计算机研究与发展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(3): 597−614, 2026

Zstandard 中 LZ77 压缩算法的高效匹配策略设计与实现

韦芳宇[1] 陈天韵[2] 周 洋[2] 谢雨来[3] 王 芳[2]
[1](华中科技大学计算机科学与技术学院 武汉 430074)
[2](武汉光电国家研究中心(华中科技大学) 武汉 430074)
[3](华中科技大学网络空间安全学院 武汉 430074)
(wfy_hust@qq.com)

Design and Implementation of Efficient Matching Strategies for LZ77 Compression Algorithm in Zstandard

Wei Fangyu[1], Chen Tianyun[2], Zhou Yang[2], Xie Yulai[3], and Wang Fang[2]
[1] (*School of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074)
[2] (*Wuhan National Laboratory for Optoelectronics* (*Huazhong University of Science and Technology*), *Wuhan* 430074)
[3] (*School of Cyber Science and Engineering*, *Huazhong University of Science and Technology*, *Wuhan* 430074)

Abstract In the context of the information age, the explosive growth of data volume and the diversification of compression application scenarios raise higher demands for the flexibility and efficiency of compression strategies. The LZ77 algorithm, a classical lossless compression method, is widely used in mainstream compression tools such as Zstandard (ZSTD). However, achieving higher compression ratios often requires larger history windows and more complex compression strategies, which in turn lead to frequent cache misses and reduced efficiency of lazy matching in ZSTD's LZ77 implementation. To address these issues, two optimization strategies are proposed. First, the multi-level region search strategy (MLRS) introduces a hierarchical matching region mechanism along with access threshold control, enabling adaptive adjustment of search depth and restricting memory access range during the matching process to alleviate cache pressure. Second, the extended search-based lazy matching strategy (ESLM) reuses search paths and incorporates approximate substitution techniques to reduce redundant computation while improving matching efficiency. These strategies are implemented and evaluated under ZSTD level 12 configuration on Kunpeng 920 server platform. Experimental results demonstrate that MLRS significantly reduces last-level cache miss rates across various datasets and improves compression throughput to 118.34%−149.50% of the baseline while maintaining a compression ratio within 94.65%−99.58%. ESLM improves compression throughput to 113.49%−117.46% of the original while further increasing the compression ratio on most datasets. When both strategies are applied jointly, the compression speed increases to 134.53%−171.17% of the original scheme, while sustaining a compression ratio in the range of 94.18%−99.80%.

Key words data compression; LZ77 algorithm; cache optimization; lazy matching; throughput optimization

摘 要 在信息时代背景下,数据规模的急剧增长与压缩应用场景的多样化,对压缩策略的灵活性和效率

收稿日期: 2025-06-17; **修回日期**: 2025-12-29
基金项目: 国家重点研发计划项目(2022YFB4501300); 国家自然科学基金项目(U22A2027, 624B2056); 中国博士后科学基金项目(2025M781469)
This work was supported by the National Key Research and Development Program of China (2022YFB4501300), the National Natural Science Foundation of China (U22A2027, 624B2056), and the China Postdoctoral Science Foundation (2025M781469).
通信作者: 王芳(wangfang@mail.hust.edu.cn)

提出了更高要求。LZ77 算法是典型的无损压缩方法,广泛应用于 Zstandard(ZSTD)等主流压缩工具中。然而,更高的压缩率指标要求应用更大的历史窗口与更复杂的压缩策略,导致在实现 ZSTD 中的 LZ77 算法存在缓存频繁未命中与延迟匹配效率低下的问题。为此,提出 2 项优化策略:其一,多级区域搜索策略(multi-level region search strategy, MLRS),通过引入匹配区域分级与访问阈值控制机制灵活调整搜索深度,限制匹配过程中的数据访问范围,缓解缓存压力;其二,基于扩展搜索的延迟匹配策略(extended search-based lazy matching strategy, ESLM),通过复用搜索路径并采用近似替代技术,降低冗余计算的同时提升匹配效率。上述优化策略基于 ZSTD level 12 配置在鲲鹏 920 服务器平台上实现并完成验证。实验结果表明:MLRS 在多种数据集上能够显著降低压缩过程中的末级缓存未命中率,将压缩率保持在 94.65%~99.58% 的同时,使压缩吞吐量提升至原方案的 118.34%~149.50%;ESLM 可将压缩吞吐量提升至原方案的 113.49%~117.46%,且在多数数据集上进一步提高压缩比;当两者联合应用时,压缩速度可提升至原方案的 134.53%~171.17%,同时维持 94.18%~99.80% 的压缩率。

关键词 数据压缩;LZ77 算法;缓存优化;延迟匹配;吞吐量优化

中图法分类号 TP311.56

DOI: 10.7544/issn1000-1239.202550480 **CSTR:** 32373.14.issn1000-1239.202550480

在信息时代,全球数据总量呈现爆发式增长。据 IDC[1] 预测,2025 年全球将产生 213.56 ZB 数据,到 2029 年将增长 1 倍以上,达到 527.47 ZB。同时,数据的高效利用蕴含着巨大的经济价值,这使得生成式人工智能、人工智能科学计算等新型数据驱动型技术备受关注,如何结合这些新兴技术的特点与需求高效地存储与利用海量数据成为当前的研究热点。

无损压缩技术能够在完整保留原始信息的前提下,有效减小数据体积,是提升数据存储与传输效率的重要手段。该类技术不仅有助于降低存储成本,还能减少传输和处理过程中的资源开销,从而提升计算系统的整体性能。其基本原理在于通过识别并消除数据中的冗余信息,实现体积压缩。然而,追求较高压缩率通常依赖于复杂的算法设计以挖掘深层次冗余,这往往会带来显著的计算资源消耗;反之,为提高压缩速度则需简化匹配流程,但常以牺牲压缩率为代价。

Zstandard[2](ZSTD)是当前主流的无损数据压缩工具之一,凭借其优异的压缩吞吐率与压缩比而得到广泛应用。为满足不同场景下的多样化需求,ZSTD 通过核心参数配置提供了多种压缩方案,相较于其他工具更具灵活性。ZSTD 结合字典压缩中的 LZ77[3] 算法以及熵编码中的哈夫曼编码[4] 与 FSE[5](finite state entropy)算法,实现了高效的无损压缩。然而,ZSTD 压缩过程的主要开销仍源于 LZ77 算法。随着对压缩率要求的不断提高,历史缓冲区的扩大和压缩流程的复杂化导致 LZ77 算法的执行开销显著增大,最终限制了压缩吞吐量。因此,如何针对不同场景需求,合理权衡压缩率与压缩速度之间的关系,成为当前值得关注的问题。

得益于 ZSTD 的性能优势与灵活性,文件共享、虚拟化等实时数据压缩场景通常将其中等压缩级别(level 8~12)的压缩率作为基准线,要求在压缩率不低于此基准的前提下实现高压缩吞吐量。然而,随着新质生产力的不断发展,对压缩吞吐量的要求持续提升,ZSTD 现有方案已难以满足当下实时压缩场景的需求。因此,本文的核心研究问题聚焦于:如何在保持高压缩率的前提下,有效提升压缩吞吐量,以满足当下文件共享、虚拟化等实时数据压缩场景的需求。

LZ77 算法的压缩率在很大程度上依赖于历史窗口的大小以及匹配搜索策略的设计。为了获得更高的压缩率,LZMA[6](Lempel-Ziv-Markov chain algorithm)支持最大 4 GB 的历史窗口,并采用数百次的匹配搜索尝试与延迟匹配策略,显著提升匹配效果,压缩率远高于大多数通用压缩算法,但也导致压缩速度较慢。相较之下,LZ4[7] 将历史窗口限制为 64 KB,且每次仅检查 1 个历史位置,极大地提高了压缩和解压缩速度,但压缩率明显低于其他通用算法。

在 ZSTD 中等级别压缩方案中,为了进一步提升压缩率,使用大历史窗口与延迟匹配策略。前者通过提供充足历史数据来增加搜索到更优匹配串的机会以提升压缩率,后者通过比较相邻处理位置间的最优匹配串搜索结果来选取最优匹配点以实现压缩率的提升。然而,与 LZMA 情况类似,这 2 种策略虽然能提升压缩率,但也显著增加计算与内存开销,成

为限制压缩吞吐量的关键因素。本文面临的主要挑战在于：如何有效利用这 2 种策略带来的压缩率收益，同时尽可能减少其引入的额外计算开销与内存消耗。

针对大历史窗口导致的吞吐量下降，本文提出了基于压缩情况动态调整搜索深度的优化思路。该思路根据最近最优匹配串查找过程中候选匹配项的质量，自适应调整后续搜索深度，有效过滤低价值候选，从而提高匹配搜索效率。针对延迟匹配策略带来的吞吐量损失，本文提出了利用相邻处理位置间搜索路径重叠性来减少冗余搜索操作的优化思路。该思路通过扩展搜索与近似替代的方式简化延迟匹配过程，实现更高的压缩效率。基于上述 2 种优化思路，本文实现了在压缩率与 ZSTD level 12 方案基本持平的前提下显著提升压缩吞吐量的目标。主要贡献如下：

1）提出基于分级思想的多级区域搜索策略（multi-level region search strategy, MLRS）。该策略通过候选项级别划分、分级匹配以及动态阈值调节，灵活调节最优匹配串查找的搜索深度。它能够有效过滤低价值候选项，减少无效计算；同时，通过缩小数据访问范围，缓解最优匹配查找过程中的缓存不友好问题，显著降低缓存未命中率，从而提升整体压缩性能。

2）提出基于扩展搜索的延迟匹配策略（extended search-based lazy matching strategy, ESLM）。该策略通过在相邻处理位置间执行扩展搜索，利用搜索路径间的重叠性以消除部分冗余的匹配操作，从而缓解原延迟匹配（lazy matching）策略中存在的搜索路径冗余问题。在保持压缩率基本不变的前提下，其能够有效提升压缩性能。

3）实验结果表明，本文提出的 MLRS 相较于原始方案，能够在多种类型的数据集上有效降低压缩过程中末级缓存访问的数据缺失率。在压缩率保持在 94.65%~99.58% 的前提下，压缩吞吐量提升幅度可达 18.34%~49.50%。此外，ESLM 可带来 13.49%~17.46% 的性能增幅，并在多数数据集上进一步提升压缩率。当 2 项优化策略联合应用时，系统整体压缩性能可提升 34.53%~71.17%，压缩率则稳定保持在 94.18%~99.80% 的范围内。综上所述，所提出的方法在保持高压缩率的同时，实现了压缩性能的提升，具有较强的实用价值。

1 研究背景

1.1 LZ77 算法简介

数据压缩源于信息论之父 Shannon[8]（香农）于 1948 年提出的香农编码。数据压缩技术根据其压缩过程是否存在信息丢失，即解压缩后数据是否与原始数据完全相同分为无损压缩和有损压缩 2 类。在实现原理上，无损压缩通过消除冗余数据实现压缩，确保数据完全还原；有损压缩通过保留与实际应用密切相关的核心特征，并有选择地舍弃对目标场景影响较小的信息，以实现更高的压缩率，但其解压后的数据无法完全还原为原始形式。

无损压缩算法主要分为字典编码和熵编码（统计编码）两大类。其中，字典编码的核心思想是通过构建字典，用索引替换重复出现的字符串或序列，从而实现数据压缩；熵编码则是通过分析数据的统计特性，为高频符号分配短编码，为低频符号分配长编码，以实现压缩。

