计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330178
60(11): 2442−2452, 2023

数据模式感知的低成本云日志存储系统

魏钧宇　张广艳　陈军超
（清华大学计算机科学与技术系　北京　100084）
（wei-jy19@mails.tsinghua.edu.cn）

Data-Pattern-Aware Low-Cost Cloud Log Storage Systems

Wei Junyu, Zhang Guangyan, and Chen Junchao
（*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084）

Abstract　Cloud-native system log service can fully boost the researching, maintaining, operating and security ability of the public cloud. Could log data are typically large in scale, requiring long preserving time, high ingestion speed and low access latency, while the information density is low. To save the storage cost, it is required to compactly compress the logs, compress the logs in a high speed, and retrieve target data with low latencies. However, it is challenging to achieve these three goals at the same time, and a customized solution should be designed for this scenario. By summarizing the typical data patterns in the cloud logs, including the static patterns, namely the formatted output statements in the source code and the runtime patterns, which are generated during the execution of the programs, a low-cost storage schema is proposed for the public cloud logs. By evaluating several low-cost storage methods of the cloud logs, their effectiveness with respect to the compression ratio, compression speed and query latency are shown. Besides, several experiences for designing a low-cost storage system for the cloud logs are proposed in expectation of inspiring relevant research in the future.

Key words　log data storage; static pattern; runtime pattern; high compression ratio; low query latency

摘　要　公共云的日志服务能够全面提升研发、运维、运营和安全保障的能力，而云日志具有数据规模庞大、留存时间长、写入速度快、有用信息密度低、访问延迟要求高等特点. 为了节省存储成本，需要满足3个要求：1）以较高压缩密度保存此类数据（压得狠）；2）以较高的压缩速度实现数据写入（压得快）；3）以低延迟对压缩数据进行快速检索（查得快）. 同时实现这三者是充满挑战的，需结合具体应用场景进行定制化设计. 通过总结云日志中的典型数据模式，给出一种低成本云日志存储范式——数据模式感知的低成本云日志存储系统，从压缩率、压缩速度和检索延迟等3个方面对若干低成本云日志存储方法进行对比测试. 最后，结合相关领域研究提出3点经验和思考，供未来的研究工作参考.

关键词　日志存储；静态模式；动态模式；高压缩率；低检索延迟

中图法分类号　TP333

随着信息技术的发展，计算系统日趋复杂，大型系统的维护和检查越来越离不开系统产生的监控日志. 这些日志数据广泛存在于各类系统平台上，并为故障诊断[1-4]、异常检测[5-9]、用户画像[10-11]、系统建模[12-13]和安全检查[14-15]等多种服务提供支撑. 同时为了确保归档审计信息的完整与可溯源性，最新出台的《网络安全法》也明确规定“采取监测、记录网络运行状态、网络安全事件的技术措施，并按照规定留

收稿日期：2023-03-28；**修回日期**：2023-09-03
基金项目：国家自然科学基金项目（62025203）
This work was supported by the National Natural Science Foundation of China (62025203).
通信作者：张广艳（gyzh@tsinghua.edu.cn）

存相关的网络日志不少于六个月”[16]，这又对日志数据提出了长期归档存储的要求.

在现代数据中心中，系统日志有成千上万个数据产生源，数据规模单日可达上百亿条（PB 量级）[17]，该类数据已经俨然成为一类新兴大数据. 为降低数据存储成本，有必要对于该类数据进行快速、高密压缩. 另一方面，在使用这类日志时，过长的检索延迟将会降低查错和系统运维的效率，进而造成不必要的损失[18]，因此在高密压缩的同时还需保证能够快速检索以获得需要的信息. 日志存储的总成本包括数据存储的空间成本、数据压缩的时间成本和数据检索的时间成本，而低成本日志存储系统的设计目标就是要尽可能降低这 3 部分成本.

本文结合公共云中日志数据的真实存储需求，给出了一种云日志数据的低成本存储范式——数据模式感知的低成本云日志存储. 首先，总结了云日志中的典型数据模式，即静态数据模式和动态数据模式；然后，给出了基于数据模式感知构建低成本云日志存储系统的方法；最后，针对国内外若干低成本云日志存储方法在压缩率、压缩速度和检索延迟等方面进行了对比测试.

此外，结合我们在低成本云日志存储系统领域的研究提出了几点经验与思考，希望为未来大规模数据存储的相关研究提供参考和帮助.

我们研制的数据模式感知的低成本云日志存储系统已经应用到某国际著名云厂商的实际生产系统中，降低了 2/3 的云日志存储成本，提升了云服务的服务质量.

1 云日志的数据特点和存储需求

系统日志原本是程序员在编写程序时出于调试需要而设计的一种输出信息. 对于小规模软件来说，通常在调试版本（debug version）中存在小部分的输出日志，在正式的发行版本（release version）中基本不存在额外输出的调试信息. 但随着软件规模的扩大，特别是随着云计算技术的发展，很多在云场景下运行的大型软件也会以一定格式持续不断地输出日志信息[19-20]. 这些信息实时记录了当前系统的运行情况以及处理工作的细节，未来可供调试、归档、审计、系统建模等多种应用使用.

在大型商业云中，通常有隶属于基础事务部的专业日志存储团队，他们将为各类企业运维团队提供包括日志采集、查询、分析、归档存储等云原生 SLS（system log service）服务. SLS 系统将持续收集来自整个系统中不同应用的日志信息，将这些信息按照时间顺序写入到同一缓冲区内，并对于缓冲区内的日志进行分类、压缩等进一步处理. 在保证将这些日志归档存储一定时间的同时，对外提供对于海量云日志的检索和分析服务.

本节将结合云日志分类介绍这类新兴大数据的具体信息，进一步归纳云日志的数据特点以及相应的存储需求.

1.1 云日志的分类

对云日志数据可以从多个维度进行分类. 从日志的级别出发，可以将云日志分成提示（INFO）、警告（WARNING）和错误（ERROR）等不同严重等级，对于按照时间顺序写入到同一片缓冲区中的日志，在进行错误溯源时，通常可以首先根据严重等级进行日志过滤.

