



## 上海交通大学硕士学位论文

# 细粒度冗余识别的数据缩减系统研究

姓        名：刘 茜  
学        号：123037910018  
导        师：姚建国教授  
院        系：计算机学院  
学科/专业：软件工程  
申请学位：专业学位硕士

2026 年 1 月 14 日



**A Dissertation Submitted to  
Shanghai Jiao Tong University for the Degree of Master**

**RESEARCH ON DATA REDUCTION SYSTEM WITH  
FINE-GRAINED REDUNDANCY IDENTIFICATION**

**Author:** Liu Zhuo

**Supervisor:** Prof. Yao

School of Computer Science

Shanghai Jiao Tong University

Shanghai, P.R. China

January 14<sup>th</sup>, 2026



## **上海交通大学**

### **学位论文原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：

日期： 年 月 日

## **上海交通大学**

### **学位论文使用授权书**

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

- 公开论文
- 内部论文，保密  1年 /  2年 /  3年，过保密期后适用本授权书。
- 秘密论文，保密 \_\_\_\_ 年（不超过 10 年），过保密期后适用本授权书。
- 机密论文，保密 \_\_\_\_ 年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“√”）

学位论文作者签名：

指导教师签名：

日期： 年 月 日      日期： 年 月 日



## 摘要

随着云计算和人工智能的快速发展，云存储系统面临着数据爆炸式增长的挑战。为了降低存储成本，数据缩减技术（如去重和压缩）被广泛应用。然而，现代云存储负载呈现出“大块化”和“模型化”的新趋势：为了提升 I/O 性能，系统倾向于使用更大的数据块，导致传统块级去重难以识别块内的局部冗余；同时，AI 模型数据在存储中占比激增，其特有的浮点数数值冗余无法被传统去重或通用压缩算法有效消除。现有的单一优化方案难以同时应对这种混合负载，甚至可能产生相互干扰。

针对上述问题，本文提出了一种面向混有模型数据的云存储负载的具有细粒度冗余识别能力的数据缩减系统——FHRD (Fine-grained Hybrid Redundancy Deduplication)。该系统在传统数据缩减流水线的基础上，引入了子块粒度的冗余识别与去重，以及模型数据的鉴别、分离与编码压缩能力。首先，针对大块数据中的局部连续冗余，设计了细粒度子块去重模块，通过指数取整的双向子块定长分块、特征提取与双向匹配等方法，高效去除块内冗余。其次，面对混合负载中干扰去重流程的模型数据，在模型数据分离模块中提出了字节粒度分组抽样熵值分析方法，能够精准、快速地鉴别混杂在负载中的模型数据块，并将其分离出常规去重流程。最后，对于分离出的模型数据，在模型编码压缩模块中设计了基于熵值分析结论的字节分组压缩方法，通过重组排列浮点数，将不可压缩的数值冗余转化为可压缩形式，从而显著提升压缩率。

本文实现了 FHRD 原型，并进行了全面的实验评估。实验结果表明，在包含模型数据与传统数据的混合负载下，FHRD 相比 Destor 等传统“变长分块 + 块级去重 + 通用压缩”方案，在保持良好吞吐性能的同时，平均数据缩减率提升了 21.7%，对比 Zipnn、Burst 等针对优化方案也有不同程度的提升。且随着大块化和混合化趋势的发展，本文系统的优势愈发显著，最高可提升 38.4% 的数据缩减率，证明其具有应对现代云存储环境下混合负载数据缩减问题的能力。

**关键词：**存储，数据缩减，去重，压缩，细粒度，大语言模型，混合负载



## Abstract

With the rapid development of cloud computing and artificial intelligence, cloud storage systems are facing the challenge of explosive data growth. To reduce storage costs, data reduction technologies (such as deduplication and compression) are widely used. However, modern cloud storage workloads exhibit new trends of "large blocks" and "modeling": to improve I/O performance, systems tend to use larger data blocks, making it difficult for traditional block-level deduplication to identify local redundancy within blocks; at the same time, the proportion of AI model data in storage has surged, and its unique floating-point numerical redundancy cannot be effectively eliminated by traditional deduplication or general compression algorithms. Existing single optimization solutions struggle to address this mixed workload simultaneously and may even interfere with each other.

To address these issues, this paper proposes a data reduction system with fine-grained redundancy identification capabilities for cloud storage workloads containing model data—FHRD (Fine-grained Hybrid Redundancy Deduplication). This system builds on the traditional data reduction pipeline by introducing sub-block level redundancy identification and deduplication, as well as model data identification, separation, and encoding compression capabilities. First, to tackle the local continuous redundancy within large data blocks, a fine-grained sub-block deduplication module is designed, which efficiently removes intra-block redundancy through methods such as exponential rounding bidirectional sub-block fixed-length segmentation, feature extraction, and bidirectional matching. Second, to address the model data that interferes with the deduplication process in mixed workloads, a byte-granularity grouped sampling entropy analysis method is proposed in the model data separation module, which can accurately and quickly identify model data blocks mixed in the workload and separate them from the conventional deduplication process. Finally, for the separated model data, a byte-grouped compression method based on entropy analysis conclusions is designed in the model encoding compression module, which reorganizes and arranges floating-point numbers to transform incompressible numerical redundancy into compressible forms, thereby significantly improving the compression ratio.

The FHRD prototype is implemented and comprehensively evaluated. Experimental re-

sults show that under mixed workloads containing model data and traditional data, FHRD achieves an average data reduction rate improvement of 21.7% compared to traditional "variable-length chunking + block-level deduplication + general compression" solutions such as Destor, while also showing different improvements compared to targeted optimization solutions like Zipnn and Burst. As the trends of large blocks and mixed workloads continue to develop, the advantages of this system become increasingly significant, with a maximum data reduction rate improvement of 38.4%, demonstrating its capability to address the data reduction challenges of mixed workloads in modern cloud storage environments.

**Key words:** storage, data reduction, deduplication, compression, fine-grained, large language model, mixed workload

# 目 录

<b>第 1 章 绪论 .....</b>	<b>1</b>
1.1 研究背景与意义.....	1
1.2 国内外研究现状.....	4
1.3 本文研究工作与主要贡献.....	5
1.4 论文结构.....	7
<b>第 2 章 相关技术概述 .....</b>	<b>9</b>
2.1 传统数据缩减系统.....	9
2.1.1 数据分块（chunk） .....	10
2.1.2 哈希指纹（hash fingerprint） /索引（index） .....	11
2.1.3 压缩（compress） 算法 .....	12
2.2 增量压缩.....	13
2.3 细粒度冗余识别.....	13
2.4 模型编码压缩技术.....	14
2.5 缓存优化技术.....	16
2.6 本章小结.....	16
<b>第 3 章 细粒度冗余识别的数据缩减系统设计 .....</b>	<b>19</b>
3.1 现有问题分析与设计目标.....	19
3.2 系统总体设计.....	22
3.3 细粒度子块去重模块设计.....	23
3.4 模型数据分离模块设计.....	24
3.5 模型编码压缩模块设计.....	26
3.6 数据缩减工作流.....	27
3.7 本章小结.....	29
<b>第 4 章 系统实现与优化 .....</b>	<b>31</b>
4.1 细粒度子块去重模块优化——指数取整的双向子块分块方法.....	32
4.2 模型数据分离模块优化——字节粒度的分组抽样熵值分析方法.....	34
4.3 模型编码压缩模块优化——基于熵值分析结论的字节分组压缩方法.....	36

---

4.4 其他系统优化.....	38
4.4.1 数据管理与缓存优化 .....	38
4.4.2 数据块重构 .....	40
4.4.3 数据缩减流水线并行 .....	41
4.5 实现侵入性分析.....	42
4.6 本章小结.....	43
<b>第 5 章 实验结果与分析 .....</b>	<b>45</b>
5.1 实验环境与测试方法.....	45
5.1.1 实验环境 .....	45
5.1.2 测试对照方案 .....	45
5.1.3 工作负载说明 .....	47
5.1.4 测试方法及指标 .....	48
5.2 基准实验.....	48
5.2.1 数据缩减率 .....	48
5.2.2 吞吐量 .....	49
5.3 系统优化效果分析.....	51
5.3.1 指数取整的双向子块定长分块方法效果分析 .....	51
5.3.2 模型数据鉴别分离效率分析 .....	55
5.3.3 基于熵值分析结论的字节分组压缩效果分析 .....	57
5.4 负载特点影响实验.....	60
5.5 本章小结.....	62
<b>第 6 章 总结与展望 .....</b>	<b>63</b>
<b>参考文献.....</b>	<b>65</b>
<b>致 谢.....</b>	<b>69</b>
<b>学术论文和科研成果目录.....</b>	<b>71</b>

## 插 图

图 1.1 云存储服务系统架构.....	1
图 1.2 系统设计与实现论文结构图.....	7
图 2.1 传统数据缩减系统架构.....	9
图 2.2 变长分块示意图 <sup>[26]</sup> .....	10
图 2.3 哈希指纹与索引管理 <sup>[28]</sup> .....	11
图 2.4 增量压缩工作流示意图 <sup>[14]</sup> .....	13
图 2.5 Burst 去重原理示意图 <sup>[6]</sup> .....	14
图 2.6 ZipLLM 系统架构示意图 <sup>[7]</sup> .....	15
图 2.7 ZipNN 原理示意图 <sup>[8]</sup> .....	16
图 3.1 系统数据缩减率随数据块平均大小变化情况.....	19
图 3.2 系统数据缩减率随模型数据比例变化情况.....	20
图 3.3 模型文件中 BF16 浮点数各 bit 位熵值.....	21
图 3.4 设计思路.....	21
图 3.5 细粒度冗余识别的数据缩减系统设计.....	22
图 3.6 非模型数据处理模块.....	23
图 3.7 16 位浮点数二进制内容 .....	24
图 3.8 基于分组熵值分析的数据类型鉴别方法.....	25
图 3.9 模型编码压缩模块.....	27
图 3.10 系统数据缩减工作流 .....	28
图 4.1 定长分块的边界偏移问题.....	32
图 4.2 变长分块解决边界偏移问题.....	32
图 4.3 固定份数为 3 的子块划分 .....	33
图 4.4 指数取整的双向子块定长分块方法 .....	34
图 4.5 比特粒度实现的复杂性 .....	34
图 4.6 模型浮点数的表示形式 .....	35
图 4.7 字节粒度的分组抽样熵值分析 .....	36
图 4.8 基于熵值分析的字节分组压缩 .....	37
图 4.9 多级索引缓存下的匹配查找时序 .....	39

---

图 5.1 各类混合负载下数据缩减率对比.....	49
图 5.2 各类混合负载下吞吐量对比.....	50
图 5.3 非模型数据数据缩减率对比.....	51
图 5.4 非模型数据吞吐量对比.....	52
图 5.5 相似块检出率.....	53
图 5.6 相似块冗余数据缩减率.....	54
图 5.7 类型鉴别准确率对比.....	55
图 5.8 类型鉴别误报率对比.....	56
图 5.9 类型鉴别漏检率对比.....	56
图 5.10 类型鉴别吞吐量对比 .....	57
图 5.11 基于熵值分析结论的字节分组压缩效果对比 .....	58
图 5.12 压缩线程吞吐量对比 .....	59
图 5.13 数据缩减率随模型数据比例变化情况 .....	60
图 5.14 数据缩减率随块大小变化情况 .....	61

## 表 格

表 3.1 鉴别粒度对比表.....	26
表 5.1 实验服务器配置.....	45
表 5.2 对照方案.....	46
表 5.3 负载数据集.....	47



# 第1章 绪论

## 1.1 研究背景与意义

云计算技术的成熟与普及，有力地推动了现代信息技术产业的发展。云存储系统作为其服务基石之一，承担着为全球海量用户提供可靠、可扩展的数据存储能力的任务。它通过虚拟化技术将海量异构的物理存储设备抽象为一个统一、弹性的逻辑资源池，为用户提供按需分配、动态伸缩、高可用且持久化的数据存储能力，实现了计算与存储的解耦，显著提升了资源利用率和系统部署的灵活性，已成为现代企业数据管理的关键支撑。

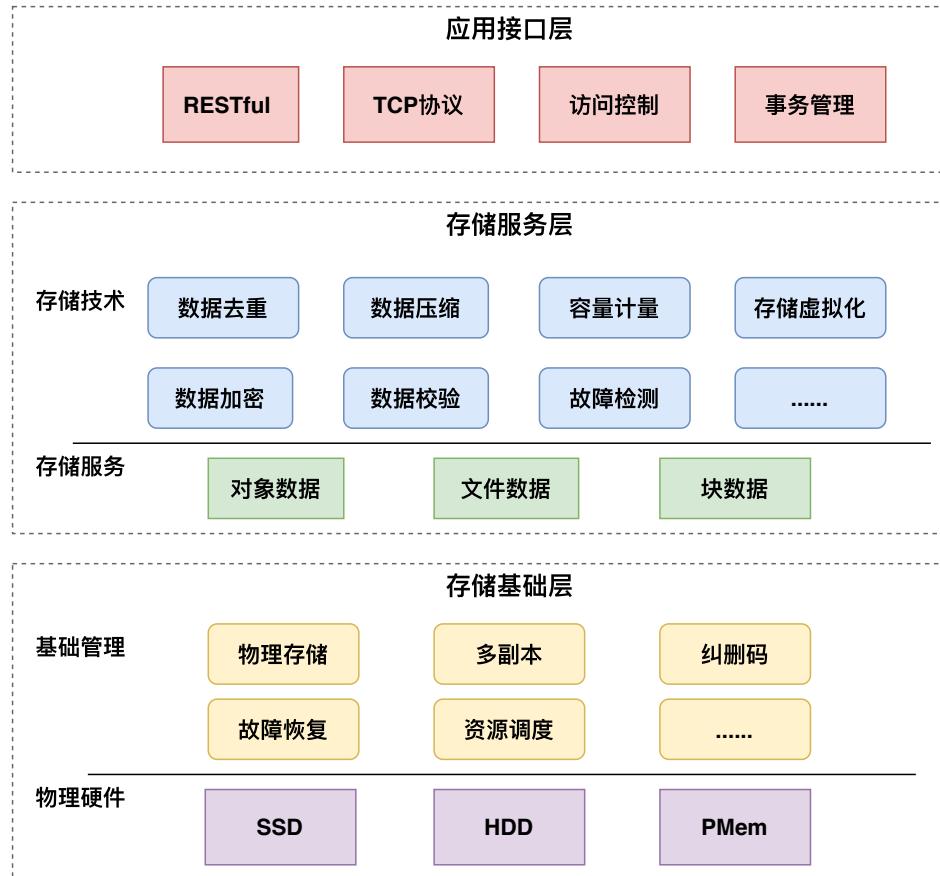


图 1.1 云存储服务系统架构  
Figure 1.1 System architecture for cloud storage services

从技术架构的视角审视，典型的云存储系统如图 1.1 所示，普遍采用分层设计：

底层是由 HDFS、Ceph 等构成的分布式存储基础层，负责数据的物理存储与高可用保障；中间是存储服务层，负责将物理资源封装为对象存储、文件存储等服务，并集成去重、压缩、计量等功能；顶层则是面向用户的应用接口层。

随着数字化的趋势以及大数据、人工智能等技术的广泛应用，全球数据总量正爆炸式增长。企业级应用、云盘存储、模型训练、物联网设备及科学计算等场景，每时每刻都在向云端注入海量数据，使得存储成本成为云服务提供商最主要的运营支出之一<sup>[1]</sup>。

为了有效应对大规模数据带来的成本压力，**数据缩减（Data Reduction）** 技术成为云存储系统中不可或缺的核心技术<sup>[2]</sup>，在数据写入物理介质前，通过识别并消除冗余信息，从而显著降低实际存储占用和网络传输开销。其中，**块级去重（Block-level Deduplication）** 技术与**通用压缩（Compression）** 技术因其普适性和有效性而得到广泛应用<sup>[3]</sup>。总体流程可概括为：首先，将连续的数据流切分成若干数据块；然后，为每个数据块计算唯一的密码学哈希值（如 SHA-256）作为其“指纹”特征；最后，通过比对已存储数据块指纹库识别重复数据块，对于重复数据块仅存储指向已有数据的元数据指针，而对新数据块则压缩后存入物理介质<sup>[4]</sup>。这套“分块—指纹—去重—压缩”的技术体系，在过去以文档、代码、虚拟机镜像等传统数据为主的时代取得了巨大成功，为云存储的规模化与商业化奠定了坚实的技术基础。

然而，近年来云存储数据缩减效率遇到瓶颈<sup>[5]</sup>，这与其所承载的工作负载变化密切相关。其一，云存储负载中的文件越来越大，为追求高 I/O 吞吐，现代存储系统倾向于采用更大的数据块尺寸，这导致传统去重技术的效果显著下降<sup>[6]</sup>。因为数据块大小的增加，找到两个完全相同数据块的概率极大地降低了。尽管这些大块内部往往包含大量局部重复片段（如重复的日志、代码或文档段落），但传统以整个块为单位的粗粒度去重对此类块内“局部连续冗余”难以检测，导致系统去重效果下降。这表明数据缩减系统需要引入更细粒度的冗余识别机制。

其二，随着人工智能，特别是大语言生成模型技术的突破性进展，云存储负载中混入越来越多的模型数据。模型训练产生大量以 .pt、.safetensors 等格式保存的模型文件，其体积动辄达到数百 GB 甚至 TB 级别，在总存储容量中的占比急剧攀升<sup>[7]</sup>。然而，面对这些数据，传统的块级去重与通用压缩技术往往同时失效。

- **去重失效：**模型文件由亿万级相互独立的浮点数张量构成，数据内容呈现高度的随机性和唯一性。当系统将不同模型切分为数据块时，几乎每个块都会拥有一个独一无二的指纹，这使得基于全局指纹匹配的去重机制完全失效，去重率

趋近于零。

- **压缩失效：**诸如 LZ4、Zstandard 等通用压缩算法，其核心原理是寻找并消除字节序列层面的重复模式。然而，模型数据的冗余并非体现在连续的字节流中，而是源于浮点数的数值分布特征与内部结构。具体而言，在 32 位单精度浮点数中，其二进制表示由符号位、指数位和尾数位构成。在模型训练产生的 checkpoints 中，绝大部分权重参数的数值集中在某个较小区间内（例如 [-2, 2]），导致其指数位高度相似甚至完全相同。这种高位重复、分布在每个浮点数内部的“数值冗余”，是传统字节级压缩算法难以捕捉和利用的<sup>[8]</sup>。

因此，这些混入存储负载中的模型文件，其数值冗余不仅无法通过传统手段缩减存储占用，反而会因其庞大的数据量而拉低系统的压缩性能，干扰数据缩减流程。这启示我们可以将模型数据从传统去重工作流中分离出来，一方面可以避免模型数据对传统流程性能产生负面影响，另一方面也可以针对其独特的数值冗余，从更细的浮点数粒度设计专用的数据缩减方法，从而提升整体数据缩减率。

然而，云存储场景下的模型数据分离面临巨大挑战。首先，模型文件通常与其他类型的数据（如训练数据、配置文件、系统日志）存在于同一存储项目中，而用户在上传文件时并不会提供明确的类型标签。其次，出于隐私和安全考虑，云存储系统难以获得用户数据的语义信息，因此，系统必须依赖对数据内容的实时分析来识别模型数据<sup>[9]</sup>。最后，模型数据分布在各类文件之中，除了典型的模型权重文件和训练过程中产生的检查点文件、优化器状态文件等，还可能混杂在归档文件或镜像文件中，增加了识别的复杂性。如何快速且准确地识别并分离模型数据，是一个亟需解决的难题。

综上所述，两种负载变化趋势共同塑造了“模型数据与大块数据共存”的混合负载，而当前云存储数据缩减技术面临的困境在于：传统技术的冗余识别粒度与新兴数据特征不匹配，以及混合负载带来的复杂性。无论是大尺寸传统数据中存在的“局部连续冗余”，还是混杂的模型数据中隐含的“数值冗余”，都超越了传统块级指纹比对技术的感知范围，因此，突破这一瓶颈的关键在于：在避免混合数据相互干扰的同时，将冗余识别从粗粒度的块级处理推进到细粒度方法。

基于此，本研究提出并构建了一套名为 FHRD (Fine-grained Hybrid Redundancy Deduplication) 的细粒度冗余识别数据缩减系统。该系统首先在传统去重流水线的基础上，增加了块内连续冗余识别能力，通过对数据块内部进行细粒度划分与比对，去除局部重复片段。其次，该框架内置了一个轻量级的模型数据鉴别分离模块，它能实

时、准确地判别数据块所蕴含的冗余类型是否符合模型数据规律，从而将模型数据从工作流中分离出来专门处理，避免对细粒度去重流程的负面影响。最后，针对分离出的模型数据，框架采用专用的分组编码方法压缩其数值冗余。通过这种协同处理架构，FHRD 能够在不显著影响系统吞吐性能的前提下，对数据施加有效的优化手段，从而系统性地提升混合负载下的整体数据缩减率。

## 1.2 国内外研究现状

数据去重与压缩技术的研究已历经数十年的发展，从传统的块级去重到细粒度的冗余识别，再到针对特定数据类型的专用压缩，技术路径呈现出不断细化与深入的趋势。

传统块级去重技术是云存储系统中数据缩减的基石，其核心原理是通过消除重复数据块来降低存储开销。根据分块策略的不同，主要分为固定大小分块与可变长度分块两种技术路线。固定大小分块方案将数据流均分为固定长度的块，实现简单，性能开销可控，但存在明显的边界效应——数据在分块边界处的微小变化会导致后续所有分块都不匹配，严重降低去重效率<sup>[10]</sup>。为克服这一局限，可变长度分块技术应运而生，它利用 Rabin 指纹等滚动哈希算法，根据数据内容本身动态确定分块边界，使分块结果对数据插入和删除操作更具弹性<sup>[11]</sup>。然而，无论是固定还是可变分块，它们都依赖于寻找完全相同的字节序列，对于 AI 模型文件中普遍存在的数值相似性冗余，传统方法完全无法识别。同时，为提升 I/O 吞吐而采用的 MB 级大数据块，也使得块内大量存在的局部连续重复无法被检测和消除。

为突破传统块级去重在大块场景下的局限，学术界提出了多种细粒度冗余识别技术，主要可分为两大类：面向相似块的增量压缩技术和面向局部连续重复的细粒度去重技术。增量压缩技术旨在通过识别数据块间的相似性，仅存储差异部分（Delta）来实现压缩。其核心挑战在于如何快速、准确地从海量数据块中找出相似块对。从早期的 N-Transform<sup>[12]</sup>到 Finesse<sup>[13]</sup>和 Odess<sup>[14]</sup>，研究者们不断优化特征提取与匹配算法以平衡精度与效率。找到相似块后，Xdelta<sup>[15]</sup>和 Gdelta<sup>[16]</sup>等差异编码技术则用于生成紧凑的补丁文件。另一研究方向专注于消除大块数据内部的局部连续重复，Burst<sup>[6]</sup>是这方面的典型代表。它通过在 MB 级大块内部进行识别并消除首尾长重复序列，有效缓解了因块尺寸增大导致的去重率下降问题。然而，这些细粒度技术在带来更高去重率的同时，也引入了额外的计算开销和系统复杂性，在混合负载环境中若不加区分地应用，可能导致资源浪费。

在去重之后，通用压缩算法，如 Gzip<sup>[17]</sup>、LZW<sup>[18]</sup>、ANS<sup>[19]</sup>，作为数据缩减流水线的第二道工序，负责进一步消除数据块内部的统计冗余。近年来，压缩算法的研究主要围绕速度与压缩率的平衡展开，产生了如 LZ4<sup>①</sup> 和 Zstandard<sup>[20]</sup> 等新一代压缩算法。LZ4 以其极致的压缩和解压速度著称，适合实时高吞吐场景，但压缩率相对较低。Zstandard 则致力于在速度与压缩率间取得更优平衡，通过引入有限状态熵编码（FSE）和预训练字典等机制，提供了灵活的压缩级别选择，在保持高速解压的同时实现了接近甚至超越传统 Gzip 的压缩率。尽管这些通用压缩算法对文本类数据表现出色，但它们对模型数据的压缩效果极为有限，其根本原因在于它们基于字节序列重复模式进行压缩，无法捕获模型数据内在的数值冗余。

