FHRD: A Fine-grained Hybrid Redundancy Data Reduction System for Mixed Workloads

LiuZhuo
Shanghai Jiao Tong University

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 dataFHRD (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. 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.

Experimental results 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. 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%.

1 Introduction

The maturity and ubiquity of cloud computing technology have significantly driven the development of the modern information technology industry. As one of its foundation stones, cloud storage systems are tasked with providing reliable and scalable data storage capabilities for massive global users. To effectively cope with the cost pressure brought by large-scale data [**?**], **Data Reduction** technology has become an indispensable core technology in cloud storage systems [**?**].

However, in recent years, the efficiency of cloud storage data reduction has encountered a bottleneck, which is closely related to the changes in the workloads it carries. First, files in cloud storage workloads are becoming larger. To pursue high I/O throughput, modern storage systems tend to adopt larger data block sizes, which leads to a significant decline in the effectiveness of traditional deduplication technologies [**?**,**?**]. As the data block size increases, the probability of finding two completely identical data blocks decreases dramatically. Although these large blocks often contain a large number of local duplicate fragments, it is difficult for traditional coarse-grained deduplication based on the entire block to detect such "local continuous redundancy" within the block.

Second, with the breakthrough progress of artificial intelligence technology, more and more model data are mixed into cloud storage workloads. Model files are composed of hundreds of millions of independent floating-point tensors, and the data content presents a high degree of randomness and uniqueness. When the system splits different models into data blocks, almost every block will have a unique fingerprint, which makes the deduplication mechanism based on global fingerprint matching completely invalid. At the same time, general compression algorithms also struggle to capture the "numerical redundancy" inside floating-point numbers.

In summary, these two workload trends jointly shape a mixed workload where "model data and large block data coexist". Based on this, this study proposes and builds a fine-grained redundancy identification data reduction system named FHRD (Fine-grained Hybrid Redundancy Deduplication). This system first adds the ability to identify intra-block continuous redundancy on the basis of the traditional deduplication pipeline. Second, the framework incorporates a lightweight model data identification and separation module. Finally, for the separated model data, the framework adopts a dedicated grouped encoding method to compress its numerical redundancy.

2 Background and Motivation

Problem Analysis. Traditional data reduction systems, which rely on coarse-grained block-level deduplication and general-purpose compression, are becoming increasingly inefficient for modern cloud storage workloads. This inefficiency stems primarily from two emerging trends: increasing block sizes and the proliferation of AI model data.

First, as storage systems adopt larger block sizes to maximize throughput, the effectiveness of traditional deduplication declines. Large blocks often contain significant local repetitive content, but even a single byte difference prevents the entire block from determining a match. As shown in Figure **??**, our experiments with Linux kernel source tarballs demonstrate that the data reduction rate of Destor+Zstd [**?**] drops noticeably as the average block size increases.

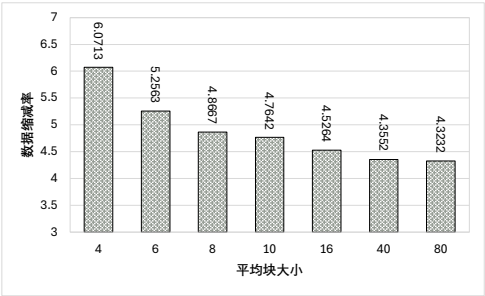

Figure 1: Deduplication rate variation with average data block size

Second, the influx of model data (e.g., checkpoints) further exacerbates this issue. Model files

consist largely of floating-point tensors, rendering fingerprint-based matching ineffective. As Figure **??** illustrates, in a mixed workload utilizing Transformer models, the data reduction rate plummets as the proportion of model data rises, approaching 1 (no reduction) when the workload is entirely model data.

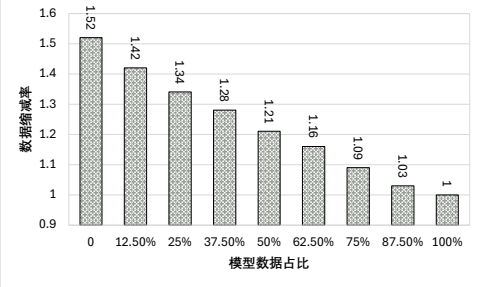

Figure 2: Deduplication rate variation with model data proportion

Analysis of Fine-grained Redundancy: A deeper analysis reveals that while traditional methods fail, significant *fine-grained redundancy* remains unexploited in these workloads:

- **Intra-block local redundancy**: In large non-model data blocks, although the global fingerprints differ, substantial redundancy still exists within similar blocks.
- **Model numerical redundancy**: In model data, floating-point numbers (BF16/FP32) exhibit high similarity in their exponent bits, while mantissa bits vary. Entropy analysis of Transformer models confirms that the exponent parts have much lower entropy than the mantissas, indicating a potential for compression if processed correctly.

