基于启发式规则的流式在线日志解析方法

蒋忠元[1]，陶梅悦[1]，赵晓庆[1]，方晓彤[2]，李兴华[1]，马建峰[1]
（1.西安电子科技大学网络与信息安全学院，陕西 西安 710071；2.中国船舶集团有限公司综合技术经济研究院，北京 100081）

摘　要：为了解决现有日志解析方法中存在的解析不准确、效果不稳定等问题，提出了一种基于启发式规则的流式在线日志解析方法——启发式正则树（HRTree）。其在Drain方法解析结构树基础上，引入启发式规则对日志进行拆分构造，并优化了解析结构树的部分构造方式，从而解决了日志参数过拟合、不同系统日志解析结果不稳定的问题。实现了解析结果分类准确，且参数内容识别准确的解析效果。大量实验结果表明，所提出的HRTree方法在不同的系统日志上均展现了90%以上的解析准确率。

关键词：海量日志；日志解析；启发式规则；HRTree方法；准确率
中图分类号：TP39
文献标志码：A
DOI：10.11959/j.issn.1000−436x.2024071

Streaming online log parsing method based on heuristic rule

JIANG Zhongyuan[1], TAO Meiyue[1], ZHAO Xiaoqing[1], FANG Xiaotong[2], LI Xinghua[1], MA Jianfeng[1]
1. School of Cyber Engineering, Xidian University, Xi’an 710071, China
2. China Institute of Marine Technology & Economy, Beijing 100081,China

Abstract: To address the issues of inaccurate parsing and unstable performance in existing log parsing methods, a streaming online log parsing method based on heuristic rules, known as heuristic regex tree (HRTree), was proposed. Based on the drain method of parsing the structure tree, heuristic rules were introduced to split and construct the log, and some construction methods of the parse structure tree were optimized, so as to solve the problems of over fitting of log parameters and unstable parsing results of different system logs. Not only the classification of parsing results was accurate, but also the parameter content recognition was accurate. A large number of experimental results demonstrate that the proposed HRTree parsing method shows more than 90% parsing accuracy on different system logs.

Keywords: massive log, log parsing, heuristic rule, HRTree method, accuracy

0　引言

随着大数据时代的到来，现代服务系统的规模和复杂性迅速增加。一方面，完整的大型服务系统通常由数以千计的计算机同时运行维护；另一方面，大型服务系统可能为数以百万计的用户提供服务。这些服务系统的任何异常行为都可能影响系统可靠性和可用性，从而影响用户体验，甚至造成巨大经济损失[1]。因此，大型系统的可靠性和稳定性至关重要。

系统异常可能由各种原因引起，如软件错误、恶意攻击、内存泄漏或错误配置。维护人员需要仔细监控这些软件系统，以便快速发现并解决异常。

日志是一种存在于系统的通用性资源，是记录系统或者业务运行的重要数据源。日志数据包含了

收稿日期：2023−11−21；修回日期：2024−02−15
基金项目：国家重点研发计划基金资助项目（No.2022YFB2701800）；陕西省重点研发计划基金资助项目（No.2023-YBGY-270）；国家自然科学基金资助项目（No.62076191, No.61502375）
Foundation Items：The National Key Research and Development Program of China (No.2022YFB2701800), The Key Research and Development Program of Shaanxi Province (No. 2023-YBGY-270), The National Natural Science Foundation of China (No.62076191, No.61502375)

各种系统活动中的事件和行为，可以用于监控系统运行。

异常检测作为一种重要的数据挖掘技术，用于发现数据中的异常行为。随着大数据时代的到来，各种系统产生的数据量不断增加，异常检测技术也变得越来越重要，适合应用于故障分析[2]、性能诊断[3]、入侵和欺诈检测[4]等系统安全维护方面。

基于日志的异常检测作为一种主动的防御性技术，可以帮助系统维护人员发现和解决问题，保证系统的稳定可靠运行，避免系统异常造成巨大损失。因此，基于日志的异常检测已成为一项重要的研究内容。对于传统的简单独立系统，维护人员可以通过检阅日志中的关键字或者创建规则来定位和匹配异常[5-7]。但随着系统规模的扩展，系统日志越来越复杂，异常频率增高、种类多样，人工检测耗时且低效，不再适用于这样的数据规模和条件[8]。

高效准确的日志解析技术是有效异常检测的前提，传统的人工检阅或基于固定规则的解析方法难以应对日益增长的日志数据规模。日志解析面临新的挑战，主要可以总结为以下几个方面。

1) 数据规模复杂庞大，日志解析困难。大型系统会产生大量的日志信息，甚至达到每小时吉比特字节的数据[9]。如何对系统产生的日志信息进行自动化高速有效的信息提取存在较大困难，已有的解析方法在面对日志数据多样、类型复杂的系统日志时，无法展现稳定的解析效果。

2) 非结构化数据解析难。许多日志数据呈现出非结构化的特点，缺乏明确的模式和规则，传统的基于规则的解析方法常常难以处理这种非结构化数据。如何高效准确地从复杂的日志文本中识别关键特征，成为一个具有挑战性的问题。

本文首先系统综述日志解析国内外最新进展并凝练痛点难题；其次，针对痛点问题提出一种新型的解析方法，提升解析效果。本文主要贡献如下。

1) 针对当前日志解析方法解析结果不稳定、解析准确率有待提高的问题，本文提出一种基于启发式规则的流式在线日志解析方法——启发式正则树（HRTree, heuristic regex tree）。该方法以Drain的流式树形解析结构为基础，基于启发式规则对日志中的部分token进行有效拆分，实现固定文本和参数的分离从而改善解析结果的稳定性和准确率。

2) 针对参数识别不准确和过拟合的问题，本文提出优化解析结构树匹配搜索和节点更新方法。该方法的启发式规则能够解决特定规则下日志参数识别不准确的问题，避免参数过拟合。同时优化了解析结构树匹配搜索和节点更新的细节，进一步提高了解析文本的准确率。

3) 大量实验证明本文提出的HRTree方法在解决日志解析不稳定和不准确率问题方面取得了显著的成果。本文进行了广泛的实验验证，表明HRTree方法在不同的系统日志上均展现了较高的解析效果，不仅达到了解析日志事件的分类准确，而且实现了参数内容识别准确，证明了本文提出的方法在日志解析领域的可行性和实用性。

1　相关工作

基于日志的异常检测已经被广泛应用于系统监控，本节调研了日志异常检测的整个流程，对各环节涉及的相关技术进行了阐述和分析，特别是对其中的日志解析算法进行了全面的综述。

1.1　日志异常检测框架

通过分析近年来异常检测相关领域的研究现状，日志异常检测可抽象构建为如图1所示的日志异常检测流程框架，包括以下4个步骤：日志收集与预处理、日志解析、特征提取和异常检测[10-11]。

日志收集与预处理。系统在运行过程中会产生记录系统运行状态详细信息的原始日志数据，每条日志数据通常都包含日志产生的时间、当前进程号、日志等级和关键事件描述等。在某些情况下，根据模型特征提取的需求，还需要对日志数据进行过滤和清洗，以删除一些重复和无用的干扰数据，只保留准确反映系统状态的有效数据。例如，Di等[12]采用基于加权相似度的时空消息过滤器，对可靠性、可用性和可维性（RAS）日志进行了重复过滤，避免了数据噪声对后续分析和特征提取的干扰，从而提高检测的准确率。

日志解析。日志信息通常为非结构化的文本信息条目，含有时间戳、进程号和描述语句等丰富的多重信息内容。时间戳和进程号等往往具有固定的格式，可以通过正则化匹配进行解析。而阐述日志行为的描述语句，通常为程序员编写的陈述性静态文本和相关动态参数信息的组合，因行为类型和业务模块的不同，陈述性静态文本语言风格也不尽相

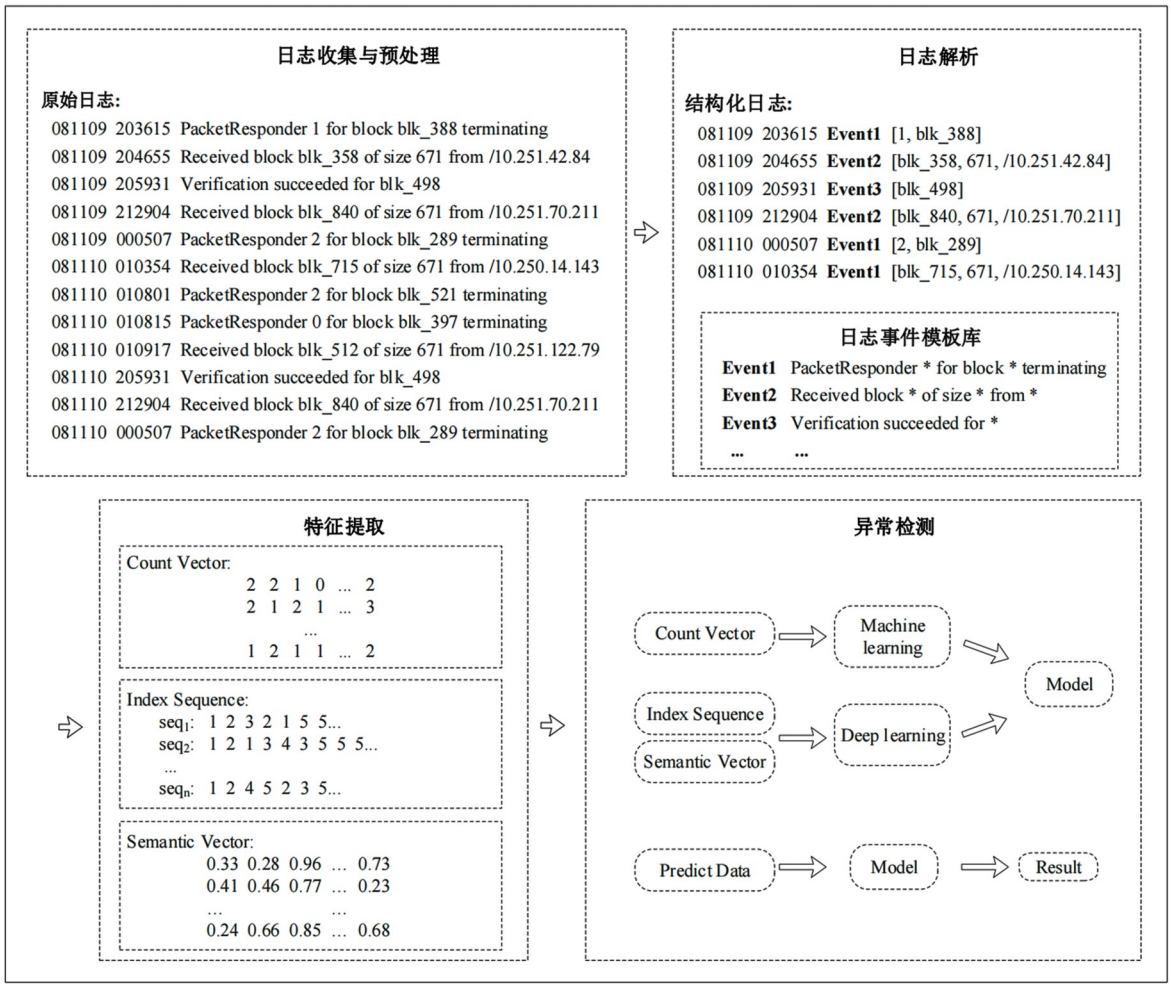


图1　日志异常检测流程框架

同，多数为非结构化的自由文本形式。特别是静态文本与动态参数信息的组合，会造成日志文本的数量类型在全局维度上的爆炸性增长。日志解析任务则通过频繁模式、聚类和启发式等数据挖掘方法，将非结构化日志文本转换为结构化数据，具体为从日志消息中分离参数信息，提取日志行为事件模板和参数。通过分离参数，使用日志行为事件模板可以表示系统的有限类行为，同时准确获取该行为特定参数信息，为后续的检测提供特征。

