计算机科学与探索
Journal of Frontiers of Computer Science and Technology
1673-9418/2022/16(10)-2264-09
doi: 10.3778/j.issn.1673-9418.2103066

高性能计算环境中日志模式提炼方法的研究

王晓东[1,2],赵一宁[1+],肖海力[1],王小宁[1],迟学斌[1,2]
1. 中国科学院 计算机网络信息中心,北京 100190
2. 中国科学院大学,北京 100049
+ 通信作者 E-mail: zhaoyn@sccas.cn

摘　要:日志分析对于计算机系统的稳定运行起着至关重要的作用,然而日志通常是非结构化的,不利于自动化分析,如何自动化将日志的模式提炼出来并变成结构化的数据具有重要的实际意义。提出了LDmatch算法,该算法以单词匹配率为基础实现了一种日志模式提炼算法。传统的日志匹配算法在进行相似度计算时使用一对一单词匹配法,而LDmatch算法根据两条日志所包含的单词之间的最长公共子序列计算日志之间的相似度,并以此为基础进行日志分类。LDmatch算法还能实时得到日志模板并更新。除此之外,该算法的模式仓库使用了基于哈希表的数据结构进行存储,该存储结构细化了日志的分类,减少了日志匹配时的比较次数,从而提高了日志模式提炼算法的匹配效率。为了验证算法的优势,将LDmatch算法应用于开源数据集以及国家高性能计算环境实际产生的日志数据集,并且使用多种其他日志模式提炼算法进行对比并得出实验结果,最终证明了该算法在准确度、鲁棒性和效率上具有优势。

关键词:日志模式提炼;单词匹配率;日志模板;哈希表
文献标志码:A　　**中图分类号:**TP316

Research on Method of Log Pattern Extracting in High-Performance Computing Environment

WANG Xiaodong[1,2], ZHAO Yining[1+], XIAO Haili[1], WANG Xiaoning[1], CHI Xuebin[1,2]
1. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
2. University of Chinese Academy of Sciences, Beijing 100049, China

Abstract: Log analysis plays an important role in the stable operation of computer system. However, logs are usually unstructured, which is not conducive to automatic analysis. How to categorize logs and turn them into structured data automatically is of great practical significance. In this paper, LDmatch algorithm is proposed, which implements a log pattern extracting algorithm based on word matching rate. Traditional log matching algorithms use one-to-one word matching method in similarity calculation, while the proposed LDmatch algorithm calculates the similarity between logs according to the longest common subsequence (LCS) of words contained in two logs, and classifies logs based on the LCS. LDmatch algorithm can also get real-time log template and update. In addition, the pattern warehouse of the algorithm uses a data structure based on hash table for storage, which refines the classification of logs and reduces the times of comparison during log matching, thus improving the matching efficiency of the algorithm. In order to verify the advantages of the algorithm, it is applied to the open source data set and the actual log data set generated by the CNGrid. A variety of other log pattern extraction algorithms are used for comparison

基金项目:中国科学院战略性先导科技专项项目(A类)(XDA19020101)。
This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA19020101).
收稿日期:2021-03-19　　**修回日期:**2021-05-20

and experimental results are obtained. Finally, the advantages of the algorithm in accuracy, robustness and efficiency are proven.

Key words: log pattern extraction; word matching rate; log template; hash table

中国国家高性能计算环境是由国内众多超算中心和高校的计算集群组成的国家级大型高性能计算环境，采用中国科学院计算机网络信息中心自主研发的网格环境中间件SCE[1]聚合了大量的通用计算资源，为全国众多高校和研究机构的用户提供了优质的计算服务。为了保障环境的稳定运行，搜集环境中产生的日志并且实时分析解决环境中出现的异常是非常必要的。然而随着环境中计算资源和用户的增多，各节点产生的日志也越来越多，使用传统的人工手动分析大量日志的方法已经成为了非常耗时耗力并且容易出错的方法，因此如何进行自动化日志分析对于环境的正常运行和安全保障具有极其重要的意义。

为了解决日志自动化分析的问题，近年来许多研究人员都采用各种数据挖掘的方法对日志进行分析并诊断异常。比如Xu等[2]就通过主成分分析法对控制台日志数据进行异常诊断，Lou等[3]通过不变量挖掘方法对在系统中的日志进行异常诊断，Lin等[4]通过聚类方法从在线服务器中的日志中寻找异常，Du等[5]通过深度学习的方法从系统日志中发现并诊断异常。上述这些方法都在自动化日志分析上取得了比较好的效果。但是，所有自动化分析时的原始日志消息通常是非结构化的，这是因为程序在实际开发时是灵活多变的，开发人员习惯使用自由文本记录日志消息。而为了能够自动挖掘非结构化日志，第一步需要解决的问题就是日志模式提炼，通过该步骤将非结构化的原始日志消息转换为结构化日志消息后才能在后续使用不同数据挖掘算法来自动化分析各种异常。

