监控时间序列的告警保持型有损压缩

Bogdan Serykh 高等经济大学计算机科学学院，俄罗斯莫斯科baserykh@edu.hse.ru ORCID0009-0007-0249-9959

2026年11月
预印本，doi:10.5281/zenodo.24504704

arXiv:2611.32894v1 [cs.DB] 24 Nov 2026

摘要

监控系统存储着大量浮点时间序列，其中大部分数据仅由告警规则读取：即对窗口聚合值进行阈值测试，并设置保持时长。若有损压缩器对所有样本使用同一误差界，那么在远离阈值处会浪费存储空间，而在阈值附近则可能改变告警。我们以告警决策作为正确性标准。对于每条规则，在任意评估间隔和相位下，重建后的序列都必须产生与原始序列相同的告警状态序列（非活动、待处理、触发中）。针对 `last`、`avg/sum/min/max_over_time` 和 `rate` 上的 PromQL 阈值规则，我们为每个样本推导出一个可靠的误差预算，可在线性时间内计算。该预算是所有包含此样本的窗口中，窗口到阈值的距离除以利普希茨常数所得值的最小值。由于聚合值是单调的，预算可拆分为向下误差和向上误差的独立界。对于 Prometheus 的外推 `rate`，保序量化器只会约束每个窗口的首尾样本。我们的编解码器结合了预测编码、区间编码器和一种惰性 步长选择策略：只要预算允许，就保留预测值，适用于非对称预算。编码后，编解码器会解码输出、重新评估每条规则，并修复任何发生变化的判定，从而确保浮点输出满足保证。浮点数论证的部分内容已使用 Kani 进行机器检验。对于来自公开 Prometheus 演示服务器的 13个真实序列（7天和30天），该编解码器在每个值仅需 1.0–1.5 比特的情况下保留了所有告警决策（9 055 次触发）。与基于同一数据调优的统一界相比，所需空间减少了 4–6.5 倍；在 26 种规则变体中的 19 种上，其表现也优于基于 SZ3 的压缩保护机制。在消融实验中，单侧预算与惰性步长选择相结合，将数据大小缩减至原来的 1/2.4–1/2.5。

1 引言

Prometheus 等监控系统 [18]会存储大量浮点时间序列（指标）。存储的字节中，很少有会被人读取。大多数数据由告警规则读取，这些规则会将窗口聚合值与阈值进行比较，例如 `avg_over_time(x[5m]) > 0.8for 10m`或 `rate(requests[5m]) < 20`.

Gorilla 系列的无损时间序列压缩器 [13] 在典型仪表盘指标上每个值需要 20–55 比特；这类指标（例如负载或内存使用量）会上下波动（定义 3）。SZ 等基于误差界限的有损压缩器3 [7] 采用统一的逐点界限 ε，并利用它来缩减数据大小。对于告警而言，为所有样本设置同一个界限并不合适。远离阈值时，几乎任何误差都无关紧要；接近阈值时，即便误差小于测量噪声，也可能触发或抑制告警。因此，用户若手动选择 ε，要么浪费存储空间，要么在不知情的情况下改变哪些告警会触发。

我们将告警决策本身作为正确性标准。若对于某个规则集中的每条规则，重构值上产生的警报状态轨迹都与原始值上的相同，则该有损压缩器就是警报保持 的；也就是说，在每次评估时，状态（`Inactive`、`Pending` 或 `Firing`）都相同。这样一来，规则无法观测到的任何信息都可以丢弃。

贡献。

- 针对单序列上的 PromQL 阈值告警建立一个形式化模型，其中包括左开窗口、 `for`/`keep_firing_for` 自动机以及 Prometheus 的外推 `rate`。基于此模型，我们定义了决策保持和相位鲁棒性；后者要求在所有评估间隔和相位下都保持告警决策，且无需预先知道窗口包含多少个样本（第 2 节）。

- 根据规则为每个样本推导出误差预算（定理 1）。一次评估所容许的误差，等于其聚合值距阈值的裕量除以该聚合值的利普希茨常数；样本的预算则是所有窗口包含该样本的评估所对应容差中的最小值。除 `rate` 外，所有聚合值均为单调函数，因此预算可拆分为独立的向下界和向上界（推论 2）。对于 `rate` ，我们证明计数器重置会使其不连续，并且在使用保序量化器时，只需为窗口端点设置预算（命题1）。每条规则的预算计算耗时为 $O(n)$。

- 一种可将非对称预算转换为小型数据流的编解码器（第 4 节）。它在二的幂次网格上使用闭环 DPCM 进行量化，*延迟选择* 步长（预算允许时保留预测值），并使用自适应二进制区间编码器对结果进行编码。验证修复循环会将解码后的输出与参考求值器进行比较，因此该保证在浮点数运算下仍然成立。乐观变体从解析预算的某个倍数开始，仅修复发生变化的评估，且绝不会低于解析预算。

- 针对 `last`、 `min_over_time`和 `max_over_time`浮点预算的位精确机器检查证明（Kani [21]），以及编解码器不变量的证明（第 5 节）。

- 在合成时间序列以及来自公开 Prometheus 演示服务器的 13个真实序列上进行评估，并使用其上游告警规则（第 6 节）。随代码一同发布的还有 7天和30天快照。我们将其与统一误差界以及最接近的现有系统 Compression Safeguards [20], 进行比较，并对编解码器的每个组件进行消融实验。

结果。 在真实数据上，规则感知编码器的输出没有改变任何告警状态；在 7 天快照中涉及 1 864 次触发，在 30 天快照中涉及 7 191 次触发。对于确实会触发告警的阈值，该编码器在真实时间序列上每个值约需 1–1.5 比特。针对相同数据手动调整的统一界需要多出 4–6.5 倍的空间，而且仍无法保证在其他数据上有效。对于 26 种真实规则变体中的 19 种以及所有合成规则变体，该编码器的体积也小于基于 SZ3 的 Compression Safeguards。在消融实验中，单侧预算与惰性步长选择相结合，将体积缩小了 2.4–2.5 倍。

本文中的代码和所有表格均可从 `https://github.com/ProgrammerPeasant/alert-preserving-compression`复现。真实序列快照存档于 doi:10.5281/zenodo.24442344 [16]。

2 模型与正确性标准

我们定义时间序列、压缩器、规则和告警状态，然后给出正确性标准。这些定义不要求读者了解 PromQL；表1列出了每个对象在 Prometheus 中对应的概念。

2.1 时间序列与压缩器

定义1（时间序列）。 长度为 n 的时间序列 是由样本 $X = ((s_i,\ x_i))_{i=1}^{n}$组成的有限序列，其中 $s_i \in \mathbb{Z}$ 是样本 i 的时间戳，单位为毫秒， $s_1 < s_2 < \cdots < s_n$，而 $x_i \in \mathbb{R}$ 是其值。我们用 $I = \{1,\ \ldots,\ n\}$ 表示索引集。

定义2（仅值压缩器）。 压缩器 由两个映射组成：将时间序列转换为比特串的编码器，以及将比特串转换回时间序列的解码器。如果解码后的时间序列具有相同的时间戳 $Y = ((s_i,\ y_i))_{i=1}^{n}$，则称其为仅值压缩器；只有值可能发生变化。我们用 $C(X) = Y$ 表示编码器和解码器的复合映射，用 $e_i = y_i - x_i$ 表示逐点误差。如果对于所有 $i \in I$ 都有 $e_i = 0$，则 C 在 X 上是无损 的。

时间戳通过现有的差分的差分方案 [13] 进行无损压缩，因此不在讨论范围内；本文中的每个压缩器都只处理值。

监控系统根据指标值可能的变化方式来区分指标类型。Prometheus 包含仪表、计数器、直方图和摘要；后两者的经典形式以仪表和计数器时间序列组的形式存储 [18], ，因此两种类型已足够用于我们的分析。