Design Goals: To effectively address the challenges posed by mixed workloads with large blocks and model data, our system design focuses on the following key objectives:

1. **Fine-grained Deduplication**: The system must identify and remove local repetitions within large data blocks, effectively handling the "large block" trend.
2. **Model Data Identification**: The system needs a low-overhead method to distinguish model data from traditional data, enabling specialized processing paths.
3. **Model-Specific Compression**: For identified model data, the system should employ encoding strategies that transform numerical redundancy into a compressible format.

3 System Design and Optimization

3.1 System Overview

The FHRD system proposed in this paper, shown in Figure **??**, integrates multiple new modules based on the traditional block-level deduplication flow.

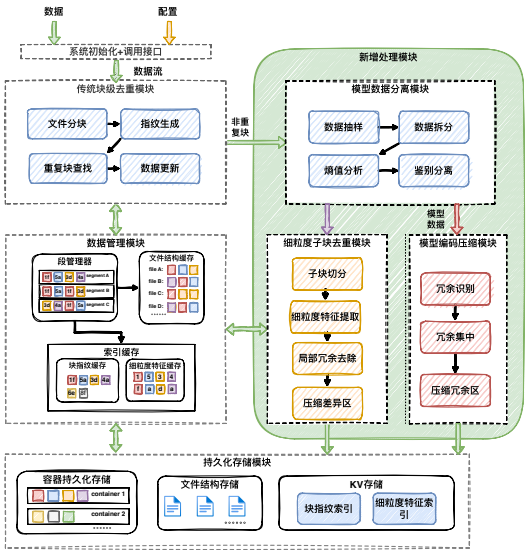

Figure 3: Architecture of Fine-grained Hybrid Redundancy Deduplication System

The workflow begins with the **Data Type Separation Module**, which analyzes incoming data blocks to distinguish between model data and non-model data. Non-model data is directed to the **Fine-**

grained Sub-block Deduplication Module, while model data is sent to the **Model Encoding Compression Module**.

3.2 Fine-grained Sub-block Deduplication for Non-model Data

For data blocks identified as non-model data, the system routes them to the Non-model Data Processing Module (Figure **??**). This module is designed to identify and remove fine-grained sub-block redundancy that traditional deduplication misses. Although large blocks often contain local repetitive content, effectively utilizing this redundancy faces several challenges in partitioning strategy.

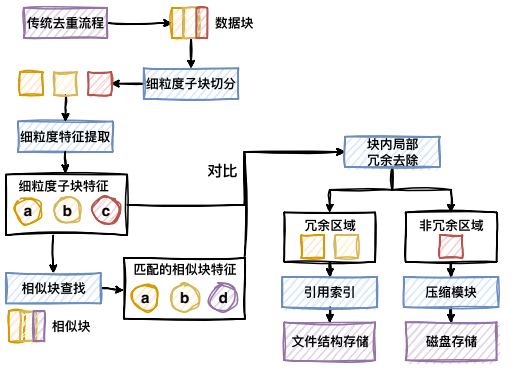

Figure 4: Non-model Data Processing Module Architecture

Challenge 1: Boundary Offset. Traditional fixed-length chunking suffers from the boundary offset problem. As shown in Figure **??**, traditional data (like text or code) often has small modifications. A simple insertion or deletion shifts the boundaries of all subsequent blocks (yellow parts), preventing deduplication even if the content remains largely consistent.

Content-Defined Chunking (CDC) (variable-length chunking) [**?**, **?**] solves this problem by determining boundaries based on content hash. As Figure **??** illustrates, boundaries shift adaptively, allowing the majority of the content to be correctly identified as duplicate blocks.

Figure 5: Boundary offset problem in fixed-length chunking

Figure 6: Variable-length chunking solves the boundary offset problem

Challenge 2: Inconsistent Sub-block Division. CDC results in variable block sizes, which complicates sub-block partitioning using fixed number of sub-blocks. If we simply divide variable-length blocks into a fixed number of sub-blocks (e.g., 3 parts), the resulting sub-blocks will have different sizes, as shown in Figure **??**. This misalignment means that even if the red parts are redundant, they cannot be matched because their boundaries do not align.