特征提取。经过日志解析后，时间戳、进程号等标签信息被结构化，日志信息被提取为含有特定行为含义的行为事件模板和特定参数的组合。根据系统日志的具体情况，可以利用时间戳和进程号等信息进行日志数据的分组采样。例如，利用时间戳信息进行时间维度的数据分组，利用进程号等标签信息进行会话维度的分组等。分组完成后可进行日志描述文本的特征化数字编码，通过统计日志序列中每一类事件发生的次数可以生成日志事件计数向量特征，根据日志事件产生的顺序关系描述系统行为执行路径构建日志索引序列特征，通过解析日志语义构建日志语义信息向量特征。

异常检测。异常检测主要是在提取的日志特征上进行模式构建，通过阈值或者参数设置，对已知的训练数据进行模式学习，生成检测模型，从而检测未知新数据并判定系统行为是否正常。主要的检测模型有基于传统机器学习的有监督模型，如决策树和逻辑回归等，以及无监督的主成分分析、聚类、不变量挖掘等；也有基于深度学习的长短时记忆（LSTM, long short term memory）网络模型，以及结合卷积神经网络（CNN, convolutional neural network）和生成对抗网络（GAN, generative adversarial network）等多网络结构的复杂模型。检测异常类型包括行为事件数量关系异常、行为序列顺序关系异常和参数值异常等。

1.2 日志结构化解析方法

日志结构化解析的目的是将非结构化的日志文本转化为包含静态文本和动态参数的事件模板，用有限状态的行为事件模板描述系统行为类型。传统的方法依赖于手工正则表达式提取，这种方法耗时且无法适应不同日志系统。一些研究尝试从源代码中提取日志事件模板[13-14]，这种方法准确率高，但是访问源代码并不总是可行。因此，近年来的研究主要关注对已生成的日志文本进行自动化解析，使用数据挖掘算法对日志信息进行自动化解析，减少人为参与的同时提高解析的准确率。本文将解析方法总结为以下几类：基于频繁模式挖掘的解析方法、基于聚类的解析方法、基于最长公共子序列等算法的解析方法和基于启发式的解析方法。

1.2.1 基于频繁模式挖掘的解析方法

频繁模式是指数据集中常见的一组项目。日志事件模板可以看作由常量标记组成的频繁模式。因此，频繁模式挖掘是一种简单的日志解析方法。

简单日志聚类技术（SLCT, simple logfile clustering tool）[15]是最早的基于频繁模式的日志聚类算法，它通过设定支持阈值来挖掘高频单词，构建频繁项集；然后根据频繁项集中的单词和位置，选出候选簇；最后从候选簇中选择支持值大于阈值的簇作为日志事件模板。SLCT的缺点是会忽略一些低频事件，因为它们的支持值小于阈值。日志特征分析（LFA, log feature analysis）方法[16]对每行日志都进行频率筛选，将相似的频率词抽象为事件类型，因此LFA可以解析所有的日志事件模式。SLCT和LFA都对单词的位置进行了编码，所以对位置信息敏感。如果日志中的参数由多个单词组成，那么一些频繁单词的位置会发生偏移，导致同一类型的日志事件被分成多个簇。日志聚类（LogCluster, log clustering）方法[17]对单词位置不敏感，从日志事件中挖掘线性模式，可以适应单词位置的变化。

基于日志的异常检测（Logram, log-based anomaly detection）[18]使用n-gram对日志进行采样，利用字典存储日志中的n-gram频率，认为高频n-gram是事件模板，低频n-gram是参数，实现高效的日志解析。Logram不同于前3种方法的是可以实现日志的在线解析，不需要遍历所有日志数据。

1.2.2 基于聚类的解析方法

基于聚类的解析方法是利用传统的聚类算法，根据日志信息的某种特征，提取相同类型的日志事件模板。

日志关键词提取（LKE, log key extraction）方法[19]通过删除参数值和使用加权编辑距离度量日志消息的相似性，将相似的日志聚为一类。该方法考虑了日志消息中的参数值，并根据编辑距离进行相似性度量，从而实现对日志事件的有效聚类。

日志签名（LogSig, log signature）方法[20]将日志消息转化为单词对的二元组集合，通过随机分组和编码计算每条日志的签名值，并根据签名值和组的相似度将日志划入最可能的组。该方法通过签名值的计算和相似度的比较，实现了对日志事件模板的准确提取。

日志挖掘（LogMine, log mining）方法[21]先对日志数据进行标记化和类型检测，然后通过计算日志间单词的相似度距离进行快速聚类，生成模式集合作为叶级。该方法快速聚类和模式集合的生成，从而实现了对日志事件模板的高效提取。

可扩展的分层日志解析（SHISO, scalable hierarchical log parsing）方法[22]通过统计单词中的字符类型和计算单词的向量信息，再通过欧氏距离进行聚类。该方法通过字符类型和向量信息进行聚类，从而实现了对日志事件模板的精确提取。

基于长度的匹配（LenMa, length-based matching）方法[23]通过统计日志语句中每个单词的长度和单词数量构建字长向量和字数向量，以此特征进行聚类。该方法利用单词长度和数量进行聚类，实现了对日志事件模板的准确识别。

1.2.3 基于最长公共子序列等算法的解析方法

基于序列的日志行模式提取（SPELL, sequence-based pattern extraction for log line）方法[24]是基于最长公共子序列来解析日志的方法，将日志拆分为单词序列，计算公共子序列来构建日志事件模式。该方法使用流式在线解析，每条日志输入后，计算该日志和已存在事件的最长公共子序列。从已知事件中，选择最大的作为候选模式，计算公共子序列与自身长度的比值作为相似度。与阈值比较，超过阈值的日志划分为同一类，用通配符对不同部分模糊化，更新事件。不超过阈值的日志定义为新模式。为减少计算次数，SPELL用前缀树结构存储已

知事件。

日志解析要尽可能多地将参数模糊化，又要尽可能多地保存事件特异性，这是一个多目标优化问题。Messaoudi 等[25]基于 NSGA-II[26]提出了基于模型的日志文件识别（MoLFI, model-based log file identification）方法，这是一种基于多目标遗传算法和权衡分析的搜索方法。

日志解析（LogParse, log parsing）方法[27]将日志模板提取转化为单词分类问题。先从历史日志中提取模板，对模板单词和可变词制定标签，用单词的字符级计数向量特征，用支持向量机算法构建分类器。一方面匹配现有模板，另一方面通过分类器区分未知模板单词和变量单词，学习新模板，并实时更新到前缀树中。

1.2.4　基于启发式的解析方法

基于启发式的解析方法是利用日志信息的一些启发式特征进行解析的方法。启发式特征是指一些经验性的但通常有效的规则或条件，用于简化问题的求解过程或提高效率。例如，日志消息的长度、单词的位置、字符的类型等都可以作为启发式特征，用于区分不同的日志事件模式。基于启发式的日志解析方法不需要访问源代码或历史日志，也不需要复杂的数据挖掘算法，只需要根据启发式特征进行简单的划分或聚类，就可以实现日志的自动化解析。

自动化事件日志（AEL, automated event log）方法[28]是一种识别日志消息中参数的启发式方法。日志语句中存在赋值对“word = value”和短语“is [are|was|were] value”，将 value 内容用通配符代替，定义为参数。然后统计每条日志中单词和通配符的数量，将数量相同的日志分为一组。之后对同一组内的日志进行调整合并。

迭代分区日志挖掘（IPLoM, iterative partitioning log mining）方法[29]是基于启发式的迭代分区聚类算法，基于以下 2 个启发式特征：特征 1，相同行为描述的日志消息，往往具有相同的日志长度；特征 2，基于日志事件的位置信息，不同单词数量最少的位置往往是静态部分。IPLoM 先基于特征 1，按日志长度划分集群；然后基于特征 2，统计每个位置的不同单词数量，以最少的位置为分割点，进一步分割每个分区；之后根据日志不同位置单词的映射关系，再次分区；最后对每个集群进行描述，标记位置唯一的为常数，含多个单词的为变量，用通配符代替。

Drain 方法[30]是一种基于固定深度解析树的在线日志解析方法。同样基于以上 2 个启发式特征，先用正则表达式代替参数值，如 IP 地址和统一资源定位符（URL）等，减少干扰参数。然后按日志长度 length 建立树节点，解析树结构[30]个如图 2 所示。之后在每个长度节点下，根据单词依次建立子节点。设定深度阈值 max_depth，深度 depth 超过阈值后不再创建，叶子节点为日志事件类型的集合。新日志先计算长度，再根据单词找到叶子节点，不存在分支时创建新分支。到达叶子节点时，通过计算相似度，寻找最相似的日志事件，通过阈值比较，不超过阈值的日志归为一类，否则定义为新模式。对不同部分用通配符模糊化，更新事件。

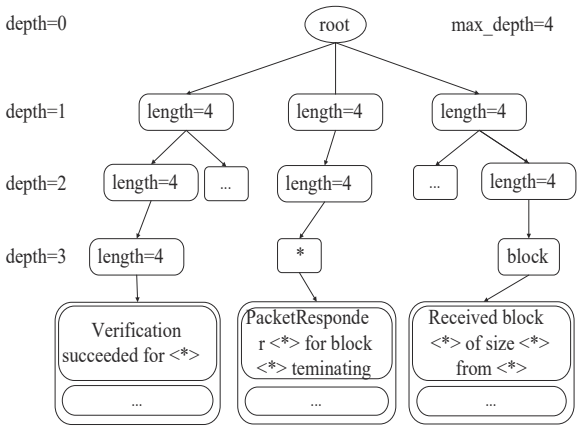


图 2　Drain 方法中解析树结构

1.3　日志解析难点分析

日志作为存在于任何系统的通用资源，记录了系统某一时刻发生的特定事件，包含时间、端口号、IP 地址、日志等级和描述信息等。HDFS 数据集日志解析示例如图 3 所示，将含有日期、时间、进程号、日志等级、组件和文本内容的原始日志，解析为结构化信息文本。其中，日期、时间、进程号等前置信息往往具有统一格式，可通过正则化提取。