定义 3（仪表与计数器）。如果一个时间序列 X 的值为任意实数，则称其为仪表；仪表用于测量可能上升或下降的当前水平（平均负载、已用内存、温度）。如果一个时间序列的值为非负数，并且自受监控进程启动以来对事件进行累积计数（已处理请求、已发送字节数），则称其为计数器；因此，除非发生重置，否则其值不会下降。重置是满足 $x_i < x_{i-1}$ 的索引 $i > 1$，表示进程重新启动并从零开始计数。

类型是时间序列的元数据（暴露格式中的 `TYPE` 行），而不是从值中推断出来的内容。计数器通过其增长率（下文的聚合速率）使用，仪表则通过其水平使用。

2.2 告警规则

定义 4（阈值规则）。阈值规则 是一个元组

$$R = (F,\ w,\ \bowtie,\ \theta,\ d_{\text{hold}},\ d_{\text{keep}}),$$

其组成部分如下：

1. F，聚合值：一个将有限非空样本序列映射为实数的函数。我们考虑 $F \in$ {last、avg、sum、min、max、rate}。对于值 v_1、.....、v_m，last $= v_m$、avg $= \frac{1}{m}\sum_j v_j$、sum $= \sum_j v_j$ 和 min、max 分别为极值；rate 是计数器每秒的增量，也取决于时间戳（第 3.3 节）；
2. $w > 0$，范围：聚合值读取的时间窗口长度，以毫秒为单位 ；
3. $\bowtie \in \{>、\ \geq、\ <、\ \leq\}$，比较；
4. $\theta \in \mathbb{R}$，阈值；
5. $d_{\text{hold}} \geq 0$，保持时长：条件必须保持为真多长时间后，告警才会触发；
6. $d_{\text{keep}} \geq 0$，持续时长：条件变为假后，处于触发中的告警仍会保持触发状态多长时间。

例如，Prometheus 告警规则

```
expr:  avg_over_time(node_load1[5m]) > 2    for:  10m
```

是元组 $R =$(平均，5 分钟 $>$，2，10 分钟，0)。表1列出了每个组件的对应关系。

T表 1：第 2 节中的符号及其在 Prometheus 告警规则中的对应表示。

符号	Prometheus	含义
X	一个时间序列（一个标签集）	输入
仪表盘指标、计数器	指标类型（`# TYPE`行）	X的类型
$F =$last	即时选择器 `x`	最新样本
$F \in$ {平均值，总和，最小值，最大值}	`avg_over_time`, `sum_over_time`, . . .	窗口聚合
$F =$速率	`rate`	计数器每秒增量
w	`x[w]`中的范围	窗口长度
$\bowtie$, θ	`expr` 中的比较运算和常数	阈值测试
d_{hold}	`for`	保持时长
d_{keep}	`keep_firing_for`	持续时长
T	评估间隔和阶段	评估网格
λ	回看增量（5 分钟）	`last`的陈旧性限制

评估。 规则在评估网格 $T = (t_1 < t_2 < \dots)$的时刻进行评估；在 Prometheus 中，网格由评估间隔和未知相位固定。在某一时刻 t，规则读取索引位于其支撑集中的样本

$$S_R(t) = \{\, i \in I : t - w < s_i \le t \,\}.$$

在 Prometheus 3 中，窗口左侧是开放的，范围选择器也是如此。对于 $F = \text{last}$，该表达式没有范围；支撑集是回看增量内最新的单个样本，$S_R(t) = \{\max\{i : t - \lambda < s_i \le t\}\}$ 其中 $\lambda = 5$ min，若没有则为 $\emptyset$。给定支撑集后，规则会计算两个量：

- 聚合值 $f_R(X)(t) = F\big((s_i,\ x_i)_{i \in S_R(t)}\big)$，其定义条件为 $S_R(t) \neq \emptyset$；
- the *condition* $c_R(X)(t) = \big[\, S_R(t) \neq \emptyset \ \wedge\ f_R(X)(t) \bowtie \theta \,\big] \in \{0, 1\}$.

A空的支撑集会使条件为假，因为 Prometheus 此时不会返回样本。

2.3 告警状态

告警是否触发还取决于保持时长和持续时长，因此规则必须记住过去的条件。Prometheus 将这部分内存保存在一个小型状态机（`rules/alerting.go` [19]）中；我们将其重新表述为一个自动机。

定义 5 （告警自动机）。 自动机 M_R的状态为 $Q = \{$非活动，待处理， 触发中$\}$，并有两个计时器：激活时间 a 和保留起始时间 $b \in \mathbb{Z} \cup \{\perp\}$。它以非活动状态启动，且$b = \perp$，并读取成对的值（$t_j$， c_j），其中 $c_j = c_R$（X）（t_j）。在每一步 $j = 1$， 2，..... 中，它依次执行：

1. 如果 $c_j = 1$：设置 $b \leftarrow \perp$；如果状态为非活动，则转为待处理并设置 $a \leftarrow t_j$；
2. 如果 $c_j = 0$：若状态为触发中且 $d_{\text{keep}} > 0$，则在 $b = \perp$ 时设置 $b \leftarrow t_j$，并在 $t_j - b < d_{\text{keep}}$ 时保持触发中；否则转为非活动并设置 $b \leftarrow \perp$；
3. 如果状态为待处理且 $t_j - a \ge d_{\text{hold}}$，则转为触发中。

步骤 j后的状态为 q_j。 R在 X上、跨越 T的警报状态轨迹 为 $D_R(X) = (q_1,\ q_2,\ \dots\dots) \in Q^{|T|}$。

使用 $d_{\text{hold}} = 0$ 时，步骤 3 会在第一个条件为真时触发，因此永远不会观察到待处理状态；使用$d_{\text{keep}} = 0$ 时，条件为假总会重置告警。

2.4 正确性标准

定义 6 （决策保持）。 设$\mathcal{R}$ 为一个有限规则集。如果对于每个 $R \in \mathcal{R}$，压缩器 C 在网格 T 上对 X 满足 $D_R(C(X)) = D_R(X)$，则称其对$\mathcal{R}$ 具有决策保持性。如果它在每个网格 T上都保持决策，则称其具有相位鲁棒性。

该标准比较完整轨迹。因此，它还要求 Pending 时段相同，这比要求触发相同更严格。放宽条件（触发时间在 δ内、仅要求触发相同）允许采用更大的预算，留待未来工作探讨（第 8 节）。

以下两个引理将决策保持归结为在有限多个时刻保持条件成立。

引理1 (自动机)。 若对所有 $t \in T$， $c_R(X)(t) = c_R(Y)(t)$，则 $D_R(X) = D_R(Y)$。

证明。 M_R 仅读取时刻 t_j 和条件 c_j，而它们对于 X 和 Y是相同的；其状态转移是确定性的。

□

引理*2*（相位鲁棒性）。设 $F \neq$为速率，使得 $f_R(X)(t)$仅通过支撑集 $S_R(t)$依赖于 t 。设 $E_R = \{s_i : i \in I\} \cup \{s_i + w : i \in I\}$（对于 *last*，以 w 替代 λ）。那么$S_R(t)$，因此 $c_R(X)(t)$和 $c_R(Y)(t)$ 在 E_R 的连续点之间的每个区间上均为常数，并且在第一个点之前 $S_R(t) = \emptyset$ 。

证明。 根据定义， $i \in S_R(t) \iff s_i \le t < s_i + w$，因此 i 的成员资格只会在 $t = s_i$ 或 $t = s_i + w$ 处发生变化。对于 last，支撑集是在同类窗口中以 λ 代替 w 时的最大索引，其只会在同类点处发生变化。

□

将 $P_R = E_R$ 的这些点称为 R 的受保护时刻；其数量至多为 $2n$ 。根据引理2，每个时刻 t 的条件都与某个受保护时刻（不晚于 t 的最后一个）的条件相同，对于 X 和 Y 均如此。结合引理1可得:

