

数据特征感知的细粒度存储单元快速编码研究

汪毛毛, 王蕾, 周晓芳

公安部第三研究所, 上海 200120

摘要

随着工业领域日志数据规模的扩大, 其存储压力日益增大。最新的日志专用压缩方法利用日志模板将日志解析成细粒度存储单元, 再使用重量级压缩算法直接压缩处理, 进而实现对日志数据的高密压缩。然而, 这一方法因使用重量级压缩算法而存在较高的压缩延迟。通过对细粒度存储单元中数据的观察和实验分析, 发现这些数据存在局部递增和局部重复的特征。基于这些数据特征, 提出了一种数据特征区间感知的编码方法。该方法利用和局部数据特征相契合的轻量级编码来处理细粒度存储单元的数据, 再使用重量级压缩算法对编码后的数据进行压缩处理。实验表明, 在Entry类数据中, 局部特征感知编码的压缩率接近全局RLE编码、略低于全局Delta编码, 而平均压缩速度提升了38%。与直接使用重量级压缩方法相比, 该方法通过轻量级编码减少了冗余数据, 进而降低整体压缩延迟。

关键词

细粒度存储单元; 局部重复; 局部单调; 高压缩率; 压缩延迟

中图分类号: TP333.4; TP301.6

文献标志码: A

doi: 10.11959/j.issn.2096-0271.2026040

Feature-aware fast encoding for fine-grained log storage units

Wang Maomao, Wang Lei, Zhou Xiaofang

The Third Research Institute of the Ministry of Public Security, Shanghai 200120, China

Abstract

With the continuous expansion of log data in various industrial fields, the pressure of storing these data has become increasingly significant. The latest log-specific compression method utilizes log templates to parse logs into fine-grained storage units and applies heavyweight compression algorithms directly to the data, thereby reducing the storage cost of logs. However, this compression process introduces higher compression latency. Through observation and experimental analysis of the data in the fine-grained storage units, some typical characteristics, such as local increment and local repetition, are discovered. Based on these features, a feature-interval-aware encoding method is proposed. This method applies lightweight encoding that aligns with the data features to process the data in the fine-grained storage units, followed by compression using heavyweight algorithms. The experimental results show that, for Entry-type data, the compression ratio of the local feature-aware encoding is close to that of global RLE and slightly lower than that of global Delta, while achieving a 38% improvement in average compression speed. The advantage of this method lies in its effective utilization of data features, reducing redundant data through lightweight encoding, and lowering overall

compression latency.

Key words

fine-grained storage units, local repetitiveness, local monotonicity, high compression ratio, compression latency

0 引言

随着工业领域数字化和智能化水平不断提高,日志数据在工业系统中的作用日益凸显。日志数据能够如实记录系统的运行状态、各类事件的发生时间和细节、关键参数的实时数值等。分析日志数据可以实现对系统的异常检测^[1-5]、故障诊断^[6-9]、安全检测^[10-11]和用户画像构建^[12-13]等功能。尤其是在对安全性要求极高的行业(如航空、医疗服务、金融领域等)需要持续记录相关系统运行的日志信息^[14-15],并利用日志数据中的大量数值信息^[16-18]检测系统中的异常行为,预测关键指标趋势,评估可能风险,以便及时采取补救措施。另一方面,相关法律法规要求将日志保存一段相对较长的时间以满足归档检查的需求。大量日志数据增加了存储压力,而采取高压缩比的方法^[19-26]对日志数据进行压缩可显著降低存储成本^[27]。

PCCTSMS^[28]根据数据所处的系统层级与生命周期阶段采用差异化的压缩策略。而最新的日志专用压缩方法(如Log-Reducer^[29]、LogGrep^[30]和CLP^[31])通过将日志数据拆分成细粒度存储单元,再使用重量级的压缩算法进行压缩。其中,LogGrep对细粒度存储单元直接进行压缩,LogReducer和CLP先对细粒度存储单元进行轻量级编码,再使用重量级的压缩算法(如gzip^[32]、LZMA^[33]和Zstd^[34]等)对这些存储单元进行压缩。然而,这两类方法在压缩过程中都会出现压缩延迟过高的问题,

尤其在处理大规模数据集时压缩延迟更明显,从而影响任务完成的速度和效率。

通过对细粒度存储单元中数据的观察,发现这些数据存在两种的不同的分布特征——局部重复和局部递增。基于这两种分布特征,提出了数据特征感知的编码方法:(1)使用滑动窗口对细粒度存储单元中的数据进行扫描以识别特征区间;(2)根据这两种分布特征,找到满足窗口大小的连续局部分布特征区间,并记录每个特征区间的元数据信息,包括起始索引位置、特征区间种类(局部重复特征区间或局部递增特征区间)以及区间的长度等;(3)在编码过程中,利用特征区间的元数据信息选择合适的轻量级编码方法,按照区间的起始索引位置和长度进行局部编码;(4)使用重量级压缩算法对编码后的数据进行压缩。该方法不仅保证了对日志数据的高压缩率,还提升了整体编码压缩过程的效率,降低了压缩延迟。

在16种公开日志数据集上使用提出的编码方法并结合3种重量级的压缩算法对日志数据进行压缩,测试对应的压缩率和压缩延迟。实验结果表明,利用数据特征感知的编码压缩方法对Entry类细粒度存储单元的压缩率与全局编码压缩方法的压缩率基本保持一致,而平均压缩速度提升37%;对Svar类细粒度存储单元的压缩率和平均压缩速度由于其数据自身的特点低于预期。

1 编码方法与细粒度存储单元

首先介绍几类典型的轻量级编码方法,

其次介绍基于模式的细粒度存储单元提取过程，最后介绍对细粒度存储单元中数据的分布特征。

1.1 轻量级编码方法

通过识别细粒度存储单元中数据的分布特征，选择合适的轻量级编码方法进行编码压缩，能够有效降低压缩延迟。常用的轻量级编码算法包括 Delta 编码^[35]、游程编码（run-length encoding，RLE）编码^[36]、Elastic^[37-38]编码和可变长度编码（variable-length quantity，VLQ）编码^[39]，这些轻量级编码算法被广泛应用于数值型日志数据的压缩。

Delta 编码是一种基于数值差异的压缩算法，通过对每个数值与前一个数值的差值进行编码并存储来减少数据冗余。在解码时，通过将差值与前一个数值进行累加的方式还原原始的数值序列。图 1 展示了 Delta 编码过程，首先保留序列的第一个值，然后计算后续数值与前一个数值的差值生成差值序列。当数据存在一定规律或递增性时，Delta 编码可以有效提高压缩效率。

RLE 编码的基本思想是将连续重复的数据表示为一个计数器和单个数据项。如图 2 所示，RLE 编码后的数值序列中的前两个数值表示数值 2 出现了 3 次。RLE 编码的压缩速度快，适用于存在大量重复数据的数据序列，但对随机分布的数据的压缩效果不明显，有时编码后的数据序列比原始数据序列更长。

Elastic 编码是一种根据数值分布特征动态调整编码长度的压缩编码方法。其基本思想是将数据按照固定长度分块，使用变长编码对每块中的整数进行编码。Elastic 编码的优势在于压缩效率高，无需

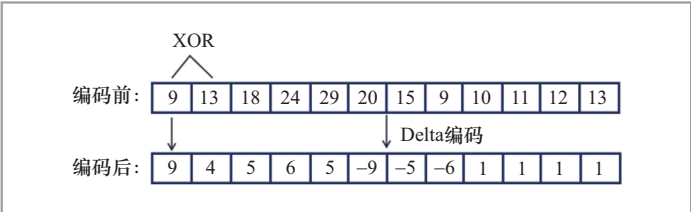


图1 Delta编码示例

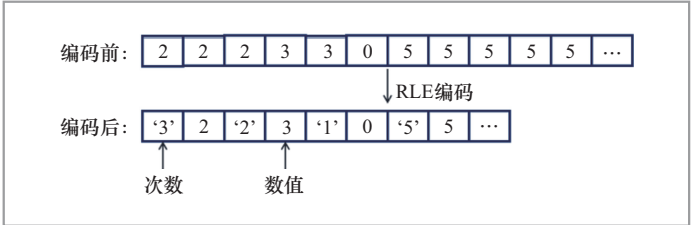


图2 RLE编码示例

预设固定编码长度，可以实现动态编码。

VLQ 编码是一种基于整数的可变长度编码方法，其基本原理是将整数按照 7 位二进制数作为一个编码单元。对于每个编码单元，如果最高位是 0，则表示还有后续编码单元，如果最高位是 1，则表示该单元是最后一个编码单元。将每个编码单元按顺序连接起来就可以还原出完整的整数值。

如果能够根据数据的分布特征选择合适的轻量级编码方法，就可以在压缩率和压缩延迟上带来一定的收益。

1.2 细粒度存储单元提取过程

基于模式的细粒度存储单元提取过程首先采集部分日志样本（Sampler）并利用解析树^[40]进行解析，从而得到日志的模板（Templates）和变量值^[41-44]，再利用模板将日志进行解析成细粒度存储单元。而这些细粒度存储单元分为两大类——Entry 类和 Svar 类。