LZ77 算法是字典编码的标志性算法，由 Lempel 和 Ziv[3] 于 1977 年提出。其核心思想是通过滑动窗口在历史数据中查找最长匹配，以减少数据冗余。如图 1 所示，LZ77 压缩算法将数据流分为 2 部分：左侧为搜索缓冲区，存放已处理的数据，用于在压缩过程中提供重复串的匹配参考；右侧为前瞻缓冲区，存放待编码的数据。算法在搜索缓冲区中查找与前瞻缓冲区起始位置匹配的子串，并将匹配结果以元组形式的符号进行编码表示。随后，将当前处理位置移动至匹配串末尾，继续执行匹配与编码过程。每

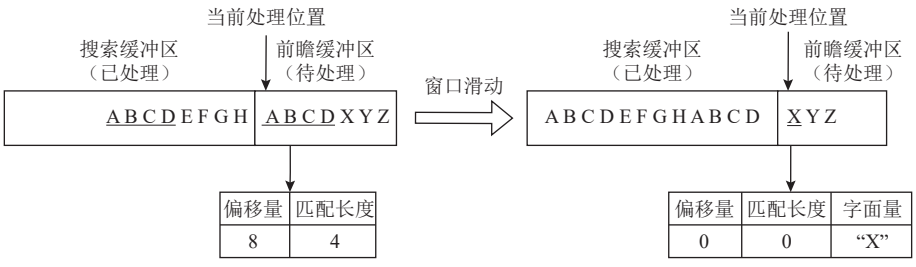


Fig. 1 Schematic diagram of LZ77 algorithm principle

图 1 LZ77 算法原理示意图

次匹配确定后,编码器生成一个固定长度的记号并写入输出流。根据是否找到匹配,记号可分为2种类型:若找到匹配,则记录偏移量(从当前位置向前追溯,匹配串在搜索缓冲区中的起始位置)和匹配长度(所找到匹配串的长度);若未找到匹配,则将偏移量与匹配长度置0,同时记录当前未压缩的字面量数据。在解压缩过程中,则直接根据所记录的偏移量、匹配长度以及不可压缩字面量进行解码即可将压缩数据进行完整还原。

LZ77算法是无损压缩算法中字典压缩的典型代表,LZ4, Snappy[9], LZSS[10], LZO[11]等一系列无损压缩算法都是由LZ77算法衍生而来的。自LZ77被提出以来,因其高效的数据冗余消除能力,在硬件与嵌入式系统[12-14]、图像与多媒体处理[15]、文本与数据检索[16]等多个领域得到广泛应用。

1.2 ZSTD中LZ77算法的实现过程

ZSTD是一款由Facebook开发的实时压缩工具,兼具高压缩比与高速压缩解压性能。其核心压缩流程如图2所示,结合了LZ77算法与熵编码技术:待压缩数据流首先通过LZ77编码器得到其对应的LZ77分解序列以去除数据冗余,然后使用熵编码对分解序列结果进行进一步压缩,最终实现高效的无损压缩。

文献[17]的测试结果显示,在ZSTD整个压缩过程中,LZ77编码时间远高于熵编码时间。同时,最优匹配串查找与延迟匹配是ZSTD中LZ77算法实现的核心步骤,因此本文主要对这2个操作进行介绍与研究。

1.2.1 最优匹配串查找

在LZ77算法的实现过程中,最优匹配串的查找是影响压缩率与压缩速度平衡的最为关键的环节。该过程需要确定最优匹配串查找的方式、范围和深度,而查找的精细程度直接决定了压缩效果与计算开销。具体而言,如图3所示,最优匹配串查找可以抽象为索引更新、索引查找和字符串匹配3个主要阶段。各阶段在压缩过程中分别承担不同的功能,共同实现高效的无损压缩。从分阶段的视角出发,可以更清晰地揭示LZ77算法的运行机制与实现细节。

索引更新是最优匹配串查找的第1步,负责将前瞻缓冲区中的数据向搜索缓冲区滑动,以确保滑动窗口能够持续跟踪当前处理的数据。在LZ77算法中,搜索缓冲区的大小是有限的,当前瞻缓冲区的数据被编码输出后,这些数据会被移入搜索缓冲区,而过早的数据则被移出窗口。同时,用于查找候选匹配项(后续将其简称为候选项)的哈希索引结构的大小也是有限的,随着搜索缓存窗口的移动,索引信息

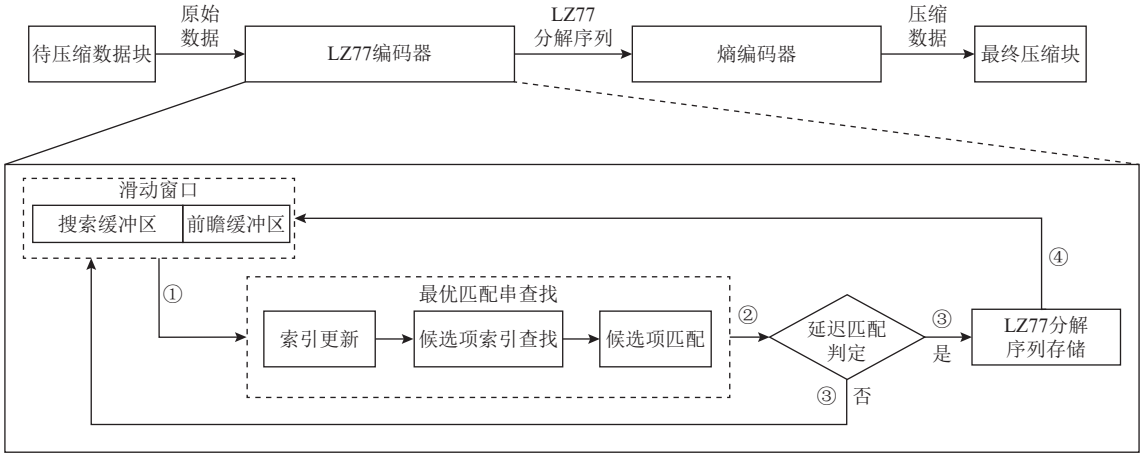


Fig. 2 Schematic diagram of ZSTD compression process

图2 ZSTD压缩流程示意图

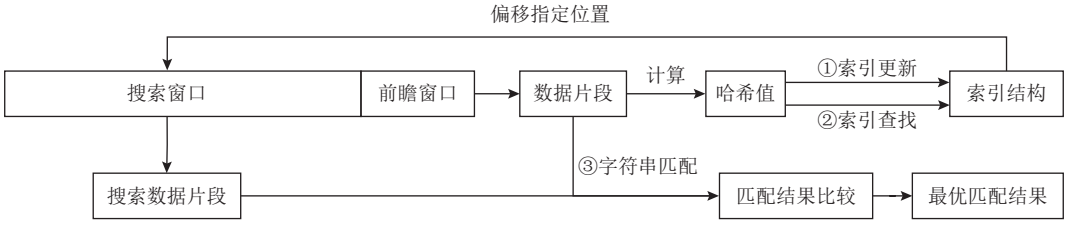


Fig. 3 Optimal matching string search process

图3 最优匹配串查找流程

数量往往会超过索引结构能够承载的表项数目。因此当新的索引信息需插入进来而索引结构中已经没有空闲位置时，必须根据淘汰替换策略移除旧的索引信息，为新索引信息腾出空间。由于冗余数据在 LZ77 分解序列中的表现形式为（偏移值，匹配长度）二元组，在匹配长度相等的情况下偏移值越小的候选项越具有压缩潜力，因此在索引结构中往往采取先进先出（first in first out, FIFO）策略进行淘汰更新，优先淘汰偏移值大的候选项而保留偏移值小的候选值。

索引更新的作用在于确保滑动窗口能够持续跟踪当前处理的数据，同时避免内存占用过高。索引更新还需要保证搜索缓冲区中所有数据的索引能够随滑动过程进行实时调整，从而在后续的索引查找中继续维持正确的匹配范围。这个过程是算法动态特性的关键，使得 LZ77 适用于流式数据处理等需要实时更新数据状态的场景。

索引查找是最优匹配串查找的关键步骤。其主要作用是通过滑动窗口中的搜索缓冲区定位前瞻缓冲区中的数据模式，为当前匹配模式提供有效的历史匹配项索引。在这一过程中，算法需要在搜索缓冲区中快速查找可能与前瞻缓冲区数据相匹配的字符串，为后续的字符串匹配提供历史数据索引。为了提高效率，实际应用中通常会使用哈希表、链表或平衡树等数据结构来存储搜索缓冲区的内容，从而加速匹配的查找过程。索引查找的核心目的是减少遍历搜索缓冲区所需的时间开销，确保算法在较大的窗口范围内也能高效完成匹配任务。

ZSTD 针对不同压缩级别分别提供了 3 种最优匹配串查找结构：哈希链、行哈希（row-Hash）以及二叉平衡树，每一种结构都对应着不同的索引查找方式。在 ZSTD 中等级别压缩方案中，采用了基于行哈希的索引查找机制，其结构如图 4 所示。该机制引入分层过滤策略，将表示数据串前缀特征的哈希值划分为 2 部分：高位作为 *row* 值，低位作为 *tag* 值。压缩过程中，系统首先利用 *row* 值定位到对应的哈希行，完成第 1 层筛选；随后，借助 SIMD 指令对该哈希行中的 *tag* 值进行快速验证，以高效判断是否存在与当前数据串拥有相同前缀的历史数据。最终，符合条件的候选项将作为后续字符串匹配操作的输入，从而提升匹配效率与压缩性能。

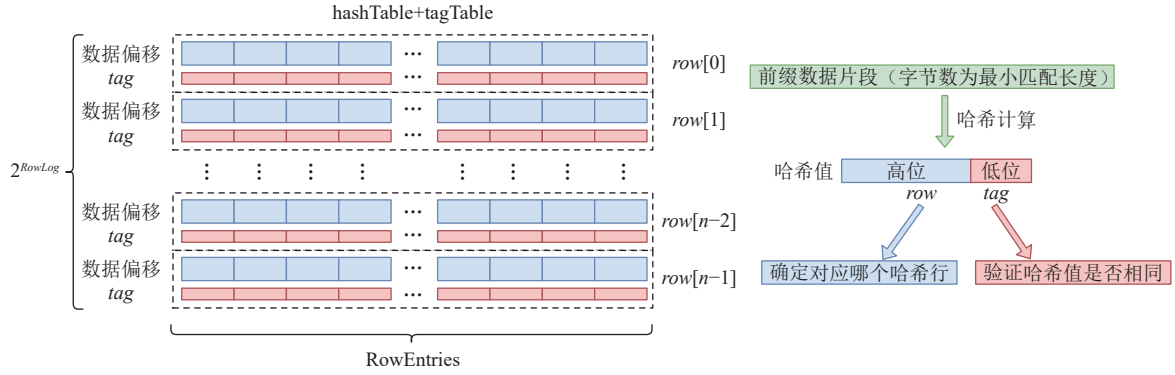


Fig. 4 Schematic diagram of row-Hash index structure

图 4 行哈希索引结构示意图

字符串匹配是最优匹配串查找的核心步骤。它的作用是将前瞻缓冲区中的待处理数据，与索引查找获得的搜索缓冲区数据分别逐一进行内容比对，从而获得各自匹配字符串的长度和偏移位置信息。在这一过程中，算法从搜索缓冲区的索引位置开始逐字符比较，向前瞻缓冲区延伸，直到找到最长的匹配或出现不匹配的字符。字符串匹配的结果通常包括 3 个关键信息：偏移量（*offset*），表示匹配模式距离当前位置的距离；匹配长度（*length*），表示最长匹配的字符数；下一个未匹配的字符（*next character*）。这些信息会被编码为三元组 (*offset*, *length*, *next character*)，作为压缩输出的一部分。字符串匹配的质量直接影响压缩效率和压缩率，尤其是在重复模式较多的情况下，找到更长的匹配能够显著提升压缩效果。

