软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.xxxxxx]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

浮点时序数据压缩综述*

朱明辉[1,2], 李政[1,2], 李瑞远[1,2], 陈超[1], 郑宇[3]

[1](重庆大学 计算机学院, 重庆 401331)

[2](重庆大学 时空实验室, 重庆 401331)

[3](北京京东智能城市大数据研究院, 北京 100176)

通讯作者: 李瑞远, E-mail: ruiyuan.li@cqu.edu.cn

摘 要: 物联网技术的发展产生了海量的浮点时序数据, 这给数据存储和传输带来了巨大挑战. 为此, 浮点时序数据压缩变得至关重要, 其按数据可逆性分为有损压缩和无损压缩. 有损压缩方法通过舍弃部分数据信息以实现较好的压缩率, 适用于对精确性要求较低的应用. 无损压缩方法在减小数据大小的同时保留了所有数据信息, 这对于需要保持数据完整性和准确性的应用至关重要. 此外为了满足边缘设备的实时监控需求, 流式压缩算法应运而生. 当前时序压缩综述论文存在梳理不全面、脉络不清晰、分类标准单一、未归纳较新的具有代表性算法等问题. 我们对历年来的时序数据压缩算法按有损压缩和无损压缩进行划分, 并进一步区分不同的算法框架, 包括基于数据表示、基于预测、基于机器学习、基于变换等, 同时对流式与批式的压缩特征进行归纳. 然后对各种压缩算法的设计思路进行深入分析, 并给出各算法的发展脉络图. 接着结合实验比较各类算法的优势与不足. 最后总结算法常见的应用场景, 并对未来研究进行展望.

关键词: 无损压缩; 有损压缩; 浮点时序压缩; 数据压缩

中图法分类号: TP311

中文引用格式: 朱明辉,李政,李瑞远,陈超,郑宇. 浮点时序数据压缩综述.软件学报,202x,xx(xx). http://www.jos.org.cn/1000-9825/xxxx.htm

英文引用格式: Zhu MH, Li Z, Li RY, Chen C, Zheng Y. Survey of Floating-point Time Series Compression. Ruan Jian Xue Bao/Journal of Software, 202x (in Chinese). http://www.jos.org.cn/1000-9825/xxxx.htm

Survey of Floating-point Time Series Compression

ZHU Ming-Hui[1,2], LI Zheng[1,2], LI Rui-Yuan[1,2], CHEN Chao[1], ZHENG Yu[3]

[1](College of Computer Science, Chongqing University, Chongqing 400044, China)

[2](Start Lab, Chongqing University, Chongqing 400044, China)

[3](JD Intelligent Cities Research, Beijing 100176, China)

Abstract: The advances of IoT (Internet of Things) have triggered off a sheer volume of floating-point time-series, which imposes great challenges in storing and transmitting these data. To this end, floating-point time series compression is extremely crucial. It can be divided into lossy and lossless compression based on data reversibility. Lossy compression methods achieve better compression ratio by discarding some data information, which are suitable for applications with lower precision requirements.

---

* 朱明辉和李政为共同第一作者

* 基金项目: 国家自然科学基金(62202070, 62322601, 62172066, 62076191), 中国博士后面上基金(2022M720567), 中央高校基金(2024IAIS-QN017), 山东省重大基础研究项目(ZR2024ZD03), 高端装备机械传动全国重点实验室自主研究课题(SKLMT-ZZKT-2024R07)

收稿时间: xxxx-xx-xx; 修改时间: xxxx-xx-xx; 采用时间: xxxx-xx-xx

Lossless compression methods, while reducing data size, retain all data information, which are essential for applications that require maintaining data integrity and accuracy. In addition, in order to meet the requirements of real-time monitoring on edge devices, streaming compression algorithms have emerged. Current reviews on time series compression encounter issues such as incomplete coverage, unclear line of thought, single classification standards, and lack of inclusion of up-to-date representative algorithms. We categorize the time series compression algorithms into lossy compression and lossless compression, further distinguish different algorithms by the adopted frameworks, including data representation-based, prediction-based, machine learning-based, and transformation-based frameworks, and summarize the compression characteristics of streaming and batch processing. Then, we analyze the ideas of various compression algorithms, and summarise the development of these algorithms. Next, we summarize the advantages and shortcomings of various algorithms with experiments. Finally, we conclude the common application scenarios, as well as the outlook for future research.

Key words: floating-point time series compression; lossless compression; lossy compression; data compression

近年来, 随着物联网技术的发展[1]浮点时序数据(简称时序数据. 如无特殊说明, 本文所述时序数据均为浮点型时序数据)生成的速度和规模达到了前所未有的水平. 例如, 在一个 60 万千瓦的中型火力发电机组中, 有超过一万个传感器, 每秒可以产生数以万计的实时浮点监控记录[2][3]; 中国最大的共享出行公司滴滴每天可以产生超过 150 亿个数据点, 其每天处理的数据量高达 70TB[4][5]; 京东 60,000 个快递员每天可以产生 1TB 的 GPS 记录点[6]. 对大规模时序数据的存储、传输和处理不仅对计算资源提出了巨大的需求, 同时也增加了能源消耗和硬件成本. 此外, 目前数据库正从本地部署向云端迁移, 从目前发展趋势看, 云数据库市场规模增长速度将高于总体数据库市场规模增长速度[7]. 在无需担心硬件问题的云数据库使用场景下, 数据传输成为限制云数据库使用的瓶颈[8]. 因此, 寻找有效的时序压缩方法成为提高计算效率和降低资源开销的关键问题.

在研究时序数据压缩过程中, 挖掘和利用其周期性和相似性等特征是提高算法压缩性能的关键. 例如, 温度监测数据常显示出日间升温和夜间降温的周期性规律, 这种周期性可以用来预测未来数据点的大致范围, 从而实现更高效的编码. 同时, 由于相邻时序数据变化较小, 利用这一特点可以挖掘数据间的共性, 进而减少所需存储空间. 相对于传统的通用压缩技术, 这些方法在处理浮点时序数据时能够更好地利用时序数据的特性及浮点数据结构, 从而在压缩率、压缩时间和解压缩时间之间取得更好的平衡. 理想的时序数据压缩技术[9]应当显著减小数据大小, 节省存储空间, 降低 I/O 开销, 并支持高效的查询处理, 无需解压或可快速解压数据. 科研人员提出了多种解决方案, 包括使用数学变换如离散余弦变换或快速傅里叶变换减少数据冗余, 以及采用基于字典的压缩技术通过识别和存储数据中的重复模式来实现高效压缩.

在面对时序数据压缩的不同应用场景时, 我们可以根据需求选择无损或有损压缩方法. 针对严格的数据准确性场景, 如数据库系统[10]、网络传输和航天计算[11]等, 通常采用无损压缩算法, 因为它能够确保数据在压缩和解压缩过程中的完整性. 而在对数据精度需求相对较低的情况下, 有损压缩算法能够以牺牲一定的精度为代价, 显著提高数据处理的效率. 此外, 考虑到实时监控和实时分析场景中时序数据的流式产生特性, 流式压缩算法也显得尤为重要[12][13]. 这类算法支持对连续生成的数据进行即时处理, 不仅减少了存储空间的需求, 还能在减少数据传输和存储延迟的同时, 提高系统的响应速度. 与传统的、以批处理模式执行的通用压缩算法相比, 现有的流式浮点时序数据压缩算法已经取得了较大进展. 对于非实时的大批量数据处理, 新型的针对浮点时序数据的批处理压缩算法的压缩性能相较于通用压缩算法也有很大的提升.

为了迅速了解时序压缩领域的相关概念和最先进的算法, 探索未来的研究方向, 已有一些关于浮点时序压缩的综述文章. Jayasankar 等人[14]从压缩算法的不同分类角度入手, 根据数据质量、编码框架、数据类型和应用范围分别介绍了数据压缩算法, 详细阐述了经典的通用压缩算法, 如 Huffman、RLE、LZ 系列[15][16]等, 并对无线传感器网络、医学影像以及其他相关应用领域的数据压缩算法进行了全面调研. Chiarot 等人[17]从时序数据压缩算法的总体方法和特点入手, 将最相关的时序压缩算法归纳为四种类型: 基于字典、函数近似、自动编码器和顺序算法, 并总结了不同算法的特征, 如无损、自适应性、对称性等. 此外, 作者还调研了不同时序压缩算法在原实验中获得的压缩率和准确性. Oliveira 等人[18]通过一系列选择标准选出了 40 篇应用于 IoT 领域的时序压缩算法, 并强调了时序压缩和机器学习的关系. 同时作者还分析了不同算法的时间趋

势、出版商、国家及作者之间的联系, 并归纳了实验设置所涉及的程序语言、数据集和指标.

当前时序压缩领域综述论文存在梳理不全面、脉络不清晰、分类标准单一、未归纳最新代表性算法等不足. 考虑到浮点时序压缩算法的迫切需求, 本文对该领域历年的研究工作进行了深入的调研, 通过广泛收集和审阅大量工作, 最终挑选出与本研究高度相关的优质文献, 以进行详细的综述和分析. 本文的主要贡献在于:

(1) 对历年来的浮点时序数据压缩算法进行总结分类, 包括无损压缩和有损压缩, 并且对每种压缩算法的主要思想进行更细粒度的分类, 包括基于数据表示、基于预测、基于机器学习、基于变换.

(2) 本文首次引入了流式/批式压缩的分类概念, 系统地总结了流数据压缩的关键技术和方法, 为流式数据压缩技术提供了指导性的概述.

(3) 本文根据时序压缩算法的核心思想梳理了详细的脉络图, 重点介绍每个里程碑的创新点和改进点.

(4) 本文对典型的时序压缩算法进行了系统的定量实验, 旨在全面评估其性能表现, 并开源了实验代码以及数据集. 此外整理汇总了部分多时序数据实验, 为时序数据处理领域的研究提供有价值的参考.

(5) 本文对浮点时序数据压缩算法按照传输和存储两大应用场景进行讨论, 展现了广泛的应用前景.

(6) 本文展望了浮点时序数据压缩的未来发展方向, 重点探讨多维时序数据压缩、不解压查询与应用、软硬件结合以及特殊浮点型数据压缩四个关键研究领域, 旨在推动数据压缩技术的创新和高效应用.

本文第 1 节介绍数据压缩的相关工作和基础知识. 第 2 节介绍分类方法. 第 3、4 节分别介绍无损压缩和有损压缩算法的分类. 第 5 节展现了代表性算法的发展脉络. 第 6 节实验对比分析了代表性的压缩算法. 第 7 节介绍浮点时序压缩算法的应用. 第 8 节对数据压缩算法的未来研究方向进行讨论. 最后总结全文.

1 相关背景

1.1 时间序列数据

时序数据(Time Series, TS)是数据点按时间顺序递增的有界序列或无界序列[19]. 根据数据维度不同, 时序数据主要分为单时序数据(Univariate Time Series, UTS)和多时序数据(Multivariate Time Series, MTS).

图 1 股票日 K 线

单时序数据中的数据是一维的, 例如某一新能源汽车品牌在最近一年中的月销量. 多时序数据中的数据则是多维的, 包含多个观测变量, 这些变量在时间上是相关的, 通常由同一个系统生成. 例如, 图 1 所示的每日股票交易摘要(包括开盘价、收盘价、最高价、最低价等其他信息)可以建模为 MTS. 时间序列的数学形式为:

$$TS=[(t_1,x_1),...,(t_n,x_n)],x_i\in\mathbb{R}^m \tag{1}$$

其中, x_i 是采集到的 m 维浮点型数据值, t_i 是数据值 x_i 获取时的时间戳, n 是数据点的数量, m 是 MTS 的数据维度, 当 $m=1$ 时 TS 为 UTS. 给定 $i\in[1,n]$, 则 $TS[i]$表示时间序列 TS 中的第 i 个数据(t_i, x_i).

理想情况下时序数据的采样时间是固定的, 即相邻数据的时间戳间隔是相同的, 此时只需要存储第一个数据的时间戳, 后续时间戳以此为基准顺序加特定值即可[20]. 但是由于传感器误差等因素, 时序数据的时间戳会存在一些误差. 因此时序数据的时间戳通常采用 Delta 编码或者 Delta-of-Delta 编码[13]进行压缩, 在时间戳间隔较小的情况下可以有着较好的压缩率. 以 Delta-of-Delta 编码为例, 时间戳的具体编码过程如下:

(1) 数据块头存储起始时间戳 t_{-1}, t_{-1} 对齐到了最近的时间窗口(假设 2 小时的时间窗口, 则 t_{-1} 可以对齐到任何 2 小时整数倍的时间点, 如 00:00、02:00、04:00 等);

(2) 第一个时间戳 t_0 在数据块中被存储为 t_0 相对于 t_{-1} 的增量 Delta, 即 t_0-t_{-1}, 若数据块小于 4 小时, 则可

以用 14 位来表示, 即 2^{14} = 16,384 可以涵盖 4 小时所能表示的 14,400 秒;

(3) 后续时间戳 t_n, 存储其增量的增量 $D=(t_n-t_{n-1})-(t_{n-1}-t_{n-2})$, 根据 D 的时间跨度, 由标志位 *flag* 和 D 来表示, 如表 1 所示.

表 1 Delta-of-Delta 编码规则表

D 的范围	*flag*	D 所需要的位数	总位数
0	1	无	1
[-63, 64]	10	7	9
[-255, 256]	110	9	12
[-2047, 2048]	1110	12	16
其他	1111	32	36

通过以下例子来进一步说明 Delta-of-Delta 和 Delta 的编码过程. 如表 2 所示, 起始时间戳 t_{-1} 使用 64 位存储原始值, 计算第一个时间戳 t_0 相对于 t_{-1} 的增量并用 14 位进行存储. 第二个时间戳 t_1, 计算其增量的增量 $D=(t_1-t_0)-(t_0-t_{-1})=-1{,}799{,}900$, 根据上述的编码规则, -1,799,900 的范围在[-2047, 2048]之外, 因此存储 *flag* 位'1111'和 D 所需的 32 位, 共 36 位. 随后的时间戳 t_n 与 t_1 编码过程一样. 经过 Delta-of-Delta 编码后, 时间戳所需存储空间为: 64 + 14 + 36 + 1 + 1 + 9 + 9 = 134 位, 相比原始大小(不包含 t_{-1}): 64 × 6 = 384 位, 压缩率为 134 ÷ 384 ≈ 0.35. 如表 2 第二列所示, 若采用 Delta 编码的方案, 每个时间戳的增量所需要的比特数为 $\lceil \log_2 Max(delta) \rceil$, 其中 $Max(delta)$表示最大增量值. 本例中增量需要用$\lceil \log_2 1{,}800{,}000 \rceil$ = 21 个比特来表示, 因此, 时间戳所需存储空间为: 64 + 21 × 6 = 190 位, 压缩率为 190 ÷ 384 ≈ 0.49, 压缩效果较前者略差.

表 2 Delta-of-Delta 和 Delta 编码示例

时间戳 t_n	增量	Delta 编码所需位数	增量的增量	压缩后的位数
t_{-1} = 1709870400000	-	64	-	64
t_0 = 1709872200000	1,800,000	21	-	14
t_1 = 1709872200010	10	21	-1,799,900	36
t_2 = 1709872200020	10	21	0	1
t_3 = 1709872200030	10	21	0	1
t_4 = 1709872200045	15	21	5	9
t_5 = 1709872200050	5	21	-10	9

鉴于针对时间戳的 Delta-of-Delta 和 Delta 压缩方法已经达到较好的压缩率, 当前研究主要集中在浮点类型数值压缩展开, 故本文主要针对浮点类型数据值的压缩进行讨论.

1.2 浮点数据类型

根据 IEEE 754 标准, 一个双精度浮点数 v 使用 64 个二进制位来存储, 其中第 1 个比特表示符号位 s, 中间 11 个比特表示指数位 $\vec{e}=<e_1,e_2,\ldots,e_{11}>$, 最后的 52 个比特表示尾数位 $\vec{m}=<m_1,m_2,\ldots,m_{52}>$, 如图 2 所示.

