華中科技大學

数据中心技术文献综述报告

院　　系 计算机科学与技术学院

班　　级 2106

学　　号 M202173723

姓　　名 李贽

2021 年　12 月 31　日

云存储压缩算法综述

李贽

摘　要　随着云存储的应用场景越来越广泛，世界上产生数据的速率越来越快，在云端信息爆炸早已成为常态。系统日志记录软件系统的详细运行时信息，并用作软件工程中许多任务的主要数据源。随着现代软件系统向大规模、复杂的结构发展，日志已成为行业中快速增长的大数据之一。特别是，在实践中，此类日志通常需要存储很长时间（例如一年），以便分析反复出现的问题或跟踪安全问题。但是，归档日志会消耗大量的存储空间和计算资源，这反过来又会带来高昂的运营成本。数据压缩对于降低日志存储成本至关重要。传统的压缩工具（如 gzip）适用于一般文本，但不适用于系统日志。本报告聚焦于多种压缩方法，即 LogReducer 和 Logzip，对云存储压缩算法进行简单的综述。对 18 种生产日志和 16 种公共日志的评估表明，LogReducer 在几乎所有情况下都实现了最高的压缩比，在大型日志上，其速度可与以高压缩比为目标的通用压缩算法相媲美。Logzip 能够通过快速迭代聚类从原始日志中提取隐藏结构，并进一步生成连贯的中间表示，从而实现更有效的压缩。在五个总大小为 63.6GB 的不同系统类型的大型日志数据集上评估 Logzip 后，结果表明，与传统的压缩工具相比，Logzip 平均可以节省大约一半的存储空间。同时，Logzip 的设计是高度并行的，只会产生微不足道的开销。

关键词　云存储；压缩算法；LogReducer；Logzip

Title Research On Cloud Storage Compression Algorithm

Zhi Li

Abstract　With the more and more extensive application scenarios of cloud storage, the data generation rate in the world is faster and faster, and the information explosion in the cloud has long become the norm. The system log records the detailed runtime information of the software system and is used as the main data source for many tasks in software engineering. With the development of modern software system to large-scale and complex structure, log has become one of the fast-growing big data in the industry. In particular, in practice, such logs usually need to be stored for a long time (for example, one year) to analyze recurring problems or track security problems. However, archiving logs consumes a lot of storage space and computing resources, which in turn brings high operating costs. Data compression is very important to reduce the cost of log storage. Traditional compression tools (such as gzip) are suitable for general text, but not for system logs. This report focuses on a variety of compression methods, namely logreducer and Logzip, and briefly summarizes cloud storage compression algorithms. The evaluation of 18 production logs and 16 public logs shows that logreducer achieves the highest compression ratio in almost all cases. On large logs, its speed is comparable to the general compression algorithm aiming at high compression ratio. Logzip can extract the hidden structure from the original log through fast iterative clustering, and further generate a coherent intermediate representation, so as to achieve more effective compression. After evaluating Logzip on five large log data sets of different system types with a total size of 63.6gb, the results show that Logzip can save about half of the storage space on average compared with traditional compression tools. At the same time, the design of Logzip is highly parallel and will only incur insignificant overhead.

Key words　cloud storage; compression algorithm; parser-tree

一、　引言

系统日志通常包括一系列日志消息，每个日志消息都记录了用户应用程序和大型系统组件执行期间的特定事件或状态。这些日志在许多软件工程任务中有着广泛的应用。它们不仅对于系统操作员诊断运行时故障、识别性能瓶颈和检测安全问题至

