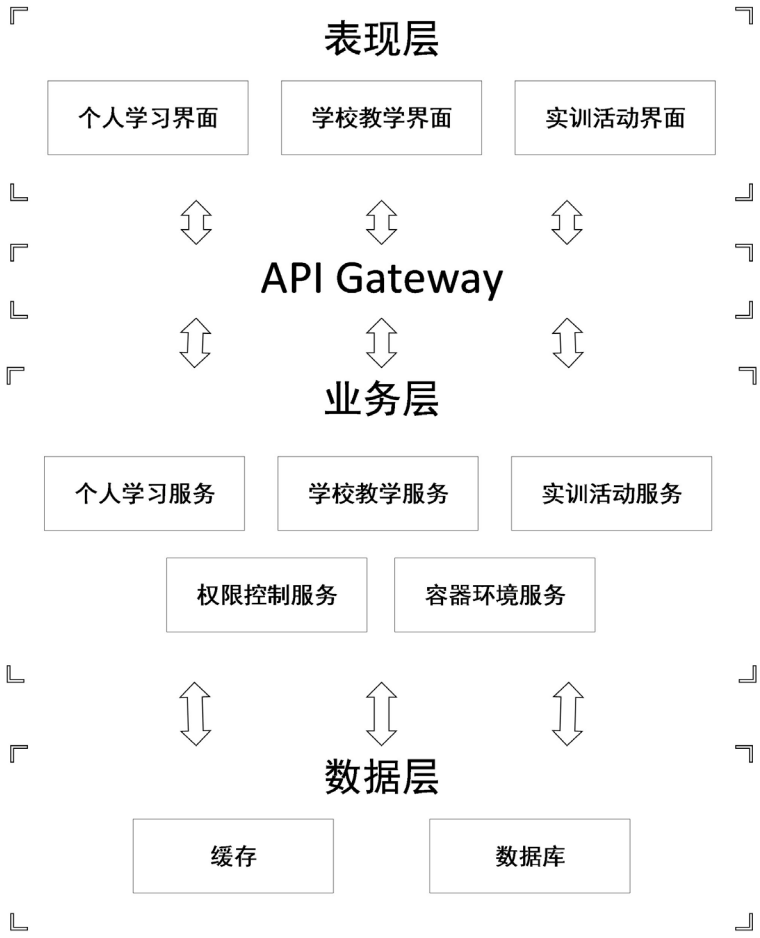


**Figure 7.** The Architecture diagram of kf-coding
**图 7.** 功夫编程架构图

功夫编程会以大概每小时 4000 条日志的速率产生日志事件，我们搜集了其中一个月的日志数据进行日志解析算法，这一个月的日志数据作为日志训练集，经过日志解析的算法，无结构的原始日志数据会解析出消息模板和事件序列，我们采用固定深度的消息模板树来存储消息模板，以实现对新的消息快速解析和异常日志的检测。

## 4.2. 实验结果

本文在搜集到的日志数据上进行日志解析，生成消息模板和事件序列，其中消息模板会生成固定深度的消息模板树，图 8 是典型的日志数据在日志解析算法中的具体效果。为了可以体现本文日志解析算法的算法效果，我们在之后的日志挖掘过程选定系统异常检测来体现日志解析算法的效果。

我们在得到的事件序列上使用固定窗口技术提取主要的特征作为异常检测的数据，其中固定窗口技术保证了特征的时间先后关系，之后使用时间序列检测异常点和异常序列的方法对系统异常进行检测，我们主要使用长短期记忆网络(LSTM)来进行时间序列的异常检测[20]，长短期记忆网络(LSTM，Long Short-Term Memory)是一种时间循环神经网络，是为了解决一般的 RNN (循环神经网络)存在的长期依赖问题而专门设计出来的。