从日志的产生机理出发，可以将云日志分成系统日志（system log）、访问日志（access log）、环境指标（environment metric）三类. 系统日志通常由系统软件产生，主要用于提示、告警和报错. 例如，分布式文件系统 HDFS 会记录缓冲区溢出、网络异常、块读写错误等信息. 访问日志主要记录用户对系统的访问，这部分日志主要在用户做出某些行为时生成，用于未来出现安全问题时对可疑操作进行溯源或对一段时间内的用户行为进行建模. 例如，在云系统的网络代理服务器中通常会记录外部服务器对于某些特定 IP 地址的访问. 环境指标则由各个服务器定期产生，通常会随集群心跳信息一起发送到中心协调服务器中. 这些信息通常包括了各服务器的当前环境状态，例如温度、CPU 利用率、内存占用率等指标.

从对于日志的访问频率出发，可以将日志分成在线日志（online log）、近线日志（nearline log）和离线日志（offline log）三类. 具体来说，在线日志是指在产生之后需要进行频繁访问的日志，例如用于系统建模的日志通常会被输入到特定的建模软件中，由建模软件对其进行频繁且种类繁多的检索访问. 和在线日志相对的是离线日志，这类日志在产生之后基本不会进行访问，例如用户访问日志、环境监控日志等，这些日志产生的初衷即是在出现严重的错误，或者运营商卷入到相关的案件时才会被调用协助案情侦查. 近线日志的访问频率介于在线日志和离线日志之间，这类日志主要以系统的各种警告、报错日志为主，广泛应用于云系统的异常诊断和故障溯源中. 很多错误通常不会立刻影响前端应用，却会被系统以

警告或者报错的形式体现在日志之中,因此在进行系统故障诊断时通常需要访问这类日志.

根据某国际著名云厂商单月内对于PB量级云日志数据检索次数的统计,44%的日志在产生之后单月内的访问次数为零,属于离线日志;2%的日志单月内的访问次数超过100,可以归类为在线日志;剩下的54%的日志则介于二者之间,对它们存在一定数量的访问,但是访问频率较低,属于近线日志.日志数据写入时通常会根据未来一段时间内是否进行密集访问分成在线日志和近线日志:对于在线日志,通常保留原始的日志数据,并通过在数据上建立索引的方法加速数据访问性能;对于近线日志和离线日志则进行压缩以降低存储开销,未来查错时可能需要对这些数据进行低延迟检索.经过一段时间之后,访问次数为零的日志将被视为离线日志,并进行高密压缩以满足归档需求.

1.2 云日志的数据特点

通过对某国际著名云厂商的实际生产日志的观察,我们归纳得到云日志的若干数据特点.

特点1: 数据规模巨大、留存时间长.由于云日志通常用于大型软件或者整个系统级别的查错,因此需要由统一的日志管理部门来协调整个平台中产生的各类日志,而所有这些日志的规模单日可以达到PB量级.另一方面,《网络安全法》规定这些日志需要留存至少180天,因此如果直接存储这些日志数据将带来巨大的存储开销.

特点2: 有用信息密度低.尽管云日志的规模庞大,但其中的有用信息密度较低.这一方面是由于日志数据存在较多冗余,例如在执行类似操作时输出的提示信息完全相同、执行多次访问时留下的记录也大同小异等.另一方面是由于在审计和故障溯源的过程中,真正需要的日志条目在所有日志条目中所占的比例极低.例如,在一次云系统的故障溯源过程中,日志库中可能相关的日志条目总量超过了70亿条,而其中真正对错误定位有帮助的日志条目仅有391条.

特点3: 访问延迟敏感.尽管大部分日志数据的访问频率较低,但单次访问通常是一个查错溯源过程中的必要环节.通过和日志维护团队的工程师进行交流,工程师通常能接受的单次检索延迟应当在秒量级,最多不超过分钟量级,一旦单次检索超过半小时,将严重影响工程师的查错效率[18].此外,如果日志检索显著拖慢了整个溯源过程,就有可能因此耽误对于致命问题的排查,进而造成更为严重的影响.

1.3 云日志的存储需求

由于日志中有用信息密度低、规模庞大且留存时间较长,对于近线日志和离线日志,通常需要进行压缩存储以降低存储成本.另一方面,对于近线日志和在线日志,由于在检索时对于延迟较为敏感,还需要在数据调用时保证稳定的低延迟.

上述2个存储需求造成了日志存储面临高压缩率和低解压开销难以兼得的问题.现有的压缩工具(包括常见的通用压缩工具[21-24]和一系列日志专用压缩工具[25-29])通常是在两者之间折中,例如LZMA和PPMd选择牺牲解压延迟以实现高压缩率,gzip选择牺牲高压缩率以实现低解压延迟,zstd则给用户提供了不同的压缩等级,允许用户自行在高压缩率和低解压开销之间折中.日志专用的压缩方法例如LogArchieve将日志分桶大小作为一个可调的超参数,分桶越大日志压缩率越高,但解压开销越大;反之分桶越小解压开销越小,但日志压缩率也会相应下降.

那么,是否有同时实现高压缩率和低检索延迟的低成本存储方法呢?事实上,云日志中所隐含的数据模式为我们提供了一种全新的数据存储思路.

2 云日志中的数据模式

模式,通常是指一个变化的数据集中相对固定的部分.云日志作为一种半结构化数据,其中存在大量的冗余信息,这些冗余信息和有用信息混合在一起便构成了云日志中的若干数据模式.

首先,由于日志数据的产生源头是用户定义的结构化输出语句,因此由同一条结构化输出语句输出的日志通常会共享结构化语句体.如图1所示,由结构化输出语句“Time taken %ds”输出的日志都共享“Time taken <*>s”的冗余字段.

其次,在程序执行过程中,同一条结构化输出语句通常会连续输出多条日志,这些日志除了具有完全相同的结构化语句体之外,它们的变量值之间也因为程序执行的局部性而共享一些相对固定的部分.例如,在“Time taken <*>s”的例子中,连续的时间戳之间的数值差异不大;在HDFS中块的编号通常以“blk_”开头;在记录IP地址的日志中,属于同一个子网的IP地址之间通常共享一些相同的字段等等.

本节将首先介绍云日志的两类代表性数据模式,接下来讨论提取云日志数据模式的一些有效方法,为进一步讨论数据模式感知的云日志存储做好准备.