推论1。 如果对于每个满足 $F \neq$ 速率 的 $R \in \mathcal{R}$ 以及每个 $t \in P_R$，都有 $c_R(C(X))(t) = c_R(X)(t)$，则 C 对这些规则具有相位鲁棒性。

因此，编码器需要规则的范围 w。它不需要评估间隔、相位、抓取间隔或每个窗口的样本数，因此该保证也涵盖漏抓取和不规则时间戳。速率聚合通过对窗口边界进行外推，连续地依赖于 t （第3.3节）；对于它，条件仅在部署网格 T 上得到保留， $P_R = T$。

3 误差预算

3.1 一个充分条件

考虑一个支撑集非空的受保护评估（R, t）。其裕量 $m_R(t) = |f_R(X)(t) - \theta|$ 是聚合值与阈值之间的距离。令 $L_R(t)$为F 在上确界范数下、定义于 $\mathbb{R}^{|S_{R(t)}|}$ 上的利普希茨常数。对于 `last`、`avg_over_time`、 `min_over_time` 和 `max_over_time` ，我们有 $L = 1$；对于 `sum_over_time` $L = |S_R(t)|$，亦有相应结论。该评估的容差 为$\tau_R(t) = m_R(t)/L_R(t)$。每个支撑集中包含某个样本的评估都会读取该样本，而该样本的预算是这些评估中最小的容差:

$$\varepsilon_i = \min\{\tau_R(t) : R \in \mathcal{R},\ t \in P_R,\ i \in S_R(t)\}, \qquad \min \emptyset = +\infty. \tag{1}$$

对于原始数据，该条件为 $f_R(X)(t) \bowtie \theta$ ，或其否定，因此它是 $f > \theta$、$f \ge \theta$、 $f < \theta$ 和 $f \le \theta$ 之一。下文将使用这个不等式的两个性质。

定义 7（紧、上界和下界评估）。若在 X 上成立的不等式是严格的，即 $c_R(X)(t) = 1$，且 $\bowtie \in \{>, <\}$，或 $c_R(X)(t) = 0$，且$\bowtie \in \{\geq, \leq\}$，则受保护评估 (R, t) 是紧的。此时，若重构值满足 $f_R(Y)(t) = \theta$，条件便会翻转，因此误差必须严格小于容差。若 $f_R(X)(t)$ 位于 θ 一侧，且只有聚合值减小才能使其离开该侧（$c_R(X)(t) = 1$，且 $\bowtie \in \{>, \geq\}$，或 $c_R(X)(t) = 0$，且 $\bowtie \in \{<, \leq\}$），则它是上界评估；否则为下界评估。

“盒预算 ”是重构值的集合$\{Y\colon |y_i - x_i| \leq \varepsilon_i$ 对所有 $i \in I\}$均成立。它将每个样本在两个方向上的误差都限制在相同范围内。

定理 1（盒预算）。如果 $|y_i - x_i| \leq \varepsilon_i$ 对所有 i均成立，并且在紧致评估的支撑集上严格成立，那么对于所有 $R \in \mathbf{R}$ 和 $t \in P_R$，$c_R(Y)(t) = c_R(X)(t)$ 均成立，因此在每个网格上（*on T for* `rate`）都有 $D_R(Y) = D_R(X)$。

证明。支撑集仅取决于时间戳。根据利普希茨界 $|f_R(Y)(t) - f_R(X)(t)| \leq L_R(t) \max_{i \in S_R(t)} |e_i| \leq m_R(t)$，在紧致时严格成立，因此 $f_R(Y)(t)$ 始终位于 θ 的同一侧。引理 1 和 2 完成证明。

□

Lipschitz步骤是标准的关注量误差传播 [5]。我们的情形在误差和裕量所处的位置上有所不同：误差属于样本，裕量属于评估，而每个样本都会被其窗口包含的所有评估读取。因此，预算是对重叠窗口取最小值，这会带来两个后果。第一，编码器必须向前查看最长窗口 $\max_R w_R$，才能确定样本的预算，因此它更适合块压缩（Prometheus 会压缩 2 小时的数据块）或缓冲代理，而非写入路径。第二，裕量为零的样本必须精确存储。较小的预算也足够，因此逐点上限 $\min(\varepsilon_i, \varepsilon_{\max})$ 可以在不削弱告警保证的情况下，限制仪表板的误差。

3.2 单侧预算

在阈值附近，时间序列通常会保持在阈值的一侧，只有朝向 θ 的误差才可能改变判定。定理 1 的盒预算也限制了远离 θ方向的误差，但这是不必要的。

推论 2（单侧预算）。令 δ_i^-（向下 预算）为支撑集包含 i 的上界评估中 $\tau_R(t)$ 的最小值，并令 δ_i^+（向上预算）为下界评估中的相同最小值，其中 $\min \emptyset = +\infty$（定义 7）。如果对于所有 i，$-\delta_i^- \leq y_i - x_i \leq \delta_i^+$（在取紧时严格成立），则每个受保护条件都得以保留。

证明。上述所有聚合值均关于每个参数单调不减。对于一个上界 (R, t)，在其支撑集上令 $z_i = \min(y_i, x_i)$。于是 $|z_i - x_i| \leq \delta_i^- \leq \tau_R(t)$，因此 z 在 (R, t) 处满足定理 1，而 $F(y) \geq F(z)$ 会使 $f_R(Y)(t)$ 保持在上侧。下界评估与此对称，只需使用 max。

□

由于 $\varepsilon_i = \min(\delta_i^-, \delta_i^+)$，盒预算中的每个重构值也都满足单侧界。单侧集合要大得多：仅由上界评估读取的样本可以向上移动任意幅度（图 1）。只有在不对称地放置重构值时，编解码器才能利用这一自由度（第 4 节）。采用舍入到最近值量化时，仅靠单侧预算几乎没有什么收益（第 6.5 节）。

3.3 计数器和 `rate`

我们针对一次评估 (R, t) 并结合 $|S_R(t)$，重新表述 Prometheus 的 `rate`（`extrapolatedRate` in `promql/functions.go` [19]）。令 f 和 l 分别为支撑集 $S_R(t)$ 的第一个和最后一个索引。该规则如下：

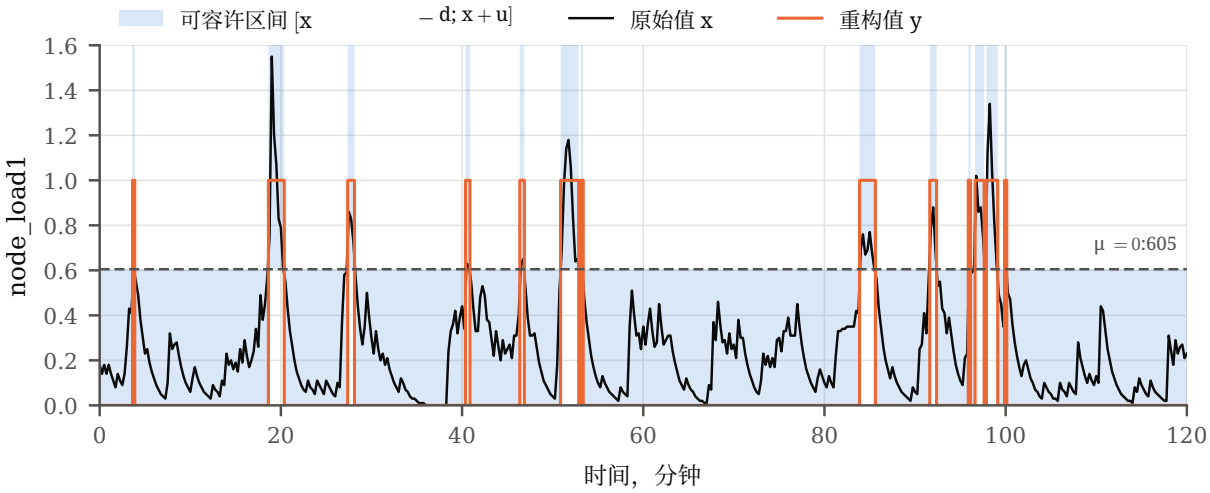


图 1： `node_load1` 两小时数据的单侧预算及编解码器重构值（7 天快照，规则 `last` > θ，在第95百分位数处 θ）。在 θ 以下，只限制向上误差；在 θ 以上，只限制向下误差。因此，可容许区间（阴影部分）是一条半直线，且重构值只有两个等级。告警决策均未改变。