日志解析的难点在于日志事件自由描述文本的结构化解析。自由文本通常由系统内核中源代码的一段打印语句生成，通常由程序员编写的一段描述性文本。其中，少部分为纯静态文本，大部分由静态文本和动态参数组合而成，例如

printf("exception syndrome register: %s", reg)

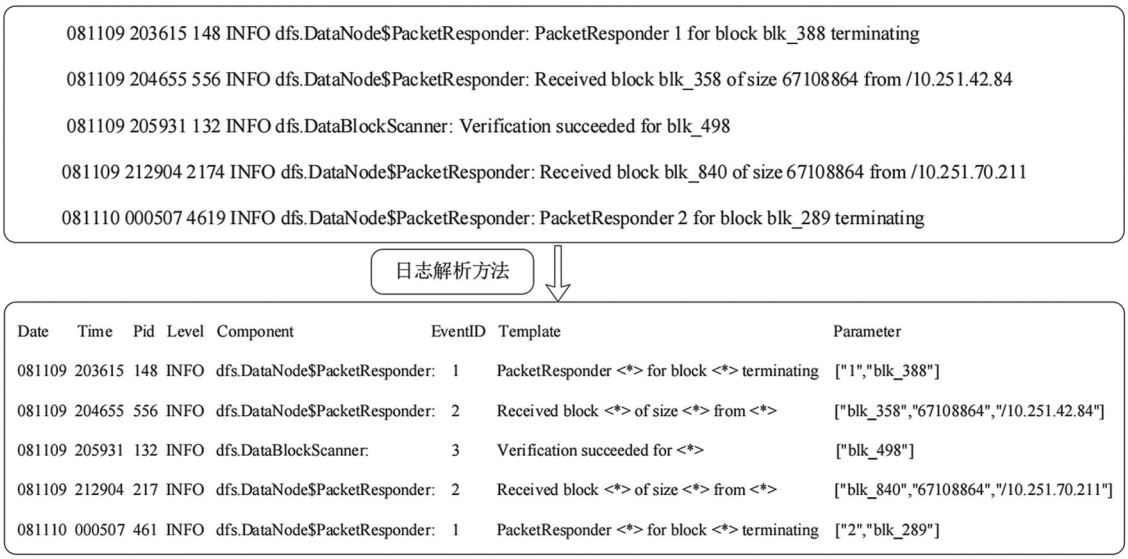


图3 HDFS数据集日志解析示例

运行过程中可能打印产生如下真实语句

exception syndrome register: 0x00800000

其中，"exception syndrome register："为静态事件文本，描述了当前日志记录的事件，即寄存器发生异常；而"0x00800000"为具体的寄存器标志，以实时动态参数的方式添加到静态语句中。日志解析的目的就是准确识别原始的静态文本和参数信息，实现描述事件和具体参数的分离，即解析成事件模板Template: "exception syndrome register: <*>"和参数Parameter: "0x00800000"。该任务即目前所有日志解析方法的研究重点。

将日志事件描述文本进行参数分离的主要原因如下。系统环境中日志事件行为种类通常有限，它们通过固定的描述信息阐述了系统运行过程中发生的具体事件。但当结合参数信息时，就会造成日志事件急剧增加，给后续的日志异常检测任务造成特征维度的诅咒。而通过分离参数提取日志事件模板表示日志事件的具体行为类型，虽然缺少了参数信息，但是不影响该文本对整体事件行为含义的理解。综上所述，日志解析进行参数分离是非常有必要的。

传统的手工编写正则表达式进行日志解析效率低下，难以应对现代大型系统的规模增长和频繁更新。虽然基于源代码分析准确率高，但是源代码往往难以获取且需要领域专家的介入，无法实现不同系统的迁移使用。目前的日志结构化解析主要集中于数据挖掘算法，高效准确的解析方法是本文的一个研究重点。

虽然目前存在多种日志解析方法，但是它们在不同系统日志上的解析效果参差不齐。部分解析方法只能实现对特定日志的有效解析，而在处理复杂系统日志时效果较差。面对不同系统条件，稳定可靠的有效解析方法依然是日志解析的一大挑战。已有解析方法主要存在以下几点问题。

全覆盖问题。有效的日志解析应尽可能覆盖所有日志行为模式，以避免模式遗漏。例如，现有的SLCT和LogCluster解析算法，由于频率支持阈值的设置，可能会遗漏低频行为模式。而这种遗漏对于之后的日志检测并不友好，因为某些具有报警信息的低频日志行为在系统中往往不常出现，但一旦出现就表现为系统的异常报警，因此遗漏该类低频行为会显著影响异常检测算法识别异常信息。

离线/在线问题。根据是否能随时产生解析结果，可以将解析方法分为在线和离线2种模式。离线模式需要在解析时对所有日志数据进行读入分析，即遍历扫描之后才能进行日志事件模板的划分提取。日志事件模板一般固定后不可扩展更新，无法对发生偏移的日志数据进行有效解析。同时离线解析方法通常需要将待处理数据全部读入内存，对于大规模日志数据处理是一种挑战。在线模式则不用提前获取全部的日志数据，可随时读入数据并输出解析结果，减轻内存压力。对于新出现的日志模板可以合并到已知相似事件或者创建新事件模板，因此更适用于频繁更新的大型系统。

准确率问题。高效准确的日志解析是异常检测任务的基础，不同解析方法必然在解析准确率上存在一定的噪声和误差。尽管有些方法已达到95%以上的解析准确率，但是就像He等[31]所发现的，日志挖掘可能对某些关键事件敏感。在关键日志上4%的解析错误可能导致异常检测性能的极大下降。通过分析现有日志解析方法，日志解析的准确率表现在以下几点。1) 稳健性问题，例如，Drain和SPELL目前在Hadoop分布式文件系统（HDFS, Hadoop distributed file system）和Apache等稳定数据集上可达95%以上的准确率，但在Linux和Proxifier等复杂数据集上准确率下降至50%左右，现有各类解析方法在不同类型系统日志上不稳定。2) 参数过拟合，部分方法虽然对日志解析的分类准确，但是实际解析的结果却不尽如人意，比如Drain方法容易出现参数的过拟合，将不是参数的单词识别为参数，影响日志事件表达。特别是在提取语义特征和参数特征的检测模型中，参数提取不准确对检测模型效果的影响极大。

解析效率问题。效率是日志解析方法的一个重要指标，特别是大数据时代下信息量爆炸增长，某些系统一天的数据量能达到千万条级别。无法高效解析会导致异常检测服务的滞后性，严重影响异常检测任务的效率。目前部分解析方法（例如LKE、MoLFI）无法做到对大规模日志数据的解析，并且部分解析方法解析速度过慢，严重影响日志解析的流程，所以解析效率是异常检测任务中解析方法选择时应着重关注的问题。

2　所提流式日志解析方法介绍

2.1　解析方法设计

为了解决日志解析的挑战，本文从2个方面出发。一方面，从各类解析方法的原理出发，综合分析了现有的各类解析方法的优缺点，发现了它们在不同系统日志上的解析效果不稳定、参数识别不准确、解析效率低等问题，表1展示了不同解析方法的特点。另一方面，从数据层面入手，总结了解析难度较大的几个数据集的特征，剖析了解析方法面临的数据挑战。

基于以上分析，本文提出了一种基于启发式规则的解析方法HRTree，该方法以目前较高效的解析结构树方法Drain为基础，在其解析原理上进行了创新改进。HRTree方法利用日志数据的一些启发式规则特征，如日志长度、单词位置、字符类型等，来辅助日志模板的提取和参数的分离。启发式规则是一种基于经验或直觉的规则，用于解决一些复杂或不确定的问题，在日志解析的问题中，启发式规则是一种有效的方法，可以利用日志数据的一些特征或规律，高效实现日志模板的提取和参数的分离。本文提出的HRTree方法基于以下几条启发式规则：1) 同类日志描述信息往往具有相同的日志长度；2) 日志数据中常含有特定格式的参数内容，例如IP地址、数字、URL等；3) 部分参数信息常跟在特定格式文本后面，例如等号、冒号和括号内。这些启发式规则是通过对不同系统日志的观察和分析得到的，HRTree基于以上规则内容进行启发式的特定数据处理，同样采用树形结构存储解析结果，实现流式在线高效解析。下面首先介绍HRTree的解析流程，同时附上主要步骤的伪代码如算法1~算法3所示；最后对相关改进和创新内容进行详细说明。

表1　不同解析方法的特点

方法	时间	覆盖	模式	技术
SLCT	2003年	部分	离线	频繁模式挖掘
AEL	2008年	全部	离线	启发式
LKE	2009年	全部	离线	聚类
LFA	2010年	全部	离线	频繁模式挖掘
LogSig	2011年	全部	离线	聚类
IPLoM	2012年	全部	离线	启发式 + 迭代分区
SHISO	2013年	全部	在线	聚类
LogCluster	2015年	部分	离线	频繁模式挖掘
LenMa	2016年	全部	在线	聚类
LogMine	2016年	全部	离线	聚类
SPELL	2016年	全部	在线	最长公共子序列
Drain	2017年	全部	在线	启发式 + 解析树
MoLFI	2018年	全部	离线	进化算法
LogParse	2020年	全部	在线	解析树 + 词分类
Logram	2020年	全部	在线	频繁模式挖掘
HRTree	2023年	全部	在线	启发式 + 正则式 + 解析树

HRTree解析流程如下。

步骤1　基于领域知识进行预处理清洗；通过自定义正则表达式，对部分常用变量进行清洗，例如IP地址、HDFS中的块地址blk_id等。这种正则表达式的定义只需要对日志数据进行总览，制定简单的正则匹配规则即可。