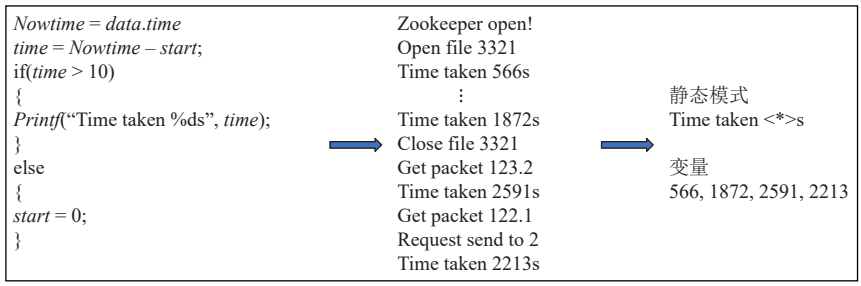

Fig. 1 The generation process of the static pattern in log data

图 1 日志数据静态模式的产生过程

2.1 云日志中的代表性数据模式

已经发现云日志有两类代表性数据模式：静态模式和动态模式.

1)静态模式. 我们将日志数据中因共享程序员定义的结构化输出语句而形成的模式称为静态模式. 这类模式通常可以通过静态代码分析或根据同一公司内的编程规范加以确定.

我们可以将由同一个结构化输出语句所输出的日志条目视为一个日志条目组，将同一日志条目组内出现在相同位置的变量视为一个变量组. 如果结构化输出语句中具有 x 个变量，就会对应 x 个变量组. 例如在图 1 所示的例子中“Time taken %ds”对应的日志条目组中包含 4 条日志，变量“%d”对应的变量组中包含 4 个变量数值.

2)动态模式. 除静态模式之外，我们将在实际运行中同一个变量组的变量数值由于局部性而共享的相对固定的部分称为动态模式. 对于这类模式，程序员不可能在程序执行前就预先获知变量的内容，同时这类模式还会随着程序运行的局部性变化而不断改变，因此这类模式无法仅仅通过静态分析程序语句的方法获得，还必须通过对运行过程中动态产生的日志执行进一步的模式识别才能得到.

动态模式能够有效地将变量分割成更细粒度的存储单元. 如图 2 所示，左侧的 4 条日志属于 2 个日志条目组，每一个日志条目组中包含 2 个变量组；在右侧，我们通过对这 4 个变量组的进一步分析得到每一个变量组的动态模式，并可将原有变量组分解成更细粒度的存储单元.

2.2 日志数据模式的提取

针对云日志的两类代表性数据模式，人们已经找到了行之有效的提取方法.

1)静态模式提取方法. 静态模式提取是指从原始云日志数据中获得对应的结构化输出语句的过程，它是日志分析领域的一个研究热点. 已有提取方法主要包括基于频繁序列的方法[7, 30-33]、基于聚类的方法[34-35]以及其他基于启发式的方法，例如基于树结构[36]、基于日志长度[37]等. 相关研究表明[38]，基于解析树的方法效果最佳，故在这里着重介绍.

基于解析树的提取方法首先使用一系列预先规定的字符(例如逗号、空格等)将日志条目分割成一系列 token 组成的串，然后根据如图 3 所示的解析树实现静态模式的提取. 该解析树共有 2 层，在第 1 层中使用日志条目中包含的 token 数量对日志条目进行归类，具有相同 token 数目的日志条目将进一步聚类为同一组. 同一组中相似度超过阈值的日志条目将被认为具有同样的静态模式，这些条目中出现在同一位置的变量将被存入到特定的变量组中.

在实际执行时，日志解析器将首先采样一部分日志条目作为训练集，训练生成如图 3 所示的解析树. 在实际压缩阶段，尝试将日志条目匹配到某一训

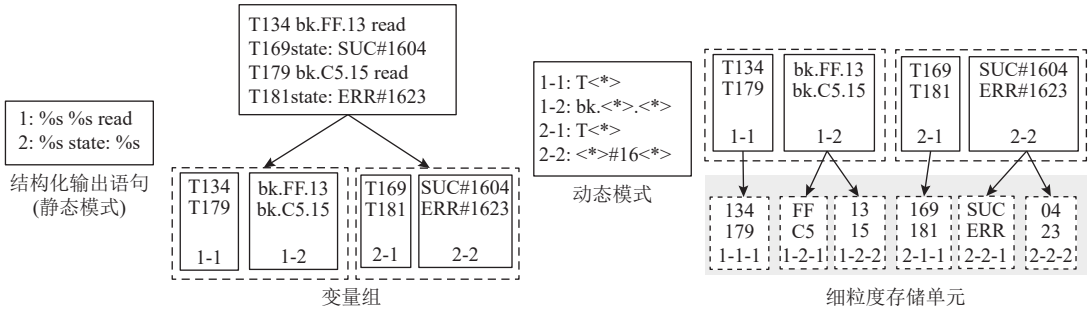

Fig. 2 Examples for static pattern and runtime pattern in log data

图 2 日志数据中的静态模式和动态模式举例

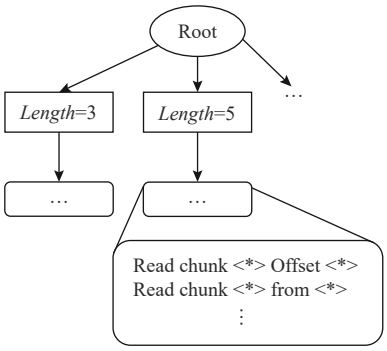

Fig. 3 Parser tree for static pattern

图 3 静态模式解析树

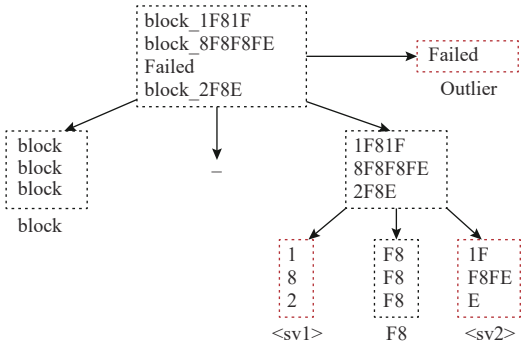

Fig. 4 Tree-expanding-based top-down runtime pattern extraction strategy

图 4 基于树扩展的自顶向下动态模式提取策略

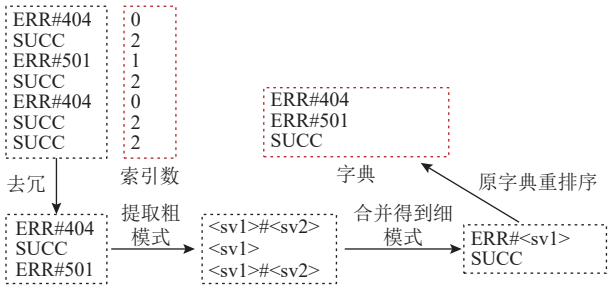

Fig. 5 Pattern-merging-based bottom-up runtime pattern extraction strategy

图 5 基于模式合并的自底向上动态模式提取策略

练阶段时生成的静态模式之中，否则认为该条目出现了失配，所有失配的日志条目将单独进行压缩.