- 该增量 $\Delta = x_l - x_f + \rho$，其中 $\rho = \sum_{i \in S_R\ (t)\quad i>f,\ \ x_i<x_{i-1}} x_{i-1}$ 会补回窗口内每次重置时损失的值；
- 样本覆盖的采样区间 $\sigma = s_l - s_f$；
- 从第一个和最后一个样本到窗口起点$t - w$ 和终点 t的外推距离 h_{s} 和 h_{e}。当其达到采样间距的 1.1 倍时，便将其替换为平均采样间距的一半。两者都只取决于时间戳；
- 零点距离 $h_0 = \sigma x_f / \Delta$，仅在 $\Delta > 0$时使用。如果计数器以窗口内的平均速度增长，那么这是计数器归零之前的时长。

增量会外推到窗口边界，但不会越过计数器的零点，然后除以以秒为单位的范围：

$$\mathrm{rate} = \frac{\Delta}{w} \cdot \frac{\sigma + \min(h_{\mathrm{s}}, h_0) + h_{\mathrm{e}}}{\sigma}.$$

命题1。

(a) 重置指示器 $[x_i < x_{i-1}]$ 比较两个样本。因此，任意小的误差都可能导致重置，并使 Δ 改变一个完整的计数器值；一般而言，不存在能保留 *rate* 的正对称预算。*(b)* 闭环向下取整量化器 $y_i = y_{i-1} + q\lfloor (x_i - y_{i-1})/q \rfloor$ 在步数会变为负数时精确保留样本，从而得到 $x_i - q < y_i \le x_i$ 并保留$x_i \ge x_{i-1} \Rightarrow y_i \ge y_{i-1}$。此外，如果每次真实重置前后的两个样本都精确保留，则重置集合也会得到保留。*(c)* 对于保留重置的重构值，速率 在远离 $x_f = 0$ 的 (Δ, x_f) 范围内满足利普希茨条件。对于包含 r 次重置的密集采样窗口，该常数约为 $(2 + r)/w$。*(d)* 在 *(b)* 的条件下， Δ 和x_f 仅取决于窗口的首末样本以及保留的重置值。因此， *rate* 评估的预算仅适用于其窗口的两个端点。

实现并未使用 (c) 中的常数。它会在盒子 $y_i \in [x_i - \varepsilon, x_i]$ 上计算 `rate` 公式（包括零点分支）的可靠区间包络，并通过二分法找出最大的安全 ε。采用 30 s 的评估网格和 15 s 的抓取间隔时，计数器约有一半的样本不是任何窗口的末端。根据 (d)，这些样本仅受单调性约束。

3.4 不存在预算的情况

在 x 处存在正预算，当且仅当 x 位于使相同条件成立的输入集合的内部。某些表达式会在其决策点处破坏这一点：相等性测试、 `changes()`、表达式中的 `floor` 和 `round` ，以及按标签选择时间序列的聚合，例如 `topk`。它们在这些点需要精确的样本。比率和 `histogram_quantile` 具有局部 Lipschitz 连续性，可以使用区间包络处理，就像 `rate`一样。

3.5 浮点数与复杂度

预算以浮点数计算，代码会从裕量中减去舍入余量。由 n 项组成的浮点数求和的误差至多为 $\gamma_{n-1} \sum_i |v_i|$，其中 $\gamma_n = nu/(1-nu)$，单元舍入误差为 $u = 2^{-53}$ [4]。对于严格比较，代码会取容差以下的下一个可表示值。对于 `last`、 `min` 和 `max`，逐位单调性成立。Prometheus 使用 Kahan 补偿计算总和，而这并不具有单调性，因此对于 `sum/avg` ，推论 2 的证明改用误差界。裕量处于舍入余量范围内的评估会受到双向约束。

支撑集会随 t单调移动，因此双指针可以找到所有支撑集。双栈滑动队列计算窗口聚合值，单调双端队列则将（1）中的逐样本最小值计算为关于时刻的滑动最小值。每条规则的预算成本为 $O(n + |P_R|)$。对于 `rate`，每个时刻都会增加一次二分法，大约需要进行十几次包围评估。

4 编解码器

编解码器接收逐样本界限（δ_i^-，δ_i^+），并生成一个比特流，其解码值位于可容许区间内

$$A_i = [\, x_i - \delta_i^-,\; x_i + \delta_i^+ \,].$$

预算为零（或非数）会强制精确存储该样本，而预算为 $+\infty$ 则不对其加以约束。流格式不依赖于规则，解码器只读取数据流。

量化器。 有损模式采用闭环 DPCM（差分脉冲编码调制），网格步长为 $q = 2^e$。样本 i 的预测值是前一个重构值 $p_i = y_{i-1}$。由于预测值不使用原始值，编码器和解码器会计算出相同的预测值，误差也不会累积。重构值为 $y_i = p_i + k_i q$，其中整数步数 k_i 从可容许计数值中选取

$$K_i = \{\, k \in \mathbb{Z} : p_i + kq \in A_i \,\}.$$

当 $q \le \delta_i^- + \delta_i^+$ 时，该集合非空。数据流存储 k_i 或差值$k_i - k_{i-1}$。

块。 时间序列被划分为每块 128 个样本的数据块，每个数据块要么使用 Gorilla 异或编码 [13] 进行无损编码，要么使用上述量化器进行编码，整个数据块使用一个指数 e 。编码器会选择成本较低的方式。数据块中的每个样本都会提出一个候选指数：满足 $2^e \le \delta_i^- + \delta_i^+$的最大 e，使得 $K_i \neq \emptyset$ （对于计数器，$2^e \le \delta_i^-$）。编码器还会尝试接下来的三个更细网格，因为对于离散数据，更小的步长可以产生更小的

数据流。对于计数器，量化器是命题 1(b)中的向下取整量化器，它能使重构值低于数据且保持非递减。

如果一个样本的步数在幅值上会超过 2^{50}，或者其浮点数重构值会偏离 A_i，则该样本会被转义，也就是说，它会以原始 64 位值的形式存储。这也涵盖了区间对于块网格而言过窄的样本。编码器会将每个重构值与其界限进行比较；这些界限由编码器逐位按照解码器的计算方式计算得出，因此这些界限对于实际的浮点输出成立。

惰性步长。最近选择 $k_i = \mathrm{round}((x_i - p_i)/q)$ 会使误差在两个方向上均匀分布。这适用于对称界，但会浪费非对称界的大部分范围。因此，对于每个网格，编码器还会尝试惰性选择，即最接近零的 $k \in K_i$（因此当预测值可接受时为 $y_i = p_i$），或最接近 k_{i-1}的值。对于单侧界，预测值通常会在很长的样本序列中保持可接受，熵编码器可以将这类序列编码为每个样本几百分之一比特。对于每个数据块，编码器会根据精确的编码大小比较四种策略（最近或惰性，结合编码计数或差分）。