关重要，而且对于服务提供商跟踪使用统计数据和预测市场趋势也具有潜在的价值。

如今，日志已经成为行业中快速增长的大数据之一。随着系统规模和复杂性的增长，日志的生成速度也在不断提高。例如，无论是在云端（例如，数据中心承载数千台机器）还是在客户端（例如，智能手机供应商在全球拥有数百万台智能设备），这些系统通常在一天内生成数十 TB 的日志。大量日志很容易导致每年数 PB 的数据增长。此外，为了提高存储弹性，每个日志通常会复制到多个副本中，例如在 HDFS 中。日志数据的某些重要部分甚至可以跨至少两个独立的数据中心进行同步，以实现灾难恢复。这对存储系统的容量造成了严重的压力。此外，许多日志需要长期存储，根据软件产品的开发生命周期，通常需要一年或更长时间。历史日志有助于发现故障模式和识别反复出现的问题。例如，许多用户经常重新发现旧问题，因为他们没有安装补丁包。同时，记录用户和管理员执行的敏感操作的审核日志通常需要保存至少两年，以便将来跟踪系统误用情况。尽管存储成本比以前低了很多，但在如此巨大的容量中归档日志仍然非常昂贵。它不仅占用了大量的存储空间和电能，而且还消耗了用于传输和复制的网络带宽。

为了降低日志数据的沉重存储成本，存在两个数据缩减方向：1）从源代码中缩减日志，2）日志压缩。通过请求开发人员打印更少的日志语句并设置适当的详细级别（例如，信息和错误），可以大大减少日志。然而，记录太少可能会丢失一些关键信息，并导致意外后果。如何建立一个最佳的测井标准仍然是一个有待解决的问题。通常的做法是在将数据存储到磁盘之前应用压缩。主流压缩方案（例如 gzip 和 bzip）通常可以将日志大小减少 10 倍。这些通用压缩算法允许对任意二进制序列进行编码，但只能利用短滑动窗口内的冗余信息（例如，gzip 的 Deflate 算法中的 32KB）。因此，它们无法利用日志消息的固有结构来实现更有效的压缩。

与传统的压缩方法相比，Logzip 可以利用系统日志的固有结构，以更高的压缩比压缩大型日志文件。日志消息由特定的日志语句打印，因此每个日志语句都有一个固定的消息模板。Logzip 的核心思想是从原始日志中自动提取这样的消息模板，然后将它们结构化为更适合通用压缩算法的连贯的中间表示。为了实现这一点提出了用于结构提取的迭代聚类算法，该算法遵循采样、聚类和匹配的迭代过程。Logzip 通过字段提取、模板提取和参数映射进一步生成三级中间表示。这些转换后的表示形式被进一步输入到传统的压缩方法中进行最终压缩。整个 Logzip 过程被设计为高效和高度并行的。作为一个副作用，Logzip 的结构化中间表示可以直接用于许多下游任务，例如日志搜索和异常检测，而无需进一步处理。

而另外一种算法，LogReducer，被应用于 18 种阿里云日志（总计 1.76TB）和 16 种公共日志（总计 77GB）。与 LZMA 相比，LogReducer 在所有情况下都可以实现 1.19 到 5.30 倍的压缩比，在 100MB 以上的日志上可以实现 0.56 到 3.16 倍的压缩速度（LogReducer 在较小的日志上由于其初始化开销而相对较慢）。与 Logzip 相比，LogReducer 可以实现 1.03×4.01×压缩比和 2.05×182.31×压缩速度。这些结果证实，通过适当的实现和优化，基于 parserbase 的日志压缩有望压缩大规模生产日志。

二、 原理与优势

2.1 基本前置概念与原理

2.1.1 日志基本结构

日志的基本结构由三部分组成：日志头、模板和变量。比如“Write chunk %s Offset %d Length %d”为文章给出的一种模板，一个实例是 Write chunk: 3242513_B Offset: 339911 Length: 11 这样一条日志。