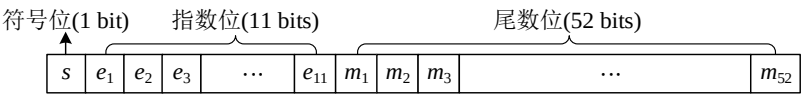


图 2 双精度浮点数格式

当 v 为正数时, s = 0, 否则 s = 1. 根据 $\vec{e}$ 和 $\vec{m}$ 的取值, 双精度浮点数 v 可以分为规约形式浮点数和非规约形式浮点数, 其中规约形式浮点数是时间序列中最常见的情况, 它的值满足:

$$\begin{aligned} v &= (-1)^s \times 2^{e-1023} \times (1.m_1 m_2 \ldots m_{52})_2 \\ &= (-1)^s \times 2^{e-1023} \times \left(1+\sum_{i=1}^{52} m_i \times 2^{-i}\right) \end{aligned} \tag{2}$$

其中 e 是 $\vec{e}$ 的十进制值, 即 $e=\sum_{i=1}^{11}e_i\times 2^{11-i}$. 单精度浮点数使用 32 个二进制位来存储, 与双精度浮点数不同的是, 其指数位为 8 个比特, 尾数位为 23 个比特. 后文中如无特别说明, 浮点数指的都是双精度浮点数.

1.3 前导零、尾随零和中心位

如图 3 所示为一个双精度浮点数的 IEEE 754 形式, 其前导零为二进制数最高位 1 之前全为零的序列, 尾随零为最低位 1 之后全为零的序列, 中心位为前导零和尾随零之间的比特位所组成的序列.

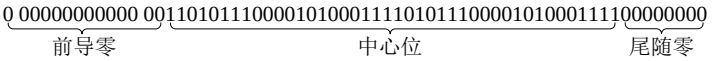


图 3 前导零、中心位和尾随零示意图

1.4 度量指标

为了验证浮点时序压缩算法的效果, 通常会考虑三个度量指标: 压缩率、压缩/解压缩时间和精度.

压缩率(Compression Ratio, CR). 这个指标衡量压缩算法的有效性, 它被定义为

$$R=\frac{s'}{s} \tag{3}$$

其中 s' 是压缩后数据的大小, s 是原始数据的大小, 压缩率 R 一般是越小越好. 但我们经常可以看到, 现有的一些文献, 如文献[21]会将 R 的倒数, 即原始数据大小与压缩后的数据大小之比, 作为压缩率来检测压缩算法的性能, 此时的压缩率 1 / R 则越大越好.如无特殊说明, 本文所述的压缩率越小表示压缩算法越有效.

压缩/解压缩时间(Compression Time/Decompression Time). 也称为压缩延迟(Compression Latency)与解压缩延迟(Decompression Latency), 分别表示从原始数据生成压缩数据所需的时间、从压缩数据恢复到重构数据所需的时间, 常用于评估压缩性能, 时间越少意味着速度越快.

吞吐量(Throughput). 不同于压缩/解压缩效率反映压缩/解压缩过程中所需的时间, 吞吐量指单位时间内处理数据的能力, 单位通常采用 MB/s. 但两者密切相关, 对于同一数据, 高速意味着短时间可以完成更多数据的压缩或解压缩, 从而提高吞吐量. 吞吐量主要受 CPU 处理能力、线程数、算法效率和数据特性的影响.

精度(Accuracy)用于有损压缩衡量重构时间序列相对于原始时间序列的保真度, 反映压缩的质量. 可以使用不同的指标来确定精度:

(1) 逐点误差(Pointwise Error), 反映有损压缩精度的一种最简单的方式, 常见的有绝对误差(Absolute Error)和相对误差(Relative Error), 计算公式如下:

$$\begin{aligned} absolute\_error_i &= |x_i-\bar{x}_i| \\ relative\_error_i &= \left|\frac{x_i-\bar{x}_i}{x_i}\right| \end{aligned} \tag{4}$$

其中, x_i 表示原始数据集 D 中的第 i 个离散数据值, $\bar{x}_i$ 表示 x_i 的重构数据值. 误差有界的有损压缩算法通过预定义或用户自定义来设置一个误差限 $\epsilon>0$, 在压缩过程中使得逐点误差小于等于误差限 ϵ 以满足精度要求.

(2) 范围误差(Range Error), 在已知一个数据集 D 的范围 (即最大值和最小值之差) 情况下, 可以由绝对误差转换而来, 计算公式如下:

$$range\_error_i=\frac{|x_i-\bar{x}_i|}{\max(D)-\min(D)} \tag{5}$$

其中, $\max(D)$和 $\min(D)$ 是数据集 D 的最大值和最小值. 范围误差适用于保持数据整体趋势和范围的场景.

(3) 均方误差(Mean Squared Error, MSE), 直接测量原始数据与压缩数据之间差异的平均平方值, 反映了误差的总体水平. 若数据集大小为 n, 计算公式如下:

$$MSE=\frac{\sum_{i=1}^{n}(x_i-\bar{x}_i)^2}{n} \tag{6}$$

(4) 均方根误差(Root Mean Squared Error, RMSE), MSE 的平方根, 提供与原数据尺度一致的误差评估, 相比 MSE 更直观反映了误差大小, 计算公式如下:

$$RMSE = \sqrt{MSE} \tag{7}$$

(5) 峰值信噪比(Peak Signal to Noise Ratio, PSNR)是 SNR 的一个变体, 常用于衡量数据的可视化质量, 基于 MSE, 考虑了数据的相对范围, 计算公式如下:

$$PSNR = 10\log_{10}\frac{[\lambda(range(D))]^2}{MSE} \tag{8}$$

其中, $range(D)$表示原始数据集 D 的数值范围, 即最大值与最小值之差, λ 是相关系数, 通常取值 1 或 1/2. 通常 PSNR 越大表示压缩质量越好.

(6) 精度增益(Accuracy Gain)由 Lindstrom[22]提出, 用于衡量压缩效果, 计算公式如下:

$$\alpha = \log_2\frac{\sigma}{RMSE} - R \tag{9}$$

其中, σ 是原始数据的标准差, R 是压缩率. 当 R 和 $RMSE$ 都有显著差异时, α 可以有效地衡量压缩效果. 当 α 增加时, 表明达到了更好的压缩效果, 即用更少的比特数表示数据的同时引入的误差更小.

对于不同的有损压缩算法, 其压缩质量的评估可能不同, 比如在科学数据的可视化分析场景下, 可以结合 PSNR 和 α 来共同衡量压缩效果. 如表 3 所示, 根据评估需求选择合适的精度指标, 有助于全面了解压缩效果.

表 3 有损压缩的精度指标

精度指标	评估需求	采用的代表性算法
逐点误差(Pointwise Error)	精确量化误差	SZ[21]、LFZip[23]
范围误差(Range Error)	控制相对精度和误差	SZ3[24]、SZ_ADT[25]
均方误差(MSE)	总体误差水平	文献[26]
均方根误差(RMSE)	总体误差水平	NUMARCK[27]
峰值信噪比(PSNR)	可视化数据压缩质量	ZFP[28]、NUMARCK[27]、MGARD+[29]
精度增益(α)	综合压缩效果	ZFP[28]、SPERR[30]

1.5 通用压缩算法

在数据压缩领域, 通用压缩算法是一种广泛应用的技术, 这些算法通常设计得比较灵活, 能够处理各种类型的数据, 从而实现对大范围数据格式的有效压缩. 相比针对浮点时序数据的压缩算法, 通用压缩算法的主要优点包括适用范围广、实现复杂度低, 能够实现较高的压缩率. 但是, 一些压缩率较高的算法可能在压缩时需要更多的计算资源, 导致它们压缩时间和解压时间较长. 此外, 大多数通用压缩算法需要利用统计信息从而无法直接支持流式数据压缩.

通用压缩算法可以分为轻量级压缩与集成压缩. 常见的轻量级压缩有位打包(Bit-packing)、字节打包(Byte-packing)、游程编码(Run Length Encode, RLE)、参考系编码(Frame of Reference, FOR) [31]、熵编码(Entropy Encode, 如 Huffman 编码、算数编码[32]、非对称数字系统(Asymmetric Number System, ANS)[33])、字典压缩(Dictionary Compression, 如 LZ77[15]、LZ78[16])、Snappy[34]等等. 得益于较低的实现复杂度和计算资源要求, 轻量级压缩常用于浮点压缩算法中以进一步提高压缩率.

集成压缩通常组合了多种轻量级编码算法, 能够实现更高的压缩率. 如 Deflate[35]首先使用 LZ77 算法进行数据压缩, 然后对结果使用 Huffman 编码进一步压缩; GZip[35]则基于 Deflate 进行压缩, 同时包含了文件头、文件尾、校验和等元数据信息. 近年来, 得益于熵编码中非对称数字系统的发展, 特别是有限状态熵(Finite State Entropy, FSE)[36]的应用, 通用压缩算法取得了巨大进步, 如 Zstd[37]、Brotli[38]、LZ4[39]等方法的压缩性能明显优于 GZip 等传统方法, 因此经常作为一些无损浮点压缩算法的对比基准.

2 分类方法

在本节中, 我们将介绍现有时序浮点压缩算法的总体分类, 具体的无损和有损压缩算法的算法框架和流

批处理特征将在下两小节进行总结.

我们从三个维度对现有算法进行分类: (1) 可逆性(包括无损压缩和有损压缩); (2) 算法框架(基于数据表示、基于预测、基于深度学习等等); (3) 流批处理特征. 具体代表性算法的分类如图 4 所示.

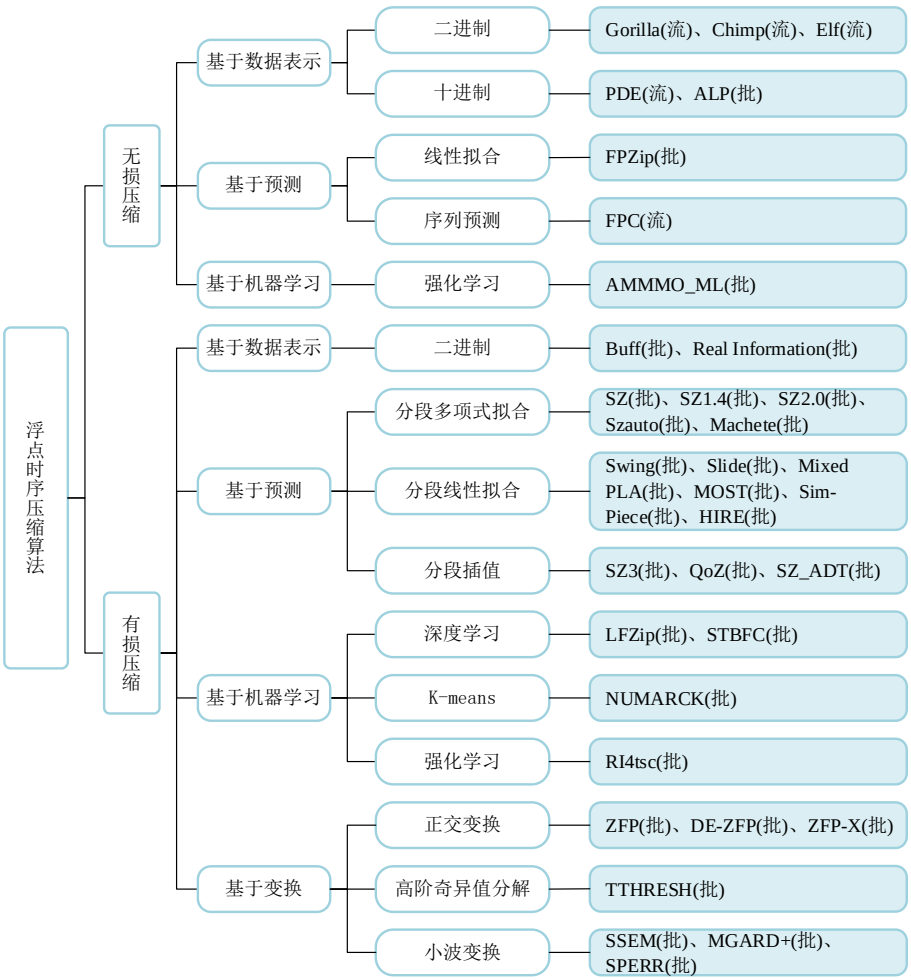
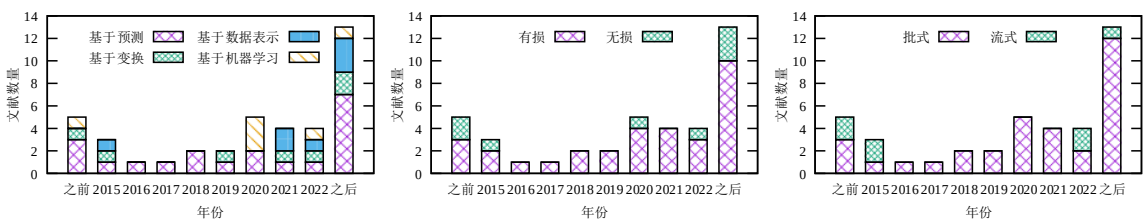


图 4 浮点时序压缩代表性算法分类



(a) 不同算法框架 (b) 有损/无损压缩 (c) 批式/流式压缩

图 5 浮点时序压缩算法的时间趋势

我们所考虑的算法(包括应用部分)的时间跨度为 18 年, 其中一个重要的无损压缩算法可以追溯到 2006 年.图 5 以柱状图的形式展示了算法论文的发表数量随时间变化的趋势. 从图 5(a)中可以看出, 基于预测和变换的浮点时序压缩算法相对成熟, 持续吸引着研究者的注意, 特别是基于预测的浮点压缩算法, 一直是浮点压缩领域的热点. 近几年来, 基于数据表示和机器学习的浮点时序压缩算法的研究有了突破性进展, 也获得了更

多关注. 图 5(b)和图 5(c)分别为有损/无损压缩和批式/流式压缩的时间趋势, 可以看出浮点时序压缩领域更多关注有损压缩和批式压缩. 此外, 无损压缩与流式压缩的时间趋势具有一定的相似性, 从后文中也可以发现, 近年来流式无损压缩算法的研究活跃度呈现出显著增长.

2.1 可逆性

在可逆压缩中, 压缩和解压缩操作是完全可逆的, 不会导致数据丢失. 原始数据可以通过解压缩过程还原到与压缩前相同的状态. 可逆压缩也称为无损压缩, 常用于那些不能容忍任何信息损失的数据, 如文本文件、数据库文件等.

不可逆压缩引入了信息损失, 解压缩后的重构数据不完全等同于原始数据. 这种压缩方法通常会去除一些不太显著的数据或冗余数据以减小文件大小. 也称为有损压缩. 有损压缩常用于音频、图像、视频等媒体文件. 其中一定程度的信息损失是可以被接受的.

2.2 算法框架

算法框架也是区分不同压缩算法的一个关键角度. 依据其设计动机, 不同的算法框架能够在满足相应需求的前提下, 实现不同的压缩率和性能. 类似于通用压缩中基于字典的 LZ 系列算法, 部分浮点时序压缩算法也借鉴了这种思想, 如 TRISTAN[40]和 CORAD[41], 但由于字典的内存开销较大、算法复杂度较高以及在处理变化的数据特征时不够灵活等问题, 最新的浮点时序压缩算法很少将字典考虑在内.

2.2.1 基于数据表示

浮点数据分为二进制表示形式和十进制表示形式, 比如我们所熟悉的 3.14 在计算机中常以 IEEE 754 格式表示: 0 10000000000 1001000111101011100001010001111010111000010100011111. 从上述两种视角出发, 又可以分为基于二进制的压缩算法和基于十进制的压缩算法, 且常见于无损浮点时序压缩中.

基于二进制的压缩算法基于假设: 连续数值之间的差异通常会比较小. 在文献[20]中, 作者首次探讨了将异或(Exclusive OR, XOR)运算应用于浮点时序数据压缩的方法, 通过异或处理后得到的结果相较于原始数据展现出更高的不均匀性, 从而提高了后续 Huffman 编码的压缩效果. 异或运算在优化浮点时序数据压缩方面展现了显著的潜力. 最近的研究发现, 在许多时序数据中, 指数的偏差一般很小, 当两个相似的浮点数进行异或运算后, 通常会产生较多的前导零(以及可能较多的尾随零)[42]. 通过记录前导零、中心位的位数, 以及完整的中心位, 可以实现无损压缩.

3.1415: 0 10000000000 1001001000011100101011000000100000110001001001101111
⊕
3.1416: 0 10000000000 1001001000011111111001011100100100011101000101001111
‖
Δ : 0 00000000000 0000000000000011010111101110110010111111101011001000

图 6 3.1415 与 3.1416 异或示例

然而, 上述记录前导零和尾随零的方法忽略了一种“蝴蝶效应”, 即十进制浮点数值的轻微变化, 会引起计算机底层二进制表示的巨大差异. 例如, 3.1415 与 3.1416 在十进制层面仅相差 0.0001, 但它们各自的 IEEE 754 格式从第 27 位开始就出现分歧, 如图 6 所示, 异或结果仅有 3 位尾随零. 虽然可以通过一些预处理操作来增加尾随零的数量, 如算术擦除[12]与多前值比较[43], 但可能影响压缩与解压缩效率.