基于解析树的静态模式匹配策略可以实现对静态模式的黑盒提取，即在无法获得日志数据生成代码的条件下即可完成对于日志数据静态模式的有效提取. 但黑盒提取方法的效果高度依赖于提取过程中所规定的 token 分隔符和所设定的相似度阈值，在实际使用时，也可以通过在模式提取阶段引入一些静态模式的白盒获取方法[39-42]，以提升日志压缩的效果.

2)动态模式提取方法. 动态模式提取是指结合系统运行时输出日志的特点，提取日志变量中共性部分的过程，主要方法包括 2 种：基于树扩展的自顶向下策略和基于模式合并的自底向上策略.

基于树扩展的自顶向下策略通常用于变量组中仅包含一种动态模式的情况. 如图 4 所示，该策略首先从变量组中随机挑选一个变量，利用其中包含的非字母、非数字的特殊字符(图 4 中为“_”)将原始变量组划分成更小的变量组，在这一过程中如果某一个变量不包含当前进行划分的字符，则被存入到一个独立存储单元中进行压缩. 针对划分完成的更小变量组，该策略将进一步识别变量值之间的公共子串(图 4 中为“F8”)并基于此对变量组做进一步的划分，如果某一个子变量组中的所有子变量都相同，则将其记为常量并归入到动态模式之中(图 4 中为“block”). 接下来对于所有的子变量组重复上述流程，直到所有子变量组都无法继续扩展为止.

基于模式合并的自底向上策略通常用于处理重复变量值较多的变量组. 如图 5 所示，该策略将首先进行去冗处理，将所有不重复的数值存储为字典. 此时由于原始变量组内的重复变量值较多，字典通常要比原始变量组的规模小很多，因此可以在字典上执行复杂度较高的基于模式合并的动态模式提取策略. 该策略首先根据变量中的非字母、非数字的特殊字符提取得到粗模式，之后再将具有相同粗模式的变量合并得到细模式，此时如果粗模式的某一子变量在各个变量值中都保持一致(例如“ERR”)，则将其记为常量. 最终，根据提取得到的细模式将原始字典进行重排序，将具有相同细模式的变量值进行连续存储，以方便后续检索.

在实际使用时可以根据变量组内重复变量值的多少将变量组分成两类：真实变量组(real variable vector)和名义变量组(nominal variable vector). 前者所包含的变量值中重复的相对较少，例如时间戳、具体的延迟数值等；后者所包含的变量值中则重复的相对较多，例如用户名称、告警级别等.

真实变量组通常只包含一种动态模式，而名义变量组则包含多种动态模式. 因此，针对真实变量组可以采用基于树扩展的自顶向下策略，针对名义变量组可以采用基于模式合并的自底向上策略.

3 数据模式感知的日志存储

日志数据模式已在多种日志应用中得到应用. 例如，借助解析得到的日志静态模式来确定不同结

构化输出语句的输出顺序，进而分析程序行为[40]；通过日志动态模式中变量的数值特征检测异常事件[1]. 本节重点讨论如何利用日志中的数据模式实现日志数据存储，以同时实现高压缩率和低检索延迟，进而构建低成本的云日志存储系统.

针对离线云日志“写一次，很少读”的特点，可以利用日志模式对数据进行更高密度的压缩. 我们发现通过静态模式提取将日志存储为若干变量组，一方面，可以从全局上去除日志中因为共享相同的结构化输出语句而具有的冗余，而又不依赖于通用压缩方法所使用的去冗手段的窗口大小；另一方面，每一个变量组内的变量数值具有一定的共性特征（动态模式），可以对不同的变量组设计定制化的压缩编码方法以提升数据压缩率.

针对近线云日志“写一次，多次读”的特点，我们借助数据模式，找到了可以同时实现高压缩率和低检索延迟的日志数据压缩粒度，即基于静态模式和动态模式划分出细粒度存储单元，对每一个存储单元可以进行独立压缩和独立解压. 一方面，在对日志数据进行检索时，同一个存储单元内的数据倾向于被同时访问，因此将它们压缩到一起就避免了访问过程中的无效解压；另一方面，可以在压缩时将各个存储单元内的数据特点描述为标签，利用标签在解压数据之前就可以对存储单元进行有效过滤，从而降低检索操作需要解压的数据量，进而实现低延迟检索.

本节将讨论2个我们设计的数据模式感知的日志存储方法，进而说明这2种思路的实现过程. 我们研制的数据模式感知的低成本云日志存储系统已经应用到某国际著名云厂商的实际生产系统中，降低了2/3的云日志存储成本，提升了云服务的服务质量.

3.1 离线云日志的低成本存储

我们提出了离线云日志的低成本存储方法LogReducer，它基于静态模式对日志数据进行结构化存储，再为每一个变量组定制化设计编码方法，最终实现了较高的压缩率. 如图6所示，LogReducer的具体压缩流程可以分成训练阶段和压缩阶段. 在训练阶段，LogReducer使用静态模式提取方法在日志样本上提取得到静态模式. 在压缩阶段，通过静态模式匹配，将原始日志结构化成若干变量组，再针对每一个变量组有针对性地进行编码，最后将所有编码后的变量组压缩得到最终的压缩文件.

依据变量的数值特点，变量组编码方法主要包括：

1）时间戳组成的变量组. 时间戳变量组在压缩

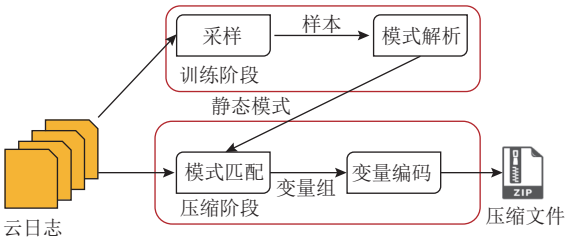

Fig. 6 The architecture of LogReducer

图6 LogReducer架构

之后得到的文件中占比普遍较高（极端情况下占比可达70%左右），同时由于在大型数据中心中日志大数据的产生速度很快，相邻的日志条目之间的时间差值普遍较小. 因此，通过引入差分编码，可显著提升压缩率和压缩速度.