现有方法通常通过设置重复阈值来判断变量是否属于字典变量。当某变量值的重复次数超过预设阈值时，该变量值被识别为字典变量。如图3所示，模板中“Write chunk<*>Offset<*>”和“Read chunk<*>Offset<*>”中的第一个“<*>”（变量值）可被识别为字典变量，将字典变量对应的索引值所构成的细粒度存储单元作为Entry类。对于非字典变量而言，现有方法直接采用其原始值，以Svar类形式组织。

Entry类和Svar类细粒度存储单元反映了不同的语义信息。Entry类由原始日志样本中被识别为字典变量的索引值组成，反映了字典变量在日志样本中的实际出现顺序，保留了字典变量的重复出现规律。Entry类用整数表示，且存储的数值较小。Svar类由非字典变量的原始值构成，存储的是日志样本中该变量值的原始值序列。这些原始值序列的分布较为随机，无显著重复规律，并且Svar类存储的数值较大。

1.3 细粒度存储单元中数据的分布特征

通过对细粒度存储单元中数据的观察，

发现这些数据存在两种不同的分布特征——局部递增性和局部重复性。

特征1：局部递增性。递增性是数值型日志数据的一种典型分布特征。例如，在系统日志中，用户某种行为的次数随着其操作次数的增加而单调递增，这些日志数据呈现出递增的分布规律。

特征2：局部重复性。重复性指在连续的时间段内，日志数据中的某个特定值或数据模式重复出现。这种重复性数据可以是各种类型的，如事件发生次数、错误码、用户请求等。这种连续重复的数据序列对编码压缩非常有利。

然而，在整体上，细粒度存储单元中的数据未呈现出这两种分布特征，直接进行编码压缩将导致较高的压缩延迟；在局部范围内，细粒度存储单元中的数据存在这两类分布特征，如图4、图5所示，Entry类细粒度存储单元中的数据表现出局部重复性，Svar类细粒度存储单元中的数据表现出局部递增性。这种局部重复性的分布特征可能源于用户对某个事件的多次请求，而局部递增趋势可能反映日志中操作频率或事件发生次数的累计效应。

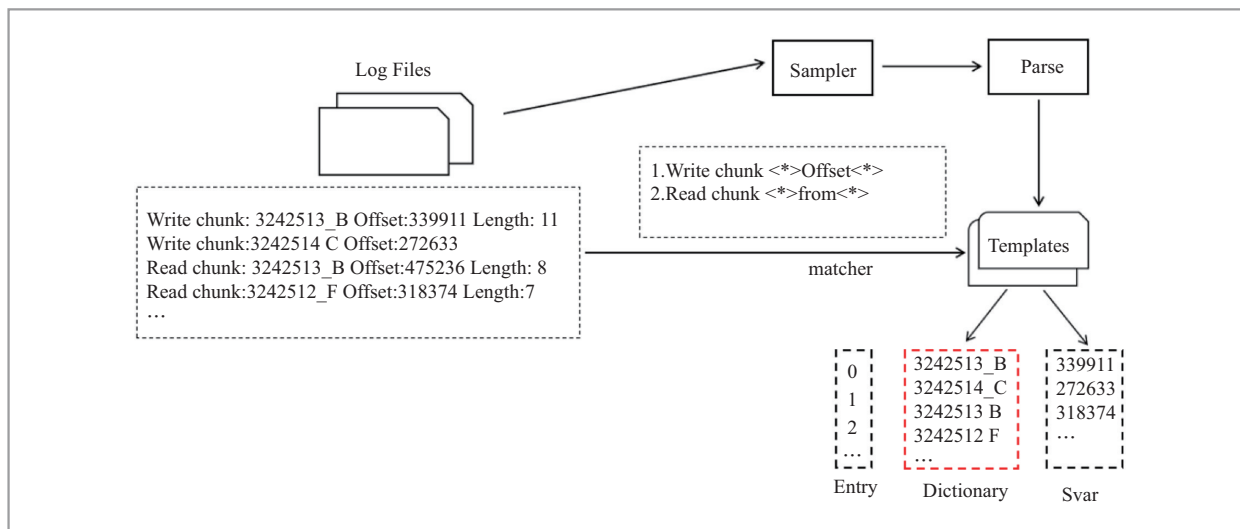


图3 细粒度存储单元提取过程

2 局部特征区间识别

结合细粒度存储单元中的数据呈现出随机性的分布特点，设定滑动窗口的阈值大小为7，利用滑动窗口依次对数据进行扫描，当找到满足阈值的局部特征区间时，记录该区间的元数据（Metadata），包括该区间类型（Type，Type=0表示为局部重复特征区间，Type=1表示为局部递增特征区间）、起始索引（S_index）和区间长度（Length），持续扫描数据并不断更新该局部特征区间的长度，直到下个数据不满足该区间的分布特征。

2.1 局部特征区间识别算法

局部特征区间识别算法利用滑动窗口识别出数据中局部递增特征区间和局部重复特征区间，以局部递增特征区间为例，具体的识别过程如图6所示。

（1）对细粒度存储单元中的数据“...1 2 3 4 5 6 7 0 5 ...”进行读取，先记录第一个值“1”，再利用滑动窗口扫描下一个值，当前值大于前一个值，设置该区间类型Type:1，这意味该区间可能为局部递增特征区间，记录第一个值的S_index:0和Length:1；

（2）继续利用滑动窗口向右扫描下一个值，并不断更新区间长度，最终Length:7；

（3）当扫描到数值0时，发现此时不符合递增关系，判断此时区间长度与阈值的大小关系，当区间长度大于或等于阈值时，存储当前区间的元数据信息（Type:1, S_index:0, Length:7）。

同理，局部重复特征区间的识别过程

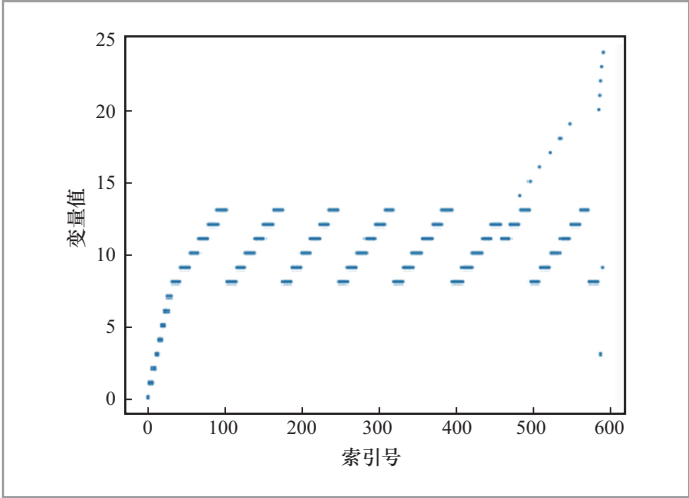


图4 Entry类中数据的局部重复性

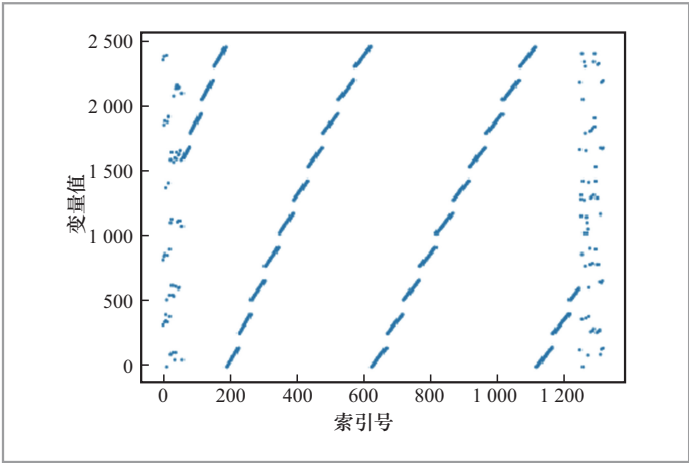


图5 Svar类中数据的局部递增性

设置区间类型Type:0，发现数据“0 2 5 5 5 5 5 5 5 5 6 ...”中存在局部重复区间，存储该区间的元数据信息（Type:0, S_index:2, Length:10）。

局部特征区间识别算法将数据的分布规律由整体转向局部，找到数据中的符合区间大小的局部特征区间。将随机性数据转换成具有局部递增和局部重复的数据，再结合相应的轻量级编码方法进行编码压缩，从而提高压缩率和降低压缩延迟。

