通用、普通的技术主题：文件压缩算法

杨其臻

Jul 18, 2026

文件大小的持续增长已成为现代计算环境的常态。多媒体内容的像素级膨胀、日志系统的毫秒级写入、深度学习模型的参数规模以及海量训练数据集的累积，都在不断推高存储需求。压缩技术因此成为工程实践中不可或缺的工具，它不仅能够显著减少磁盘占用，更能降低网络传输延迟并提升 I/O 吞吐。
本文旨在构建一个系统化的知识框架，涵盖无损与有损两大范式，梳理从经典算法到现代实现的演进脉络，并提供场景化的选型策略。读者将逐步理解信息熵的理论基础、主流算法的内部机制，以及工程落地的权衡要点。

1　基础概念与分类

无损压缩与有损压缩构成压缩领域的两大分支。前者通过消除数据中的统计冗余实现可逆还原，适用于文本、源代码、可执行文件等任何不能容忍信息损失的场景；后者则借助人类感知系统的局限性，主动丢弃难以察觉的细节，主要服务于图像、音频与视频等多媒体内容。
信息论为理解压缩提供了数学根基。香农熵量化了消息的不确定性，其值越低意味着数据中可预测的模式越多，压缩潜力越大。任何压缩算法的理论下限都受限于源的熵值，实际实现则需在编码效率、计算复杂度与内存占用间做出权衡。
压缩流程通常可分解为建模、编码与后处理三个环节。建模阶段建立符号的概率分布或重复模式，编码阶段将模型映射为紧凑的比特串，后处理则负责格式封装与索引构建。三个环节相互依存，共同决定了最终的压缩率与解压速度。

2　经典无损压缩算法

行程长度编码是最早被广泛采用的简单方法。它将连续重复的符号替换为「符号 + 计数」的二元组，例如对序列 AAAAABBB 可编码为 A5B3。该方法在扫描图像或传真数据中效果显著，但面对随机性较强的内容时几乎无法带来收益。
哈夫曼编码通过构建最优前缀码树来逼近熵下限。静态版本需要两遍扫描：首遍统计频率并建树，次遍完成编码；动态版本则在单遍扫描中边统计边更新树结构；规范哈夫曼则进一步限制码长分布以加速解码。建树过程遵循自底向上的贪心策略，每次合并当前最小的两个节点，直至根节点出现。
LZ77 通过维护一个滑动窗口来捕捉历史重复。编码器在当前位置向后搜索与窗口内最长匹配的子串，若找到则输出「距离、长度」对，否则输出原始符号。LZ78 及其变体 LZW 则改用显式字典树，逐步将新出现的字符串加入字典，无需回溯窗口。Deflate 算法将两者结合：先用 LZ77 消除重复，再用哈夫曼编码对距离与长度进行熵编码，从而成为 ZIP、Gzip 与 PNG 的核心。
算术编码放弃了符号级整数码长的限制，转而用一个浮点区间表示整个消息。随着符号逐个读入，区间不断细

分，最终用区间内的一个二进制小数表示整条消息。实际实现中常用 Range Coder 以整数运算近似浮点运算，避免精度溢出。

Burrows-Wheeler 变换通过对所有循环移位进行字典序排序，将局部重复集中到相邻位置，随后配合 Move-to-Front 变换与 RLE 即可获得极高的压缩率。bzip2 正是这一组合的经典实现，其压缩率往往优于 Deflate，但解压速度较慢。

3 有损压缩算法

有损压缩的核心在于利用感知冗余。人类视觉对高频细节不敏感，听觉则对掩蔽效应下的微弱信号难以察觉。算法设计者据此引入量化操作，将连续值映射到有限的离散集合，从而实现不可逆的信息丢弃。

JPEG 针对静止图像采用离散余弦变换，将 8×8 像素块转换到频域，再按量化表丢弃高频系数。色度分量通常以 4:2:0 格式采样，进一步降低数据量。解码端通过逆变换与反量化重建图像，量化表的设计直接影响最终画质。

音频领域，MP3 与 AAC 均基于改进离散余弦变换，将时域信号映射到 576 或 1024 个频带。心理声学模型计算每个频带的掩蔽阈值，据此分配比特。Spectral Band Replication 等技术可在低码率下合成高频成分，进一步提升感知质量。

视频编码在图像编码基础上增加了时域预测。H.264/H.265 通过运动补偿寻找参考帧中的相似块，仅传输残差与运动矢量。CABAC 算术编码则对语法元素进行上下文自适应熵编码。AV1 作为开源标准，在编码工具集上进一步扩展，试图在同等码率下提供更优画质。

近年来，基于生成对抗网络与神经编解码器的学习型压缩开始崭露头角。模型通过端到端训练直接学习感知最优的量化与熵编码策略，部分方案已在低码率场景下超越传统方法。

4 主流工具与文件格式对比

不同工具在算法选择、实现细节与应用场景上存在显著差异。gzip 基于 Deflate，兼顾压缩率与速度，适合 Web 传输与日志归档；bzip2 采用 BWT 变换，压缩率更高但速度偏慢，适合源码与大文本；xz 使用 LZMA 算法，压缩率极高，适用于软件包与固件分发。

Zstandard 通过融合 LZ 字典匹配与有限状态熵编码，在极高速度下仍能提供优秀压缩率，并原生支持字典与随机访问，适合数据库与实时流场景。Brotli 针对 Web 静态资源优化，静态字典与上下文建模使其在同等速度下压缩率优于 gzip。7z 通过 LZMA2 与 BCJ 过滤器，在离线归档中表现突出。