具体来说，日志模式提炼的目标是将一条日志中非结构化的部分分类并将同一类模式的日志拆分成常量部分和变量部分。比如高性能计算集群中Linux的一条系统日志如下：

Sep, 30, 03:50:55, client51, sshd, 32019, Invalid user UserName from 192.168.0.1

如果按照逗号分开，可以看出这条日志的前三个字段代表时间，中间三个字段分别代表主机名、守护进程名称和PID号。这些字段属于日志的结构化部分，使用简单的正则表达式就能提取出来，而日志模式提炼主要关注的是最后一个字段的内容，通过日志模式提炼算法需要把最后一个字段的内容抽象成：

Invalid user <*> from <*>

其中，Invalid user from是日志的常量部分，通配符<*>代表日志的变量部分。在对高性能计算环境日志进行模式提炼的研究上，Zhao等[6]提出了算法Match，该算法通过单词匹配率来确定两条日志的相似程度，并以此来确定日志所属的模式，同时还提出了树匹配的算法来实现日志模式提炼算法。之后在文献[7]中描述了算法Lmatch（longest common subsequence match algorithm），该算法改进了单词匹配率算法，即将日志中的每一个单词作为一个基本单元，然后通过两条日志的最长公共子序列来计算单词的匹配数目，最后与两条日志的总单词数进行比较来计算单词匹配率。虽然以上方法取得了比较好的效果，然而其中仍然有需要改进的地方：

首先，在使用最长公共子序列进行日志模式提炼时，仅仅能够得到日志模式，并没有进一步将模式抽象成常量部分和变量部分，将日志拆分成常量部分和变量部分有以下优势：当使用日志进行下游任务的分析时，日志模板的变量部分可以和实时日志进行匹配，通过将匹配到的变量提取出来可以得到相同变量的日志在事件上的关联关系，对后续进行日志分析具有重要作用。其次，在日志模式的储存结构上，原文使用基于首个单词的散列表进行存储，由于日志复杂多变，可能出现首个单词种类过多的问题，同时当首个单词属于日志变量部分的情况下只能通过人工处理。最后，日志模式提炼算法的调参方式不够明确。

基于以上问题，本文详细描述了日志模式提炼算法LDmatch（longest common subsequence dictionary match algorithm）。该算法通过确定两条待比较日志的最长公共子序列作为常量部分，待比较日志的其余部分作为变量部分来确定一条日志的模式，同时该方法还支持实时增量的在线运行并提取日志模式。在日志模式的存储结构上，使用日志的首字母作为散列函数的输入，从而约束了散列函数字典的长度上限。在整个算法的参数优化上，使用大量的实验进行调整，最终为多种不同类别的日志模式提炼确定了最佳参数。总体来说，本文有以下两个贡献点：

（1）进一步优化了国家高性能计算环境中使用

的日志模式提炼算法，为后续自动化异常检测分析提供了支持；

（2）该日志模式提炼算法在开源日志数据集以及国家高性能计算环境中产生的真实日志数据集上进行多维度的实验分析，实验结果证明了该方法的优势。

1 相关工作

本章简要介绍日志模式匹配算法近年来在不同方向的研究进展。

1.1 基于频繁模式的日志模式提炼算法

Vaarandi[8]在对日志文件数据进行模式分类时使用了一个名叫SLCT（simple logfile clustering tool）的聚类算法，该聚类算法是基于Apriori频繁项集的算法，因此需要使用者手动输入调整支持阈值。SLCT会对日志进行两次整体的扫描：第一次对日志中所有的单词进行词频的统计，在第二次扫描时根据第一次扫描时得到的词频建立起日志的模式集群。经过两次扫描后，该算法最终根据建立的集群为每一个集群生成一个日志模板。Nagappan等人[9]也提出了一个基于频繁模式挖掘的日志模式提炼方法LFA（abstraction of log lines），该算法与SLCT不同的地方在于考虑了每条日志消息中的单词的频率分布，而不是对整个日志数据进行罕见日志消息的解析。