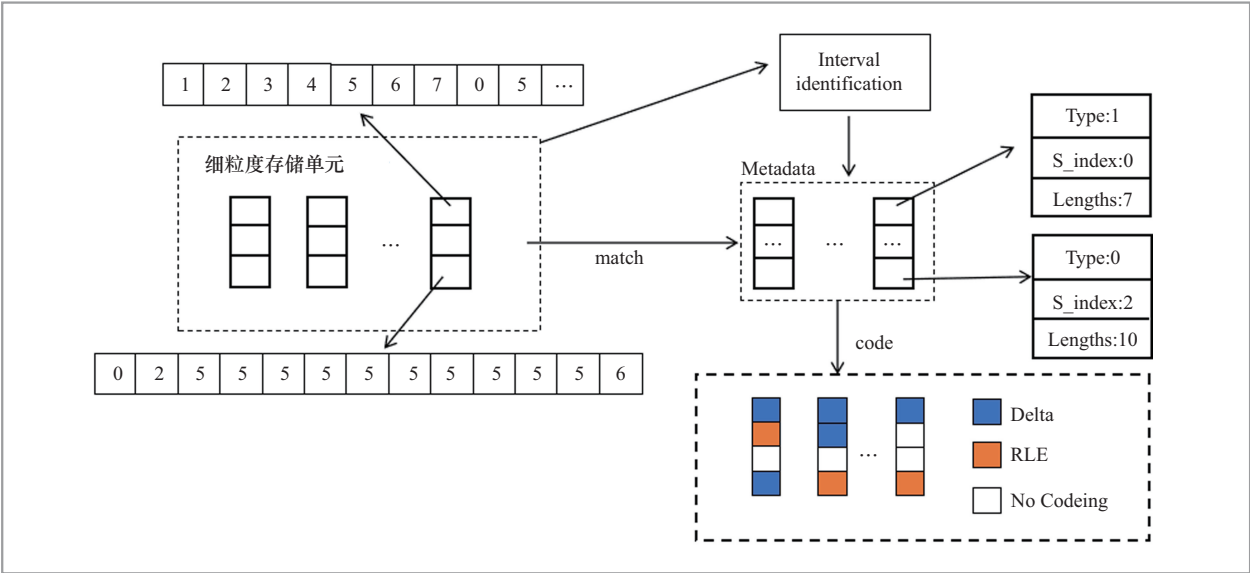


图6 整体编码方案流程

2.2 局部特征区间的识别情况

为了探究局部特征区间对于编码的可行性，收集了37种日志数据集样本，包括21种生产日志数据集样本(为了保护用户数据隐私，对生产数据集样本采用匿名代号进行表示)和16种公开日志数据集样本。设定滑动窗口的阈值大小为7，连续读取数据，当连续数据的递增区间和重复区间的长度大于或等于7，记录该区间的长度，

观察局部递增特征区间和局部重复特征区间在数据整体中的占比情况。对每种日志的Entry类和Svar类数据进行统计分析，局部重复区间占比和局部递增区间占比如图7和图8所示，结果如下：

- (1) 在Entry类数据中，32种日志数据的局部重复特征区间占比超过40%；
- (2) 在Svar类数据中，8种日志数据的特征区间长度占比超过40%，其中LogS类日志的局部递增特征区间长度占比达到

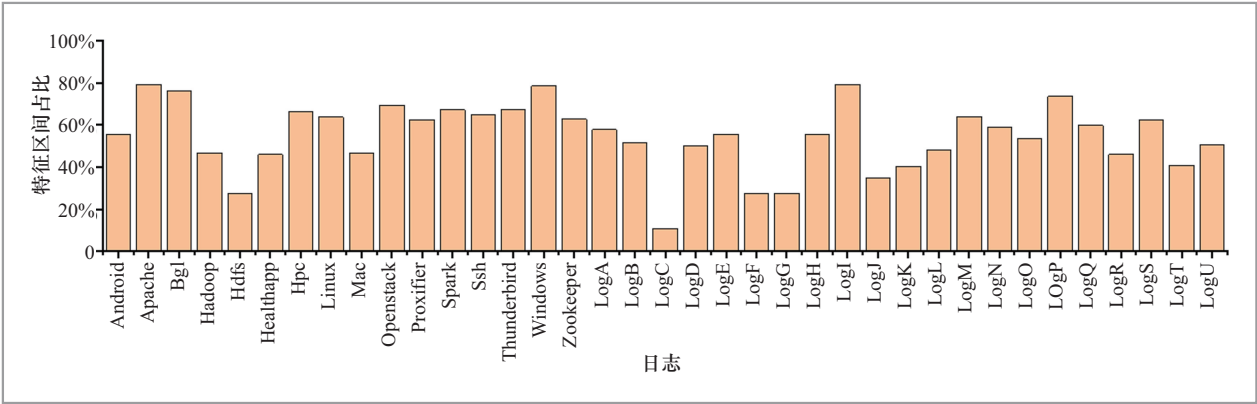


图7 Entry类中局部重复区间占比

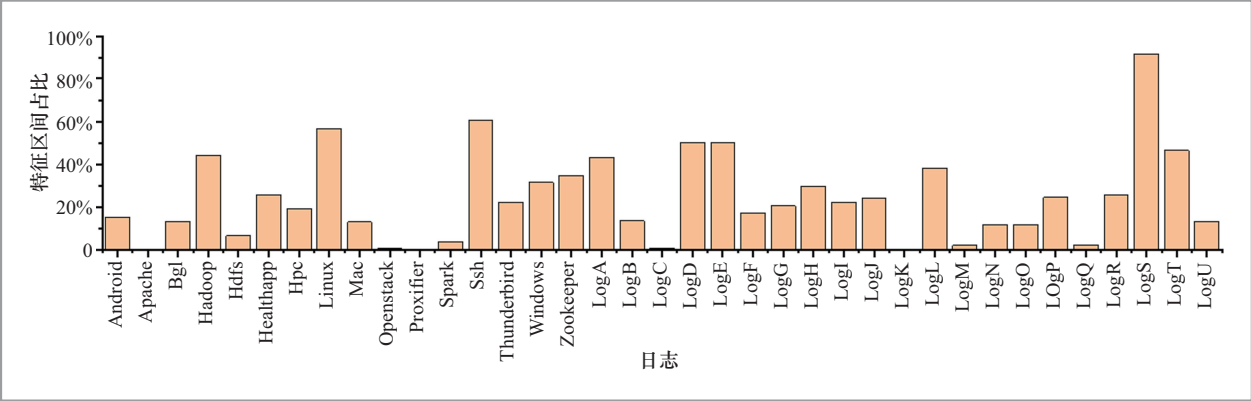


图8 Svar类中局部递增区间占比

91.94%。显著高于其他类型日志。

3 系统设计

本文面向日志数据中细粒度信息的高效存储需求，提出了一种基于数据特征感知的编码策略，旨在实现高压缩率与低编解码开销的协同优化。整体方法遵循特征识别、分类建模、自适应编码和协同压缩的技术路线，核心思想是通过 对日志中的整型数据进行细粒度语义分析，识别其访问模式与数值分布特征，进而划分为不同类型的存储单元，并采用差异化的编码策略进行精准压缩。

本文所提出的编码策略在预处理阶段对原始日志进行结构化解析，提取其中的整型字段并统计其出现的频次和检测变化趋势；依据设定的重复阈值，将高频、稳定取值的数据归类为 Entry 类，通过静态字典映射实现高效复用；对于低频、连续变化或离散性强的数值，则划分为 Svar 类。在此基础上，引入特征区间感知的策略，根据数值范围、差值特性等动态选择最优编码模式。

系统的总体架构如图 9 所示，主要包

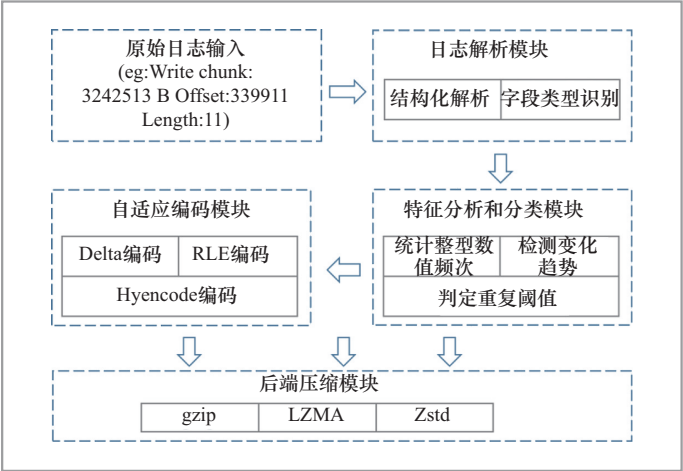


图9 系统总体架构

括四大模块：

- (1) 日志解析模块：负责日志文本的词法分析与结构化字段抽取；
- (2) 特征分析和分类模块：统计整型数据的频次，检测分布与局部趋势，完成 Entry 类和 Svar 类的自动划分；
- (3) 自适应编码模块：针对不同类型的数据采用相应编码策略，输出二进制流；
- (4) 后端压缩模块：利用集成通用压缩算法对编码后数据进行二次压缩，进一步消除数据冗余。

通过对采集得到的日志样本的观察，

发现Entry类日志呈现局部重复性特征,而Svar类日志则表现出局部递增性特征。基于这两类差异化的分布模式,分别采用RLE编码和Delta编码进行针对性压缩编码。

3.1 Hyencode 编码方法

为了充分发挥局部特征区间在编码方法中的作用,本文提出了混合特征区间编码(Hyencode编码)和单一特征区间编码(Part_Delta编码和Part_RLE编码)。与单一特征区间编码相比,Hyencode编码能够充分利用数据中的这两种不同的局部特征区间进行编码,提高日志数据的压缩率,具体的做法如下:

(1) 读取需要编码的日志数据,记录每个读取到的数据的索引号,索引号大于元数据中的索引号(S_{index})表示该数值位于局部特征区间内,进行编码选择;

(2) 检查该元数据中的区间类型(Type)的值,若Type值为0,表示该区间为局部重复特征区间,则根据区间长度(Length)进行RLE编码;若Type值为1,则进行Delta编码;