另一方面，面对模型数据的独特性，研究者提出了多种专用压缩技术，其中量化和剪枝是最为重要的两种方向。模型量化<sup>[21]</sup>通过降低权重和激活值的数值精度来减少存储与计算开销，而模型剪枝<sup>[22]</sup>则通过移除冗余参数来实现模型压缩。但这两类方法属于有损压缩且严重依赖模型训练过程，通常不适用于要求数据完整性的云存储服务。此外，还涌现了如 ZipNN<sup>[8]</sup>、FM-Delta<sup>[23]</sup> 和 ZipLLM<sup>[7]</sup> 等无损压缩方法，它们利用模型权重间的数值分布规律或跨模型版本的比特级相似性，实现了比通用算法更高的压缩率。但这些方法往往只针对特定格式的模型文件，无法处理打包文件或识别模型文件中的非模型部分（如文件头、量化内容），限制了其在云存储场景中的应用。

综上所述，当前数据缩减技术研究面临以下挑战。

- 传统块级去重技术在大块场景下去重率下降，难以利用块内的局部连续冗余。
- 通用压缩算法无法捕捉模型数据中的数值冗余，导致压缩效果不佳。
- 现有细粒度冗余识别技术多未考虑模型数据的影响，且存在性能方面的不足。
- 现有模型专用压缩方法多针对单一数据类型设计，难以适应混合负载环境。

### 1.3 本文研究工作与主要贡献

块级去重技术和通用压缩技术在应对新兴的混合工作负载时效果有限。本研究深入分析发现，问题的本质在于传统技术体系与新兴数据特征之间的“粒度不匹配”——传统粗粒度冗余识别方法难以利用大尺寸传统数据内部的局部连续重复模式，也无法有效捕捉模型数据中的数值相似性冗余。更为严峻的是，针对单一数据类型优化的先进技术（如面向模型数据的专用压缩和面向传统数据的细粒度去重）在混合负载环

① <https://github.com/lz4/lz4>

境下会产生相互干扰，导致性能下降。

基于此，本研究的核心任务是设计并实现一套面向混合负载的细粒度冗余识别的数据缩减系统，旨在解决三个关键问题：(1) 如何设计并实现细粒度块内冗余的高效去重，以应对数据大块化的挑战；(2) 如何设计并实现模型数据的数值冗余识别与压缩方法，突破传统数据缩减技术的瓶颈，以应对日益增多的模型数据；(3) 如何实现各种优化策略的协同工作，避免混合场景下模型数据对数据缩减流程的负面影响。

为解决上述问题，本文创新性地提出了名为 FHRD (Fine-grained Hybrid Redundancy Deduplication) 的细粒度冗余识别数据缩减系统架构。该系统在传统数据缩减流水线的基础上，添加了细粒度子块去重模块、模型数据鉴别分离模块和模型编码压缩模块，以应对上述问题。本文的工作和贡献主要体现在以下几个方面：

1. 揭示了云存储负载呈现出大块化、模型化及混合化的趋势。系统性地分析了传统数据缩减技术在现代云存储环境中效果下降的根源，明确了两种影响显著的关键冗余：大块数据中的“局部连续冗余”和模型数据中的“数值冗余”，并阐述了它们的特征与分布规律。
2. 通过局部子块的指纹计算与匹配，定位并消除大块数据的局部连续冗余，有效缓解了因块尺寸增大导致的去重率下降问题。
3. 提出了基于内容特征的高精度块级数据类型鉴别方法，将模型数据的处理从原有流程中剥离。该方法发现了模型数据在字节层面呈现的“循环相似模式”，并设计了相应的轻量级检测算法。与现有方法相比，该鉴别机制克服了依赖文件元数据进行模型数据识别的局限性，具有粒度细、精度高、适用性广的优势，且能够准确识别出模型文件中的非模型部分（如文件头、量化内容），避免了误用专用编码器导致的性能退化。
4. 设计并整合了面向混合场景的差异化冗余处理逻辑。分离了模型数据的处理流程，针对其数值冗余，基于浮点数位结构分析，重组高度相似的指数位部分，将不可压缩数据块转化为可压缩形式；实现了对模型冗余的精准优化与整体去重效能的提升。
5. 实现并优化了完整的原型系统，进行了系统性的实验验证。本研究基于开源去重系统 Destor<sup>[24]</sup>实现了 FHRD 原型。实验结果表明，在混合负载下，FHRD 相比传统方法在去重率上取得了高达 38% 的提升。我们通过全面的实验，涵盖了不同模型数据比例、块大小配置及文件形式，详细分析了各模块的性能特征和系统整体开销，为混合负载下的数据缩减提供了有力的评估数据。

## 1.4 论文结构

本文内容共分为六章，其组织结构如下：

第一章为绪论。阐述研究背景与意义，分析云存储在人工智能时代面临的新挑战，指出传统数据缩减技术在大块化趋势和混合负载下的局限性。在此基础上，明确本文的研究目标与主要创新点，并概述论文的组织结构。

第二章为相关技术。系统回顾并对比分析数据去重、通用压缩、细粒度冗余识别及模型专用压缩等方向的代表性工作。通过归纳现有方法的适用场景与不足，为本文方法的设计提供理论依据。

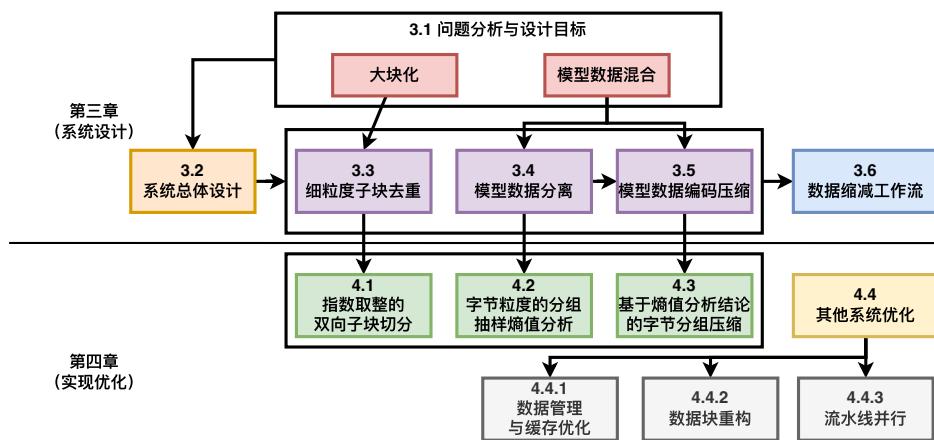


图 1.2 系统设计与实现论文结构图  
Figure 1.2 Logical structure of system design and implementation

第三章为系统设计。针对现有技术的痛点，提出面向混合负载的 FHRD 系统总体设计。首先分析细粒度冗余特征，明确系统设计目标；继而详细阐述细粒度子块去重、模型数据鉴别分离、模型数据编码等方法的设计；最后梳理系统工作流，为后续实现奠定基础。

第四章为系统实现与优化。面对实现过程中的难点，给出具体的实现与优化方案。详细介绍指数取整的双向子块定长分块、字节粒度分组抽样熵值分析、基于熵值分析结论的字节分组压缩等核心优化的实现细节；同时，为提升系统整体性能，实现了数据段聚合、多级索引缓存及流水线并行处理等细节优化。论文三四章行文逻辑如图 1.2 所示

第五章为实验评估。搭建标准化实验平台，选取多种数据集，设置多种对照方案，从去重率、吞吐量、鉴别准确率等核心指标展开全面评估。通过基准实验、消融实验和负载特点影响实验，系统地验证 FHRD 在混合负载下的整体优势、各关键模

块的贡献及其在不同场景下的适应性。

第六章为总结与展望。总结全文研究成果，提炼 FHRD 系统的核心技术创新与工程实践价值。同时，分析当前研究的局限性，并结合存储技术发展趋势，对未来的研方向进行展望。

## 第2章 相关技术概述

### 2.1 传统数据缩减系统

传统数据缩减系统，其架构如图 2.1 所示，通常采用“先去重，后压缩”的两阶段处理流水线，核心目标在于通过消除数据冗余以最大化存储空间节省。其中去重（通常为块级去重）由数据分块、指纹计算、匹配去重三个关键步骤构成<sup>[25]</sup>，与数据压缩共同组成一个完整的数据缩减工作流。

系统处理流程始于数据分块（Chunking）阶段。在此阶段，原始数据流被分割成一系列较小且易于管理的单元，即数据块（Chunk）。

随后进入指纹计算（Fingerprinting）阶段。系统利用密码学哈希函数（如 SHA-1 或 SHA-256），依据数据块内容为每个数据块生成一个唯一的数字指纹（Fingerprint）。作为后续去重决策的依据。

匹配去重（Duplicate Check）是系统的核心环节。系统维护一个全局的指纹索引库（Index），用于存储所有已处理过的数据块的指纹。当新数据块的指纹生成后，系统会查询索引库。若发现匹配的指纹，则表明该数据块是重复的，系统仅需存储一个指向已有物理数据块的元数据指针即可，无需再次存储相同的数据。反之，若未发现匹配，则该数据块被判定为非重复块/唯一块（Unique Chunk），并被送入下一处理阶段。

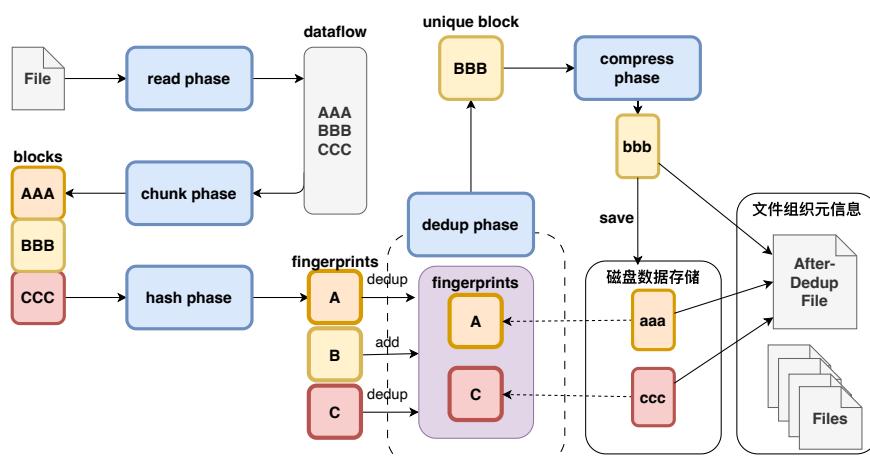


图 2.1 传统数据缩减系统架构  
Figure 2.1 Architecture of traditional data reduction system

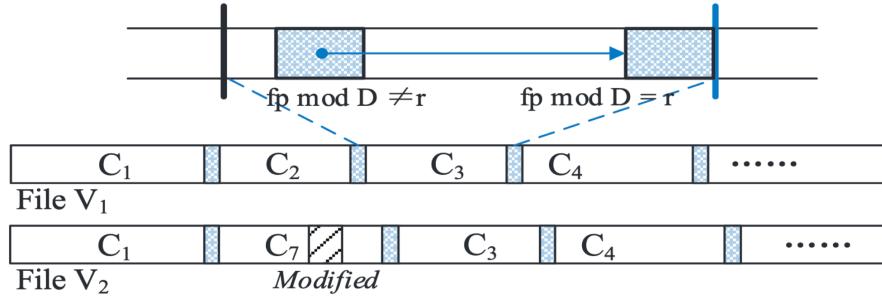


图 2.2 变长分块示意图<sup>[26]</sup>  
Figure 2.2 Illustration of variable-length chunking

最后，在数据压缩（Compression）阶段，所有唯一数据块会经过通用压缩算法（如 Gzip、LZ4、Zstd 等）的进一步处理。这一步骤旨在消除单个数据块内部的统计冗余，从而实现存储空间的二次优化。

### 2.1.1 数据分块（chunk）

**固定大小分块（Fixed-Size Chunking, FSC）**是最基础、最直观的数据分块方法。该方法将输入的数据流视为一个连续的字节序列，并按照预设的固定长度（如 4KB、8KB 或 64KB）进行均匀切分，从而生成一系列大小完全相同的数据块<sup>[10]</sup>。FSC 的实现简单，计算开销低。由于分块边界仅由数据在流中的偏移量唯一确定，无需进行任何复杂的内容分析，因此其处理速度具有显著优势。在数据内容相对静态的场景中，如虚拟机镜像的首次全量备份、大规模软件分发包的归档等，该技术能够提供稳定且可靠的去重效果。此外，大小均一的数据块也简化了存储空间的管理与分配，降低了系统设计的整体复杂性。

然而，FSC 技术存在一个致命的缺陷，即**边界偏移问题（Boundary Shift Problem）**。当数据流中发生任何插入或删除操作时，哪怕只是单个字节的变动，都会导致该变动点之后的所有数据块边界发生连锁式偏移。例如，在一个文本文件中部插入一行文字，将导致从插入点到文件末尾的所有后续数据块的内容全部改变。这种现象使得原本相同的大段内容，在新生成的数据流中因其所处位置的变化而被识别为全新的、不同的数据块。这种连锁反应严重削弱了去重系统识别重复内容的能力，从而导致去重效率急剧下降。

如图 2.2 所示，**内容定义分块（Content-Defined Chunking, CDC）**，又称变长分块，是为解决固定分块的边界偏移问题而设计的关键技术。其核心思想是根据数据内

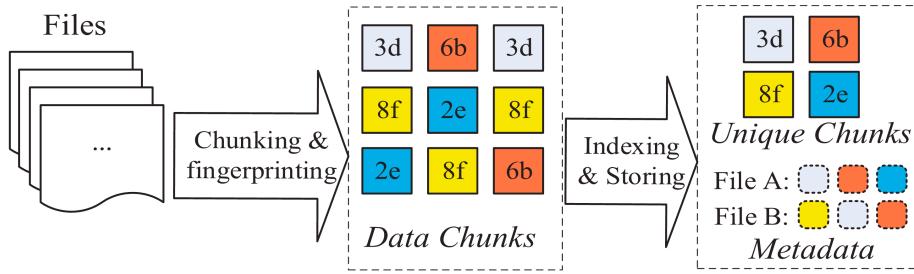
图 2.3 哈希指纹与索引管理<sup>[28]</sup>

Figure 2.3 Hash fingerprint and index management

容自身的特征来动态确定分块边界，从而使得分块结果对数据的插入和删除操作具有良好的鲁棒性<sup>[11]</sup>。Rabin 算法<sup>[27]</sup>是 CDC 技术中最具代表性且应用最广泛的一种实现。该算法采用一个固定大小的滑动窗口（Sliding Window）在数据流上逐字节滑动，并为窗口内的数据高效地计算一个哈希值，即 Rabin 指纹。当该哈希值的某些特定位（通常是低位）与一个预定义的模式（Pattern）匹配时，就将当前窗口的结束位置标记为一个切分点。Rabin 算法的核心优势在于其“滚动哈希”（Rolling Hash）的特性：当窗口向前滑动一个字节时，新的哈希值可以通过一次简单的数学运算从旧的哈希值推导得出，无需重新对整个窗口的数据进行完整计算，这使得 CDC 过程极为高效。

在实际应用中，CDC 算法需要精细地控制生成块的大小分布。理想情况下，块大小应在一个合理的范围内波动，既要避免产生大量过小的块（这会导致元数据急剧膨胀，增加管理开销），也要防止出现少数过大的块（这会降低发现重复数据的概率）。为实现这一目标，通常会设定一个最小块大小（Min-Size）和最大块大小（Max-Size）。只有当滑动窗口超过最小块大小时，才开始检查哈希值；而当窗口滑动距离超过最大块大小时，则无论哈希值是否匹配，都会强制进行一次切分。

### 2.1.2 哈希指纹 (hash fingerprint) /索引 (index)

哈希指纹是数据缩减系统中最重要的元数据，系统据此进行数据块的匹配去重和索引管理。如图 2.3 所示，一方面，哈希指纹能够唯一标识每个数据块，帮助快速确定待存储数据块是否为重复块；另一方面，通过在文件元数据中加入索引构成，记录文件所含数据块的具体位置，通过指纹的索引管理确保了系统在海量数据中快速定位和访问相关数据块。

一个理想的哈希函数需要满足以下条件：首先，对于相同内容的数据块，必须生成完全相同的指纹（确定性）；其次，对于不同内容的数据块，生成相同指纹的概率

(即哈希碰撞) 必须极低 (抗碰撞性); 最后, 哈希计算本身的速度要足够快, 以满足高吞吐的数据处理需求。

在早期的去重系统中, MD5 和 SHA-1 等加密哈希函数因其良好的抗碰撞性而被广泛采用<sup>[29-30]</sup>。然而, 随着计算能力的提升和安全标准的演进, 这些算法的安全性受到挑战。因此, 现代云存储系统逐渐转向使用如 SHA-256 等更为安全的哈希算法<sup>[31]</sup>, 尽管这会带来一定的计算成本增加<sup>[32]</sup>。

### 2.1.3 压缩 (compress) 算法

在去重流程之后, 压缩算法作为数据缩减流水线的最后一道防线, 负责进一步消除每个唯一数据块内部的统计冗余。

LZ 系列算法是基于字典编码原理的经典压缩方法, 其通过查找并替换数据中重复出现的字符串来实现压缩。其中, LZ4 算法以其极致的速度而著称, 它采用简化的哈希链机制和字节对齐的编码格式, 实现了极高的处理吞吐量。在典型的 x86 平台上, LZ4 的压缩速度可达数百 MB/s, 而解压速度更是接近 GB/s, 使其成为实时数据处理、高速缓存等对延迟高度敏感场景的理想选择。

Zstandard (Zstd) 算法则代表了速度与压缩率的平衡。它在经典的 LZ77<sup>[33]</sup> 算法基础上, 创新性地引入了非对称数字系统 (Asymmetric Numeral Systems, ANS) 的一种高效实现——有限状态熵编码 (Finite State Entropy, FSE)。FSE 能够以接近信息论香农极限的效率进行熵编码, 同时保持对现代 CPU 硬件友好的特性。Zstandard 提供了从负到正共 22 个压缩级别, 允许用户根据具体应用需求, 在压缩速度与压缩率之间做出灵活的权衡。在默认级别下, Zstd 通常能提供比传统 Zlib 更高的压缩率, 同时保持更快的压缩和解压速度。

在选择压缩算法时, 系统需要综合考量数据类型、性能要求及硬件特性。例如, 文本、日志等非结构化数据通常具有较高的统计冗余, 能够获得良好的压缩效果; 而已经过压缩的媒体文件 (如 JPEG 图片、MP4 视频) 或加密数据, 其内容近似于随机噪声, 压缩空间极为有限。在某些对 I/O 延迟极度敏感的应用场景中, 系统甚至会选择完全禁用压缩, 以避免引入额外的 CPU 计算开销。

综上, 2.1 节中“分块-指纹-匹配去重-压缩”的经典设计<sup>[34-35]</sup>, 使得传统数据缩减系统能够在保证数据完整性的前提下, 显著提升存储效率。然而, 随着数据形态的不断演变, 在处理 AI 模型数据中普遍存在的数值冗余, 以及大块数据内部的局部连续重复时, 这一架构在面对新型工作负载时逐渐暴露出其局限性。

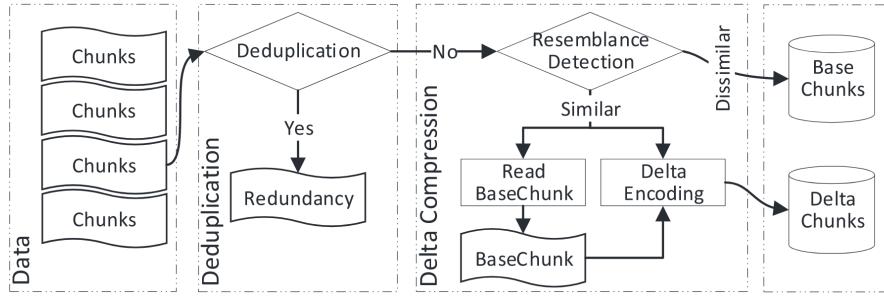


图 2.4 增量压缩工作流示意图<sup>[14]</sup>  
**Figure 2.4 Illustration of incremental compression workflow**

## 2.2 增量压缩

传统去重技术基于精确匹配机制，仅能识别和消除完全一致的数据块，对于存在高度相似性但并不完全相同的数据块处理效果有限。为解决这一问题，更细粒度地挖掘块内冗余，增量压缩技术应运而生，它通过识别并利用数据块之间的相似性，仅存储它们之间的差异，从而突破了精确块匹配的限制<sup>[36]</sup>。

以 Odess<sup>[37]</sup>为代表的典型增量压缩工作流如图 2.4 所示，在块级去重后衔接了更细粒度的冗余去除，包含三个关键环节：特征提取、相似匹配和差分计算<sup>[38]</sup>：

- **特征提取**：为每个数据块计算一个或多个能够代表其内容的“特征”，用于相似性匹配。理想的特征应保证语义相似的块具有相似的特征。
- **相似匹配**：通过快速比对新数据块的特征与索引中已有特征的相似度，寻找最相似的候选参考块。
- **差分计算**：在找到参考块后，利用差分算法（如 xdelta）计算目标块与参考块之间的精确差异，生成紧凑的差异补丁文件，而后系统仅存储该较小的差异文件，达到数据缩减的效果。

增量压缩技术在备份服务<sup>[39]</sup>、分布式存储<sup>[40]</sup>和容器镜像分发等场景中展现出显著价值。然而，其计算复杂度高、系统设计复杂，且在处理 AI 模型等缺乏明显序列的数据时，可能产生不必要的计算开销却得不到数据缩减效果。

## 2.3 细粒度冗余识别

除增量压缩外，还有研究者关注到大块数据内部的局部重复问题，以另一种方式进行细粒度冗余识别。例如，Burst<sup>[6]</sup>发现备份系统中的大量相似数据块存在“仅有中间部分突发差异”的特点，通过识别相同的文件头或文件尾部，消除大块内部的边缘冗

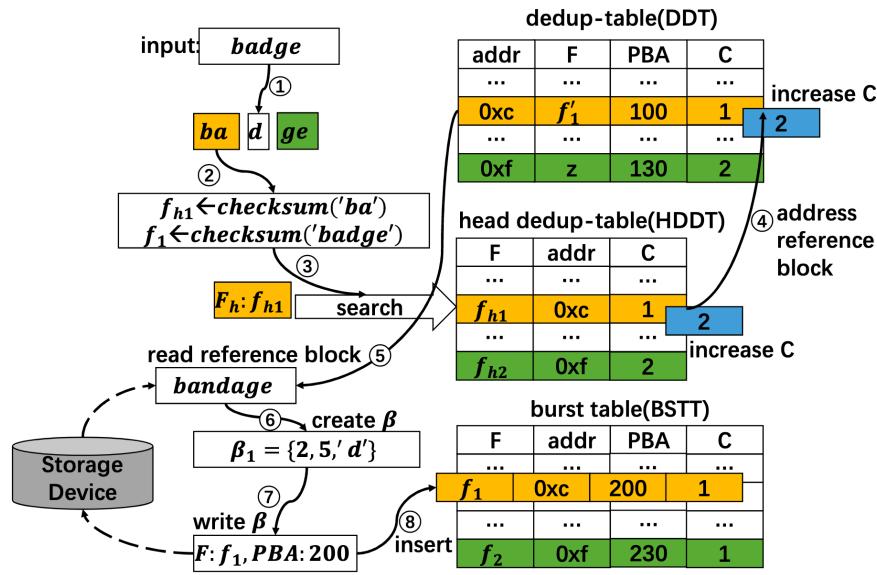
图 2.5 Burst 去重原理示意图<sup>[6]</sup>

Figure 2.5 Illustration of Burst deduplication principle

余数据。如图 2.5 所示，Burst 系统在哈希阶段不仅扫描数据块生成块指纹（checksum (badge)），而且会依据头/尾内容生成头/尾指纹（checksum (ba)）。Burst 据此在匹配阶段识别出具有相同前缀或后缀的块（bandage）作为参考块，去除大块头部冗余（ba）和尾部冗余（ge），只对中间的差异部分进行存储（nda->d），从而实现更细粒度的冗余消除。Burst 只针对定长分块做了优化，其头尾划分依赖于块大小，且其方法仅针对数据块中的边缘部分，因此尚未充分地利用数据中的连续冗余，但这种将大块切割成更细粒度的组成部分，并提取子块特征的思想值得进一步探索。