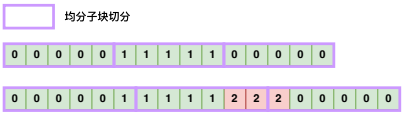

Figure 7: Sub-block division into 3 fixed parts

To address these issues, we propose the **Exponential Rounding Bidirectional Sub-block Fixed-length Segmentation** method. First, the system calculates a sub-block length that is the power of 2 closest to 1/10 of the variable-length data block size. This ensures sub-blocks are appropriately scaled to the parent block and maintain a consistent fixed size across similar blocks. Then, the system performs fixed-length chunking from both the start and end of the block towards the center. As

shown in Figure **??**, this bidirectional approach ensures that even if the middle of the block changes, sub-blocks at both ends remain consistent (Green parts: 00, 11, 33) and can be matched, effectively mitigating boundary shift impacts within the block.

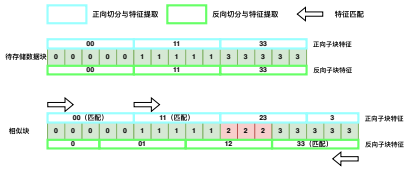

Figure 8: Exponential rounding bidirectional sub-block fixed-length segmentation

3.3 Model Data Separation Module

This module aims to accurately identify model data blocks. Since reliance on file extensions is unreliable, we analyze content characteristics. The architecture of the Model Data Separation Module is illustrated in Figure **??**.

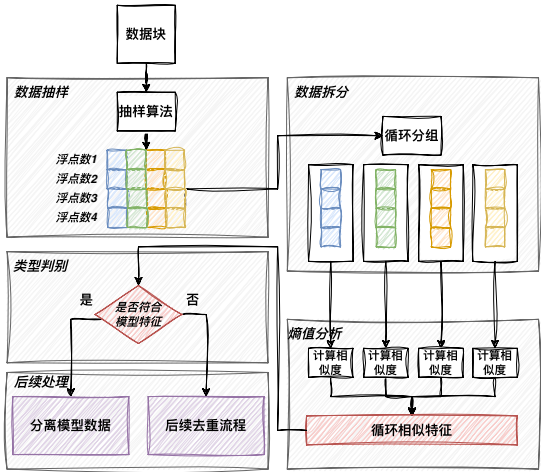

Figure 9: Model Data Separation Module Architecture

Principle: Entropy Distribution in Floating-point Numbers. The core distinction of model data lies in its representation. Deep learning models consist of massive arrays of floating-point numbers (FP32, BF16, FP16). As shown in Figure **??**, a floating-point number is composed of a sign bit, exponent bits, and mantissa bits. In a trained model, the weights within a layer usually share a similar dynamic range, meaning their *exponent bits* are highly similar (low entropy).

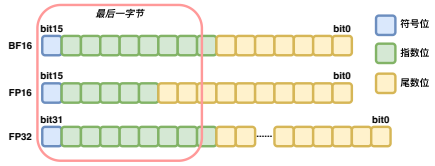

Figure 10: Representation of model floating-point numbers

To quantitatively verify this, we analyzed the entropy [**?**] of each bit of BF16 floating-point numbers in Transformer model files. As shown in Figure **??**, the entropy of the exponent part (bits 9~14) is significantly lower than that of the mantissa part. This confirms that while the mantissa bits are highly random (high entropy), the exponent bits contain a large amount of redundancy that traditional methods fail to exploit.

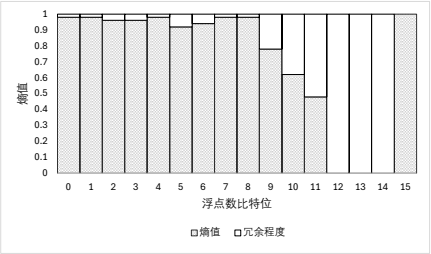

Figure 11: Bit entropy values of BF16 floating-point numbers in model files

Challenge. To detect this pattern, one might check the entropy of specific bits. However, operating at the bit granularity is computationally expensive due to complex bitwise shifting and masking (Figure **??**).