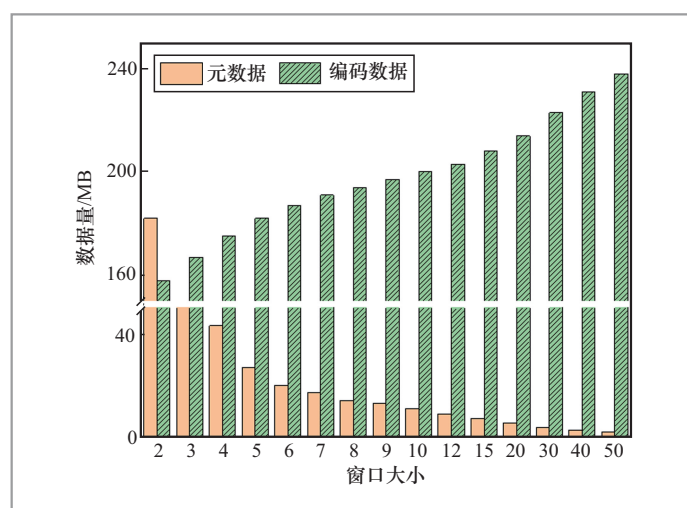


图10 窗口大小对元数据和编码数据数据量的影响

(3) 当读取数据的索引号不属于元数据中的索引号范围时,表明该数据无需编码,可直接写入数据。继续读取下一个数据,直至完成对所有数据的编码。

与编码前的日志数据相比,Hyencode编码利用特征区间的元数据信息对日志数据进行编码的压缩率提高了44.9%。这意味着Hyencode编码能够显著减少日志数据的数据量,从而节省存储空间,这一结果显示了Hyencode编码在数据压缩方面的有效性和潜力。

3.2 滑动窗口阈值大小设定

利用设定好阈值大小的滑动窗口扫描原始日志数据,得到局部特征区间的元数据信息。窗口的阈值大小影响特征区间的识别。如果滑动窗口的阈值过大,可能导致部分特征区间无法被准确识别,进而影响利用特征区间进行编码压缩的效果。相反,如果滑动窗口的阈值太小,可能会引入过多的元数据,占用更多的存储空间。因此,在选择窗口大小时需要权衡考虑,找到最合适的特征区间进行编码压缩。

为了研究窗口大小对元数据和编码数据数据量的影响,在37种日志数据集样本上进行验证。如图10所示,随着窗口大小的不断增大,元数据数据量持续减少。这是由于随着窗口大小的增大,满足该窗口大小的特征区间减少,需要存储的元数据信息减少。而随着窗口大小的增大,编码数据的数据量不断增大。这是因为无法获得更多的特征区间,局部编码的效果也逐渐下降。

根据实验结果可知,当滑动窗口的大小在[5,7]范围内时,元数据和编码数据的数据总量较小。当窗口大小设置为7时,元数据的数据量最小,并且读取元数据所

需的时间最短。因此，在实验中，选择了7作为近似最优的窗口阈值大小。

3.3 元数据的管理与存储

本文还对如何对元数据进行管理和存储、对元数据的读取时间是否会影响编码效率进行了额外的讨论。

在对细粒度存储单元中的数据进行局部特征区间提取后，得到了关于局部特征区间的元数据信息。该元数据由三部分组成：区间类型（Type）、起始索引（S_index）和区间长度（Length）。将元数据以三元组的形式进行存储，利用元数据可进行有效的编码和解码。

元数据的数据量和存储方式影响了编码效率。为了减少元数据的数据量和提高对元数据的访问速度，使用VLQ编码方法对元数据进行编码。通过观察发现，在元数据的3种信息中，索引值的波动幅度较大，而区间类型和区间长度的变化较小。根据该特征信息，对元数据的3种不同数据信息采用变长的编码方式。

如图11所示，在不同窗口大小的滑动窗口中，经过VLQ编码后，元数据的平均数据量减少了55.8%。这意味着VLQ编码能够显著减少元数据的数据量。此外，从图12可知，经过VLQ编码后，元数据的平均读取时间减少了86%。这表明元数据经过VLQ编码后不仅能够减少数据量，还能加速对元数据的读取过程，从而提高了整个编码过程的效率。

4 实验评估

本文通过对细粒度存储单元中数据的观察发现数据内部的局部特征，并利用这

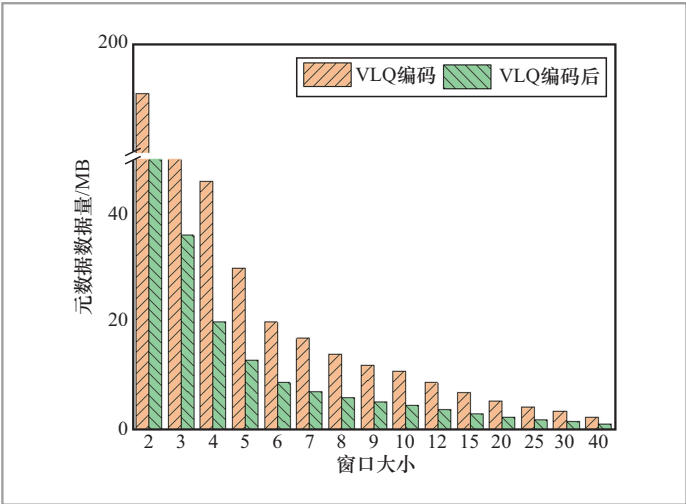


图11 VLQ编码前后的元数据数据量

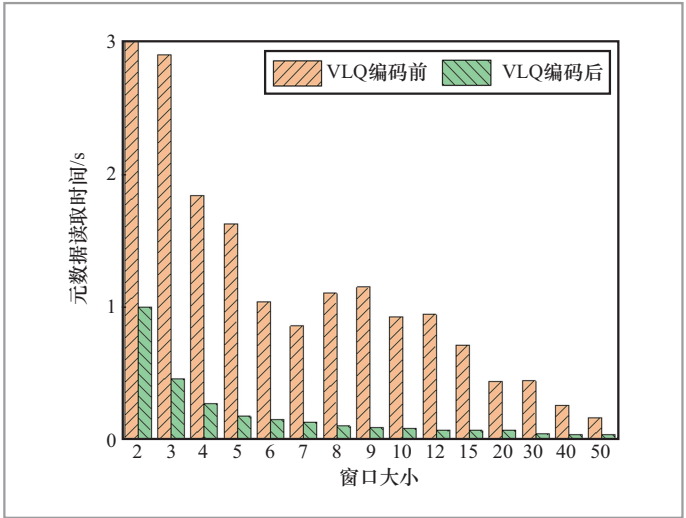


图12 VLQ编码前后的元数据读取时间

些特征信息进行轻量级编码，再结合3种不同的重量级压缩方法进行压缩。压缩算法包括gzip压缩、LZMA压缩和Zstd压缩。其中LZMA压缩算法默认选择的压缩级别为6，而Zstd压缩算法选择的压缩级别为10。

本节采用不同的编码方法对细粒度存储单元进行编码压缩，并对不同方法下的日志数据的压缩率、压缩速度和压缩延迟进行性能测试。

4.1 实验环境

本实验在一台通用高性能服务器上进行，硬件配置如表1所示。该服务器搭载两颗 Intel 处理器，每颗包含 10 个物理核心，共 20 核 40 线程。本实验所用的压缩算法均采用广泛使用的开源版本：gzip 版本为 v1.10，LZMA 版本为 v5.4.0，Zstd 版本为 v1.5.2。所有测试程序使用 C++ 编写，编译时启用 C++11 标准支持。

数据集。收集了 16 种不同平台环境下的公开日志数据集，大小为 5.8 GB，其中 Entry 类数据为 4.7 GB。

评测指标。压缩率 = 原日志数据大小 / 压缩后数据大小，压缩率越大，压缩效果越好；压缩延迟包括编码的时间和压缩的时间；压缩速度 = 压缩后数据大小 / 压缩延迟，压缩延迟越高，压缩速度越慢。

对比对象。在 16 种公开日志数据拆分的细粒度存储单元上进行相关性能测试实验。对比对象包括：（1）全局编码压缩方法（LogReducer），即对数据进行全局的轻量级编码后再结合重量级压缩算法进行压缩处理；（2）直接压缩方法（LogGrep），即对细粒度存储单元直接利用重量级压缩算法进行压缩处理。

本文提出了局部特征区间感知编码方法：Part_Delta 编码、Part_RLE 编码和 Hyencode 编码方法。利用识别出数据中的局部特征区间，根据局部特征区间的元数据信息对数据进行局部特征区间感知编

码，再结合重量级压缩算法进行压缩，测试在不同编码压缩方法下压缩率、压缩时间和压缩速度的性能变化。

4.2 细粒度存储单元的平均压缩比和平均压缩速度

基于细粒度存储单元的异构特性，将其划分为 Entry 类和 Svar 类两类存储单元。在相同实验环境下，本文以 gzip、LZMA 和 Zstd 这 3 种通用压缩算法为性能基准，系统对比以下方法的压缩效能：

（1）局部特征感知编码方法：包括基于数据块内差异的 Part_Delta、基于数据块重复数值的 Part_RLE 以及结合数据的差异和重复特征的 Hyencode；