## 2.4 模型编码压缩技术

随着大语言模型规模的急剧扩张，模型压缩技术已成为提升部署效率、降低存储与计算成本的关键。主流技术可分为两大类：

第一类是通过牺牲一部分模型精度来换取更小的模型体积，即缩小模型本身，主要包括剪枝（Pruning）和量化（Quantization）技术。剪枝技术通过移除神经网络中冗余或不重要的连接和神经元，减少模型参数数量，从而降低存储需求和计算复杂度。量化技术则通过将高精度的浮点数参数转换为低精度表示（如 8 位整数），显著减少模型的存储空间，同时加速推理过程。然而，剪枝与量化技术属于有损压缩，且严重依赖模型的训练过程。在云存储场景中，用户上传的通常是预训练好的模型文件，并

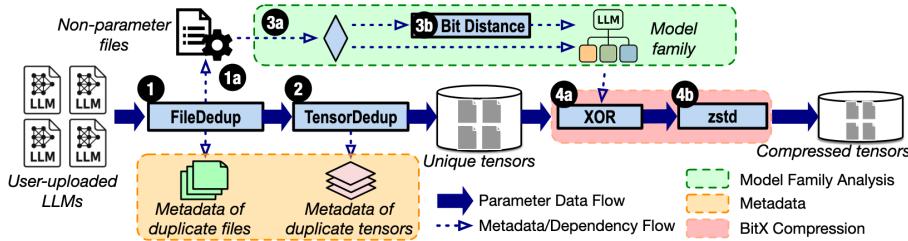


图 2.6 ZipLLM 系统架构示意图<sup>[7]</sup>  
Figure 2.6 Illustration of ZipLLM system architecture

且期望无损存储，因此无法应用这些技术。

第二类是在不改变模型内容的前提下，通过编码压缩模型，即缩小存储的空间。由于通用无损压缩算法（如 Gzip 等）在处理模型文件时效果不佳，难以有效处理模型参数中普遍存在的数值冗余，研究者提出了多种专用的无损模型压缩方法。例如，ZipNN<sup>[8]</sup>通过重排浮点数结构来提升通用压缩器的效率；ZipLLM<sup>[7]</sup>则利用微调模型间的高度相似性，通过比特位异或操作生成稀疏的差异文件；而 FM-Delta<sup>[23]</sup>专注于模型更新场景，通过计算版本间的增量差异进行压缩。

如图 2.6 所示，ZipLLM 系统架构包括模型预处理、差异计算和压缩存储三个主要模块。首先，系统对上传的 LLM 模型进行预处理，提取代表特征，搜索相似的同类模型。接着，利用参考模型（通常是同类的预训练模型）计算目标模型与参考模型之间的比特级差异（如 XOR 等编码方法），生成一个稀疏的差异文件。最后，该差异文件经过高效的无损压缩算法处理后存储，从而实现显著的存储空间节省。这些方法在 Huggingface 等模型存储库中展现出优异的压缩效果，但其缩减的冗余来源于同系列微调模型间的不变数据，即多个 LoRA 模型。而在云存储负载中，同一用户空间内通常不会保存多个同系列模型，因此难以取得理想效果。

另一方面，如图 2.7 所示，ZipNN<sup>[8]</sup>通过重排浮点数来提升通用压缩器的效率。与 ZipLLM 不同，该方法更加关注模型内部的数值冗余，能够在不依赖于同类模型的情况下，独立地对单个模型进行压缩，利用的是模型参数中的浮点数数值冗余，这可以为我们实现模型数据压缩提供思路。

这些专用方法的局限性在于其与模型特征的高度绑定，它们深度依赖于 LLM 数据的特定结构（如张量布局、浮点数格式等）。若将它们应用于文档、图片等传统用户数据，不仅无法发挥作用，甚至可能因引入额外开销而导致压缩后体积增大。因此，这些技术无法作为通用算法直接集成到处理混合数据类型的云存储去重工作流中。

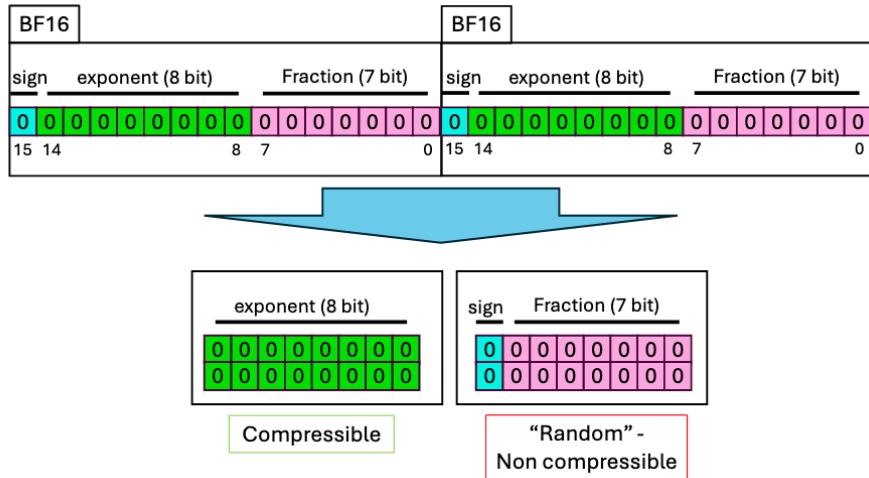


图 2.7 ZipNN 原理示意图<sup>[8]</sup>  
Figure 2.7 Illustration of ZipNN principle

## 2.5 缓存优化技术

哈希索引的管理是去重系统面临的另一大挑战。随着存储数据量的爆炸式增长，指纹索引的规模也在不断扩大，带来了存储和查询效率的压力<sup>[41]</sup>。为了解决这一问题，学术界和工业界提出了多种高效的索引优化与缓存技术。例如，使用布隆过滤器（Bloom Filter）<sup>[42]</sup>及其变种（如级联布隆过滤器）作为内存中的快速查询过滤器，可以有效拦截绝大多数新数据的查询请求，仅让少量可能重复的数据访问磁盘上的完整索引。此外，基于数据局部性原理的容器化管理（Container-based Management）<sup>[43]</sup>和索引分区（Index Partitioning）等策略，通过将相似或相关的数据块及其指纹组织在一起，优化了磁盘 I/O 模式，进一步提升大规模索引的查询性能。

## 2.6 本章小结

本章首先深入剖析了传统去重系统的工作原理，详细探讨了数据切块、哈希指纹与索引、以及数据压缩等关键技术环节，并指出了其在面对新数据场景时暴露出的局限性。接着，本章介绍了为克服传统去重技术限制而发展的细粒度去重技术，特别是增量压缩的原理、应用与挑战，同时也提及了 Burst 等面向特定场景的细粒度去重方法。随后，我们概述了模型压缩技术的发展现状，区分了有损（剪枝、量化）与无损压缩方法，并重点分析了 ZipLLM、ZipNN 等模型专用无损压缩技术在云存储环境中的适用性与局限性。此外，本章还简要讨论了去重系统中的缓存与索引优化技术。总

体而言，本章通过对相关技术的全面梳理，为后续章节中混合数据类型的细粒度冗余识别的数据缩减方法，奠定了坚实的理论基础，并明确了当前研究的技术需求。



## 第3章 细粒度冗余识别的数据缩减系统设计

### 3.1 现有问题分析与设计目标

随着数据块大小的持续增长，以及模型数据在云存储中的比重日益增加，传统数据缩减系统在处理现代云存储工作负载时，其局限性愈发凸显。具体而言，传统系统主要面临以下挑战：

- 对大块数据的去重能力有限：随着存储块平均大小的增加，数据块内部往往包含大量局部重复内容。传统系统在分块时无法有效捕捉这些局部重复，导致大量冗余未被识别和消除。如图 3.1 所示，在使用多个版本的 Linux 系统源码 tar 包作为负载时，开源块级去重系统 Destor 结合 Zstd 压缩算法的数据缩减率<sup>①</sup>随数据块平均大小的增加而明显下降，反映了其处理大块数据局部重复的不足。

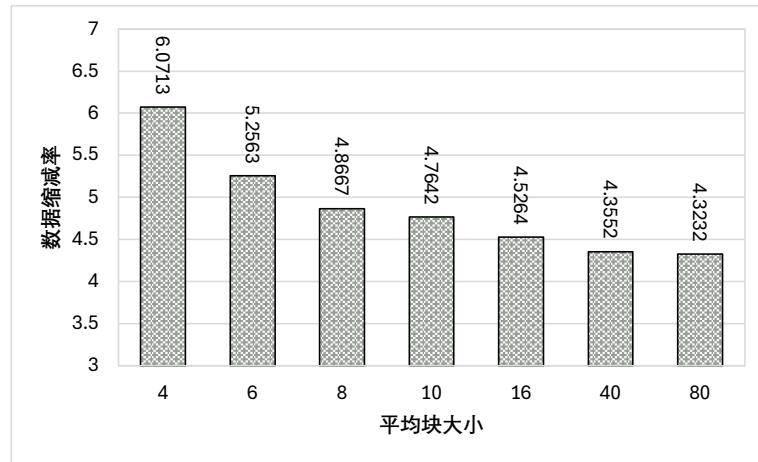


图 3.1 系统数据缩减率随数据块平均大小变化情况  
Figure 3.1 deduplication rate variation with average data block size

- 难以应对混入的模型数据：传统系统难以通过简单的指纹比对有效识别模型数值冗余，导致数据缩减率大幅下降。如图 3.2 所示，在一个包含文件数据（非模型）与 Transformer 模型 checkpoints（模型）的混合负载下，开源块级去重系统 Destor 结合 Zstd 压缩算法的数据缩减率随着模型数据占比的增加而显著降低。当模型数据占比达到 100% 时，数据缩减率几乎降至 1，直观地暴露了其在处理模型数值冗余方面的短板。

<sup>①</sup> 数据缩减率 = 存储数据原始大小 / 去重压缩后大小

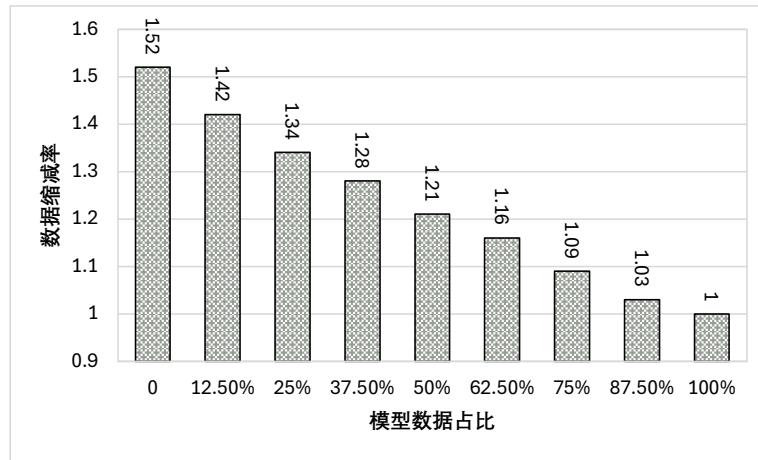


图 3.2 系统数据缩减率随模型数据比例变化情况  
Figure 3.2 Deduplication rate variation with model data proportion

3. 存储与计算开销较高：为了提升去重效果，传统系统不得不采用更复杂的分块和指纹计算方法，这无疑增加了系统的存储和计算开销，进而影响了整体性能和可扩展性。这也是增量压缩等更细粒度的去重技术难以在工业界大规模应用的主要原因。

这些新型数据中蕴含着大量未被有效利用的**细粒度冗余**，具体包括：

- **大块数据的局部重复**：在非模型数据的大块中，即使整个数据块不完全相同，其内部也常常包含大量相同或高度相似的局部内容。
- **模型数值冗余**：主要表现为浮点数（如 BF16/FP32）的指数高位高度相似，而尾数位存在细微差异。如图 3.3 所示，通过分析 Transformer 模型文件中 BF16 浮点数各比特位的熵值<sup>①</sup>可以发现，9~14 位的指数部分熵值远低于尾数部分，表明这些位存在大量冗余，而传统方法无法利用这一点。

针对上述问题，现有的优化方案各有局限。Finesse、Odess 等系统引入了增量压缩，能更深入地挖掘块内冗余，但计算开销高昂，且未能针对模型数据进行有效处理。另一方面，专用的模型压缩方法（如 ZipNN、ZipLLM）虽然高效，但高度依赖模型数据的特定结构，无法应用于混合工作负载的云存储系统，甚至可能对非模型数据产生负面效果。基于以上分析，我们提出以下设计目标：

从功能设计出发，主要为以下三点，本章将对其进行详细阐述：

1. **块内冗余的细粒度去除**：系统应具备对大块数据的局部重复进行细粒度（如子块粒度）识别和去重的能力，以应对数据大块化的趋势。

<sup>①</sup> 熵值用于衡量该位的不确定性或不相似程度，数据越集中则熵值越低。熵为 0 表示该位在所有采样数据中取值完全相同（全 0 或全 1）<sup>[44]</sup>。

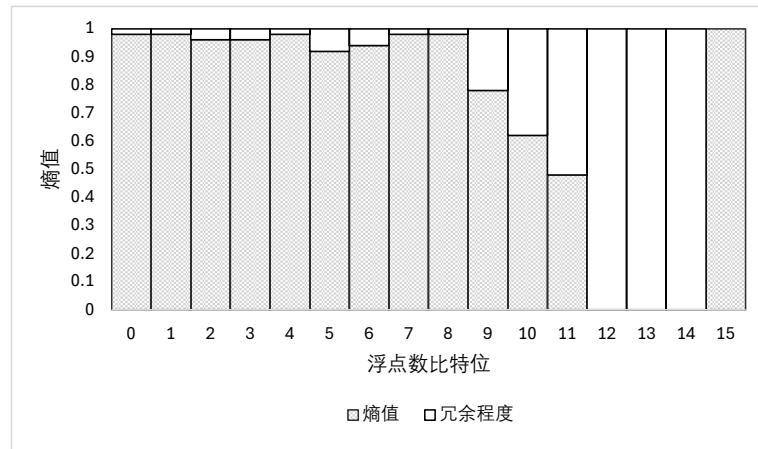


图 3.3 模型文件中 BF16 浮点数各 bit 位熵值

Figure 3.3 Bit entropy values of BF16 floating-point numbers in model files

2. 模型数据的鉴别分离：系统应具备识别模型数据，并将其从数据缩减工作流中分离出来的能力，以防止对细粒度去重造成影响。
3. 模型数据处理优化：对于被分离出的模型数据，系统应充分挖掘其数值冗余，设计相应的处理策略以进一步提升整体的数据缩减效果。

综上，系统的设计思路如图 3.4 所示。

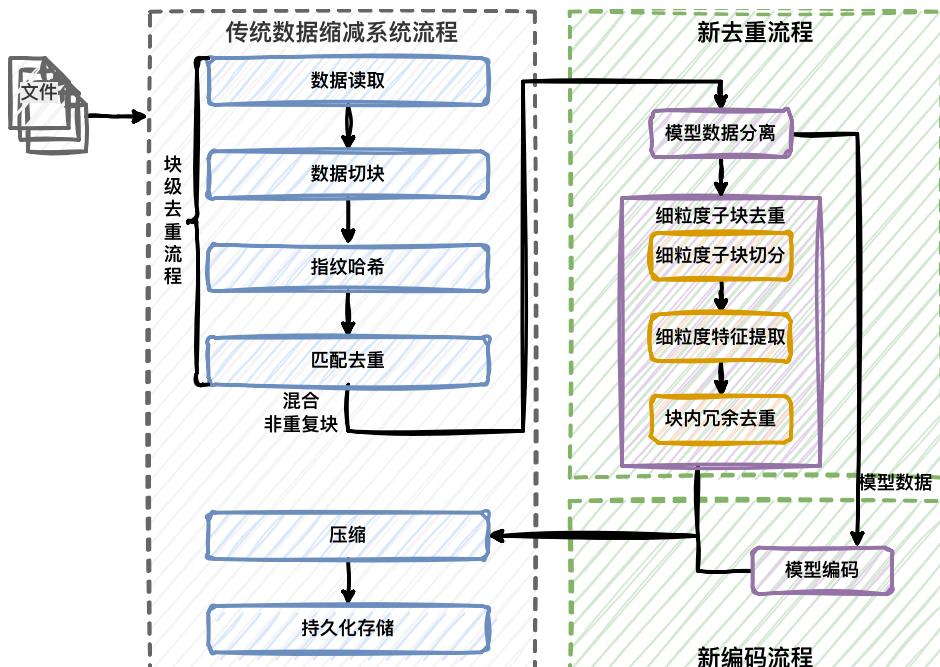


图 3.4 设计思路  
Figure 3.4 Design idea

## 3.2 系统总体设计

基于上述设计思路，本文提出了一种面向混合负载<sup>①</sup>的细粒度冗余识别数据缩减系统（Fine-grained Hybrid Redundancy Deduplication, FHRD）。如图 3.5 所示，该系统在传统块级去重和通用压缩工作流的基础上，集成了多个新模块，赋予了系统数据块局部冗余去除、模型数据鉴别分离以及模型数据编码压缩的能力。

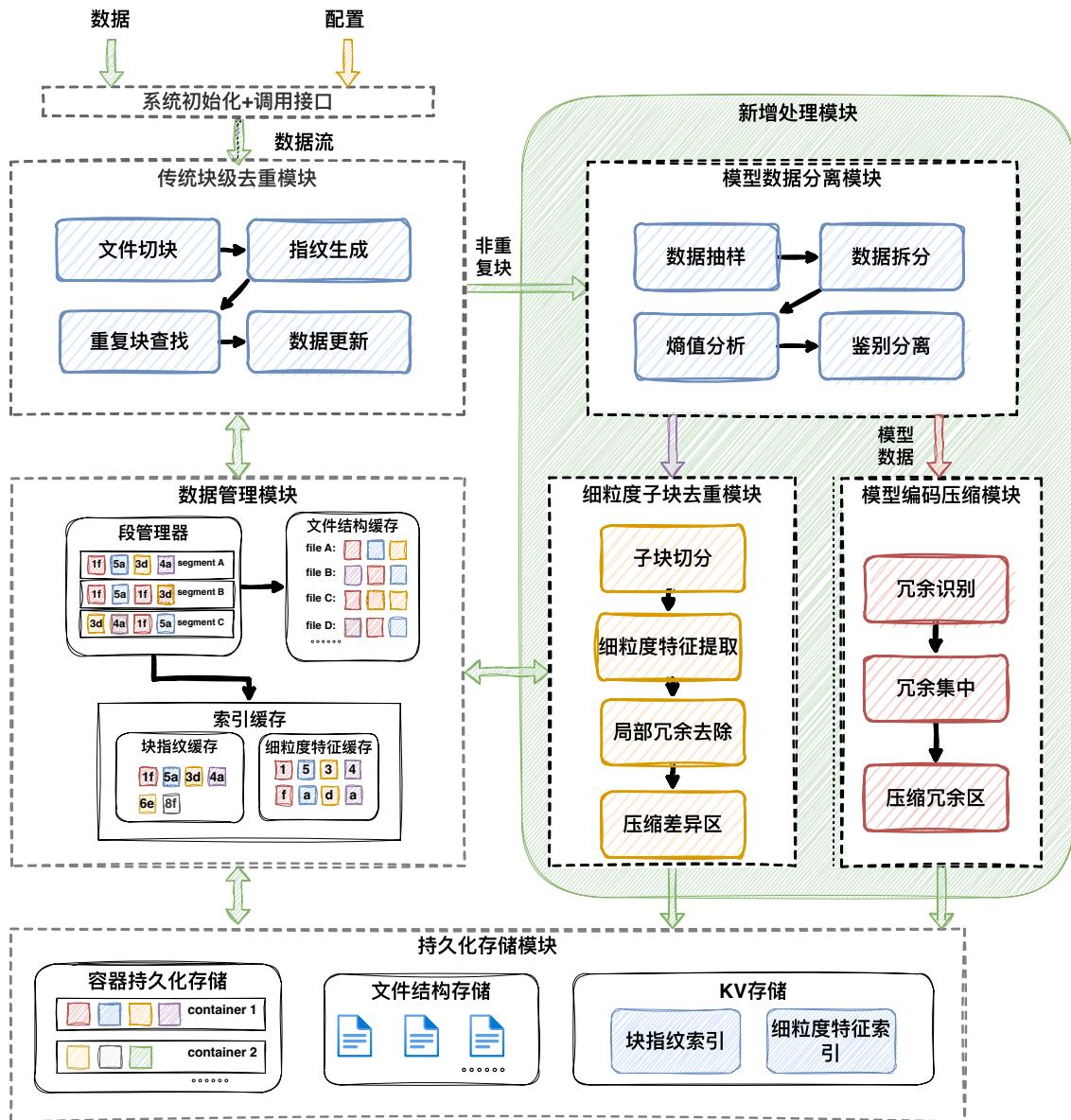


图 3.5 细粒度冗余识别的数据缩减系统设计

Figure 3.5 design of fine-grained redundancy-aware deduplication system

① 混合负载指存储目标中既有模型数据，又有传统的文本、日志、代码等数据

### 3.3 细粒度子块去重模块设计

数据块（如文本、日志、代码）通常在版本迭代中仅发生微小修改，因此不同数据块之间，尤其是在大块数据中可能存在大量局部重复内容。为有效利用此种冗余，本文设计了专门的细粒度子块去重模块，如图 3.6 所示。

该模块首先对数据块进行细粒度划分：将大块数据进一步切分为更小的子块，并通过滚动哈希为每个子块计算唯一指纹，作为数据块的细粒度特征。随后，相似块查找阶段查询细粒度特征索引，识别出与当前块具有部分相同细粒度特征的相似块。通过比较二者的细粒度特征序列，可以快速找出所有匹配的子块。

对于匹配的子块，系统将其视为冗余区域，仅保存指向参考块及其偏移量的元数据指针；对于不匹配的子块，则经由通用压缩算法压缩后存储。

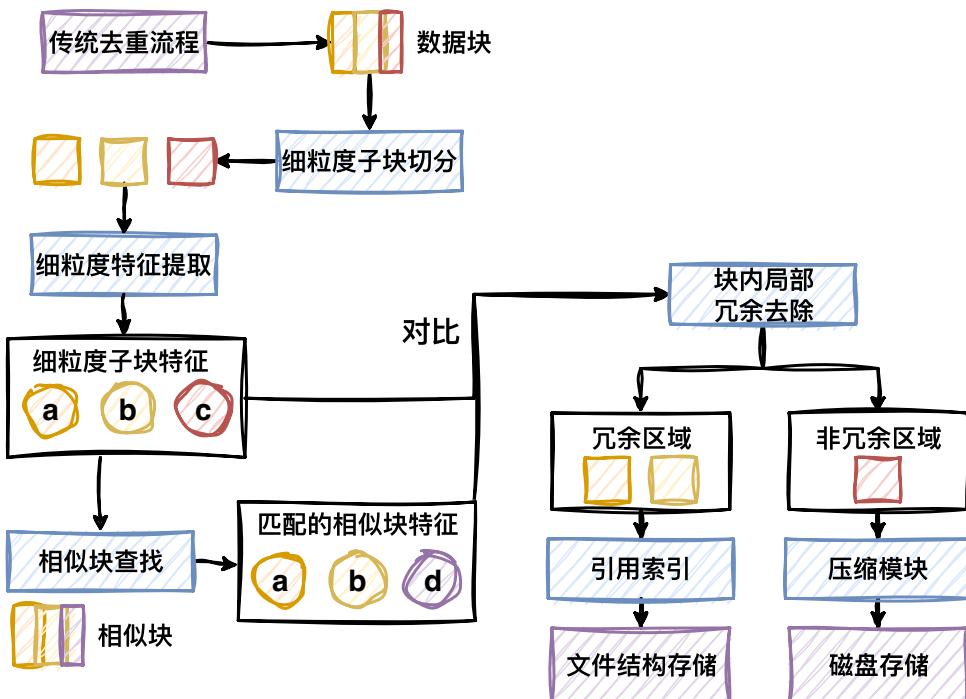


图 3.6 非模型数据处理模块  
Figure 3.6 Non-model data processing module

与传统的块级去重相比，该模块能更深入地挖掘块内冗余。与增量压缩相比，该方法在计算效率上更具优势：特征提取采用高效的滚动哈希，冗余去除则通过指纹比较快速定位并跳过整个冗余子块，避免了字节级的差分计算所需的大量开销。通过这种优化的设计，系统能够在保持较低计算成本的同时，实现对大块数据局部重复的高