基于十进制的压缩算法, 主要思想是将浮点数乘以 10 的 n 次幂转换为整数(n 为大于等于 0 的整数), 例如, 将浮点数 3.14 乘上 10^2 得到整数 314, 然后使用轻量级通用压缩算法对 2 (表示 10 的 2 次方) 和 314 进行编码以实现压缩效果. 但是, 由于 IEEE 754 格式的限制, 并不是所有浮点数都能精确表示, 在经过乘法和除法运算后, 得到的解压缩结果可能与原始值不同, 无法实现无损压缩. 因此, 基于十进制的压缩算法需要对异常情况(即无法做到无损恢复的数值)进行单独处理.

2.2.2 基于预测

时序分析的基本假设是过去的数据包含了影响未来数据的有用信息. 例如, 历史数据中的模式, 如季节性

变化、趋势或周期性, 有可能在未来重复出现. 通过分析数据中的统计规律和结构, 可以构建模型来预测未来的数据点, 从而实现对数据的有效压缩.

浮点时序数据的预测通常利用统计学模型来分析历史数据并进行预测, 常见的方法包括自回归模型、多项式回归、插值, 还有一些算法引入了 Lorenzo 预测器[44]. 对于有损压缩而言, 目前主流的基于预测的浮点压缩算法是分段函数拟合, 且可以分为分段多项式拟合(Piecewise Polynomial Approximation, PPA)和分段线性拟合(Piecewise Linear Approximation, PLA).

函数拟合的主要思想是, 使用函数来近似表示时间序列, 通常是低阶多项式. 由于真实数据的复杂多样性, 只用一个函数来拟合整个时间序列是相对困难的, 因此需要对时间序列进行分段, 为每个分段找到一个合适的拟合函数. PPA 使用多个低阶多项式, 如常数函数、一次函数、二次函数, 来拟合不同的时序分段. PLA 则仅使用线性函数进行拟合, 且分段数量越少压缩效果越好. 虽然函数拟合使得压缩有损, 但是不依赖于数据域, 且不需要训练阶段, 因为回归算法只考虑孤立的单个分段. 插值方法也常用于浮点时序压缩的预测阶段. 线性样条插值通过简单地连接相邻数据点构成直线段, 为变化平缓的数据提供了一种快速且有效的预测方法. 相比之下, 三次样条插值采用三次多项式来连接各数据点, 适用波动较大的时序数据.

2.2.3 基于机器学习

近年来, 随着计算机硬件算力的显著提升, 机器学习算法获得了广泛的应用, 在数据压缩领域也广受欢迎. 机器学习算法能够更好地理解并保留数据的重要特征, 特别是非结构化数据, 可以显著改善数据压缩的效果. 机器学习模型还可以根据数据的特定属性和需求自适应地调整压缩策略, 为用户提供个性化的压缩解决方案.

基于机器学习的浮点时序压缩通常结合了无监督学习、深度学习、强化学习等其他学习方式. 无监督学习是指在没有标签或注释指导的情况下对数据进行模式识别的学习过程, 主要关注于发现数据中的内在结构和关联. 常用于浮点数据压缩的是一些聚类方法, 如 K-means, 可以发现数据中的重复模式和连续迭代之间的新的数据分布, 随后编码到一个简洁表示的索引空间中.

深度学习是一种特殊的机器学习方法, 它使用深层的神经网络结构来学习数据的高级抽象表示. 强化学习是指智能体(Agent)通过与环境(Environment)的交互来学习如何在特定情境下做出决策的过程, 旨在学习一个策略, 使得智能体在长期内获得最大的奖励. 通过与深度学习结合, 设计了一个具有强化学习功能的神经网络结构, 如深度 Q 网络(DQNs)等, 可以自动调节参数, 同时解决了压缩上下文中没有可用的标记训练样本的问题.

2.2.4 基于变换

在信号处理领域, 原始信号的样本点之间通常存在高度的相关性, 通过变换可以将这些高度相关的数据转换为一组去相关的且更加紧凑的表示形式, 使得信号中的许多系数接近于零或可以被忽略, 这非常有助于数据的压缩. 例如, 许多自然信号(如音频、图像等)在频域中表现出能量集中的特性, 通过离散傅立叶变换(Discrete Fourier Transform, DFT)后, 只有少部分的频率系数是重要的, 而大量的系数可以被忽略或用更少的比特数来表示. 对系数进行编码是变换用于压缩的核心, 而压缩的实质就是对系数的量化压缩.

基于变换的浮点压缩算法根据上述思想, 将数据从原始域转换到另一域, 通常是一组基底函数(变换)的域, 来寻找更紧凑的表示. 离散小波变换(Discrete Wavelet Transform, DWT)常用于浮点数据压缩, 使用一组称为小波的基本函数对时间序列进行变换, 特别是 Haar 小波, 得益于较低的计算复杂度和较好的压缩效果, 近年来多应用于高性能计算系统检查点. 另有一些文献采用 CDF 9/7 双向正交小波变换[45], 并使用优化的集合分裂嵌入块编码算法(SPECK)[46]对变化系数进行编码. 高阶奇异值分解(Higher Order Singular Value Decomposition, HOSVD) [47]也广泛应用于数据压缩, 可以把数据分解为多个维度上的正交成分, 将数据的信息集中在较少的奇异值上. 通过存储这些带有较大权重的成分, 可以用较少的数据量实现数据的压缩.

2.3 流批处理

随着物联网和 5G 技术的迅速发展, 实时数据源如传感器等不断产生大量数据流, 这些数据流需要被即时处理和分析以支持实时决策. 由此提出的数据流批一体化技术, 在处理传统的历史批处理数据的同时, 协同对

实时数据流进行处理, 这对现有的数据压缩算法也提出了更高要求. 批式/流式浮点数据压缩算法通常由数据预处理、数据关系挖掘和数据编码三大部分组成, 如图 7 所示. 有些算法在挖掘数据关系之后应用预处理操作(如 SZ[21]先预测, 再量化); 也有算法先执行数据预处理操作然后再挖掘数据关系(如 Elf[12]先执行尾数擦除, 然后执行异或运算); 还有算法跳过数据预处理, 直接进行关系挖掘(如 Gorilla[13]直接执行异或运算). 但所有算法都包含数据编码阶段. 数据预处理和数据关系挖掘本质上是找到数据的另一种表示, 方便后续数据编码.

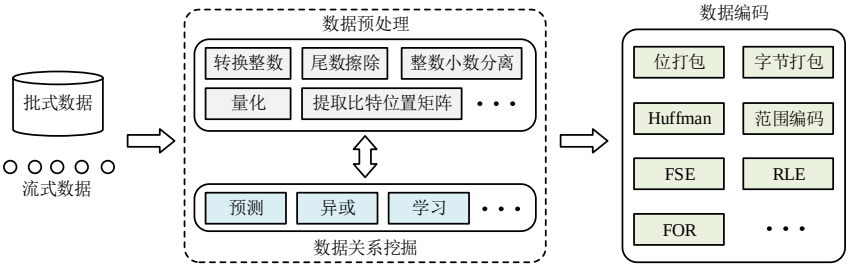


图 7 批式/流式浮点数据压缩技术框架

传统的压缩算法多为批式处理, 通常用于处理大量积累的数据, 这些数据通过一次性加载并压缩, 可以更好地优化整体资源的利用. 批式压缩针对静态数据, 因此更容易实现. 但是面对实时处理场景, 则需要等待足够的数据积累后才可以开始处理, 导致批式压缩算法通常要面临高延迟的问题. 例如, 全球定位系统(Global Positioning System, GPS)传感器通常每 1~10 秒报告一次位置[48], 如果批量地将 1000 条记录传输到监控中心, 延迟会达到惊人的 16~167 分钟, 这显然不满足实时监控的要求. 尽管如姿态传感器等一些传感器能够达到 1000Hz 的采样频率, 如果对 1 秒产生的数据进行批式压缩, 缓存双精度浮点型的记录至少需要 8KB 的存储空间, 这对于存储空间通常不超过 250KB 的边缘传感器设备而言[49], 构成了较大的存储负担.

流式压缩算法适合于连续生成数据的场景, 例如监控系统、实时交易等, 其处理速度快, 延迟低, 可以提供更加快速的数据分析和决策支持. 狭义上的流式压缩算法可以实现对单个数据的即时处理, 即数据来一个处理一个. 但是真实生产场景中可能会考虑压缩效率和实时性的平衡, 因此广义的流式压缩算法包括微批式处理(Mini-batch processing)的方式, 在符合延时要求的前提下, 对小批量数据进行集中处理.

本文对历年来的代表性算法进行归纳总结的同时, 也考虑到了算法是否支持狭义流式压缩的特性, 以对后续的研究工作提供有效的参考.

3 无损压缩

无损浮点时序压缩算法在处理大规模时序数据时起着至关重要的作用, 尤其是在物联网和流场景等领域. 无损压缩算法能够在不损失任何信息的前提下, 显著减少数据的存储空间和传输时间. 根据不同的实现原理, 无损浮点时序压缩算法主要可以分为三类: 基于数据表示的方法、基于预测的方法和基于机器学习的方法.

基于变换的压缩方法在无损压缩领域较为少见, 主要是因为无损压缩要求数据能够被完全恢复, 因此必须保留所有变换后的系数, 包括那些接近零或等于零的不重要系数, 导致基于变换的无损压缩效果并不明显. 同时, 存储变换矩阵可能会引起负压缩增益, 即压缩处理反而增加了数据的存储空间. 此外, 变换过程中的浮点乘法运算可能会生成无法精确表示的数值, 这导致解压结果不能满足无损要求.

3.1 基于数据表示

3.1.1 Gorilla

Pelkonen 等人[13]提出了基于内存的时序数据库 Gorilla, 并部署在了 Facebook. Gorilla 针对浮点时序数据采用异或计算来对相邻的浮点数进行处理. 根据经验观察到, 时序数据中两个连续的值之间的差异不会太显著. 因此, Gorilla 认为, 将当前值与前一个值进行异或计算的结果会得到较多的前导零和尾随零, 然后对异或结果进行编码, 以更少的比特数进行存储.

如图 8 (a)所示, Gorilla 具体编码方案为: (1) 第一个值不进行压缩; (2) 从第二个值开始, 如果异或值为 0, 即当前值与前一个值相等, 只存储一个标志位‘0’; (3) 如果异或值不为 0, 先写入一个标志位‘1’, 再分两种情况讨论: 如果当前异或值的前导零和尾随零数量分别大于前一个异或值的前导零和尾随零数量, 只需存一个标志位‘0’, 然后存储相应位置的中心位. 这样可以共享前一个异或值的前导零和尾随零数量, 从而提高压缩率. 否则, 先存储一个标志位‘1’, 再分别用 5 比特和 6 比特来记录前导零和中心位的数量, 最后存储完整的中心位. 由于异或运算仅仅与上一个数据有关, 因此 Gorilla 可以很好地支持流式数据压缩.

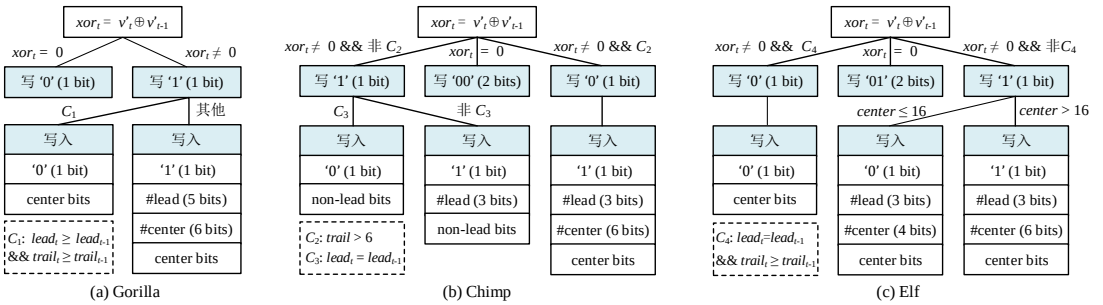


图 8 Gorilla、Chimp 和 Elf 的压缩编码方案

3.1.2 Chimp

Liakos 等人[43]对大量的数据集进行统计分析, 提出了压缩率更好的无损流式浮点压缩算法 Chimp. 与 Gorilla 相似, Chimp 也是基于当前值与先前值的异或计算, 同样可以支持流式压缩处理. 但是 Chimp 更多地利用了异或后结果的前导零和尾随零分布特征, 即大多数时序数据集的逐点异或结果中, 前导零数量大于 7, 尾随零数量较少. 利用这一特征, 如图 8 (b)所示, Chimp 改进了编码方案, 考虑了尾随零的数量, 并且前导零计数采用了一种近似映射方法, 即利用 3 个比特来近似表示前导零数量, 其映射值为[0, 8, 12, 16, 18, 20, 22, 24]. 例如, 前导零数量分别为 6、8、15, 使用上述映射方法可以使用 000→(0)、001→(8)、010→(12)记录下来, 分别对应 0、8、12 个前导零, 剩余的前导零按照中心位进行处理. 大多数时序数据集的异或结果的前导零数量大于 7, 因此 Chimp 仅使用 3 个比特来记录前导零数量可以获得更好的压缩率.

为了获取更多的尾随零, $Chimp_{128}$ 在内存中维护了一个环形缓冲区记录前 128 个值, 以及一个大小为 2^{14} 的数组, 将前值 v_i 的编号 i 记录在数组的 v_i & (2^{14} - 1)位置, 以便后续待压缩值 v_j 快速获得数组中第 v_j & (2^{14} - 1)个位置的编号, 并从缓冲区中读取前值进行异或运算, 这样可以保证异或结果至少有 14 个尾随零.

考虑到 Chimp 较长的解压缩时间, 作者还提出了其变体 Patas[50], 利用单一模式 (即标志位为‘01’时的情况) 进行编码, 同时采用字节对齐的方式, 以轻微牺牲压缩率为代价提供了更快的解压缩速度.

相比 Gorilla, Chimp 主要是对编码方案进行了修改, 所以同样可以很好地支持流式数据压缩. $Chimp_{128}$ 对当前值的压缩与前 128 个值相关, 因此可以支持流式压缩, 但是其对内存大小有了更高的要求.

3.1.3 Elf

Li 等人[12]提出的 Elf 是一种基于擦除的流式无损浮点压缩算法. 为了使异或后的结果产生更多的尾随零, 以此来提高压缩率, Elf 提出了一个巧妙的无损擦除策略. 由于每个浮点类型数据的尾数的低位对数值的影响较小, 在能恢复成原来数据值的情况下尽可能地将更多的尾数低位擦除为零. Elf 提出了 IEEE 754 格式下最优擦除位置的数学计算, 并进行了正确性和高效性的证明. 即:

$$g(\alpha) = 52 - \lceil \alpha \times \log_2 10 \rceil - (e - 1023) \tag{10}$$

其中 α 为十进制的小数位数, e 为二进制指数位的值. $g(\alpha)$即为最优擦除位数, 满足 1) 可以通过向上取整的方式恢复成原数据; 2) 满足前述条件下尾随零位数最多. 如图 9 所示, 通过计算可以得到 3.17 最优的擦除个数为 44, 这样就有效地将 3.17 的尾随零个数从 2 位提高到 44 位. 可以看出擦除位置与数据的指数位值 e 和十进制小数位数 α 相关, 在时间序列中 e 和 α 变化较小, 保证了擦除后的数据进行异或操作, 尾随零同样会增加.

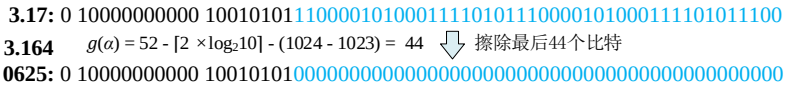


图 9 Elf 擦除示例

Elf 针对擦除后的数据, 又 Chimp 的编码基础上进行了优化, 如图 8 (c)所示, 主要改进是对比了尾随零的数量, 并根据统计数据调整标志位所需的位数. 擦除后的数据根据保存的十进制小数精度信息即可恢复成原数据. 例如 3.17 擦除后十进制的值为 3.1640625, 十进制小数精度为 2, 只需要对 3.1640625 向上取整保留两位小数就可以恢复成原数据. 在后续的 Elf+[51]中, 主要通过优化有效值位数的编码, 提高了整体压缩效果.