（2）全局编码方法：以 LogReducer 为代表，直接在整个日志数据上应用 RLE 和 Delta 编码；

（3）直接压缩方法：以 LogGrep 为代表，直接对原始日志数据（Source）进行压缩，不进行预编码处理。

评估指标包括：

（1）平均压缩比：计算 16 种异构日志数据集的压缩后体积与原始体积比值的算术平均，反映了算法对多类型日志的整体压缩效率；

（2）平均压缩速度：统计 16 种日志数据在单位时间内处理的数据量（MB/s）的算术平均，表征算法的时间开销特征。

其中，实验数据集涵盖压缩后的有效载荷数据及对应的元数据信息。

Entry 类日志数据的平均压缩比如图13所示。与直接压缩的方法（LogGrep）相比，利用局部特征感知编码的 Part_Delta 编码压缩方法的平均压缩比在 gzip 中提升了 22%。相较于全局 RLE 编码方法，Hyencode 编码在 Entry 类日志数据上的平均压缩比与其表现相当。然而，相较于全局 Delta 编码方法，

表1 服务器配置情况

操作系统	Ubuntu11.3.0
内核版本	Linux 5.15.0
CPU	2 × Intel Xeon Silver 4210R CPU @ 2.40GHz (10 cores)
内存	64 GB DDR4
硬盘	希捷 2 TB 7200 转

Entry 类日志在 3 种主流压缩算法（gzip/LZMA/Zstd）作用下的 Hyencode 编码方法的平均压缩比均显著降低。

图 14 展示了 Entry 类日志数据的平均压缩速度性能对比。实验结果表明，相较于直接压缩方法（LogGrep），Hyencode 编码方法在 gzip、LZMA 和 Zstd 这 3 种压缩算法下分别实现 66.4%、67.0% 和 53.5% 的压缩速度提升；相较于全局编码方法（Delta 和 RLE 编码），Hyencode 编码方法在 gzip、LZMA 和 Zstd 压缩算法中平均压缩速度分别提升了 40.7%、34% 和 40.6%。相比较通用压缩算法，Hyencode 编码方法的整体平均压缩速度提升了 38%。

本文方法利用数据中的局部特征进行轻量级编码后再使用重量级压缩方法进行压缩。实验结果表明，对于 Entry 类日志数据而言，和直接压缩方法相比，本文方法不仅在压缩率上有所提升，还能够降低压缩过程中的延迟。

在 Svar 类日志数据上进一步评估了本文方法的压缩效果。如图 15 与图 16 所示，此类数据普遍体积较大且局部重复模式稀疏，导致基于局部特征的编码方法整体性能受限。相较于全局 Delta 编码方法，Hyencode 编码方法在这 3 种压缩算法（gzip、LZMA、Zstd）下的平均压缩比和平均压缩速度均较低。然而，相较于全局 RLE 编码方法，Hyencode 编码方法的平均压缩比分别提升了 27.7%、12.8% 和 32.8%。

上述结果表明，在 Svar 类数据上，Hyencode 编码方法与全局 RLE 编码方法相比，其压缩比显著更高；但和全局 Delta 编码相比，其压缩比和压缩速度均更低。

4.3 不同种类日志的压缩速度和压缩率

为了测试不同的编码方法对每种日志的

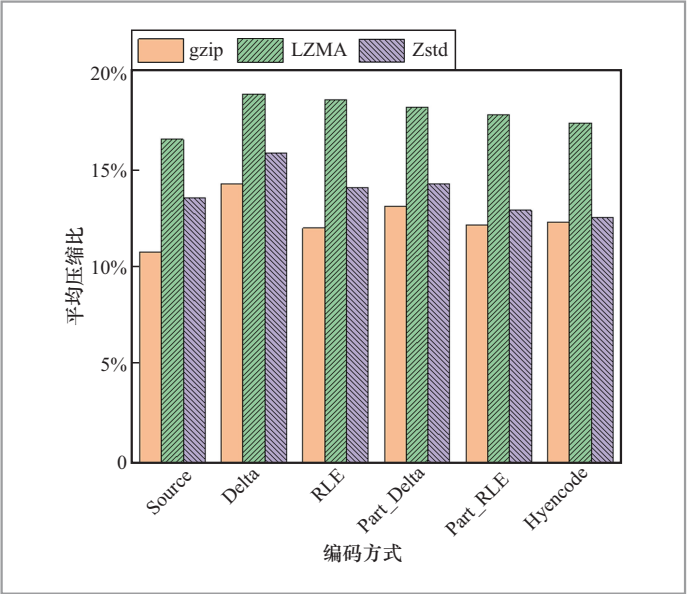


图 13 Entry 类日志数据的平均压缩比

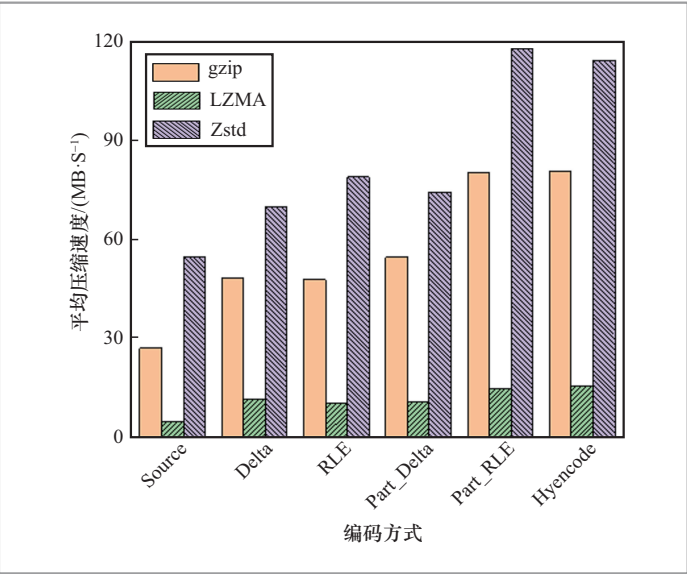


图 14 Entry 类日志数据的平均压缩速度

压缩率和压缩速度的影响，本实验测试了在 3 种不同压缩方法（gzip、LZMA、Zstd）下，使用不同的编码方法对 Entry 类细粒度存储单元中的数据进行编码压缩，测试各种日志数据的压缩率和压缩速度的性能。