效识别与处理。

### 3.4 模型数据分离模块设计

模型数据分离模块旨在精确鉴别出模型数据块，将其从数据缩减工作流中分离出来，避免对去重流程造成干扰，从而防止性能退化。同时可以对模型数据进行专门的数值冗余消除处理，进一步提升整体数据缩减效果。因此，本文所定义的模型数据，并非指具有特定文件后缀（如 `.safetensors`、`.pt`）的文件，而是指那些整体内容缺乏相似性，拖累匹配去重流程，但蕴含着显著的指数位冗余（如图 3.7 中所示，每行代表一个 FP32 格式浮点数，红框内部为指数高位），因此适合被分离出主流程，后续进行专门的细粒度数值冗余消除。

0	0111011000010000011110010001111
0	01110010111000111011001011111
0	0111011100110010011101110100000
0	01110111110111011101011111010
1	0111010011101110011101110011100
1	0111010000010001011101110011011
0	011101111000111011101110110000
0	011100100000101101110111111100
1	01110110001100110111100110111
1	011101001100111011101100100010
0	0111100101010110011101011010101
1	0111010010011010011101000011010

图 3.7 16 位浮点数二进制内容

Figure 3.7 Binary representation of 16-bit floating-point numbers

基于上述定义，若仅依赖文件格式进行判断，将导致大量误判。例如，一个归档文件（如 `.tar`）或磁盘镜像（如 `.img`）可能同时包含模型与非模型数据；反之，一个模型权重文件内部也可能包含文件头、元数据等不具备数值冗余特性的“非模型”部分。因此，本文创新性地提出采用块级鉴别的方法，深入数据内容特征进行分析。

如图 3.7 所示，模型数据块中的浮点数，其指数位通常集中在特定的比特位范围内（例如，16 位浮点数的第 9~14 位），参考图 3.3，这些位的熵值显著低于其他部分。而非模型数据块则缺乏这种清晰的熵值分布特征。利用这一差异，我们设计了基于分组熵值分析的数据类型鉴别方法，如图 3.8 所示。

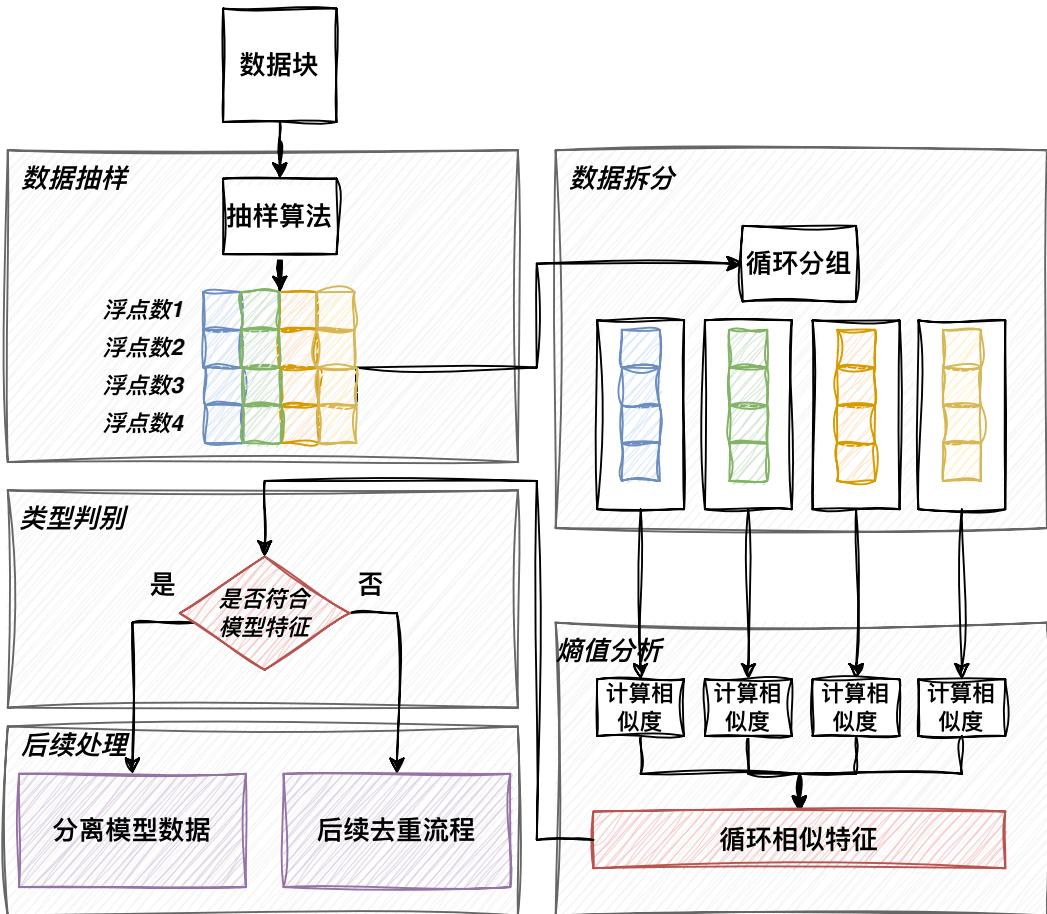


图 3.8 基于分组熵值分析的数据类型鉴别方法

Figure 3.8 Data type classification method based on grouped entropy analysis

该方法首先对数据块进行偏移抽样以降低计算开销，然后将采样浮点数按位分组，并计算每个比特位分组的熵值。通过分析熵值分布规律：

- 若第 9~14 位熵值显著低于其他位，则判定为 16 位浮点数模型数据块。
- 若第 25~30 位熵值显著低于其他位，则判定为 32 位浮点数模型数据块<sup>①</sup>。
- 若无明显低熵区间，则判定为非模型数据块。

随后将模型数据从主流程分离，送入模型编码压缩模块。

与文件级鉴别相比，在块级粒度进行数据类型鉴别具有显著优势，如表 3.1 所示：

- 精确识别：**块级鉴别能够针对数据块的实际内容进行分析，精准区分混合在同一文件中的模型数据与非模型数据，避免了文件级鉴别带来的误判风险。
- 灵活适应：**能够灵活适应包含多种数据类型的混合负载场景，确保每个数据块都得到最恰当的处理。

<sup>①</sup> 分别对应 BF16 和 FP32 格式浮点数的指数高位。

表 3.1 鉴别粒度对比表  
Table 3.1 Comparison of identification granularity

鉴别粒度	块级	文件级
自定义后缀的模型文件	能够识别模型数据	无效
<b>Q4</b> 等高熵量化模型	可识别为非模型数据，避免无效处理	误判为模型数据
<b>tar</b> 、 <b>iso</b> 等打包文件	能识别其中的模型数据部分	全部误判为非模型文件
<b>header</b> 等模型文件中的非模型部分	可识别为非模型数据，避免无效处理	误判为模型数据
集成侵入性	低，可作为独立模块嵌入块处理流水线	高，需改造文件处理逻辑

3. 易于集成：块级鉴别可以作为独立模块无缝嵌入到传统块级去重流水线中，对现有系统侵入性低，易于部署。

### 3.5 模型编码压缩模块设计

模型数据中存在的数值冗余无法被传统通用压缩算法有效识别，其根本原因在于此类冗余的独特分布模式与通用压缩算法（如 LZ 系列）的核心工作原理不匹配。通用压缩算法擅长处理字节流中的局部连续性冗余，它们通过滑动窗口识别并压缩在窗口内重复出现的连续字节序列。

然而，模型数据中的冗余模式（即重复的指数位高位）在字节流层面是非连续的。相似的指数高位分散，例如，在 32 位浮点数序列中，具有相似模式的指数位可能间隔数个字节才出现一次。这种跨字节的非连续分布，使得通用压缩算法难以捕捉到数值冗余，反而会因尝试对高度随机的尾数位进行无效匹配而引入不必要的计算开销，导致“性能退化”。

为解决此问题，如图 3.9 所示，本文采用专用编码方法对模型数据块进行处理。该方法首先对数据块中的每个浮点数进行细粒度的按位拆分，将所有比特位分别集中在一起，形成不同的比特组。其中具有高度冗余的若干组（即图 3.7 中的红色区域），构成冗余集中区。该区域由浮点数相似指数位组成，包含了大量频繁出现的相似序列，因此可以被通用压缩算法高效处理。得到数据缩减率的提升。从本质上讲，这一变换并未减少数据量，而是通过重组数据结构，将分散在模型数据块中不同位置的相似数据集中起来，变为连续的冗余，将不可压缩的数据块转化为可压缩的数据块，从

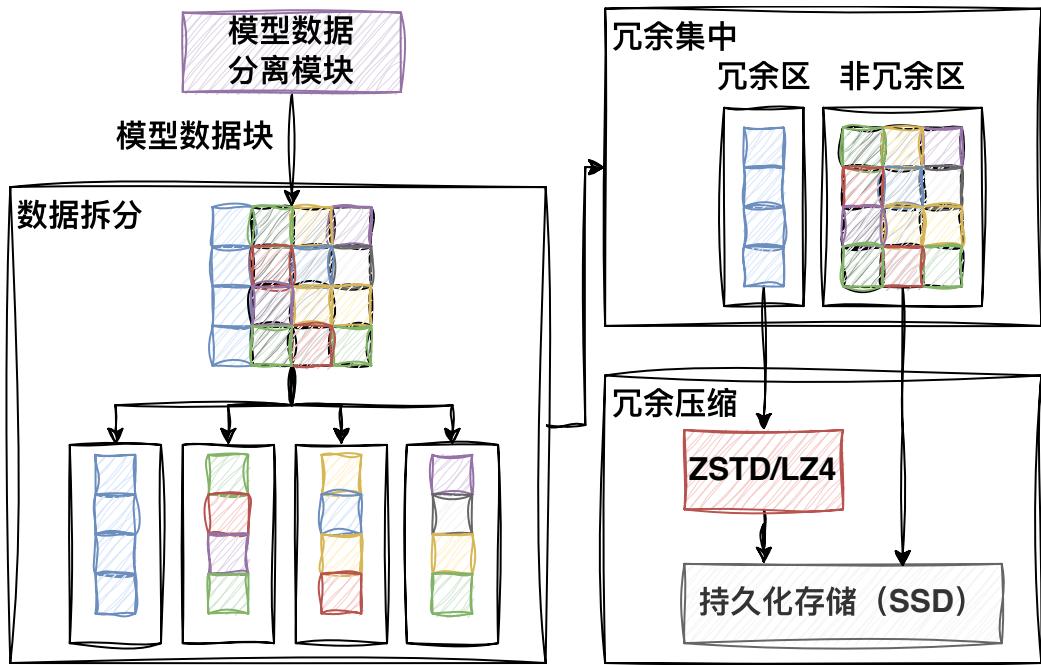


图 3.9 模型编码压缩模块  
Figure 3.9 Model encoding compression module

而使得后续的通用压缩算法能够更有效地发挥作用。

### 3.6 数据缩减工作流

在介绍系统的总体设计与各模块原理之后，本节将进一步阐明系统在处理存储请求时的具体工作流，以便理解其实际运行过程和数据在各模块间的流转与变化。如图 3.10 所示，系统的存储去重工作流涵盖了从数据接收、初步去重、模型数据鉴别分离、细粒度冗余处理，直至最终数据存储的完整闭环。

工作流始于读取与分块阶段。原始负载（例如，一个包含 AAaa、BBbb、CCcc、DDdd 和 MODEL<sup>①</sup> 数据段的文件）被读入系统，数据流经由分块器分割为若干数据块。此步骤将连续的数据流转化为离散的、易于管理的处理单元，为后续的精细化操作奠定基础。

紧接着进入块级去重阶段。系统通过哈希引擎为每个数据块计算唯一指纹，并查询全局指纹索引。此阶段旨在快速消除全局范围内完全一致的重复块。如图所示，数据块 AAaa 和 CCcc 的指纹在索引中被找到，因此被判定为重复块，系统仅保留指向已存储数据的元数据指针；而块 BBbb、DDdd 和 MODEL 块作为唯一块，记录新指

<sup>①</sup> MODEL 部分数据段为模型数据

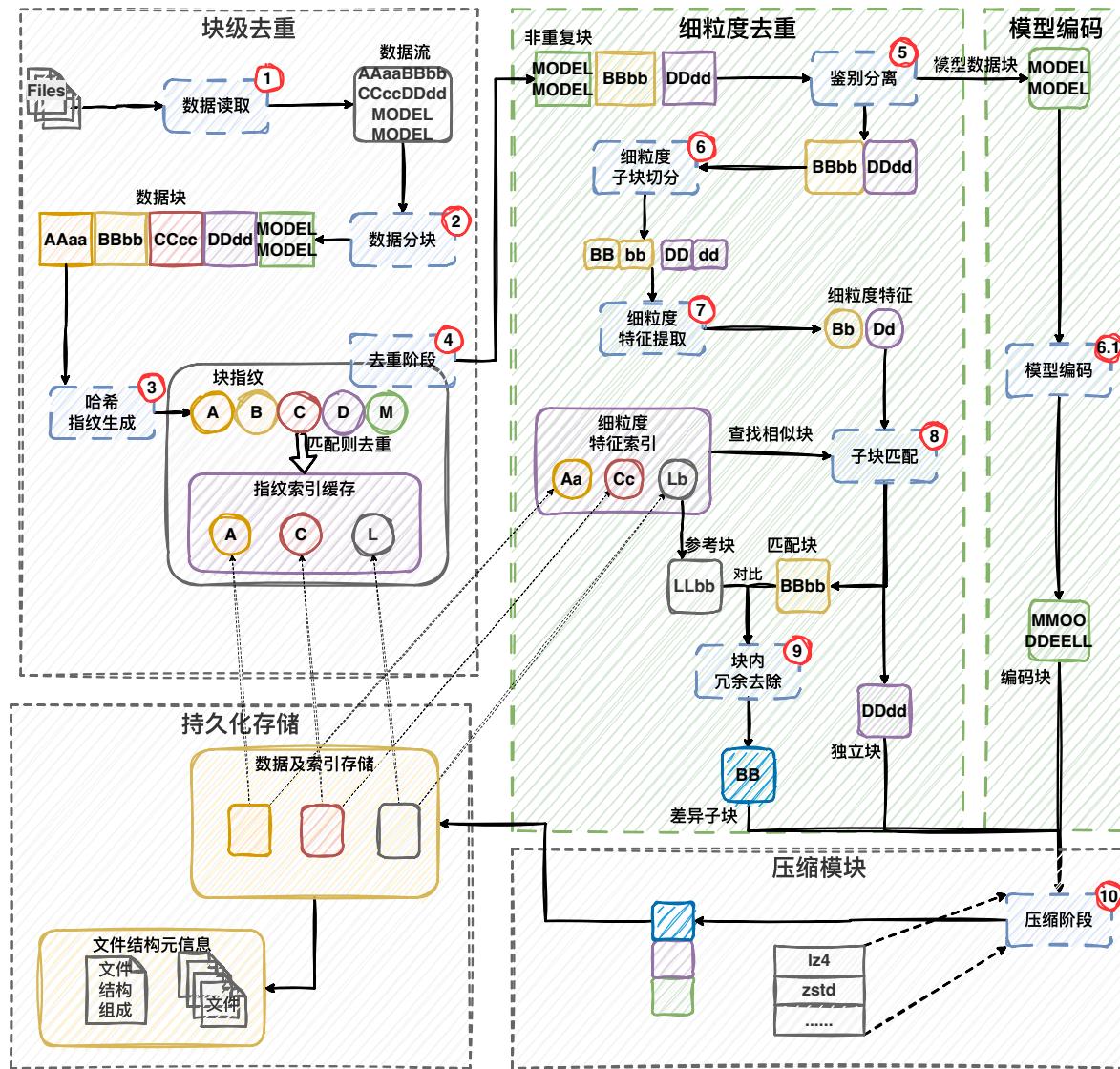


图 3.10 系统数据缩减工作流  
Figure 3.10 System data reduction workflow

纹并进入下一处理阶段。此环节显著减少了需要进一步处理的数据量。

在**细粒度去重**阶段。首先，鉴别分离模块基于数据内容特征对接收的数据块进行分析，精确地识别出“模型数据块”，并据此将数据（如 MODEL）分离出主流程，送入不同的专有处理路径。随后，BBbb、DDdd 等数据块进入细粒度去重流程，将大块进一步切分为若干子块并生成细粒度特征（如 Bb、Dd）。随后，通过查询细粒度特征索引，发现块 BBbb 的细粒度特征 Bb 与索引中的 Lb 存在部分匹配。通过依次比对细粒度特征，系统识别出匹配块 BBbb 和已存储参考块 LLbb 之间的连续冗余部分 (bb)，系统仅保存指向参考块子块的元数据指针，而非直接存储冗余内容。对于剩余的不匹配子块 (BB)，则经由通用压缩算法压缩后存储。

对于被分离出的模型数据 (MODEL)，进行**模型压缩编码**，该模块编码重组浮点数指数高位，将原始块的指数位集中排列。这一变换本身并不缩减数据大小，而是通过重组数据结构，将一个难以压缩的数据块转化为一个可压缩的数据块。

最后是**压缩与持久化**阶段。所有经过差异化处理的数据单元，包括来自模型路径的编码块、来自非模型路径的差异块和唯一块，被统一送入压缩模块，由通用压缩算法（如 LZ4、Zstd）进行最终处理，以消除剩余冗余。处理完毕的数据块被组织进容器，并与更新后的指纹索引、细粒度特征索引等元数据一同写入持久化存储，完成整个工作流。

### 3.7 本章小结

本章针对现有去重系统在处理现代云存储工作负载时所面临的挑战，首先深入分析了传统数据缩减技术在面对大块数据和模型数据时的局限性，明确了系统设计的三大核心功能目标：高效去除块内冗余、精准鉴别分离模型数据以及优化模型数据处理，提出了一种面向混合负载的细粒度冗余识别数据缩减系统 (FHRD)。

随后，本章介绍了 FHRD 的总体设计及其关键模块的设计原理。针对大块数据的局部重复问题，设计了细粒度子块去重模块，通过细粒度划分和相似块查找技术，实现了对块内冗余的深度挖掘；针对模型数据的数值冗余特性，设计了基于分组熵值分析的数据类型鉴别模块，实现了模型数据的精准识别与分离，并进一步设计了模型编码压缩模块，通过重组浮点数指数位，将不可压缩的数值冗余转化为可压缩形式。

最后，本章通过对系统存储去重工作流的详细阐述，清晰地展示了各模块如何协同工作，以及数据在处理过程中的具体变化，为后续章节的性能评估与实验验证奠定了坚实的基础。



## 第4章 系统实现与优化

在实现层面，本系统以开源的传统去重系统 Destor<sup>①</sup> 为基础，在其上扩展实现了细粒度子块去重、模型数据鉴别分离、模型数据编码压缩等核心功能，并集成了 Zstandard (Zstd) 压缩模块，最终构建了一个面向混合负载、支持细粒度冗余识别与去除的数据缩减系统。在将设计方案具体实现的过程中，我们进行了如下优化与改进：

1. **细粒度子块的划分**：为了有效去除不同数据块间的连续重复区域，系统需要对数据块进行高效的子块划分和特征提取。然而，现代块级去重系统普遍采用内容定义分块 (CDC) 以规避边界偏移问题，这导致生成的数据块长度不一。若简单地对变长块进行均匀分块，将难以保证子块大小的一致性，从而影响重复区域的识别效果。因此，本文提出了指数取整的双向子块切分方法，详见 4.1 节。
2. **高效准确的数据类型鉴别**：模型分离需要在不引入过多计算开销的前提下，准确识别出以 BF16 和 FP32 为代表的、具有浮点数指数位相似模式的模型数据。然而，按位遍历整个数据块来寻找循环相似规律的计算成本过高，难以满足性能要求。因此，本文采用一种字节粒度的分组抽样熵值分析方法，详见 4.2 节。
3. **模型数据的冗余区域识别**：系统需要有效定位模型数据中的指数高位，以便将这部分冗余集中起来进行针对性处理。但实际分块过程中，系统无法感知浮点数的具体格式，甚至可能将一个浮点数分块到两个数据块中，数据不对齐。因此，本文实现了一种基于熵值分析结论的字节分组压缩方法，详见 4.3 节。

此外，为了确保系统的整体性能与集成稳定性，我们还对系统进行了若干优化，详见 4.4 节：

1. **数据管理与缓存**：细粒度去重引入了额外的计算与存储开销，系统需要实现高效的数据管理和索引缓存机制，以应对这些新增的性能压力。
2. **数据块结构重构**：在传统去重流程的基础上，系统集成了多个细粒度处理阶段及通用压缩模块。因此，必须对数据块的存储结构进行重构，以容纳新增的细粒度元信息，并明确指示解压或解码方式。
3. **并行处理**：为充分利用多核处理器的计算能力，提升细粒度去重的处理速度，系统需要实现高效的并行处理。

---

① <https://github.com/fomy/destor>

## 4.1 细粒度子块去重模块优化——指数取整的双向子块分块方法

针对大块数据内部的连续冗余，本文受到 Burst 方法对首尾连续冗余识别的启发，提出通过提取数据子块的特征来代表连续子块内容，并通过匹配相同特征来识别连续冗余区域。然而，Burst 采用的固定长度分块无法有效应对边界偏移问题。现代块级去重系统普遍采用基于内容的分块方法，以确保数据块边界与实际内容对齐，但这导致了数据块长度不一，为子块划分带来了新的挑战。



图 4.1 定长分块的边界偏移问题  
Figure 4.1 Boundary offset problem in fixed-length chunking

定长分块会引发边界偏移问题。传统数据（如文本、日志、代码）在修改后常会产生副本，二者间往往仅在中间存在少量变化，而大部分内容保持不变，如图 4.1 所示。但在定长分块中，即使仅在中间插入或删除少量数据（如图中加粗斜体所示），后续所有分块的边界都会发生偏移（如图中黄色部分所示），导致无法匹配到重复块。

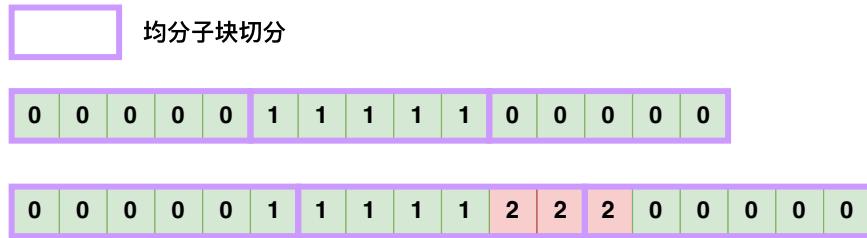


图 4.2 变长分块解决边界偏移问题  
Figure 4.2 Variable-length chunking solves the boundary offset problem

基于内容定义的变长分块方法可以有效解决此问题。通过内容哈希确定分块边界，使得插入或删除数据后，未受影响的内容仍能被正确分块为相同的数据块，从而实现重复数据的识别与去除，如图 4.2 所示。

然而，变长分块在解决边界偏移问题的同时，也给子块划分带来了挑战。

- 若采用定长子块切分，则会在块内重新引入边界偏移问题，且固定的子块长度可能不适用于各种长度的变长块。
- 若采用内容定义的变长子块分块，则会引入额外的计算开销，且滚动哈希方法面对较小的子块时可能效果不佳。
- 若简单地将数据块按固定份数均分得到子块，由于变长分块产生的数据块长度不一，则无法保证子块大小的一致性，难以匹配冗余区域，如图 4.3 所示，图中相似数据块仅在红色区域有修改，但均分子块无法匹配。