算法 1 HRTree 主体算法

输入 所有日志集合 log，相似度阈值 st，最大深度 depth，简单正则表达式 reg

输出 树形存储的日志结构 Root

1) 初始化树形存储结构 Root
2) function HRtree(log,st)
3) for i = 1to length(log) do
4) log_i ← regex(reg) //基于正则进行常用变量清洗
5) log_i ← preprocess(log_i) //对日志数据进行符号替换，拆分成 token 列表
6) if treeSeacrch(log_i,Root,st,depth) then
7) MatchLog←搜索到的相似模板日志
8) for j = 1 to length(log_i)do
9) if $log_i[j]$ ≠ MatchLog[j]then //合并日志模板时，若出现相同位置 token 值不同，则认定其为变量，使用通配符" "替换
10) MatchLog[j] = " < * > "
11) end if
12) end for
13) else
14) updateTree(log_i, Root, depth) //如果未搜索到符合相似度日志模板，则将当前日志信息更新到 Root 树形结构中
15) end if
16) end for
17) return Root

步骤 2 基于启发式规则进行部分 token 拆分，如 AEL 方法中发现日志数据中常含有 "word = value""word：value" 和 "(word，value)"，其中，word 是有具体含义的实词，而 value 是真实的参数内容。由于它们没有空格拆分，在 Drain 中被识别为一个 token，无法准确分离 value 参数部分，导致参数过度解析问题。在拆分日志语句时，加入"=""：""（""）""，"等符号的判断，在其符号两端加入空格符分隔。由于在分词之后还需要合并，因此为了区分原有空格和新加入空格，需要先将原语句中空格使用特殊符号替换，分词之后再转换回去。Drain 解析方法和 HRTree 解析方法中 token 拆分示例分别如图 4 和图 5 所示。

步骤 3 搜寻解析树第一层；根据启发式规则，相同日志种类的日志消息往往含相同的长度，统计当前日志数据的 token 数量（此处不再统计步骤 2 中使用的各类分割符号）。搜寻根节点下第一层子节点，若搜寻到当前长度的子节点，则更新到当前长度子节点上，转至步骤 4；若无该长度节点，则返回空值，转至步骤 7，更新创建新节点。

算法 2 HRTree 搜索算法

输入 当前搜索日志 log_i，解析结构树 Root，相似度阈值 st，最大深度 depth

输出 若搜索到符合相似度的叶节点，则返回该日志模板信息；否则返回 None

1) function treeSearch（log_i, Root, st, depth）
2) curdepth = 1
3) fatherNode ← Root[length(log_i)] //搜寻第一层长度节点
4) if fatherNode then
5) curdepth ← curdepth + 1// 依次搜寻 token 路径；
6) while log_i[curdepth] ∈ fatherNode and curdepth< depth do

```
// 原始日志;
ciod: Received signal 15,code=0,errno=0,address=Ox000001b0

// 使用空格进行拆分;生成含token 的列表;    "
["ciod":, "Received", "signal","15,", " 有7个code=0,", " errno=0,", "address=Ox000001b0"]

// 使用Drain解析，通过比较，部分内容被模糊化成通配符，其中包括code/errno/address等内容;
["ciod:", "Received", "signal","<*>","<*>","<*>","<*>"]

// 将原始日志进行拼接，得到日志模板，  出现过度模糊化的问题
coid: Received signal<*><*><*><*>
```

图 4 Drain 解析方法中 token 拆分示例

```
// 原始日志；
ciod: Received signal 15,code=0,errno=0, address=Ox000001b0

// 先使 '$' 替换空格符；
ciod $Received$signal$15,$code=0,$errno=0,$address=Ox000001b0

// 在'=' '$' ',' 两侧加入空格；
ciod : $ Received $ signal $ 15 , $ code=0 , $ errno=0 , $ address =Ox000001b0

// 再次使用空格符进行拆分，生成含有23个token 的列表，其中6个原始空格 '$'；
["ciod",":","$","Received","$","signal","$","15",",","$","code","_","0",".","$","errno","=","0",",","$","address","=","Ox000001b0"]

// 使用HRTree解析，通过与其他日志比较，参数部分被准确模糊化成通配符；
["ciod",":","$","Received","S","' signal","$","<*>",",","$","code"_ _"*>",".","$","""errno","=","<*>",",","$""","address","""="","<>"]

// 将原始日志进行拼接；
ciod:$Received$signal<*>,$code=<*>,$errno=<*>,$address=<*>

// 使用空格替换'$', 得到解析结果；
ciod: Received signal<*>, code=<*>,errno=<*>,address=<*>
```

图 5　HRTree 解析方法中 token 拆分示例

```
7)        fatherNode←fatherNode [log;[cur-
          depth ]]
8)        curdepth ← curdepth + 1// 在叶子节点
          列表中搜寻相似度最高的日志模板；
9)      end while
10)       for logTemple in fatherNode.logCluste
          do
11)         MaxSim ← simSeq(log_i,logTemple)
            // 进行相似度阈值比较;
12)       end for
13)     if MaxSim ⩾ st then
14)       return logTemple
15)     else
16)       return None
17)     end if
18)   else
19)     return None
20)   end if
```

步骤 4　依次搜寻 token 路径，直到搜寻到叶子节点；在完成步骤 3 中长度节点的搜寻后，对 token 内容进行搜寻（跳过步骤 2 的各类分割符号 token）。若当前深度的叶子节点含有当前 token 内容，则更新到新节点中，循环步骤 4，直到搜寻到叶子节点，转至步骤 5。若无 token 节点匹配成功，则返回无匹配结果，转至步骤 7。

步骤 5　通过相似度搜寻匹配日志模板；到达此步说明已经搜寻到叶子节点处，该叶子节点为一组相同路径信息的日志组合；在该组日志中寻找和当前日志信息最相似的一组。进行阈值判断，若最相似日志组相似度大于当前设定阈值，则返回当前找到的最相似组日志信息内容，并转至步骤 6；否则返回未搜寻到结果，转至步骤 7。本文定义相似度的计算方式如下。

$$\mathrm{simSeq}=\frac{\sum_{i=1}^{m}\mathrm{equ}\left(\mathrm{seq}_1(i),\mathrm{seq}_2(i)\right)}{m-\mathrm{count}\left(\mathrm{symb}\right)} \tag{1}$$

$$\mathrm{equ}\left(t_1,t_2\right)=\begin{cases}1, & t_1=t_2 \text{ 且 } t_1\notin \mathrm{symb}\\ 0, & \text{其他}\end{cases} \tag{2}$$

其中，$\mathrm{seq}(i)$ 为当前序列的第 i 个 token，symb 为字符 token 列表 t_i 日志消息序列中的第 i 个 token，M 为日志序列中的 token 数量。

步骤 6　若搜寻到当前结果，则对当前结果和当前日志消息进行比对，若相同位置的 token 相同则认定该 token 为常量，否则认定其为变量信息，对其进行通配符<*>的模糊化处理。

算法 3　HRTree 更新算法

输入　当前搜索日志 $\log_i$，解析结构树 Root，相似度阈值 st，最大深度 depth

输出　返回更新后的树形存储日志结构 Root

```
1) function treeSearch(log_i, Root, max _ depth)
2)     curdepth = 1
3)     seqLen ← length(log_i)
4) if seqLen ∈ Root then // 如果根节点下存在当
   前长度子节点，则更新到该节点下；
5)   fatherNode ← Root [ length(log_i) ]
6) else //如果根节点下未存在当前长度子节点，
```

则创建该长度子节点，并更新到该节点下；

7) Root [length($\log_i$)]←Node (length($\log_i$))

8) fatherNode ← Root [length ($\log_i$)]

9) end if

10)for token in $\log_i$ do

11) if token ∈ fatherNodethen // 如果token 为当前父节点的子节点，则更新到该节点下；

12) fatherNode = fatherNode [token]

13) else //如果 token 不是当前父节点的子节点，则创建该长度子节点，并更新到该节点下；

14) fatherNode [token]←Node (token)

15) fatherNode = fatherNode [token]

16) end if

17) curdepth ← curdepth + 1

18) if curdepth > depth then

19) break

20) end if

21) end for

22) fatherNode .logClustaddlog$_i$ // 将当前日志所有信息更新到该节点上；

步骤 7 创建更新解析结构树节点；若未找到当前日志相似叶子节点，则在原解析结构树中创建新的节点；若存在当前日志 token 长度节点，则更新到当前长度节点，若不存在则创建该长度节点；创建 token 节点，若存在于当前节点的叶子节点列表中，则更新到下一深度，否则创建该层节点。在创建过程中若超过最大深度节点，则将最大深度节点定义为叶子节点，不再继续创建。将当前日志信息更新到叶子节点中。由此创建了一条由根节点到叶子节点的 token 路径，为日志序列的前缀信息。

在测试中，本文发现可能出现日志长度小于树的最大深度的日志信息，Drain 算法未对该情况进行讨论，HRTree 对该情况日志 token 提前创建叶子节点，打破了 Drain 固定深度的规则，仅对最大深度进行控制，实现在较小深度也可创建叶子节点。同时在创建 token 节点的过程中，对于第一步正则表达式解析的常用变量，Drain 方法虽然也对其进行了识别，但是在模板创建过程中并未更新到日志模板信息中；本文方法将其实时更新到日志模板库中。

2.2 算法复杂度分析

结合算法流程步骤，可以推理出 HRTree 方法与 Drain 方法具有相同的线性时间复杂度。具体地，HRTree 时间复杂度为 $O\left((d+cm)n\right)$，解析方法以流式依次对每条日志信息进行解析。其中，n 为待解析日志信息的数量；d 为解析树的最大深度，在搜索和更新过程中，最多进行 d 次比较，寻找到叶子节点；c 为当前叶子节点的候选日志组数量；m 为日志信息含有的 token 数量，当寻找最大相似度的日志组时，进行 cm 次比较。d 显然为常数，m 和 c 也可以视为一个波动的常量，因为日志信息的 token 数量和每一个叶子节点的日志组数量十分有限，即使是它们的乘积，其值也远远小于日志信息的数据量 n。因此，HRTree 的时间复杂度近似为 $O(n)$。

2.3 算法特点分析

结合以上内容，HRTree 是一种基于 Drain 的启发式规则改进解析方法，具有 Drain 全覆盖、流式在线解析的特点，同时克服了 Drain 存在的参数过度解析、部分叶子节点无法创建和常用变量无法及时更新等问题，能够实现更高效的解析效果。HRTree 特点总结如下。

1) 采用树形解析结构方式，通过日志长度计算，缩短日志搜索的范围，提高解析速度。

2) 采用最大深度和最大叶子节点数来控制日志结构树的规模，避免爆炸性增长。

3) 创新提出使用启发式方法对部分 token 进行拆分，将 Drain 以空格为标志的 token 拆分方式转换为规则化的启发式拆分，对组合 token 进行有效拆分以解决日志参数识别不准确问题。

4) 优化相似度计算方式，解决 Drain 方法中相似度要求较低，容易过拟合的问题。

5) 解决了日志更新步骤中，序列长度小于最大深度的节点无法创建为叶子节点问题。

6) 解决了明显参数类节点，将其模糊化处理实时更新到解析树的问题。

3 实验对比评估

本节从日志解析方法的典型指标、准确率指标和运行效率指标多个方面，对 HRTree 与其他解析方法在不同数据集上进行全面的测试，评估 HRTree 的解析效果和解析效率。