2）存在线性关联关系的变量组. 通过对线性关联关系的挖掘，可以缩减变量组内数值的大小以提升压缩效果. 例如，一条日志条目同时包含一次写操作的块编号、偏移数值和写入长度. 由于很多写入是连续的，因此针对同一个写入块而言，上一次写入的偏移数值加上写入大小就等于下一次的写入偏移. 通过利用该方法，可以仅记录这一关联关系，而将很多原始变量组中的数值削减为零.

3）数值偏小的整数变量组（包括通过差分处理和关联关系挖掘之后的变量组）. 可以采用弹性编码策略将数值比较小的整数（很多高位的字节都是零）采用更少的字节来存储. 此外，由于差分处理和关联关系挖掘引入了很多绝对值较小的负数，该方法也通过移位加比特翻转的途径，使用更少字节表示小绝对值的负数.

3.2 近线云日志的低成本存储

我们提出了近线云日志的低成本存储方法LogGrep，它借助静态模式和动态模式将日志分割成细粒度的存储单元，在保证高压缩率的同时，通过细粒度单元的高效过滤实现了日志的低延迟检索. 如图7所示，LogGrep在压缩时首先通过静态模式解析将原始云日志存储为变量组，接下来又通过动态模式解析将每一个变量组存储成若干细粒度的存储单元，并根据每一个存储单元内的数据特点生成对应的标签以支持高效检索，最后将每一个存储单元独立压缩编码. LogGrep在检索时通过模式匹配和存储单元过滤得到需要解压的存储单元，解压这些单元并在这些单元上进行检索定位，在定位到被检索的目标日志之后，进行日志重构并将结果返回给用户.

LogGrep使用动态模式对变量组内信息进行抽

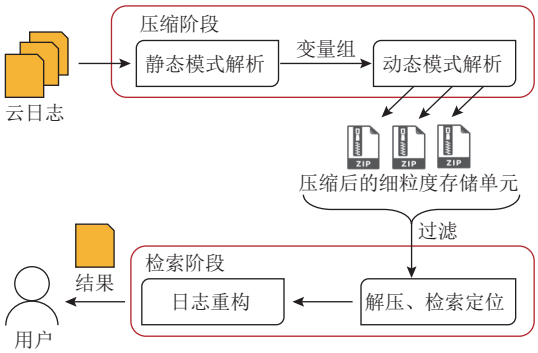

Fig. 7 The architecture of LogGrep

图 7 LogGrep 架构

象和提炼，因此可以在进行真正解压之前就利用动态模式提前获知对应的变量组内是否存在要检索的内容. 同时较之于原始的变量组，细粒度的存储单元内的数据种类较为单一(通常都是整数或都是十六进制数)，这就为更精确地标注创造了条件. 我们发现如果将所有字符分成 6 类(0~9, a~f, g~z, A~F, G~Z, 其他)，一个静态模式结构化后的日志变量组平均包含 3.1 种字符，而一个经过动态模式结构化处理的细粒度存储单元则平均仅包含 1.5 种字符.

LogGrep 通过动态模式的结构化处理，在过滤阶段需要解压的日志量从原始的整个变量组降低到细粒度的存储单元，这相当于缩小了变量组的大小，从而降低了检索延迟. 同时，我们发现通过将原始变量组存储成细粒度存储单元，每一个细粒度存储单元独立压缩后的大小之和与原始变量组整体的压缩大小基本相同. 换言之，引入动态模式在降低检索延迟的同时，能够保持静态模式带来的高压缩率，相当于并没有因为变量组缩小而给压缩率带来负面影响. 这主要是因为通过识别变量组中存在的动态数据模式，变量组内的数据冗余要么被提取到动态模式之中，要么就在细粒度存储单元之内，因此不同存储单元之间基本不存在冗余，进而可以保证独立压缩和整体压缩的压缩率相差不大.

基于静态模式和动态模式的存储方式，在处理检索请求时，可以首先在日志的静态模式以及动态模式中匹配检索串，确定检索串中哪些部分属于静态模式或动态模式，哪些部分可能被包含在细粒度存储单元内，这部分可能包含在细粒度存储单元内的部分被称为待检索关键字.

图 8 展示了一个在动态模式上进行匹配的例子，对应于在图 4 提取得到的动态模式上匹配“8F8F”串，在整个匹配过程中临时结果都表示为一个 Bitmap，Bitmap 的总长为变量组中的变量值个数，由于属于同一个静态模式的变量组长度相同，因此该 Bitmap 也可用于记录该静态模式中对应的日志条目组中的命中情况，如果某一个日志条目在当前的检索条件下被命中，则 Bitmap 对应位置标 1，否则标 0. 对于例子中的检索串“8F8F”而言，可能有 2 种匹配方案：一个是“F8”被包含在模式中，而“8”和“F”均包含在子变量中；另一个是“8”被包含在模式中，而“F8F”被包含在子变量中. 对于第 1 种情况，对应产生 2 个子变量匹配过程 *A* 和 *B*，原始结果需要先在 *A* 过程中匹配当前已经命中的条目，然后再在 *B* 过程中匹配已命中条目. 对于第 2 种情况则只对应一个匹配过程 *C*. 最终结果是上述 2 种情况得到的中间结果的并集.

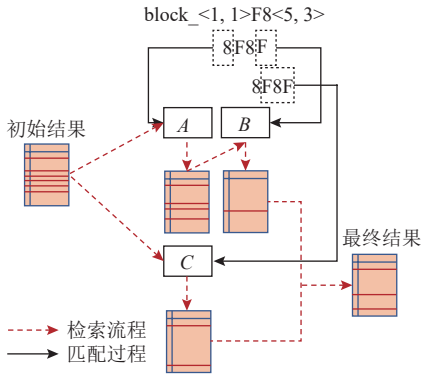

Fig. 8 An example for runtime pattern matching

图 8 一个在动态模式上进行匹配的例子

在使用细粒度存储单元标签和检索关键字进行匹配的过程中，LogGrep 过滤掉不可能包含关键字的细粒度存储单元，解压剩余的细粒度存储单元并同检索关键字进行匹配. 经过上述的一系列过滤，最终仅需要解压并检索少部分的细粒度存储单元就可以完成对日志大数据的检索，解决了日志大数据有效信息密度低以及对于检索延迟要求高的挑战.

在细粒度存储单元内进行检索时，LogGrep 采用定长检索策略，即在压缩时根据一个细粒度存储单元内的最长项将各项进行补齐，由于同一个细粒度存储单元内各项的长度接近，采用该补齐方法对于压缩率的影响较小，但在检索时由于细粒度存储单元内所有项长度相同，显著提升了检索性能.