索引更新、索引查找和字符串匹配这 3 个过程是最优匹配串查找的核心环节，各司其职并相互配合。索引更新确保滑动窗口能够实时反映当前数据流的状态，索引查找通过高效的数据结构定位潜在的匹配区域，字符串匹配则对匹配模式进行精确分析并生成压缩编码。3 个步骤的有序衔接使得 LZ77 算法能够在处理冗余数据时实现高效压缩。

1.2.2 延迟匹配策略

为了进一步提升压缩率，ZSTD 还采用了延迟匹配策略，其目的是基于延迟匹配期望获得更好的匹

配效果。因此算法在得到当前处理位置的最优匹配结果后，通常不会直接使用该匹配串进行压缩处理，而是继续对后续相邻点位进行最优匹配串查找，并将查找结果与先前已找到的最优匹配串进行比较，选择收益最高的匹配点位进行压缩编码以获取更优的压缩率效果。

延迟匹配策略中最为关键的参数是延迟匹配深度，它决定了需要对后续相邻的几个点位再进行最优匹配串查找。ZSTD 中等压缩级别方案所使用的 lazy2 策略，即深度为 2 的延迟匹配策略，需要将当前点位的搜索结果与后续相邻的 2 个点位的搜索结果进行比较。若当前搜索结果并非最优，则继续往后搜索最佳点位，直到当前处理位置的搜索结果优于后续相邻的 2 个处理位置为止。

图 5 给出了 lazy2 策略的有限状态机图，其核心机制通过动态调整匹配基准位置实现最优压缩决策：以当前处理位置指针 *ip* 为基准，$depth_0$, $depth_1$, $depth_2$ 分别对应 *ip*, *ip*+1, *ip*+2 的连续偏移位置。首先在 $depth_0$ 处进行搜索匹配得到最优匹配串 M_0，并在 $depth_1$ 处进行搜索匹配得到最优匹配串 M_1。若 M_0 的压缩效果优于 M_1，则继续在 $depth_2$ 处进行搜索，否则将当前处理位置 *ip* 向后移动 1 个位置至原先 $depth_1$ 对应的位置，同时将 M_1 赋值给 M_0，即将原先的 $depth_1$ 视为新一轮延迟匹配中的 $depth_0$ 继续进行延迟匹配操作。同理，若 M_2 的压缩效果优于 M_0，则将当前处理位置 *ip* 向后移动 2 个位置至原先 $depth_2$ 对应的位置，并将 M_2 赋值给 M_0，将原先的 $depth_2$ 视为新一轮延迟匹配中的 $depth_0$ 继续进行延迟匹配操作。通过逐级比较匹配串压缩效果并动态调整基准位置，最终收敛于满足 lazy2 策略约束的最优压缩位置。

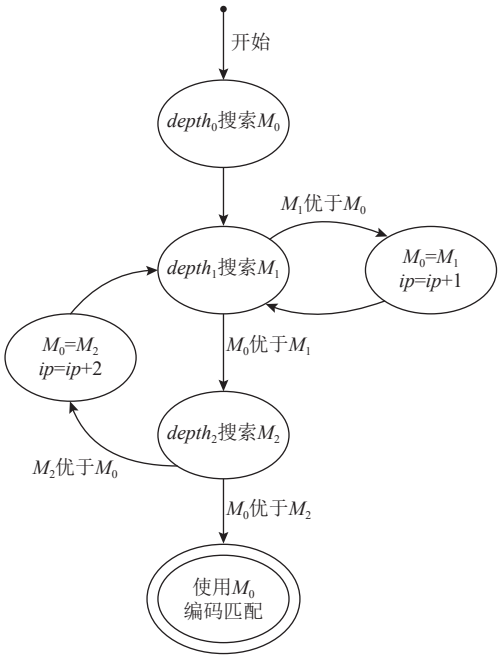


Fig. 5 lazy2 strategy finite state machine diagram

图 5 lazy2 策略有限状态机图

1.3 相关工作

近年来，为应对多样化的压缩场景需求，研究者对多种无损压缩算法开展了系统评估和改进优化。Giri 等人[18] 对 LZ77 与 LZ78 算法在数据压缩与信源编码中的机制与应用效果进行了实验对比，指出两者在压缩率与算法复杂度之间存在权衡关系。Aruna 等人[19] 则比较了 Huffman, LZ77, DEFLATE[20] 等算法在不同文件大小下的适应性，为实际应用中算法选择提供了基础参考。然而，上述研究主要聚焦于算法本身的通用性评估，未充分考虑算法执行过程中计算瓶颈对压缩性能的影响，尤其在高压缩率配置下存在实际吞吐瓶颈的问题。

为提升压缩效率，不少研究对字典压缩与熵编码算法进行了优化。Chen 等人分别提出了基于 LZ4 的 LZ4r[21] 算法及基于 Snappy 的 SnappyR[22] 算法，均在不显著降低压缩速度的前提下提升了压缩比。然而，由于其所基于的原始算法本身更侧重压缩速度，即便在改进后压缩率有所提升，仍难以满足如文件共享、虚拟化等对压缩率要求较高的应用场景。Barina 等人[23] 提出的 x3 算法在压缩比方面与 ZSTD 和 Brotli[24] 相当，但该算法引入了算术编码等复杂机制，导致其在执行开销和硬件适配性方面表现不佳，不适用于轻量化或实时性要求较高的场景。

在熵编码方面，Cheng 等人[25] 为规范哈夫曼编码提供了一种新方法，存储规范霍夫曼树每一层中的内部节点数量，并根据内部节点之间的特定属性对数字串进行紧凑编码。Köppl 等人[26] 提出了一种基于文本前缀共词典序的 Lempel-Ziv 因子距离高阶熵编码方法，该方法选择的偏移趋向于接近 *k* 阶的经验熵，在高阶熵较小的数据集上优于最右偏移和位最优 LZ 解析方法。Dinklage 等人[27] 则在支持超宽寄存器的硬件环境下探索了熵编码的加速实现。这些工作多侧重编码阶段的压缩率与空间性能，缺乏与 LZ 匹配策略协同优化的研究。

与此同时，面向 ZSTD 的硬件化路径也在通过算法与架构协同优化的方式，缓解控制流依赖与大滑动窗口带来的加速瓶颈。BeeZip2[28] 将元历史匹配、并行哈希与共享匹配处理单元结合，并辅以延迟匹配策略，在保持与软件相当压缩比的前提下显著提升吞吐并改善单位面积效率，表明对 LZ77 类算法进

行软硬件协同设计可有效突破单纯软件优化的性能边界。

相较于上述研究，本文所提出的 MLRS 与 ESLM 立足于 ZSTD 中等级别压缩配置下的实际瓶颈，从缓存命中行为与匹配计算开销出发，对匹配过程本身进行了结构性优化。区别于 LZ4r, SnappyR 等更侧重压缩率提升的工作，本文关注在压缩率保持的前提下，实现压缩速度的显著提升，并进一步验证其在鲲鹏 920 等国产多核平台上的适配性。相关实验表明，本文方法不仅在一定程度上缓解了压缩过程中末级缓存访问数据缺失率高的问题，同时提升了匹配效率，为 LZ77 类压缩算法在高压缩率场景中的实时性优化提供了新思路。

2 研究动机

本文选择 ZSTD 压缩级别 12（ZSTD level 12）中的 LZ77 算法作为优化对象，其动机如下：首先，ZSTD 是由 Facebook 开发的高性能通用压缩工具，相较于其他同类工具，其在压缩率与压缩/解压速度方面均表现出色。ZSTD 提供丰富的核心参数配置接口，预置了 22 个压缩级别（level 1~22），其中较高级别通过引入更复杂的匹配与编码策略，力求在多样化应用场景中实现更高的压缩率。凭借其卓越的压缩性能与良好的配置灵活性，ZSTD 已广泛应用于文件共享、虚拟化等对实时性和压缩比均有较高要求的场景。在实际部署中，ZSTD 的中等压缩等级（level 8~12）通常被视为性能与压缩率的权衡基准。为满足场景需求，压缩算法需在压缩率不低于该基准线的前提下，尽可能维持较高的压缩吞吐量。相比之下，LZ4, Snappy 等轻量化压缩工具虽然压缩速度更快，但在压缩率方面难以达到 ZSTD 中等压缩级别的表现。因此，本文在后续的对比实验中不会将这些算法作为对照对象。

然而，随着实时压缩场景对压缩吞吐量要求的持续提升，当前 ZSTD 的中等级别压缩方案已难以充分满足需求。因此，本文聚焦于优化 ZSTD level 12 方案，目标是在基本保持其压缩率的前提下显著提升压缩吞吐量，从而更好地服务于实时压缩场景。ZSTD 融合了 LZ77 字典压缩算法与熵编码算法（哈夫曼编码、FSE）共同实现高效的无损压缩，但实验测试表明，ZSTD 主要的时间开销来源于 LZ77 算法的执行过程。如图 6 所示，ZSTD 中等压缩级别方案中 LZ77 模块的执行耗时占比显著，成为当前系统压缩速度的主要制约因素。因此，本文将重点针对 ZSTD 中 LZ77 算法的执行效率进行优化。

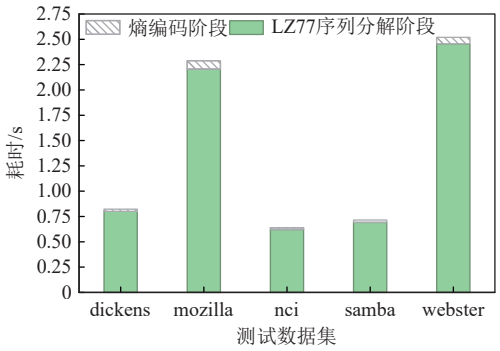


Fig. 6 Time consumption of different algorithms in ZSTD medium compression level scheme in execution stages

图 6 ZSTD 中等压缩级别方案中不同算法的执行阶段的耗时

在 ZSTD level 12 方案中，为追求更高的压缩率，采用了大历史窗口与延迟匹配策略。大历史窗口通过提供更丰富的历史数据上下文，增加了查找到更长、更优匹配串的概率；延迟匹配策略则通过比较当前位置与后续潜在位置的匹配结果，选择全局更优的匹配点进行编码。这 2 种策略均有效提升了压缩率。然而，实验观测表明，ZSTD 的 LZ77 算法执行过程中面临着 2 个挑战：

1）大历史窗口导致的缓存不友好性。现代 CPU 的 L2 缓存容量难以容纳整个大历史窗口数据及其相关数据结构。这导致压缩过程中缓存未命中率显著升高，频繁的随机内存访问造成严重的性能损失。