3.1　实验环境

实验环境为一台 64 位 Intel Core i9-12900KF 3.20 GHz 16 核 24 线程 CPU， 64 GB 内存，NVIDIA GeForce RTX3090Ti GPU 的 Ubuntu 20.4 系统服务器。本文所有实验均在该服务器上进行，为避免其他应用程序的占用产生干扰，本文所展示的每组数据均为三次及以上测试结果的平均值。

3.2　基线方法和模型介绍

为全面评估本文提出的基于启发式规则的流式日志解析方法 HRTree 的性能，本节将介绍选定的几种作为基线的日志解析方法。这些方法覆盖了频繁模式挖掘、聚类分析、最长公共子序列算法以及启发式解析策略等不同技术路线，旨在通过多维度对比展现 HRTree 的优势与创新性。

在频繁模式挖掘领域，LFA[16]通过辨识高频词与低频词来提取日志事件模式，尽管适应性强但在处理大规模数据时效率受限。与之相辅的 LogCluster[17]，采用聚类算法优化了对复杂日志格式的解析能力，虽然在大数据场景下成本较高，但准确度与适用性较为出色。

聚类分析方法包括 LKE[19]、LogSig[20]、LogMine[21]、SHISO[22]及 LenMa[23]等。这些方法通过计算日志条目之间的相似度来聚类，从而提取模板。例如，LKE[19]依据加权编辑距离和经验规则进行聚类，LogSig[20]将日志消息转化为二元组集合以提取模板，LogMine[21]通过层次聚类分析单词相似度。这些方法虽然在模板提取上各有优势，但在处理大规模数据集时，效率仍是一个挑战。

基于最长公共子序列的方法，如 SPELL[24]与 MoLFI[25]通过分析日志条目间的公共子序列来构建事件模型，这类方法在模糊处理和模板优化上展现了高效与精确的平衡。

启发式解析方法，如 AEL[28]、IPLoM[29]和 Drain[30]，特别强调利用日志数据的特定特征进行有效解析。例如，AEL[28]通过启发式分析与消息匿名化实现日志参数识别与优化，IPLoM[29]通过迭代分区聚类与位置信息分析挖掘日志模式，Drain[30]通过构建解析树和启发式规则快速识别相似事件，显示了在线日志解析中的高效性。

通过对比这些基线方法全面展示 HRTree 方法在日志解析领域的技术创新及其相较于现有方法的优越性。

3.3　实验数据

真实的日志数据具有一定的机密性，能公开获取的资源十分有限。本文调研发现 Zhu 等[32]公开了一个日志数据仓库 loghub，其中包含 16 种不同系统的日志数据，涵盖了分布式系统、超级计算机、操作系统、服务器应用程序和独立软件等。

每个系统的日志规模均不相同，有的日志条目达到了千万级别，日志种类达到了上百种。为方便测试，本节实验使用 Zhu 等[32]提供的一个小型标注数据集，该数据集从各数据集中随机抽取了 2 000 条日志消息，并手动标注了日志事件模板。虽然 2 000 条日志数据为随机采样，但是保留了全体数据的关键属性，如日志事件的冗余和事件的多样性。

3.4　实验评估

3.4.1　典型指标评估

首先使用分类算法的典型评价指标精确率（Precision）、召回率（Recall）和 F1-measure 指数来评估 HRTree 的解析性能，之前的多个研究中也使用了这个评估度量[30-31]，该组指标主要评估日志消息在日志行为种类上分类是否准确，测试解析日志行为事件分类的效果。指标定义如下

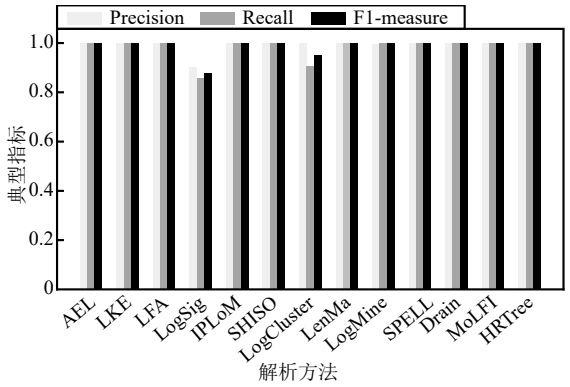
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

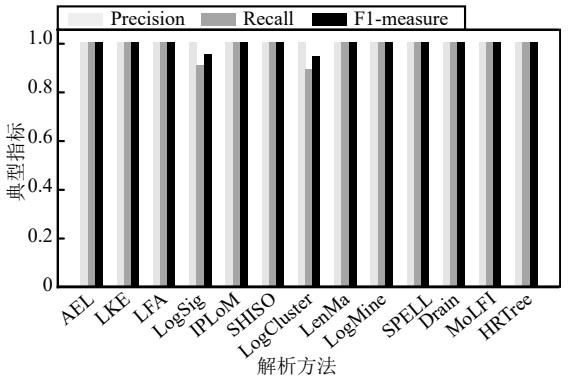
$$\text{F1-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

其中，真阳性（TP, true positive）表示将 2 个具有相同日志事件的日志消息分配到同一个日志组；假阳性（FP, false positive）表示将 2 个具有不同日志事件的日志消息分配到同一个日志组；假阴性（FN, false negative）表示将 2 个具有相同日志事件的日志消息分配给不同的日志组。

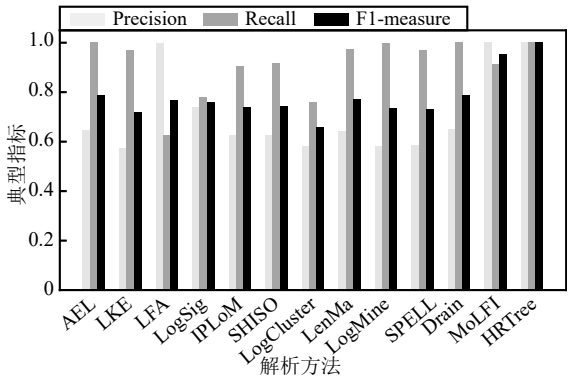
不同解析方法在不同数据集上典型指标评估结果如图 6 和图 7 所示，可以看出 HRTree 在所有数据集上均取得了较好的典型指标表现。图 6(a)和图 6(b)展示了所有解析方法在 HDFS 和 Apache 数据集上的解析结果，除 LogSig 和 LogCluster 这 2 种检测方法效果略微较低外，其余解析方法均对其在典型指标上具有较高的解析评价。分析原因为该类数据集较稳定，日志种类数量较少，解析难度不高，所以多数解析方法在该类数据集上均能取得较好的典型指标结果。