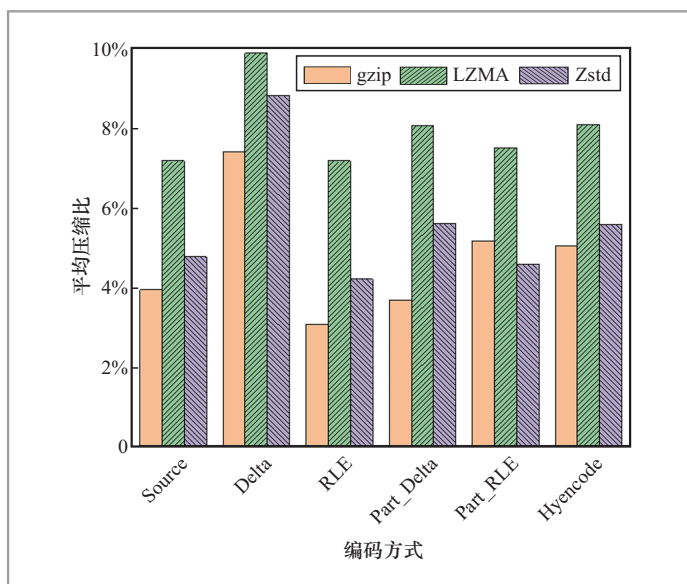


图15 Svar类日志数据的平均压缩比

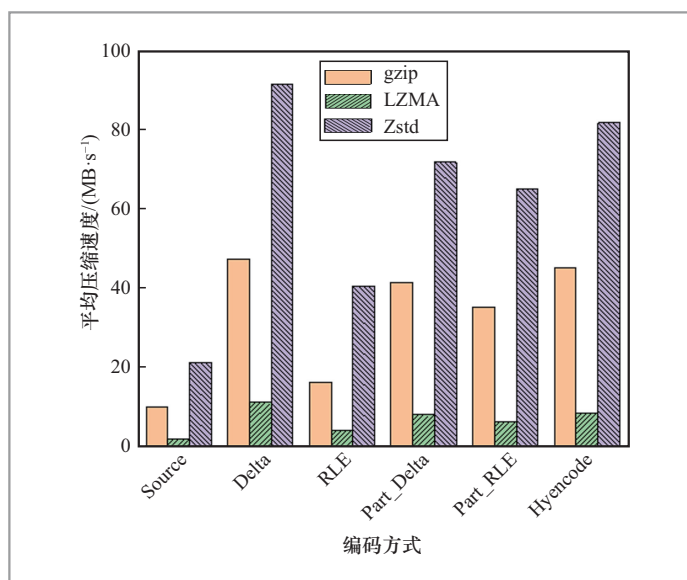


图16 Svar类日志数据的平均压缩速度

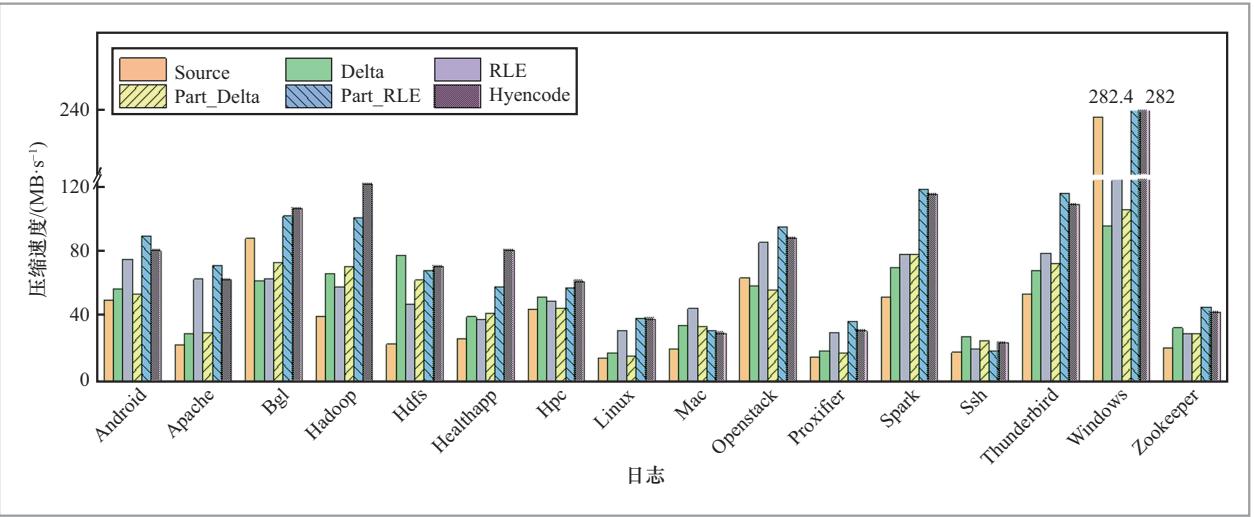
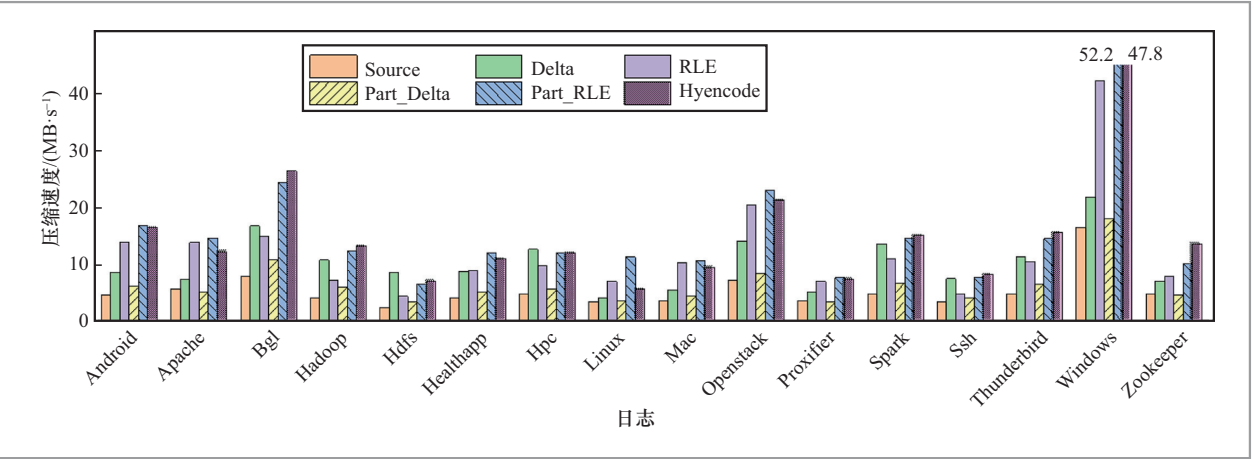
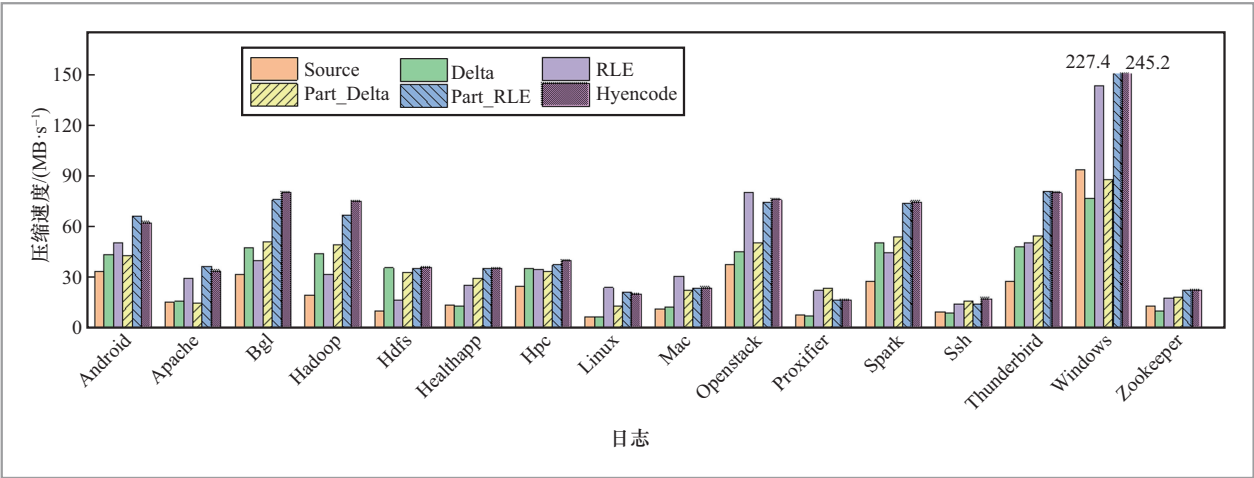
图17展示了在gzip压缩算法下不同的编码方法对各种日志数据的压缩速度的影响。实验结果表明，在全部16种日志数据集中，Hyencode编码方法相比直接使用gzip压缩算法均表现出更快的压缩速度；和全局编码压缩方法（Delta和RLE）相

比，该方法在11种日志数据集上取得了更快的压缩速度。其中，Hyencode编码方法在Windows系统日志中的压缩速度是直接gzip压缩方法的压缩速度的2.61倍，是全局Delta编码压缩方法的压缩速度的3.19倍。

进一步对比采用特征感知编码的数据在LZMA与Zstd两种压缩算法下的性能。如图18所示，在压缩速度方面，相较于全局编码方法（Delta和RLE）和直接使用LZMA压缩算法，Hyencode编码方法在11种日志数据集上表现更优，而Part_RLE编码方法在14种数据集上取得更快的压缩速度。图19的结果进一步表明，与全局编码方法及直接使用Zstd压缩算法相比，Hyencode编码方法在12种数据集上的压缩速度更高，Part_RLE则在13种数据集上表现更好。其中，在基于LZMA压缩算法的测试中，Windows类日志表现最为显著。与直接LZMA压缩方法（Source）相比，Hyencode编码方法的压缩速度是前者的3.33倍；与全局Delta编码方法相比，Hyencode编码方法的压缩速度是前者的2.70倍。

为了更好地测试编码方法对压缩率的影响，测试Entry类的细粒度存储单元在3种不同编码压缩方法（gzip、LZMA、Zstd）下的压缩率。如图20所示，Hyencode编码方法整体优于其他对比方法。相较于直接使用gzip压缩算法，Hyencode编码方法在16种日志数据集上均取得了更高的压缩率；与全局RLE编码方法相比，Hyencode编码方法在16种日志数据集上表现更优；而与全局Delta编码方法相比，Hyencode编码方法在多数日志数据集上展现出更高的压缩率。

