On the Feasibility of Parser-based Log Compression in Large-Scale Cloud Systems

01 Backgrounds

02 Model

03 Conclusion

Backgrounds

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，文章提出了基于解析器（Parser-based）的日志压缩方法。

Backgrounds

LZMA, gzip, PPMd, 和 bzip，这四种算法通过识别并替换重复的字节以压缩文件。

最新的基于解析器树的算法Logzip在文章的日志数据集上的表现不够优秀。 Logzip比LZMA的速度慢七倍，共计18类日志数据中，Logzip在其中13种日志的压缩率不如LZMA。究其原因，Logzip限制了日志长度，因此本文的算法在大规模的日志数据上表现更好。

Model

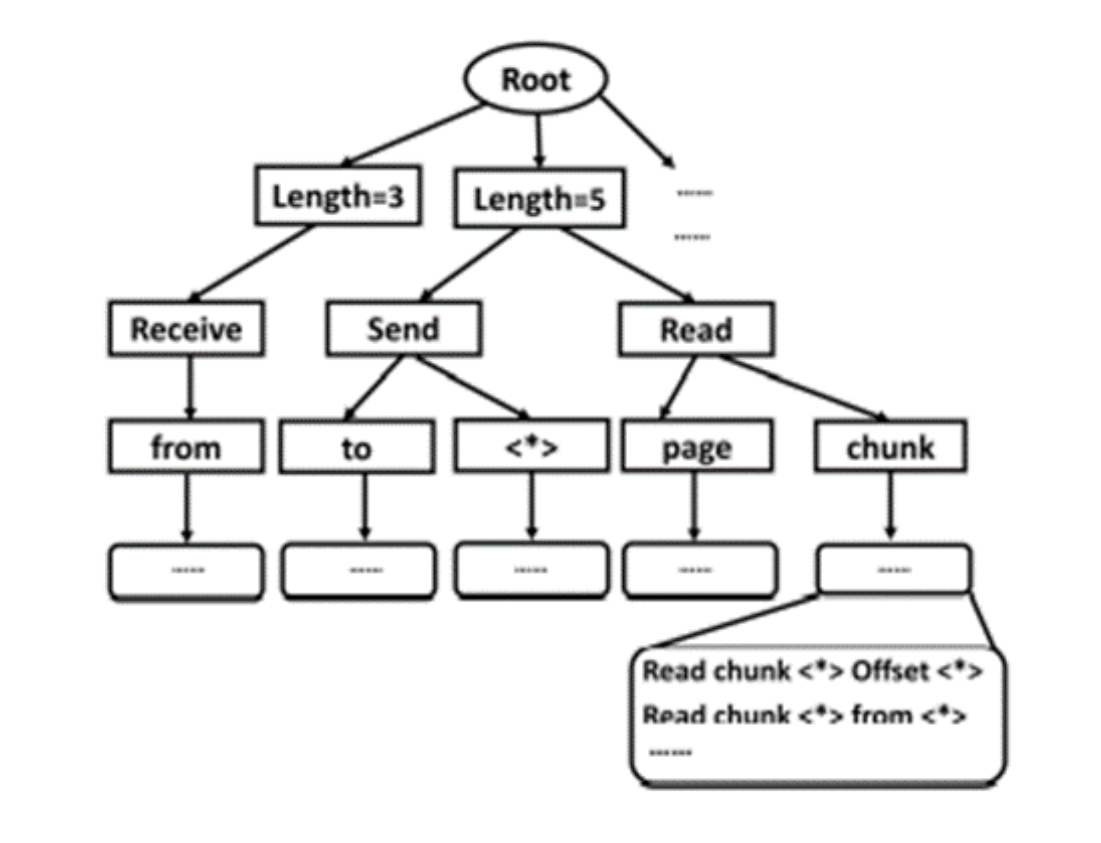
日志基本结构

文章首先提出日志的基本结构由三部分组成：日志头（包含时间信息，日志等级等信息）、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。

Model

解析器树（parser tree）

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。



Model

构建解析器树

第一步：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”。
第二步：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。
第三步：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。