**图 4.3 固定份数为 3 的子块划分**  
**Figure 4.3 Sub-block division into 3 fixed parts**

基于上述考量，本文提出了一种指数取整的双向子块定长分块方法，以高效地将变长数据块划分为定长子块，同时缓解边界偏移问题。

具体而言，系统首先根据变长数据块的长度，计算出最接近且不大于该长度  $1/10$  的 2 的幂次方，作为该块的子块长度。在大小相近的相似块中，该子块长度通常为相同的定值，避免了类似图 4.3 中的问题。同时，由于子块长度适应性地设定为 2 的幂次方，能够较好地适应不同长度的数据块，避免了过大或过小的子块划分。

然后，系统从数据块的起始和结束位置分别向中间方向，以该子块长度进行定长分块，得到若干子块，并计算其哈希指纹作为特征。在相似块的冗余匹配过程中，也同时进行两个方向的子块特征值匹配。由于相似块的修改通常集中在中间的连续区域，这种双向分块方法能够在一定程度上缓解边界偏移问题，因为即使数据块在中间部分发生了插入或删除，双向匹配仍能识别出相同的子块，从而定位连续的冗余区域。

如图 4.4 所示，以定长“5”作为子块大小，从正向和反向两个方向进行分块处理，最终得到正向子块特征值（00、11、23、34）和反向子块特征值（33、12、01、0）。在待存储块和相似块的匹配过程中，系统也将同时利用这两组特征值进行冗余区域的识别与定位。正向的 00、11，反向的 33 所对应的子块（图中绿色部分）即为局部连续冗余，可以被去重。

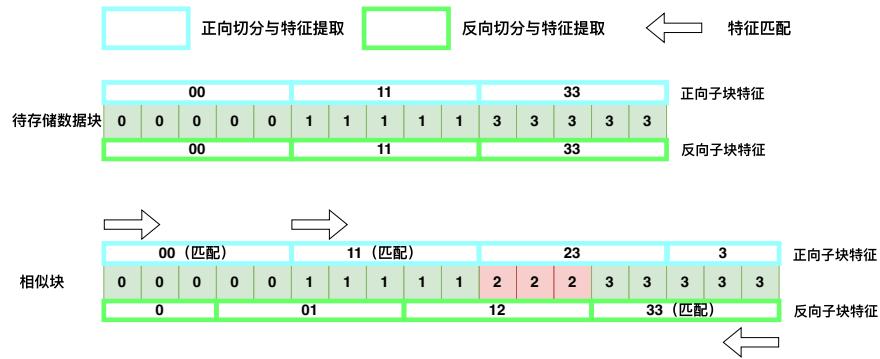


图 4.4 指数取整的双向子块定长分块方法

Figure 4.4 Exponential rounding bidirectional sub-block fixed-length segmentation method

## 4.2 模型数据分离模块优化——字节粒度的分组抽样熵值分析方法

为了有效分离模型数据，系统需要检测数据块各比特位的熵值以衡量其相似性。然而，直接在比特粒度上进行操作存在两大难题。首先，对整个数据块进行频繁的位操作会显著增加实现的复杂性和计算开销。其次，如果采用抽样方式计算熵值，小样本中的某些比特位可能因偶然性而表现出较低的熵值，从而导致误判，如图 4.5 所示。

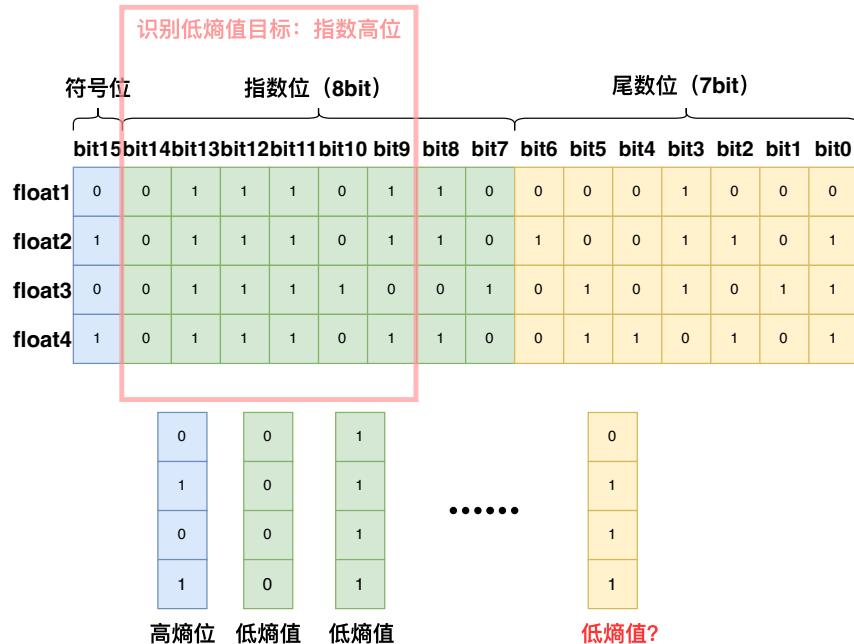


图 4.5 比特粒度实现的复杂性

Figure 4.5 Complexity of bit-level implementation

为解决此问题，我们通过分析不同类型的模型数据，发现其浮点数的冗余和表示形式具有明确的规律性：

1. **数值冗余规律**：回顾图3.3和图3.7，可以发现存在去重空间的模型数据块，其冗余主要集中在指数高位。这些高位表现出极高的相似度，即较低的熵值。从模型训练的角度看，归一化层将激活值约束在零均值、单位方差的分布附近，使得绝大多数数值落在有限范围内（如[-1, 1]）。随着模型收敛，权重更新幅度减小，数值分布更趋集中。在浮点数表示中，这种特性直接体现为指数高位部分较为相似。
2. **表示形式规律**：如图4.6所示，训练过程中产生的、存在数值冗余的模型文件，其浮点数格式主要为FP32、BF16和FP16。在这三种表示方法中，指数高位恰好完整地分布在后一字节内部，构成低熵字节。同时，这三种浮点数的长度分别为4字节或2字节，这意味着数据块中的低熵字节会以4字节（FP32）或2字节（BF16/FP16）的周期重复出现。

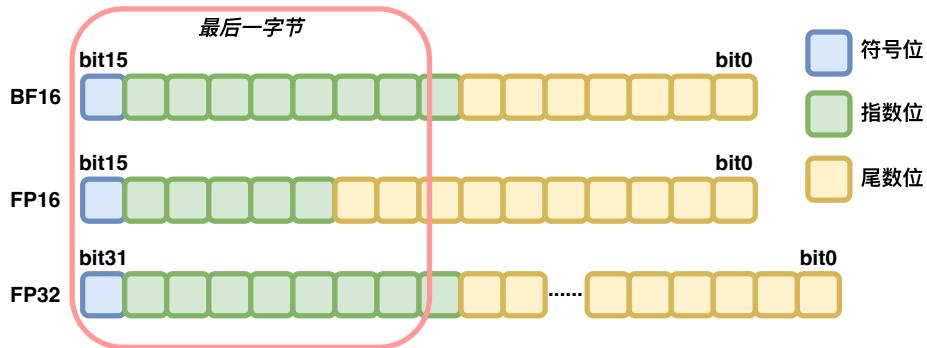


图4.6 模型浮点数的表示形式  
Figure 4.6 Representation of model floating-point numbers

基于上述规律，我们提出了一种字节粒度的分组抽样熵值分析方法，以实现高效、准确的数据类型鉴别。该方法首先对数据块进行抽样，每个抽样点相对于块头的偏移量均为4字节的整数倍。然后，将抽样数据按字节逐一分为4组，并分别计算每组字节的熵值。最后，系统根据熵值分布情况判断数据块类型：

- 若出现一组低熵字节组和三组高熵字节组，则判定为FP32类型的模型数据块。
- 若出现两组低熵字节组，且组索引之差为2，则判定为BF16或FP16类型的模型数据块。

如图4.7所示，四组中第0组，第2组为低熵字节组，说明其具有较高冗余，对应着浮点数的指数高位部分，该数据块为模型数据。每4字节出现两次指数高位说明

该浮点数大小为 2 字节 (BF16/FP16)。

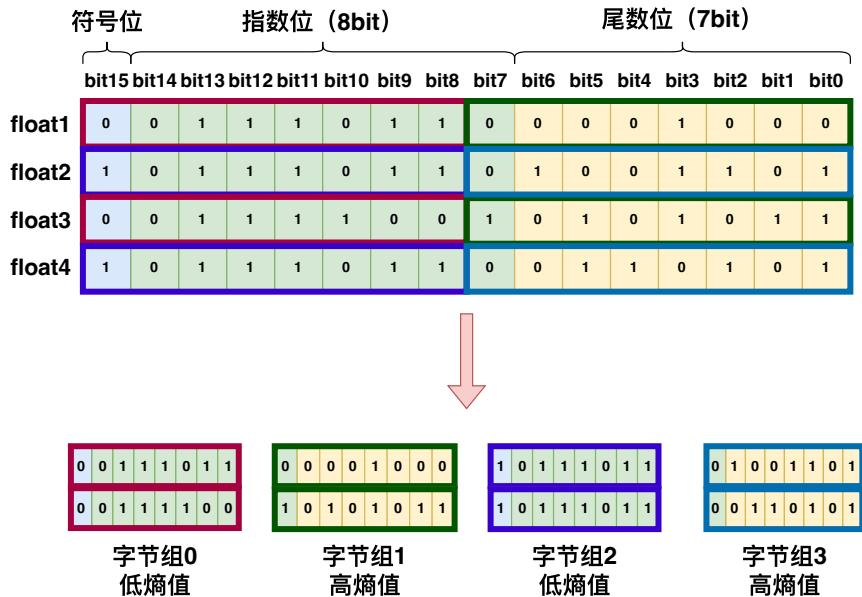


图 4.7 字节粒度的分组抽样熵值分析  
Figure 4.7 Byte-wise grouped sampling entropy analysis

与原设计相比，该实现方法具有以下优点：

1. **实现简单：**字节粒度的处理避免了复杂的位操作，简化了实现难度，提升了鉴别过程的计算效率。
2. **高效准确：**通过字节分组，利用指数高位在字节内的集中特性，形成了显著的低熵字节组，而其他部分则构成高熵字节组。这种设计有效避免了因采样过小而导致的尾数位偶然低熵对鉴别结果的干扰，从而显著降低了误判率，并允许在保证精度的前提下进一步缩减采样点数量，降低计算开销。

### 4.3 模型编码压缩模块优化——基于熵值分析结论的字节分组压缩方法

模型数据处理模块的目标是识别并集中模型数据块中数值相似的浮点数指数位，以便利用通用压缩算法进行数据缩减。与图 4.7 的流程类似，该模块也对整个模型数据块按字节进行拆分，分为 4 组，从而将指数位数据集中在特定组内，形成可压缩的冗余部分。

然而，由于数据块分块的盲目性，系统既无法预知浮点数的具体类型，也无法保证其在数据块中的对齐，甚至可能将一个浮点数分块到两个数据块中。因此，系统难

以直接确定哪一组是指数位所在组。若像鉴别阶段一样重新计算分组熵值，将因处理整个数据块而产生巨大的计算开销。ZipNN 等相关工作选择对所有组进行压缩，但这会引入对非冗余高熵组的无效压缩运算。

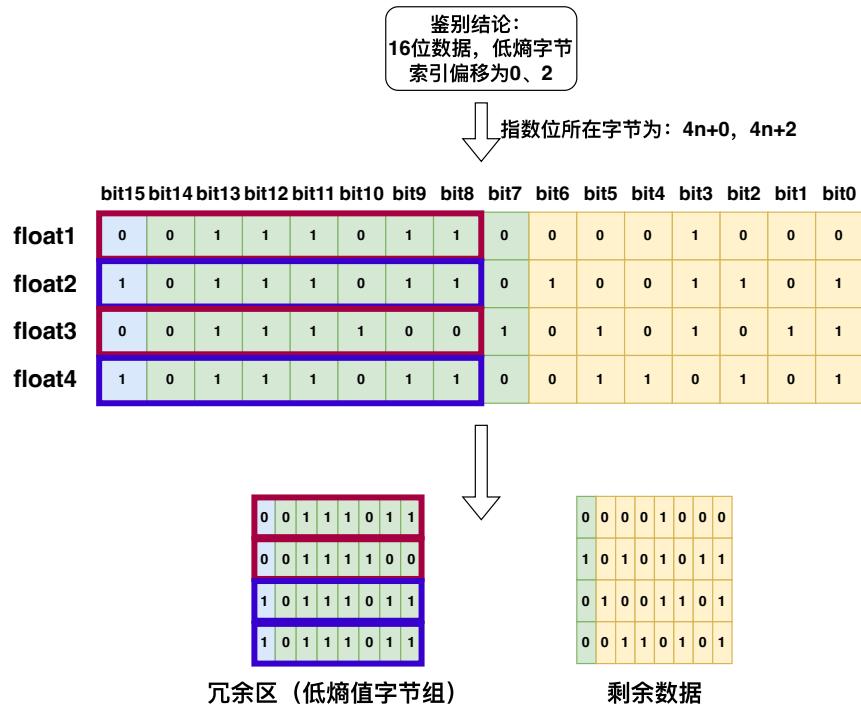


图 4.8 基于熵值分析的字节分组压缩  
Figure 4.8 Byte-wise grouped compression based on entropy analysis

本文提出了一种基于熵值分析结论的字节分组压缩方法，以高效识别指数位所在组并进行针对性压缩。正如前一小节所述，系统在模型数据鉴别分离阶段已经计算出各组的熵值分布，并获得了低熵字节组的索引（如图 4.7 中的第 0 组，第 2 组），该索引实际上指明了浮点数指数位所在字节相对于数据块的偏移量。因此，如图 4.8 所示，在模型数据处理阶段，系统可以直接复用这一结论，仅提取低熵字节组作为冗余区域进行压缩，从而显著提升压缩效率。与盲目压缩所有组的方法相比，该方法对 32 位数据块缩减了 75% 的压缩输入，对 16 位数据块缩减了 50% 的压缩输入。

## 4.4 其他系统优化

### 4.4.1 数据管理与缓存优化

细粒度去重流程引入了额外的计算和存储开销，因此系统采用了高效的数据管理和索引缓存策略以提升整体性能。系统通过数据段（Segment）在逻辑上聚合相邻数据块，并通过容器（Container）在物理存储上管理临近数据块，二者协同工作以提升 I/O 局部性与批处理效率。同时，系统设计了多级索引缓存策略，以减少对底层存储的随机访问次数和索引查找延迟。

#### 1. 局部聚合的数据块批处理

随着存储规模的持续增长，单个数据块的随机 I/O 和逐一索引查找所带来的开销愈发显著。为此，系统将数据流按逻辑邻近性划分为若干数据段（Segment），每个数据段代表一组“相邻”的数据块，既作为预取的单位，也作为批处理的基本粒度。

一方面，当发现某个数据块重复或相似时，其同一数据段内的其他块很可能也具有相同或相似的属性。以数据段为单位预取对应的容器或索引条目，可以显著减少随机读次数与索引查找延迟，从而提升 I/O 局部性与缓存命中率。

另一方面，将数据按数据段聚合后作为批量单元传递给后续模块（如去重、重写与写容器），可以将多次对单块的重复访问合并为一次批处理操作，从而降低每个数据块的平均 I/O 与计算开销，并提高整体吞吐量与并行效率。

#### 2. 基于容器的磁盘信息管理

与数据段管理类似，容器（Container）是系统的基本物理存储单元，它将若干去重后的数据块按一定策略打包并持久化到后端存储。容器兼具数据布局与元数据管理两项职责。

首先，容器通过将相邻或逻辑相关的多个块聚合到同一个写入单元，提升了写入的顺序性，减少了系统对底层介质的每块随机写入与读回次数，从而降低了每个块的随机 I/O 开销。

其次，容器保存了必要的元信息（如每个块的偏移、长度、校验和，以及块指纹到容器内偏移的映射），以便在后续的读取操作中快速定位与检索块数据。同时，可以基于容器级别的预取策略一次性缓存相关容器或其索引条目，将对单块的随机读转化为对整个容器（或容器元素索引）的少量顺序读，从而大幅降低随机访问与底层索引查找的成本。

#### 3. 多级索引缓存策略

为了应对细粒度去重带来的额外索引查找开销，系统采用了多层次的索引缓存

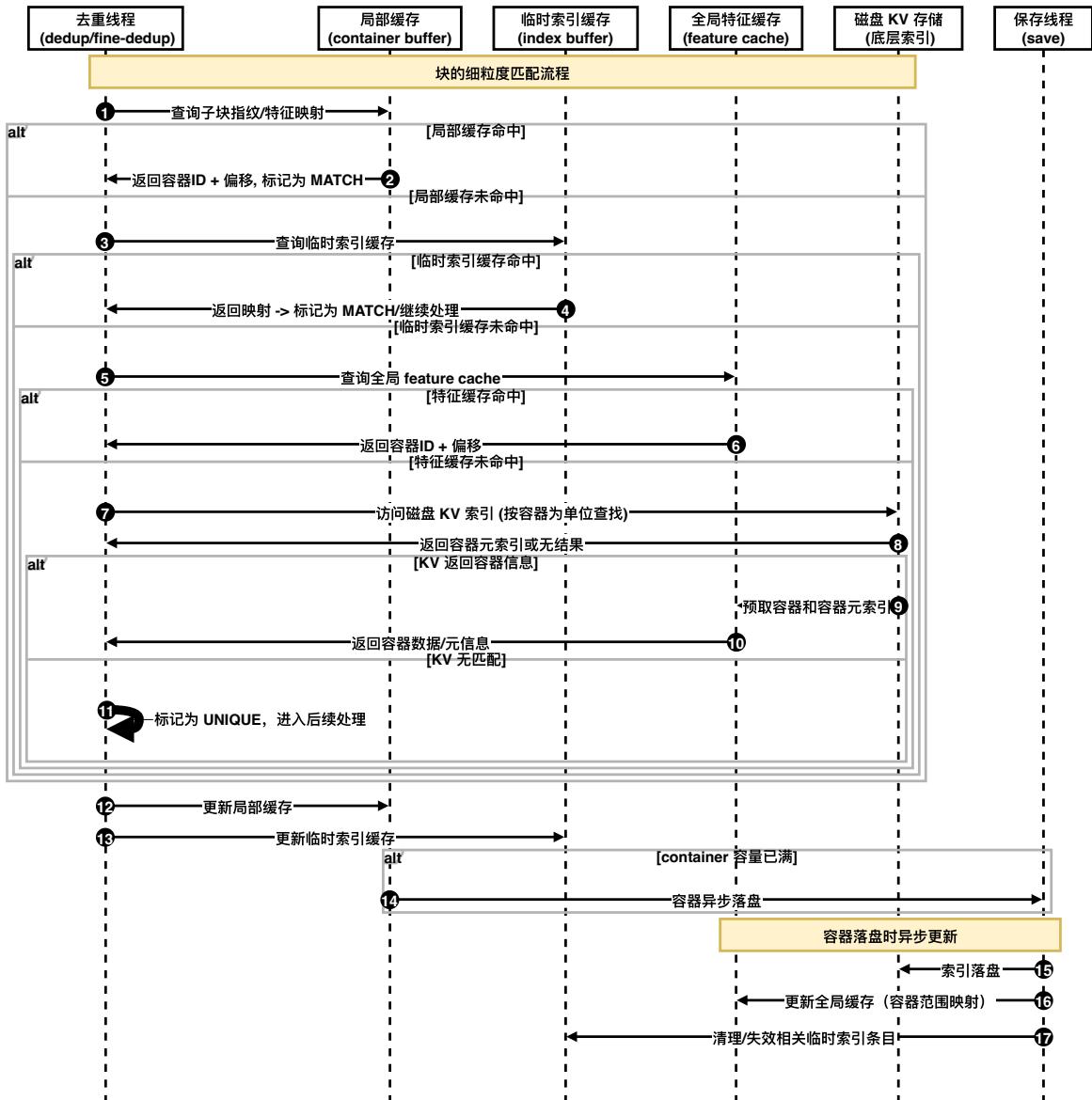


图 4.9 多级索引缓存下的匹配查找时序  
Figure 4.9 Matching lookup timing with multi-level index cache

策略, 如图 4.9 所示, 以提升索引查找效率, 减少对底层存储的随机访问次数与延迟。

最底层是磁盘中的键值 (KV) 索引存储, 负责存储数据指纹或特征到容器位置的映射关系。在此之上, 系统设计了两级内存缓存:

- **一级缓存 (全局缓存):** 采用 LRU (最近最少使用) 策略保存最常用的数据指纹或特征的映射关系, 并以容器为单位从底层存储预取数据, 以提升整体缓存命中率。
- **二级缓存 (局部缓存):** 以数据段为单位保存当前正在处理的、尚未落盘的容器内的数据指纹或特征的映射关系, 以减少对底层存储的随机访问次数与索引查找延迟。

此外, 由于系统采用流水线并行处理, 可能存在已完成去重处理但尚未落盘的容器。这部分容器对应的指纹或特征映射尚未在全局缓存中更新, 但又可能超出了局部缓存的范围。因此, 介于二者之间, 系统还设计了临时索引缓存, 以避免重复计算与查找。当存储线程将容器落盘后, 会更新全局缓存, 并清理临时索引缓存中的对应条目 (详见 4.6 节对流水线并行的描述)。

在一次重复块匹配或相似块查找的过程中, 系统会依次查询局部缓存、临时索引缓存和全局缓存。若均未命中, 则访问底层的 KV 存储, 并以容器为单位更新全局缓存。

#### 4.4.2 数据块重构

为了支持细粒度去重, 系统对数据块的结构进行了重构, 增加了新的元信息以指示解压/解码方式并存储细粒度信息。

具体而言, `size` 字段表示数据块的原始大小, `data` 指向数据块内容。`features` 字段仅在数据块类型为 **MATCH** 时生效, 用于存储细粒度特征信息, 即存储两个指向双向子块特征数组的指针。`block_type` 字段表示数据块的类型(如唯一块 **UNIQUE**、重复块 **DEDUP**、模型块 **MODEL**、非模型匹配块 **MATCH** 等)。`container_id` 字段表示数据块所在的容器标识, 而数据块在容器内的偏移位置则由容器元数据管理。`fingerprint` 字段存储数据块的哈希指纹, 用于唯一标识数据块内容。`ref_info` 字段存储冗余差分信息, 包含基准块 ID 以及基准块子块特征与本数据块的匹配情况。以图 4.4 为例, 记录匹配子块索引为 1、2、-1 (负值表示反向子块索引)。具体的子块偏移量可以通过子块大小和索引计算得出, 而子块大小则可根据 4.4 节中描述的方法由 `size` 字段计算得到。`model_info` 字段仅在数据块类型为 **MODEL** 时生效, 用于存储模型数据的熵值分析结论, 即低熵字节组的索引。正值表示 16 位浮点数的指

数位组索引为 `model_info` 和 `model_info+2`, 负值则表示 32 位浮点数的指数位组索引为 `-model_info`。`compress_type` 表示去重后数据的压缩方法和预期的解压缩方法, 默认为 0, 表示 ZSTD 压缩, 其他值可扩展为其他压缩算法。

#### 4.4.3 数据缩减流水线并行

为了提升系统的整体处理性能, 本文在整个端到端流程中引入了流水线并行的处理方式。具体实现参考图 3.10, 图中每个蓝色圆角矩形代表一个处理阶段, 每个阶段均由不同的线程并行处理, 线程间的数据通信则通过全局队列实现。具体流程如下:

- **读取 (Read) 线程:** 将待存储数据源分割为数据流, 并推入 `read_queue`, 供下一阶段并行消费。
- **分块 (Chunk) 线程:** 从 `read_queue` 读取数据, 通过内容定义分块方式进行分块, 为每个数据块生成元数据 (块大小、位置指针等), 并聚合成段 (**Segment**) 后送入 `handler_queue`。
- **哈希 (Hash) 线程:** 以段为粒度从 `handler_queue` 获取数据, 计算每个数据块的哈希指纹, 填充段内各块的元数据, 并更新数据索引。
- **去重 (Dedup) 线程:** 按顺序对每个块检查四层缓存或索引 (如 4.4.1 节所述), 检测其是否为重复数据块, 并更新相应元数据, 将其 `block_type` 标记为 DEDUP。
- **鉴别 (Identify) 线程:** 对队列中所有 `block_type` 不为 DEDUP 的数据块进行类别鉴别 (如 4.2 节所述), 将模型数据块的 `block_type` 标记为 MODEL, 分离出去重流程, 并更新其低熵索引元数据。
- **子块去重模块:**
  - **细粒度哈希 (Fine-hash) 线程:** 数据块完成双向分块和特征提取 (如 4.1 节所述), 并更新数据块的细粒度特征元数据。
  - **细粒度去重 (Fine-dedup) 线程:** 依据细粒度特征元数据, 查找 4.4.1 节所述的缓存索引结构, 匹配相似数据块, 并通过子块匹配去除局部重复冗余。
  - **冗余差分 (Diff) 线程:** 提取非冗余区域, 并记录在块的元信息中。
- **模型编码 (Model Encode) 线程:** 对标记为 MODEL 的模型数据块进行基于熵值分析结论的字节分组压缩 (如 4.3 节所述)。
- **压缩 (Compress) 线程:** 从 `handler_queue` 中读取需要写入的原始数据, 包括编码后的模型块、局部去重后的差异块以及唯一块。对这些数据块进行压缩后, 推入 `save_queue`。

- **存储 (Save) 线程**: 将最终需要写入的压缩数据块聚合到容器 (Container) 中, 更新缓存和文件结构元信息, 并清理临时索引缓存中的对应条目。

## 4.5 实现侵入性分析

本节从三方面分析本系统设计对已有块级去重流水线的侵入性: 集成灵活性、可配置性以及性能影响与缓解策略。目标是在尽量不破坏原有流程和接口的前提下, 提供可插拔、可调优的细粒度去重能力, 方便在不同部署场景中权衡吞吐、延迟与去重收益。

模块化与流程隔离: 系统以模块化形式新增细粒度处理、数据鉴别分离、模型编码等功能模块, 这些模块在逻辑上位于传统块级去重与压缩之间。默认情况下, 原有的读/切块/哈希/去重/压缩/写入流程保持不变; 当开启细粒度功能时, 仅将“唯一块”通过队列传递给鉴别模块并在处理后返回压缩模块。该设计保证了对原流程的最小侵入性: 未启用时不改变已有数据路径, 启用时仅在必要路径插入处理。

为了兼顾不同工作负载与部署资源, 系统将关键策略参数化, 可在运行时或通过配置文件调整, 以下参数均以配置文件或运行时接口暴露, 便于在不同场景下快速调优以平衡压缩收益与性能开销, 也方便实现不同对照组以比较优化效果。

- 分块相关: 是否开启内容定义分块 (CDC) 或使用固定分块; 变长块最小/最大阈值; 双向子块划分的子块最小比例 (例如: 1/10) 与指数取整策略。默认: 保留原有系统的 CDC 配置, 细粒度子块长度按 1/10 并向下取最近的 2 的幂。
- 子块策略: 子块最小/最大尺寸、哈希窗口与滚动哈希步长, 以及双向匹配的最大偏移容忍度。默认: 子块长度按章节中描述的指数取整策略, 滚动哈希步长按字节对齐。
- 鉴别策略: 采样率 (抽样字节数/块)、抽样偏移步长 (4 字节对齐默认)、熵判定阈值与最小采样点数; 允许将鉴别算法切换为“保守”“平衡”“激进”三档以控制误判率与计算开销。默认: “平衡”档。
- 模型编码/处理: 是否对所有组进行压缩或仅压缩低熵组, 压缩前是否先进行字节重组。默认: 仅压缩鉴别出的低熵组。
- 压缩后端: 可选择 ZSTD/LZ4/无压缩等。默认: ZSTD。

系统在引入细粒度去重功能的同时, 尽量减小对整体吞吐与延迟的影响。主要考虑以下两个方面:

运算方面: 额外的细粒度处理流程引入计算开销, 通过以下优化策略缓解:

- 子块划分（双向定长分块）：通过指数取整的定长分块，避免了内容定义子块划分的高计算开销，同时缓解边界偏移问题。
- 细粒度鉴别分离（熵计算、抽样）：通过字节粒度分组，减少位操作复杂度，并降低了抽样数据量以降低计算量。
- 模型编码（分组压缩）：利用鉴别阶段的熵值结论，避免对非冗余高熵组的无效压缩运算，减少 50%–75% 的压缩输入。
- 流水线并行处理：通过多线程流水线设计，充分利用多核处理器资源，提升整体处理吞吐，尽可能掩盖细粒度处理的延迟开销。  
内存方面：细粒度特征存储与索引增加了内存使用，通过以下优化策略缓解：
  - 多级缓存策略：通过局部缓存、临时索引缓存与全局缓存的协同工作，减少对底层存储的随机访问次数与延迟。
  - 数据段与容器管理：通过数据段聚合与容器化存储，提升 I/O 局部性与批处理效率，降低每块的随机 I/O 开销。

## 4.6 本章小结

本章详细阐述了细粒度冗余识别的去重系统的具体实现细节与优化方法，包括指数取整的双向子块切分方法、字节粒度的分组抽样熵值分析、基于熵值分析结论的字节分组压缩、多级索引缓存策略、数据块结构重构以及流水线并行处理等。最后，分析了系统设计对已有块级去重流水线的侵入性，强调了其集成灵活性、可配置性以及性能影响与缓解策略。通过本章的实现与优化，系统在实际运行中能够高效识别与去除细粒度冗余数据，为后续的性能评估与实验提供了坚实的基础。



## 第 5 章 实验结果与分析

本章对 FHRD 系统进行了全面的实验评估。其中 5.1 节介绍了实验环境与测试方法，说明了本章选用的对照系统及数据集。5.2 节展示了在不同混合类型工作负载下的去重效果和性能表现，并与其他相关工作进行了对比与分析。5.3 节通过对比消融实验，深入分析了指数取整的双向子块定长分块方法、模型数据鉴别分离、基于熵值分析结论的字节分组压缩方法等关键技术在不同类型负载下的实际效果，验证了各优化技术的优越性，以及系统在各种负载数据集上的适应情况。5.4 节通过控制混合负载比例和块大小等参数，深入探讨了这些因素对 FHRD 效果的影响，说明了本文优化对 3.1 节中所论述的现有云存储负载问题的解决效果。

### 5.1 实验环境与测试方法

#### 5.1.1 实验环境

本章实验所使用的服务器配置如表 5.1 所示。测试服务器配备了两个 AMD EPYC 7763 处理器，每个处理器拥有 64 个物理核心。服务器配备有 128GB 的 DRAM、2 个 1TB 英特尔 SSDSC2KB9 型号的 SSD、5 个 16TB 的东芝 MG08ACA 型号的 HDD。操作系统为 Ubuntu 24.04.2 LTS，内核版本为 6.14.0。

表 5.1 实验服务器配置  
Table 5.1 Configuration of the experimental server

配置项	配置内容
CPU	2 × AMD EPYC 7763 (64 cores each)
DRAM	4 × 32GB DDR4 RAM
磁盘	2 × 1TB SSD + 5 × 16TB HDD
操作系统	Ubuntu 24.04.2 LTS (Kernel 6.14.0)

#### 5.1.2 测试对照方案

本文在实验中设置了若干对照组，如表 5.2 所示。BASE 代表主流的传统方案：在开源系统 Destor 的基础上集成 ZSTD 压缩，体现当前工业界常用的变长分块 + 块级去重 + 通用压缩流水线。FHRD 为本文提出的方法，在块级去重基础上集成了细粒度

子块去重、数据类型鉴别分离、面向模型数据的字节分组编码等改进。为评估优化技术的贡献，我们设计了消融对比（FHRD-1、FHRD-2）分别去除或替换关键子模块以量化其影响。

另外，为了与已有细粒度或专用方法做对比，在非模型数据去重方面我们在系统中实现了 BURST 及其与变长分块结合的变体（BURST+），并将 ODESS（全局字节粒度增量压缩）纳入对照以代表理想化的增量压缩上界；需要指出的是，ODESS 在计算与内存开销上远高于在线云存储可接受的范围，因此更适合作为效果上的衡量，而非直接可部署的在线方案。

针对模型数据的专用编码压缩方法（如 ZIPNN、DIFFNN），为公平比较我们也实现了若干组合变体（ZIPNN+、ZIPNN++），以评估专用编码方法在与块级去重及变长分块流水线协同下的实际收益。

**表 5.2 对照方案**  
**Table 5.2 Comparison of the tested schemes**

对照组	方案说明
BASE	基础对照组，在开源系统 Destor 的基础上集成 ZSTD 压缩算法，代表目前工业界主流的变长块级去重加通用压缩的传统方案。
FHRD	本文提出的面向混合负载的细粒度冗余识别的去重系统，设计实现参照第 3、4 章。
FHRD-1	FHRD 消融实验版本 1，模型处理模块去除基于熵值分析结论的字节分组压缩，改为全部压缩。
FHRD-2	FHRD 消融实验版本 2，非模型处理模块去除指数取整的双向子块定长分块方法，仅均分子块。
BURST	面向块内突发修改的去重系统，采用头尾指纹技术去除大块边缘冗余。
BURST+	BURST + 变长分块技术
ODESS	全局匹配的字节粒度增量压缩去重系统。
ZIPNN	针对模型文件的编码去重系统。
ZIPNN+	ZIPNN + 块级去重
ZIPNN++	ZIPNN + 变长分块 + 块级去重
DIFFNN++	与 ZIPNN 原理类似，但具有文件级模型数据鉴别的编码去重系统，仅对模型文件进行编码压缩 + 变长分块 + 块级去重

### 5.1.3 工作负载说明

本文实验负载选取了多种公开数据集，涵盖了典型的非模型数据和模型数据，用以评估系统在各种不同类型混合负载下的去重效果，并对比相关优化在不同数据集上的效果。此外，本文还设计了人工数据集 `dataset` 和训练数据集 `checkpoints`，以便在定量实验中更精细地控制模型数据占比。具体数据集信息如表 5.3 所示。

表 5.3 负载数据集  
Table 5.3 Load datasets

数据集	数据简介	数据类别
<code>backup</code>	多个版本的容器镜像文件。	非模型数据
<code>vmi<sup>a</sup></code>	多个版本的虚拟机镜像文件。	非模型数据
<code>code<sup>b</sup></code>	多个版本的开源工具（ <code>gcc</code> 、 <code>gdb</code> 等）源代码文件。	非模型数据
<code>lnx-ker<sup>c</sup></code>	多个版本的 Linux 内核源码文件。	非模型数据
<code>lnx-tar</code>	多个版本的 Linux 发行版 tar 包文件。	非模型数据
<code>gzip</code>	多个版本的 <code>gzip</code> 压缩文件。	非模型数据
<code>webh</code>	多个日期的某学校网页文件，包含大量 HTML、CSS 和 JavaScript 格式的网页内容。	非模型数据
<code>dataset</code>	由随机内容文件经过多次随机修改生成的，包含多个版本的人工数据集，便于定量分析。	非模型数据
<code>checkpoints</code>	<code>transformer</code> 模型训练产生的多版本检查点文件，模型大小由人工控制，便于定量分析。	模型数据
<code>model16<sup>d</sup></code>	16 位浮点数格式的 <code>Qwen3</code> 模型切片	模型数据
<code>modelQ4<sup>e</sup></code>	<code>Q4</code> 量化格式的 <code>Gemma-3</code> 模型切片	无冗余模型数据
<code>mixQ4FP32<sup>f</sup></code>	混合量化格式模型切片， <code>GPT-OSS-120</code> 模型为主。	混合模型数据
<code>no-normal</code>	非经典后缀名的模型数据。	非典型模型数据
<code>tarmodel</code>	含有模型文件的 tar 包。	非典型模型数据
<code>dockermodel</code>	含有模型文件的 docker 镜像文件。	非典型模型数据

<sup>a</sup> <https://cn.ubuntu.com/download/server>

<sup>b</sup> <https://ftp.gnu.org/gnu/gcc>

<sup>c</sup> <https://www.kernel.org>

<sup>d</sup> [www.modelscope.cn/models/Qwen/Qwen3-32B](http://www.modelscope.cn/models/Qwen/Qwen3-32B)

<sup>e</sup> [www.modelscope.cn/models/AI-ModelScope/gemma-3-4b-it-qat-q4\\_0-gguf](http://www.modelscope.cn/models/AI-ModelScope/gemma-3-4b-it-qat-q4_0-gguf)

<sup>f</sup> [www.modelscope.cn/models/openai-mirror/gpt-oss-120b](http://www.modelscope.cn/models/openai-mirror/gpt-oss-120b)

### 5.1.4 测试方法及指标

本章中，由 5.1.3 节中的一种数据集或多种数据集混合构成不同工作负载，作为待存储数据，经由 5.1.2 节中的各种对照组系统进行去重，得到各对照组的数据缩减效果，具体指标包括：

1. 数据缩减率：衡量系统在处理不同类型数据时，成功去除冗余数据的比例，由以下公式计算，数据缩减率越高，效果越好：

$$\text{数据缩减率} = \frac{\text{原始数据量}}{\text{去重压缩后数据量}}$$

2. 吞吐量：评估系统在进行数据缩减操作时的处理速度，由以下公式计算，吞吐量越高，效果越好：

$$\text{吞吐量} = \frac{\text{处理的总数据量}}{\text{处理时间}}$$

## 5.2 基准实验

为模拟云存储任务的混合负载场景，本节实验将 5.1.3 节中所述的若干模型数据与非模型数据混合，构成具有代表性的工作负载。我们在 5.1.2 节中所述的各个实验对象上分别运行存储任务，直至去重、压缩与持久化流程完全结束。通过精确记录存储前后数据量的变化和任务总执行时间，我们计算出各方案的任务数据缩减率和吞吐量，旨在深入对比和分析在当前云存储环境下，FHRD 相较于其他现有工作的实际效果与性能差异。

### 5.2.1 数据缩减率

如图 5.1 所示，我们展示了各类混合负载下不同方案的数据缩减率对比结果。从图中可以观察到几个关键现象：

- ZIPNN、ZIPNN+ 和 BURST 的表现在所有测试中均不理想，其数据缩减率甚至在部分数据集中低于基准方案 BASE（变长分块 + 块级去重 + 通用压缩）。这一结果有力地揭示了变长分块 (Variable-Size Chunking) 在现代去重流水线中的核心地位。缺少变长分块，系统无法有效识别和消除由数据插入、删除或修改引起的偏移 (Shift) 问题，导致大量冗余数据块因边界变化而无法被识别，从而严重影响了去重效率。
- 以 ZIPNN++ 和 DIFFNN++ 为代表的方案，在 BASE 的基础上集成了针对模型数据的专用处理能力。这使得它们在处理包含模型数据的混合负载时，相较于

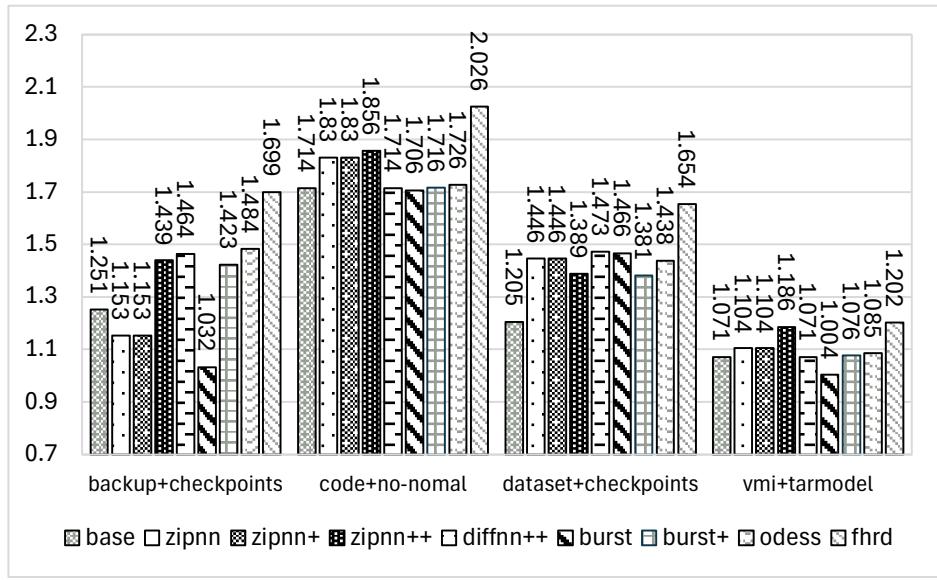


图 5.1 各类混合负载下数据缩减率对比

Figure 5.1 Comparison of deduplication rates under various mixed workloads

BASE 能取得约 6.5% 至 10.1% 的数据缩减率提升，证明了模型数据专用优化的价值。

- 以 BURST+ 和 ODESS 为代表的方案，它们在 BASE 的基础上引入了对块内冗余（Intra-Chunk Redundancy）的处理能力。BURST+ 通过优化数据块的边缘冗余，ODESS 则通过全局字节级增量压缩，分别实现了 5.2% 和 14.1% 的平均数据缩减率提升。这表明，随着数据块尺寸的增大和数据内容的日趋复杂，块内冗余已成为不可忽视的优化对象。
- 本文提出的 FHRD 在所有测试负载下均表现出最优的数据缩减率。FHRD 不仅集成了对模型数据和非模型数据的双重优化，还通过更精细的冗余识别策略，有效地应对了现代云存储中的复杂数据构成。其平均数据缩减率相较于 BASE 提高了 21.7%。这一结果初步验证了 FHRD 在处理多样化、大规模混合负载时的综合优势。

### 5.2.2 吞吐量

如图 5.2 所示为各类混合负载下吞吐量的对比结果。可以看到，所有引入了额外优化步骤的方案，由于增加了更为细粒度的处理流程，其整体吞吐量相较于简洁的 BASE 方案均有不同程度的下降。

其中，ODESS 作为增量压缩方法的理论上界代表，其性能表现最差。这是因为

它需要在全局范围内进行滚动哈希特征提取、相似性搜索以及字节级的差分计算，这些操作带来了巨大的计算和 I/O 开销。其吞吐量仅为 BASE 方案的 15.3%，下降幅度之大，说明此类高精度但高消耗的方案难以直接部署于要求高响应速度的在线云存储（Online Cloud Storage）环境中。

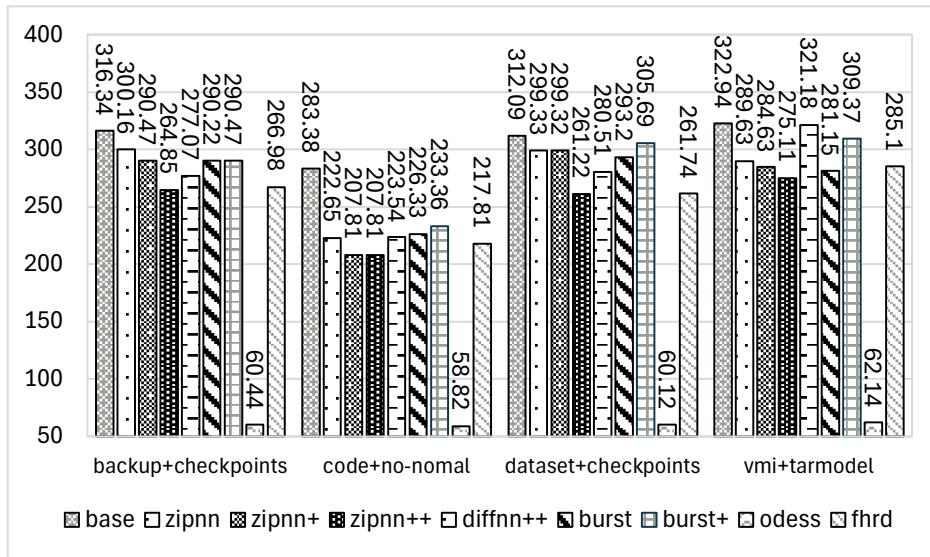


图 5.2 各类混合负载下吞吐量对比  
Figure 5.2 Comparison of throughput under various mixed workloads

相比之下，ZIPNN++，BURST+ 和 FHRD 等方案在吞吐量上虽有下降，但其性能损失仍在可接受范围内，展现了较好的实用性。具体来看，三者的吞吐量保持在 BASE 方案的 80% 以上。结合图 5.1 的数据缩减率结果，我们可以观察到一个宏观趋势：去重效果越精细，性能开销越大。这一方面源于更细粒度的冗余处理（如块内去重、模型编码）本身带来的额外计算开销；另一方面，这也与通用压缩算法（如 ZSTD）的特性相关——当输入数据的冗余度更高（即可压缩性更强）时，压缩算法需要消耗更多时间来查找和编码重复模式，从而导致处理速度下降。

综合 5.2 节的实验结果，我们可以得出以下结论：单纯基于定长分块的优化方案（如 ZIPNN，BURST）已无法满足现代复杂负载的需求，因此在后续的实验中，我们将重点与它们的变长分块优化版本进行比较。同时，尽管 ODESS 在去重效果上表现优异，但其过高的性能开销使其不具备在线应用的可行性，后续我们仅将其作为衡量去重效果的理想上界进行参考和讨论。

### 5.3 系统优化效果分析

基准实验初步验证了 FHRD 在混合负载下的整体有效性。为深入探究其内部关键技术的具体贡献，本节将通过一系列消融实验和对比分析，对第四章提出的各项优化——包括指数取整的双向子块定长分块方法、模型数据鉴别分离模块、基于熵值分析的字节分组压缩——进行逐一剖析。为此，我们设计了两种 FHRD 的消融版本：FHRD-1 和 FHRD-2（具体定义见表 5.2），旨在量化评估每个模块在不同场景下的实际效果与性能影响。

#### 5.3.1 指数取整的双向子块定长分块方法效果分析

本节将聚焦于 FHRD 针对大块化的优化——“指数取整的双向子块定长分块方法”，旨在阐明其在处理块内冗余方面的技术优势。

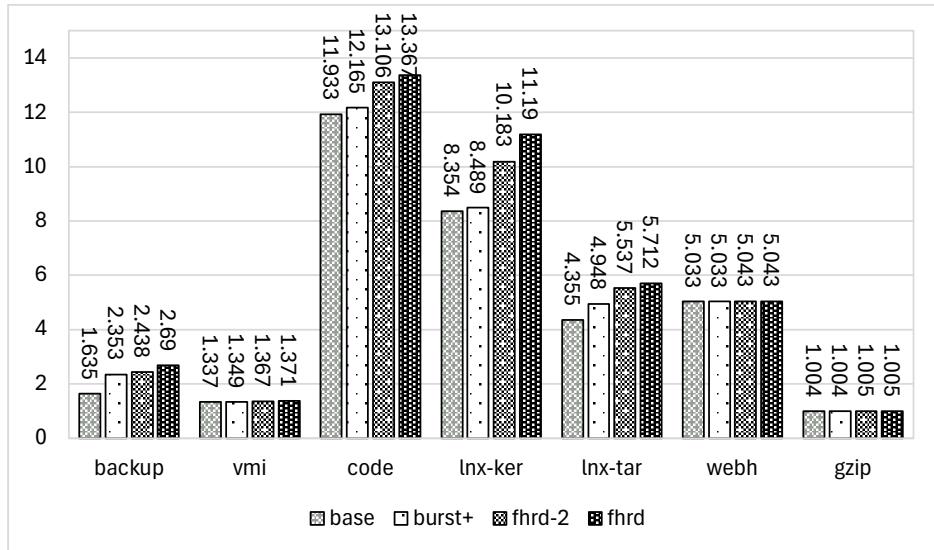


图 5.3 非模型数据数据缩减率对比  
Figure 5.3 Comparison of non-model data deduplication rates

图 5.3 展示了 FHRD、其消融版本 FHRD-2，以及 BURST+ 在多种非模型数据集上的数据缩减率对比。

- 整体趋势分析：从宏观上看，FHRD 的数据缩减率在所有数据集上均显著高于 FHRD-2，提升范围在 7.4% 至 10.3% 之间，平均高出 8.9 个百分点，这直接证明了“指数取整”这一核心思想的有效性。与简单的均分子块相比，指数取整能够更智能地划分数据，从而识别出更多隐藏的冗余。同时，FHRD 和 FHRD-2 的数据缩减率都一致优于 BURST+，平均分别高出 13.5% 和 4.6%，这是因为