2.1.2 解析器与解析器树

基于解析器的压缩方法建立在解析器树（如图 1）上。

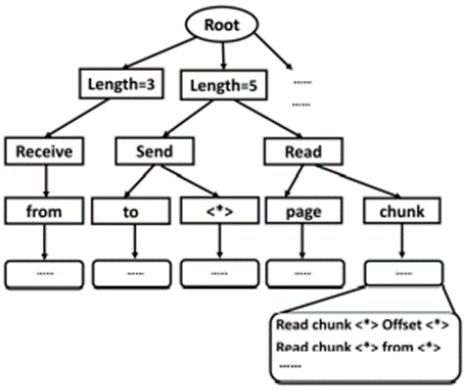


图 1 解析器树

给定模板和日志项的列表，将条目与模板匹配的简单方法是将条目与每个模板进行比较，并找到与条目最相似的模板。基于解析器的日志压缩器首

先通过解析日志项的样本来构建解析器树。

在第一步中，日志解析器使用预定义的拆分字符（如空格或逗号）将日志条目拆分为一个称为标记的字符串列表。在文章的示例中，原始日志消息将被相应地划分为“Read”、“chunk”、“3242514_C”、“Offset”、“272633”。在第二步中，日志解析器将检查长度的内部节点是否存在。如果没有，日志解析器将创建一个新的内部节点。最后，它移动到相应的内部节点。在第三步中，日志解析器根据日志条目中的标记遍历树，并移动到相应的内部节点（在文章的示例中为“Read”和“chunk”）。到达树深度限制后，它到达一个包含一组模板的叶节点。如果相应的内部节点不存在，日志解析器将构建内部节点并将该节点添加到前缀树中。

2.1.3 相似度

一个日志和某一个模板之间的相似度（Similarity）被定义为：

$$Similarity(L,T)=\frac{\sum \emptyset(l_i,t_i)}{|L|}$$

如果一个日志和某个模板的相似度大于设定的阈值，则将日志套入模板，否则在解析器树上创建新的结点。

2.1.4 迭代结构提取

从日志中提取模板有三种主要方法：（1）从日志中手动构造模板；（2） 从源代码中的日志语句中提取；（3）从原始日志中提取。在实践中，软件日志具有复杂的隐藏结构，并且是大规模的。因此，手动构造模板是劳动密集型的，并且容易出错。此外，系统特定组件的源代码通常无法访问（例如，第三方库）。因此，从软件日志中提取模板是应用最广泛的方法，因此 Logzip 提出了一种迭代聚类算法来自动从日志中提取模板。根据基准测试，现有的模板提取方法可以在软件日志上准确执行。然而，这些方法需要所有历史日志作为输入，这导致了严重的效率低下，并阻碍了它们在实践中的应用。受级联聚类的启发，迭代结构提取（ISE）只从一小部分历史日志中有效地提取模板。图 2 显示了 ISE 的概述。

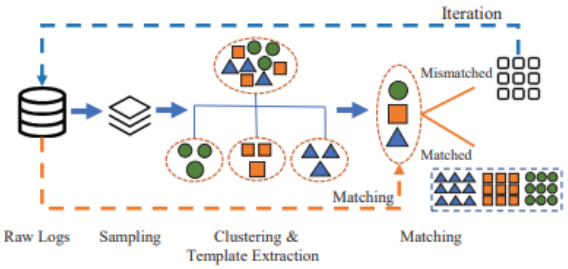


图 2 ISE 的概述

ISE 是一种迭代算法，每次迭代包含 3 个步骤：采样、聚类和匹配。ISE 的输入是由原始日志消息组成的日志文件，输出是提取的模板和结构化日志。具体地说，在迭代中，我们首先对输入日志的一部分进行采样。然后对样本日志应用分层划分方法生成多个聚类，从中可以自动提取模板。在匹配步骤中，我们尝试将所有未采样的原始日志与这些模板匹配，收集未匹配的日志，并将它们作为输入提供给下一次迭代。通过迭代这些步骤，所有日志消息都可以通过适当的模板分配准确高效地匹配。这背后的原因是，在每次迭代中，采样日志通常可以覆盖隐藏在大多数输入日志中的模板。特别是，日志记录语句中的一小部分可以比其他语句执行得更频繁。因此，从一小部分日志生成的模板通常可以在最初的几个迭代中匹配大多数原始日志。