2）延迟匹配策略引入的冗余计算。延迟匹配策略虽提升了压缩率，但其实现中存在搜索路径冗余问题，即相邻位置的匹配搜索操作存在大量重叠计算。这些冗余搜索带来了额外的无效计算开销，最终导致压缩吞吐量下降。

为此，本文分别提出 MLRS 与 ESLM，旨在克服上述挑战，最终实现在保持高压缩率的同时显著提升压缩吞吐量的目标。

3 MLRS 设计实现

3.1 优化动机

在应用 ZSTD 中等级别压缩方案对不同类型数据集进行压缩调研测试中（测试环境详见 5.1 节）发现：

1）最优匹配串查找过程中最耗时的操作来自索引更新或字符串匹配。分析表明，索引更新与字符串匹配相较于索引查找需要访问大范围的历史数据，

而当下 CPU 无法实现对历史窗口数据及其相关数据结构的全量缓存,因此在压缩过程中会频繁出现缓存未命中的情况,随机访存导致压缩性能受损,最终造成较高的时间开销。

2)压缩过程中绝大多数匹配项来源于历史窗口的近端区域,且这部分匹配项成为最优匹配项的概率明显高于来自远端的数据区域。在 ZSTD 的压缩过程中,将每轮匹配所获得的全部候选项根据其相对于当前处理位置的偏移值大小平均划分为 4 个候选等级。其中,第 0 级表示偏移值最小的一组候选项,即距离当前处理位置最近;第 3 级则表示偏移值最大的一组候选项,即最远的匹配来源。对最优匹配串在 4 个等级候选项中的分布情况进行了统计,测试结果如图 7(a)所示。实验结果表明,在排除无法找到满足最小匹配长度的有效匹配项的情形后,绝大多数最优匹配串来源于第 0 级的候选项。同时,随着候选项偏移值的增大,其成为最优匹配串来源的概率显著下降。

3)原有压缩方案中,对于不同偏移值范围内的候选项进行字符串匹配的开销差距不大。在测试中,将所找到的候选项(其数量受最大搜索深度参数 *searchLog* 限制)按照偏移值均分为 4 级,分别统计并计算各级候选项的字符串匹配开销占比。测试结果如图 7(b)所示,除了相对偏移值最小一级候选项的字符串匹配开销略高于其他级别外,其他级别的时间开销占比基本持平。

上述测试结果说明,压缩过程中大范围的数据访问易造成缓存频繁未命中的问题从而影响压缩性能。同时,在 ZSTD 中等级别压缩方案的最优匹配串查找过程中,存在历史窗口远端候选项有效比率与搜索开销占比不成正比的情况。因此,本文提出基于分级思想的多级区域搜索策略,通过候选项级别划分、分级匹配以及上、下限阈值机制来灵活控制每次匹配查找的深度,以实现高效的最优匹配串查找操作。

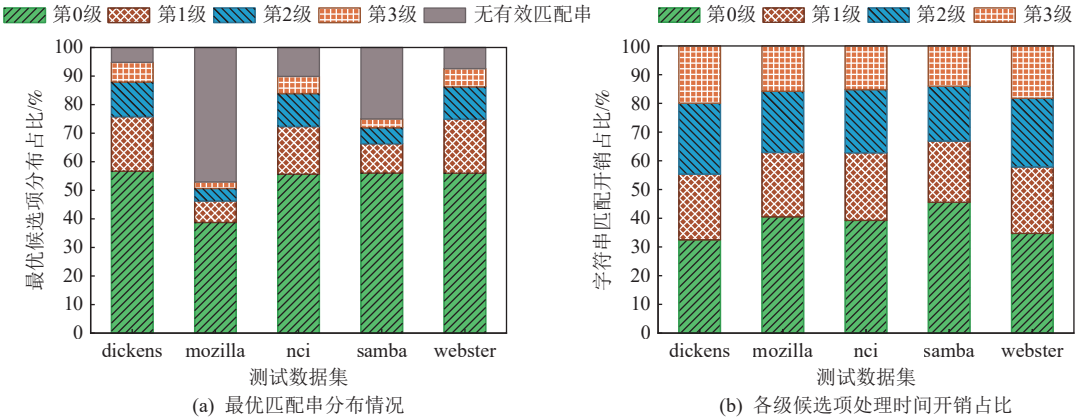


Fig. 7 MLRS optimization motivation test results

图 7 MLRS 优化动机测试结果

3.2 基于分级匹配与动态阈值控制的搜索优化机制

如第 2 节分析,LZ77 算法中采用大历史窗口对于当下 CPU 缓存并不友好,在进行最优匹配串查找的过程中可能需要访问到的历史数据范围远大于其缓存大小,从而造成频繁缓存未命中现象。为缓解这一问题,最直接的思路是对每次最优匹配串查找时所需历史数据的范围加以限制,而这一目标可通过调整匹配搜索深度予以实现。

在 ZSTD 中采用的是静态搜索深度,即参数 *searchLog* 在压缩前就被确定下来并且在整个压缩过程中保持不变。然而,若直接将搜索深度限制为一个较低的固定值,虽然能够减少需要访问的数据范围,但会造成大历史窗口中数据利用率低的问题,导致压缩率的损失。因此,本文考虑采取“分级匹配”与“阈值控制”相结合的方式,来控制单次最优匹配串查找操作的数据访问范围大小。该方式能够根据数据特征以及当前压缩情况,更加灵活地调整搜索深度,从而实现更高效的压缩处理。

从具体实现机制来看,最优匹配查找过程首先对所找到的候选项进行级别划分。随后,依次对最高优先级别的候选项进行匹配,以获得该级别的最优匹配结果。然后,将该结果与预设阈值进行比较和评估:若当前级别的匹配结果已满足阈值要求,则直接采用该结果,无需对后续低优先级候选项进行匹配;否则,继续对次低级别候选项执行相同操作。该方法通过引入级别划分与阈值设置来调控搜索深度,其能否在保持压缩率基本稳定的前提下有效提升最优匹配串查找效率,关键在于所采用的分级标

准、分级粒度以及阈值设置的合理性。

1）分级标准

对于分级标准，结合前期调研测试中得到的最优匹配串大部分来自历史窗口近端候选项的结论，决定以候选项的偏移值大小作为分级标准。考虑到候选项在偏移值范围内的分布通常不均匀，并且结合行哈希的表项更新机制与候选项选取方式，直接按照候选项偏移值大小进行固定范围分级的效率较低。因此，本文选择将候选项对应表项在哈希行中的插入顺序作为其分级标准。由于行哈希采取 FIFO 更新策略，越新插入的表项对应历史数据的偏移值越小，因此可直接根据哈希表项的插入顺序对候选项分级。最新插入的一批表项对应候选项被赋予最高匹配优先级，次新插入的被赋予次高匹配优先级，依此类推，直至对所有候选项完成分级。

2）分级粒度

在候选项分级设计中，若分级粒度过小，例如每个级别仅包含 1 个候选项，则在匹配操作中需对每一个候选项单独进行匹配并与阈值比较，这将引入大量的结果判断与分支开销，难以有效提升压缩性能。相反，若分级粒度过大，例如将所有候选项仅划分为 2 个等级，则每次进行最优匹配时至少需要遍历一半的候选项，难以体现基于阈值机制所带来的匹配判断的灵活性与效率优势。此外，考虑到候选项的索引查找过程中使用了 SIMD 指令对候选项掩码值进行加速计算，且 SIMD 指令可实现对 16 个表项的 *tag* 值并行验证，因此若每一级候选项的数量设为 16 的倍数，将有助于充分发挥 SIMD 指令的并行处理能力。综合考虑匹配效率与硬件加速支持，本文最终采用将 64 个表项根据偏移值大小平均划分为 4 个等级的分级策略，即每个等级包含 16 个候选项。

3）阈值设置

对于阈值设置，考虑同时设置下限阈值与上限阈值，分别用于控制匹配质量的最低预期与搜索提前终止的上界判断。

下限阈值主要用于识别当前候选级别中匹配质量较差的情形。具体而言，当某一级别中获得的最优匹配结果未达到设定的下限阈值时，结合前文关于匹配分布的分析，可以推断后续更远位置的候选项产生更优匹配的概率较低。因此，系统可在此时终止后续级别的匹配操作，从而减少无效匹配尝试所带来的计算开销。

上限阈值的作用则在于判断当前匹配结果是否已具备足够的压缩效果。当某一级别中得到的最优匹配结果达到或超过上限阈值时，说明该匹配已满足压缩质量要求，同时考虑到更远候选项通常匹配概率较低，继续搜索所获得的收益有限。此时，可直接采用当前的匹配结果进行压缩编码，并跳过后续级别的匹配过程，以进一步提升整体匹配效率。

根据阈值在压缩过程中是否发生改变，可以将其分为静态阈值和动态阈值 2 种类型。静态阈值在整个压缩过程中保持不变，其优势在于压缩过程中无需进行调整，因此可有效避免由阈值频繁更新带来的额外计算开销。然而，静态阈值的缺点是数据适应性较差，无法根据当前匹配情况动态调整阈值大小，在处理数据特性未知或差异较大的输入时，难以取得良好的优化效果。相比之下，动态阈值的机制是依据实时匹配搜索结果对阈值进行调整，无需依赖对输入数据的先验假设，能够更准确地反映压缩过程中的实际最优路径，提升匹配判断的适应性与灵活性。

综合考虑 2 类阈值设定方式的优缺点，并结合上限阈值与下限阈值在压缩过程中的不同功能定位，本文所提出的优化策略采用了动态上限阈值与静态下限阈值的阈值设定方式。其中，上限阈值的动态调整策略是阈值机制能够较好发挥作用的关键。为了适应数据集的多样性以及数据本身的局部特性，需要针对上限阈值设计一个动态调整策略。

在 LZ77 压缩中，由于滑动窗口机制利用了数据的空间局部性，因此考虑采用以前一个窗口中匹配项长度的分布为参考指标来动态调整上限阈值。具体设计思路如图 8 所示，如果前一个窗口中大部分匹配项的长度已达到上限阈值，则可适当提高该阈值，从而允许更多的匹配项参与压缩过程，以挖掘更多潜在的压缩收益。相反，若前一个窗口中大多数匹配项的长度未达到上限阈值，则将上限阈值适当降低，以避免过长的搜索范围导致的最终无法充分发挥阈值机制带来的性能提升优势。

本文设计的最优匹配串查找流程如图 9 所示。与 ZSTD 原有方案中采用固定搜索深度的策略不同，该策略首先根据候选项的插入顺序将其划分为 4 个优先级等级，随后按照优先级从高到低依次对各级候选项执行匹配操作，并结合各级最优匹配串情况与阈值机制判断是否提前终止搜索，最后根据本轮匹配结果对上限阈值进行动态调整。