Optimization: Byte-granularity Grouped Sampling. To avoid bit-level complexity, we ex-

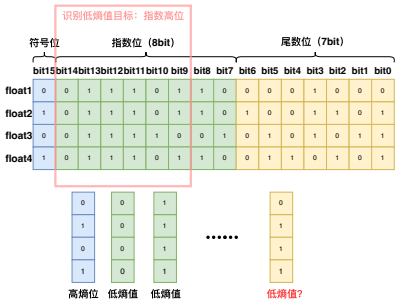

Figure 12: Complexity of bit-level implementation

ploit the byte alignment of floating-point formats. As Figure **??** shows, although exponent bits cross byte boundaries, the "most significant" parts of the exponent often dominate specific bytes. We propose a **Byte-granularity Grouped Sampling Entropy Analysis** method. Instead of parsing bits, the system simply treats the data block as a byte stream. It samples bytes at fixed offsets (e.g., stride of 4) to form 4 "byte groups". By calculating the entropy of each group, we can identify specific "low-entropy byte groups" (Figure **??**).

- If a low-entropy byte group and three high-entropy byte groups appear, it means that the data block is a model data block composed of 32-bit floating point numbers.
- If low-entropy occurs every 2 bytes, it indicates BF16/FP16.

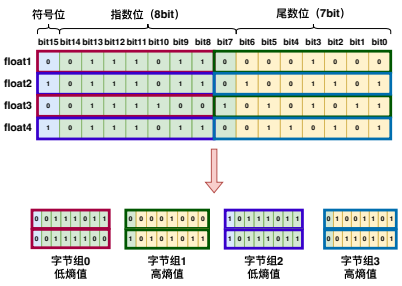

Figure 13: Byte-wise grouped sampling entropy analysis

3.4 Model Encoding Compression Module

The goal of the model data processing module is to identify and concentrate the numerically similar floating-point exponent bits in the model data block, so as to use general compression algorithms for data reduction.

Principle: Data Reorganization for Locality. Typical general-purpose compression algorithms (e.g., LZ4, Zstd) are designed to compress text or binary streams by finding *continuous local redundancy*. They use valid windows to match repeated byte sequences. However, in raw model data, the "redundant" bytes (exponent parts) are not continuous; they are separated by "random" bytes (mantissa parts) every few bytes. This "stride" pattern breaks the locality required by sliding-window compressors, causing them to fail to match the exponents and wasting effort on the random mantissas. To fix this, we must reorganize the data to make redundant parts *spatially adjacent*.

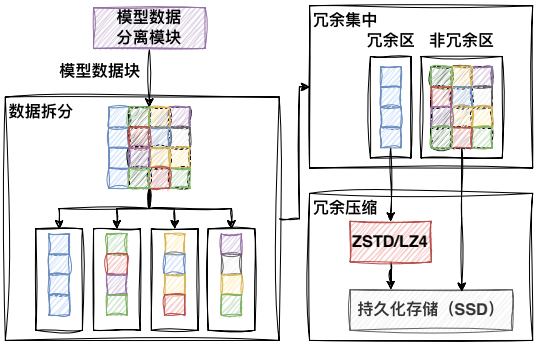

Figure 14: Model Data Processing Module Architecture

Optimization: Byte Grouped Compression Based on Entropy Analysis. We propose a **Byte Grouped Compression Method** as shown in Figure **??**. Crucially, we do not blindly reorganize all bytes. Since the separation module (Section 3.3) has already calculated the entropy distribution and identified which byte group contains the low-entropy exponents, we reuse this conclusion. As

shown in Figure **??**, the system extracts *only* the identified low-entropy byte groups (e.g., Group 1 for FP32) and concatenates them into a continuous block. This block, now consisting of a pure stream of similar exponents, is highly compressible by Zstd. The remaining high-entropy groups are stored essentially uncompressed or lightly compressed. This selective approach reduces the compression inputs by 50-75% compared to re-encoding the entire block, significantly optimizing CPU efficiency.

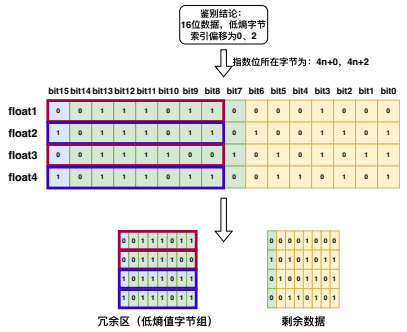

Figure 15: Byte-wise grouped compression based on entropy analysis

4 Implementation

4.1 Prototype Implementation

We implemented a prototype of FHRD based on the open-source deduplication system Destor. The core functions, including fine-grained sub-block deduplication, model data identification, and encoding compression, were implemented with C++ code. The system also integrates the Zstandard (Zstd) library [**?**] for general-purpose compression.