三、 研究与进展

3.1 解析器树算法

压缩器使用解析器树来压缩日志。该过程类似于构建解析器树，只是在此阶段，压缩器不会更新解析器树。它将首先利用解析器树尝试将每个日志条目与模板匹配。如果找到匹配项，日志条目将转换为模板 ID 和变量；如果没有匹配的模板，日志条目将被视为不匹配，并且不会被转换。

此外解析器树压缩算法有三种优化方法：

3.1.1 修剪解析器树

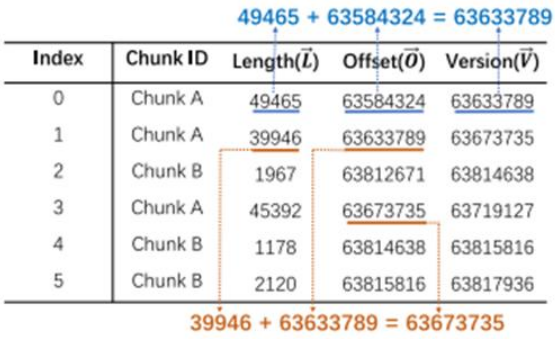
在发现日志的模板数一般较小后，文章将模板按长度组织起来，在每一组中得到了更少数量的解析器树。因此在压缩阶段，文章将解析器树修剪到只有一层，从而大大提高了解析器树的表现。

3.1.2 批处理

如果是顺序压缩大量小日志文件，不停地开始和结束压缩进程会拖慢压缩速度，所以文章允许压缩器一次处理一批文件，从而提高压缩效率。

由于文章的研究对象阿里云要求日志文件包含确切的时间信息，因此文章时间戳在日志中往往是占据主要地位的数字数据。为了节省压缩时间，文章用记录时间戳之间的变化值而非绝对值以缩减时间戳的数据量。

文章还发现，用户进行一系列连续的 I/O 操作时，下一个 I/O 的起始值是以往数据的终点值。比如下图所示：



Index	Chunk ID	Length($\vec{L}$)	Offset($\vec{O}$)	Version($\vec{V}$)
0	Chunk A	49465	63584324	63633789
1	Chunk A	39946	63633789	63673735
2	Chunk B	1967	63812671	63814638
3	Chunk A	45392	63673735	63719127
4	Chunk B	1178	63814638	63815816
5	Chunk B	2120	63815816	63817936

图 3 数据规律

3.1.3 相关关系及相关关系识别算法

对于三个变量$\vec{V}$，$\vec{L}$，$\vec{O}$分别代表版本，长度和起始值。存在三种相关关系：一、变量间相关关系：$\vec{V}=\vec{L}+\vec{O}$。二、变量内相关关系：$\vec{L}[i]=\vec{L}[i-1]+\Delta\vec{L}$。三、混合相关关系：$\vec{O}=\vec{O}[i-1]+\vec{L}[i-1]$，剩余向量：$\vec{O}-\vec{O}[i-1]-\vec{L}[i-1]=\Delta\vec{O}-\vec{L}$。借助这三种文章原创的相关关系，压缩器可以在一定程度上使压缩更有效率。

相关关系识别算法：目标集合Ψ，恢复集合$\mathbb{R}$（包含所有能从Ψ中恢复的原始向量），所有的可能向量的集合T包含了所有可能出现的向量。map($\vec{C}$)函数将返回构造出$\vec{c}$向量的向量（例如map$(\vec{A}-\vec{B})=\vec{A}$和$\vec{B}$）

变量向量的香农熵$E(\vec{A})$：$E(\vec{A})=-\sum_{s\in S_A}\frac{\#(s)}{|A|}log\frac{\#(s)}{|A|}$，其中$S_A$表示$\vec{A}$中出现的所有值，$\#(s)$表示$s$在$\vec{A}$中出现的次数。