Model

相似度

一个日志L和某一个模板T之间的相似度（Similarity）。

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

l_i和t_i分别是日志和模板的第i个标志，|L|是日志的总标志数。 $l_i=t_i$时， $\emptyset(l_i,t_i)=1$，反之$\emptyset(l_i,t_i)=0$。

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。

Model

压缩日志

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。

Model

压缩过程中两种优化方法：

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。

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：

49465 + 63584324 = 63633789

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

39946 + 63633789 = 63673735

Model

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：$\vec{O}\,[i]=\vec{O}\,[i-1]+\vec{L}[i-1]$，剩余向量：$\vec{O}\,[i-1]-\vec{L}[i-1]=\Delta\vec{O}-\vec{L}$。借助这三种相关关系，压缩器可以在一定程度上使压缩更有效率。

对于目标集合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: $\quad\mathbb{C}=\{\vec{C}\in\mathbb{T}:|map(\vec{C})-\mathbb{R}|=1\}$
6: $\quad\vec{C}_{min}$ = vector with the smallest entropy in $\mathbb{C}$.
7: $\quad\Psi\leftarrow\Psi\cup\vec{C}_{min}$
8: $\quad\mathbb{R}\leftarrow\mathbb{R}\cup map(\vec{C}_{min})$
9: **until** $\mathbb{R}$ contains all original vectors
10: Output Ψ

Model

算法实现过程：

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：
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。

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: $\quad \mathbb{C} = \{\vec{C} \in \mathbb{T} : |map(\vec{C}) - \mathbb{R}| = 1\}$
6: $\quad \vec{C}_{min}$ = vector with the smallest entropy in $\mathbb{C}$.
7: $\quad \Psi \leftarrow \Psi \cup \vec{C}_{min}$
8: $\quad \mathbb{R} \leftarrow \mathbb{R} \cup map(\vec{C}_{min})$
9: **until** $\mathbb{R}$ contains all original vectors
10: Output Ψ

Index	Chunk ID	$\overrightarrow{\Delta L}$	$\overrightarrow{\Delta O} - \overrightarrow{L}$	$\overrightarrow{V} - \overrightarrow{O} - \overrightarrow{L}$
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

Model

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）。

Conclusion

实验及结果:

文章根据以上算法思想建立了LogReducer。具体操作过程分为训练和压缩两个阶段。训练阶段利用示例数据，抽取模板并寻找可能的相关关系。
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，因为文章认为它几乎总能在日志上实现最高的压缩比。

Conclusion

为了说明整个压缩过程，文章展示了一个完整的压缩案例。假设我们有下图中所示的四个log F输入日志条目，以及在培训阶段建立的模板和相关性。
LogReducer首先提取它们的头并将它们的主体与模板匹配。结果如下表所示。
其中每个日志条目分为三个部分，即日志头、模板ID和相应的变量。这里，第二个日志项属于模板：“Read chunk <*> Offset:<*>”，其模板ID为2，其他三个日志项属于模板：“Write chunk <*> Offset:<*> Length:<*>”，其模板ID为1。因此，模板2有两个变量，模板1有三个变量。

User behavior tracing (Log F)
[2019-08-27 15:21:24.456234] [INFO] Write chunk: 3242513_B Offset: 339911 Length: 11
[2019-08-27 15:21:24.463321] [INFO] Read chunk: 3242514_C Offset: 272633
[2019-08-27 15:21:24.464322] [INFO] Write chunk: 3242512_F Offset: 318374 Length: 7
[2019-08-27 15:21:24.474433] [INFO] Write chunk: 3242513_B Offset: 339922 Length: 55

Headers	Template ID	V1	V2	V3
[2019-08-27 15:21:24.456234] [INFO]	1	3242513_B	339911	11
[2019-08-27 15:21:24.463321] [INFO]	2	3242514_C	272633	-
[2019-08-27 15:21:24.464322] [INFO]	1	3242512_F	318374	7
[2019-08-27 15:21:24.474433] [INFO]	1	3242513_B	339922	55