4 典型低成本云日志存储方法的比较

基于数据模式感知的思想，国内外多个团队进行了探索和尝试，设计了多种低成本云日志存储方法，典型的有多伦多大学设计的 CLP[43] 与我们设计的 LogReducer[17] 和 LogGrep[18]. 其中 CLP 主要利用了

静态模式，LogReducer 和 LogGrep 同时使用了静态模式和动态模式.

CLP 使用静态模式进行日志存储，它主要采用基于白盒的方法来进行静态模式的提取，并使用提取得到的静态模式和相应的变量格式构建倒排表并用作压缩日志段的索引. 在日志检索时，系统将首先检查倒排表索引，确定需要解压的日志块并进行解压. CLP 的优点是索引构建迅速，日志写入速度较高；缺点是不同静态模式的变量连续存放在一起进行压缩，压缩率和检索性能都相对较低.

LogReducer 针对用于审计的离线日志设计，这部分日志平时的访问数量几乎为零，但是一旦被调用，就需要实现无损还原，因此它无法有效支持直接针对压缩日志的检索操作，需要在检索前先将日志还原.

LogGrep 针对近线日志设计，同时满足云日志的高压缩率和低检索延迟的存储需求.

我们选用 15 种某国际著名云厂商的总大小为 1.6 TB 的真实生产日志对上述 3 种典型的日志存储方法进行了测评. 评价指标主要包括压缩率、压缩速度和检索延迟. 其中由于 LogReducer 无法支持对于压缩格式的直接检索，因此我们设置 LogReducer 的检索延迟得分为 0.

图 9 以一种较为直观的方式显示了 3 种日志数据存储方法在 3 个维度上的性能. 在每一类日志上，所有方法在 3 个维度的性能都同性能最好的方法做归一化(压缩率最高的方法得分为 1，检索延迟最低的方法得分为 1)，之后将对某一方法某一维度上 15 种日志的结果计算平均值.

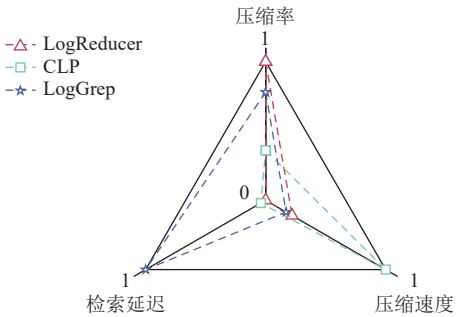

Fig. 9 The evaluation of three typical methods

图 9 三类典型的日志存储方法测评

在压缩率方面，LogReducer 的平均得分为 1，相对于其他两种方法，它在 15 种日志中的 13 种上都实现了最高的压缩率. LogGrep 的平均得分为 0.78，在 LogReducer 未达到最佳的 2 种日志上，LogGrep 的压缩率最高，在最差情况下 LogGrep 的压缩率大概为 LogReducer 的一半，这主要是因为 LogGrep 在使用静态模式提取变量组之后，对于所有的变量组都采用了统一的编码方法，而并非像 LogReducer 那样针对不同类型的变量组采用不同的编码方法. 相比于其它两类方法，CLP 的压缩率具有明显劣势，其压缩率得分仅有 0.36，这主要是因为它仅使用静态模式构建检索时的索引，并未使用该模式实现日志的高密压缩.

在压缩速度方面，CLP 的平均得分为 1，相比于其他两类方法具有明显的优势，在全部 15 种日志上都达到了最佳性能. LogReducer 和 LogGrep 的得分分别为 0.22 和 0.17. 相比于 CLP，LogReducer 和 LogGrep 都涉及到黑盒模式提取步骤，该步骤涉及到相对复杂的训练过程和日志拷贝过程，因此对应的压缩速度相对较低.

在检索延迟方面，LogGrep 表现出了明显的性能优势，它在全部 15 种日志上的检索延迟都要远低于 CLP，在一些日志上甚至要低 2 个数量级，平均来看，CLP 的检索延迟得分仅有 0.04.

如表 1 所示，针对不同日志类型和日志存储需求，可以选用不同的日志存储方法. 当我们仅需要高压缩率时可以优先选用 LogReducer，当我们仅需满足高压缩速度时，可以优先选用 CLP. LogGrep 可以同时实现需要高压缩率(压得狠)和低检索延迟(查得快)的双重目标，但它依旧未能攻克如何实现高速模式提取和数据编码的难题，因此其压缩速度较差.

Table 1 Storage Requirements and Suitable Storage Methods for Different Logs

表 1 不同日志类型的存储需求与适用存储方法

日志类型	存储需求	适用存储方法
离线日志	高压缩率、高压缩速度	LogReducer
近线日志	高压缩率、高压缩速度、低检索延迟	CLP, LogGrep
在线日志	低检索延迟	ElasticSearch

5 云日志存储系统设计的经验与思考

公共云日志数据以其产生迅速、规模庞大、留存时间长、有用信息密度低、访问延迟敏感等特点对存储系统提出了新的挑战，本节将介绍在设计相关低成本存储系统过程中的经验与思考，希望能对未来相关领域大数据低成本存储系统的设计起到借鉴作用.

1)低成本云日志存储系统设计的核心思路在于以全生命周期的视角考虑数据压缩和成本缩减,即压缩存储系统的设计不仅要考虑高压缩率和存储成本,而且还需考虑数据写入时的压缩成本和数据在被访问时的解压量以及对应的检索成本.这就要求相关设计者不能顾此失彼,只有通盘考虑了数据的全生命周期才能真正实现数据存储的低成本.现有的很多压缩工具(例如 LZMA, zstd)仅仅追求高压缩率和高压缩速度,默认数据在访问时必须通过全解压的方式进行访问,在设计压缩算法时并未考虑检索延迟的因素,这就造成相关方法尽管可以实现较低的存储开销但却无法保证低检索开销,极端情况下甚至会造成系统工程师误判错误根因,进而影响整个错误溯源的结果.

2)数据模式感知的低成本云日志存储系统的相关实践表明,领域专用的低成本解决方案不仅在压缩率方面优于通用的解决方案,而且带来了全生命周期设计低成本数据存储系统的可能.LZMA 和 zstd 等压缩工具在压缩时不考虑数据实际访问模式的根本原因在于它们属于通用压缩方法,在压缩过程中无法结合实际数据特征调整相应的压缩策略,进而不仅错失通过针对性编码提升数据压缩率的机会,而且无法达到"压缩与检索联合设计"所能够同时实现的高压缩率、低检索延迟的双重目标.