1.2 基于聚类的日志模式提炼算法

Fu等人[10]使用LKE（log keys)方法对日志进行模式提炼，该方法结合使用了聚类算法和启发式规则法，一共有三个步骤：第一步是日志聚类，聚类时使用了自定义加权编辑距离作为衡量两条日志之间的距离度量，然后使用了层次聚类算法对原始日志消息进行聚类；第二步是聚类结果拆分，执行基于启发式规则法来进一步拆分聚类结果；第三步是日志模板生成，该步骤为每个聚类的群集生成日志模板，类似1.1节中所描述的SLCT算法的最后一步。Tang等人[11]提出了LogSig算法来提炼日志模式，该算法的工作流程也可分为三个步骤：第一步生成单词对，将每个日志消息转换为一组单词对，然后对该单词及其位置信息进行编码。第二步进行了日志聚类，该聚类算法基于单词对，为每个日志消息计算一个潜在的值，从而确定日志消息可能属于的集群。经过多次迭代后，最终就可以得到聚类之后的日志消息集群。第三步是日志模板的生成，即利用每个日志集群中的一系列日志消息为每个集群都生成一个日志模板。Mizutani[12]提出SHISO（scalable handler for incremental system log）算法，通过构造一个结构化的树结构得到日志模板，该算法属于在线算法，并且不需要任何先验知识就能得到结果。

1.3 基于其他方法的日志模式提炼算法

Messaoudi等人[13]提出了MoLFI（multi-objective log message format identification）算法，该算法将日志解析建模为一个多目标优化问题，并用进化算法进行日志模式生成。Dai等人[14]提出了一种自动的日志解析方法Logram，该算法利用n-gram字典来实现日志解析。Nedelkoski等人[15]将日志模式分类任务应用于深度学习中的掩码语言模型（masked language model, MLM）中，在日志解析的过程中，该模型以向量嵌入的形式从日志中提取关键信息，通过反向传播训练后，最终模型可以输出日志模板。Zhao等人[6]提出了模式提炼算法Match来确定两条日志的相似程度，该算法基于两条日志的单词匹配率。同上述提到的其他算法相比，该算法的最大优势是可以实时计算，也不需要预先提炼模式，因此该算法利于线上的日志模式提炼分析。之后该作者又在文献[7]中对Match算法进行改进而得到Lmatch算法，该算法将两条日志固定位置的一对一匹配改进成根据最长公共子序列进行匹配，这样做可以有效解决相同模式的两条日志在常量部分位置不同的问题。

大多数基于频繁模式的算法和基于聚类的日志模式提取算法都是离线算法，因为这些算法需要在第一步扫描所有的历史日志，而本文方法可以在线处理日志解析，更适合生产实践。另一方面，基于深度学习的方法虽然能够取得较好的精度，但是运行时参数过多，对于实时的日志分析来说过于繁重，因此不适合高性能计算环境中日志模式提炼的场景。

2 日志模式提炼算法

本章首先介绍国家高性能计算环境中的日志分析框架以及在日志解析时需要解决的关键性问题，然后针对该问题，详细介绍本文提出的日志模式提炼算法和相关改进工作。

2.1 问题背景描述

国家高性能计算环境的网格环境日志分析框架（log analysing framework in grid environment，LARGE）是针对中科院超级计算环境中各类日志进行分析处理的框架式结构，它定义了框架内各模块的工作内容以及整个日志分析流程的数据传输流向和处理步骤[1]。其基本结构和工作流程图如图1所示。