Conclusion

然后LogReducer将计算相邻时间戳的差异，并利用数值变量的相关性。除第一个时间戳外，所有时间戳都变得更小；由于LogReducer识别3242513_B的顺序访问模式，因此它不需要存储第二次访问的偏移量，并且我们计算相同块的写入长度的增量结果（即日志条目4）。最后，LogReducer使用弹性编码器对所有数值结果进行编码，以列方式组织所有变量，并使用LZMA对其进行打包。

用$\frac{\text{Original size}}{\text{Compression size}}$作为评价指标，分别测试了LogReducer的整体表现、LogReducer的各种技术（如裁剪解析器树、批处理、时间戳增量等）的性能、LogReducer在阿里云日志上的性能。

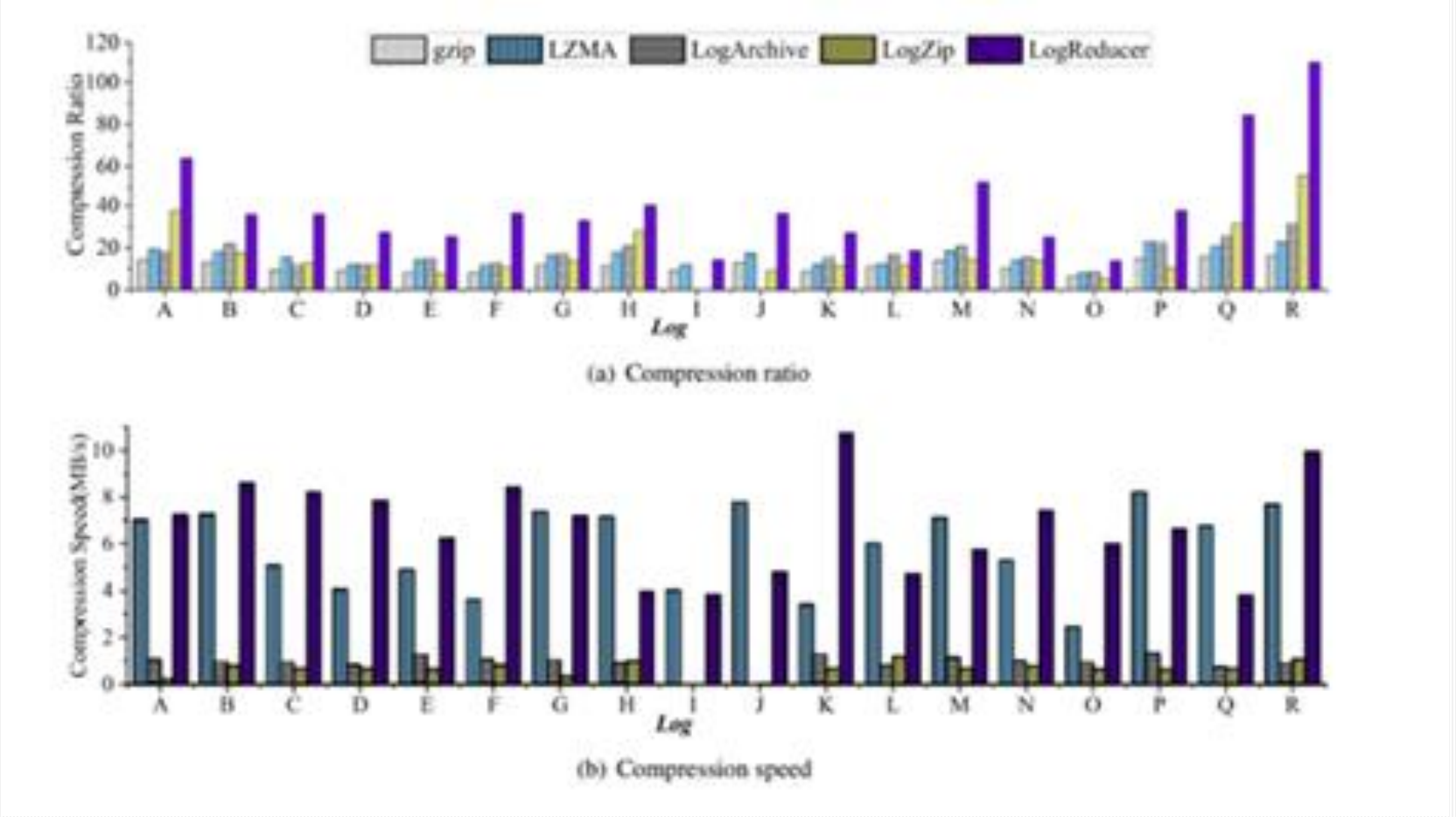
Conclusion

日志的情况：

Log type	Log A	Log B	Log C	Log D	Log E	Log F	Log G	Log H	Log I
Total Size(GB)	18.67	16.05	45.82	65.74	34.98	443.30	148.88	0.19	14.20
Total Line(10^6)	74.74	72.60	231.43	406.98	77.56	1425.37	579.94	1.08	8.65
Time Span(H)	476	75	87	20	6	8	1563	32	8977
Log type	**Log J**	**Log K**	**Log L**	**Log M**	**Log N**	**Log O**	**Log P**	**Log Q**	**Log R**
Total Size(GB)	18.67	16.05	45.82	65.74	34.98	443.30	148.88	0.19	14.20
Total Line(10^6)	74.74	72.60	231.43	406.98	77.56	1425.37	579.94	1.08	8.65
Time Span(H)	85	3335	238	30	174	1512	62	165	722

Conclusion

各算法压缩率和压缩速度的比较：



Conclusion

不同的处理技巧的表现：

	Compression Ratio						Compression Speed (MB/s)						
	LZMA	Logzip	B & NB	D	ED	LR	LZMA	Logzip	NB	B	D	ED	LR
Log A	19.30	37.34	53.96	61.94	63.79	63.86	7.03	0.22	3.99	6.42	6.31	7.58	7.23
Log B	17.91	17.64	32.66	33.55	35.63	35.67	7.25	0.79	3.80	6.75	7.21	9.52	8.63