熵编码。 每个块决策和每个整数都使用 LZMA 风格的自适应二进制区间编码器 [9] 以 15 位概率进行编码。整数（步数或差值）以零标志、符号及其一元编码的位长度发送。这些符号在由前一个整数的幅值所给出的上下文中建模。随后是紧邻最高位 1 下方的那一位，它同样被建模，以及其余按原始方式写入的位。模型在数据流开始时快速适应，并随着计数增长而放缓，因此短数据流也能很好地压缩。对于消融实验，我们保留一种定宽变体，它以数据块的最大位长度存储该数据块的整数。

数据块。 Prometheus 在 120 个样本处切分头部数据块，且每个数据块必须能够独立解码。编解码器可以将时间序列编码为独立流，每条数据流的开销为 3–4 字节。预算仍基于整个时间序列计算，因为窗口会跨越数据块边界。计数器数据块从上一数据块的 last 重构值开始，并以其作为向下取整。否则输出可能在数据块边界处下降，并且 `rate`会看到数据中并不存在的重置。

验证与修复。 编码后，编解码器会解码数据流，并使用参考语义重新评估每个受保护条件。对于条件发生变化的每次评估，它会将该评估支撑集的预算设为零，然后再次对时间序列进行编码。该循环必定终止，因为逐位精确存储的支撑集会给出原始值。对于 `sum/avg`，三个组件以不同顺序求和：验证器使用滑动求和，参考求值器从左到右求和，而 Prometheus 使用 Kahan 求和。它们的结果相差至多 $2\gamma_{n-1}\sum|y_i|$（第 3 节）。因此，验证器仅在某次评估的聚合值与 θ 的距离超过该差值，或其支撑集逐位精确时，才接受该评估。解析预算保留了该裕量的两倍，并且在任何数据集上都不需要修复。该循环使这一保证相对于参考求值器端到端地成立。

乐观预算。 推论 2 的界限假设所有误差会同时取其最坏值，而在窗口聚合中，实际误差大多会相互抵消。因此，编码器可以从 $\kappa\delta_i^{\pm}$和 $\kappa > 1$开始。在每次发生变化的评估中，它都会将支撑集的预算减半，直到达到解析预算；一旦达到解析预算，就将其设为零。根据定理 1，处于解析预算以内的支撑集不会改变其评估值。因此，解析预算既是 $\kappa = 1$ 的可靠起始点，也是限制轮次数量的下限。压缩保护机制 [20] 以类似方式纠正违规，但我们的编码器只纠正评估值发生变化的支撑集。

5 机器检查证明

定理 1针对的是实数，而代码使用 IEEE-754 双精度计算裕量、容差和聚合值。我们使用 Kani [21]，（一个基于 CBMC [6] 、面向 Rust 的有界模型检查器）检查浮点实现；该工具能够以位精确的方式对浮点运算进行编码。每个测试框架都会对固定形状的所有有限输入进行量化，并在有明确说明时涵盖非数和无穷大。下面的时间是 Kani 0.68（使用 CaDiCaL 的 CBMC）在一台运行 12 个测试框架的 24 核 x86-64 机器上的实际耗时。

我们首先尝试建立一个测试框架，声明预算能够保证在一个包含三个样本的窗口内满足相同条件。对于任何聚合值，该证明都未能在 6 小时内完成。符号执行速度很快，但针对浮点运算链（聚合值、减法、`next_down`、比较）的最终 SAT 查询过于困难。因此，我们将定理 1 针对 `last`、 `min_over_time`和 `max_over_time` 分为三部分证明。抽象容差 τ 将开销较大的容差计算与窗口结构分离开来：

(A) 标量容差，每个比较对应一个测试框架：若 fl$|y' - v| \le \tau(v)$，其中 τ是已实现的容差函数，则 y' 和 v 位于 θ 的同一侧（每次需 56–69 分钟）。(B)窗口引理，涉及至多四个变量。极值由某个样本取得。对于极值位置 j 和 k，fl$|y_j - x_k| \le$ max(fl$|y_j - x_j|$、fl$|y_k - x_k|$)（分别需 198 和 226 分钟）。舍入减法是单调的（需 130 分钟）。(C) 窗口的已实现容差在逐位意义上等于 $\tau(f(x))$（需≤ 3分钟），且聚合值是单调的（推论 2； ≤ 8 秒）。

综合起来，(A)–(C) 给出了针对这些聚合值及任意窗口长度的定理 1 的浮点版本。

对于编解码器，我们证明，对于每个块参数（包括 -0.0），解码器都能逐位精确重建编码器计算出的值。我们还证明了量化器的仪表盘指标和计数器不变量。量化器会尝试多个候选步数（四舍五入后的步数、惰性选择以及向内缩进一步），并且只返回通过单一接受性检查的候选值。对完整候选搜索进行测试的测试框架在 6 小时内未能完成，因此我们证明了接受性检查应用于任意候选值时的不变量（在一台 16 核机器上分别运行，耗时 7 分钟和 15 分钟）。最后，我们证明了之字形编码、变长整数编码以及位输入输出的位级规范均可往返。

尚未证明。`sum/avg_over_time` 需要一个引理来证明舍入和的误差，以替代 (B)。区间编码器和具体的位写入器在符号执行时会耗尽内存或超时，这是因为 `Vec` 无界增长，以及循环边界取决于符号计数器。属性测试覆盖了这些部分。往返测试使用非数、无穷大和零预算以及计数器，对两种格式都进行测试；并将 5×2000 随机决策保持测试用例与会重新计算每个时刻的参考求值器进行核对。测试捕获到一种将预算乘以 3 的变异。

6 评估

6.1 设置

真实数据。我们使用来自公共 Prometheus 演示服务器 [14] 的 13 条原始时间序列，这些序列以 15 秒为间隔抓取，且没有缺口。它们包括 node_exporter 仪表指标（负载、可用内存、根文件系统的可用空间、时钟偏移量）、Prometheus 自身的计数器和仪表指标、一个 Caddy 请求计数器，以及一个演示服务的计数器。我们提供了两个快照，分别涵盖 7天（每条时间序列包含 40 320 个样本）和 30天（172 800 个样本）。服务器只保留 31 天的数据，因此我们连同代码一起发布这些快照。

表 2：13个真实序列中每个值的平均比特数。每个规则感知单元格中的变更的决策均为零。“/120”：Prometheus 大小的独立数据块。

	规则	无损	均匀安全	均匀调优	规则感知	κ	规则感知/120	κ/120
7 d	prod	27.71	0.08	0.09	0.07	0.07	0.45	0.42
	q95	25.52	6.91	7.29	1.13	1.00	1.77	1.59
	q50	25.52	7.69	6.01	1.49	1.38	2.16	2.00
30 d	prod	27.77	0.08	0.08	0.06	0.06	0.44	0.41
	q95	25.59	7.99	5.72	0.96	0.86	1.58	1.43
	q50	25.59	7.47	5.91	1.15	1.08	1.80	1.68

演示服务器运行上游 node-mixin 和 prometheus-mixin 告警 [12]。其中九条规则在将常数分母折叠到 θ后，只需读取一条时间序列一次。没有上游规则的计数器使用规则 `rate(x[5m]) > θ for 5m` 和 `rate(x[5m]) < θ for 10m`。在健康主机上，已部署阈值（`prod`）从不触发。因此，我们还为每条时间序列分别使用两个重新校准的阈值，这些阈值根据规则自身的聚合值计算得出：第95百分位数（`q95`，告警会频繁触发）和中位数（`q50`，聚合值会 θ 一直越过阈值）。每个此类阈值都设在两个不同取值的中间，因此离散指标不会恰好落在 θ 上。这样，每个快照包含 26 个规则变体，7天内触发 1 864 次，30天内触发 7 191 次。

合成数据。 五个固定种子的生成器以 15 秒为间隔生成 7天的数据：CPU 利用率、请求延迟、内存、请求计数器，以及一个带有 8% 漏抓取、1–6 分钟中断且流量始终接近阈值的相同计数器。每个数据集包含两条规则，每天触发数次。