BURST+ 的优化策略局限于数据块的首尾区域，未能充分挖掘块内部存在的冗余。最后，这三种细粒度优化方法的数据缩减率都高于基准的 BASE 组，再次凸显了在现代数据存储中，处理块内冗余对于提升整体去重效率的普遍重要性。

- 分数据集细化分析：

- 在 webh 数据集上，各方法的数据缩减率差异不大。这是因为 webh 主要由大量小文件（HTML, CSS, JavaScript）组成，这些文件本身尺寸小，块内几乎不存在连续的大段冗余，因此细粒度去重的优化空间天然受限。
- 在 gzip 数据集上，所有方法的数据缩减率都相对较低。这同样符合预期，因为 gzip 文件本身是经过压缩的，其内部冗余已被消除，后续任何去重方法都难以再获得显著的空间节省。
- 在 backup、code 等数据集中，不同方法之间的数据缩减率差距被显著放大。在这些场景下，FHRD 的优势尤为突出，表明其“指数取整的双向子块定长分块”方法在处理包含大量版本迭代、代码复制、数据移动的数据集中的块内冗余时，表现极为出色。

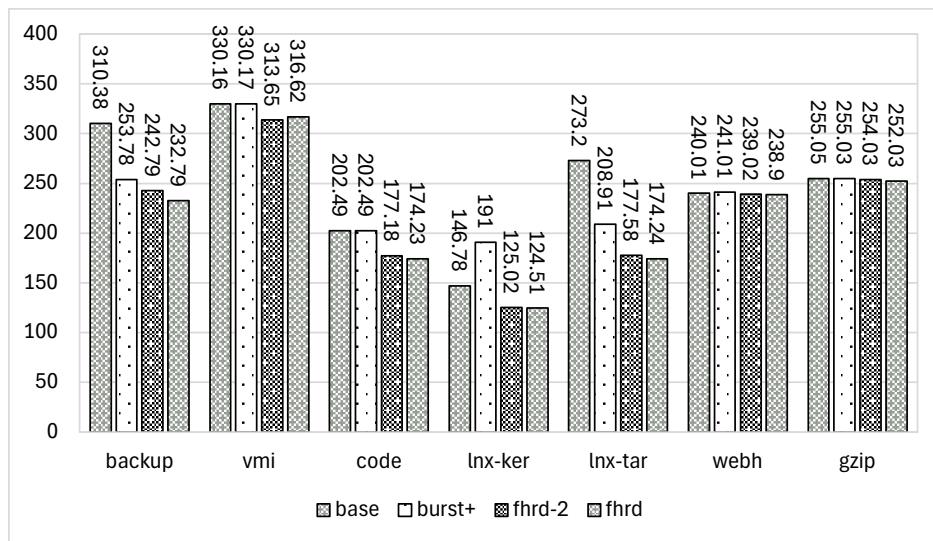


图 5.4 非模型数据吞吐量对比  
Figure 5.4 Comparison of non-model data throughput

性能方面，参考图 5.4 的吞吐量数据，FHRD 的处理速度略低于 FHRD-2。这一下降是符合逻辑的，因为它反映了 FHRD 识别并处理了更多的块内冗余，这些额外的操作（如更多的子块分块、哈希计算和索引查找）自然会带来一定的计算开销。然而，值得注意的是，二者的吞吐量差距非常小，这表明 FHRD 的精细化分块策略在计算上是高效的，其带来的性能影响在实际应用中完全可以接受。

为进一步剥离块级去重和通用压缩等因素的干扰，我们设计了更精细的实验，直接衡量不同方法在“相似块”检出和冗余去除上的核心能力。

如图 5.5 所示，从全局相似块检出率来看，三种方法呈现出清晰的层级关系：FHRD 在所有数据集上均实现了最高的检出率，FHRD-2 次之，而 BURST+ 则始终最低（BASE 组的相似块检出率为 0）。这一结果精准地反映了各方法策略的精细度差异：

- **FHRD**: 其“指数取整双向子块定长分块”策略，极大地提升了在不同数据块中划分出相同（或相似）子块的概率，从而能够捕捉到最多隐藏的块内相似性。
- **FHRD-2**: 作为消融版本，其均匀分块逻辑虽然也能发现部分冗余，但由于无法保证在相似但略有偏移的数据块中划分出大小一致的子块，其相似块识别能力相比 FHRD 显著降低。
- **BURST+**: 其优化焦点仅在数据块的边界，完全忽略了块内部的相似性，因此其检出率自然最低。

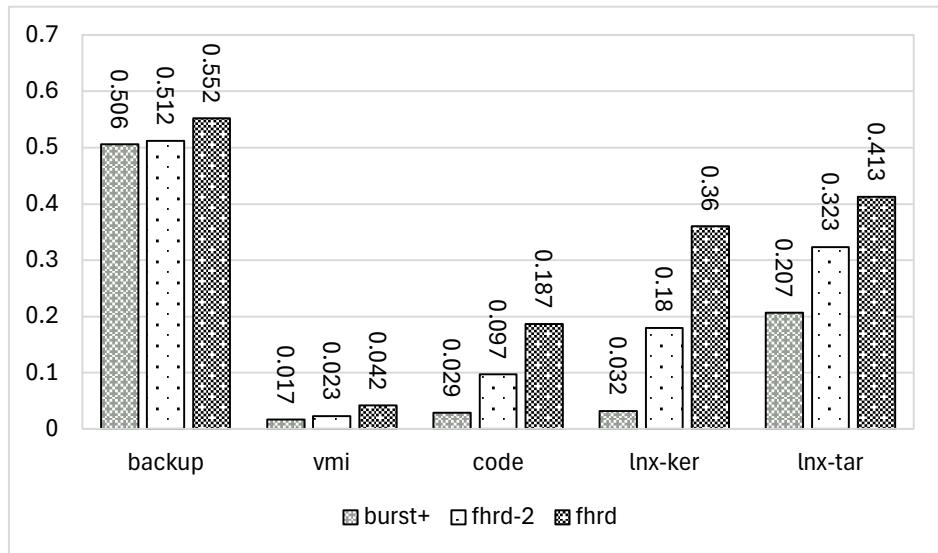


图 5.5 相似块检出率  
Figure 5.5 Comparison of similar block detection rates

不同数据集上的检出率差异进一步揭示了数据特性与算法的适配关系。在冗余度极高的 backup 数据集中，所有方法都表现不错，但 FHRD 仍以 0.552 的检出率表现最佳，展现了其在高冗余场景下的极限挖掘能力。而在冗余度极低的 vmi 数据集中，尽管所有方法的检出率都低于 0.05，FHRD 依然以 0.042 的优势领先，证明其在低冗余场景下仍具备一定的优化潜力。最能体现 FHRD 优势的是 code、lnx-ker 和

`lnx-tar` 这类数据集。以 `lnx-ker` 为例, FHRD 的检出率 (0.36) 是 FHRD-2 (0.18) 的 2 倍, 是 BURST+ (0.032) 的 11 倍之多。这是因为这类数据中包含了大量因代码复用、文件版本演进、模块化结构等产生的块内相似片段, 这与 FHRD 的细粒度分块策略形成了完美匹配, 使其强大的相似块识别能力得到了充分的发挥。

更高的相似块检出率是实现更优去重效果的直接前提。如图 5.6 所示, 在对已检出的相似块进行去重时, 我们观察到一个有趣的现象: FHRD 的“相似块冗余数据缩减率”反而低于其他方法。这并非说明 FHRD 的去重能力更弱, 恰恰相反, 这正是其检出能力更强的体现。因为 FHRD 检出了其他方法无法发现的、冗余度相对较低的“困难”相似块。这些低冗余块的加入, 虽然对总去重量有所贡献, 但却拉低了平均数据缩减率的统计数值。相比之下, BURST+ 和 FHRD-2 因为只能检出那些冗余度极高、最容易处理的相似块, 其计算出的平均数据缩减率自然更高。

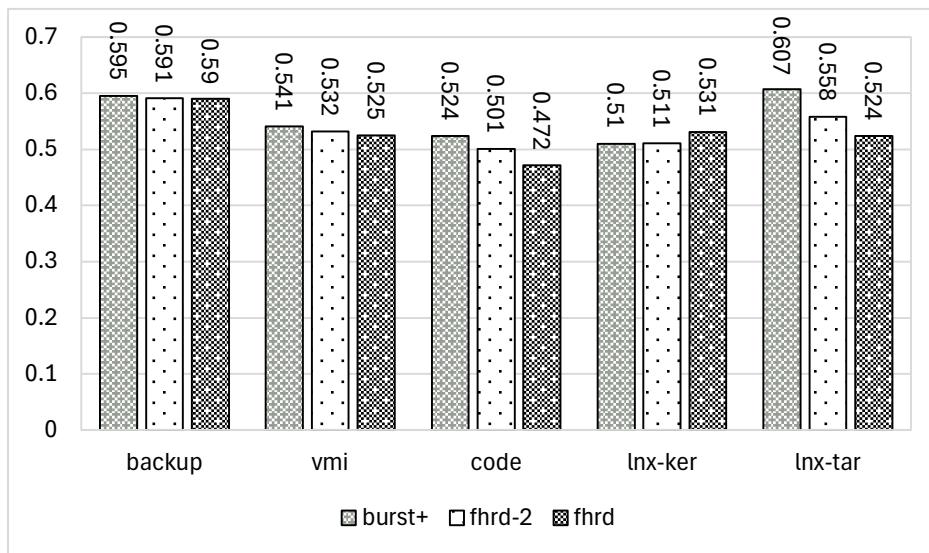


图 5.6 相似块冗余数据缩减率  
Figure 5.6 Comparison of similar block redundancy deduplication rates

综合本节所有实验结果, 我们可以得出结论: “指数取整的双向子块定长分块方法”通过其创新的分块策略, 能够以可接受的微小性能开销, 换取相似块检出率的巨大提升。这种能力使得 FHRD 能够发现并消除更多传统方法无法触及的块内冗余, 最终转化为显著的整体去重效果提升, 充分证明了该方法在处理非模型数据冗余方面的实用性与先进性。

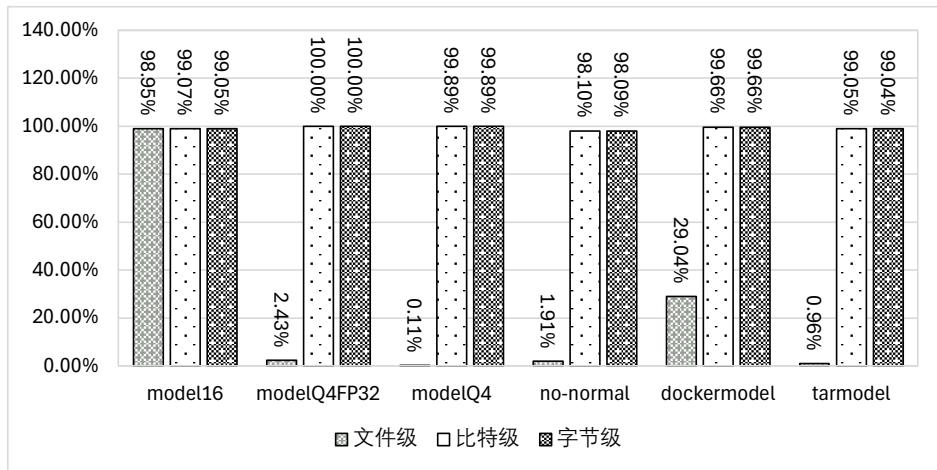
### 5.3.2 模型数据鉴别分离效率分析

本节首先对模型数据鉴别方法的表现进行评估。在 FHRD 的设计中，类型鉴别的核心目标是准确识别出那些具有规律字节熵、不适用于块去重，但适合采用专用编码处理的数据块。因此，我们将“模型数据块”操作性地定义为：能够通过 4.5 节所述的编码压缩方法实现 10% 以上空间节省的数据块。这个标准将作为我们评估类型鉴别准确性的基准。

实验中，我们对比了三种不同的鉴别方法在多个典型模型数据集上的表现：

1. **文件级 (File-level)**：基于文件后缀名（如 .pt, .safetensors）进行识别的传统方法。
2. **比特级 (Bit-level)**：基于数据块的比特粒度熵值分析进行判断。
3. **字节级 (Byte-level)**：本文提出的基于数据块的字节粒度熵值分析进行判断。

我们采用正确率、误报率和漏检率这三个关键指标来全面衡量它们的准确性。



**图 5.7 类型鉴别准确率对比**  
**Figure 5.7 Comparison of type identification accuracy**

如图 5.7 所示为三种鉴别方法的正确率对比结果，可见字节级和比特级的鉴别准确率在各种数据集上均能达到 98% 以上，且二者鉴别效果基本相同，差距在 0.02% 以内。而文件级在部分数据集（如 model16）上能够取得较高的正确率，在其他数据集上正确率明显低于前两者。

为深入探究正确率差异的根源，我们进一步分析了各方法的误报率和漏检率，如图 5.8 和图 5.9 所示。文件级方法的问题主要体现在两个方面：

- **高误报率**：在 modelQ4 这类以量化模型为主的数据集上，文件级方法仅凭后缀名就将许多已经过量化、熵值较高、不具备压缩潜力的数据块错误地判断为

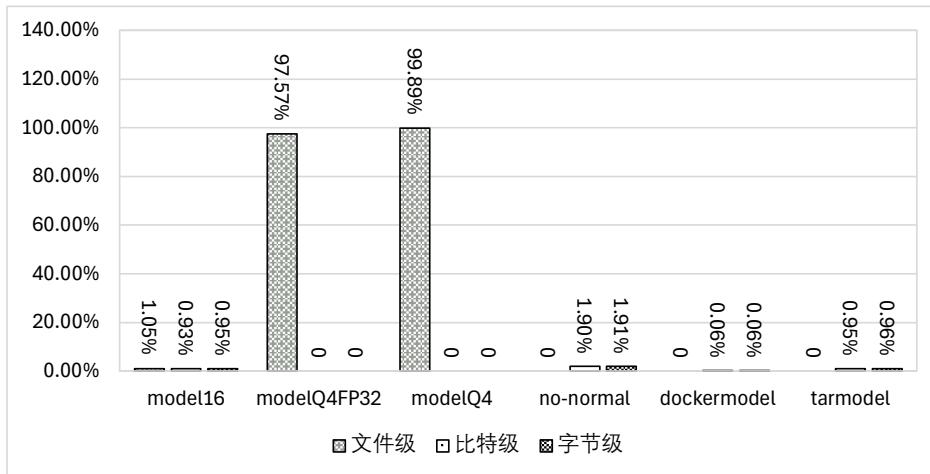


图 5.8 类型鉴别误报率对比

Figure 5.8 false positive rate

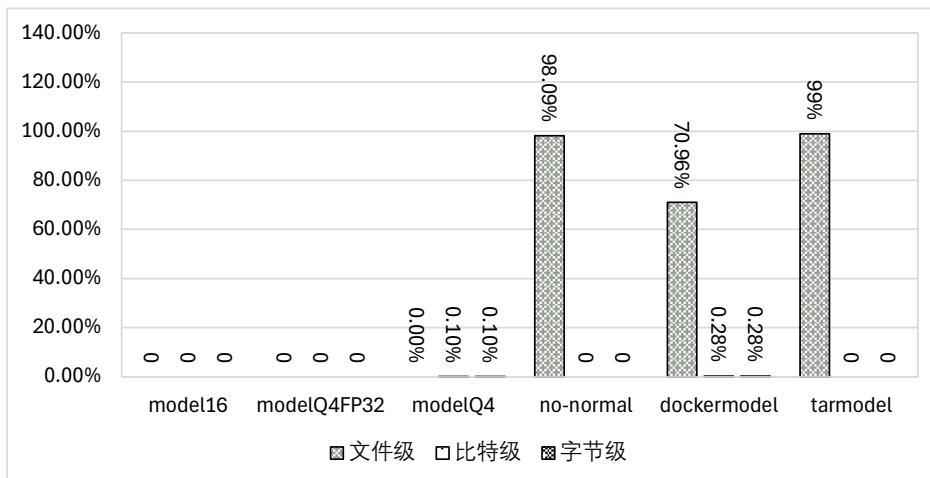


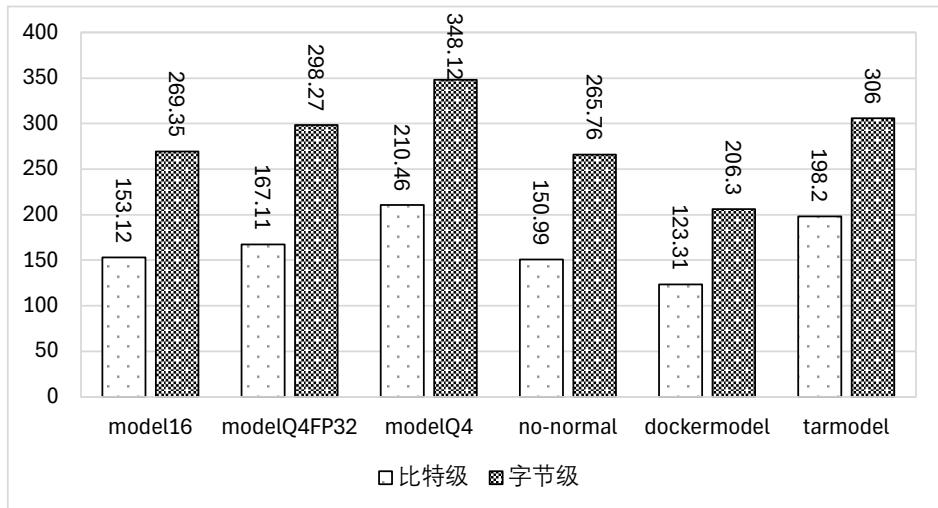
图 5.9 类型鉴别漏检率对比

Figure 5.9 false negative rate

“可压缩”的模型数据，导致了较高的误报率。

- **高漏检率：**在 dockermodel 这类非典型模型数据集上，由于模型数据被封装在容器镜像等复杂结构中，或使用了非常规的后缀名，文件级方法无法有效识别，导致大量模型数据被遗漏。

相比之下，字节级和比特级方法凭借对数据内容本身的分析，其误报率和漏检率始终保持着极低水平，展现了强大的场景适应性和鲁棒性。



**图 5.10 类型鉴别吞吐量对比**  
**Figure 5.10 Comparison of type identification throughput**

最后，我们评估了这几种方法的时间开销。文件级方法的开销与文件数量相关，与数据总量关系不大，其耗时极低，几乎可以忽略不计。因此，我们重点对比了字节级和比特级方法的吞吐量，如图 5.10 所示。实验结果清晰地表明，字节级方法的吞吐量显著高于比特级方法，平均提升幅度达到 69.6%（范围在 54.4% 至 78.5% 之间）。

综合来看，尽管文件级方法速度最快，但其准确性严重不足，无法适应复杂的云存储环境。而在两种基于熵值的方法中，字节级方法在保持与比特级方法几乎同等高准确率的同时，大幅提升了处理效率。这一结果证明了 4.4 节所述的字节级熵值分析方法在准确性和性能之间取得了更优的平衡，是 FHRD 系统中理想的数据类型鉴别方案。

### 5.3.3 基于熵值分析结论的字节分组压缩效果分析

在验证了类型鉴别的有效性后，本节将深入评估 FHRD 针对模型数据的优化——基于熵值分析结论的字节分组压缩方法。我们将通过对比 FHRD、其消融版本 FHRD-

1 以及专为模型数据设计的 ZIPNN++，来分析该方法在去重效果和性能开销上的综合表现。

如图 5.11 所示，我们比较了这几种方案在不同模型数据集上的数据缩减率。实验结果揭示了以下几点：

- **对于低冗余数据：**在 modelQ4 这类主要由量化模型构成的数据集上，所有方法的数据缩减率均不理想。这符合预期，因为量化过程本身就是一种信息压缩，其产生的数据冗余度极低，无论是通用压缩还是专用去重方法都难以进一步发掘空间节省潜力。
- **对于高冗余模型数据：**在 model16 这类纯净的、未量化的模型数据集中，FHRD 的去重效果与专门为此类数据优化的 ZIPNN++ 基本持平（差距小于 1%），证明了 FHRD 在处理其目标模型数据类型时，能够达到与专用工具相媲美的效率。
- **对于混合负载：**在 dockermodel 这类包含了模型数据和少量非模型数据的混合场景中，FHRD 的去重效果明显优于 ZIPNN++。这凸显了 FHRD 作为一个综合性方案的优势：它不仅能处理模型数据，还能兼顾负载中的其他数据类型，从而在真实的复杂负载下获得更高的整体数据缩减率。

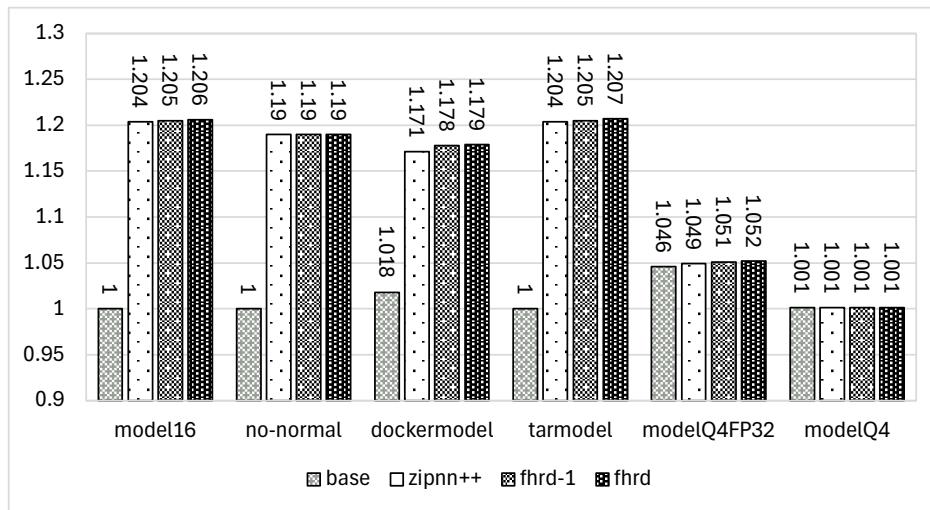
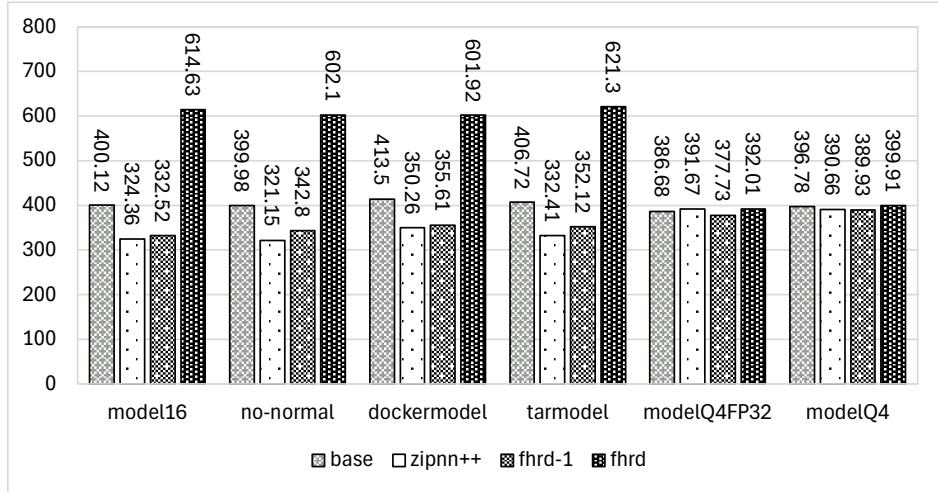


图 5.11 基于熵值分析结论的字节分组压缩效果对比

**Figure 5.11 Comparison of byte-wise grouped compression effects based on entropy analysis conclusions**

此外，通过对比 FHRD 与其消融版本 FHRD-1（该版本对所有字节组进行压缩，而非选择性压缩），我们发现 FHRD 的数据缩减率并未下降，反而在多个数据集上略有提升。这看似有悖常理，但其背后原因是：FHRD-1 在尝试压缩那些本身不含冗余