Data Block Structure. To accommodate fine-grained metadata, we reconstructed the data block structure. The container format was extended to store sub-block headers and determining flags, allowing the restoration engine to correctly identify and reassemble fine-grained deduplicated blocks and encoded model data.

4.2 Pipeline Parallelism

To improve the overall processing performance, we introduce a pipeline parallel processing method in the end-to-end flow. As shown in Figure **??**, each processing stage is handled by different threads in parallel, and data communication between threads is implemented through global queues. The pipeline includes: **Read** thread, **Chunk** thread, **Hash** thread, **Dedup** thread (checking multi-level cache), **Identify** thread (separating model data), **Sub-block Dedup** module (Fine-hash/Fine-dedup/Diff threads), **Model Encode** thread, **Compress** thread, and **Save** thread. This design maximizes the utilization of multi-core processor resources and hides the latency overhead of fine-grained processing.

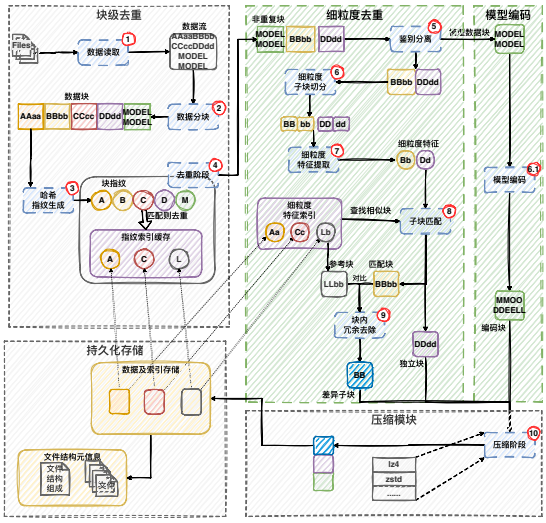

Figure 16: System deduplication storage workflow

4.3 Multi-level Index Cache

To cope with the extra index lookup overhead brought by fine-grained deduplication, the system adopts a multi-level index cache strategy [**?**], as shown in Figure **??**. The lowest layer is the Key-Value (KV) index storage in the disk. Above this, we design two levels of memory cache:

- **L1 Cache (Global Cache)**: Uses LRU policy to store the most frequently used data fingerprints mappings.
- **L2 Cache (Local Cache)**: Stores mappings for the currently processing data segment to reduce random access to the underlying storage.

Ideally, the system sequentially queries the Local Cache, Temporary Index Cache, and Global Cache during a duplicate block match or similar block lookup.

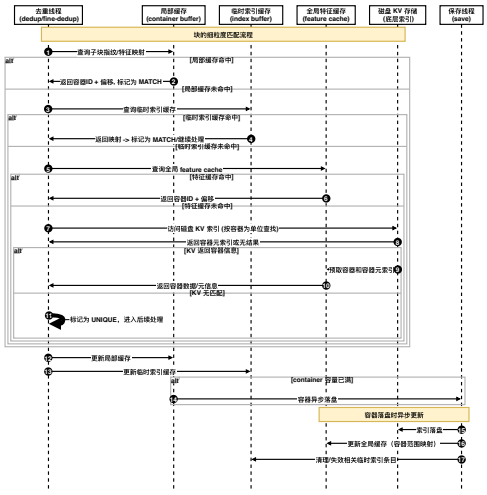

Figure 17: Matching lookup timing with multi-level index cache

5 Evaluation

5.1 Experimental Setup

Hardware Environment. We evaluate FHRD on a server equipped with two AMD EPYC 7763 processors (64 cores each, 128 logical cores total) and 128GB DDR4 DRAM. The storage setup consists of two 1TB SSDs for hosting the operating system and metadata, and five 16TB HDDs for data storage. The operating system is Ubuntu 24.04 LTS with Linux kernel 6.14.0.

Datasets. To comprehensively evaluate the system's performance on mixed workloads, we use a combination of non-model and model datasets. Non-model datasets include *vmi* (virtual machine images), *code* (source code repositories), and *backup* (container images). Model datasets include *checkpoints* (Transformer training checkpoints), *model16* (FP16/BF16 model slices), and *mixQ4FP32* (quantized models). We mix these datasets in varying proportions to simulate real-world cloud storage scenarios.