评估。 规则每 30 秒评估一次，时间在抓取样本之后 7 秒。我们比较完整的告警状态轨迹，这些轨迹由代码仓库中的参考求值器计算，该求值器实现了 Prometheus 3 语义。大小以数值流的每个值的比特数表示，不包含时间戳；未压缩的值占用 64 比特。

基线。 我们自己的配置均使用相同的编解码器和区间编码器，唯一差异在于误差预算。无损 表示预算为零的编解码器，其输出为 Gorilla 异或块。均匀安全 是满足定理 1 的最大统一界。均匀调优 通过在数值范围的四次幂上扫描统一界得到：它是这样一个最大界，使得该界以及扫描中任何更小的界都不会改变这些数据上的判定。手动调优界的用户也不可能做得更好，而该界对其他数据不提供任何保证。*SZ3+SG*是采用压缩保护机制封装的 SZ3 [7] [20] (1.0.0rc3)。对于规则能够在任意相位读取的每个窗口长度，它都有一个模板关注量，其中比较本身作为关注量，界为零。我们扫描 13 个 SZ3 界，并报告其中不改变任何判定的配置中的最小输出。规则感知 表示解析预算（推论 2），而 κ 表示带有 $\kappa = 1024$的乐观预算。

6.2 真实时间序列

表 2 总结了真实数据。在已部署阈值下，没有任何告警触发，而且每种编解码器（包括统一编解码器）所需的每个值的比特数都小于 0.1，这是因为数据从未接近过的阈值无法施加任何约束。这是生产环境中的常见情况，但无法区分不同方法。在重新校准的阈值下，在 7天数据上，规则感知编码器的大小分别是调优的统一界（q95）和调优的统一界（q50）的 1/6.5 和 1/4.0；在 30

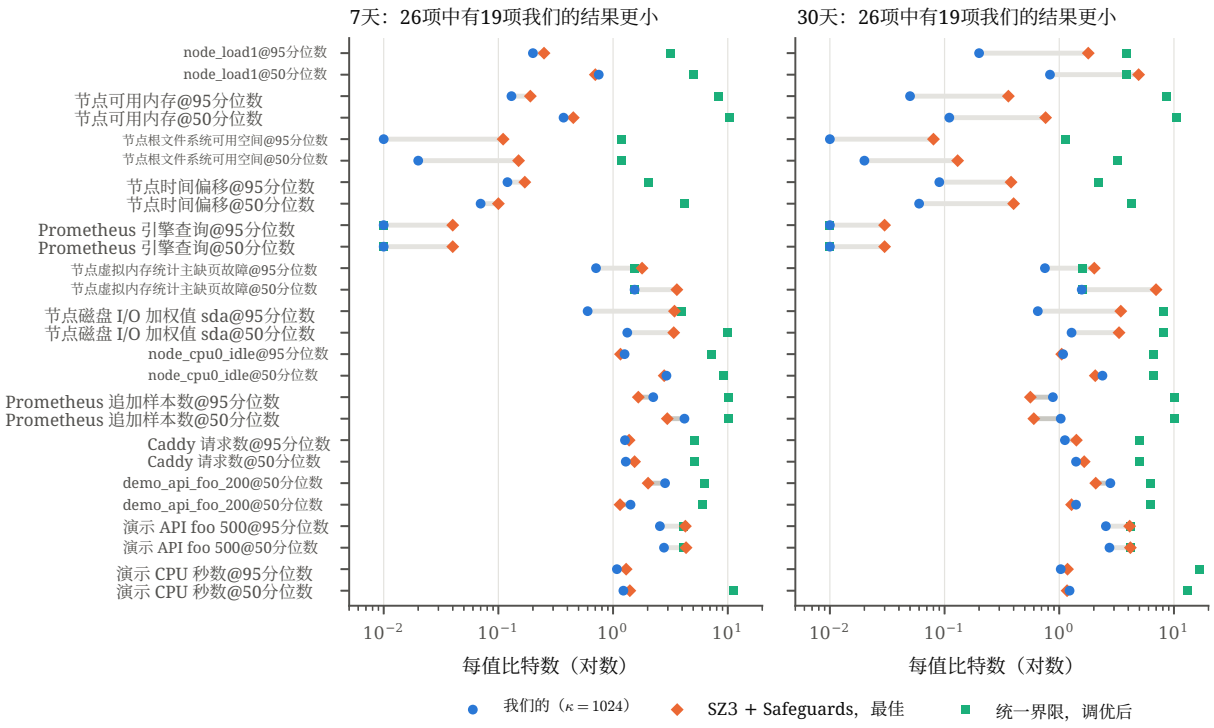


图2：真实时间序列上的每个值的比特数：我们的乐观编解码器（$\kappa = 1024$）、在其界限扫描中判定始终不变的最小 SZ3+SG 输出，以及在这些数据上保持每个判定不变的最大统一界限对应的我们的编解码器（已调优，不提供保证）。灰色条将我们的编解码器与 SZ3+SG 连接起来。

天数据上则分别为 1/6.0 和 1/5.1。与调优的统一界不同，它还能够保证判定结果。对于每条时间序列，其大小范围从几乎不发生变化的空闲文件系统空间的每个值 0.01 比特，到 `rate` 采用中位数阈值时密集计数器的 3–4.5 比特。在每个包含 120 个样本的 Prometheus 大小数据块中，由于每个数据块的固定开销，以及模型在每个数据块开始时尚未经过训练，大小会增加 0.6–0.7 个每个值的比特数。

6.3 与压缩保护机制的比较

在两个快照中，乐观编解码器在 26 个变体中的 19 个上都比 SZ3+SG 更小（图2）。大小比率 κ/SG 的几何平均数在 7天时为 0.60，在 30天时为 0.42。在密集计数器上，SG 仍然更小（`rate`、`prometheus_samples_appended`、`node_cpu0_idle` 和`demo_api_foo_200`）。对于这些时间序列，我们的量化器对整个 128 个样本的数据块使用一个步长，而 SG 则修正单个样本。

SG 对 `rate` 的转换无法表达 Prometheus 的外推。在两个快照中，8 个真实计数器里有 4 个计数器在某些扫描点上出现了判定变化；对于包含漏抓取的合成计数器，13 个扫描点中有 11 个出现了判定变化。对于该计数器，若要确保每个判定都保持不变，最小的 SG 输出需要 8.45 每个值的比特数，是我们的解析编解码器（2.91）的 2.9 倍，也是我们的乐观编解码器（2.74）的 3.1 倍。用户无法预先判断哪个 SG 界限是安全的。在一个 30 天的计数器上，SG 每个扫描点最多需要 60 秒；而我们的完整经验证的编解码器运行速度为 11–16 MB/s（第 6.6 节）。

6.4 合成序列

在合成仪表（表 3）上，规则感知编码器的大小比调优的统一界小 33–36 倍，在计数器上则小 2.2 倍和 3.2 倍。解析编解码器在全部五个数据集上都比 SZ3+SG 更小。按照定理 1 的盒预算，它在三个仪表上比 SZ3+SG 更大，这一结果促成了我们提出推论 2。在

表 3：合成序列，7天，每个值的比特数（每个序列对应一个数据流）。SZ3+SG：在整个扫描范围内实现决策零变化的最小输出。

数据集	无损	均匀安全	均匀调优	SZ3+SG	规则感知	κ
cpu	53.96	11.57	2.55	0.16	0.07	0.05
延迟	55.24	7.22	2.81	0.20	0.08	0.07
内存	49.91	12.96	6.92	0.57	0.21	0.14
请求	19.68	2.02	0.78	0.69	0.35	0.35
请求（有缺口）	21.76	11.46	9.46	8.45	2.91	2.74