的字节组时，不仅无法减小数据体积，反而会因为增加了压缩元数据而引入额外的存储开销。**FHRD** 通过基于熵值分析结论的选择性压缩策略，精确地只对具有冗余的字节组（即熵值较低的组）进行压缩，从而避免了这种“无效压缩”带来的负面影响。



**图 5.12 压缩线程吞吐量对比**  
**Figure 5.12 Comparison of compression thread throughput**

为了进一步验证该方法在节省计算资源方面的优势，我们测量了压缩线程的吞吐量，如图 5.12 所示。实验数据显示，ZIPNN++ 和 FHRD-1 的压缩吞吐量均低于基准的 BASE 方案。这是因为它们的编码优化使得输入到 ZSTD 压缩器的数据更具可压缩性，而 ZSTD 在处理高度可压缩数据时需要消耗更多计算资源来查找匹配项，导致速度变慢。

相比之下，FHRD 的表现则完全不同。通过其字节分组压缩策略，FHRD 在处理以 16 位浮点数为主的模型数据时，理论上可以将送入压缩算法的数据量减少一半（只压缩高位字节组）。实验结果也证实了这一点：FHRD 的压缩吞吐量相较于 FHRD-1 提升了约 80%。吞吐量提升未达到理论上的 100%，同样是因为被压缩的冗余字节组（高位字节）具有高度可压缩性，其处理速度慢于非冗余字节组，从而对整体吞吐量产生了一定影响。

综上所述，基于熵值分析的字节分组压缩方法，不仅在去重效果上不输于甚至略优于全量压缩，更重要的是，它通过避免无效计算，显著提升了压缩阶段的吞吐量，有效降低了系统的整体计算开销。这充分证明了 4.5 节所述方法在系统性能方面的优越性。

## 5.4 负载特点影响实验

前述实验充分证明了 FHRD 各项优化技术的独立有效性。本节将回归到第 3.1 节提出的云存储负载演进趋势，通过参数化实验，系统性地探讨混合负载比例和数据块大小这两个关键因素对 FHRD 及其他方案去重效果的影响。实验负载选用 dataset 和 checkpoints，因为这两个人工生成的数据集能让我们精确地控制模型数据的占比和版本间的差异。

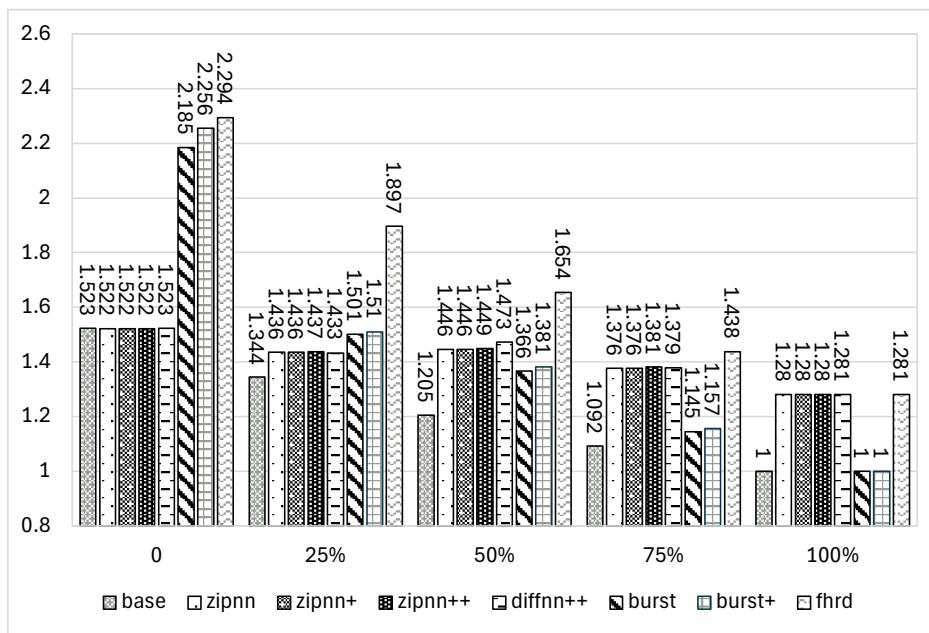


图 5.13 数据缩减率随模型数据比例变化情况

Figure 5.13 Comparison of deduplication rates with varying model data proportions

首先，我们来考察云存储负载的第一个变化趋势：工作负载中模型数据的比例日益增加。如图 5.13 所示，我们绘制了各系统数据缩减率随模型数据比例变化的曲线。

- 对于专攻模型压缩的 ZIPNN++ 和 DIFFNN++，其去重效果与模型数据占比呈现明显的正相关。当模型数据占比为 0 时，它们的数据缩减率退化至与 BASE 方案相同，因为其优化模块没有被激活。
- 相反，对于专攻块内冗余的 BURST+，其优势主要体现在非模型数据上。随着模型数据占比的增加，其去重效果逐渐下降，当模型数据占比达到 100% 时，其数据缩减率同样退化至 BASE 水平。
- FHRD 的表现则截然不同。得益于其内置的数据类型鉴别和混合优化策略，FHRD 能够动态地为不同类型的数据施加最合适的去重方法。它不仅结合了模型数据优化和块内冗余优化的优点，还通过更精细的算法进一步放大了这些优势。因

此，在从 0% 到 100% 的整个模型数据占比范围内，FHRD 始终保持着最高的数据缩减率，展现了其在应对任意混合比例负载时的强大适应性和鲁棒性。

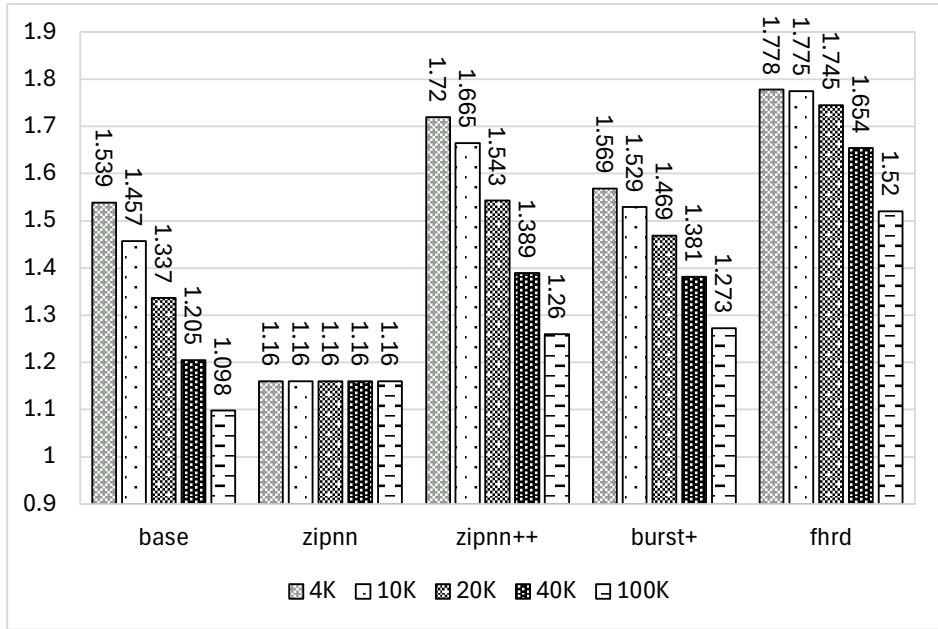


图 5.14 数据缩减率随块大小变化情况  
Figure 5.14 Comparison of deduplication rates with varying block sizes

接下来，我们探讨云存储负载的第二个变化趋势：为了适应海量数据的存储和管理，数据块的平均大小越来越大。如图 5.14 所示，我们展示了各系统数据缩减率随块大小变化的趋势。

- BASE 组的数据缩减率随着块大小的增加而显著下降。这揭示了传统变长分块去重的一个核心缺陷：块越大，因数据内容的小幅改动而导致整个块的哈希值发生变化的概率就越大，从而使得识别出完全相同的块变得更加困难，去重效率随之降低。
- ZIPNN 由于完全不依赖块级去重，其数据缩减率基本不受块大小变化的影响。而其增强版 ZIPNN++，虽然在 BASE 的基础上增加了对唯一块的编码处理，提升了整体去重效果，但它并不能缓解大块尺寸对块级去重效率的负面冲击。其数据缩减率曲线的下降趋势与 BASE 类似，只是因为有 ZIPNN 的基础保障，其下降的下界更高。
- BURST+ 通过处理块的首尾冗余，在一定程度上缓解了块大小增加带来的数据缩减率下降问题。尽管它缺乏对模型数据的处理能力，在较小块大小下其数据缩减率低于 ZIPNN++，但随着块大小的增加，其数据缩减率下降得更慢，在块

大小达到 100KB 时成功反超了 ZIPNN++。

- **FHRD** 在此项测试中再次展现了其设计的优越性。通过其细粒度块内冗余去重机制, **FHRD** 能够有效地从大尺寸数据块中挖掘出局部冗余, 极大地缓解了块大小增加对数据缩减率的负面影响。同时, 它还保留了对模型数据的强大处理能力。因此, 在所有测试的块大小范围内, **FHRD** 均表现出最高的数据缩减率, 相较于 **BASE** 组实现了 15.5% (4KB) 至 38.4% (128KB) 的显著提升。

综合本节实验, **FHRD** 不仅在处理不同比例的混合负载时表现出色, 而且在应对日益增大的数据块尺寸趋势时也展示了强大的鲁棒性。这充分说明了本文提出的方法能够有效解决 3.1 节中所述的现代云存储负载演进带来的挑战。

## 5.5 本章小结

本章通过实验, 对本文提出的 **FHRD** 进行了系统性的评估。

- **基准性能评估:** 初步的基准实验结果清晰地表明, 在模拟真实云存储的各类混合负载下, **FHRD** 相较于现有的主流方案 (**BASE**)、模型专用方案 (**ZIPNN++**) 以及细粒度去重方案 (**BURST+**), 均表现出最优的数据缩减率, 初步验证了其在应对现代复杂数据负载时的综合优势。
- **关键技术消融分析:** 通过细致的消融实验, 我们逐一剖析了 **FHRD** 内部各项关键技术的实际贡献:
  - **指数取整双向子块分块:** 此方法能够以较小的性能代价, 换取对块内冗余识别能力的提升, 是 **FHRD** 实现高数据缩减率的关键。
  - **模型数据分离模块:** 基于字节熵的类型鉴别方法, 在保持极高准确率的同时, 显著提升了处理效率, 在性能和精度之间取得了理想的平衡。
  - **字节分组压缩:** 该方法通过选择性地压缩, 不仅保证了对模型数据的高效去重, 还通过避免无效计算, 大幅提升了压缩吞吐量, 降低了系统开销。
- **负载影响分析:** 最后, 通过对混合负载比例和块大小这两个关键参数进行对比实验, 我们发现 **FHRD** 不仅在各种数据混合比例下都能保持领先, 而且能够有效抵抗因块尺寸增大而导致的数据缩减效率衰减问题。

综上所述, 本章的实验结果验证了 **FHRD** 整体设计的先进性, 以及其各项核心优化技术的优越性。实验数据表明, 本文提出的方法能够有效应对当前云存储环境中数据大块化和混合化的挑战, 实现了在复杂负载下的高效去重。

## 第6章 总结与展望

随着人工智能技术的飞速发展，现代云存储系统正面临着由传统数据与模型数据共同构成的混合工作负载所带来的严峻挑战。本论文深入研究了传统数据缩减技术在这一新背景下的局限性，其核心问题在于传统基于块的粗粒度去重机制，既无法识别模型文件中普遍存在的“数值相似性冗余”，也难以处理大尺寸数据块内部的“局部连续冗余”。同时，简单地堆砌针对特定数据优化的算法，会在混合数据流中引发相互干扰，导致“性能退化”。

为系统性地解决上述问题，本文提出并实现了一种面向混合负载的细粒度冗余识别数据缩减系统——FHRD (Fine-grained Hybrid Redundancy Deduplication)。主要研究成果与贡献可归纳为以下几点：

1. 本研究系统性地分析了传统数据缩减技术面对现代混合负载的局限性，首次明确定义了两种对存储效率影响显著的关键冗余——模型数据中的“数值冗余”与大块数据中的“局部连续冗余”，并深入阐述了它们的特点与分布规律，为后续的针对性优化提供了坚实的理论基础。
2. 为解决上述问题，本文在传统数据缩减流水线的基础上实现了细粒度子块去重，通过对数据块内部连续冗余的识别与消除，显著提升了传统数据的缩减效果。同时，为了应对负载中混入的模型数据，本文创新性地引入了块级数据鉴别机制，通过对分组比特熵值的规律性分析，鉴别出模型数据块，将其分离出去重流程，并针对性地应用编码算法，实现了对模型数据的高效压缩。该机制有效避免了不同冗余处理模块间的负面干扰，提升了整体系统的适应性与性能。
3. 本文设计并实现了一系列系统优化策略，如指数取整的双向子块分块方法提升块内冗余的识别率；字节粒度的分组鉴别策略提高模型数据分离的效率；基于鉴别结论的模型编码压缩方法减少压缩开销等。
4. 本研究基于开源去重系统 Destor，实现了 FHRD 系统原型，并设计了包括数据管理、多级缓存、流水线并行在内的一系列性能优化策略。在涵盖多种数据类型、块大小及混合比例的综合实验评估中，结果表明，FHRD 相比传统去重系统在混合负载场景下取得了最高 38.4% 的数据缩减率提升，验证了其在处理云存储负载时的有效性与先进性。

尽管本研究在面向混合负载的细粒度数据缩减方面取得了阶段性成果，但该领

域仍充满广阔的探索空间。结合当前研究的局限性与存储技术的发展趋势，未来的研究可从以下几个方向展开：

1. **扩展对更广泛冗余类型的支持。** FHRD 系统目前主要关注数值相似性冗余和局部连续冗余。然而，现实世界的数据冗余模式可能不止于此。例如，不同版本的压缩文件之间、结构化数据（如 CSV、JSON）的语义重复、乃至跨模态数据（如同时包含文本描述的图片）中都蕴含着独特的冗余模式。未来的工作可以致力于扩展 FHRD 的框架，集成更多的冗余识别探针与处理模块，使其成为一个能够全面感知并处理多种冗余类型的综合性数据缩减平台。
2. **探索硬件加速与软硬件协同设计。** 细粒度冗余识别不可避免地引入了额外的计算开销。为了进一步降低其对系统性能的影响，可以探索将计算密集型任务（如熵值计算、滚动哈希、特征匹配等）卸载到专用硬件（如 FPGA）上进行加速。通过软硬件协同设计，将数据处理逻辑与硬件特性紧密结合，有望在维持甚至超越当前去重效果的同时，实现与传统粗粒度去重相媲美的处理吞吐量。
3. **构建自适应的动态策略调整机制。** 当前系统的优化策略（如采样率、子块大小、压缩算法选择等）主要依赖于静态配置。未来的系统可以引入动态反馈与自适应调整机制。通过实时监控系统自身的性能指标（如数据缩减率、吞吐量、CPU 负载、I/O 延迟等）以及输入数据流的特征变化，系统可以动态调整其内部参数，甚至在不同处理模块间进行智能切换，以节省计算资源，实现成本效益的最优化。

总之，随着数据形态的不断演进，数据缩减技术应当从通用方法，迈向精细化、细粒度、自适应的新阶段。本论文的工作为此方向提供了一次尝试，我们相信，未来的云存储系统将能够以更高的效率、更低的成本承载云计算的需求。

## 参考文献

- [1] KHAN A Q, MATSKIN M, PRODAN R, et al. Cloud storage cost: a taxonomy and survey[J]. World Wide Web, 2024, 27(4): 36.
- [2] DUBOIS L, AMALDAS M, SHEPPARD E. Key considerations as deduplication evolves into primary storage[J]. White Paper, 2011, 223310.
- [3] 奎晓燕, 张敏, 肖伶, 等. 数据去重与缩减技术的系统分类与性能分析[J]. 浙江大学学报(工学版), 1-16.
- [4] XIA W, JIANG H, FENG D, et al. A comprehensive study of the past, present, and future of data deduplication[J]. Proceedings of the IEEE, 2016, 104(9): 1681-1710.
- [5] 胡浩. 面向云平台的大数据存储空间优化系统设计与实现[D]. 哈尔滨工业大学, 2021.
- [6] PENG B, WU Y C, ZHANG Z, et al. BURST: A Chunk-Based Data Deduplication System with Burst-Encoded Fingerprint Matching[J].
- [7] WANG Z, LAN T, SU Z, et al. ZipLLM: Efficient LLM Storage via Model-Aware Synergistic Data Deduplication and Compression[C]//. 2026.
- [8] HERSHCOVITCH M, WOOD A, CHOSHEN L, et al. Zipnn: Lossless compression for ai models [C]// 2025 IEEE 18th International Conference on Cloud Computing (CLOUD). 2025: 186-198.
- [9] 伍高飞, 袁紫依, 孙思贤, 等. 云数据安全去重技术研究综述[J/OL]. 密码学报, 2023, 10(06): 1099-1117. DOI: 10.13868/j.cnki.jcr.000663.
- [10] QUINLAN S, DORWARD S. Venti: A new approach to archival data storage[C]// Conference on file and storage technologies (FAST 02). 2002.
- [11] MUTHITACHAROEN A, CHEN B, MAZIERES D. A low-bandwidth network file system[C]// Proceedings of the eighteenth ACM symposium on Operating systems principles. 2001: 174-187.
- [12] BRODER A Z. On the resemblance and containment of documents[C]// Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171). 1997: 21-29.
- [13] ZHANG Y, XIA W, FENG D, et al. Finesse:{Fine-Grained} feature locality based fast resemblance detection for {Post-Deduplication} delta compression[C]// 17th USENIX Conference on File and Storage Technologies (FAST 19). 2019: 121-128.
- [14] XIA W, PU L, ZOU X, et al. The design of fast and lightweight resemblance detection for efficient post-deduplication delta compression[J]. ACM Transactions on Storage, 2023, 19(3): 1-30.
- [15] MACDONALD J P. File system support for delta compression. Master's thesis[J]. University of California at Berkeley, 2000.
- [16] TAN H, XIA W, ZOU X, et al. The design of fast delta encoding for delta compression based storage systems[J]. ACM Transactions on Storage, 2024, 20(4): 1-30.

- [17] GAILLY J L, ADLER M. Gnu gzip[J]. GNU Operating System, 1992: 8-18.
- [18] NELSON M R. LZW data compression[J]. Dr. Dobb's Journal, 1989, 14(10): 29-36.
- [19] DUDA J, TAHBOUB K, GADGIL N J, et al. The use of asymmetric numeral systems as an accurate replacement for Huffman coding[C] // 2015 Picture Coding Symposium (PCS). 2015: 65-69.
- [20] COLLET Y, KUCHERAWY M. Zstandard compression and the application/zstd media type[R]. 2018.
- [21] GHOLAMI A, KIM S, DONG Z, et al. A survey of quantization methods for efficient neural network inference[G] // Low-power computer vision. Chapman, 2022: 291-326.
- [22] MA X, QIN M, SUN F, et al. Effective model sparsification by scheduled grow-and-prune methods [J]. arXiv preprint arXiv:2106.09857, 2021.
- [23] NING W, WANG J, QI Q, et al. Fm-delta: Lossless compression for storing massive fine-tuned foundation models[J]. Advances in Neural Information Processing Systems, 2024, 37: 66796-66825.
- [24] FU M, FENG D, HUA Y, et al. Design tradeoffs for data deduplication performance in backup workloads[C] // 13th USENIX Conference on File and Storage Technologies (FAST 15). 2015: 331-344.
- [25] 敖莉, 舒继武, 李明强. 重复数据删除技术[J]. 软件学报, 2010, 21(05): 916-929.
- [26] XIA W, ZHOU Y, JIANG H, et al. {FastCDC}: A fast and efficient {Content-Defined} chunking approach for data deduplication[C] // 2016 USENIX Annual Technical Conference (USENIX ATC 16). 2016: 101-114.
- [27] RABIN M O. Fingerprinting by random polynomials[J]. Technical report, 1981.
- [28] XIA W, ZOU X, JIANG H, et al. The design of fast content-defined chunking for data deduplication based storage systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(9): 2017-2031.
- [29] SRINIVASAN K, BISSON T, GOODSON G R, et al. iDedup: Latency-aware, inline data deduplication for primary storage.[C] // Fast: vol. 12. 2012: 1-14.
- [30] XIA W, JIANG H, FENG D, et al. {SiLo}: A {Similarity-Locality} based {Near-Exact} deduplication scheme with low {RAM} overhead and high throughput[C] // 2011 USENIX Annual Technical Conference (USENIX ATC 11). 2011.
- [31] DRAGO I, MELLIA M, MUNAFÒ M, et al. Inside dropbox: understanding personal cloud storage services[C] // Proceedings of the 2012 internet measurement conference. 2012: 481-494.
- [32] RACHMAWATI D, TARIGAN J, GINTING A. A comparative study of Message Digest 5 (MD5) and SHA256 algorithm[C] // Journal of Physics: Conference Series: vol. 978. 2018: 012116.
- [33] ZIV J, LEMPEL A. A universal algorithm for sequential data compression[J]. IEEE Transactions on information theory, 2003, 23(3): 337-343.
- [34] WALLACE G, DOUGLIS F, QIAN H, et al. Characteristics of backup workloads in production

- systems.[C]//FAST: vol. 12. 2012: 4-4.
- [35] MEYER D T, BOLOSKY W J. A study of practical deduplication[J]. ACM Transactions on Storage (ToS), 2012, 7(4): 1-20.
- [36] 夏文. 数据备份系统中冗余数据的高性能消除技术研究[D]. 华中科技大学, 2014.
- [37] 邹翔宇. 备份存储系统高性能细粒度数据去重技术研究[D]. 哈尔滨工业大学, 2023.
- [38] 浦理峰. 面向备份去重系统的相似检测和数据恢复性能优化研究[D]. 哈尔滨工业大学, 2022.
- [39] XIA W, JIANG H, FENG D, et al. Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets[C]//2014 Data Compression Conference. 2014: 203-212.
- [40] SHILANE P, HUANG M, WALLACE G, et al. Wan-optimized replication of backup datasets using stream-informed delta compression[J]. ACM Transactions on Storage (ToS), 2012, 8(4): 1-26.
- [41] 王国华. 高效重复数据删除技术研究[D]. 华南理工大学, 2014.
- [42] BLOOM B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.
- [43] ZHU B, LI K, PATTERSON R H. Avoiding the disk bottleneck in the data domain deduplication file system.[C]//Fast: vol. 8. 2008: 1-14.
- [44] SHANNON C E. A mathematical theory of communication[J]. ACM SIGMOBILE mobile computing and communications review, 2001, 5(1): 3-55.



## 致 谢

研究生生涯即将结束，回首过去的几年，感慨良多。在此，谨向给予我帮助和支持的老师、同学和家人表示衷心的感谢！

首先要感谢计算机学院的姚建国老师和彭博老师，从研究方向的探索与确立，到课题的设计与实施，再到论文的撰写与修改，二位老师给予了我丰富的指导和帮助。感谢他们在学术上对我的严格要求和耐心指导，使我不断进步和成长。同时，也感谢学院的各位老师和工作人员，感谢你们在我研究生期间提供的支持。

本论文的完成也离不开同学们的帮助，感谢实验室的王文杰、吴晨鹏、刘尧等同门，帮助我细化选题，并指出实验设计中的不足以及论文写作中的错误以便于修改。感谢你们在学术和生活中给予的支持与鼓励。

衷心感谢我的家人，感谢你们一直以来的理解和支持，是你们的鼓励让我能够坚持不懈地追求梦想。你们永远是我最温暖的港湾，最强大的后盾。谨在此表达我对父母最衷心的感谢与祝福。

最后，感谢参与论文评审与答辩的诸位老师，感谢你们拨冗对本文进行审阅，为本文提出宝贵建议。



## 学术论文和科研成果目录

### 专利

- [1] 姚建国, 刘茁, 彭博, 管海兵, “面向混合负载的存储数据缩减方法及系统”, 专利申请号: CN202511664109.X.