Log C	15.48	12.61	30.36	32.30	34.80	35.81	5.06	0.68	3.47	6.31	6.17	9.50	8.22
Log D	12.16	11.57	23.08	24.50	26.56	27.26	4.08	0.66	2.84	5.64	5.29	9.80	7.83
Log E	14.19	7.73	22.99	23.35	24.73	25.22	4.89	0.64	4.33	5.34	5.42	6.93	6.22
Log F	11.58	10.69	16.32	16.47	17.62	36.42	3.60	0.81	3.32	4.33	4.33	8.00	8.44
Log G	16.58	13.42	30.23	31.76	33.00	32.99	7.35	0.34	4.07	5.89	6.52	7.27	7.17
Log H	17.73	27.73	34.85	38.58	40.05	40.08	7.15	0.99	3.71	3.64	3.83	3.96	3.98
Log I	11.95	/	13.88	13.88	14.03	14.26	4.05	/	5.26	3.81	3.57	3.85	3.81
Log J	17.46	9.04	31.16	33.25	34.94	36.22	7.76	0.03	2.72	4.37	4.60	4.82	4.78
Log K	12.14	11.20	23.88	24.51	25.74	26.97	3.39	0.67	4.53	6.82	6.24	10.76	10.72
Log L	12.38	11.62	17.75	17.96	18.43	18.48	6.01	1.17	2.55	4.74	4.80	5.47	4.71
Log M	18.42	14.20	37.56	39.14	43.56	43.99	7.10	0.67	4.90	5.22	6.55	7.21	5.75
Log N	14.11	13.64	22.43	22.63	23.71	25.01	5.28	0.77	3.56	5.68	5.66	7.61	7.38
Log O	8.25	5.23	11.35	11.28	12.05	13.67	2.48	0.64	2.52	3.42	3.44	7.15	5.98
Log P	22.73	10.61	34.90	35.98	36.92	37.58	8.22	0.63	5.75	5.52	7.14	9.32	6.64
Log Q	20.55	31.27	76.72	79.05	83.09	84.25	6.78	0.68	2.41	3.67	3.72	3.76	3.77
Log R	22.82	55.63	80.73	100.44	109.21	109.51	7.67	1.07	4.94	8.23	7.85	10.87	9.95

Conclusion

结论：

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。

THANK YOU FOR WATCHING