图21和图22分别展示了各类Entry日志数据集在LZMA和Zstd压缩算法下的压



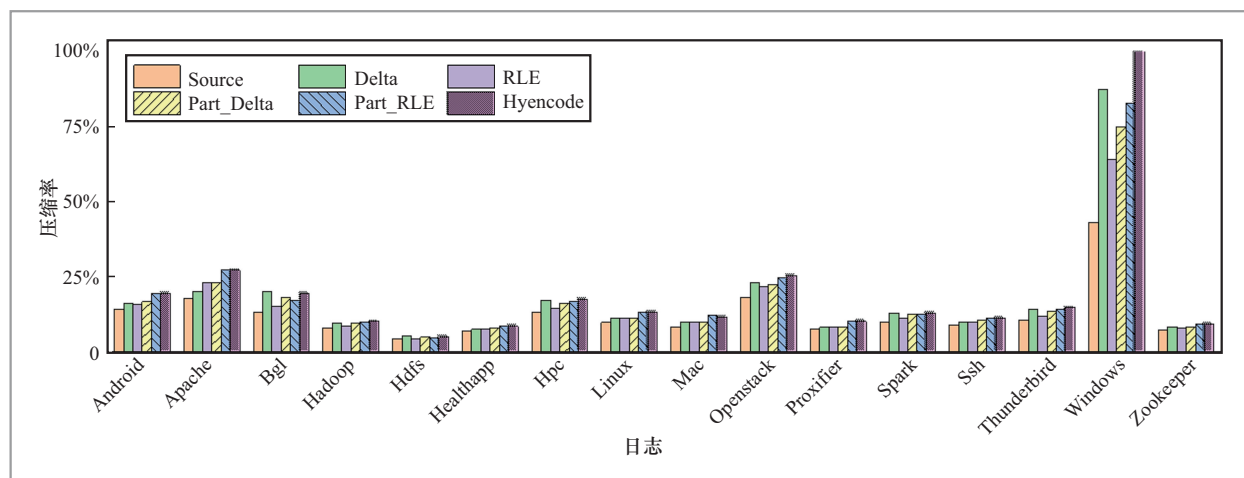


图20 在gzip压缩下各种Entry类日志的压缩率

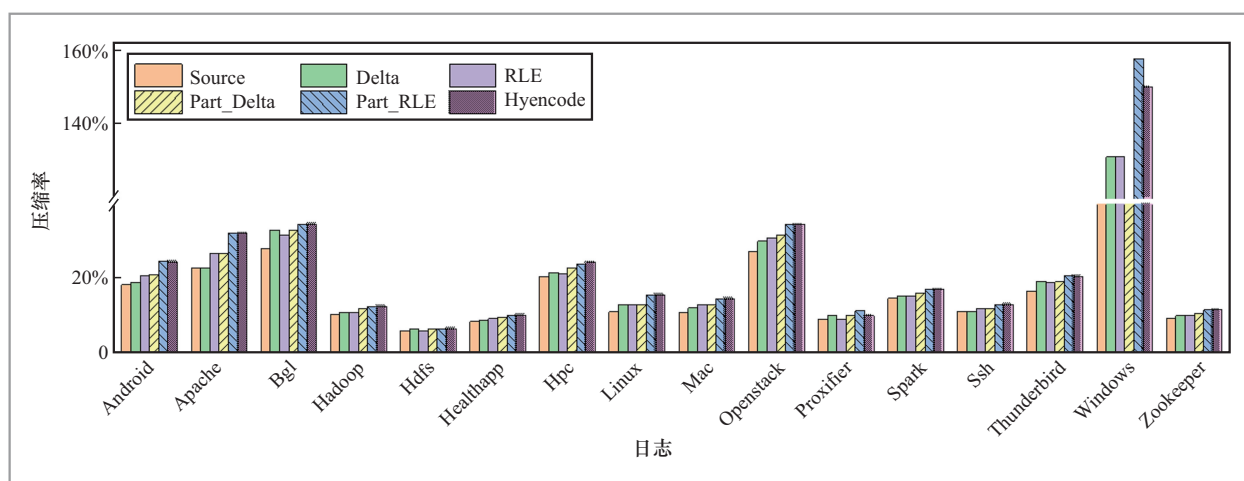


图21 在LZMA压缩下各种Entry类日志的压缩率

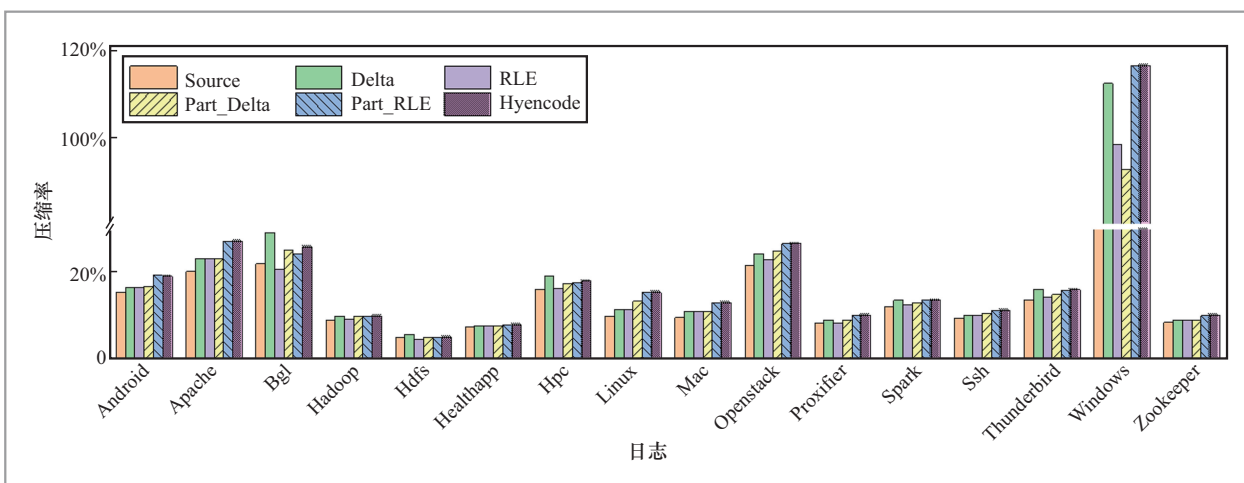


图22 在Zstd压缩下各种Entry类日志的压缩率

缩率。在 LZMA 压缩中，Hyencode 编码方法和 Part_RLE 编码方法的压缩率均高于直接压缩方法 Source、全局 Delta 编码方法和全局 RLE 编码方法。图 22 进一步表明，Hyencode 编码方法在 15 种日志数据集上的压缩率高于直接使用 Zstd 压缩算法（Source），在 12 种日志数据集上高于全局编码方法（Delta 和 RLE）；Part_RLE 编码方法则在 11 种日志数据集上的压缩率高于全局编码方法（Delta 和 RLE）；而 Part_Delta 编码方法在 14 种日志数据集上的表现优于直接使用 Zstd 压缩算法（Source）。

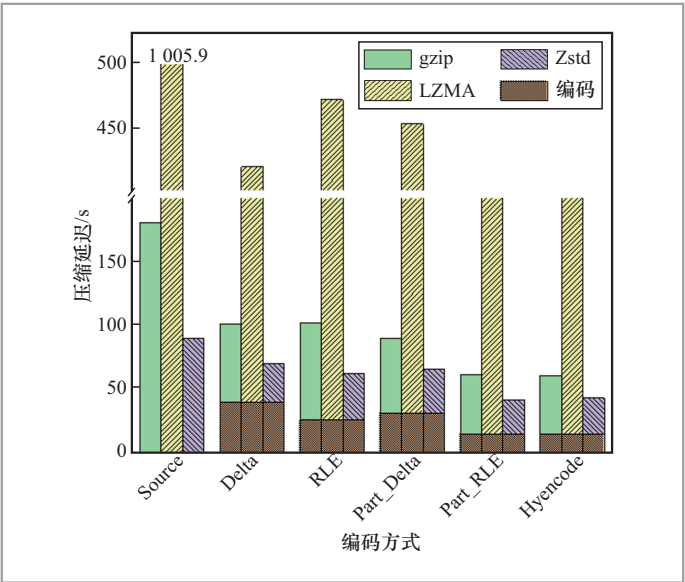


图 23 Entry 类日志的压缩延迟

4.4 细粒度存储单元的压缩延迟

为了探究不同编码方法对压缩时间的影响，对细粒度存储单元（Entry 类和 Svar 类）在编码阶段和压缩阶段所需的时间分别进行了统计。如图 23 所示，和直接压缩方法相比，Hyencode 编码方法的在 3 种不同的压缩方法下的压缩延迟分别减少了 60%、70% 和 52%；和全局 RLE 编码方法相比，Hyencode 编码方法的在 3 种不同的压缩算法下的压缩延迟分别减少了 40%、31% 和 38%。

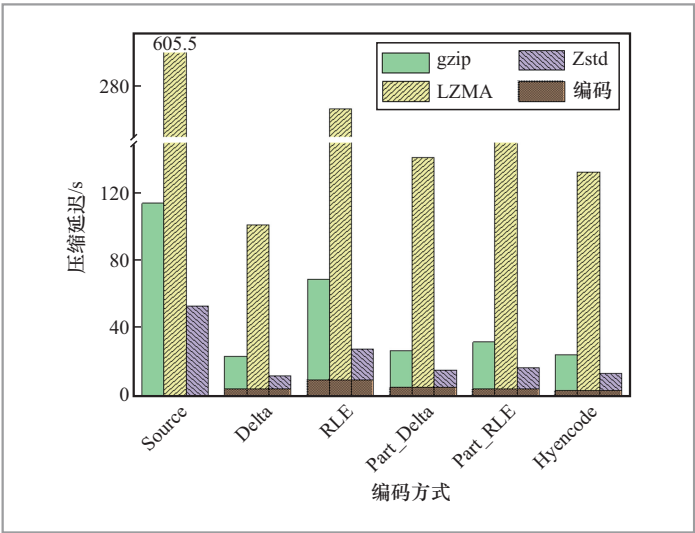


图 24 Svar 类日志的压缩延迟

图 24 展示了 Svar 类日志数据集的压缩延迟。实验结果表明，相比直接使用 gzip 压缩算法（Source），Hyencode 编码方法的压缩延迟仅为前者的 22.3%；和全局 RLE 编码方法相比，Hyencode 编码方法在 gzip 和 LZMA 压缩算法下的压缩延迟约降至前者的 39.4% 与 50.3%。而与全局 Delta 编码方法相比，由于 Svar 类日志数据的数值大且局部重复区间较少，导致局部编码收益有限，Hyencode 编码方法在 LMZA 压缩过程中的压缩延迟反而有所增加。

5 结束语

最新的日志专用压缩方法（如 LogReducer、CLP 和 LogGrep）都是通过将日志解析拆分成细粒度存储单元进行编码压缩。现有方法未充分考虑日志数据的局部特征并根据局部特征选择合适的编

码方法, 导致在压缩某些日志数据时, 压缩效率不理想且延迟较高。

本文提出了一种数据特征感知的细粒度存储单元快速编码方法, 以解决现有压缩方法中压缩延迟过高的问题。通过观察发现, 细粒度存储单元中的数据具有两种分布特征, 即局部重复和局部递增。根据这两种分布特征, 提出了利用数据的局部特征进行编码压缩方法。首先识别出数据中的两种局部特征区间, 对于局部重复特征区间, 使用RLE码进行编码处理; 对于局部递增特征区间, 使用Delta编码进行编码处理。采用轻量级的Delta编码和RLE编码对日志数据进行局部编码能够更好地适应日志数据的局部特征, 从而提高压缩效率, 降低压缩延迟。