Elf 对数据的擦除是仅仅针对数据本身的处理, 而且与 Gorilla 和 Chimp 相比, Elf 进一步修改了编码方案, 但是依然只与上一个数据有关, 因此 Elf 可以进行流式压缩, 而且对内存没有过多的要求.

3.1.4 PDE

Kuschewski 等人[52]提出的 PDE 是一种基于十进制的轻量级压缩方案, 可以实现快速的解压缩和较高的压缩率. 由于 IEEE 754 格式下双精度浮点类型最多可以精确表示十进制数值 10^{22}, PDE 按指数递增的顺序为每个数据找到能精确表示时的指数值, 然后经过乘法和舍入操作将其转换为整数. 如图 10 所示, 对于 0.989…、3.25 和-6.425, 其能精确表示的指数值分别为 2、2、3, 然后乘上各自对应的 10 的幂, 并进行舍入, 得到 99、325 和-6425. 若指数值大于 22 则无法精确表示, 单独存储为异常值. 通过整数转换得到整数列后, 使用其他整数压缩算法进行级联压缩,如 FastFOR[53]、RLE 等等. PDE 算法作为一个独立模块, 可以流式地处理浮点时序数据. 但是, 由于 PDE 算法本身并不具备数据压缩功能, 通常与下游的整数算法级联使用. 因此, 组合算法是否能够实现流式处理, 还需要依赖于下游级联算法的流式处理能力.

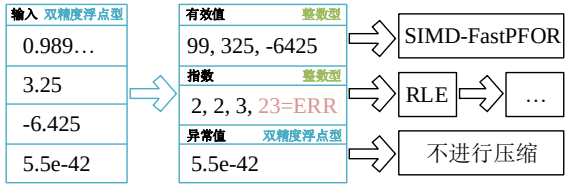


图 10 PDE 级联压缩示例[52]

3.1.5 ALP

为了提高数据吞吐量, Afroozeh 等人[42]在 PDE 的基础上进行了优化, 摒弃了 PDE 的逐点指数计算方案, 转而基于向量, 使用一个较大的指数 e 进行整数化, 并通过 FFOR[53]对数组向量进行编码. 虽然使用较大的指数 e 可以保证整数化后的浮点数仍然能够精确表示, 但是其整数值尾部可能会有冗余的零导致整数编码过长, 从而影响了压缩效果. 因此 ALP 引入了另一个指数系数 f, 以减少整数编码位数. 对于无法恢复的数值, 视为异常值来进行单独存储. 如表 4 所示, 在参数对 $(e, f) = (14, 10)$ 时, 8.0605 整数化后结果为 80605, 可以在该参数下恢复成原始值, 而 1.23456789 无法恢复成原值则视为异常值. ALP 通过计算向量中每对参数值对应的编码长度来选择最优的参数组合.

表 4 ALP 整数化过程

原始浮点数	指数 e = 14 整数化	去零尾指数 f = 10	恢复值	是否为异常值
8.0605	80605000000	80605	8.0605	否
1.23456789	12345678900	12345	1.2345	是

ALP 的优化策略包括, 使用了 Lua[54]中快速舍入方法代替传统舍入操作. 虽然指数值计算是基于向量的, 但 e 和 f 的组合总共有 253 种. 为了获取最佳组合会导致较大的时间复杂度, 因此 ALP 引入了双重均匀采样, 第一层从整个组合空间中选取 k 个候选组合, 第二层从特定的 k 个组合中为每个向量选择最佳 e、f 组合.

对于含有大量异常值的数据, 为了避免压缩负增益, 改用偏斜字典压缩和位打包压缩的方法. 由于算法基于向量且需要进行二次采样来确定参数, 所以无法支持流式场景.

3.2 基于预测

3.2.1 FPZip

Lindstrom 等人[55]于 2006 年提出的这一流式无损浮点压缩算法. FPZip 的整体压缩步骤可以看成“预测-量化-编码”, 这与后面将会阐述的同样基于预测的有损压缩算法采用的框架大同小异.

在预测阶段, FPZip 采用了 Lorenzo 预测器[44], 通过加上或减去邻近的历史值来估计新值. 量化阶段分为映射和残差计算. 考虑到直接将原始浮点数和预测浮点数进行相减可能导致数据向下溢出, 无法实现无损恢复, FPZip 将原始值和预测值均映射到无符号的整数. 随后对计算得到的残差使用两级表示方案, 即根据前导零数量确定组号和组内的位置来表示残差大小的一种粗略度量. 之后对残差的具体值进行编码. 最后熵编码阶段, 使用了基于 Schindler 准静态概率模型的范围编码器[56]来压缩残差. 范围编码为残差分配概率, 并不断缩小表示这些残差序列的区间, 最后选择区间内任意一个值作为残差序列的编码.

FPZip 可以针对数据流进行处理, 但是需要数据达到一定数量时对残差进行编码, 无法以流的形式输出. 在本文中将其归纳为批式处理.

3.2.2 FPC

为了保证在具有竞争力的压缩率的同时提高压缩算法的吞吐量, Burtscher 等人[57]放弃了真实值与预测值的残差计算, 采用了异或操作. FPC 预测器采用了 FCM(固定上下文模型)[58]和 DFCM(差分固定上下文模型)[59]的组合, 能够通过 Hash 表快速预测序列中的下一个值. FCM 和 DFCM 各自生成不同的预测值, FPC 从中选择与实际值进行异或操作形成更多前导零的值, 并使用 1 位标记位 $pred_n$ 记录下所选择的预测器.

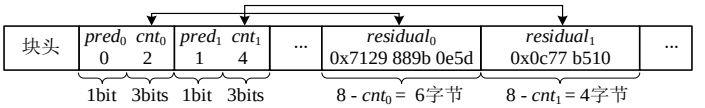


图 11 FPC 压缩块格式[57]

对于预测值和实际值的异或结果, 首先使用 3 比特位 cnt_n 来表示 0-7 个字节的前导零, 随后的残差位则直接记录为 $residual_n$. 压缩结果分块输出, 如图 11 所示, 每个分块都有一个块头来指示块内的浮点数目和各浮点数残差位的字节长度. 随后是 4 比特的编码, 即 $pred_n$ 和 cnt_n. FPC 面向字节编码, 将连续的两个 4 比特编码打包到一个字节中. 并且 4 比特编码和残差位是分开的, 而非连续的. 解压可以看作为压缩的逆过程, 根据压缩块读取指定字段, 然后零扩展为完整的 64 位, 根据位 $pred_n$ 与预测器进行异或操作可以无损恢复原始值.

FPC 在压缩时采用预测值和实际值进行异或处理, 可以支持流式处理, 但是为了加快解压效率可以采用数据块的形式对压缩后的数据以块的形式进行重构后输出.

3.3 基于机器学习

Yu[60]等人提出了批式的两级压缩方案选择模型, 即 AMMMO(Adaptive Multi-Model Middle-Out). 根据基本的数据压缩策略, 将压缩过程包括转换和差分编码两个阶段. 因此, AMMMO 设计了 6 种转换原语: 增量 $(v_i - v_{i-1})$、反向增量 $(v_{i-1} - v_i)$、异或 $(v_i \ xor \ v_{i-1})$、增量的增量 $((v_i - v_{i-1}) - (v_{i-1} - v_{i-2}))$、反向增量的增量 $((v_{i-1} - v_{i-2}) - (v_i - v_{i-1}))$、增量异或 $((v_i - v_{i-1}) \ xor \ (v_{i-1} - v_{i-2}))$, 和 3 种差分编码原语: 偏移、位掩码、尾随零, 以及 9 种控制参数, 并且总计有 331,776 种可能的控制参数组合. 这些原语和参数设置的选择是经验性的, 而且与作者的模型是正交的, 可以任意替换或扩展.

两级压缩方案中, 第一级为每一个时间分段选择方案空间, 包含多个候选的压缩方案, 为主模式; 第二级则独立地为每个数据点选择最佳编码方案, 为子模式. 由于过多的参数组合, 作者引入了强化学习与神经网络架构, 提出了 AMMMO_ML, 通过学习各种数据模式来自动执行模式选择和调整参数. 其中, 神经网络将块作为状态, 生成控制参数作为动作. 在此设置下, 子模式选择压缩率作为反馈, 以交互方式调整神经网络.

AMMMO_ML 需要针对数据进行压缩方案的选择, 需要将数据全部收集后才可以进行压缩, 因此无法支持流式数据压缩.

3.4 总结

无损压缩算法都具有对称性的特征, 即解压缩过程可以看作压缩的逆过程. 表 5 对上述无损压缩算法进行了总结, 并对是否支持流处理、支持不解压查询的特征做了归纳.

表 5 无损压缩算法特征比较

框架	算法	年份	流处理	不解压查询
基于数据表示	Gorilla[13]	2015	√	—
	Chimp[43]	2022	√	—
	$Chimp_{128}$[43]	2022	√	—
	Elf[12]	2023	√	—
	PDE[52]	2023	√	—
	ALP[42]	2023	—	√
基于预测	FPZip[55]	2006	—	—
	FPC[57]	2007	√	—
基于机器学习	AMMMO_ML[60]	2020	—	—

4 有损压缩

面对庞大的浮点时间序列数据, 有损压缩算法提供了一种平衡数据完整性和压缩率的方法. 与无损压缩不同, 有损压缩允许在压缩过程中丢弃部分信息, 以换取更好的压缩率和更快的处理效率. 有损浮点时序压缩算法分为四个主要方向: 基于数据表示的方法、基于预测的方法、基于机器学习的方法和基于变换的方法.

4.1 基于数字表示

4.1.1 Buff

Liu 等人[61]提出的 Buff 考虑小数点后的位数, 只保留尾数所需的比特数. Buff 在每种精度要求下进行实验验证, 以得到小数部分的最大所需比特数, 如图 12 所示. 但是这种实验验证的结果可能不是最佳方案, 理论上可舍弃的位数会更多. 且 Buff 若要实现无损压缩, 需要提前知道输入数据的最大精度.

小数位数	1	2	3	4	5	6	7	8	9	10
所需比特数	5	8	11	15	18	21	25	28	31	35

图 12 Buff 根据目标精度确定所需的尾数位数[61]

Buff 的压缩操作主要分为两个层次: 1) 基于给定的精度舍弃非有效位; 2) 提取整数和小数部分, 使用两种编码策略分别进行压缩. 如图 13 所示为 23.1415 的单精度浮点格式, 首先根据指数值和 4 位小数位数, 提取尾数位 (蓝色和橙色字段) 中的 4 位整数部分和 15 位小数部分, 再根据给定的数据范围, 将整数部分减去范围最小值 20, 以进一步压缩范围空间. 然后对整数部分使用 Delta 编码[13]和位打包编码, 对小数部分使用有界精度编码[61]. 并且采取固定长度编码的方式, 以实现部分解压查询.

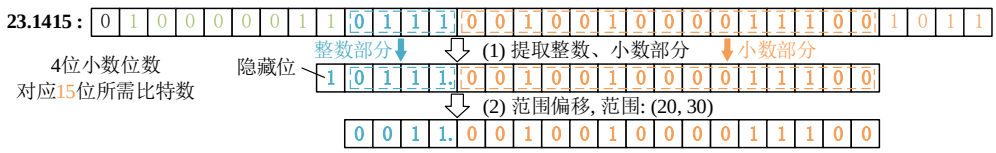


图 13 Buff 压缩示例[61]

由于需要对两个分量分别进行批式压缩, Buff 不适用于流式场景下的压缩操作. 若用户未提供输入数据的最大可能精度, Buff 需要较长时间去进行统计, 效率降低.

4.1.2 Real Information

Klöwer 等人[62]从信息论的视角定义了数据的位实信息(Bitwise Real Information, 以下简称 RI). RI 应用于网格化的二进制数据, 将邻近网格点的互信息(Mutual Information)[63]定义为逐位真实信息, 计算公式如下:

$$M(r,s)=\sum_{r=0}^{1}\sum_{s=0}^{1}p_{rs}\log_2\left(\frac{p_{rs}}{p_{r=r}p_{s=s}}\right) \tag{11}$$

其中 $r=r_1\ldots r_k\ldots r_l$ 和 $s=s_1\ldots s_k\ldots s_l$ 为比特流，$p_{r=r}$ 和 $p_{s=s}$ 是边缘概率，p_{rs} 是 $r_k=r$ 和 $s_k=s$ 同时成立的概率.

相邻网格点的 RI 计算如图 14 所示. 步骤 (1) 将结构化网格上的数据用矩阵表示, 矩阵中的相邻元素在网格中也相邻; 步骤 (2) 体现每个浮点数元素的 IEEE 754 格式; 步骤 (3) 将多个数的浮点格式转化成比特位置相同情况下的比特序列, 第一个矩阵由相邻网格点元素的符号位组成, 第二个矩阵由相邻网格点元素的第一个指数位组成, 以此类推; 步骤 (4) 和步骤 (5) 计算互信息 M, 即 RI; 步骤 (6) 保留包含 99% RI 的比特位, 其他比特位则被舍入为 0, 以便于后续的无损压缩.

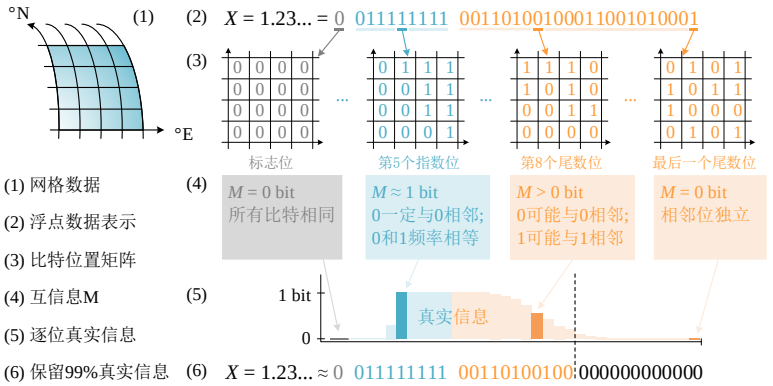


图 14 相邻网格点的逐位真实信息计算[62]

作者观察到大部分数据所包含的 RI 少于 7 位, 并且由于时空相关性而具有高度可压缩性. RI 算法通过保留大部分真实信息, 并结合无损压缩算法, 在气象数据上具有很好的压缩效果.

由于 RI 压缩需要批量处理相邻网格的数据, 因此无法针对流式数据进行压缩.

4.2 基于预测

4.2.1 SZ 系列

为了优化误差约束的高性能计算数据的压缩性能, Di 等人[21]提出了批式算法 SZ. SZ 采用了预处理-预测-量化-编码的框架. 首先对多维的快照数据进行线性化处理, 通过多维数组固有的内存序列来构建一维的数据. 采用三种曲线拟合模型, 包括前邻近拟合、线性曲线拟合和二次曲线拟合, 基于先前连续的重构值, 逐点选择误差最小的模型进行拟合. 如图 15 所示, 黑点表示当前数据值 v_i , 绿色、蓝色和红色方点分别表示当前数据值 v_i 的三个预测值, 都是由前一个(多个)连续解压值预测的. 图 15 中二次拟合曲线的预测值与实际值的残差满足用户定义的误差限, 且误差是三种模型中最小的, 故使用二次曲线模型对当前数据值 v_i 进行预测.

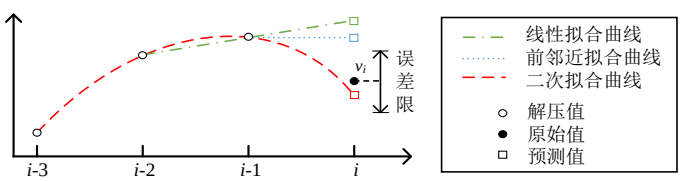


图 15 SZ 曲线拟合示例[21]

对于曲线拟合模型无法逼近的数据, 即不可预测数据, SZ 将其不重要的信息进行擦除. 首先进行批量归一化, 将所有无法预测的数据减去它们的中位数, 映射到较小的范围. 然后通过全局的数据范围和用户指定的预定义误差限来确定需要保留的浮点尾数位, 将不重要的尾数去除. 对于连续值可进行异或操作, 使用前导零计数和剩余有效位来进一步减少存储大小.

解压缩是上述算法的一个逆过程. 首先需要解析压缩数据, 包括压缩时记录的曲线拟合模型类型、预测的数据点、以及未能通过曲线拟合预测的数据点. 如果当前值是可预测的, 便通过相应的曲线拟合模型来进行恢复; 对于不可预测的点, 根据其前导零计数和异或结果进行恢复.