1）在 ZSTD 哈希表中，哈希行用于存放具有相同哈希值的多个候选位置，为节省空间并支持快速替换，行内采用环形存储结构。同时，基于 LZ77 编码

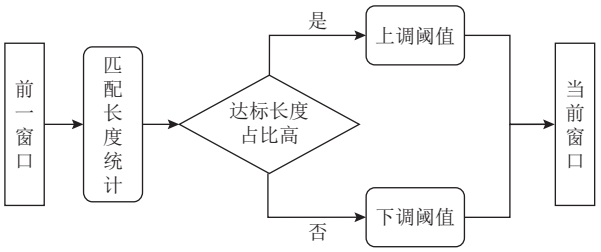


Fig. 8 Upper threshold adjustment strategy

图 8 上限阈值调整策略

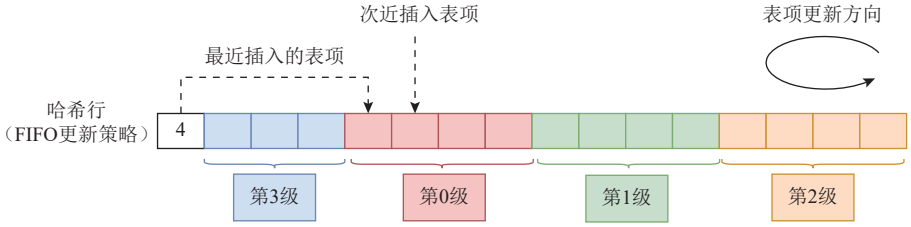
原理，在相同匹配长度下，历史窗口中距离较近的候选项更有压缩潜力。因此，哈希表的更新采用 FIFO 替换策略，并在第 0 表项中记录该行最近插入表项的索引位置，以便于快速定位最近插入表项、加速哈希表的更新以及候选项的级别划分。例如，图 9(a)中的“4”表示该哈希行最近插入的候选项位于第 4 个表项。

哈希行更新时，首先根据第 0 表项快速定位该行最近插入表项的索引；并沿环形存储的更新方向前进 1 个位置，确定新表项的插入位置；随后，插入新表项并按 FIFO 策略替换最旧表项(如该位置已被占用)；最后，更新第 0 表项为新的插入位置。在对候选项进行级别划分时，由于行内数据按 FIFO 顺序排列，结合环形结构的更新方向，在定位到该行最近插入表项后，可直接推导出该行所有候选项在历史窗口中的距离顺序，从而以最近插入候选项为起点，快速完成候选项的多级划分。因此，每次新表项的插入会使候选项的级别划分整体沿更新方向前移 1 个表项。

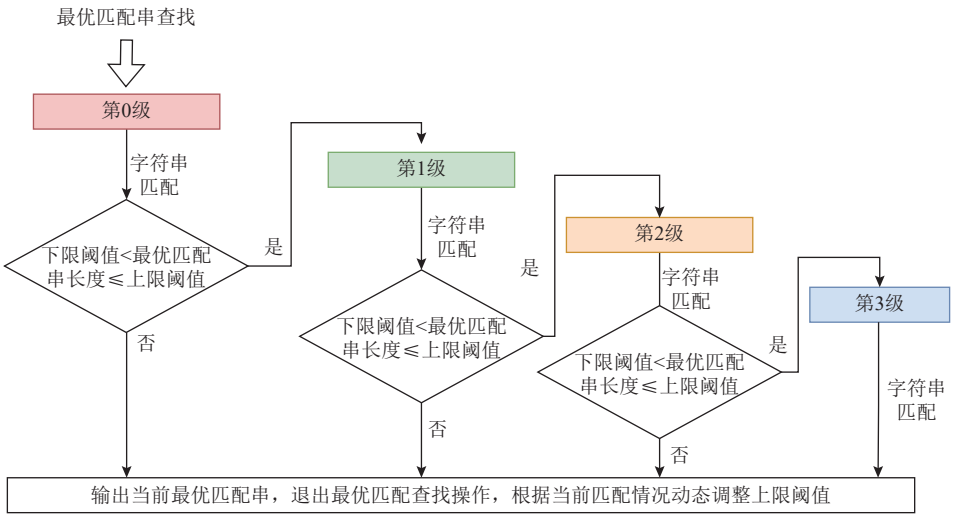
2)按照优先级依次对各级别候选项进行最优匹配串查找。若在某一级别中找到的最优匹配串已满足设定的匹配阈值(即匹配长度大于上限阈值或匹配长度未达到下限阈值)，则直接选用该匹配串对当前数据进行压缩编码，后续级别的候选项将不再参与匹配。若当前级别未能找到满足条件的匹配串，则继续处理下一级别，直到某一级别的最优匹配串满足阈值要求，或所有候选项均已完成匹配为止。

3)在完成 1 次最优匹配并输出匹配结果后，系统会根据本次分级匹配的情况，采用上限阈值调整策略对阈值进行动态更新。

通过分级匹配与动态阈值控制机制，系统能够根据实际匹配效果灵活调整搜索深度，从而将计算资源优先分配给匹配潜力更高的近端候选项，减少对远端候选项的无效开销，最终实现更加高效的最优匹配查找过程。



(a) 候选项级别划分



(b) 分级匹配流程示意图

Fig. 9 Schematic diagram of MLRS process

图 9 MLRS 流程示意图

4 ESLM 设计实现

4.1 优化动机

在对 ZSTD 中 lazy2 策略进行分析后发现，由于该策略至少需要对 3 个相邻位置依次进行最优匹配串查找操作，因此在查找过程中极易出现搜索路径冗余的情况。然而，最终的匹配结果只会选取其中的最优值进行压缩编码，即冗余的搜索路径不仅不会带来压缩率上的收益还会造成计算资源的浪费。为了进一步分析搜索路径冗余带来的性能影响，本文对原延迟匹配过程进行了调研测试，结果表明：

1）$depth_1$ 搜索结果无效比率高。如图 10(a)所示，在采用 ZSTD 中等级别压缩策略对不同类型数据集进行测试时，$depth_1$ 搜索结果与 $depth_0$ 或 $depth_2$ 结果发生重叠的比例较高。具体而言，若 $depth_1$ 所得的最长匹配串为 $depth_0$ 最长匹配串的超集，或为 $depth_2$ 结果的子串，则认为其匹配存在重叠。统计显示，该类重叠匹配在各类型数据集中均超过 55%。此外，$depth_1$ 虽未与其他级别匹配重叠，但其匹配结果最终未能成为最优匹配的比例也普遍高于 15%。综合来看，$depth_1$ 在 75% 以上的匹配尝试中未能提供有效的最优匹配结果，整体贡献有限。

2）$depth_1$ 搜索开销显著。如图 10(b)所示，在不同类型数据集上的压缩过程中，$depth_1$ 匹配所引入的计算时间占压缩总耗时的比例普遍超过 18%。这一比例表明 $depth_1$ 搜索在整个 LZ77 压缩过程中占据了相当可观的计算资源。若能通过优化策略减少 $depth_1$ 的搜索负担，理论上可将其计算资源转化为压缩速度的提升，从而改善整体压缩效率。

综上所述，对 $depth_1$ 的搜索匹配操作不仅在多数情况下无法显著提升压缩质量，还引入了较高的计算成本。因此，若能够充分挖掘 $depth_0$ 与 $depth_2$ 之间的冗余信息，构建用于近似替代 $depth_1$ 搜索的策略，有望在压缩率基本保持稳定的前提下，有效降低冗余计算，提高压缩性能。

4.2 基于扩展搜索的延迟匹配策略优化设计

针对 lazy2 策略中存在的冗余搜索问题，结合前文调研与实验分析结果，本文提出一种优化策略，旨在不显著影响压缩率的前提下减少不必要的计算开销。在原 lazy2 策略中，如图 11(a)左图所示，系统会在第一轮延迟匹配阶段对 $depth_1$ 执行完整的匹配搜索操作。然而前述测试表明，$depth_1$ 搜索结果有效比率较低且计算成本较高。为此，本文针对 lazy2 提出一种基于扩展搜索的 lazy1.5 优化策略。如图 11(b)左图所示，该策略在第 1 轮延迟匹配过程中不再对 $depth_1$ 执行完整的匹配搜索。取而代之的是，利用 $depth_0$ 和 $depth_2$ 匹配查找过程中附加的扩展搜索操作，对 $depth_1$ 的搜索结果进行近似估计，并将所得的扩展匹配结果作为 $depth_1$ 的替代输入，参与后续的多深度匹配结果比较。

在原 ZSTD level 12 压缩方案中，匹配搜索按候选项在哈希行中的插入顺序(由近及远)依次进行，且仅以匹配长度作为评估标准，最终选取最长匹配串作为最优结果。然而，LZ77 压缩输出为由(匹配项偏移值，匹配串长度)二元组及字面量组成的序列，忽略偏移值可能导致编码成本偏高，进而影响压缩性能。为此，本文在匹配搜索阶段引入偏移值作为辅助指标，参考 lazy2 策略中的评估方式，构建匹配长度与偏移值的联合评判机制，以统一搜索过程与延迟匹配过程中的最优匹配选择标准。该策略有助于提升扩展搜索结果的有效性，在以近似方式替代

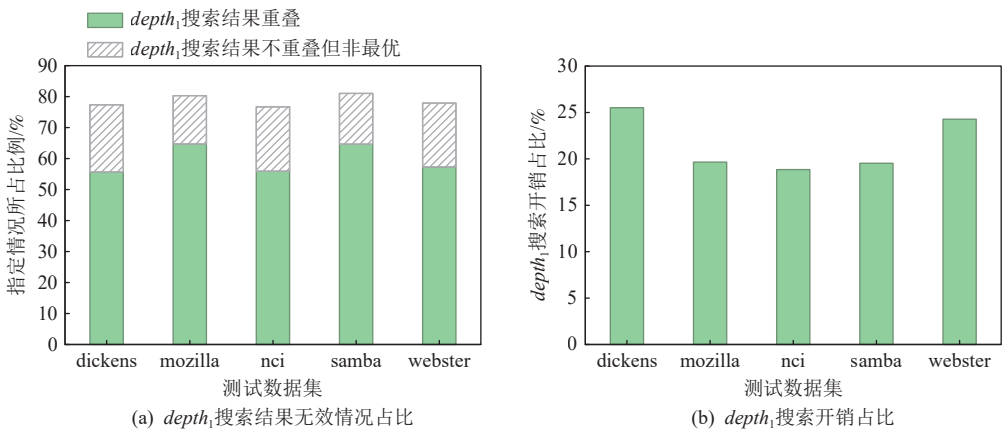
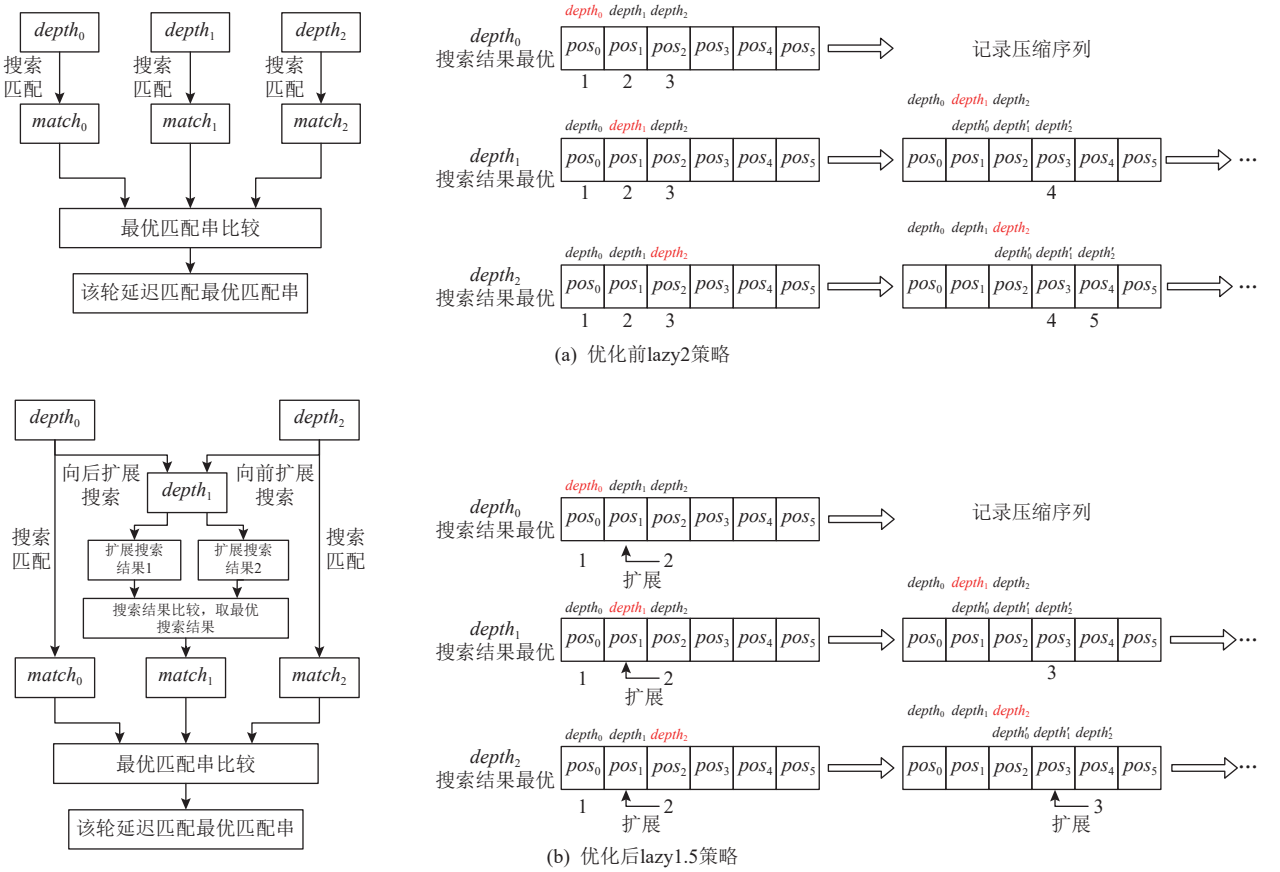