列式存储格式如 Parquet 通常内置 Snappy 或 Zstandard，可对列内数据独立压缩，同时保留块级索引，支持高效的谓词下推与随机读取。ZIP 格式因跨平台兼容性仍被广泛用于文件交换，尽管其默认 Deflate 压缩率一般，但也支持切换到 LZMA 以提升效果。

5 工程实践与选型策略

在延迟敏感场景下，应优先选择低级别 Zstandard、Snappy 或 LZ4。这些算法在单核性能上可达数 GB/s，适合网络协议栈与内存数据库。存储或归档优先的场景则可接受稍高的计算开销，选用 Zstandard 的 ultra 模式、xz 的最高级别或 7z，以换取更小的归档体积。

Web 资源压缩需区分静态与动态内容。静态文件可在构建阶段预压缩为 Brotli，动态响应则保留 gzip 以降低 CPU 开销。数据库与日志系统通常结合列式布局、字典编码与位图索引，多级压缩策略可进一步降低存储成本。

嵌入式与微控制器环境受内存与算力限制，需选用 LZ4、LZO 或专为小内存设计的 Heatshrink。固件更新场景下，压缩率与解压内存的平衡尤为关键。

实际选型时应建立量化 Benchmark，记录不同级别下的压缩率、压缩时间、解压时间与峰值内存。曲线可视化有助于直观判断拐点，避免盲目追求极致参数。

6 进阶主题

当数据集合中存在大量相似文件时，可通过预训练字典或块级重复数据删除进一步提升效果。Zstandard 的字典训练接口允许从样本集中学习高频短语，显著改善小文件压缩率。

并行化是提升吞吐的常规手段。Pigz 通过多线程并行 Deflate，Zstandard 的 `-T` 参数可自动分配线程。FPGA 与 GPU 加速方案如 Intel QAT 与 NVIDIA nvCOMP，则将核心循环卸载到硬件，适合数据中心批量处理。

加密与压缩的顺序选择直接影响安全性与效率。先压缩后加密可获得更好压缩率，但密文本身已接近随机，难以再次压缩；先加密后压缩则几乎无效。因此生产环境通常在应用层完成压缩，再由传输层或存储层负责加密。

硬件加速正从可选变为标配。Intel QAT 支持 Deflate 与 LZ4 的硬件卸载，Apple 芯片内置 Zlib 引擎可在不占用 CPU 的情况下完成 gzip 兼容的压缩。开发者需关注平台差异，合理选择软件回退路径。

理论前沿方面，Kolmogorov 复杂度提供了不可计算的绝对下限，在线算法则研究单遍、有限内存下的最优策略。学习型压缩通过神经网络直接建模条件概率，代表了下一代自适应压缩的方向。

7 常见误区与避坑指南

「压缩率越高越好」是一个常见误区。高压缩率往往伴随更高的解压延迟与内存峰值，对在线服务而言可能得不偿失。应根据服务水平协议确定可接受的解压时间上限，再在此约束下追求压缩率。

「文本一定比二进制好压缩」的说法同样片面。关键在于局部重复度与符号分布的熵值。随机生成的 JSON 可能比结构化的二进制协议更难压缩，而高度重复的日志文本则极易压缩。

「压缩后还要再压缩」的做法通常无效。一次高质量的熵编码已将数据逼近理论下限，再次压缩几乎不会带来收益，反而增加开销。

文件系统级压缩与应用层压缩可能产生重复工作。Btrfs 与 ZFS 的透明压缩在块级别运行，若应用已完成高效压缩，文件系统压缩反而可能增加 CPU 负担且无法进一步缩小体积。建议在部署前评估两者的叠加效应。

8 动手实验与可视化

推荐使用 Linux 内核源码、enwik8 语料库与标准 JPEG 测试图作为 Benchmark 数据集。这些数据集覆盖文本、混合二进制与图像，具备代表性。

命令行实验可借助 `time` 与 `/usr/bin/time` 统计真实 CPU 时间与最大常驻内存。脚本示例可遍历多个文件与级别，输出 CSV 以便后续绘图。

在线工具如 CyberChef 可直观展示各阶段的中间结果，hexyl 则能以十六进制视图观察压缩前后字节分布。结合 `zstd --show-dict` 可检查训练得到的字典内容。

Python 代码示例可快速验证流式压缩：

```
import zstandard as zstd
compressor = zstd.ZstdCompressor(level=3)
```

```
with open('input.bin', 'rb') as fin, open('output.zst', 'wb') as fout:
    compressor.copy_stream(fin, fout)
```

这段代码首先实例化级别为 3 的压缩器，随后以二进制模式打开输入与输出文件，最后调用 copy_stream 方法完成流式压缩。copy_stream 内部维护输入输出缓冲，避免一次性加载整个文件，适合大文件处理。level=3 参数控制字典大小与搜索深度，数值越高压缩率通常越高，但速度与内存占用也会上升。

9 结论与展望

文件压缩没有银弹，不同场景对压缩率、速度、内存与随机访问的需求各异。工程实践的核心在于建立可量化的 Benchmark 流水线，根据 SLA 持续调优参数。

展望未来，AI 驱动的自适应压缩将在边缘设备与云端同时落地。端到端学习型编解码器有望在极低码率下提供可接受的感知质量，而硬件加速将进一步降低延迟。开发者应保持对新算法与硬件特性的关注，及时将成熟技术纳入工具链。

10 附录

术语表包含 Entropy（信息熵）、Dictionary（字典）、Sliding Window（滑动窗口）与 Quantization（量化）等核心概念。

进一步阅读建议包括 RFC 1951（Deflate 规范）、Zstandard 官方论文以及 JPEG 标准白皮书。

Benchmark 原始数据与图表源码已整理至公开仓库，读者可据此复现实验结果并扩展测试用例。