本文所提出基于局部特征的编码压缩方法在Entry类细粒度存储单元上表现良好。对于Svar类单元, 虽然其平均压缩速度仍低于全局编码方法, 但与直接使用gzip压缩算法相比, 压缩延迟可降低78%。因为Svar类数据具备数值大、离散性强、分布无规律且出现频次低等特点, 局部特征区间占比较少, 导致编码效果较差。为应对离散性强且数值较大的整型数据带来的编码挑战, 未来工作将引入自适应编码策略, 借鉴Elastic Encoding的设计思想, 根据数据特征动态选择最优编码模式, 结合运行时数据分布感知机制, 在压缩率与压缩延迟之间实现更优平衡。下一步工作将融合更多先进的编码方法, 提高低局部重复性数据的表示效率, 从而构建更具普适性的细粒度编码框架, 持续提升整体压缩率并减少处理开销。

参考文献:

- [1] Ren H S, Xu B X, Wang Y J, et al. Time-series anomaly detection service at Microsoft[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York: ACM, 2019: 3009-3017.
- [2] Geiger A, Liu D Y, Alnegheimish S, et al. TadGAN: time series anomaly detection using generative adversarial networks[C]//Proceedings of the 2020 IEEE International Conference on Big Data (Big Data). Piscataway: IEEE Press, 2020: 33-43.
- [3] Fu Q, Lou J G, Wang Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis[C]//Proceedings of the 2009 IEEE International Conference on Data Mining. Piscataway: IEEE Press, 2009: 149-158.
- [4] He S L, Zhu J M, He P J, et al. Experience report: system log analysis for anomaly detection[C]//Proceedings of the 2016 IEEE International Symposium on Software Reliability Engineering (ISSRE). Piscataway: IEEE Press, 2016: 207-218.
- [5] 张圣林, 李东闻, 孙永谦, 等. 面向云数据中心多语法日志通用异常检测机制[J]. 计算机研究与发展, 2020, 57(4): 778-790.
Zhang S L, Li D W, Sun Y Q, et al. Unified anomaly detection for syntactically diverse logs in cloud datacenter[J]. Journal of Computer Research and Development, 2020, 57(4): 778-790.
- [6] Wang S B, Zhang G Y, Wei J Y, et al. Understanding silent data corruptions in a large production CPU population[C]//Proceedings of the 29th Symposium on Operating Systems Principles. New York: ACM, 2023: 216-230.
- [7] Du M, Li F F, Zheng G N, et al. Deep-Log: anomaly detection and diagnosis from system logs through deep learning [C]//Proceedings of the 2017 ACM

- SIGSAC Conference on Computer and Communications Security. New York: ACM, 2017: 1285–1298.
- [8] Nagaraj K, Killian C, Neville J. Structured comparative analysis of systems logs to diagnose performance problems [C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. Berkeley: USENIX, 2012: 1–14.
- [9] Yuan D, Mai H H, Xiong W W, et al. SherLog: error diagnosis by connecting clues from run-time logs[C]//Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2010: 143–154.
- [10] Xu H W, Chen W X, Zhao N W, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications[C]//Proceedings of the 2018 World Wide Web Conference. New York: ACM, 2018: 187–196.
- [11] Dumais S, Jeffries R, Russell D M, et al. Understanding user behavior through log data and analysis[M]//Olson J S, Kellogg W A. Ways of Knowing in HCI. New York: Springer, 2014: : 349–372.
- [12] Fan Y C, Chen Y C, Tung K C, et al. A framework for enabling user preference profiling through Wi-Fi logs[J]. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(3): 592–603.
- [13] 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 conference on Operating Systems Design and Implementation. Berkeley: USENIX, 2014: 217–231.
- [14] Ghassemian M, Hofmann P, Prehofer C, et al. Performance analysis of Internet gateway discovery protocols in ad hoc networks[C]//Proceedings of the 2004 IEEE Wireless Communications and Networking Conference. Piscataway: IEEE Press, 2004: 120–125.
- [15] Fazzinga B, Flesca S, Furfaro F, et al. Online and offline classification of traces of event logs on the basis of security risks[J]. Journal of Intelligent Information Systems, 2018, 50(1): 195–230.
- [16] Oprea A, Li Z, Yen T F, et al. Detection of early-stage enterprise infection by mining large-scale log data[C]//Proceedings of the 2015 Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway: IEEE Press, 2015: 45–56.
- [17] Deligiannakis A, Kotidis Y, Roussopoulos N. Dissemination of compressed historical information in sensor networks[J]. The VLDB Journal, 2007, 16(4): 439–461.
- [18] Palpanas T. Big sequence management: a glimpse of the past, the present, and the future[M]//SOFSEM 2016: Theory and Practice of Computer Science. Berlin: Springer, 2016: 63–80.
- [19] Sharma A B, Ivančić F, Niculescu-Mizil A, et al. Modeling and analytics for cyber-physical systems in the age of big data[J]. ACM SIGMETRICS Performance Evaluation Review, 2014, 41(4): 74–77.
- [20] Cleary J, Witten I. Data compression using adaptive coding and partial string matching[J]. IEEE Transactions on Communications, 1984, 32(4): 396–402.
- [21] 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.
- [22] Liu J Y, Zhu J M, He S L, et al. Logzip: extracting hidden structures via iterative clustering for log compression[C]//

- Proceedings of the 2019 IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2019: 863–873.
- [23] 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.
- [24] 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. New York: ACM, 2018: 167–177.
- [25] Sayood K. Introduction to Data Compression, 5th edition[M]. San Francisco: Morgan Kaufmann, 2017.
- [26] 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.
- [27] 蔡涛, 雷天乐, 牛德姣, 等. 面向NVM的IoT时序数据多态协作压缩策略[J]. 大数据, 2024, 10(4): 34–50.
- Cai T, Lei T L, Niu D J, et al. A polymorphic cooperative compression strategy for IoT time series data based on NVM[J]. Big Data Research, 2024, 10(4): 34–50.
- [28] Mansouri Y, Toosi A N, Buyya R. Data storage management in cloud environments: taxonomy, survey, and future directions[J]. ACM Computing Surveys, 2018, 50(6): 1–51.
- [29] Wei J Y, Zhang G Y, Wang Y, et al. On the feasibility of parser-based log compression in large-scale cloud systems [C]//Proceedings of the 19th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2021: 249–262.
- [30] Wei J Y, Zhang G Y, Chen J C, et al. LogGrep: fast and cheap cloud log storage by exploiting both static and runtime patterns[C]//Proceedings of the 18th European Conference on Computer Systems. New York: ACM, 2023: 452–468.
- [31] Rodrigues K, Luo Y, Yuan D. CLP: efficient and scalable search on compressed text logs[C]//Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2021: 183–198.
- [32] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 2003, 23(3): 337–343.
- [33] Pavlov I. Lempel–Ziv–Markov chain algorithm[C]//Proceedings of the Data Compression Conference. Snowbird: IEEE, 2004: 172–181.
- [34] Internet Engineering Task Force. RFC 8878. Zstandard Compression and the application/zstd Media Type[S]. 2021.
- [35] Langdon G G. An introduction to arithmetic coding[J]. IBM Journal of Research and Development, 1984, 28(2): 135–149.
- [36] Abadi D, Madden S, Ferreira M. Integrating compression and execution in column-oriented database systems[C]//Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2006: 671–682.
- [37] Schindler M. A fast block-sorting algorithm for lossless data compression[C]//Proceedings of the Data Compression Conference. Piscataway: IEEE, 1997: 469.
- [38] Williams R N. An extremely fast Ziv–Lempel data compression algorithm[C]//Proceedings of the IEEE Data Compression Conference. Piscataway: IEEE,

1991: 123–132.

[39] Salomon D. Variable-length codes for data compression[M]. London: Springer, 2007.

[40] Gao Y H, Huang S L, Parameswaran A. Navigating the data lake with DATAMARAN: automatically extracting structure from log datasets[C]//Proceedings of the 2018 International Conference on Management of Data. New York: ACM, 2018: 943–958.

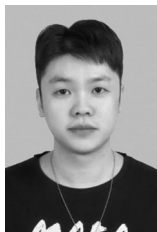
[41] Hamooni H, Debnath B, Xu J W, et al. LogMine: fast pattern recognition for log analytics[C]//Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. New York: ACM, 2016: 1573–1582.

[42] Nagappan M, Vouk M A. Abstracting log lines to log event types for mining software system logs[C]//Proceedings of the 2010 IEEE Working Conference on Mining Software Repositories (MSR 2010). Piscataway: IEEE Press, 2010: 114–117.

[43] 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, 2011: 785–794.

[44] 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



汪毛毛（1997–），男，公安部第三研究所研究实习员，主要研究方向为大数据流处理、数据存储与数据压缩、云计算等。



王蕾（1989–），女，公安部第三研究所研究实习员，主要研究方向为大数据、云计算等。



周晓芳（1981–），女，博士，公安部第三研究所副研究员，主要研究方向为人工智能、大数据等。

收稿日期: 2025–11–04
通信作者: 周晓芳, fw339@126.com