Algorithm 1 Correlation identification algorithm

1: Recoverable set $\mathbb{R}=\emptyset$
2: Final vector set $\Psi=\emptyset$
3: Initialize candidate set $\mathbb{T}$
4: **repeat**
5: $\mathbb{C}=\{\vec{C}\in\mathbb{T}:|map(\vec{C})-\mathbb{R}|=1\}$
6: $\vec{C}_{min}$ = vector with the smallest entropy in $\mathbb{C}$.
7: $\Psi\leftarrow\Psi\cup\vec{C}_{min}$
8: $\mathbb{R}\leftarrow\mathbb{R}\cup map(\vec{C}_{min})$
9: **until** $\mathbb{R}$ contains all original vectors
10: Output Ψ

图 4 相关性识别算法

在每次迭代中，它首先尝试找到与当前可恢复集$\mathbb{R}$（第 5 行）相比能够恢复一个以上变量的所有候选向量$\vec{C}$；然后在其中选择压缩比最高的一个（第 6 行）。在这里，文章使用香农熵预测候选压缩比；最后它相应地更新了Ψ和$\mathbb{R}$（第 7 行和第 8 行）；它重复这个过程，直到Ψ可以恢复所有原始向量（第 9 行）。枚举的成本是可以接受的，因为它是在日志样本上执行的。

训练阶段数字相关关系的输出是目标向量Ψ，在压缩阶段算法将计算Ψ中的每一个剩余向量并且丢弃没有在Ψ中出现的原始向量。将三种相关关系应用到算法中得到结果如下：

Index	Chunk ID	$\Delta\vec{L}$	$\Delta\vec{O}-\vec{L}$	$\vec{V}-\vec{O}-\vec{L}$
0	Chunk A	49465	0	0
1	Chunk A	-9519	0	0
2	Chunk B	1967	0	0
3	Chunk A	5446	63673735	0
4	Chunk B	-789	0	0
5	Chunk B	942	63815816	0

图 5 相关性识别算法得到的结果

可以发现的是如果某些变量很好地满足某一规则，它们的剩余变量就会有许多项为 0。就算其他变量并没有很好地满足任一规则，剩余变量的值也很小。

3.1.4 弹性编码器

固定大小编码（如整数用四个字节，长整型数用八个字节）在大多数数据值较小时会导致数据前面有很多个 0（表示整数时）或 1（表示负数时），导致空间的浪费。文章将 32 比特的整数缩减到 7 比特大小的片段，除此之外在每个片段添加 1 比特的内容来标注该片段是否为最后一段（用 1 表示该片段为最后一段），然后就可以将只包含 0 的片段前

缀丢弃。片段大小为 7 的原因是每个片段加上 1 比特的标注之后总共占一个字节，易于管理。对于负数而言，只需要把第一个比特移到最后一位然后把所有比特的数据翻转即可。具体来说，对于属于 $[-2^{7n}, -2^{7(n-1)}) \cup (2^{7(n-1)} - 1, 2^{7n} - 1]$ $(0 < n < 6)$，与固定 32 比特相比弹性编码器节省（32-8n）比特。在文章的测试中，60%以上的数据都能节省 24 字节（n=1）。

3.1.5 小结

与传统的 Logzip 等方法只在训练阶段利用随机采样的数据训练模型不同，LogReducer 考虑相邻的数据直接的相关关系，而随机抽样会忽视这种相关关系。因此 LogReducer 在采样时先随机选定一些起点，然后从这些起点开始连续地选择日志。这一方法的表现很好。而在压缩阶段，对于每个日志条目，LogReducer 将首先提取其头部。LogReducer 将从报头中提取时间戳，并计算连续时间戳的增量值。然后，LogReducer 将尝试使用解析器将日志条目与模板匹配，并将建立的相关性应用于数值变量。然后 LogReducer 将使用弹性编码器对所有数字数据进行编码，包括时间戳、数字变量和模板 ID。最后，LogReducer 将使用 LZMA 打包所有数据，因为它几乎总能在日志上实现最高的压缩比。