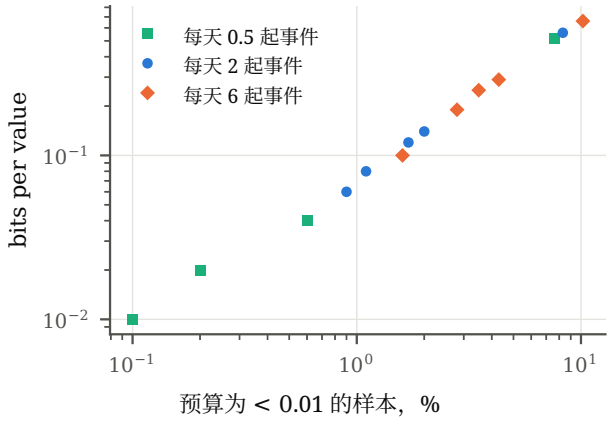


图3：规则感知编码器在合成数据 cpu （7天，1条规则）上的大小与预算低于0.01的样本占比之间的关系，其中 $\theta \in \{0.55、\ldots\ldots、0.95\}$，以及三种事件率。

样本接近阈值的占比越高，大小就越大：在 cpu 上，当该占比从 0.1 % 增至 10 % 时，大小从 0.01 增至 0.66 每个值的比特数（图3）。

保真度。 没有上限时，重构值仅对规则有用，因为样本在某一方向上不受约束时，会不断漂移，直到触及界限为止。加上逐点上限 $\varepsilon_{\max}$后，规则感知编码器的大小与统一界限 $\varepsilon_{\max}$ 相比相差不超过2%，并且保留了每个判定。将范围的 10^{-3} 设为统一界限，会在 `memory`、 `requests` 和`requests-gappy`上分别改变17、929和10 655个告警状态。即使是调优的统一界限，在应用于120个样本的数据块，而不是其调优所基于的全序列编码时，也会在两个数据集上改变判定。针对数据的一种编码调优出的界限，无法沿用于同一数据的另一种编码。

6.5 消融实验

表 4 逐次添加一个组件。在相同的流格式下，单侧预算和惰性步长选择结合使用时，可将大小缩减至原来的 2.4–2.5 分之一。单独使用其中任一项，收益都小得多。采用舍入到最近值时，网格根据可容许区间的宽度选择，舍入也以该宽度为对称范围，因此仅使用单侧界的收益很小，有时甚至会使输出变大。对于计数器，所有收益都来自惰性步长；惰性步长使预测值落在不属于任何窗口末端的样本上（命题1(d)）。我们还曾对预算的缩放副本（$\times\frac{1}{2}$、14、18）进行编码，因为编码后的大小并不随预算单调变化。一旦编码器针对每个数据块尝试接下来的三个更细网格，这种搜索便是多余的：在 31 个数据集中，没有一个因该搜索得到更小的输出，因此编解码器不再执行此操作。乐观预算可进一步节省 4–20%，但在真实数据上，它并不能取代单侧预算。熵编码避免了定宽编码每个值至少占用一位的问题，并使输出

表 4：消融实验：相对于完整解析编解码器（第 3 行）的大小比率几何平均数，统计对象为合成时间序列和 q95/q50 真实变体（7 天快照；代码仓库中的 30 天快照显示出相同的排序）。每个单元格都保留了所有判定。

配置	合成数据	真实 q95	真实 q50
盒预算、最近步长（定理 1）	2.44	2.53	2.47
+ 单侧预算（推论 2）	2.27	2.06	1.91
+惰性步长（完整解析编解码器）	1.00	1.00	1.00
+ 乐观预算，$\kappa = 1024$	0.80	0.92	0.95
完整解析 + 预算尺度搜索	1.00	1.00	1.00
完整解析 –单侧预算	1.37	1.21	1.20
完整解析 –惰性步长	2.27	2.06	1.91
$\kappa = 1024$ – 单侧预算	0.93	1.05	1.08
完整解析的定宽编码	8.01	7.37	5.92

在两个快照上都缩小了 5.9–8.6 倍。

6.6 吞吐量

我们在单核上测量吞吐量，单位为原始浮点数64输入的 MB/s，测试数据为合成数据 `cpu` 和 `requests`，每组包含 40 320 个样本。$O(n)$ 预算计算的速率为 42–67 MB/s。编码器的速率为 22 MB/s，因为每个数据块都会尝试 4 个网格，每个网格采用 4 种步长策略；解码器的速率为 2.5–2.9 GB/s。整个经验证的压缩器的速率为 11–16 MB/s；使用预算尺度搜索时则为 3–4 MB/s，而预算尺度搜索已无法再改变数据大小（第 6.5 节）。一个 2 小时的 Prometheus 数据块在 15 秒采样间隔下，每个时间序列包含 480 个样本，因此单核每秒可压缩约 2 900 个时间序列块。

7 相关工作

基于校正的保障机制。压缩保护机制 [20] 可为任意压缩器添加声明式要求，例如逐点界限、相对于某个偏移量的符号保持，或对模板上关注量的界限。压缩后，它们会计算校正值以满足这些要求。在规则网格上的窗口平均阈值测试可以写成模板关注量：该关注量就是比较本身，其界限为零。我们以这种方式运行 Safeguards，作为基线（第 6.3 节）。

这两种方法的区别在于：需要将哪些内容翻译成需求语言，以及如何强制执行该需求。我们的判据无需翻译持续时长（引理1），但为了实现相位鲁棒性，模板则需要进行翻译。模板以样本数固定窗口长度，因此，漏抓取和未知的评估相位要求每一种实际出现的窗口长度都使用一个模板，此外还需要处理边缘处的截断窗口。Prometheus 的 `rate` 外推和计数器重置（命题1）完全没有对应的模板形式。我们为基线采用的翻译，即对采样区间上的增量进行重置校正，会改变存在间隙的计数器的判定。Safeguards 会校正每一个违反需求的点。我们的解析预算从一开始就是可靠的，无需校正；而我们的乐观变体只校正评估发生变化的支撑集，绝不会低于解析预算。

关注量保持。 Jiao等人 [5] 通过误差传播，根据导出量的界推导逐点误差界，而我们的 Lipschitz步骤正是其方法的一个实例。我们界定的是判定，而不是某个量的误差，因此我们的预算取决于与阈值的距离。我们还处理了时间结构：重叠窗口、评估相位和告警自动机。

特征与拓扑保持压缩。 TopoSZ及其后续方法 [22]保持科学场的临界点和等值线树，自适应方案会收紧检测到的特征附近的界 [2]。在 θ 处保持水平集意味着保持 $x - \theta$的符号，这就是我们将设置限制为 `last`时的情形。窗口聚合和告警状态在该工作中没有对应物。

时间序列压缩。 Gorilla [13] 和 ALP [1] 是无损的。SZ3 [7] 以及近期的有误差界时间序列压缩器 [17]保证统一的逐点界限。CAMEO [10] 界定自相关函数的偏差，Muñiz-Cuza 等人 [11] 则衡量有误差界的压缩如何影响预测。Plato [8] 为压缩时间序列上的聚合查询提供确定性误差界，从而界定查询的数值结果。这些系统都无法界定结果落在阈值的哪一侧，因此都无法保持离散决策。

下游保证与工业实践。 在时间序列之外，我们所知最接近的保证是粒子数据的簇保持 [15]。朋友关系簇成员资格取决于对距离进行的阈值测试，而在压缩后、在逐点界限内应用校正即可恢复该成员资格，这与我们的修复循环恢复发生变化的评估大致相同。工业成本控制通过选择要保留的时间序列，而不是界定其值来实现控制。Grafana 的 Adaptive Metrics 建议聚合掉任何仪表板、规则或近期查询未使用的标签，但没有针对记录规则或告警规则读取的指标提出建议 [3]。我们的方法面向这些指标。我们检索了文献（Crossref、arXiv、EDBT/VLDB 会议论文集、2015–2026 年）和供应商文档，未发现任何能够保证告警决策的压缩器。