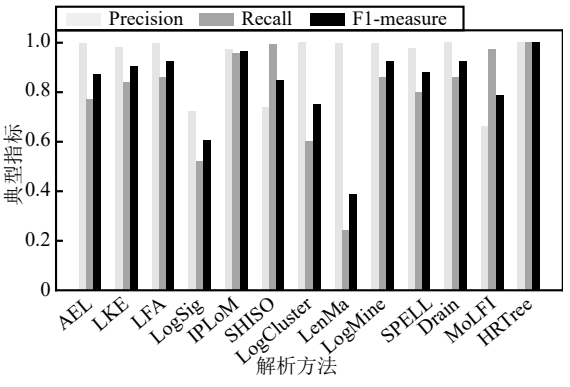
(a) HDFS



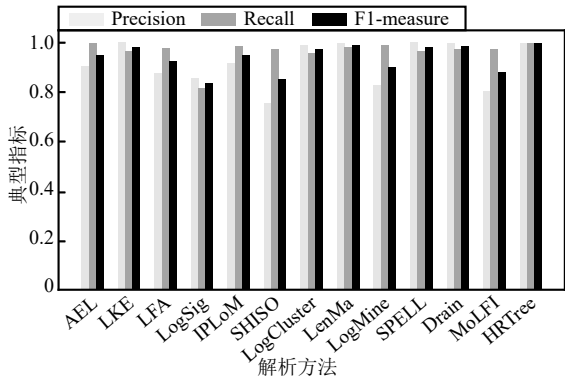
(b) Apach



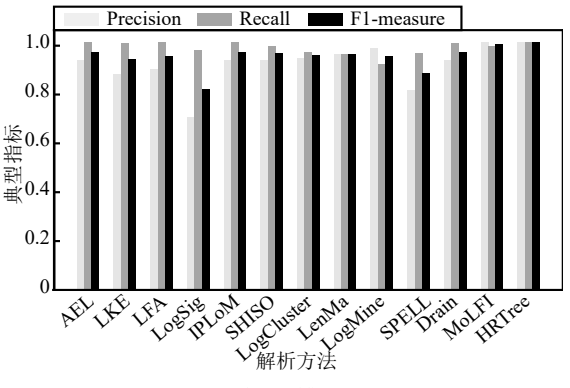
(c) Linux



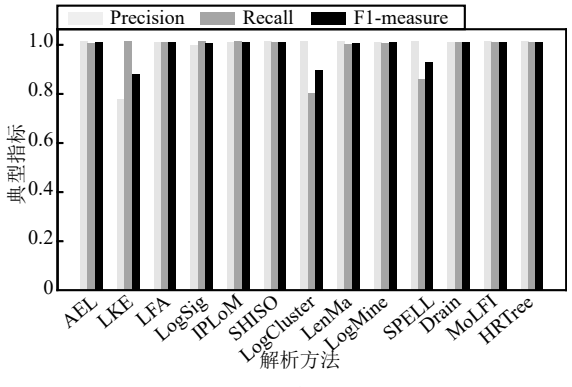
(d) HealthApp



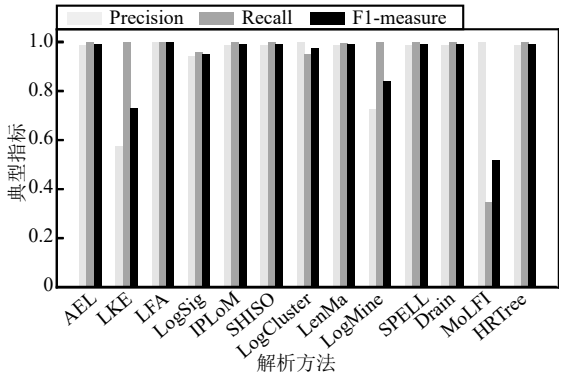
(e) Android



(f) Proxifier

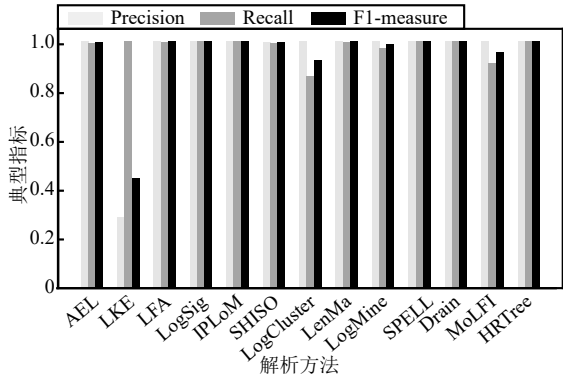


(g) Hadoop

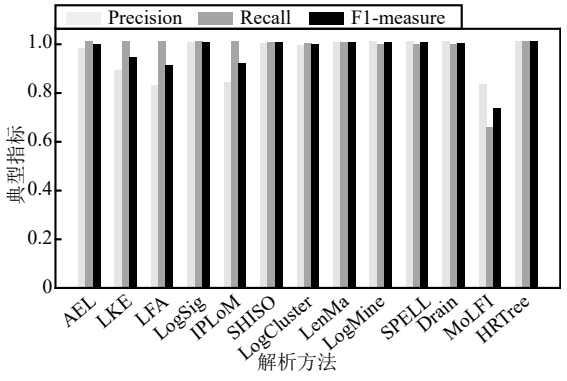


(h) Spark

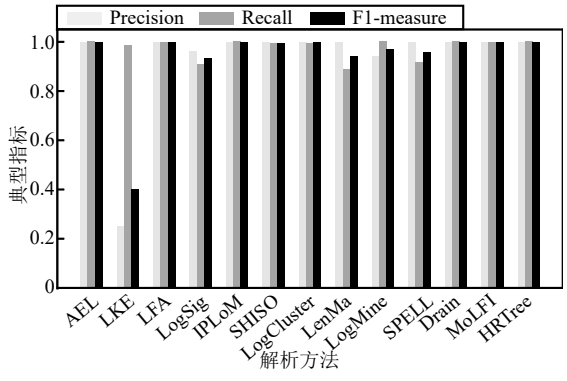
图6 不同解析方法在不同数据集上典型指标评估结果1



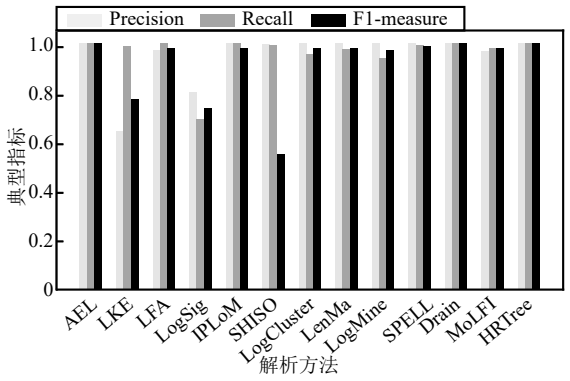
(a) Zookeeper



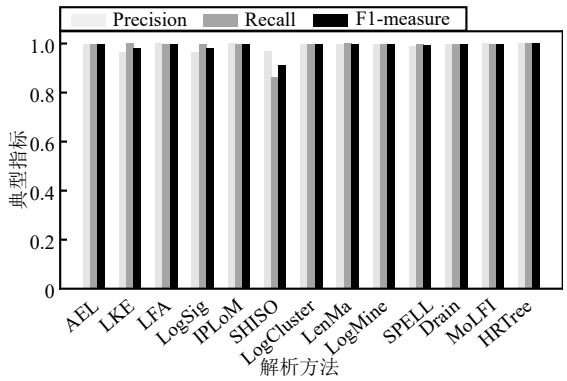
(b) OpenStack



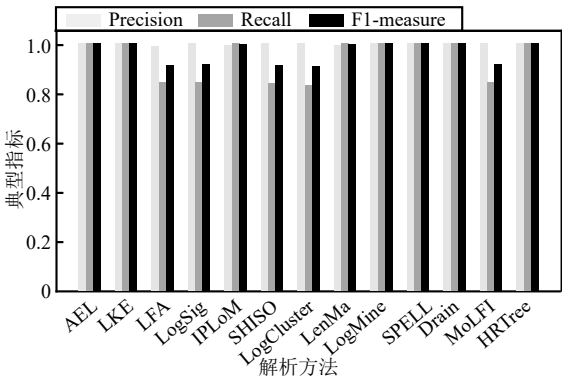
(c) BGL



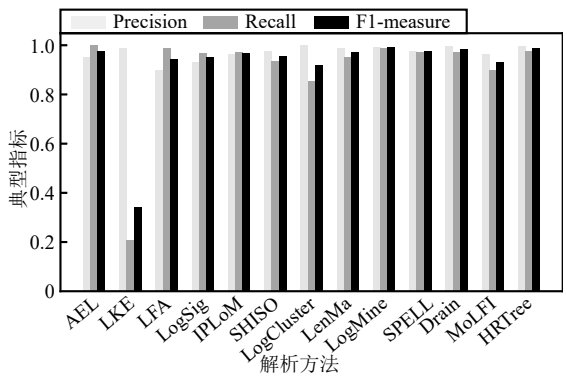
(d) HPC



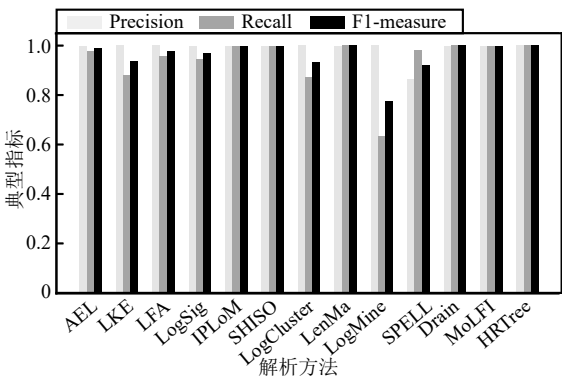
(e) Thunderbird



(f) Windows



(g) Mac



(h) OpenSSH

图 7　不同解析方法在不同数据集上典型指标评估结果 2

如图 6(c)～图 6(h)和图 7 所示，除 HRTree 外，其余解析方法在 Linux、HealthApp 等数据集上，均出现了解析效果的较大波动，有些方法 F1-measure 指标甚至低于 80%。解析效果评价较高的 Drain 方法也在 Linux 和 HealthApp 上出现了解析效果的下降，分析原因为该类数据集日志行为事件种类较多，例如，Linux 数据集仅在 2 000 条日志数据中就包含了 115 种日志事件类型，相较于 HDFS 的 14 种事件类型，解析任务难度显著增加。对于该类复杂日志数据集，其他方法出现了将多类行为合并为一类行为的解析不准确现象。而 HRTree 由于更加严格的相似度计算和 token 拆分处理，避免了相似行为的过度合并，在分类的典型指标上效果更好，证明了 HRTree 解析效果面对不同数据集的稳定性。

3.4.2 准确率指标评估

在以上评估指标的基础上，本文对准确率指标 Accuracy 进行评估，在日志解析任务中准确率指标相对于上面 3 个指标更加严格。主要对日志解析的日志内容进行评估，检验日志消息的参数识别是否准确。该指标统计在所有解析结果中，日志事件内容解析正确的日志数量占所有日志总数的比例，其定义如下

$$\text{Accuracy} = \frac{\text{解析正确日志数量}}{\text{总日志数量}} \tag{6}$$

与 3.4.1 节一样，本节使用 Accuracy 对所有数据和方法进行了全面的评估，结果如图 8 和图 9 所示。在部分数据集上，虽然典型指标评估效果可以接受，但是当使用更加严格的准确率指标时，有些方法出现了大范围的性能下降。这是因为虽然解析方法对于日志事件的分类是正确的，例如都准确解析成同一类行为，但是对于具体的解析事件模板的内容，部分解析方法并没有对其中的部分参数进行准确识别，出现了参数的误识别和漏识别，导致在该性能指标上评估效果的下降。而准确识别日志参数，可能成为影响后续日志异常检测的关键。因此其他解析方法未能在精准识别参数和模板上展现很好的能力。对于本文 HRTree 解析方法，由于加入了启发式的规则，特别是对于 "word = value″、"word:value″和 "(word,value)″这类日志信息，能够在字符层面准确识别具体的参数，避免了误识别。同时，加入的部分规则化条件可以对常用变量提前处理，避免了漏识别，因此展现了较好的解析结果。同时可以帮助后续的日志检测方法，特别是语义层面的特征提取上，实现更准确的特征提取。

3.4.3 时间效率评估

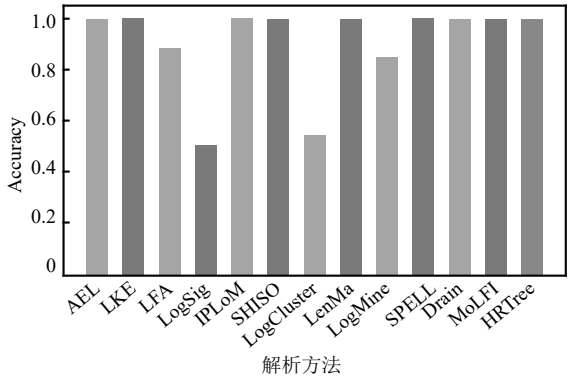
为了处理大规模的日志数据，效率是日志解析方法需要考虑的一个重要方面。部分方法虽然解析效果相近，但是无法胜任大规模日志数据的解析问题，因此能否在大数据条件下高效解析十分重要。为了衡量日志解析方法的效率，本文对日志解析方法在不同数量级日志数据上进行了解析运行时间的测试，对每种方法从读取、解析、到最后数据结果的输出，进行了全流程运行时间的统计。

由于 Zhu 等公开的日志数据仓库提供的每个数据集的数据规模较小，各类解析方法均能在较短时间内完成，不能检验其在大规模日志数据上的解析效果。通过查看公开的全量数据集，部分数据集规模并不大，例如 Linux 和 Proxifier 数据集仅有 2 万条。为满足百万级别以上实验条件，本文选取了 HDFS、BGL 和 Android 这 3 个日志数据集，分别为分布式系统日志、超级计算机系统日志和移动系统日志，其中 HDFS 数据集达到了千万条级别，BGL 数据集为 400 万条，Android 数据集为 100 万条。