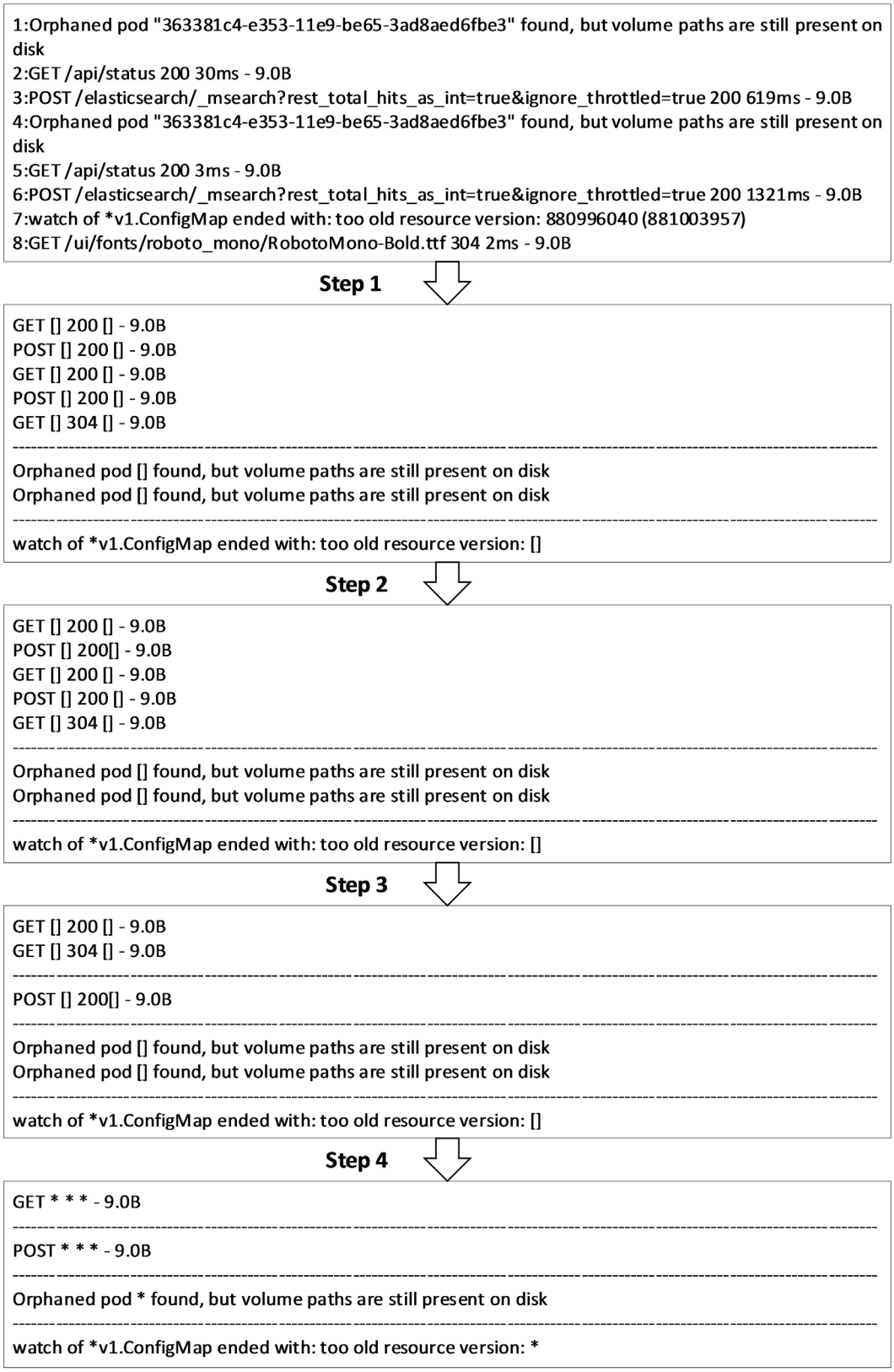


**Figure 8.** Sample log resolution results
**图 8.** 日志解析结果示例

本文使用固定窗口技术选取的特征主要为事件的总体频数和单独的事件频数[21]，由于系统异常检测对准确性要求较高，因此我们需要能够准确完备展现系统状态的特征，本文在对系统进行分析和对候选的一些特征进行观察之后，选定事件总体频次和独立事件频次作为特征供时间序列异常检测步骤使用。

在提取特征之后，我们将所有的数据按照时间顺序排列得到多条时间序列，按照时间顺序选取前面70%的数据作为训练集进行 LSTM 的训练，并使用后面的 30%的数据作为测试集进行测试。