Comparison Schemes. We compare FHRD with several state-of-the-art schemes:

- **BASE**: A standard pipeline widely used in industry, consisting of variable-length chunking (CDC), block-level deduplication (SHA-256), and general-purpose compression (Zstd).
- **BURST+**: An optimized approach for intra-block redundancy that removes duplicate content at block boundaries [**?**], combined with variable-length chunking.
- **ZIPNN++**: A scheme that integrates model-specific compression (ZipNN [**?**]) into the deduplication pipeline to handle model data.
- **ODESS**: A fine-grained delta compression system that performs global byte-level matching [**?**]. We include it as an ideal baseline for reduction ratio, though its overhead is prohibitive for primary storage.

5.2 Overall Performance

Data Reduction Ratio. Figure **??** compares the data reduction rates (Reduction Ratio = Original Size / Reduced Size) of different schemes under various mixed workloads.

The results show that FHRD consistently outperforms other practical schemes.

- **vs. BASE**: FHRD achieves an average improvement of 21.7% over the BASE baseline. In workloads with high model data proportions, BASE's effectiveness drops significantly, while FHRD maintains a high reduc-

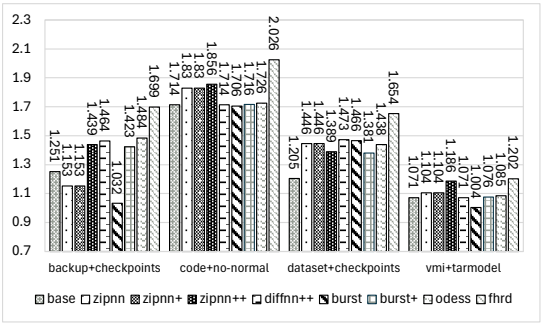

Figure 18: Comparison of Data Reduction Rates under Various Mixed Workloads

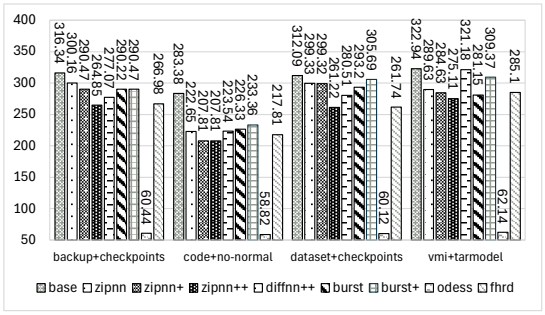

Figure 19: Comparison of System Throughput under Various Mixed Workloads

tion rate due to its model identification and encoding module.

- **vs. Partial Solutions**: Schemes like ZIPNN++ (model-only optimization) and BURST+ (boundary-only optimization) improve upon BASE by 5-10% but fail to address the full spectrum of mixed workload redundancy. FHRD combines the strengths of both, handling both intra-block local redundancy and model numerical redundancy.

- **vs. ODESS**: While ODESS achieves high reduction rates (sometimes exceeding FHRD slightly) due to its exhaustive byte-level matching, FHRD offers a comparable reduction ratio (within 5% margin) with significantly lower overhead, as discussed in the next section.

System Throughput. We further evaluate the system throughput (MB/s) of the end-to-end write path, as shown in Figure **??**.

Performance trade-offs are inevitable when introducing finer-grained processing.

- **FHRD Efficiency**: FHRD sustains more than 80% of the throughput of the lightweight BASE scheme. The overhead primarily comes from the additional entropy calculation and sub-block hashing. However, thanks to the pipeline parallelism and multi-level index cache optimizations described in Section 4, FHRD mitigates these costs effectively.

- **Impracticality of Delta Compression**: In contrast, ODESS exhibits a severe performance drop, achieving only 15% of BASE's throughput. The computational cost of global byte-level delta compression makes it unsuitable for high-performance online storage systems.

In summary, FHRD provides an optimal balance, delivering near-ideal data reduction rates with high, practical system throughput.

5.3 Detailed Analysis of System Components

To further validate the effectiveness of our design choices, we present a detailed analysis of the core components: sub-block deduplication, model data separation, and byte-grouped compression.