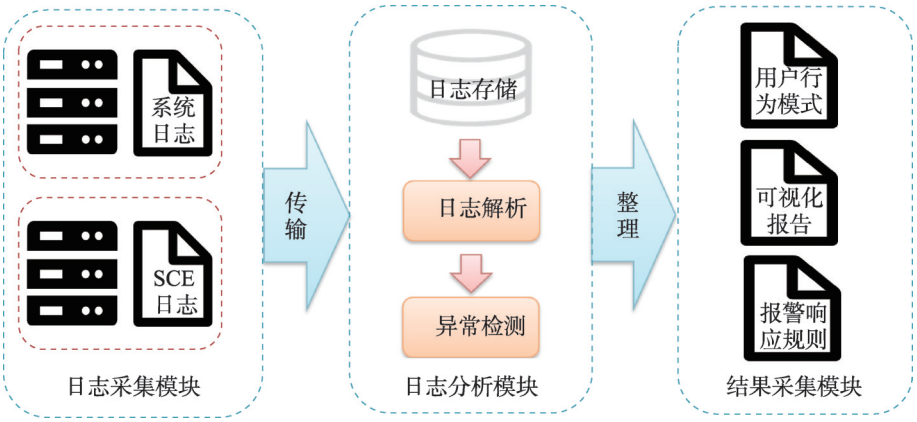

图1 LARGE系统结构

Fig.1 System structure of LARGE

从图1中可以看出，国家高性能计算环境中的日志搜集到需要进行分析的日志主要包括SCE日志和系统日志。SCE日志是由SCE网格环境中间件生成的事件日志，该类日志格式固定，因此非常有利于机器读取并分类。而系统日志格式就比较复杂，比如引言中的日志示例。可以看出该日志的前三个字段很容易得到，然而最后一个字段才是需要重点关注的信息，该字段的内容虽然是人类可以阅读的英文句子，然而这种人类容易理解的句子属于非结构化的数据，这种数据反而不利于机器识别，因此日志模式提炼的主要关注点是日志的非结构化部分。如何使用合理的模式匹配算法对日志该部分的信息进行分类是国家高性能计算环境日志分析首先要解决的问题。另一方面，日志模式提炼算法总体来说分为线上方法和线下方法。线下方法通常需要有历史的日志数据，然后经过一轮遍历对历史日志进行聚类，之后为每一类日志提取出一个模板。这种方法虽然直观，但是在实际应用时对于一条新的日志如果不满足已知类别的情况就没法实时形成新的类别。因此实际应用的时候还是线上日志分类方法比较有价值。国家高性能计算环境中日志的类别数量无法提前确定，因此需要使用线上方法进行分类。为了使线上日志分类算法可以在新日志出现时更新已有模板，同时还能在新日志进行匹配时效率有所提升，该算法还需要对模板的实时更新方法以及日志模板仓库的存储方式进行优化。面对以上问题，后续详细介绍线上日志分类算法以及相关优化方案。

2.2 日志模式提炼算法与线上日志分析流程

对于如何确定两条日志是否属于同一个模式的问题，最直观的方法就是确定两条日志匹配的单词是否足够多，因此本文使用单词匹配率来确定日志的模式。该算法将日志中的每一个单词作为一个基本单元，然后对其进行匹配。具体来说，假设待匹配的两条日志分别为 l' 和 l，其包含的单词数量分别为 m 和 n，则两条日志的单词匹配率的计算公式如下：

$$r(l,l')=\frac{|Match(l,l')|\times 2}{n+m} \tag{1}$$

其中，$|Match(l,l')|$ 代表两条日志对应位置一对一匹配单词数目。然而该算法无法处理两条日志的常量部分位置不同的问题。因此改进的单词匹配率的计算公式如下：

$$r(l,l')=\frac{|Lmatch(l,l')|\times 2}{n+m} \tag{2}$$

其中，$|Lmatch(l,l')|$ 代表两条日志使用最长公共子序列进行匹配时匹配到的单词数目。比如以下三条日志：

Invalid user User1 from IP1 （a）

Invalid user User2 from IP2 （b）

Accepted publickey for User3 from IP3 （c）

分别记作a、b、c，并且假设阈值为0.45，则其中a和b这两条日志的前两个字符和第四个字符相匹配，这两条日志都有五个字符，因此根据式（1），可以计算得到 $r(a,b)=3\times 2/(5+5)=0.6>0.45$，说明a、b为同一种模式的日志，同理可以计算出a和c这两条日志的单词匹配率 $r(a,c)=0.18<0.45$，说明a、c为不同模式的日志。

有了单词匹配率的定义，就可以得到整个日志模式提炼模块的处理流程，具体步骤如下：

（1）读入一条新的日志 l，然后与日志模式仓库中已有的日志模式 $l_1,l_2,\cdots,l_n$ 分别计算出单词匹配率 $r(l,l_1),r(l,l_2),\cdots,(l,l_n)$；

（2）从这些计算出来的结果中找到最大的单词匹配率 $R=\max\{r(l,l_1),r(l,l_2),\cdots,r(l,l_n)\}$ 以及计算该单词

匹配率对应的日志模式 l_m；

(3)将 R 与提前设定好的阈值 t 进行对比；

(4)如果 $R>t$，则将该日志 l 与对应日志模式 l_m 进行模板提取并更新该日志模式；

(5)如果 $R<t$，则使用该条日志 l 生成新的日志模式 l_{n+1} 并加入到日志模式仓库中。

整个日志模式提炼模块的处理流程图如图2所示，其中阈值 t 在3.3节的实验中确定，日志模板提取算法将在2.3节中详细介绍。

2.3 日志模板提取以及日志模式仓库的优化

上一节已经构建了日志模式提炼算法的整体流程，本节将进一步改进本文算法，使得该算法可以实时提炼并更新日志的模式。除此之外，日志仓库的存储类型也在本节中改进以提高整个算法的准确度和效率，最终构建出完善的LDmatch算法。

为了满足基于最长公共子序列的日志模式提炼算法的实时性，生成的模式必须满足一些条件，即可以自动将两条日志的匹配部分作为最终模板的常量部分，非匹配部分作为最终模板的变量部分。要想满足这一点，需要首先计算出两条待匹配日志的最长公共子序列，然后进行后续操作。具体来说，日志模式提炼算法流程如算法1。

算法1 计算日志模板算法

输入：日志仓库中的模式日志 l_m，待比较日志 l。

输出：构造出日志模板retVal。

1. 初始化：设置 *retVal* =[]
2. *lcs* ← *LCS*(l_m,l)
3. *lcs.reverse* ()
4. for i = 1,2,⋯,*length*(l) do
5. if l [i]==*lcs*[*length*(*lcs*)−1] then
6. *retVal.append* (l[i])
7. *lcs.pop*()
8. else
9. *retVal.append*(<*>)
10. end if
11. if *length*(*lcs*)==0 do
12. break
13. end if
14. end for
15. if i<*length* (l) do
16. *retVal.append*(<*>)
17. end if