3)数据模式感知的低成本云日志存储系统最大的成功经验在于通过引入数据模式实现了"压得狠"和"查得快"这一对看似矛盾的目标的和解."压得狠"和"查得快"的矛盾焦点在于数据的压缩粒度,即将多少数据、哪些数据压缩到一个存储单元的问题:如果压缩粒度过大,更多的数据被压缩到一起,去冗机会增多,但是每一次访问时解压的数据量也随之增加;反之如果压缩粒度过小,每次访问时可以按需要进行解压,但却降低了数据去冗的机会.通过挖掘数据模式,我们找到了数据压缩的最佳粒度,即将数据模式中出现在同一个位置的变量值压缩到一起,例如静态模式中的变量组、动态模式中的子变量组等.一方面,该方法可以最大限度保证不同组之间的冗余相对较少,进而保证针对每一个组单独进行编码和将所有组放在一起进行编码的压缩率差别不大,但前者的编码粒度显然更细.另一方面,该方法还能保证属于同一组内的数据具有一定的共性特征,同时在访问时也倾向于被一起访问,这就为有针对性的数据编码和减少解压的细粒度的数据过滤提供了可能.

未来的低成本存储系统设计应当结合数据的全生命周期和领域特征,有针对性地开展数据编码与压缩.模式感知的思路不仅可以应用于半结构化的日志数据,在海量非结构化的数据中,也可以探索更为先进的模式识别方法找到对应的模式和变量组,并通过变量组编码来实现"压得狠"和"查得快"的双重目标.

6 总结与展望

本文总结了一类新兴的大数据——云日志大数据的数据特点和存储挑战,介绍了数据模式感知的低成本云日志存储系统的核心设计思想.详细介绍了目前国内外已有的低成本云日志存储系统所使用的日志数据模式及其具体使用方法,并对若干典型方法进行了测评与分析.最后介绍了我们在设计数据模式感知的低成本云日志存储系统时的经验与思考.

目前还没有一款云日志存储系统能够在压缩率、压缩速度和检索延迟这 3 个维度上同时达到最优,该目标的主要挑战在于如何在低提取成本但效果有限的全局模式提取方法与高提取成本但效果更好的局部模式提取方法之间做权衡.希望相关领域的研究可以不断进步、推陈出新.另一方面,日志压缩也可以考虑从日志产生的源头上入手,通过改进程序中的结构化输出语句和日志产生的方式尽量降低日志数据中的冗余(例如 ErrorLog[44], LogAdvisor[45] 等),并最终实现日志程序和日志处理程序的协同设计.

同时,我们也希望看到更多其他领域的大数据存储可以结合本领域特征有针对性地设计低成本存储系统,以进一步降低数据存储成本和能源资源消耗.

作者贡献声明:魏钧宇和张广艳提出了写作思路和实现方案;魏钧宇和陈军超负责实现并撰写论文;张广艳提出指导意见并修改论文.

参 考 文 献

[1] Du Min, Li Feiei, Zheng Guineng, et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning[C]//Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2017: 1285−1298

[2] Nagaraj K, Killian C E, Neville J. Structured comparative analysis of systems logs to diagnose performance problems[C]//Proc of Symp on

Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: No. 353

[3] Yuan Ding, Mai Haohui, Xiong Weiwei, et al. Sherlog: Error diagnosis by connecting clues from run-time logs[C]//Proc of the 15th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2010: 143−154

[4] Xu Haowen, Chen Wenxiao, Zhao Nengwen, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications[C]//Proc of the World Wide Web Conf. New York: ACM, 2018: 187−196

[5] Ren Hansheng, Xu Bixiong, Wang Yujing, et al. Time-series anomaly detection service at microsoft[C]//Proc of the 25th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2019: 3009−3017

[6] Geiger A, Liu Dongyu, Alnegheimish S, et al. Tadgan: Time series anomaly detection using generative adversarial networks[C]//Proc of IEEE Int Conf on Big Data (Big Data). Piscataway, NJ: IEEE, 2020: 33−43

[7] Fu Qiang, Lou Jianguang, Wang Yi, et al. Execution anomaly detection in distributed systems through unstructured log analysis[C]//Proc of the 9th IEEE Int Conf on Data Mining. Piscataway, NJ: IEEE, 2009: 149−158

[8] He Shilin, Zhu Jieming, He Pinjia, et al. Experience report: System log analysis for anomaly detection[C]//Proc of IEEE 27th Int Symp on Software Reliability Engineering (ISSRE). Piscataway, NJ: IEEE, 2016: 207−218

[9] Zhang Shenglin, Li Dongwen, Sun Yongqian, et al. Unified anomaly detection for syntactically diverse logs in cloud datacenter[J]. Journal of Computer Research and Development, 2020, 57(4): 778−790 (in Chinese)
(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)

[10] Dumais S, Jeffries R, Russell D M, et al. Ways of Knowing in HCI [M]. New York: Springer, 2014

[11] 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, 2015, 28(3): 592−603

[12] Michael C, David M, Jason F, et al. The mystery machine: End-to-end performance analysis of large-scale Internet services[C]//Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 217–231

[13] Ghassemian M, Hofmann P, Prehofer C, et al. Performance analysis of Internet gateway discovery protocols in ad hoc networks[C]//Proc of IEEE Wireless Communications and Networking Conf. Piscataway, NJ: IEEE, 2004: 120−125

[14] 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: 195−230

[15] Oprea A, Li Zhou, Yen T F, et al. Detection of early-stage enterprise infection by mining large-scale log data[C]//Proc of 2015 45th Annual IEEE/IFIP Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2015: 45−56

[16] The Standing Committee of Twelfth National People's Congress of PRC. 2021-1623139204312 Cybersecurity Law of the People's Republic of China[S]. Beijing: China Legal Publishing House, 2016 (in Chinese)
(第十二届全国人民代表大会常务委员会. 2021-1623139204312 中华人民共和国网络安全法[S]. 北京: 中国法制出版社, 2016)

[17] Wei Junyu, Zhang Guangyan, Wang Yang, et al. On the feasibility of parser-based log compression in large-scale cloud systems[C]//Proc of the 19th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2021: 249−262