Fig. 10 ESLM optimization motivation test results

图 10 ESLM 优化动机测试结果



(a) 优化前lazy2策略

(b) 优化后lazy1.5策略

Fig. 11 Comparison on before and after optimization of lazy matching strategy

图 11 延迟匹配策略优化前后对比

完整 $depth_1$ 搜索的同时，尽可能减少压缩比的损失，提升整体压缩效率。

为支持 lazy1.5 策略中的扩展搜索机制，在每完成一次匹配操作后，还需额外比较当前处理串与匹配串前/后一个字符是否相同。若匹配成功，则视为该操作通过扩展搜索获取了相邻位置的潜在匹配结果。随后，从所有扩展搜索得到的匹配结果中选取最优项作为相邻位置的近似匹配结果来参与后续的延迟匹配阶段。进一步结合 $depth_0$ 与 $depth_1$ 的位置关系分析可知，$depth_0$ 向后一位的扩展搜索仅能产生不优于原始 $depth_0$ 匹配的结果。因而在方案实现中，lazy1.5 策略仅对 $depth_2$ 执行向前一位的扩展搜索，用于近似获取 $depth_1$ 的匹配结果，以提升最优匹配串查找效率并降低冗余计算。

1)从压缩率的角度来看，lazy1.5 策略通过扩展搜索近似替代 $depth_1$ 的匹配结果，对整体匹配质量的影响较小，不会显著干扰最优匹配的选择。该策略在匹配过程中始终保证对 $depth_0$ 与 $depth_2$ 执行完整的最优匹配搜索，对于 $depth_1$，则复用 $depth_2$ 的搜索路径并在此基础上进行有限扩展以获得近似结果。在最不利的情况下，若所有由扩展搜索得到的 $depth_1$ 结果均为非全局最优，则最终匹配仅由 $depth_0$ 与 $depth_2$ 的结果决定，此时匹配质量等价于深度比较间隔为 2 的 lazy1 策略(即仅比较 $depth_0$ 与 $depth_2$ 的最优匹配)。因此，对于任意输入数据集，lazy1.5 的匹配质量始终优于或至少等于深度比较间隔为 2 的 lazy1 策略。

在典型数据集上，如图 10(a)所示，原 lazy2 策略中 $depth_1$ 匹配结果与 $depth_0$ 或 $depth_2$ 出现重叠的比例超过 55%；在剩余不重叠的情况下，约有 15% 的结果并非全局最优。由此可见，在超过 75% 的匹配事件中，lazy1.5 的匹配质量与 lazy2 一致，不会受到影响。结合这一退化上界与统计结果，可以认为 lazy1.5 在匹配质量上的风险是受控且可预测的，从而在保持较高压缩率的同时降低了计算开销。

2)从压缩吞吐量的角度来看，第 1 轮延迟匹配往往无法直接产生满足延迟条件的匹配结果，需依据当前匹配情况在相邻位置继续执行最优匹配查找操作。lazy1.5 策略可利用扩展搜索在此过程中替代部分完整搜索操作，从而降低冗余计算开销。

图 11(a)右图展示了原 lazy2 策略在前 2 轮延迟

匹配中的完整搜索分布。首轮延迟匹配依次对 $depth_0$, $depth_1$, $depth_2$ 所对应的位置执行完整的最优匹配串查找，并根据结果确定该轮的最佳匹配点后进入下一轮延迟匹配：若 $depth_0$ 为最佳匹配点，则已满足 lazy2 策略要求，可直接记录压缩序列；若 $depth_1$ 为最佳匹配点，则将其作为下一轮延迟匹配的 $depth_0$（图中标记为 $depth_0'$），由于 pos_2 在首轮已完成最优匹配查找，该轮仅需对 pos_3 进行最优匹配查找，并与 pos_1 的结果比较以确定最佳匹配点；若 $depth_2$ 为最佳匹配点，则将其作为 $depth_0'$，此时需对 pos_3 和 pos_4 分别进行最优匹配查找，并与 pos_2 的结果进行比较，以确定最佳匹配点后继续延迟匹配，直至找到满足 lazy2 条件的位置。

图 11(b)右图则展示了 lazy1.5 策略在相同场景下通过扩展搜索减少完整搜索执行路径的过程。该策略在首轮延迟匹配中仅对 $depth_0$ 与 $depth_2$ 执行完整的最优匹配串查找，并在 $depth_2$ 查找过程中利用扩展搜索近似获得 $depth_1$ 的匹配结果。随后，将 $depth_0$ 的完整匹配结果、$depth_1$ 的近似匹配结果以及 $depth_2$ 的完整匹配结果进行比较，以确定最佳匹配点并进入下一轮延迟匹配：若 $depth_0$ 为最佳匹配点，则满足 lazy2 条件，可直接记录压缩序列；若 $depth_1$ 为最佳匹配点，则将其作为下一轮的 $depth_0$(图中标记为 $depth_0'$)，此时仅需对 pos_3 进行完整匹配查找，并与 pos_1 的结果比较以确定最佳匹配点；若 $depth_2$ 为最佳匹配点，则将其作为 $depth_0'$，不同于原 lazy2 策略，此时仅需在对 pos_4 进行完整匹配查找的同时，通过扩展搜索近似获得 pos_3 的匹配结果，并将其与 pos_2 的结果一同参与最佳匹配点的比较，即可确定最佳匹配点并继续进行延迟匹配，直至找到满足 lazy2 条件的位置。通过在部分位置以扩展搜索替代完整搜索，相较于原 lazy2 策略，lazy1.5 能够将前 2 轮延迟匹配的完整匹配查找次数由 4~5 次降至 3 次，在保持较高匹配质量的同时显著降低延迟匹配阶段的计算开销。

此外，lazy1.5 所采用的扩展搜索策略通过挖掘匹配路径间的重叠性，有效规避了低效冗余搜索，其优化思想具备良好的通用性，可扩展应用于其他基于相邻位置最优匹配查找的延迟匹配策略中。

5 实验结果及分析

5.1 实验说明

1)实验环境

本节的所有实验测试均在华为云基于 ARM 架构的弹性云服务器上完成，具体软硬件配置如表 1 所示。

Table 1 Experimental Environment Specifications

表 1 实验环境规格

组件	规格
CPU	Kunpeng 920 @ 2.6 GHz
一级缓存/KB	64
二级缓存/KB	512
内存/GB	16
操作系统	Ubuntu 22.04 server 64bit
Linux 内核版本	5.15.0-124-generic
gcc 版本	11.4.0

2)测试用例

Silesia Compression Corpus 是一种标准且广泛应用的无损数据压缩语料库。其数据类型多样、规模适中、公开获取，且具有良好的实验可比性，因而被文献 [2,19,21] 等多项压缩算法研究广泛采用，成为评估压缩率与压缩速度的重要基准数据集。本实验从中选取了 5 个具有代表性的数据子集：dickens, mozilla, nci, samba, webster, 涵盖文本、可执行文件、数据库和源代码等多种典型数据类型。表 2 列出了这 5 个数据集的基本信息，包括其数据类型及原始文件大小。

Table 2 Type and Size of Test Datasets

表 2 测试数据集的类型与大小

数据集	数据集类型	数据集大小/MB
dickens	英文文本	10.2
mozilla	可执行文件	51.2
nci	数据库	33.6
samba	源代码	21.6
webster	HTML 文本	41.5

3)性能指标

测试的主要性能指标包括压缩吞吐量与数据压缩比，表 3 给出了这些性能指标的计算方法。其中压缩吞吐量用于衡量压缩方案在单位时间内处理数据的能力，压缩吞吐量越高，说明该压缩算法在执行效率方面具有更优的性能表现；数据压缩比用于衡量压缩后数据相对于原始数据的大小比例，压缩比越高，说明压缩算法在数据冗余去除方面的效果越好，压缩效果越优。

4)基准线选择

当前主流压缩工具如 LZ4, Snappy 等，虽具有极高的压缩速度，但在压缩率方面普遍难以达到 ZSTD

Table 3 Performance Indicators and Their Calculation Methods

表 3 性能指标及其计算方法

性能指标	计算方法
压缩吞吐量	$\frac{\text{原始数据大小}}{\text{数据集压缩时间}}$
数据压缩比	$\frac{\text{原始数据大小}}{\text{压缩后数据大小}}$

中等压缩级别方案(如 level 12)的压缩效果。另一方面, zlib, Brotli 等工具虽具备相对更高的压缩率, 但在压缩吞吐量方面则不如 ZSTD, 难以满足实时性要求较高的应用场景。因此, 综合考虑压缩率与压缩速度的平衡性, 本文在实验部分选取 ZSTD level 12 作为唯一基准线, 并以该配置作为评估所提出优化策略性能改进效果的参考依据。