根据算法1所示，可以看出当一条日志进入日志类型仓库中时，首先计算日志模板和日志的最长公共子序列，然后反转最长公共子序列，之后对待比较日志的单词进行遍历：如果该单词与最长公共子序列第一个单词一样，则说明该单词属于日志的常量部分，此时在日志模板中加入该单词；否则说明该单词属于变量部分，此时在日志模板中加入变量匹配符“<*>”。加入成功后，则将最长公共子序列中该单词删除，然后进行下一次判断，直到最长公共子序列为空时退出该循环。最后，如果待匹配日志后续还有单词，说明都是变量，因此直接加入变量匹配符“<*>”。按照上述算法流程，就能对日志仓库中已有的日志模板进行实时更新。同时这个算法也满足两条日志非结构化的公共部分是模板的常量，而不同的部分是模板的变量。

回顾日志模式提炼算法的流程可以发现，每当有新的日志出现，该处理流程都需要分别计算新日志和日志仓库中所有日志模式的最长公共子序列。随着日志模式的增加，计算开销就会越来越大，因此如果缩减比较次数就能显著提高算法的整体效率。基于以上讨论，LDmatch算法使用哈希表存储日志模式，哈希表的输入为日志的第一个字母，这样可以保证哈希表不会无限增大。如果日志的第一个字符不是字母，则统一存储到通配符<*>开头的哈希表中。这样构造的日志模式仓库的示意图见图3。

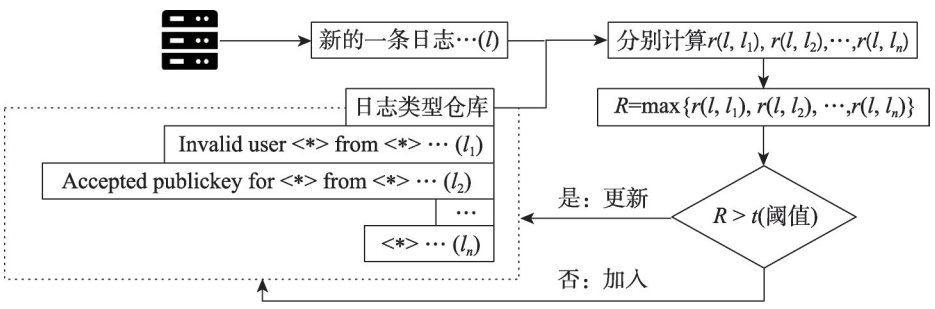

图2 日志模式提炼算法流程图

Fig.2 Flowchart of log pattern extracting algorithms

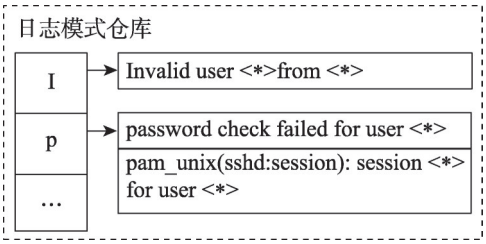

图3 日志模式仓库

Fig.3 Log patterns warehouse

在图3中，可以看到只要按照这种哈希表的存储格式进行日志仓库的保存，就能使得新的一条日志进行匹配时，先根据日志开头信息进入日志模式仓库的对应子集，然后新日志仅需要和该子集中的所有日志模板进行比较即可，从而大大减少了日志的比较次数。

最后，将改进的日志仓库加入日志模式提炼算法的流程中，就得到了LDmatch算法。下一章将会通过实验证明该算法的优势。

3 实验结果与分析

本章先通过实验确定LDmatch算法对高性能计算环境中的日志进行模式提炼时的最优参数，然后通过与其他多种方法对比来证明本文算法在准确度、鲁棒性以及效率上的优势。

3.1 实验数据

本文使用以下两个数据集对实验中所有涉及的方法进行分析评价：

A. Zhu等人[16]在论文中公布了一个开源的数据集Loghub，里面包含16种不同系统和平台产生的日志，每种日志都经过随机挑选2 000条日志，并且经过专业人员手动标注并得到对应的日志模式。

B. 国家高性能计算环境系统在实际工作中产生的系统日志。本文选取系统在2018年9月整个月产生日志中的secure类别日志作为实验数据。

3.2 评价方法

为了对本文方法做尽可能详细的验证，本文通过以下评价指标进行实验。

准确度：正确解析的日志模式与日志模式总数的比率。解析后，每个日志消息都有一个事件模板，该事件模板对应于同一模板会得到一组消息。当且仅当这一组消息与真实的人工标记数据对应的一组日志消息完全相同时，才认为该条日志模式的解析结果是正确的。