5.3.1 Effectiveness of Sub-block Deduplication

We evaluate the "Exponential Rounding Bidirectional Sub-block Fixed-length Segmentation" method by comparing it with a simplified version (FHRD-2, which uses uniform splitting) and BURST+ (which only handles boundary redundancy). Figure **??** shows the reduction rates on non-model datasets.

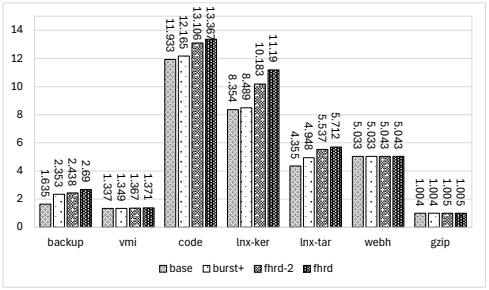

Figure 20: Comparison of Non-model Data Deduplication Rates

FHRD consistently outperforms FHRD-2 (by 8.9%) and BURST+ (by 13.5%). This proves that our exponential rounding strategy is superior to simple uniform division in capturing intra-block redundancy, especially in datasets with frequent modifications like *code* and *lnx-ker*. While BURST+ only addresses boundary redundancy, FHRD effectively uncovers redundancy deep within the blocks.

5.3.2 Accuracy and Efficiency of Model Separation.

The accuracy of the Model Data Separation Module is critical. We define a "model block" as one that gains >10% space savings from our specialized compression. We compared our **Byte-level** entropy analysis against File-level (extension-based) and Bit-level entropy analysis.

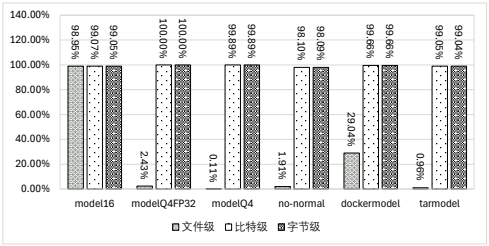

Figure 21: Accuracy of Model Data Identification Methods

As shown in Figure **??**, both Byte-level and Bit-level methods achieve >98% accuracy, significantly outperforming File-level detection (which suffers from high false positives on quantized models and false negatives on non-standard file extensions). However, in terms of throughput (Figure **??**), our Byte-level method is 69.6% faster than the Bit-level approach. This confirms that our byte-granularity optimization achieves the optimal balance between high accuracy and low overhead.

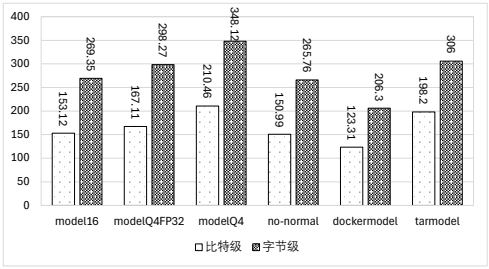

Figure 22: Throughput of Model Data Identification Methods

5.3.3 Effectiveness of Model Compression.

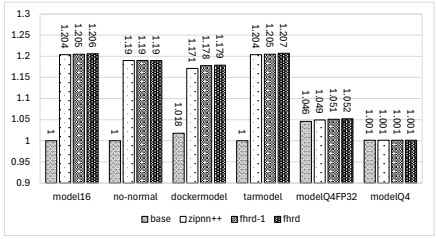

Figure 23: Effectiveness of Entropy-based Byte Grouped Compression

FHRD matches the reduction rate of the specialized ZIPNN++ on pure model data as shown in Figure **??**. Importantly, compared to FHRD-1, FHRD achieves similar reduction rates but with 80% higher compression throughput (Figure **??**). By selectively compressing only low-entropy byte groups, FHRD avoids wasting CPU cycles on random mantissa bytes, significantly optimizing the computational efficiency of the compression stage.

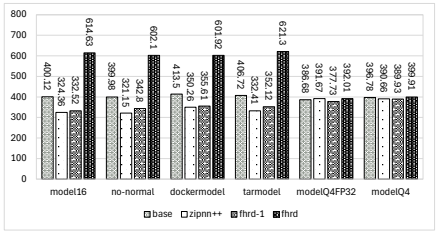

Figure 24: Comparison of Compression Thread Throughput

5.4 Impact of Workload Characteristics

Finally, we analyze how FHRD performs under the two evolving trends identified in Section 1: increasing model data proportion and larger block sizes.