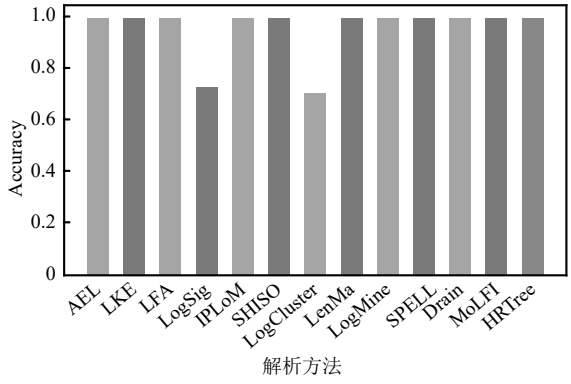
本文在相同数据集上进行了不同数据规模的拆分，构建了不同数量条目的测试数据。例如，HDFS 有 1 000 万条日志，本文测试了 2×10^3～$10\,000\times10^3$ 的 7 个数据规模，而对于 BGL 数据集和 Android 数据集，由于数据仅在百万规模，本文测试了 2×10^3～$1\,000\times10^3$ 的 5 个数据规模。

由于方法众多，本文仅从相同原理方法中选择了 1~2 种解析方法进行对比，频繁模式挖掘中选择了 LFA 方法，聚类解析方法中选择了 SHISO 解析方法，启发式方法中选择了分层聚类的 IPLoM 和 Drain，其他方法中选择了 SPELL。其中，LFA 和 IPLoM 为离线解析方法，其余为在线解析方法。

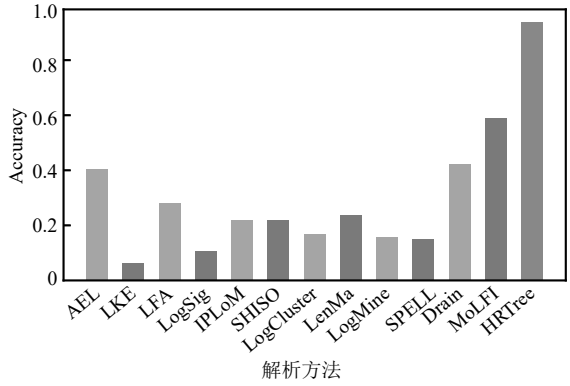
不同解析方法时间效率对比如图 10 所示。从图 10 可以看出，几种方法的解析时间随着数据规模的增大整体呈线性增加趋势。SHISO 在 3 个数据集上均表现为最差的解析速率，在百万级别的日志规模上，与其他方法相差了近 10 倍。IPLoM 和 LFA 解析方法在 3 个数据集上展现了相近的解析效率，解析速度最快，主要原因为其使用了分层聚类的思想，通过日志长度的启发式思想，分层解析大