为了避免实验产生的随机误差，本文对每组实验结果都经过多次计算并取得平均值。所有实验都是在一台装有英特尔第三代酷睿i5-3230M的处理器、8 GB内存以及Windows 7旗舰版64位系统的计算机上进行的。

3.3 参数确定实验分析

本节进行日志模式提炼算法关键参数确定的实验。其中，阈值参数 t 在取值范围内每隔0.5进行一次准确度计算的实验，因为该阈值是为了应用于国家高性能计算环境，所以本实验选择A数据集中包含模式类别已经被人工标注好的Linux数据集。实验的结果如图4。

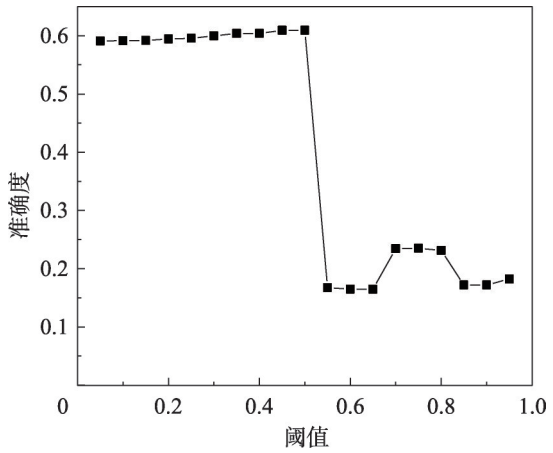

图4 阈值-准确度

Fig.4 Threshold and accuracy

由图4可以看出，LDmatch算法在Linux数据集中的准确度在阈值为0.5以下都比较高，超过0.5则迅速下降，说明Linux系统日志中的变量部分占比较大，基本超过50%，因此不能使用太大的阈值进行模式提炼。当阈值取0.45时，准确度最大。因此后续在高性能计算环境中使用该模式提炼算法时，就使用 $t=0.45$ 作为参数进行其他相关实验。

3.4 不同模式提炼方法对比分析

本节将本文的LDmatch算法和其他模式提炼算法在Linux日志数据上进行准确度的对比，因此本实验选择A数据集中的Linux数据集作为实验对象。本节对比的方法包含基于频繁模式的模式提炼算法SLCT[8]和LFA[9]，基于聚类的模式提炼算法LKE[10]和LogSig[11]，以及其他类别的模式提炼算法MoLFI[12]、Match[6]和Lmatch[7]。经过实验得到准确度的结果如图5所示。

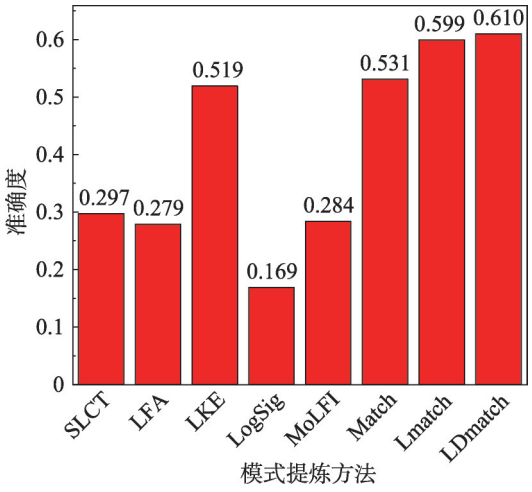

图5 不同模式提炼方法下的准确度

Fig.5 Accuracy in different pattern extracting methods

由图5可以看出,在Linux数据集上,基于频繁模式的模式提炼算法SLCT和LFA的精确度较低,基于聚类的模式提炼算法LKE和LogSig各有好坏,而其他类方法中的Match和Lmatch算法精确度都比较高,说明了基于单词匹配率算法的模式提炼方法的优势。而LDmatch所得的精确度是最高的,由此可见,本文方法非常适用于Linux系统日志的数据。根据2.1节的介绍可知,国家高性能计算环境的日志分析框架LARGE中需要重点解析的日志就是Linux系统中的日志,和普通Linux系统日志的区别在于日志结构化部分的主机名字段来源于环境中的不同节点,因此该字段会出现不同主机名,而对于日志模式提炼算法所关注的日志的非结构化部分没有影响。综上所述,该数据集上的实验结果可以证明本文方法也适用于国家高性能计算环境中日志的解析步骤。

3.5 鲁棒性分析

为了验证本文方法的鲁棒性,本节将使用本文方法LDmatch在不同种类的日志数据集上进行准确度的实验,因此本实验选择A数据集中所包含的16种不同系统和平台产生的日志进行实验。同时本文也使用基于单词匹配率算法的模式提炼方法Match和Lmatch进行对比。最终的实验结果如表1所示。