3.2 Logzip 算法

3.2.1 Logzip 综述

Logzip 能够通过快速迭代聚类从原始日志中提取隐藏结构，并进一步生成连贯的中间表示，从而实现更有效的压缩。

Logzip 背后的主要思想是减少原始日志文件中包含的冗余信息。图 6 描述了 Logzip 的总体框架。

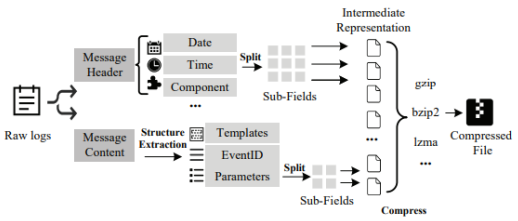


图 6 Logzip 的总体框架

对于日志数据中的每个原始日志消息，首先通过手动定义的正则表达式将其结构化为消息头和消息内容。然后，根据字段和其他子字段将消息头拆分为多个对象。对于消息内容，通过应用 ISE 提取隐藏结构。之后，我们将每个日志消息表示为模板、事件 ID 以及相应的参数。此外，参数列表中的每个项目都被拆分为子字段。然后，共享相同事件 ID 的日志以紧凑的方式存储到同一对象中。最后，利用现有的压缩工具将所有生成的对象压缩成压缩文件。

3.2.2 Logzip 实现方法

Logzip 可以在 3 个级别执行压缩，并实现不同的有效性和效率。图 7 是 Logzip 工作流的示例。

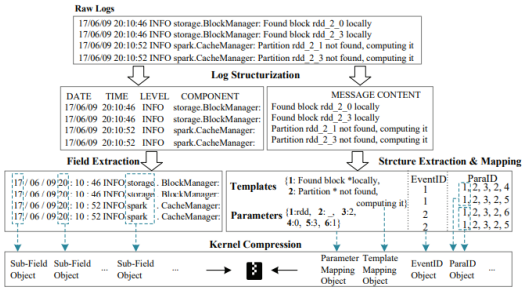


图 7 Logzip 工作流的示例

级别 1：字段提取。我们首先通过应用用户定义的正则表达式来提取给定原始日志的字段。对于图 5 中的示例，可以通过识别空白分隔符来提取“数据”、“时间”和“级别”。“组件”和“消息内容”由冒号分隔。我们强调，手动构造正则表达式很容易，对于特定的系统，正则表达式通常保持不变，因为日志框架会自动记录消息头（例如，Java 中的 log4j，Python 中的 logging）。然后，根据特殊字符（例如，非字母字符）将每个字段进一步划分为子字段，以增加子字段中的一致性。然后将这些子字段存储到单独的对象中。对于级别 1，不处理消息内容并将其存储到对象中。

第 2 级：模板提取。通过提取隐藏结构（即消息模板），进一步处理消息内容。如第三节所述，我们直接应用 ISE。之后，消息内容可以表示为其模板和参数，我们为唯一模板分配初始化为 0 的自增量 EventID，这形成了模板映射字典（EventID 是键，对应的模板是值）。事实上，一个模板可能由许多日志消息共享。例如，我们研究的 HDFS 日志包含大约 1120 万条日志消息，但它们只共享 39 个模板。因此，我们使用相应的 EventId 来表示每个日志消息的模板。在此过程中，日志消息被转换为包含短 EventId 和参数的紧凑形式。最后，模板映射字典单独存储在一个对象中，而 EventId 存储在一个 EventId 对象中。对于从消息内容中提取的参数，我们以与级别 1 中类似的方式拆分每个参数项。显然，每个参数都以非字母字符作为分隔符进行分割。然后，组中每个生成的子字段分别构成一个对

象。这里，一个组表示共享同一模板的所有日志。背后的直觉是，一个组中的参数可能是重复的或类似的，将类似的项放入一个文件可以充分利用现有的工具，例如 gzip。

第 3 级：参数映射。我们进一步优化了 3 级中的参数表示。根据我们的观察，一些不可分割且非常长的参数（即内部没有分隔符）浪费了太多空间。例如，块 ID（例如，“blk-5974833545991408899”）占用空间，并且可能具有高发生率。为了避免这个问题，将唯一的子字段值编码为顺序 64 个基数（ParaID），这形成了一个参数映射字典（ParaID 是键，对应的参数是值）。为了节省更多空间，来自所有组的参数共享同一个参数映射字典。总之，在第 3 级，参数被编码到 ParaID 中。最后，分别为每个组生成一个参数映射字典对象和 ParaID 对象。