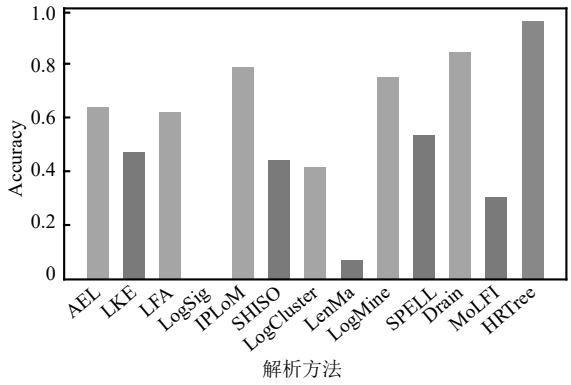
(a) HDFS



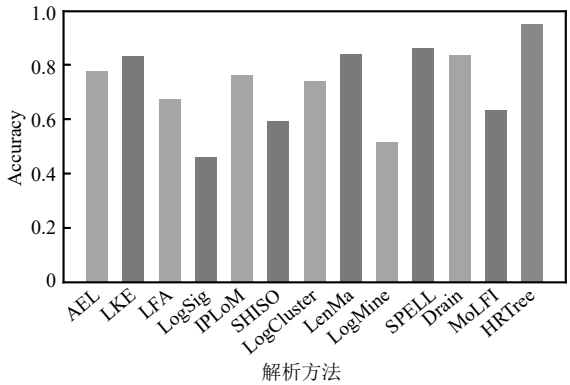
(b) Apach



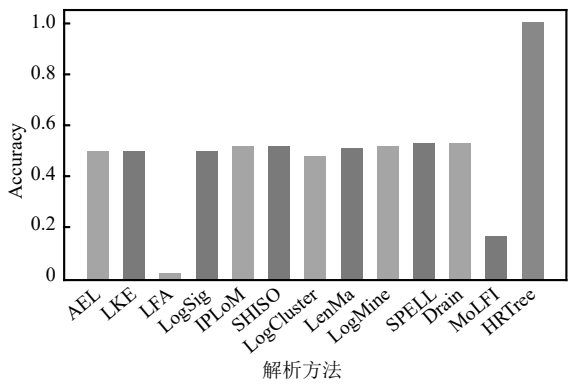
(c) Linux



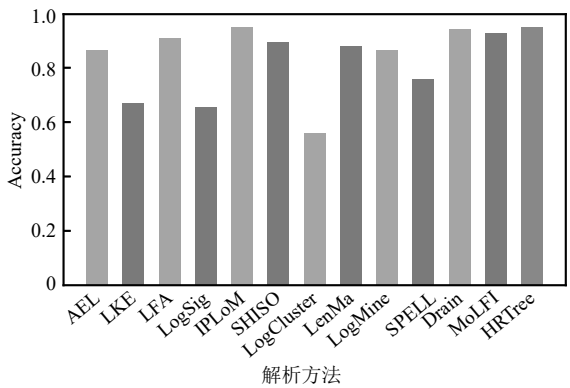
(d) HealthApp



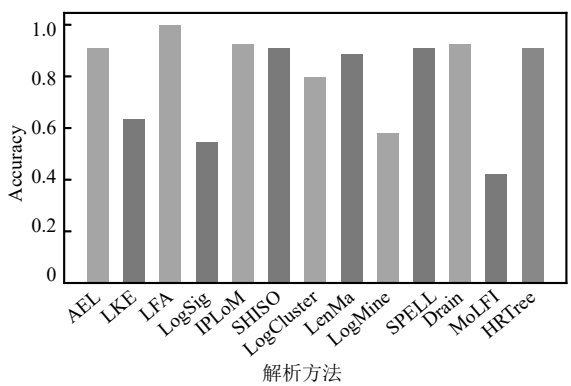
(e) Android



(f) Proxifier

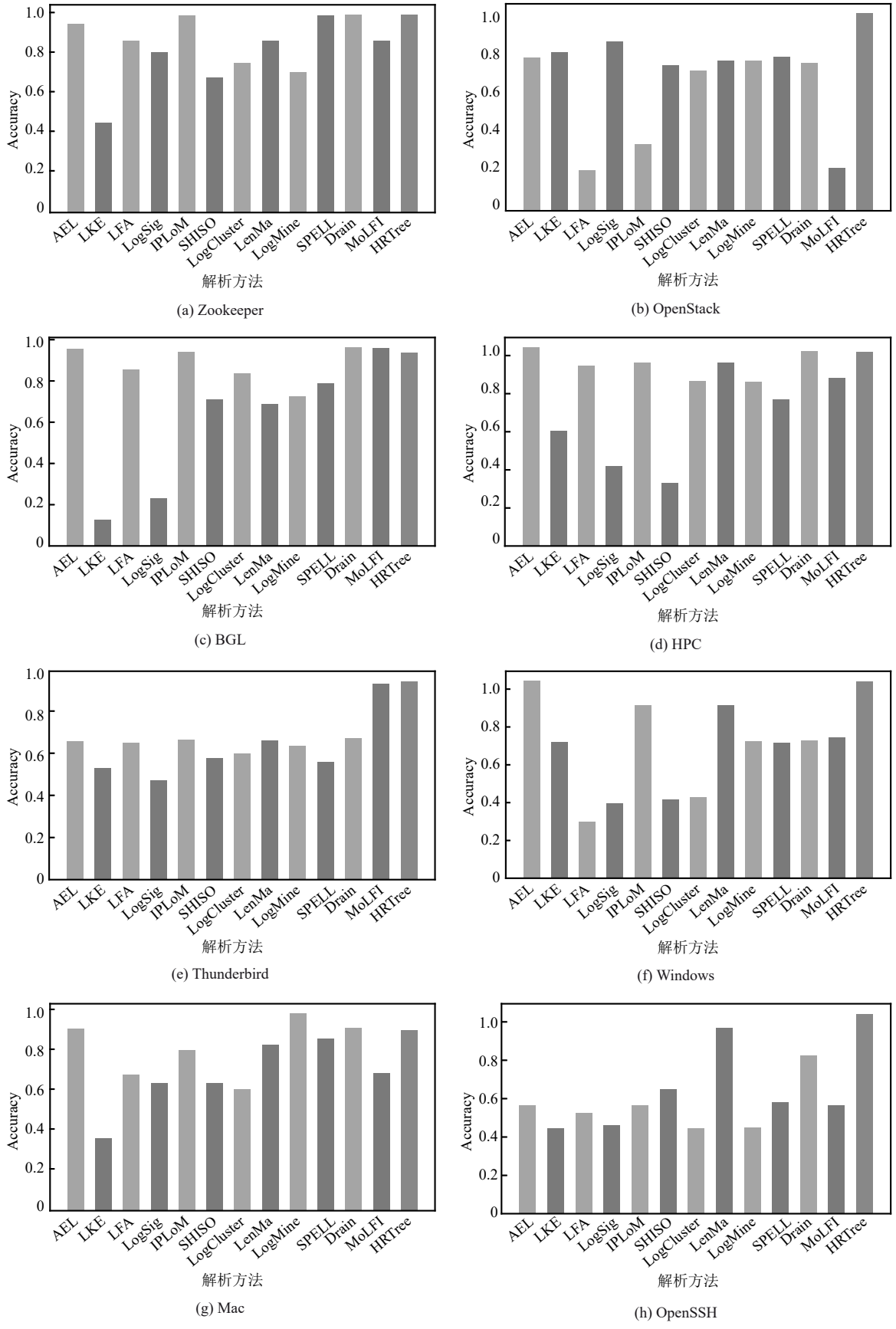


(g) Hadoop



(h) Spark

图 8　不同解析方法在不同数据集上准确率指标评估结果 1



(a) Zookeeper　(b) OpenStack

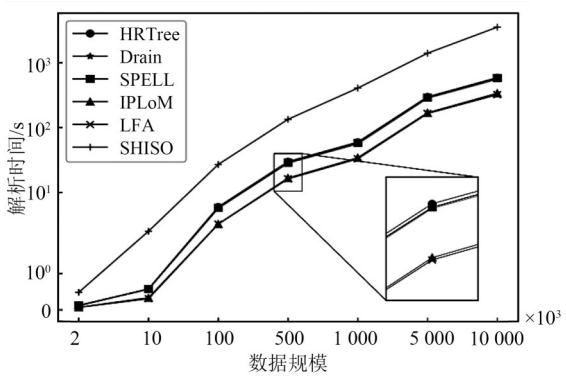
(c) BGL　(d) HPC

(e) Thunderbird　(f) Windows

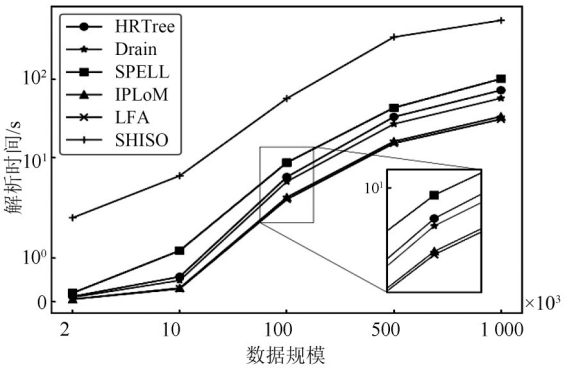
(g) Mac　(h) OpenSSH

图9　不同解析方法在不同数据集上准确率指标评估结果2

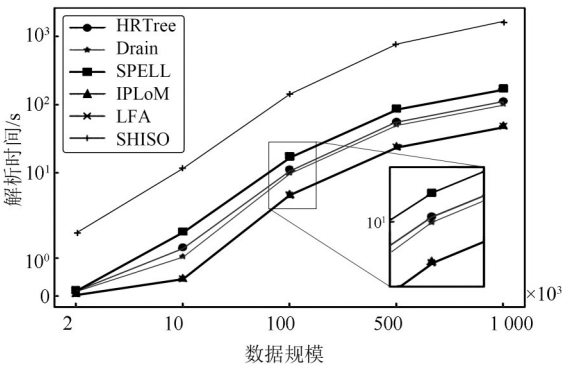
大提高了解析的速度。但是这2种方法均为离线解析，需要全部将日志数据读入内存，对计算资源占用较大，当测试数据超过处理器内存时，无法进行有效解析，所以对于更大规模的日志数据集，无法胜任解析工作。



(a) HDFS



(b) BGL



(c) Android

图10　不同解析方法时间效率对比

Drain、HRTree 和 SPELL 解析速度相差不大，其中，Drain 的解析速度相对较快，HRTree 居中。HRTree 由于和 Drain 具有类似的解析原理，加入 token 拆分和规则化过程，解析速度略慢于 Drain，但两者相差不大。SPELL 虽然采用了最长公共子序列算法，但是其结构也使用前缀树，在解析效率上也有不错的表现。整体分析，HRTree 具有和 Drain 相当的解析效率，并且为在线流式解析方法，能够胜任大规模日志数据的实时解析，不受计算机内存问题的限制，是一种高效率的解析方法。

4　结束语

本文主要提出了一种基于启发式规则的流式日志解析方法，在此之前分析了目前日志解析问题的详细定义，并且对几大难点问题进行了分析。之后对 HRTree 的解析流程结合算法步骤进行了详细的介绍，阐述了 HRTree 解决的几大问题。最后在典型指标、准确率指标和运行时间效率指标几个方面进行了详细的评估，无论是在典型指标还是准确率指标方面，HRTree 均表现了较理想的解析效果，特别是在面对不同的日志数据集时，当其他解析方法的解析效果变差时，HRTree 展现了稳定的解析效果。在运行效率方面，HRTree 作为一种在线的解析方式，可以不受内存的限制，并且展现出了和 Drain 相近的解析速度。通过以上几个指标的评估实验，证明了本文提出的 HRTree 方法是一种可以覆盖全部日志信息，实现在线高效率和高准确率解析的一种有效方法。

参考文献:

[1] 廖湘科, 李姗姗, 董威, 等. 大规模软件系统日志研究综述[J]. 软件学报, 2016, 27(8): 1934-1947.
LIAO X K, LI S S, DONG W, et al. Survey on log research of large scale software system[J]. Journal of Software, 2016, 27(8): 1934-1947.

[2] FU Q, LOU J G, LIN Q W, et al. Contextual analysis of program logs for understanding system behaviors[C]//Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR). Piscataway: IEEE Press, 2013: 397-400.

[3] CHOW M, MEISNER D, FLINN J, et al. The mystery machine: end-to-end performance analysis of large-scale internet services[C]//Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). Berkeley: USENIX Association, 2014: 217-231.

[4] AIT E H M, KHOUMSI A, BENKAOUZ Y, et al. Efficient security policy management using suspicious rules through access log analysis [M]. Belin: Springer, 2019.

[5] CINQUE M, COTRONEO D, PECCHIA A. Event logs for the analysis of software failures: a rule-based approach[J]. IEEE Transactions on

Software Engineering, 2013, 39(6): 806-821.

[6] PREWETT J E. Analyzing cluster log files using logsurfer[C]//Proceedings of the 4th Annual Conference on Linux Clusters. PA: Citeseer, 2003: 1-12.

[7] ROUILLARD J P. Refereed papers: real-time log file analysis using the simple event correlator (SEC) [C]//Proceedings of the 18th USENIX Conference on System Administration. Berkeley: USENIX Association, 2004: 133-150.

[8] LIN Q W, ZHANG H Y, LOU J G, et al. Log clustering based problem identification for online service systems[C]//Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). Piscataway: IEEE Press, 2016: 102-111.

[9] MI H B, WANG H M, ZHOU Y F, et al. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1245-1255.

[10] ZHAO X Q, JIANG Z Y, MA J F. A survey of deep anomaly detection for system logs[C]//Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN). Piscataway: IEEE Press, 2022: 1-8.

[11] HARDY S, HENECKA W, IVEY-LAW H, et al. Private federated learning on vertically par- titioned data via entity resolution and additively homomorphic encryption[J]. arXiv Preprint, arXiv:1711.10677, 2017.

[12] DI S, GUO H Q, PERSHEY E, et al. Characterizing and understanding HPC job failures over the 2K-day life of IBM BlueGene/Q system[C]//Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Piscataway: IEEE Press, 2019: 473-484.

[13] XU W, HUANG L, FOX A, et al. Detecting large-scale system problems by mining console logs[C]//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. New York: ACM Press, 2009: 117-132.

[14] NAGAPPAN M, WU K S, VOUK M A. Efficiently extracting operational profiles from execution logs using suffix arrays[C]//Proceedings of the 20th International Symposium on Software Reliability Engineering. Piscataway: IEEE Press, 2009: 41-50.

[15] VAARANDI R. A data clustering algorithm for mining patterns from event logs[C]//Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003). Piscataway: IEEE Press, 2003: 119-126.

[16] NAGAPPAN M, VOUK M A. Abstracting log lines to log event types for mining software system logs[C]//Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). Piscataway: IEEE Press, 2010: 114-117.

[17] VAARANDI R, PIHELGAS M. LogCluster - A data clustering and pattern mining algorithm for event logs[C]//Proceedings of the 11th International Conference on Network and Service Management (CNSM). Piscataway: IEEE Press, 2015: 1-7.

[18] DAI H, LI H, CHEN C S, et al. Logram: efficient log parsing using n-Gram dictionaries[J]. IEEE Transactions on Software Engineering, 2020,47(12):2803-2822.

[19] FU Q, LOU J G, WANG Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis[C]//Proceedings of the 2009 Ninth IEEE International Conference on Data Mining. Piscataway: IEEE Press, 2009: 149-158.

[20] TANG L, LI T, PERNG C S. LogSig: generating system events from raw textual logs[C]//Proceedings of the 20th ACM international conference on Information and knowledge management. New York: ACM Press, 2011: 785-794.

[21] HAMOONI H, DEBNATH B, XU J W, et al. LogMine: fast pattern recognition for log analytics[C]//Proceedings of the Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. New York: ACM Press, 2016: 1573-1582.

[22] MIZUTANI M. Incremental mining of system log format[C]//Proceedings of the 2013 IEEE International Conference on Services Computing. Piscataway: IEEE Press, 2013: 595-602.

[23] SHIMA K. Length matters: Clustering system log messages using length of words[J]. arXiv Preprint, arXiv:1611.03213, 2016.

[24] DU M, LI F F. SPELL: streaming parsing of system event logs[C]//Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM). Piscataway: IEEE Press, 2016: 859-864.

[25] MESSAOUDI S, PANICHELLA A, BIANCULLI D, et al. A search-based approach for accurate identification of log message formats[C]//Proceedings of the 26th Conference on Program Comprehension. Piscataway: IEEE Press, 2018: 167-177.

[26] DEB K, PRATAP A, AGARWAL S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197.

[27] MENG W B, LIU Y, ZAITER F, et al. LogParse: making log parsing adaptive through word classification[C]//Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN). Piscataway: IEEE Press, 2020: 1-9.

[28] JIANG Z M, HASSAN A E, FLORA P, et al. Abstracting execution logs to execution events for enterprise applications (short paper) [C]//Proceedings of the 8th International Conference on Quality Software. Piscataway: IEEE Press, 2008: 181-186.

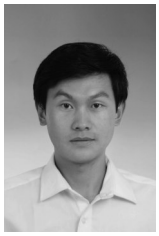
[29] MAKANJU A, ZINCIR-HEYWOOD A N, MILIOS E E. A lightweight algorithm for message type extraction in system application logs[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(11): 1921-1936.

[30] HE P J, ZHU J M, ZHENG Z B, et al. Drain: an online log parsing approach with fixed depth tree[C]//Proceedings of the 2017 IEEE International Conference on Web Services (ICWS). Piscataway: IEEE Press, 2017: 33-40.

[31] HE P J, ZHU J M, HE S L, et al. An evaluation study on log parsing and its use in log mining[C]//Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Piscataway: IEEE Press, 2016: 654-661.

[32] ZHU J M, HE S L, LIU J Y, et al. Tools and benchmarks for automated log parsing[C]//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Piscataway: IEEE Press, 2019: 121-130.

[作者简介]



蒋忠元（1988– ），男，陕西榆林人，博士，西安电子科技大学教授、博士生导师，主要研究方向为隐私保护、社会计算、城市计算和网络功能虚拟化。

陶梅悦（2001– ），女，安徽芜湖人，西安电子科技大学硕士生，主要研究方向为异常检测、软件可靠性等。

赵晓庆（1996– ），男，山东泰安人，西安电子科技大学硕士生，主要研究方向为大数据安全、数据挖掘和异常检测等。

方晓彤（1989–），女，安徽合肥人，中国船舶集团有限公司高级工程师，主要研究方向为舰船通用质量特性设计与分析。

李兴华（1978– ），男，河南南阳人，博士，西安电子科技大学教授、博士生导师，主要研究方向为隐私保护、网络与信息安全。

马建峰（1963– ），男，陕西西安人，博士，西安电子科技大学教授、博士生导师，主要研究方向为密码学、无线网络安全、数据安全和移动智能代理系统安全。