由表1可以看出,Match方法在所有数据集上得到的准确度都相对较低,这说明一对一的字符匹配算法在日志模式提炼上并没有优势,也证明了日志中常量部分的位置通常情况下并不相同。Lmatch的精确度比Match整体高出很多,说明了最长公共子序列在日志模式提炼中的应用价值是很高的。而本文方法LDmatch达到了最高的精度,说明基于哈希表的存储结构不仅优化了日志的匹配效率,还在一定程度上对日志模式提炼准确度有促进作用。表格中还给出了LDmatch在不同种类日志数据集下的最优参数以供参考。可以看出,同一种日志模式提炼算法在不同数据集下的最优参数差别比较大,这不仅说明了不同种类的日志在常量部分和变量部分的比重差别较大,也说明了调整阈值的参数对于日志模式提炼算法具有重要作用。

表1 不同种类日志上的实验结果

Table 1 Experimental results on different kinds of logs

日志种类	LDmatch最优阈值	准确度		
		LDmatch	Lmatch	Match
Andriod	0.90	0.912	0.912	0.689
Apache	0.55	1.000	1.000	1.000
BGL	0.50	0.845	0.836	0.645
Hadoop	0.40	0.881	0.877	0.871
HDFS	0.60	0.841	0.841	0.841
HealthApp	0.30	0.622	0.557	0.628
HPC	0.60	0.906	0.904	0.895
Linux	0.45	0.610	0.599	0.531
Mac	0.60	0.858	0.846	0.510
OpenSSH	0.75	0.678	0.678	0.373
OpenStack	0.70	0.838	0.807	0.733
Proxifier	0.70	0.517	0.517	0.517
Spark	0.25	0.920	0.718	0.599
Thunderbird	0.55	0.916	0.922	0.825
Windows	0.70	0.993	0.990	0.705
Zookeeper	0.60	0.964	0.964	0.894
平均值	—	**0.831**	0.810	0.703

3.6 效率分析

为了验证本文方法LDmatch在效率上的优势,本文使用高性能计算环境中实际产生的日志数据集B进行本轮实验。首先将B数据集拆分成1 000、5 000、10 000、50 000、100 000条日志,然后分别在这些不同大小的日志上进行日志模式提炼的实验。作为对比,本文还将Match以及Lmatch算法进行了相同的实验,最后记录每组实验的完成时间,实验结果见图6。

由图6可以看出,当日志数量比较小的时候,三种算法在进行日志模式提炼时所用的时间差别不大。但是随着日志规模的增大,Match算法所用的时间显著上升,说明该算法在处理较多的日志时得到的模式增加较快,因此每次比较的次数增多而导致消耗的时间快速上升。而Lmatch算法相比来说时间增加得缓慢,说明了基于最长公共子序列的模式提炼算法对日志进行模式提炼的合理性。而LDmatch算

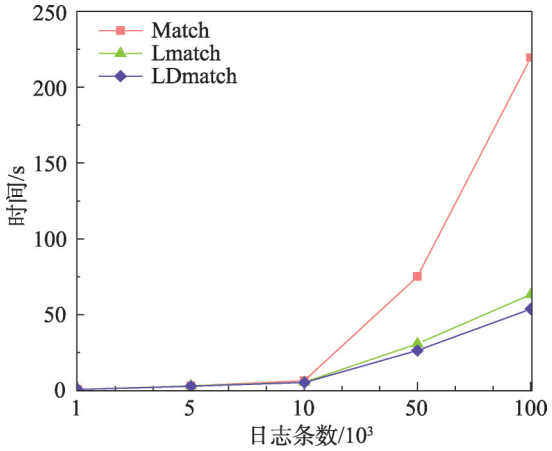

图6 不同日志数量下模式提炼方法消耗的时间

Fig.6 Time consumed by pattern extracting methods under different number of logs

法所用的时间最短，因为LDmatch算法的时间复杂度是 $O((cm_1m_2)n)$，其中 c 是当前搜索到的哈希表节点内所包含的日志模板的数量。m_1 和 m_2 分别代表进行匹配的两条日志的单词数量。很明显，其中的 m_1、m_2 是常量。c 相对整体算法进行时日志条数的增长来说也是常量，因为模板的数量本身就远小于日志的总量，同时又经过哈希表的划分，每一个字母键下存储的模板数量就更少了。通过以上讨论，可以看出LDmatch算法具有 $O(n)$ 级别的时间复杂度。上述讨论和实验结果也说明了引入了哈希表的模式提炼算法虽然增加了日志模式仓库存储的复杂性，但是在效率上具有较大的优势。