5.2 性能评估

为了验证所提出优化策略的有效性, 实验在 ZSTD level 12 压缩方案的基础上分别测试了不采用任何优化策略(原方案)、单独应用 MLRS(MLRS)、单独应用 ESLM(ESLM)、同时应用 MLRS 和 ESLM (MLRS+ESLM)的 4 种压缩方案在上述 5 种不同类型的典型数据集上的压缩效果, 通过比较分析应用优化策略后方案与原方案之间性能指标的差距验证所提出优化策略的有效性。测试结果如图 12 所示, 对于所提优化策略的有效性分析如下:

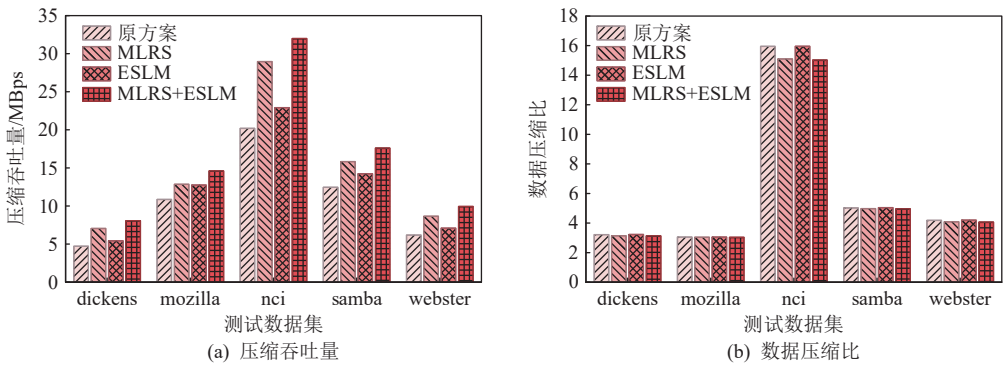
1)相较于原方案中对每个处理位置进行完全匹配查找的方式, MLRS 虽然引入了级别转换与动态阈值调整的额外开销, 但由于跳过了部分可能冗余的匹配查找操作, 在压缩吞吐量方面实现了 18.34%~49.50% 的提升。同时, 由于匹配搜索结果由原来的“实际最优”变为“预期最优”, 在压缩比方面产生了一定损失, 但数据压缩比仍保持在原来的 94.65%~99.58%。总体而言, MLRS 在提高压缩速度的同时, 保持了较高的压缩比, 证明了其在数据压缩过程中的可行性与有效性。

2)相较于 lazy2 策略, lazy1.5 策略(ESLM 在深度为 2 的延迟匹配策略的具体表现形式)在 $depth_0$ 与 $depth_2$ 的搜索过程中, 通过扩展搜索的方式获取了 $depth_1$ 的近似搜索结果。尽管此方法引入了一定的额外计算开销, 但该开销低于 lazy2 策略中对 $depth_1$ 进行完整搜索所产生的开销。因此, 相较于 lazy2 策略, 该优化策略在压缩速度上提升了 13.49%~17.46%。然而, 值得注意的是, lazy1.5 策略仅在对结构复杂、压缩难度较高的 mozilla 数据集测试中造成了数据压缩比的损失, 且仍保持为原来的 99.94%, 在对另外 4 个数据集进行压缩测试时, 其最终的数据压缩比还能够得到 0.03%~0.43% 的提升。这表明 lazy1.5 策略不仅能消除部分搜索路径冗余的情况从而提升压缩性能, 还可能通过扩展搜索的方式找到潜在更优的匹配串, 最终实现数据压缩比的提升。综上所述, 相较于原 lazy2 策略, lazy1.5 策略能够在所选取的大部分数据中实现压缩效率与压缩效果的双提升, 证明了其在实际应用中的有效性与可行性。

3)MLRS 与 ESLM 优化策略均旨在减少无效匹配或冗余搜索路径带来的计算开销, 以提高压缩速度。其中, MLRS 策略通过阈值机制控制搜索流程, 有效规避无效匹配对后续搜索的影响, 从而降低计算负担; 而 ESLM 策略则通过扩展搜索范围, 减少冗余搜索路径所导致的额外时间开销。尽管这 2 种策略可能会引入一定程度的压缩率损失, 但整体上能够显著提升压缩速度。将 2 种优化策略综合应用于 ZSTD level 12 压缩方案中, 相较于原方案, MLRS+ESLM 能够在保持 94.18%~99.8% 的压缩比水平的同时, 实现 34.53%~71.17% 的压缩性能提升, 验证了其在提升压缩效率方面的有效性。

5.3 MLRS 缓存优化测试

为了验证 MLRS 在压缩过程中确实能够起到缓



(a) 压缩吞吐量 (b) 数据压缩比

Fig. 12 Overall performance test

图 12 整体性能测试

存优化的作用，实验使用 Valgrind 的 Cachegrind 工具分别测试了原方案与应用 MLRS 优化方案后压缩过程中的缓存使用情况，通过分析其末级缓存访问的数据缺失率验证 MLRS 在压缩过程中的缓存优化能力。

测试结果如图 13 所示，基于分级思想的 MLRS 策略能够根据当前最优匹配串查找过程中候选项的匹配质量情况灵活地调整后续搜索深度，在一定程度上控制了单次查找匹配过程的数据访问范围，有效缓解了大历史窗口带来的缓存压力。相较于 ZSTD level 12 中的原搜索策略，MLRS 能够将压缩过程中末级缓存访问的数据缺失率降低 0.3~2.8 个百分点，验证了该策略在压缩过程中缓解缓存频繁未命中情况的有效性。

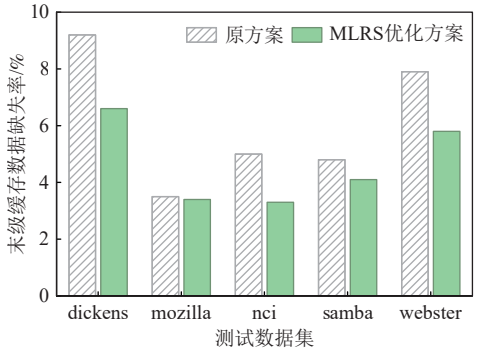


Fig. 13 Cache optimization effect of MLRS

图 13 MLRS 缓存优化效果

5.4 ESLM 参数敏感性与扩展性测试

1）ESLM 参数敏感性测试

考虑到 ZSTD 中等级别压缩方案中都选择采用 lazy2 策略，但它们的核心压缩参数并不相同，为了进一步验证 lazy1.5 优化策略在不同压缩参数下的适用情况，分别使用 lazy1.5 优化策略替代 ZSTD level 8~12 中原有 lazy2 策略并进行压缩性能测试，其中选择的测试数据集为由多个不同类型的文件组成的常用的压缩测试数据集 Silesia.tar。ZSTD 中等级别压缩方案的关键参数见表 4，其中 *windowLog* 为历史窗口大小的对数值，*hashLog* 为哈希表总表项数目的对数值，*searchLog* 为每次最优匹配串查找的搜索深度的对数值。

Table 4 Key Parameter Values of Compression Schemes for Different Levels in ZSTD

表 4 ZSTD 不同级别压缩方案关键参数值

ZSTD level	*windowLog*	*hashLog*	*searchLog*
8	21	20	4
9	22	21	4
10	22	22	5
11	22	22	6
12	22	23	6

最终得到的测试结果如图 14 所示，当历史窗口大小、哈希表大小以及搜索深度不断增大时，lazy1.5 优化策略较原 lazy2 策略：在压缩吞吐量方面仍能得到提升，并且提升幅度达到 11.61%~14.77%；在压缩比方面，优化效果明显，在 ZSTD level 8~10 中只会造成略微的损失，其中损失比率为 0.03%~0.14%，而在 ZSTD level 11~12 中则实现了压缩比的反超，并且提升幅度为 0.02%~0.04%。

2）ESLM 扩展性测试

在原 lazy2 策略中，lazy1.5 通过扩展搜索近似替代 $depth_1$ 的完整匹配搜索以降低计算量。基于 ESLM 设计原理，该思路可推广至更高延迟深度，从而构造相应的优化策略。例如，lazy2.5 由原 lazy3 改进，首轮保留 $depth_0$, $depth_2$, $depth_3$ 的完整匹配，并以扩展搜索近似获得 $depth_1$ 的匹配结果，后续阶段继续以扩展搜索减少完整查找；lazy3.5 由原 lazy4 改进，首轮保留 $depth_0$, $depth_2$, $depth_4$ 的完整匹配，并以扩展搜索近似替代 $depth_1$ 与 $depth_3$ 的匹配结果，后续同样优先采

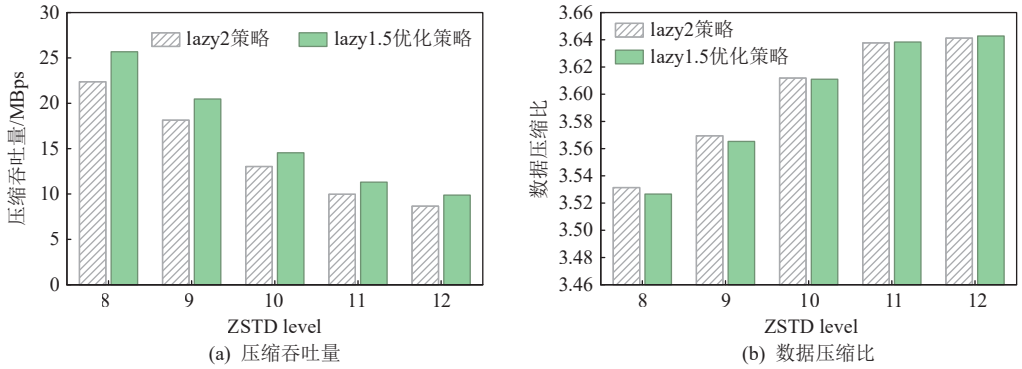


Fig. 14 Optimization effect of lazy1.5 under different compression levels of ZSTD

图 14 ZSTD 不同压缩级别下 lazy1.5 的优化效果

用扩展搜索以降低完整查找次数。为评估 ESLM 在不同 lazy 深度下的优化效果，本文分别对 lazy2, lazy3, lazy4 以及对应的 ESLM 优化策略进行了实验测试与结果比较，测试所用数据集为 Silesia.tar。

测试结果如图 15 所示，ESLM 在不同 lazy 深度配置下均展现出较为显著的优化效果，其中 lazy1.5 相较于对应 lazy 深度的原延迟匹配策略(lazy2)的提升效果最为显著。具体而言，lazy3.5 较 lazy4 在压缩比相近的情况下进一步提升可压缩吞吐量，lazy1.5, lazy2.5 较 lazy2, lazy3 在压缩比与压缩吞吐量 2 方面均实现了全面优化。测试结果表明，ESLM 在不同 lazy 匹配深度配置下均展现出良好的适应性与优化能力。