现有的工作对 SZ 的预处理器、预测器、量化器和编码器均进行了广泛的优化, 同时针对其在 GPU 上的执行也提出了改进版本. SZ1.4[64]采用多层预测模型, 同时提出了自适应误差控制量化和变长编码, 支持自定义定长量化区间的数目, 并对变长的量化系数进行 Huffman 编码, 可以处理变化剧烈的数据. SZ2.0[65]在一阶 Lorenzo 预测器的基础上, 提出了平均积分 Lorenzo 预测器和线性回归预测器以分别应对密集聚类数据和均匀分布数据, 基于均匀采样数据自适应选择最佳预测器. SZ(hybrid)[66] 结合了 SZ 和 ZFP 两种压缩算法, 在经过对角线采样之后, 经过评估选择其一, 并对 SZ 线性回归系数和 ZFP 变换后矩阵的压缩均进行了优化. SZauto[67]在一次 Lorenzo 和一次线性回归的基础上, 采用了二次 Lorenzo 和二次线性回归, 同时总结了 12 个关键参数, 支持离线与在线时评估选择. SZ3[24]指出线性回归预测的不足: 误差限越小, 系数开销越大, 并提出了 Lorenzo 预测器和插值预测器的自适应选择策略. 基于插值的预测器可以显著提高预测准确度, 尤其是对于平滑的数据集, 同时, 提供了线性样条插值、三次样条插值和多级多维插值的自适应选择. QoZ[68]在 SZ3 基础上, 提出了参数化多级插值预测器, 利用网格锚点缓解了长程插值导致的预测不准确问题, 并基于均匀采样分级选择最佳插值算法和动态调整误差限. SZ_ADT[25]使用 FSE 编码器来代替 SZ 的 Huffman 编码器, 并针对几何分布的量化因子提出了自适应编码表, 将小块压缩率和解压缩吞吐量分别提高了 5 倍和 2 倍. 同时提出了一种基于随机采样的压缩率预测机制, 使用 1%的小样本数据可以实现 91%的预测准确率. Machete[69]简化了预测器, 只采用前近邻预测, 编码方案混合了 Huffman 和变长量化(VLQ), 其中 VLQ 采用了 DP 策略, 并将 Huffman 树优化为规范树, 且 Huffman 解码表使用 Zstd 以加速解码. 专为 GPU 设计的 cuSZ[70], 利用 GPU 的并行处理能力, 显著提高了 SZ 算法的压缩效率. 其中预测量化阶段改进为双重量化框架, 避免了 RAW 依赖问题. 同时基于 CUDA 开发了一套高效的 GPU 优化 Huffman 编码策略, 进一步优化了 GPU 内存带宽利用.

上述 SZ 系列算法需要针对批量数据处理, 并且采用插值、熵编码等批处理方法, 故无法支持流式压缩.

4.2.2 Swing 与 Slide

Swing[71]和 Slide[71]均是基于角度的流式分段线性拟合 PLA 算法. 如图 16 所示, 原点设置为(t_1, v_1), 在添加 (t_2, v_2)的过程会产生一个角度, 该角度由分别连接 (t_1, v_1) 与 $(t_2, v_2 + \epsilon)$ 和 $(t_2, v_2 - \epsilon)$ 的上斜线 a_{u2} 和下斜线 a_{l2} 形成. 这个角度指定了在误差限 ϵ 范围内可能近似于这两点的所有直线. 下一个点 (t_3, v_3) 距离上斜线和下斜线都超过了 ϵ . 因此, 在添加这一点时需要减小角度, 使新的斜线 a_{u3} 和 a_{l3} 分别经过 $(t_3, v_3 + \epsilon)$ 和 $(t_3, v_3 - \epsilon)$. 最后, 添加 (t_4, v_4) 时需要进一步减小角度, 因为 (t_4, v_4) 距离上斜线 a_{u3} 超过误差限 ϵ. 因此, 连接 (t_1, v_1) 和 $(t_4, v_4 + \epsilon)$ 的斜线 a_{u4} 被设为新的上斜线. 由于 a_{l3} 距离 (t_4, v_4) 小于 ϵ , 因此不需要更新下斜线. 图 16 的蓝色区域为最终候选线, 如果后续数据与该区域的距离超过 ϵ, 则开始创建新的线段.

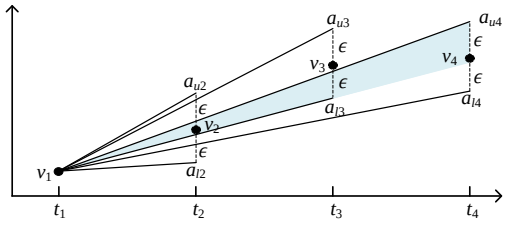


图 16 基于角度的 PLA[71]

Swing 采用了连接线段的方法, 即前一个分段的终点就是后一个分段的起点. 而 Slide 会生成不连接的线段, 起始点不受上一个段约束. 在 Slide 方法中, 不连接的线段虽然能减少分段数以提高压缩率, 但是可能导致更高的计算复杂度. 因此, Slide 延迟了线段终点的确定, 以尝试生成连接的线段. 最后需要根据最小均方误差的要求, 确定每一个分段的目标斜率, 因此 Swing 与 Slide 无法支持数据的流式压缩.

4.2.3 Mixed PLA

考虑到单独使用连接线段或不连接线段的方法可能无法实现最优压缩率, Luo 等人[72]提出采用了连接与不连接的混合方案. Mixed PLA 引入了拓展多边形, 定义 PLA 过程中可能的解空间, 并满足逐点误差要求, 同时还提出了“可见区域”和“闭合窗口”概念. 可见区域指从拓展多边形的某个窗口出发, 沿着 PLA 的可能路径所能够到达的所有点的集合. 闭合窗口是指可见区域没有到达最终窗口时, 将其包围的那个窗口. 这三个几何结构共同支撑了 Mixed PLA 的核心, 使得压缩算法能够在严格的误差限下压缩数据.

4.2.4 Sim-Piece

Kitsios 等人[73]通过分段匹配寻找最小分段组数, 进一步提高了 PLA 的压缩率和压缩质量. Sim-Piece 在 Swing 的基础上进行优化, 并采用不连接线段的方法. 其压缩过程包括: (1) 分段匹配. 通过量化每个分段起始点的数据值 v 为 b, 即 $b = \lfloor v / \epsilon \rfloor \times \epsilon$, 使得多个不同分段可以共享同一个量化值 b. 在基于角度的 PLA 过程中, 以量化值 b 作为下标构造间隔(Interval), 将 b 对应的分段以三元组(a_u, a_l, t)的形式添加到间隔中; (2) 分段合并. 同一个间隔中的分段可能有重叠, 即它们的斜率区间部分重合, 如图 17 (a) 所示. 通过对不同的重叠区间进行分组以最小化分组数量, 可以有效提高压缩率. Sim-Piece 使用间隔图(Interval Graph), 如图 17 (b) 所示, 将分段抽象为顶点, 两顶点之间的边表示两个分段重叠, 于是把最小化分组数问题转化求解最少完备子图数问题. 由于分段合并阶段建立在批式的分段匹配之上, 因此 Sim-Piece 不适用流式场景.

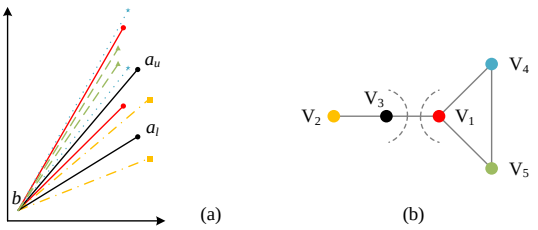


图 17 Sim-Piece 分段合并示意图[73]

4.2.5 MOST

Yang 等人[9]在 PLA 基础上加入了离群值检测, 以进一步减少分段数量. MOST 可以以微批式的方式应用在流式数据中, 主要分三步: (1) 离群值检测采用 SC(Shrinking Cone, 类似于 Swing 的基于角度的 PLA), 基于这种离群值检测方法下的算法压缩率要显著高于基于密度的离群值检测, 此外, 算法没有选用计算开销大的基于直方图和 ML 的离群值检测; (2) 分段拟合阶段也采用了 SC, 基于角度生成不连接的分段; (3) 编码阶段, 离群值使用线性缩放量化, 然后将其按 Zigzag[74]存储, 最后进行变长编码, 其斜率和截距分别使用 BFloat16[75]和单精度浮点格式存储, 并截断尾数位以实现压缩.

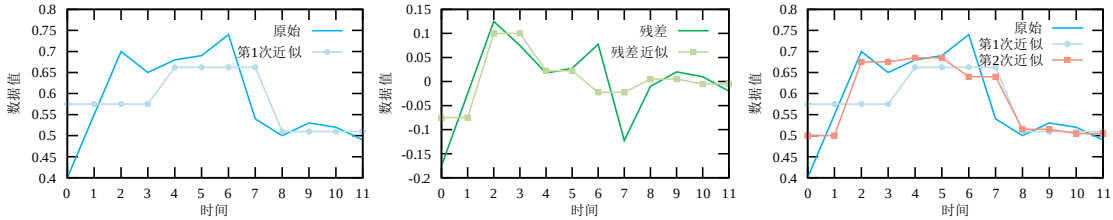
以上执行流程会执行两次 SC, 增加了计算开销, MOST 对其进行了优化, 通过只执行一次 SC 便完成了异常值检测和线段拟合. 由于 MOST 需要对离群值进行线性量化和变长编码, 因此无法实现完全的流式处理, 面对流式场景仅能采用微批式的方式进行压缩.

4.2.6 HIRE

Barbarioli 等人[76]针对下游应用不同的误差限要求, 提出了批式的多精度压缩算法. 其主要思想类似于泰勒多项式, 通过多级函数(残差)累加来不断逼近原数据. 如图 18 (a)所示是一个时序数据, 可以用 3 个常量线段来粗略地近似 (图 18 (a)浅蓝色圆点线段). 近似值和原始序列之间有一个残差序列(图 18 (b)深绿色线段). 这个残差序列可以通过另一个 6 段的常数分段来近似表示(图 18 (b)浅绿色方点线段). 每一次近似都会将之前的残差近似序列累加, 如图 18 (c)红色方点线段所示, 显然第二次近似更接近原始序列. 只要随后的每次近似都包含更多信息(即增加分段数), 剩余残差序列的幅度就会减小, 近似值的总和会越来越接近原始值.

基于上述思想, 针对不同精度要求的下游应用, 边缘服务器只需要执行一次压缩, 然后根据精度要求发送相应层级的近似序列. HIRE 基于分治方法采用了二叉树结构, 逐层级对每一个结点都进行池化、量化和残差计算, 直到满足条件为止; 解压缩是压缩的逆过程, 其中池化的反向过程中, 基于复制插值的方法对池化值进

行采样. 由于 HIRE 每次精度迭代的时候, 都需要遍历完整的残差序列, 因此无法流式地处理.



(a) 原始时间序列与第1次近似序列 (b) 残差序列与残差近似序列 (c) 重构数据

图 18 HIRE 分层残差编码示例[76]

4.3 基于机器学习

由于机器学习的算法需要收集一定的数据进行训练才能得到较好的压缩率, 只能采用批式压缩的方法来进行处理, 因此本节不对算法的流批处理进行单独讨论.

4.3.1 LFZip

LFZip[23]是一种误差约束的适用于多时序的有损浮点压缩算法, 类似于 SZ, 也基于预测-量化-熵编码框架. LFZip 的预测器主要使用了线性模型和神经网络模型. 预测阶段默认采用归一化最小均方误差预测器, 因其参数少、不需要预训练, 故执行效率高; 为进一步提高压缩率, LFZip 还采用了基于神经网络的预测器, 如使用全连接网络, 先进行离线训练, 并在输入数据中加入均匀分布的噪声以避免误差限过大, 压缩过程中还可以在线更新参数.预测残差采用均匀标量量化, 使用 16 比特来表示 65,535 个量化的范围, 并单独存储异常值. 最后在熵编码阶段将量化后的时间序列数据通过通用无损压缩器 BSC[77]进一步压缩.

在 LFZip 的基础上, STBFC[78]算法对基于深度学习的预测器进行了优化, 采用 CNN 提取时间特征、图像分类器提取空间特征, 并集成自回归线性模型来增强模型的鲁棒性.

4.3.2 NUMARCK

Chen 等人[27]基于传统的机器学习方法提出了 NUMARCK, 其基本思想是通过捕捉时间变化来学习数据每次迭代后的新分布, 从而逼近检查点, 在保证逐点误差范围的前提下实现压缩. NUMARCK 主要分两步: (1) 计算相对变化率 $\Delta D_{i,j} = (D_{i,j} - D_{i-1,j}) / D_{i-1,j}$, 其中, $D_{i,j}$ 和 $D_{i-1,j}$ 分别是第 j 个数据点在第 i 次和第 i-1 次迭代中的值. 通过计算相对变化率, 可以将单个快照中很少出现重复模式的数据, 转化为变化百分比空间, 更容易挖掘共同的模式; (2) 通过并行 K-means 学习数据的分布以逼近原始数据. K-means 聚类的一个内在局限是, 初始聚类中心点的选择已被证明会极大地影响算法的性能和结果的质量. 为了克服这一局限, NUMARCK 利用等宽直方图的先验知识来初始化 K-means 的聚类中心点, 从而获得更可靠的分割结果.

4.3.3 Rl4tsc

Jiang 等人[79]提出的 Rl4tsc 使用基于深度学习的强化学习模型 DQN 来实现时序数据的有损压缩.

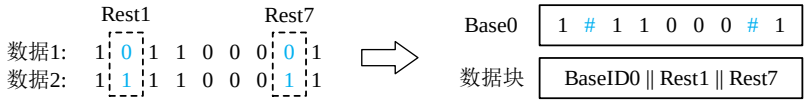


图 19 RI4tsc 存储格式[79]

考虑到同一数据源的时序数据之间有很强的相似性, 例如图 19 中的 101100001_2 和 111100011_2, 它们共享 7 个比特. 因此, 可以把不同数据中的共同部分提取出来作为 Base, 并附上索引 BaseID, 各数据中除 Base 外的其他部分作为 Rest, 最后存储为“BaseID0 || Rest1 || Rest7”格式的数据块. 其中, Base 是唯一值. 由此可以实现较好的压缩率. 将数据块分为 Base 部分和 Rest 部分, 并将所有 Base 部分单独存储, 该方法能够在不解压整个数据块的情况下直接对压缩数据进行查询.

为实现更好的压缩率, 要为不同的数据块分区确定合适的 Base. 由于缺少带标记的训练样本[60], Rl4tsc 采

用强化学习, 为时序数据训练合适的模型, 将压缩问题转化为寻找最佳 Base 的优化问题. 如图 20 所示, 先根据相应的规则将时序数据划分为合适长度的数据块, 再提取数据块和 Bases 的特征并转化为特征向量, 然后使用 DQN 强化学习网络进行 Base 选择, 按存储空间的优化计算奖励反馈, 以此不断更新模型.

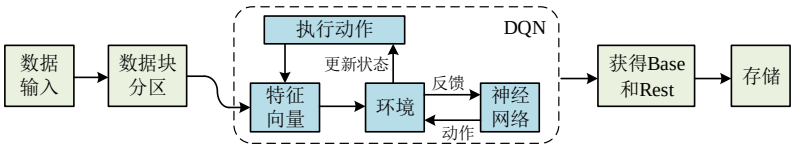


图 20 RI4tst 数据压缩框架[79]

4.4 基于变换

基于变换的有损压缩算法通常是基于向量执行的, 多为批式的压缩算法. 因此本节不对算法的流批处理进行单独讨论.

4.4.1 ZFP

Lindstrom[28]提出了固定压缩率的压缩算法 ZFP, 同时支持以 4×4×4 的块粒度对 3 维浮点数组进行随机读写访问. ZFP 主要步骤为: (1) 量化, 将块中的浮点数转换为定点数, 如[-8,+8)范围的数据可以用 Q3.6(即 3 比特表示整数部分, 6 比特表示小数部分)定点二进制补码形式表示, 然后通过指数对齐把数据归一化到(-1, 1)范围, 块中所有浮点数共享一个指数, 又称为块浮点存储; (2) 正交块变换, 提出统一形式的正交矩阵, 定义了可分离的基函数, 并按 3 维的类 Zigzag 顺序排序, 执行变换过程中通过提升框架[80]、消除符号变化、参数值自适应选择、移位等操作来进行优化; (3) 嵌入式编码, 基于比特平面(bit plane). 如图 21 所示, 比特平面是变换系数的尾数矩阵中同一列的比特集合. 使用分组测试(group testing)以定位有效位(即 1), 并写入相应比特流中. 此外, ZFP 还采用了软件缓存的策略, 仅对从缓存块中调出的被修改块来进行修改, 以提高压缩效率.