[18] Wei Junyu, Zhang Guangyan, Chen Junchao, et al. LogGrep: Fast and cheap cloud log storage by exploiting both static and runtime patterns[C]//Proc of the 18th European Conf on Computer Systems. New York: ACM, 2023: 452−468

[19] Lee G, Lin J, Liu Chuang, et al. The unified logging infrastructure for data analytics at Twitter[J]. Proceedings of the VLDB Endowment, 2012, 5(12): 1771−1780

[20] Slee M, Agarwal A, Kwiatkowski M. Thrift: Scalable cross-language services implementation[J]. Facebook white paper, 2007, 5(8): No. 127

[21] Zip Developer Group. 7-zip file achiever home page [EB/OL]. 2019 [2023-03-13].https://www.7-zip.org/

[22] Cleary J, Witten I. Data compression using adaptive coding and partial string matching[J]. IEEE Transactions on Communications, 1984, 32(4): 396−402

[23] Peter D. DEFLATE compressed data format specification version 1.3. [EB/OL]. 1996 [2023-03-13].https://tools.ietf.org/ html/rfc1951

[24] Facebook Developer Group. z-standard compression tool [EB/OL]. 2021 [2023-03-13].https://github.com/facebook/zstd

[25] Christensen R, Li Feifei. Adaptive log compression for massive log data[C]// Proc of Special Interest Group on Management of Data. New York: ACM, 2013: 1283−1284

[26] Liu Jinyang, Zhu Jieming, He Shilin, et al. Logzip: Extracting hidden structures via iterative clustering for log compression[C]//Proc of 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2019: 863−873

[27] Feng Bo, Wu Chentao, Li Jie. MLC: An efficient multi-level log compression method for cloud backup systems[C]//Proc of IEEE Trustcom. Piscataway, NJ: IEEE, 2016: 1358−1365

[28] Lin Hao, Zhou Jingyu, Yao Bin, et al. Cowic: A column-wise independent compression for log stream analysis[C]//Proc of the 15th IEEE/ACM Int Symp on Cluster, Cloud and Grid Computing. Piscataway, NJ: IEEE, 2015: 21−30

[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, 2011, 24(11): 1921−1936

[30] Tang Liang, Li Tao, Perng C S. LogSig: Generating system events from raw textual logs[C]//Proc of the 20th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2011: 785−794

[31] Mizutani M. Incremental mining of system log format[C]//Proc of 2013 IEEE Int Conf on Services Computing. Piscataway, NJ: IEEE, 2013: 595−602

[32] Hamooni H, Debnath B, Xu Jianwu, et al. Logmine: Fast pattern recognition for log analytics[C]//Proc of the 25th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2016: 1573−1582

[33] Du Min, Li Feifei. Spell: Streaming parsing of system event logs[C]//Proc of IEEE 16th Int Conf on Data Mining (ICDM). Piscataway, NJ: IEEE, 2016: 859−864

[34] Vaarandi R. A data clustering algorithm for mining patterns from event logs[C]//Proc of the 3rd IEEE Workshop on IP Operations & Management. Piscataway, NJ: IEEE, 2003: 119−126

[35] Nagappan M, Vouk M A. Abstracting log lines to log event types for mining software system logs[C]//Proc of the 7th IEEE Working Conf on Mining Software Repositories (MSR 2010). Piscataway, NJ: IEEE, 2010: 114−117

[36] He Pinjia, Zhu Jieming, Zheng Zibin, et al. Drain: An online log parsing approach with fixed depth tree[C]//Proc of IEEE Int Conf on Web Services (ICWS). Piscataway, NJ: IEEE, 2017: 33−40

[37] Messaoudi S, Panichella A, Bianculli D, et al. A search-based approach for accurate identification of log message formats[C]//Proc of the 26th Conf on Program Comprehension. New York: ACM, 2018: 167−177

[38] Zhu Jieming, He Shilin, Liu Jinyang, et al. Tools and benchmarks for automated log parsing[C]//Proc of IEEE 41st Int Conf on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Piscataway, NJ: IEEE, 2019: 121−130

[39] Cao Wei, Feng Xiaojie, Liang Boyuan, et al. Logstore: A cloud-native and multi-tenant log database[C]//Proc of the 2021 Int Conf on Management of Data. New York: ACM, 2021: 2464−2476

[40] Christodorescu M, Kidd N, Goh W H. String analysis for x86 binaries[C]//Proc of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. New York: ACM, 2005: 88−95

[41] Xu Wei, Huang Ling, Fox A, et al. Detecting large-scale system problems by mining console logs[C]//Proc of the 22nd ACM SIGOPS Symp on Operating Systems Principles. New York: ACM, 2009: 117−132

[42] Zhang Maosheng, Zhao Ying, He Zengmingyu. Genlog: Accurate log template discovery for stripped x86 binaries[C]//Proc of IEEE 41st Annual Computer Software and Applications Conf (COMPSAC). Piscataway, NJ: IEEE, 2017, 1: 337−346

[43] Rodrigues K, Luo Yu, Yuan Ding. CLP: Efficient and scalable search on compressed text logs[C]//Proc of USENIX Symp on Operating Systems Design and Implementations. Berkeley, CA: USENIX Association, 2021: 183−198

[44] Yuan Ding, Park S, Huang Peng, et al. Be conservative: Enhancing failure diagnosis with proactive logging[C]// Proc of USENIX Symp on Operating Systems Design and Implementations. Berkeley, CA: USENIX Association, 2012: 293−306

[45] Zhu Jieming, He Pinjia, Fu Qiang, et al. Learning to log: Helping developers make informed logging decisions[C]//Proc of 2015 IEEE/ACM Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2015: 415−425

Wei Junyu, born in 1997. PhD candidate. His main research interests include log management and data compression.

魏钧宇, 1997 年生. 博士研究生. 主要研究方向为日志管理和数据压缩.

Zhang Guangyan, born in 1976. PhD, associate professor, PhD supervisor. Distinguished member of CCF, winner of the National Science Fund for Distinguished Young Scholars. His current research interests include big data computing, storage systems, and distributed processing.

张广艳, 1976 年生. 博士, 副教授, 博士生导师. CCF 杰出会员, 国家杰出青年科学基金获得者. 主要研究方向为大数据计算、存储系统和分布式处理.

Chen Junchao, born in 1986. Received his master's degree from the Department of Computer Science and Technology, Tsinghua University in 2023. His main research interests include data compression, and big data storage and management.

陈军超, 1986 年生. 2023 年获清华大学计算机科学与技术系硕士学位. 主要研究方向为数据压缩和大数据存储管理.