经过三个级别的拆分、编码和映射，生成了多个对象。最后一步是将所有这些对象打包为压缩文件，而不丢失任何信息。由于我们的主要兴趣在于前面提到的三个处理级别，因此在这一步中我们直接使用那些离线压缩算法和工具，例如 gzip、bzip2 和 LZMA。通过这种方式，我们的 Logzip 与现有的压缩工具和算法兼容。值得注意的是，大多数日志分析算法（例如，异常检测）将模板作为输入，而不带参数。因此，在这种情况下，Logzip 可以通过在使用现有工具进行压缩之前丢弃所有参数对象来执行有损压缩，这可能更有效。减压。作为日志压缩的反向过程，解压缩应该能够在不丢失任何信息的情况下恢复原始数据集。首先，解压缩压缩文件后会生成多个对象。然后，我们通过简单地按顺序合并在级别 1 中提取的所有子字段值来恢复消息头。至于恢复消息内容，我们首先通过使用 EventID 索引事件编码字典来获取模板。类似地，通过使用 ParaID 索引参数编码字典来检索参数列表。通过按顺序使用参数替换模板中的“*”，可以完全恢复消息内容。

3.2.3 小结

Logzip 是一种日志压缩方法，可以大大降低日志存储的操作成本。Logzip 通过一种新的迭代聚类技术提取和利用日志的固有结构。Logzip 旨在与现有的数据压缩实用程序（如 gzip）无缝集成。此外，Logzip 生成的半结构中间表示可以直接用于各种下游日志挖掘任务（例如，异常检测）。在五个真实世界的系统日志数据集上进行了大量实验，以评估 Logzip 的有效性。实验结果表明，与三种广泛使用的数据压缩工具相比，Logzip 显著提高了压缩比，并且优于最先进的日志压缩方法。此外，Logzip 高度并行，在 32 核机器上实现了与 gzip 相当的效率。开源的 Logzip 工具可以使面临同样问题的工程团队受益。

3.3 LZMA 算法

7z 是一种文件压缩格式，具有高压缩比率，它采用了多种压缩算法进行数据压缩。因此，与其它压缩格式相比，得到的压缩文档较小。现在流行的好压软件支持这种压缩格式。

LZMA 是 7z 格式默认的压缩算法，它的主要特征有：高压缩比率；可变的字典大小（高达 4 GB）；压缩速度在 2 GHz CPU 上，大约为 1 MB/s；解压缩速度在 2 GHz CPU 上，大约为 10-20 MB/s ；解压缩内存较小（依赖于所选的字典大小）；解压缩代码较小，大约 5 KB；支持多线程；

四、 总结与展望

基于解析器树的压缩算法 LogReducer 主要针对模板数量少、变量多的大型日志而设计。当这些假设成立时，LogReducer 可以比现有方法表现得更好；当这些假设不成立时，LogReducer 的效率较低，但仍然可以实现最高的压缩比。

随着云端数据越来越多，云存储场景越来越多样，针对特定场景量身定制的云存储压缩方法势必会在自己的领域具有更大的优势。因此随着应用场景的多样化，云存储的算法也会越来越多样，模型建立的假设会越来越丰富多变，因此不妨从实际问题出发，针对特定的数据设计特定的压缩算法。

参 考 文 献

[1] Wei J, Zhang G, Wang Y, et al. On the Feasibility of Parser-based Log Compression in Large-Scale Cloud Systems[C]//19th {USENIX} Conference on File and Storage Technologies ({FAST} 21). 2021: 249-262.

[2] Zhu J, He S, Liu J, et al. Tools and benchmarks for automated log parsing[C]//2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2019: 121-130.

[3] Liu J, Zhu J, He S, et al. Logzip: extracting hidden structures via iterative clustering for log compression[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 863-873.