图 9 中给出了我们选定的四条时间序列，其中真实的异常点用红色标注。

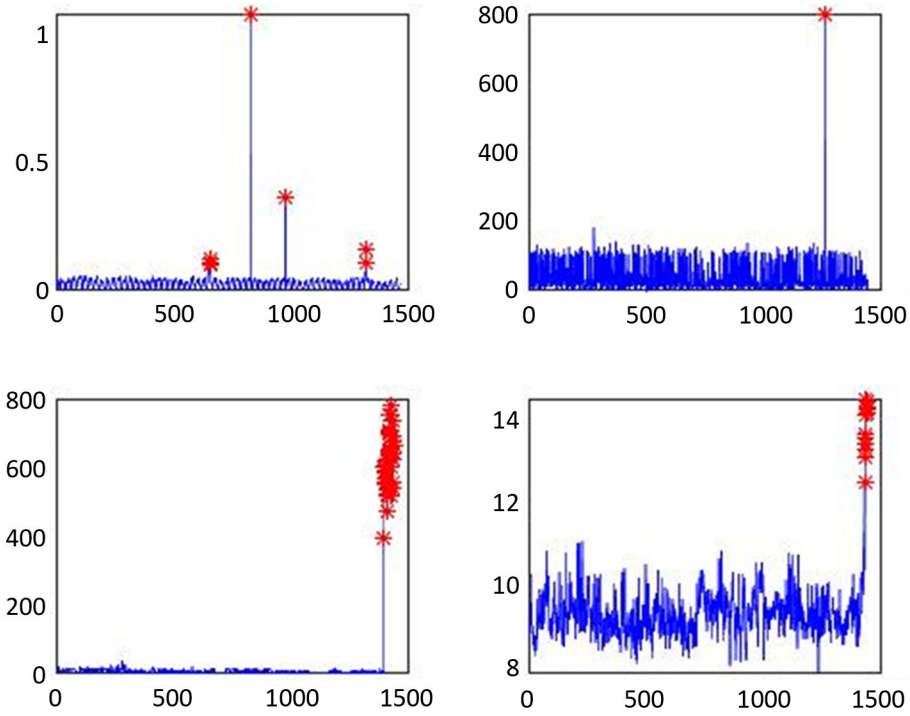


**Figure 9.** Sample log resolution results
**图 9.** 日志解析结果示例

表 4 给出了本文的 LSTM 算法在几条时间序列上的检测正确率，本文给出了 TP、FP、FN、precision、recall 和 f1-score。

**Table 4.** The effect of anomaly detection on four time series
**表 4.** 四条时间序列上的异常检测效果

| Dataset | TP | FP | FN | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|
| TimeSerie1 | 1057 | 12 | 98 | 0.9152 | 0.9888 | 0.9505 |
| TimeSerie2 | 980 | 4 | 19 | 0.981 | 0.9959 | 0.9884 |
| TimeSerie3 | 875 | 30 | 43 | 0.9532 | 0.9669 | 0.96 |
| TimeSerie4 | 967 | 23 | 290 | 0.7693 | 0.9768 | 0.8607 |

## 4.3. 结果分析

本文研究日志解析算法，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，通常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，本文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。本文给出的日志解析算法被用于特定分布式系统的日志数据，生成的结构化日志可以被用来进行特征提取和系统异常检测。

## 5. 结论

本文针对大型分布式系统日志数据，给出了一种基于聚类算法的日志解析方法，通过四个步骤，可以从原始的无结构日志中提取消息模板以及事件序列，将原始系统日志结构化，除此以外，本文提出一种固定深度的树结构模型，该模型可以被用来对新的日志消息进行解析以及对异常的日志消息进行检测。本文将提出的算法使用在特定的分布式系统日志上以得到结构化日志数据，并从中提取关键特征以用于系统异常检测，在异常检测的步骤中我们使用了深度学习的方法，实验结果表明本文的日志解析算法具有较高的准确性和通用性，并能较好地适用于各类日志挖掘算法流程中。

## 参考文献


[1] Yuan, D., Park, S. and Zhou, Y.Y. (2012) Characterizing Logging Practices in Open-Source Software. 34*th International Conference on Software Engineering*, Zurich, 2-9 June 2012, 1-11. https://doi.org/10.1109/ICSE.2012.6227202

[2] Oliner, A. and Stearley, J. (2007) What Supercomputers Say: A Study of Five System Logs. 37*th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Edinburgh, 25-28 June 2007, 575-584.