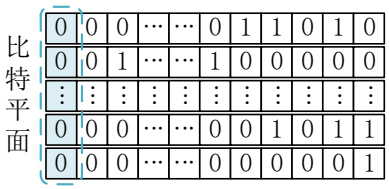


图 21 比特平面示意图

现有的相关工作对 ZFP 的嵌入式编码方式做了相关调整, 并基于 FPGA 的算法实现加速. DE-ZFP[81]是基于 FPGA 的 ZFP 算法实现, 使用基于字典的编码替代 ZFP 原始的嵌入式编码, 在保持高吞吐量的同时将压缩率损失控制在原始 ZFP 实现的 4%-13%之内, 以最大化硬件性能. ZFP-X[82]则针对嵌入式编码进行了优化, 采用后端嵌入式编码, 用比特群组(bit group)代替了比特平面, 通过检测行阶梯, 改进了编码效率和压缩率.

4.4.2 TTHRESH

Ballester-Ripoll 等人[83]提出了基于 Tucker 分解[47]的算法 TTHRESH, 其高阶奇异值分解的转换系数包括核心系数和因子系数, 均采用位平面编码, 然后通过 RLE 和算数编码进一步压缩. 但奇异值分解的计算开销较大, 可能会导致较的压缩速度. 此外, TTHRESH 在解压缩时需要遍历整个核心, 因此不支持随机访问.

4.4.3 SSEM

Sasaki 等人[84]提出了基于 Haar 小波变换的有损算法 SSEM, 旨在提高气候模拟的整体检查点/重启性能. 作者指出基于小波变换的压缩方案在处理平滑数据时是非常有效的, 而压力、温度、速度等物理量在空间上又是平滑的, 即没有显著变化, 因此将小波变换应用于有损压缩. SSEM 主要步骤为: (1) Haar 小波变换. 若原始检查点数据是 1 维数组 A, 则将其变换为两个子波段数组, 即低频波段数组 L 和高频波段数组 H, 其中

$$L[i]=\frac{A[2i]+A[2i+1]}{2}(i=0,\ldots,n)$$
$$H[i]=\frac{A[2i]-A[2i+1]}{2}(i=0,\ldots,n) \tag{12}$$

对于 2 维数组, 则依次按行、列分别执行 1 维小波变换, 3 维同理; (2) 量化, 将高频数组元素按值的大小均分成 d 个区域, 对于数值密集区再均分成 n 个区域, 然后单独对 n 个区域内的元素求均值, 得到数组 $average[i](i=0,\ldots,n\text{-}1)$; (3) 编码, 对于量化过的元素值, 使用数组 *average* 的下标代替, 以 char 类型进行存储; (4) 格式化, 使用位图 *bitmap* 以判断元素值是否经过量化、编码, 未进行量化、编码的值以原始浮点值进行存储. 最后再执行 Gzip 算法以进一步提高压缩率.

4.4.4 MGARD+

Liang 等人[29]基于应用数学的 MGARD(MultiGrid Adaptive Reduction of Data)[85]方法, 针对误差有界的科学数据优化了渐进式压缩方案. MGARD 利用小波变换和 L2 投影进行数据去相关, 然后对系数进行线性缩放量化以及变长编码. MGARD+在 MGARD 基础上, 结合了分层量化方法和自适应分解策略, 其中分层量化通过为不同级别的系数设置不同的误差限, 从而更加精细地控制压缩过程, 自适应分解则基于 Lorenzo 预测器和分段多线性插值动态确定分解的最佳层级, 减少不必要的计算和存储开销. 为了提高压缩性能与质量, MGARD+引入了一系列优化算法, 如数据重排、直接负载向量计算、批量校正计算以及中间变量的优化利用, 大幅度提升了多层分解和重构的性能.

4.4.5 SPERR

Li 等人[30]在大量可用的小波中选择了 CDF 9/7 双正交小波变换[45], 并通过信号处理工具 QccPack[80]实现了 CDF 9/7 变换的性能提升. 小波系数通过优化集合分裂嵌入块编码算法(SPECK)[46]进行编码, 关键步骤包括: (1) 集合划分, 将经过小波变换的数据集划分为不同的子集, 并组织成树状结构; (2) 重要性排序, 遍历这些子集, 根据设定的阈值评估每个数据点的重要性, 并将其分类. 随着编码过程的进行, 阈值逐渐降低; (3) 细化过程, 对于被标记为重要的数据点, 对其进行逐次逼近量化, 以确保在压缩过程中保留足够的信息.

SPECK 编码的逐位平面方式只要达到用户规定的输出大小, 编码过程就会终止, 和 ZFP 一样可以实现固定大小的压缩. 针对不满足用户定义逐点误差限的异常值, SPERR 设计了一种单独的编码过程, 与小波系数的编码完全分开. 其主要步骤也分为排序过程和细化过程, 并且阈值也动态降低. 最终输出面向字节的编码.

4.5 总结

和无损压缩一样, 有损压缩算法也都具有压缩和解压缩过程的对称性特征. 表 6 对上述有损压缩算法进行了总结. 值得注意的是, 有损压缩通常会在压缩过程中逐点地进行最大误差限, 这样可以有效地降低数据失真程度. 因此, 相比无损压缩, 有损压缩会考虑误差约束的特征.

表 6 有损压缩算法特征比较

框架	算法	年份	流处理	误差约束	不解压查询
基于数据表示	Buff[61]	2021	—	—	√
	Real Information[62]	2021	—	—	—
基于预测	Swing[71]	2009	—	√	—
	Slide[71]	2009	—	√	—
	Mixed PLA[72]	2015	—	√	—
	SZ[21]	2016	—	√	—
	SZ3[24]	2021	—	√	—
	SZ_ADT[25]	2023	—	√	—
	Sim-Piece[73]	2023	—	√	—
	HIRE[76]	2023	—	√	—
	MOST[9]	2024	—	√	—
	Machete[69]	2024	—	√	—

基于机器学习	LFZip[23]	2020	—	√	—
	NUMARCK[27]	2014	—	√	—
	Rl4tsc[79]	2023	—	—	√
基于变换	ZFP[28]	2014	—	—	—
	SSEM[84]	2015	—	—	—
	TTHRESH[83]	2019	—	—	—
	MGARD+[29]	2021	—	—	—
	SPERR[30]	2023	—	√	—

5 脉络分析

本节整理了部分代表性系列的算法, 列出了它们的主要发展脉络, 如图 22 所示, 主要包括无损压缩算法中的 XOR 系列和 PDE 系列, 以及有损压缩算法中的 PLA 系列、SZ 系列和 ZFP 系列. 其中, 实线箭头表示, 所指算法在之前算法的基础上进行改进实现了较大的提升, 或者体现着类似的思想并在时间上有承接关系.

无损浮点时序数据压缩算法通常采用异或操作, 如基于预测的 FPC 和基于二进制的 Gorilla、Chimp 和 Elf. 特别地, XOR 系列中, 基于二进制的算法在发展过程中, 不仅提出了新颖的预处理策略, 而且也对编码策略进行了优化, 压缩率方面获得了显著的提升.

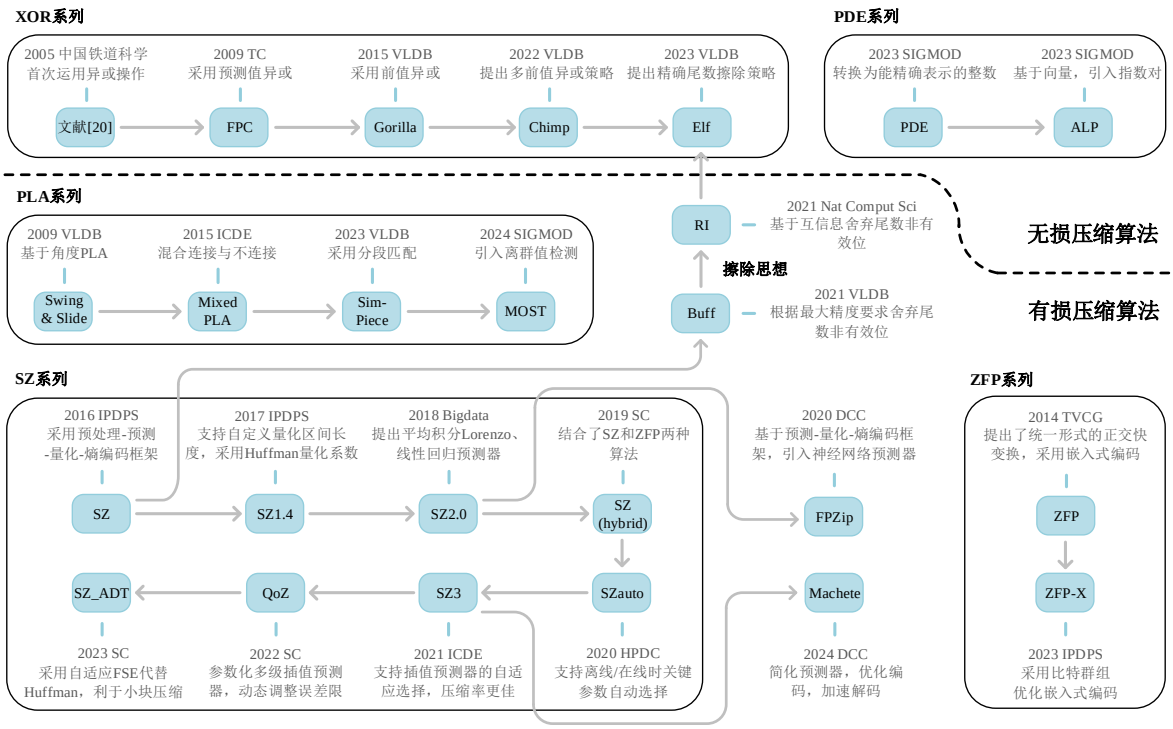


图 22 浮点时序数据压缩算法的发展脉络图

针对基于 PLA 的有损浮点时序数据压缩, 在过去十多年中, 新颖的算法也在不断涌现, 其主要优化方向为不断减少分段数, 同时保持较高的压缩效率.

SZ 算法自诞生之初, 就在有损浮点数据压缩领域占有一席之地. 无论是在多时序科学数据集还是常见的单时序数据集上, SZ 都有着不错的压缩表现. 近些年来的工作对 SZ 的预处理-预测-量化-编码阶段中的一个或多个进行了广泛的优化, 更进一步完善了算法的压缩效果和效率. 另有一些团队的算法, 如 FPZip 和 Machete, 分别在 SZ2.0 和 SZ3 的基础上做了相应的优化, 更好得契合特定应用场景.

有一条擦除思想的分支贯穿了无损压缩算法和有损压缩算法, 主要表现为 SZ、Buff 和 RI 分别舍弃了不可拟合点的、最大精度要求的、基于互信息的尾数非有效位, 以及 Elf 精确计算出尾数的擦除位并将其后的比特全部转换为尾随零. 还有一些脉络分支, 如 PDE 系列和 ZFP 系列, 由于算法较新颖或者底层原理的创新性, 留给了研究人员未来有很大的优化机会.

6 实验分析

本节对多种代表性算法展开实验对比分析, 依次介绍了用到的数据集、实验设置和实验结果分析.

6.1 数据集

为了对比不同的压缩算法, 我们选用了表 7 中的 12 个单时序数据集. 此外为了清晰地对比不同的方法, 我们整理汇总了 2 个多时序数据集在部分有损压缩算法中的对比结果, 数据集信息如表 8 所示.

表 7 单时序数据集

数据集	有效位数	数据个数	大小 (KB)	描述
Air-pressure (AP)[86]	6~7	95928	937	校正后海平面和地表的气压
Air-sensor (AS)[87]	12~17	8664	165	空气传感器数据
Basel-temp (BT)[88]	7~9	100001	1060	巴塞尔过去的温度数据
Basel-wind (BW)[88]	0~9	100000	1010	巴塞尔过去的风速数据
Bird-migration (BM)[87]	2~7	89867	829	在线动物追踪的数据集
City-temp (CT)[89]	3~4	100001	556	全球主要城市的温度
Dewpoint-temp (DT)[90]	3~4	100001	684	河流和湖泊上漂浮的传感器观测到的温度
IR-bio-temp (IR)[91]	0~4	99602	670	红外生物温度的变化情况
PM10-dust (PM10)[92]	1~4	93662	651	大气中 PM10 颗粒物的实时测量数据
Stocks-DE (SDE)[93]	2~6	100002	716	记录了德国的股票价格变化
Stocks-UK (SUK)[93]	4~6	100002	704	记录了英国的股票价格变化
Stocks-USA (SUSA)[93]	3~5	100002	693	记录了美国的股票价格变化
Wind-Speed (WS) [94]	1~3	99132	571	记录了风速变化情况

表 8 多时序数据集[24]

数据集	大小	维度	浮点类型	描述
Miranda	1.97GB	256×384×384	Double	湍流
QMCPack	601MB	288×115×69×69	Float	量子结构

6.2 实验设置

本文选择开源的具有代表性的无损压缩算法 Gorilla、Chimp$_{128}$、Elf、ALP、FPC, 在单时序数据集上进行实验. 同时针对部分有损压缩算法, SZ2、Machete、Sim-piece、Buff 和 SZ_ADT, 使用单时序数据集进行实验. 此外我们还汇总了 SZ2.1、SZ3、ZFP、MGARD+在多时序数据集的实验结果. 部分算法, 例如基于机器学习的压缩算法, 由于其开源代码较少, 数据集不统一参数难以确定, 我们暂未进行实验分析. 我们开源了实验的代码和相关数据集, 链接如下: https://github.com/Spatio-Temporal-Lab/FloatingCompressionSurvey.

我们通过压缩率、压缩时间和解压缩时间三个指标来验证了各种方法的性能. 对于批式算法我们设置每块 1000 个数据压缩进行实验. 对于有损压缩算法, 我们设置绝对误差限为 1E-2、1E-3、1E-4 三个参数进行实验. 其中压缩和解压时间我们统计其 1,000 个数据的压缩和解压时间的平均值. 汇总的多时序数据集实验结果中, 误差限所采用的是范围误差.

本文实验是在一台配备了 Windows 11, 第 11 代英特尔(R)核心(TM) i5-11400@2.60 GHz CPU 和 32GB 内存的个人电脑上进行的, 采用 C++语言, 其中 gcc 版本 13.2.0, cmake 版本 3.28.6.

6.3 实验结果分析

6.3.1 无损压缩算法

表 9 无损压缩算法实验结果

度量指标	Gorilla	$Chimp_{128}$	Elf	FPC	ALP
压缩率	0.763	0.397	0.310	0.744	**0.285**
压缩时间(μs)	**14.09**	15.74	47.61	18.13	41.08
解压时间(μs)	20.76	20.08	**14.85**	19.44	17.50

注: 加粗部分表示最佳性能, 下划线部分表示次优性能, 下同

本文对无损压缩算法进行实验, 实验结果如表 9 所示. 从表中可以看出 ALP、Elf 算法在浮点类型数据的压缩率有着相当显著的优势. 由于 Elf 在压缩时需要进行精度的计算, 导致其压缩速度相比于简单的 Gorilla、$Chimp_{128}$ 等算法较为逊色; 而 ALP 算法在压缩时采用二级采样的方法来确定压缩参数, 因此压缩速度稍慢. 但是二者在压缩率和解压速度上有着更为领先的水平. Gorilla 和 $Chimp_{128}$ 算法压缩时只需进行简单的 XOR 算法以及一些标志位的判断, 因此二者压缩速度较快.

6.3.2 有损压缩算法

本文对有损压缩算法在不同的误差限进行实验, 实验结果如表 10 所示. 最新 Machete 算法压缩率最为出色, 同时为了加速压缩和解压速度, 算法加入了 Zstd 的解码表. Buff 在给出精度与元数据精度一致的情况下可以做到无损压缩. 但当给出精度不一致时, 压缩率相较于其他专注于有损压缩的算法稍差. Buff 针对压缩后的数据提供了一系列高效查询的方案, 其在解压和查询时更加有优势. SZ_ADT 所使用的自适应 FSE 更适用于小文件压缩, 相比与 SZ2, 在不降低压缩效果的情况下, 显著提高了压缩和解压效率.