8 局限性与未来工作

规则范围。 我们处理读取单个时间序列的规则。跨序列聚合（`sum by`）会耦合组内所有时间序列的预算，因此编码器必须同时查看整个组。比率需要区间包络，而相等性测试和 `topk` 在其决策点不允许分配正预算。

已知规则。 预算是针对固定规则集计算的。之后添加的规则读取的是为其他规则优化的重构值，只有可选的逐点上限能够约束其误差。只要保留原始值，就可以再次根据原始值压缩时间序列，例如一直保留到压实为止。

前瞻。 样本的预算取决于其后一个窗口内的评估，因此编码器必须缓冲最长的窗口。这在压实期间或缓冲代理中可行，但不适用于不得增加任何延迟的写入路径。

数据。 我们的真实数据来自两个公共演示服务器上的 13 个时间序列。已部署阈值在那里从未触发，因此我们将阈值重新校准为分位数。仍然需要使用包含更多规则、更多事件以及数据间隙的生产跟踪数据进行评估。

放宽条件。 要求 Pending 时段保持不变，会使该标准过于严格。如果只要求保留触发，或将触发时间的偏差限制在 δ以内，那么对于短暂的条件翻转，预算可能会超过其裕量，因为 `for` 过滤器会将其滤除。此时，分析就必须针对自动机进行推理，而不能只考虑单次评估。(1) 中的最小值给出了可行预算，但并非最优预算。对于线性聚合值，如何在窗口内的各个样本之间最优地分配裕量，是一个类似背包的问题。

9 结论

以告警决策作为正确性标准时，监控数据的有损压缩具有可靠的逐样本误差预算，并且可以在线性时间内计算。聚合值是单调的，因此该预算是单侧的。只要预测值仍符合要求，编码器就会保留该预测值；这样得到的输出比使用对称利普希茨框时小 2.4–2.5 倍。在真实的 Prometheus 序列上，该编解码器每个值只需几十分之一比特到几比特，而且不会改变任何告警决策。

参考文献

[1] Azim Afroozeh、Leonardo X. Kuffó 和 Peter Boncz。ALP：自适应无损浮点压缩。*Proc. ACM Manag. Data*，1(4):230:1–230:26，2023 年。 doi:10.1145/3626717.[2] Qian Gong、Chengzhu Zhang、Xin Liang、Viktor Reshniak、Jieyang Chen、Anand Rangarajan、Sanjay Ranka、Nicolas Vidal、Lipeng Wan、Paul Ullrich、Norbert Podhorszki、Robert Jacob 和 Scott Klasky。用于科学数据集的时空自适应压缩与特征保持——极端气候事件分析模拟数据案例研究。载于*Proc. IEEE e-Science*，第 1–10 页，2023 年。 doi:10.1109/e-Science58273.2023.10254796.[3]Grafana Cloud 自适应指标：推荐的聚合规则。https://grafana.com/docs/grafana-cloud/adaptive-telemetry/adaptive-metrics/manage-recommendations/understand-recommended-rules/。访问日期：2026年11月22日。[4] Nicholas J. Higham。《数值算法的精度与稳定性》。SIAM，第 2 版，2002 年。[5] Pu Jiao、Sheng Di、Hanqi Guo、Kai Zhao、Jiannan Tian、Dingwen Tao、Xin Liang 和 Franck Cappello。面向科学数据的感兴趣量保持型有损压缩。*Proc. VLDB Endow.*，16(4):697–710，2022 年。 doi:10.14778/3574245.3574255.[6] Daniel Kroening 和 Michael Tautschnig。CBMC——C 有界模型检查器。载于*Proc.TACAS*，*LNCS*第 8413 卷，第 389–391 页，2014 年。 doi:10.1007/978-3-642-54862-8_26.[7] Xin Liang、Kai Zhao、Sheng Di、Sihuan Li、Robert Underwood、Ali M. Gok、Jiannan Tian、Junjing Deng、Jon C. Calhoun、Dingwen Tao、Zizhong Chen 和 Franck Cappello。SZ3：用于组合基于预测的误差有界有损压缩器的模块化框架。*IEEETransactions on Big Data*，9(2):485–498，2023 年。 doi:10.1109/TBDATA.2022.3201176.[8] Chunbin Lin、Etienne Boursier 和 Yannis Papakonstantinou。Plato：在压缩时间序列上进行具有严格确定性误差保证的近似分析。*Proc. VLDB Endow.*，13(7):1105–1118，2020 年。 doi:10.14778/3384345.3384357.[9] G. Nigel N. Martin。范围编码：一种从数字化消息中移除冗余的算法。载于*Video and Data Recording Conference*，南安普敦，1979 年。[10] Carlos Enrique Muñiz-Cuza、Matthias Boehm 和 Torben Bach Pedersen。CAMEO：用于有损时间序列压缩的自相关保持型折线简化。载于*Proc.EDBT*，第 15–28 页，2026年。 doi:10.48786/edbt.2026.02.[11] Carlos Enrique Muñiz-Cuza、Søren Kejser Jensen、Jonas Brusokas、Nguyen Ho 和 Tor- ben Bach Pedersen。评估误差有界有损压缩对时间序列预测的影响。载于*Proc. EDBT*，第 650–663 页，2024年。 doi:10.48786/edbt.2024.56。

[12]node-mixin 和 prometheus-mixin 告警规则。https://github.com/prometheus/node_exporter/tree/master/docs/node-mixin，https://github.com/prometheus/prometheus/tree/main/documentation/prometheus-mixin。[13] Tuomas Pelkonen、Scott Franklin、Justin Teller、Paul Cavallaro、Qi Huang、Justin Meza 和 Kaushik Veeraraghavan。Gorilla：一种快速、可扩展的内存时间序列数据库。*Proc. VLDB Endow.*，8(12):1816–1827，2015年。doi:10.14778/2824032.2824078.[14] Prometheus 演示服务器。https://prometheus.demo.prometheus.io，https://demo.promlabs.com。访问日期：2026年11月2日和2026年11月4日。[15] Congrong Ren、Sheng Di、Katrin Heitmann、Franck Cappello 和 Hanqi Guo。误差有界有损压缩中科学粒子数据的簇保持，2026年。arXiv:2604.18801。[16] Bogdan Serykh。用于告警保持型有损压缩的真实 Prometheus 序列（7天和30天快照）。Zenodo，2026年。doi:10.5281/zenodo.24442344.[17] Guoyou Sun、Panagiotis Karras 和 Qi Zhang。Shrink：通过语义提取和残差编码进行数据压缩。载于*Proc. IEEE BigData*，第 650–659 页，2024年。doi:10.1109/BigData62323.2024.10825287。[18] Prometheus 作者团队。Prometheus：监控系统和时间序列数据库。https://prometheus.io.[19] Prometheus 作者团队。Prometheus 源代码，版本3。https://github.com/prometheus/prometheus，2026年。[20] Juniper Tyree、Robert Underwood、Clément Bouvier、Daniel Köhler、Tim Reichelt、Pe- ter Dueben、Sara Faghih-Naini、Heikki Järvinen 和 Milan Klöwer。压缩防护措施：在有损数据压缩中建立信任。EGUsphere 预印本，2026年。软件：https://github.com/juntyr/compression-safeguards，版本 1.0.0rc3。doi:10.5194/egusphere-2026-4266。[21]Alexa VanHattum、Daniel Schwartz-Narbonne、Nathan Chong 和 Adrian Sampson。在 Rust 中验证动态 trait 对象。载于*Proc. ICSE-SEIP*，第 321–330 页，2022年。doi:10.1145/3510457.3513031.[22] Lin Yan、Xin Liang、Hanqi Guo 和 Bei Wang。TopoSZ：在误差有界有损压缩中保持拓扑。*IEEE Trans. Vis. Comput. Graph.*，30(1):1302–1312，2024年。doi:10.1109/TVCG.2023.3326920。