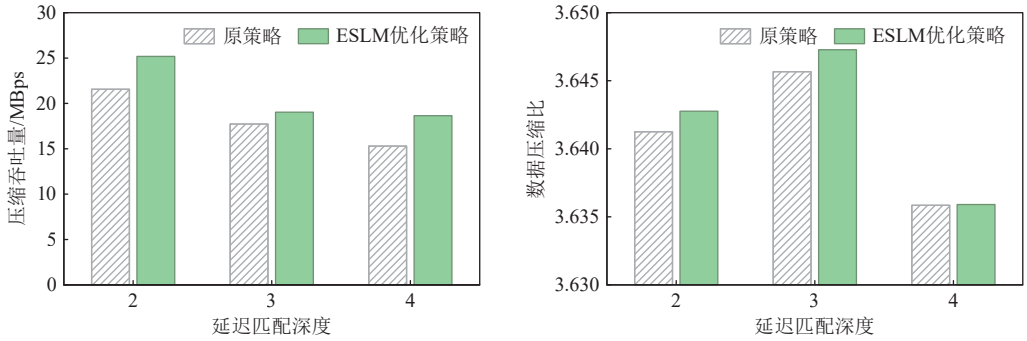


Fig. 15 Optimization effect of ESLM at different lazy depths

图 15 不同 lazy 深度下 ESLM 的优化效果

6 结 论

本文分别提出了基于分级思想的多级区域搜索策略(MLRS)与基于扩展搜索的延迟匹配策略(ESLM)。MLRS 在最优匹配查找过程中将候选项按等级进行组织，结合分级匹配与阈值控制，实现对后续搜索深度的动态调节。该策略能够有效剔除低价值候选项，提升搜索效率，并在一定程度上减少每次最优匹配串查找的数据访问量，从而缓解压缩过程中的缓存压力。ESLM 通过引入扩展搜索和近似替代技术，实现对已有搜索路径的复用，从而有效减少延迟匹配过程中的冗余计算，提升匹配处理效率。

经实验验证，在面向通用高压缩率需求的代表性压缩配置 ZSTD level 12 上，采用 MLRS 替代原有匹配搜索方案，能够有效降低压缩过程中末级缓存访问的数据缺失率。在保证平均压缩率达到 97.71% 的前提下，压缩吞吐量提升幅度达到 35.65%，显著优化了系统整体压缩性能。

同时，ESLM 在不同 ZSTD 压缩参数配置及 lazy 匹配深度设定下均展现出良好的适应性和稳定的优化效果。以 ZSTD level 12 为例，在将 ESLM 替代原始延迟匹配策略的场景中，系统压缩吞吐量平均提升 15.01%，同时压缩比也实现了 0.22% 的小幅提升，表明该策略在提升压缩效率的同时对压缩率几乎无负面影响，甚至具备一定增强效果。

在同时集成 MLRS 与 ESLM 策略后，系统表现出更为显著的性能优势。实验结果表明，该组合优化方案在保持平均压缩率为 97.36% 的基础上，压缩吞吐量整体提升高达 53.28%，进一步验证了优化方法在保障压缩质量的同时，具备高效性与实用性。

作者贡献声明： 韦芳宇完成实验设计、实验测试、论文撰写；陈天韵提出算法优化思路，参与实验测试分析；周洋指导论文思路与写作；谢雨来指导理论分析；王芳负责论文的整体指导。

参 考 文 献

[1] Wright A. Worldwide IDC global DataSphere forecast update, 2025–2029: Regional analysis[EB/OL]. [2025-06-14]. https://www.idc.com/getdoc.jsp?containerId=US53383425

[2] Collet Y. Zstandard: Real-time data compression algorithm[EB/OL]. [2025-06-14]. https://facebook.github.io/zstd

[3] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 1977, 23(3): 337−343

[4] Huffman D A. A method for the construction of minimum-redundancy codes[J]. Proceedings of the IRE, 1952, 40(9): 1098−1101

[5] Collet Y. Finite state entropy (FSE): Entropy coder based on ANS, with Huffman-comparable speeds and range-coder-like compression ratios [CP/OL]. [2025-06-14]. https://github.com/Cyan4973/FiniteStateEntropy

[6] Pavlov I. 7-Zip LZMA SDK[CP/OL]. 1999[2025-06-14]. https://www.

7-zip.org/sdk.html

[7] Collet Y. LZ4: Extremely fast compression algorithm[CP/OL]. [2025-06-14]. https://github.com/lz4/lz4

[8] Shannon C E. A mathematical theory of communication[J]. Bell System Technical Journal, 1948, 27(3): 379−423

[9] Google. Snappy: A fast compressor/decompressor[CP/OL]. [2025-06-14]. https://github.com/google/snappy

[10] Storer J A, Szymanski T G. Data compression via textual substitution[J]. Journal of the ACM, 1982, 29(4): 928−951

[11] Oberhumer M. LZO real-time data compression library[EB/OL]. [2025-06-14]. https://www.oberhumer.com/opensource/lzo/

[12] Choi S, Kim Y, Lee D, et al. Design of FPGA-based LZ77 compressor with runtime configurable compression ratio and throughput[C]//Proc of the 2019 IEEE Int Conf on Reconfigurable Computing and FPGAs (ReConFig). Piscataway, NJ: IEEE, 2019: 149583−149594

[13] Fang Jian, Chen Jianyu, Lee J, et al. An efficient high-throughput LZ77-based decompressor in reconfigurable logic[J]. Journal of Signal Processing Systems, 2020, 92(1): 931−947

[14] D'costa A K, Raksha K P, Vasanthi D R. Implementation of robust solid state drive controller using LZ77 compression and SHA-encryption technique[G]//LNCS 940: Proc of the 2018 Intelligent Systems Design and Applications. Berlin: Springer, 2020: 1−12

[15] Shportko A, Bomba A, Postolatii V. Rejection of the inefficient replacements while forming the schedule of the modified algorithm LZ77 in the process of progressive hierarchical compression of images without losses[C]//Proc of the 6th Int Conf on Computational Linguistics and Intelligent Systems (COLINS 2022). Berlin: Springer, 2022: 1594−1605

[16] Sun Xiuwen, Wu Di, Mo Da, et al. Accelerating Knuth-Morris-Pratt string matching over LZ77 compressed text[C]//Proc of the 2021 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2021: 372−372

[17] Gao Ruihao, Li Zhichun, Tan Guangming, et al. BeeZip: Towards an organized and scalable architecture for data compression[C]//Proc of the 29th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems, Volume 3. New York: ACM, 2024: 133−148

[18] Giri R K, Mishra A, Rongali A S, et al. An innovation analysis of LZ77 and LZ78 compression algorithms for data compression & source coding[C]//Proc of the 15th Int Conf on Computing Communication and Networking Technologies (ICCCNT). Piscataway, NJ: IEEE, 2024: 1−5

[19] Aruna P, Devi L Y, E D, et al. Analysis on text compression algorithms[C]//Proc of the 7th Int Conf on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). Piscataway, NJ: IEEE, 2023: 430−437

[20] Deutsch P. DEFLATE compressed data format specification version 1.3[EB/OL]. 1996[2025-06-14]. http://www.zlib.org/rfc-deflate.html

[21] Chen Rui, Xu Lihao. LZ4r: A new fast compression algorithm for high-speed data storage systems[C]//Proc of the 2023 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2023: 1−1

[22] Chen Rui, Xu Lihao. SnappyR: A new high-speed lossless data compression algorithm[C]//Proc of the 2023 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2023: 334−334

[23] Barina D, Klima O. x3: Lossless data compressor[C]//Proc of the 2022 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2022: 441−441

[24] Google. Brotli: A general-purpose lossless compression algorithm [CP/OL]. 2015[2025-06-14]. https://github.com/google/brotli

[25] Cheng Qi, Wei Yan, Lin Sian-Jheng, et al. Compressing the tree of canonical Huffman coding[C]//Proc of the 2022 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2022: 450−450

[26] Köppl D, Navarro G, Prezza N. HOLZ: High-order entropy encoding of Lempel-Ziv factor distances[C]//Proc of the 2022 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2022: 83−92

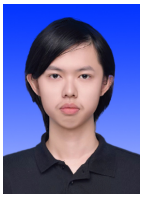
[27] Dinklage P, Fischer J, Kurpicz F, et al. Bit-parallel (compressed) wavelet tree construction[C]//Proc of the 2023 Data Compression Conf (DCC). Piscataway, NJ: IEEE, 2023: 81−90

[28] Gao Ruihao, Shi Shunchen, Li Xueqi, et al. BeeZip2: A domain-specific accelerator for high performance lossless data compression [J]. Journal of Computer Research and Development, 2025, 62(6): 1562−1580 (in Chinese)
(高睿昊, 史舜晨, 李雪琦, 等. BeeZip2: 高性能无损数据压缩领域专用加速器[J]. 计算机研究与发展, 2025, 62(6): 1562−1580)



Wei Fangyu, born in 2003. Bachelor. Student member of CCF. His main research interest includes data compression coding.

韦芳宇, 2003 年生。学士。CCF 学生会员。主要研究方向为数据压缩编码。



Chen Tianyun, born in 2001. Master candidate. His main research interests include redundant data deletion and data compression coding.

陈天韵, 2001 年生。硕士研究生。主要研究方向为冗余数据删除、数据压缩编码。



Zhou Yang, born in 1998. PhD. Member of CCF. His main research interests include flash memory technology, cache technology, and storage reliability.

周　洋, 1998 年生。博士。CCF 会员。主要研究方向为闪存技术、缓存技术、存储可靠性。



Xie Yulai, born in 1984. PhD, professor, PhD supervisor. His main research interests include big data storage security, cloud storage security, storage reliability, and intrusion detection.

谢雨来, 1984 年生。博士, 教授, 博士生导师。主要研究方向为大数据存储安全、云存储安全、存储可靠性与入侵检测。



Wang Fang, born in 1972. PhD, professor, PhD supervisor. Her main research interests include parallel file systems, new storage systems based on non-volatile storage devices, graph computing and storage, and data deduplication and encoding.

王　芳, 1972 年生。博士, 教授, 博士生导师。主要研究方向为并行文件系统、基于非易失性存储设备的新型存储系统、图数据计算与存储、数据重删与编码。

《计算机研究与发展》投稿指南

征稿范围

计算机体系结构; 计算机网络与通信; 网络与信息安全; 人工智能; 软件技术; 并行与分布式计算; 图形图像、信息检索等应用技术; 其他计算机相关领域。

征稿类型

1)学术论文: 有创新学术见解的研究成果的完整论述, 对该学术领域的发展有积极意义, 或者支持我国学术生态和产业生态的重要成果。

2)综述: 对新兴的、活跃的学术研究领域或技术开发领域的评述。

3)快报短文: 对目前国内外计算机领域新思想、新观点、新技术的解读文章。

征稿要求

1)本刊只接收中文稿, 不受理英文稿。学术论文建议不超过 15 页, 综述不超过 20 页, 短文不超过 4 页。

2)要求来稿未在正式出版物上发表过, 不存在一稿多投问题。

3)作者在投稿时, 需同时向编辑部提交所有作者手写签字确认的“投稿声明”扫描版, 如不提供不予受理.对于违反投稿声明相关条款的来稿, 本刊将视情节做严肃处理, 后果自负。

4)本刊双盲评审, 作者投稿时上传系统的电子版中需要隐去作者、单位、基金、作者简介等信息, 初审通过后给编辑部邮寄的纸质版本需要包含前述署名信息。改后复审的稿件在提交修改版本以及修改说明时, 同样需符合双盲评审要求。

5)稿件评审通过后, 作者需提交单位审核盖章的“不涉密证明”和所有作者签字确认的“著作权转让声明”。

6)投稿时间在 6 个月以上的, 若没有收到编辑部发出的审理结果通知, 在书面通知编辑部并收到编辑部确认回复之后, 作者方可自行处理此稿件。

7)无论何种原因撤回投稿, 需以书面形式通知编辑部并确保所有作者知情。