表 10 有损压缩算法在单时序数据集实验结果

	压缩率			压缩时间(μs)			解压时间(μs)		
误差限	1E-4	1E-3	1E-2	1E-4	1E-3	1E-2	1E-4	1E-3	1E-2
Buff	0.352	0.289	0.242	**17.96**	**17.79**	**16.93**	16.76	15.80	13.91
Machete	**0.148**	**0.119**	0.087	80.01	66.44	52.48	**7.27**	**6.74**	**5.64**
Sim-Piece	0.291	0.256	0.194	315.38	294.16	238.51	40.68	38.23	30.78
SZ2	0.307	0.235	0.168	422.40	145.11	119.89	27.83	26.56	28.45
SZ_ADT	0.205	0.142	**0.084**	53.01	43.66	37.55	19.05	14.47	9.46

本文同样汇总了部分代表性有损压缩算法的实验结果. 为了结果的一致性, 实验使用表 8 所示的科学数据集. 由于数据集维度的不同, 这里压缩和解压时间我们标准化为每 1KB 数据集下的压缩和解压的时间. 如表 11 所示, SZ3 采用自适应策略有效地减小系数开销问题, 在不同误差限精度下有着最佳的压缩率, 而且所采用的插值预测器使其预测得更加准确. ZFP 的变换机制加上软件缓存策略, 使其压缩和解压效率相对更好.

表 11 有损压缩算法在多时序数据集实验结果[24]

	压缩率			压缩时间(μs)	解压时间(μs)
误差限	1E-4	1E-3	1E-2	1E-3	1E-3
SZ2.1	0.0428	0.0181	0.0065	8.19	6.85
SZ3	**0.0185**	**0.0056**	**0.0016**	8.00	7.52
MGARD+	0.0586	0.0238	0.0075	7.31	7.35
ZFP	0.0826	0.0431	0.0234	**5.10**	**6.33**

7 应用

数据压缩的主要目的是为了加快数据传输和减少数据存储. 本节讨论这两种场景下时序压缩算法的应用.

7.1 传输场景应用

在传输场景中, 浮点数据的压缩技术能够显著减少传输数据量, 提高传输效率, 降低传输延迟, 同时确保数据的完整性和准确性. 浮点数据传输是许多现代应用的核心, 涉及到高频、海量数据的实时流式传输和批

量传输. 通过使用适当的压缩算法, 可以优化这些传输过程, 达到更高的性能和更低的成本.

在实时传输场景中, 数据是不断生成的, 需要对其进行实时响应和处理, 因此流式压缩算法可以胜任这种场景. 例如在物联网 (IoT) 应用中, 成千上万的传感器设备会持续生成大量时序数据. 实时传输这些数据对网络带宽和设备处理能力提出了很高的要求. 在金融交易监控、工业设备监控等系统中, 数据需要实时传输和处理. Gorilla[13]、Chimp[43]和 Elf[12]等轻量流式压缩算法能够在低资源消耗的情况下实现高效的实时数据处理, 确保物联网系统的高效运行. 在分布式计算中, 各个节点中的数据传输同样采用流式的压缩算法进行加速.

在大规模数据集的传输中, 例如数据中心之间进行数据迁移和同步、数据备份和恢复等应用, 高压缩率的压缩技术能够保证大量数据在传输过程中保持完整性和准确性, 同时降低网络负载. 因此高效的批式压缩算法在处理大规模数据集的传输任务中成为最佳选择, 满足了现代数据管理和传输的高效需求.

7.2 存储场景应用

在存储场景中, 压缩算法的主要目标是减少数据的存储需求, 提高数据存取效率, 同时保持数据的关键特征和质量. 通过将数据压缩成更小的表示形式, 可以显著节省存储空间, 降低硬件成本. 无损时序压缩算法能确保数据完整性, 适用于高精度数据存储, 而有损时序压缩则适用于多媒体数据. 总体而言, 压缩算法优化了存储资源利用, 提高了系统性能和成本效益.

在实时数据库中, 流式浮点类型压缩算法通过高效的实时压缩, 显著提升数据处理性能, 特别适用于高频率数据写入的场景. 这些算法能够快速压缩和解压缩数据, 减少存储需求并提高查询速度, 从而确保系统能够及时响应和处理大量数据, 满足现代数据密集型应用的需求.

在大数据存储场景中, 高压缩率是核心需求, 例如在数据仓库中, 通过先进的压缩技术可以显著减少存储需求, 并提高查询性能. 这些压缩技术不仅减少了存储空间的占用, 还支持不解压查询, 使得在数据压缩状态下直接进行查询操作, 大大提升了数据处理效率. 此外, 在科学数据存储中, 使用高效的压缩方法可以在保证数据精度的同时, 实现高效的批量压缩, 进一步优化存储空间和处理性能, 从而满足大规模数据集的存储需求.

8 展望

随着时序数据爆发式的增长和 IoT 行业蓬勃的发展, 浮点时序压缩将扮演越来越重要的角色. 本文从多维时序数据压缩、不解压查询与应用、软硬件结合、特殊浮点型压缩四个角度展望未来的研究方向.

8.1 多维时序数据压缩

真实世界中的时间序列通常为多维数据, 这些数据由传感器共同采集和处理. 例如, 自动驾驶汽车中的惯性测量单元(Inertial Measurement Unit, IMU) 每个时间戳会同时采集陀螺仪、加速度和磁力计的数据.

现有的支持多维浮点时序压缩的算法, 如 SZ 利用数据的逻辑存储结构, 将多维浮点数组映射到一维索引上, ZFP 对多维数据块采取可分离变换, 沿着每个维度进行一维变换.本质上, 这些算法仅在单一维度上进行操作, 没有利用到多维时序数据的内在特征. 例如, 股市交易时股票的成交量、价格、换手率等序列之间可能表现出统计上的相关性, 可以是线性的或非线性的. 利用这一相关性可以挖掘潜在的压缩空间. 未来工作的方向之一可能涉及更深入地探讨和利用多维时序数据中不同维度之间的相关性, 以进一步改善数据压缩的效果.

8.2 不解压查询与应用

大数据场景中的日志文件、科学计算等数据需要频繁访问, 使用不解压或部分解压查询可以显著降低数据处理的时间和空间复杂度, 提高查询效率. 算法 Buff 面向字节压缩, 通过查询重写将原始查询操作分解成子列查询的组合, 从而实现部分解压查询. RI4tsc 通过建立索引以很小的空间开销来获得高效的不解压查询. 除了上述设计查询算法和索引机制, 算法还可以将数据分块压缩, 并允许对单个块进行解压和查询, 而不影响整体数据集. 但是现有的不解压或部分解压查询通常只支持简单的查询操作, 如指定行查询、条件查询, 对于较复杂的聚合查询、范围查询等查询操作的高效实现还有进一步优化空间.

压缩数据不但可以用于快速数据检索, 还能作为机器学习任务的输入, 直接用于数据挖掘. 这通常涉及到

一些特定的技术, 如特征提取、近似计算或者通过压缩感知技术直接在压缩域进行数据分析. 这些技术使得算法可以从压缩数据中直接提取有用的信息, 进行训练和预测, 无需恢复到原始数据. 然而, 如何在不牺牲精确度的前提下, 从压缩数据中有效地提取特征和进行高质量的数据挖掘, 仍是一个值得进一步研究的问题.

8.3 软硬件结合

现有的软件层面解决方案在数据压缩领域已取得一定成就, 但它们在处理器性能和能效方面的限制, 尤其是在需求实时处理的场合, 已成为提高压缩效率和降低能耗的瓶颈. 通过利用 GPU(图形处理单元) 、ASIC(应用特定集成电路) 以及新兴的非易失存储器(NVM) 技术, 硬件加速方法提供了一种显著提高数据压缩和解压缩效率的可能性[95]. 相关研究通过广泛实验发现, 基于 GPU 的无损压缩算法, 相比基于 CPU 的方法, 在吞吐量方面最高可以提升 350 多倍[96].

虽然已有研究开始探讨硬件加速的潜力, 但在如何充分利用这些硬件技术以优化浮点类型时序数据压缩的研究还相对较少. 此外, 新型存储介质如 SSD 和 HDD[97]已经开始内置数据压缩功能以提高存储密度和效率, 然而, 针对这些存储介质的优化压缩算法的研究仍需深入. 近数据处理(NDP) [98]的概念, 即将计算任务转移到底层存储设备以提高数据压缩算法效率的策略, 为存储层面的数据处理提供了新的方向, 但如何在实践中实现高效的 NDP 仍然是一个挑战.

8.4 特殊浮点型压缩

在深度神经网络(DNN)的推动下, 人工智能在图像识别、自动驾驶等领域取得最新进展[99]. 但是 DNN 模型对计算资源的需求也日益增长. 通过降低浮点数据的精度, 比如在 DNN 模型中使用半精度浮点型, 可以显著提高计算速度. 其中, Float16 是 IEEE 754 标准中的半精度浮点型, 指数部分和尾数部分分别为 5 位和 10 位, 和单精度浮点型格式之间有较大的转换开销, 由此引入另一种半精度浮点格式 BFloat16[75], 其指数部分为 8 位, 可以通过直接截断单精度浮点数的后 16 位来实现高效转换. BFloat16 格式的浮点数已经成为深度学习领域事实上的标准, 已有一些深度学习“加速器”支持 BFloat16, 如谷歌的 TPU、英伟达的 GPU 等. 将浮点压缩算法与 BFloat16 格式结合起来, 利用深度学习模型中矩阵和张量的高度稀疏性, 确保压缩后的数据满足应用的精度要求, 以减少数据传输和存储的需求, 这对于加速深度学习模型的训练和推理有广阔的应用前景.

9 总 结

本文对近年来具有代表性的浮点时序压缩算法进行了广泛的归纳和总结. 本文首先介绍了浮点时序压缩的相关概念和研究现状, 然后从无损压缩和有损压缩两大视角出发, 根据算法框架将浮点时序压缩算法分为四大类: (1) 基于数据表示; (2) 基于预测; (3) 基于变换; (4) 基于机器学习. 针对每一类压缩算法, 介绍了它们的主要思想和常用处理方法, 重点对具体压缩算法的原理和压缩流程进行阐述, 并总结流式压缩与批式压缩特征. 本文还梳理了算法的发展脉络, 同时通过实验对比了主流浮点时序压缩算法的性能. 最后对浮点时序数据压缩算法的应用及展望进行了讨论.

References:

[1] Ruan SJ, Xiong KQ, Wang SL, Geng J, Bao J, Zheng Y. A Survey of Urban Geographic Information Inference Driven by Crowd-Sourced Spatio-Temporal Data. Acta Electronica Sinica. 2023, (8): 2238-2259. [doi: 10.12263/DZXB.20230131]

[2] Yu ZS, Li RY, Guo Y, Jiang ZY, Bao J, Zheng Y. Distributed Time Series Similarity Search Method Based on Key-value Data Stores. Ruan Jian Xue Bao/Journal of Software, 2021, 33(3): 950–967 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6445.htm [doi: 10.13328/j.cnki.jos.006445]

[3] Zhan XY, Xu HR, Zhang Y, Zhu XY, Yin HL, Zheng Y. Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning. In Proc. of the AAAI Conf. on Artificial Intelligence, 2022, 36: 4680–4688.

[4] Li RY, He HJ, Wang RB, Ruan SJ, He TF, Bao J, Zhang JB, Hong L, Zheng Y. Trajmesa: A distributed nosql-based trajectory data management system. IEEE Trans. on Knowledge and Data Engineering, 2021, 35(1): 1013-1027.

[5] Zuo YM, Lin XL, Ma S, Jiang JH. Road network aware online trajectory compression. Ruan Jian Xue Bao/Journal of Software, 2018, 29(3): 734−755 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5438.htm [doi: 10.13328/j.cnki.jos.005438]

[6] Ruan SJ, Xiong Z, Long C, Chen YH, Bao J, He TF, Li RY, Wu SN, Jiang ZY, Zheng Y. Doing in one go: delivery time inference based on couriers' trajectories. In: Proc. of the 26th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining, 2020, 2813-2821.

[7] Dong HW, Zhang C, Li GL, Feng JH. Survey on Cloud-native Databases. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 899–926 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6952.htm [doi: 10.13328/j.cnki.jos.006952]

[8] Zhou BY, Chen CY, Wang Q, Zhou FC. Publicly Verifiable Outsourced Database with Full Delegation. Ruan Jian Xue Bao/Journal of Software, (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6129.htm [doi: 10.13328/j.cnki.jos.006129]

[9] Yang ZH, Chen SM. MOST: Model-Based Compression with Outlier Storage for Time Series Data. Proc. ACM Manag. Data, 2023, 1(4): 250:1-250:29.

[10] He WD, Xia TR, Song SX, Huang XD, Wang JM. Multimodal Data Encoding and Compression in Apache IoTDB. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1173-1193 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7077.htm [doi: 10.13328/j.cnki.jos.007077]

[11] Deng AH, Qiao L, Yang MF. Double Index Data Compression Method for Onboard Computer. Ruan Jian Xue Bao/Journal of Software, 2022, 33(10): 3844-3857 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6308.htm [doi: 10.13328/j.cnki.jos.006308]

[12] Li RY, Li Z, Wu Y, Chen C, Zheng Y. Elf: Erasing-Based Lossless Floating-Point Compression. Proc. VLDB Endow., 2023, 16(7): 1763–1776.

[13] Pelkonen T, Franklin S, Teller J, Cavallaro P, Huang Q, Meza J, Veeraraghavan K. Gorilla: a fast, scalable, in-memory time series database. Proc. VLDB Endow., 2015, 8(12): 1816–1827.

[14] Jayasankar U, Thirumal V, Ponnurangam D. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. Journal of King Saud Univ. - Comput. Inf. Sci., 2021, 33(2): 119-140.

[15] Ziv J, Lempel A. A Universal Algorithm for Sequential Data Compression. IEEE Trans. on Information Theory, 1977, 23(3): 337-343.

[16] Ziv J, Lempel A. Compression of individual sequences via variable-rate coding. IEEE Trans. on Information Theory, 1978, 24(5): 530-536.

[17] Chiarot G, Silvestri C. Time Series Compression Survey. ACM Comput. Surv., 2023, 55(10): 198:1-198:32.

[18] Oliveira MAD, Rocha AMD, Puntel FE, Cavalheiro GGH. Time Series Compression for IoT: A Systematic Literature Review. Wireless Commun. Mobile Comput., 2023, Article ID 5025255, 23 pages.

[19] Jensen SK, Pedersen TB, Thomsen C. Time Series Management Systems: A Survey. IEEE Trans. on Knowledge and Data Engineering, 2017, 29(11): 2581-2600.

[20] Zheng Y, He DK, Zhang WF, Lu XH. An Efficient Scheme for GPS Data Compression. China Railway Science, 2005, 26(3): 134-138.

[21] Di S, Cappello F. Fast Error-Bounded Lossy HPC Data Compression with SZ. In: 2016 IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). 2016. 730-739.

[22] Lindstrom P. MULTIPOSITS: Universal coding of Rn. In Conf. on Next Generation Arithmetic, 2022, 66–83.

[23] Chandak S, Tatwawadi K, Wen C, Wang L, Aparicio Ojea J, Weissman T. LFZip: Lossy Compression of Multivariate Floating-Point Time Series Data via Improved Prediction. In: 2020 Data Compression Conf. (DCC). 2020. 342-351.

[24] Zhao K, Di S, Dmitriev M, Tonellot T-LD, Chen ZZ, Cappello F. Optimizing Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering(ICDE). 2021, 1643-1654.

[25] Lu T, Zhong Y, Sun ZB, Chen X, Zhou Y, Wu F, Yang Y, Huang YX, Yang YF. ADT-FSE: A New Encoder for SZ. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). 2023, 45:1–45:13.

[26] Meß J-G, Schmidt R, Fey G, Dannemann F. On the compression of spacecraft housekeeping data using discrete cosine transforms. 2016 Int'l Workshop on Tracking, Telemetry and Command Systems for Space Applications (TTC). 2016, 1-8.

[27] Chen Z, Son SW, Hendrix W, Agrawal A, Liao W-K, Choudhary A. NUMARCK: Machine Learning Algorithm for Resiliency and Checkpointing. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis(SC). 2014, 733-744.

[28] Lindstrom P. Fixed-Rate Compressed Floating-Point Arrays. IEEE Trans. on Visualization and Computer Graphics, 2014, 20(12): 2674-2683.

[29] Liang X, Whitney B, Chen JY, Wan LP, Liu Q, Tao DW, Kress J, Pugmire D, Wolf M, Podhorszki N, Klasky S. MGARD+: Optimizing Multilevel Methods for Error-Bounded Scientific Data Reduction. IEEE Trans. on Computers, 2022, 71(7): 1522-1536.