Impact of Model Data Proportion. Figure **??** illustrates the reduction rate as the proportion of model data increases from 0% to 100%. Partial

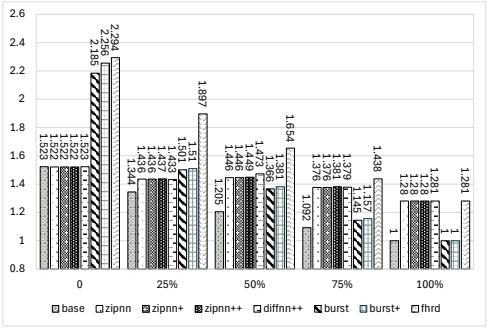

Figure 25: Deduplication Rate vs. Model Data Proportion

solutions like BURST+ degrade as model data increases, while model-specific tools like ZIPNN++ fail on non-model data. FHRD, however, maintains the highest reduction rate across the entire spectrum. This demonstrates its robustness in handling arbitrary mixed workloads, dynamically applying the best strategy for each data block.

Impact of Data Block Size. Figure **??** shows the reduction rate as the average block size grows from 4KB to 100KB. Traditional BASE deduplication suffers severely as blocks get larger. FHRD,

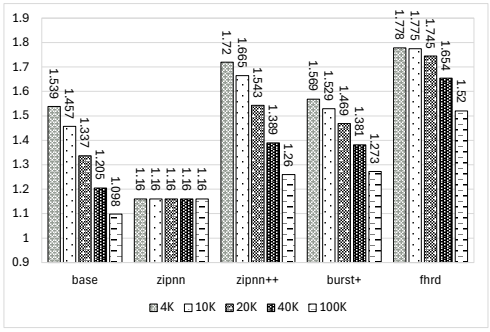

Figure 26: Deduplication Rate vs. Average Block Size

thanks to its sub-block deduplication, mitigates this decline. At 100KB block size, FHRD improves the reduction rate by 38.4% compared to BASE. This confirms that FHRD effectively solves the "large block" challenge by uncovering fine-grained redundancy that coarse-grained deduplication misses.

6 Related Work

Traditional Deduplication. Data reduction systems typically employ a pipeline of chunking, fingerprinting, deduplication, and compression [**?**]. **Fixed-Size Chunking (FSC)** segments data into uniform blocks, which is efficient but suffers from the "boundary shift problem". A single byte insertion can offset all subsequent boundaries, nullifying deduplication. **Content-Defined Chunking (CDC)**, using algorithms like Rabin fingerprinting [**?**], solves this by determining boundaries based on content rolling hashes [**?**, **?**]. This approach is the industry standard for handling data modifications but struggles with the "large block" trend, where local redundancy within large chunks remains undetected.

Delta Compression and Fine-grained Deduplication. To address intra-block redundancy, **Delta Compression** techniques like ODESS [**?**] enable byte-level deduplication. By globally searching for similar base chunks and storing only the differences (deltas) [**?**, **?**], they achieve high reduction rates. However, the computational overhead of fea-

ture extraction and delta calculation is prohibitive for high-performance primary storage. Other approaches like **Burst** [**?**] focus on specific redundancy patterns, such as identical file headers or footers, but fail to capture redundancy deep within the data blocks. FHRD's sub-block deduplication offers a more general and efficient solution for intra-block redundancy without the extreme cost of global delta compression.

Model Compression. With the rise of AI, model compression has become critical. Lossy methods like pruning and quantization are unsuitable for cloud storage where bit-exact retrieval is required. Lossless methods like **ZipNN** [**?**] reorganize floating-point data to improve compression. **ZipLLM** [**?**] exploits similarity between fine-tuned models. **FM-Delta** [**?**] handles massive fine-tuned models. However, these tools are highly specialized. They lack the ability to distinguish model data from mixed workloads. Applying them blindly to non-model data can degrade performance or even increase data size. FHRD bridges this gap by integrating model identification and specific compression into a general-purpose deduplication framework.

7 Conclusion

Focusing on the challenge of mixed workloads where model data and large block data coexist in modern cloud storage, this paper proposes the fine-grained redundancy identification data reduction system FHRD. By introducing fine-grained sub-block deduplication and model data identification and separation mechanisms, FHRD can effectively identify and remove intra-block local redundancy and model numerical redundancy. Experimental results show that FHRD significantly improves the data reduction rate under mixed workloads and has good system performance, providing an efficient data reduction solution for modern cloud storage systems.