4 结束语

本文主要对日志模式提炼算法进行了分析和研究，对以往的模式提炼算法进行了改进，在日志的匹配方式上和匹配规则上进行了优化，最后通过多组实验验证了本文方法在准确度、鲁棒性和效率上的提升。本文只是针对日志模式提炼做了一些前期探索工作，未来还有很多值得关注的研究点。今后工作的主要重点是将该日志模式提炼算法应用到国家高性能计算环境中的日志流量分析中，从而进行异常日志流量检测以及日志类型序列的关联性分析等。

参考文献：

[1] 赵一宁, 肖海力. 网格环境日志分析框架LARGE的设计[J]. 科研信息化技术与应用, 2016, 7(3): 3-7.
ZHAO Y N, XIAO H L. The design of LARGE: log analyzing framework in grid environment[J]. E-science Technology & Application, 2016, 7(3): 3-7.

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

[3] LOU J G, FU Q, YANG S Q, et al. Mining invariants from console logs for system problem detection[C]//Proceedings of the 2010 USENIX Annual Technical Conference, Boston, Jun 23-25, 2010. Berkeley: USENIX Association, 2010: 231-244.

[4] LIN Q W, ZHANG H Y, LOU J G, et al. Log clustering based problem identification for online service systems[C]//Proceedings of the 38th International Conference on Software Engineering Companion, Austin, May 14-22, 2016. Piscataway: IEEE, 2016: 102-111.

[5] DU M, LI F F, ZHENG G N, et al. DeepLog: anomaly detection and diagnosis from system logs through deep learning[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Oct 30-Nov 3, 2017. New York: ACM, 2017: 1285-1298.

[6] ZHAO Y N, XIAO H L. Extracting log patterns from system logs in LARGE[C]//Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops, Chicago, May 23-27, 2016. Washington: IEEE Computer Society, 2016: 1645-1652.

[7] ZHAO Y N, WANG X D, XIAO H L, et al. Improvement of log pattern extracting algorithm using text similarity[C]//Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops, Vancouver, May 21-25, 2018. Washington: IEEE Computer Society, 2018: 507-514.

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

[9] 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, Cape Town, May 2-3, 2010. Piscataway: IEEE, 2010: 114-117.

[10] FU Q, LOU J G, WANG Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis [C]//Proceedings of the 9th IEEE International Conference on Data Mining, Miami Beach, Dec 6-9, 2009. Piscataway: IEEE, 2009: 149-158.

[11] TANG L, LI T, PERNG C S. LogSig: generating system events from raw textual logs[C]//Proceedings of the 20th ACM Conference on Information and Knowledge Management, Oct 2011. New York: ACM, 2011: 785-794.

[12] MIZUTANI M. Incremental mining of system log format [C]//Proceedings of the 2013 IEEE International Conference on Services Computing, Santa Clara, Jun 28-Jul 3, 2013. Piscataway: IEEE, 2013: 595-602.
[13] MESSAOUDI S, PANICHELLA A, BIANCULLI D, et al. A search-based approach for accurate identification of log message formats[C]//Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension, May 2018. New York: ACM, 2018: 167-177.
[14] DAI H T, LI H, CHEN C S, et al. Logram: efficient log parsing using n-gram dictionaries[J]. IEEE Transactions on Software Engineering, 2022, 48(3): 879-892.
[15] NEDELKOSKI S, BOGATINOVSKI J, ACKER A, et al. Self-supervised log parsing[J]. arXiv:2003.07905, 2020.
[16] ZHU J M, HE S L, LIU J Y, et al. Tools and benchmarks for automated log parsing[C]//Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, Montreal, May 25-31, 2019. Piscataway: IEEE, 2019: 121-130.

王晓东(1989—),男,河南洛阳人,博士研究生,主要研究方向为日志处理分析、数据挖掘、机器学习。
WANG Xiaodong, born in 1989, Ph.D. candidate. His research interests include log processing and analyzing, data mining and machine learning.

赵一宁(1983—),男,河北河间人,助理研究员,主要研究方向为日志处理分析、数据挖掘。
ZHAO Yining, born in 1983, research assistant. His research interests include log processing and analyzing and data mining.

肖海力(1978—),男,湖北天门人,硕士,研究员,主要研究方向为网格计算、分布式计算。
XIAO Haili, born in 1978, M.S., professor. His research interests include grid computing and distributed computing.

王小宁(1981—),女,四川资阳人,博士,副研究员,主要研究方向为网格计算、分布式计算。
WANG Xiaoning, born in 1981, Ph.D., associate professor. Her research interests include grid computing and distributed computing.

迟学斌(1963—),男,吉林梅河口人,研究员,主要研究方向为并行计算与软件、网格计算技术、高性能计算机系统维护与管理。
CHI Xuebin, born in 1963, professor. His research interests include parallel computing and software, grid computing technology, high performance computer system maintenance and management.