[30] Li S, Lindstrom P, Clyne J. Lossy Scientific Data Compression With SPERR. In: 2023 IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). 2023, 1007-1017.

[31] Goldstein J, Ramakrishnan R, Shaft U. Compressing Relations and Indexes. In: Proc. of the 14th Int'l Conf. on Data Engineering(ICDE). 1998. 370-379.

[32] Witten IH, Neal RM, Cleary JG. Arithmetic Coding for Data Compression. Communications of the ACM, 1987, 30(6): 520-540.

[33] Duda, J. Asymmetric numeral systems. arXiv:0902.0271v5, 2009.

[34] Google. Snappy: A fast compressor/decompressor. 2023, https://github.com/google/snappy.

[35] Gailly JL, Adler M. 2003, https://www.gzip.org.

[36] FiniteStateEntropy. 2013, https://github.com/Cyan4973/FiniteStateEntropy.

[37] Collet Y. Zstd github repository from facebook. 2016, https://github.com/facebook/zstd.

[38] Alakuijala J, Farruggia A, Ferragina P, Kliuchnikov E, Obryk R, Szabadka Z, Vandevenne L. Brotli: A general-purpose data compressor. ACM Trans. on Information Systems, 2018, 37(1): 1–30.

[39] Lz4: Extremely fast compression algorithm. 2013, https://github.com/lz4/lz4.

[40] Marascu A, Pompey P, Bouillet E, Wurst M, Verscheure O, Grund M, Cudre-Mauroux P. TRISTAN: Real-time analytics on massive time series using sparse dictionary compression. In: Proc. of the IEEE Int'l Conf. on Big Data(Big Data). 2014, 291-300.

[41] Khelifati A, Khayati M, Cudre-Mauroux P. Corad: Correlation-aware compression of massive time series using sparse dictionary coding. In: Proc. of the IEEE Int'l Conf. on Big Data(Big Data). 2019, 2289-2298.

[42] Afroozeh A, Kuffo LX, Boncz P. ALP: Adaptive Lossless floating-Point Compression. Proc. ACM Manag. Data, 2023, 1(4): 230:1-230:26.

[43] Liakos P, Papakonstantinopoulou K, Kotidis Y. Chimp: efficient lossless floating point compression for time series databases. Proc. VLDB Endow., 2022, 15(11): 3058–3070.

[44] Ibarria L, Lindstrom P, Rossignac J, Szymczak A. Out-of-core compression and decompression of large n-dimensional scalar fields. Eurographics, 2003, 343–348.

[45] Cohen A, Daubechies I, Feauveau J-C. Biorthogonal bases of compactly supported wavelets. Communications on Pure and Applied Mathematics, 1992, 45(5): 485–560.

[46] Perlman E, Burns R, Li Y, Meneveau C. Data exploration of turbulence simulations using a database cluster. In: Proc. of the 2007 ACM/IEEE conf. on Supercomputing, 2007, 1–11.

[47] Tucker LR. Some mathematical notes on three-mode factor analysis. Psychometrika, 1966, 31: 279–311.

[48] Ruan SJ, Fu X, Long C, Xiong Z, Bao J, Li RY, Chen YH, Wu SN, Zheng Y. Filling delivery time automatically based on couriers' trajectories. IEEE Trans. on Knowledge and Data Engineering, 2021, 35(2): 1528–1540.

[49] Lin J, Zhu LG, Chen WM, Wang WC, Gan C, Han S. On-device training under 256kb memory. Advances in Neural Inf. Processing Systems, 2022, 35: 22 941–22 954.

[50] Liakos P, Papakonstantinopoulou K, Bruineman T, Raasveldt M, Kotidis Y. How to Make your Duck Fly: Advanced Floating Point Compression to the Rescue. Proc. 27th Int'l Conf on Extending Database Technology(EDBT). 2024, 826-829.

[51] Li RY, Li Z, Wu Y, Chen C, Guo ST, Zhang M, Zheng Y. Erasing-based lossless compression method for streaming floating-point time series. arXiv:2306.16053, 2023.

[52] Kuschewski M, Sauerwein D, Alhomssi A, Leis V. BtrBlocks: Efficient Columnar Compression for Data Lakes. Proc. ACM Manag. Data, 2023, 1(2): 118:1-118:26.

[53] Lemire D, Boytsov L. Decoding billions of integers per second through vectorization. Software: Practice and Experience, 2015, 45(1): 1-29.

[54] Ierusalimschy R. Programming in lua. Roberto Ierusalimschy, 2006.

[55] Lindstrom P, Isenburg M. Fast and Efficient Compression of Floating-Point Data. IEEE Trans. on Visualization and Computer Graphics, 2006, 12(5): 1245-1250.

[56] Schindler M. A Fast Renormalisation for Arithmetic Coding. In: Proc. Data Compression Conf. (DCC). 1998, 572-.572.

[57] Burtscher M, Ratanaworabhan P. FPC: A High-Speed Compressor for Double-Precision Floating-Point Data. IEEE Trans. on Computers, 2009, 58(1): 18-31.

[58] Sazeides Y, Smith JE. The Predictability of Data Values. Proc. 30th Int'l Symp. Microarchitectur. 1997, 248-258.

[59] Goeman B, Vandierendonck H, Bosschere K. Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency. Proc. 17th Int'l Symp. High Performance Computer Architecture. 2001, 207-216.

[60] Yu XY, Peng YQ, Li FF, Wang S, Shen XW, Mai HJ, Xie Y. Two-Level Data Compression using Machine Learning in Time Series Database. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering(ICDE). 2020, 1333-1344.

[61] Liu CW, Jiang H, Paparrizos J, Elmore AJ. Decomposed bounded floats for fast compression and queries. Proc. VLDB Endow., 2021, 14(11): 2586–2598.

[62] Klöwer M, Razinger M, Dominguez JJ, Düben PD, Palmer TN. Compressing atmospheric data into its real information content. Nat Comput Sci, 2021, 1: 713–724.

[63] Shannon CE. A mathematical theory of communication. Bell Syst. Tech. J. 1948, 27(3): 623–656.

[64] Tao DW, Di S, Chen ZZ, Cappello F. Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization. In: 2017 IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). 2017, 1129-1139.

[65] Liang X, Di S, Tao DW, Li SH, Li SM, Guo HQ, Chen ZZ, Cappello F. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In: Proc. of the IEEE Int'l Conf. on Big Data (Big Data). 2018, 438-447.

[66] Li S, Di S, Liang X, Chen ZZ, Cappello F. Optimizing Lossy Compression with Adjacent Snapshots for N-body Simulation Data. In: Proc. of the IEEE Int'l Conf. on Big Data (Big Data). 2018, 428-437.

[67] Zhao K, Di S, Liang X, Li SH, Tao DW, Chen ZZ, Cappello F. Significantly Improving Lossy Compression for HPC Datasets with Second-Order Prediction and Parameter Optimization. In: Proc. of the 29th Int'l Symp. on High-Performance Parallel and Distributed Computing (HPDC). 2020, 89–100.

[68] Liu JY, Di S, Zhao K, Liang X, Chen ZZ, Cappello F. Dynamic Quality Metric Oriented Error Bounded Lossy Compression for Scientific Datasets. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). 2022, 1-15.

[69] Shi Y, Zou XY, Chen XY, Jin S, Tao DW, Cai D, Chen YF, Xia W. Machete: An Efficient Lossy Floating-Point Compressor Designed for Time Series Databases. In: 2024 Data Compression Conf. (DCC). 2024.

[70] Tian JN, Di S, Zhao K, Rivera C, Fulp MH, Underwood R, Jin S, Liang X, Calhoun J, Tao DW, Cappello F. CuSZ: An Efficient GPU-Based Error-Bounded Lossy Compression Framework for Scientific Data. In: Proc. of the ACM Int'l Conf. on Parallel Architectures and Compilation Techniques, 2020, 3–15.

[71] Elmeleegy H, Elmagarmid AK, Cecchet E, Aref WG, Zwaenepoel W. Online piece-wise linear approximation of numerical streams with precision guarantees. Proc. VLDB Endow., 2009, 2(1): 145–156.

[72] Luo G, Yi K, Cheng SW, Li ZG, Fan W, He C, Mu YD. Piecewise linear approximation of streaming time series data with max-error guarantees. In: Proc. of the 31st IEEE Int'l Conf. on Data Engineering(ICDE). 2015, 173-184.

[73] Kitsios X, Liakos P, Papakonstantinopoulou K, Kotidis Y. Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. Proc. VLDB Endow., 2023, 16(8): 1910-1922.

[74] Google. Protocol Buffers Encoding. 2001. https://developers.google.com/protocol-buffers/docs/encoding#types.

[75] Bfloat16 floating-point format. 2021, https://en.wikipedia.org/wiki/Bfloat16.

[76] Barbarioli B, Mersy G, Sintos S, Krishnan S. Hierarchical Residual Encoding for Multiresolution Time Series Compression. Proc. ACM Manag. Data, 2023, 1(1): 99:1-99:26.

[77] Grebnov I. BSC, 2015, http://libbsc.com/.

[78] Feng HM, Ma RZ, Yan Li, Ma ZM. 2023. Spatiotemporal Prediction Based on Feature Classification for Multivariate Floating-Point Time Series Lossy Compression. Big Data Res. 2023, Vol.32.

[79] Jiang N, Xiang QP, Wang HZ, Zheng B. Time series compression based on reinforcement learning. Inf. Sci., 2023, 648-119490.

[80] Daubechies I, Sweldens W. Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications, 1998, 4(3): 247–269.

[81] Habboush M, El-Maleh AH, Elrabaa MES, AlSaleh S. DE-ZFP: An FPGA implementation of a modified ZFP compression/decompression algorithm. Microprocessors and Microsystems, 2022, 90-104453.

[82] Lu B, Li Y, Wang J, Luo H, Li K. ZFP-X: Efficient Embedded Coding for Accelerating Lossy Floating Point Compression. In: 2023 IEEE Int'l Parallel and Distributed Processing Symp.(IPDPS). 2023, 1041-1050.

[83] Ballester-Ripoll R, Lindstrom P, Pajarola R. TTHRESH: Tensor Compression for Multidimensional Visual Data. IEEE Trans. on Visualization and Computer Graphics, 2020, 26(9): 2891-2903.

[84] Sasaki N, Sato K, Endo T, Matsuoka S. Exploration of Lossy Compression for Application-Level Checkpoint/Restart. In: 2015 IEEE Int'l Parallel and Distributed Processing Symp.(IPDPS). 2015. 914-922.

[85] Ainsworth M, Tugluk O, Whitney B, Klasky S. Multilevel techniques for compression and reduction of scientific data—the univariate case. Comput. Visualization Sci., 2018, 19(5–6): 65–76.

[86] National Ecological Observatory Network (NEON). 2022. Barometric pressure (DP1.00004.001). https://doi.org/10.48443/ZR37-0238 Retrieved March 19, 2023.

[87] InfluxDB 2.0 Sample Data. Retrieved March 19, 2023, https://github.com/influxdata/influxdb2-sample-data.

[88] Historical Weather Data Download. Retrieved March 19, 2023, https://www.meteoblue.com/en/weather/archive/export/basel_switzerland.

[89] Daily Temperature of Major Cities. Retrieved March 19, 2023, https://www.kaggle.com/sudalairajkumar/daily-temperature-of-major-cities.

[90] National Ecological Observatory Network (NEON). 2022. Relative humidity above water on-buoy (DP1.20271.001). https://doi.org/10.48443/1W06-WM51 Retrieved March 19, 2023, https://data.neonscience.org/data-products/DP1.20271.001/RELEASE-2022.

[91] National Ecological Observatory Network (NEON). 2022. IR biological temperature (DP1.00005.001). https://doi.org/10.48443/7RS6-FF56 Retrieved March 19, 2023, https://data.neonscience.org/data-products/DP1.00005.001/RELEASE-2022.

[92] National Ecological Observatory Network (NEON). 2022. Dust and particulate size distribution (DP1.00017.001). https://doi.org/10.48443/RDZ9-XR84 Retrieved March 19, 2023, https://data.neonscience.org/data-products/DP1.00017.001/RELEASE-2022.

[93] Financial data set used in INFORE project. Retrieved March 19, 2023, https://zenodo.org/record/3886895#.Y4DdzHZByM_.

[94] National Ecological Observatory Network (NEON). 2022. 2D wind speed and direction (DP1.00001.001). https://doi.org/10.48443/77N6-EH42 Retrieved March 19, 2023, https://data.neonscience.org/data-products/DP1.00001.001/RELEASE-2022.

[95] Chen XZ, Sha EHM, Abdullah A, Zhuge QF, Wu L, Yang CS, Jiang WW. UDORN: A design framework of persistent in-memory key-value database for NVM. In: 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symp. (NVMSA). 2017, 1-6.

[96] Chen XY, Tian JN, Beaver I, Freeman C, Yan Y, Wang JG, Tao DW. 2024. FCBench: Cross-Domain Benchmarking of Lossless Compression for Floating-Point Data. Proc. VLDB Endow., 2024, 17(6): 1418-1431.

[97] Li J, Chen XZ, Liu D, Li L, Wang JP, Zeng ZY, Tan YJ, Qiao L. Horae: A Hybrid I/O Request Scheduling Technique for Near-Data Processing-Based SSD. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 3803-3813.

[98] Shi WS, Zhang XZ, Wang YF, Zhang QY. Edge Computing: State-of-the-Art and Future Directions. Journal of Computer Research and Development, 2019, 56(1): 69-89. (in Chinese) [doi: 10.7544/issn1000-1239.2019.20180760]

[99] Xie Z, Raskar S, Emani M. Throughput-oriented and Accuracy-aware DNN Training with BFloat16 on GPU. In: Proc. of the IEEE Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW). 2022, 1084-1087.

附中文参考文献:

[1] 阮思捷, 熊可钦, 王树良, 等. 众包时空数据驱动的城市地理信息推测综述[J]. 电子学报, 2023, 51(8): 2238-2259. [doi: 10.12263/DZXB.20230131]

[2] 俞自生, 李瑞远, 郭阳, 蒋忠元, 鲍捷, 郑宇. 基于键值存储的分布式时序相似性搜索方法. 软件学报, 2022, 33(3): 950-967. http://www.jos.org.cn/1000-9825/6445.htm [doi: 10.13328/j.cnki.jos.006445]

[5] 左一萌,林学练,马帅,姜家豪.路网感知的在线轨迹压缩方法.软件学报,2018,29(3):734−755. http://www.jos. org.cn/1000-9825/5438.htm [doi: 10.13328/j.cnki.jos.005438]

[7] 董昊文,张超,李国良,冯建华.云原生数据库综述.软件学报,2024,35(2):899-926. http://www.jos.org.cn/1000-9825/6952.htm [doi: 10.13328/j.cnki.jos.006952]

[8] 周搏洋,陈春雨,王强,周福才.全委托的公共可验证的外包数据库方案.软件学报,2021,32(12):3901-3916. http://www.jos.org.cn/1000-9825/6129.htm [doi: 10.13328/j.cnki.jos.006129]

[10]贺文迪, 夏天睿, 宋韶旭, 黄向东, 王建民. Apache IoTDB 中的多模态数据编码压缩.软件学报, 2024, 35(3): 1173–1193. http://www.jos.org.cn/1000-9825/7077.htm [doi: 10.13328/j.cnki.jos.007077]

[11] 邓岸华,乔磊,杨孟飞. 面向星载计算机的双重索引数据压缩方法. 软件学报, 2022, 33(10): 3844–3857. http://www.jos.org.cn/1000-9825/6308.htm [doi: 10.13328/j.cnki.jos.006308]

[20] 郑宇, 何大可, 张文芳, 路献辉. 一种有效的 GPS 数据压缩方案. 中国铁道科学, 2005, 26(3): 134-138.

[98] 施巍松, 张星洲, 王一帆, 张庆阳. 边缘计算: 现状与展望[J]. 计算机研究与发展, 2019, 56(1): 69-89. [doi: 10.7544/issn1000-1239.2019.20180760]

朱明辉(1998－),男,硕士生,CCF 学生会员,主要研究领域为时空数据压缩.

李政(2000－),男,博士生,CCF 学生会员,主要研究领域为数据压缩,时空数据管理.

李瑞远(1990－),男,博士,副教授,CCF 高级会员,主要研究领域为时空数据管理与挖掘,城市计算.

陈超(1985－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为普适计算,时空数据挖掘.

郑宇(1979－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为时空数据挖掘,城市计算.