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

[4] He, S.L., Zhu, J.M., He, P.J. and Lyu, M.R. (2016) Experience Report: System Log Analysis for Anomaly Detection. 27*th International Symposium on Software Reliability Engineering*, Ottawa, 23-27 October 2016, 207-218.

[5] Xu, W., Huang, L., Fox, A., Patterson, D. and Jordon, M. (2009) Detecting Large Scale System Problems by Mining Console Logs. *Proceedings of the ACM Symposium on Operating Systems Principles*, Haifa, October 2009, 117-132. https://doi.org/10.1145/1629575.1629587

[6] Mi, H., Wang, H., Zhou, Y., Lyu, R. and Cai, H. (2013) Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, **24**, 1245-1255. https://doi.org/10.1109/TPDS.2013.21

[7] Zou, D.-Q., Qin, H. and Jin, H. (2016) UiLog: Improving Log-Based Fault Diagnosis by Log Analysis. *Journal of Computer Science and Technology*, **31**, 1038-1052. https://doi.org/10.1007/s11390-016-1678-7

[8] Nagaraj, K., Killian, C. and Neville, J. (2012) Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems. *Proceedings of the* 9*th USENIX Conference on Networked Systems Design and Implementation*, San Jose, 25-27 April 2012, 1-14.

[9] Lin, Q.W., Zhang, H.Y., Lou, J.-G., Zhang, Y. and Chen, X.W. (2016) Log Clustering Based Problem Identification for Online Service Systems. *IEEE/ACM* 38*th IEEE International Conference on Software Engineering Companion*, Austin, 14-22 May 2016, 102-111.

[10] Mi, H.B., Wang, H.M., Zhou, Y.F., Lyu, M.R.-T. and Cai, H. (2013) Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, **24**, 1245-1255. https://doi.org/10.1109/TPDS.2013.21

[11] Du, M. and Li, F.F. (2019) Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Transactions on Knowledge and Data Engineering*, **31**, 2213-2227. https://doi.org/10.1109/TKDE.2018.2875442

[12] Debnath, B., Solaimani, M., Gulzar, M.A., Arora, N., Lumezanu, C., Xu, J.W., Zong, B., Zhang, H., Jiang, G.F. and Khan, L. (2018) LogLens: A Real-Time Log Analysis System. *IEEE* 38*th International Conference on Distributed Computing Systems*, Vienna, 2-6 July 2018, 1052-1062. https://doi.org/10.1109/ICDCS.2018.00105

[13] Lou, J.-G., Fu, Q., Yang, S.Q., Xu, Y. and Li, J. (2010) Mining Invariants from Console Logs for System Problem Detection. *Proceedings of the* 2010 *USENIX Conference on USENIX Annual Technical Conference*, Boston, 23-25 June 2010, 1-14.

[14] Lang, D. (2013) Using SEC. *USENIX*; *Login*: *Magazine*, **38**, 38-43.

[15] Stearley, J. (2004) Towards Informatic Analysis of Syslogs. *IEEE International Conference on Cluster Computing*, San Diego, 20-23 September 2004, 1-10.

[16] Vaarandi, R. (2003) A Data Clustering Algorithm for Mining Patterns from Event Logs. *IP Operations and Management*, Kansas City, 3 October 2003, 119-126.

[17] Makanju, A., Zincir-Heywood, A. and Milios, E. (2009) Clustering Event Logs Lusing Iterative Partitioning. *Proceedings of International Conference on Knowledge Discovery and Data Mining*, Paris, 28 June-1 July 2009, 1255-1264. https://doi.org/10.1145/1557019.1557154

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

[19] Makanju, A., Zincir-Heywood, A.N. and Milios, E.E. (2012) A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Transactions on Knowledge and Data Engineering*, **24**, 1921-1936. https://doi.org/10.1109/TKDE.2011.138

[20] Bontemps, L., Cao, V.L., McDermott, J. and Le-Khac, N.-A. (2017) Collective Anomaly Detection Based on Long Short Term Memory Recurrent Neural Network.

[21] Lim, C., Singh, N. and Yajnik, S. (2008) A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems. *International Conference on Dependable Systems & Networks with FTCS and DCC* (*DSN*), Anchorage, 24-27 June 2008, 398-403. https://doi.org/10.1109/DSN.2008.4630109