华中科技大学

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。计算能力跟不上数据产生速率的问题时有发生。为此服务器不得不将数据先存放起来，日后再进行分析处理。因此如何高效地压缩数据成为了亟待解决的问题，本报告聚焦于多种压缩方法，对云存储压缩算法进行简单的综述。

关键词　云存储；压缩算法；解析器树

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 problem that computing power can not keep up with the data generation rate occurs from time to time. Therefore, the server has to store the data first and then analyze and process it in the future. Therefore, how to efficiently compress data has become an urgent problem to be solved. This report focuses on a variety of compression methods and briefly summarizes cloud storage compression algorithms.

Key words　cloud storage; compression algorithm; parser-tree

一、　引言

大多数系统出于各种原因记录内部事件，例如诊断系统错误，分析用户行为，建模系统性能，以及检测潜在的安全问题。在今天的数据中心中，此类日志的大小可能会变得很大。由于多种原因，这些日志通常需要存储很长时间。比如有时检测到的异常比记录的异常晚得多，因此开发人员需要分析以往的日志。或是某些分析可能需要长时间的统计数据才能得出结论。因此云端会产生大量的日志文件。为了压缩这些日志文件以节约空间，本文考虑多种压缩方法，对这些算法进行总结。

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。

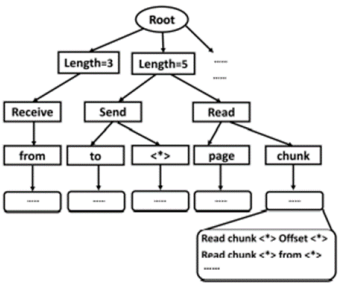

图 1 解析器树

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。

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：

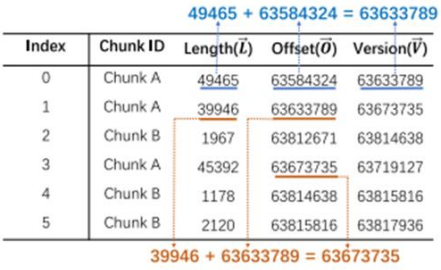

图 2 数据规律

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$\mathbf{T}$包含了所有可能出现的向量。$\mathrm{map}(\vec{C})$函数

将返回构造出$\vec{C}$向量的向量（例如$\text{map}(\vec{A}-\vec{B})=\vec{A}$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 3 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	$\overrightarrow{\Delta L}$	$\overrightarrow{\Delta O}-\vec{L}$	$\vec{V}-\vec{O}-\vec{L}$
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 4 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

$$[-2^{7n},-2^{7(n-1)})\cup(2^{7(n-1)}-1,2^{7n}-1]\quad(0<n<6)$$

，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、LZMA算法

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。

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 4GB）；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 5KB；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.