Hybrid-Rewrite: A Rewriting Framework for Hybrid Deduplication and Delta Compression

Qiao Li*, Hong Jiang†, Zichen Xu*, Yucheng Zhang*§ Junyun Wu* Puchen Lu*
*School of Mathematics and Computer Sciences, Nanchang University, Nanchang, China
†Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, USA
§Corresponding author: zhangyc@ncu.edu.cn
Email:{liqiao@email.ncu.edu.cn, hong.jiang@uta.edu, xuz@ncu.edu.cn, zhangyc@ncu.edu.cn, wujunyun@ncu.edu.cn, lupuchen@email.ncu.edu.cn}

Abstract—**Data deduplication eliminates redundant data in backup systems by replacing duplicate chunks with compact references and consolidating unique chunks into larger containers. While effective, this process introduces fragmentation that degrades restore performance. Rewriting techniques mitigate fragmentation by identifying sparse containers (i.e., those referenced by the fewest chunks) and rewriting dependent duplicates. Recent work further integrates delta compression to exploit redundancy among similar but non-duplicate chunks. However, this hybrid approach introduces two challenges for rewriting: (1) prohibitive computational overhead from weak-hash-based sketching required for similarity detection, and (2) dynamic reference conflicts arising from multiple candidate base chunks during sparse container identification.**

In this paper, we propose Hybrid-Rewrite, a rewriting framework for backup systems that combine deduplication and delta compression. Hybrid-Rewrite integrates two techniques to address these challenges: (1) sketch echoing, which stores chunk sketches in containers and retrieves them via metadata prefetching during deduplication, thus eliminating redundant sketch computations for duplicate chunks, and (2) greedy reference locking, which iteratively detects containers with the most referenced chunks by aggregating all possible references and invalidates conflicting references, thereby isolating the sparse containers. Experimental results demonstrate that, compared to direct extensions of existing rewriting methods to hybrid systems, Hybrid-Rewrite achieves 1.42 to 1.8x higher compression ratios and up to 1.81x faster restore performance. Additionally, sketch echoing reduces sketch computation by 58.15% to 96.95%.

Index Terms—**delta compression, data deduplication, fragmentation, restore performance, backup storage.**

I. Introduction

Backup systems commonly employ data reduction techniques to enhance storage efficiency. Data deduplication is a widely adopted data reduction technique characterized by two features: (1) dividing data streams into small chunks (e.g., 8 KB) and (2) identifying duplicate chunks by comparing their cryptographic hash values (fingerprints) [1]. In deduplication-based backup systems, a chunk is stored only once, while its duplicate occurrences are replaced with references to the stored copy. Unique chunks are grouped into larger containers, typically several megabytes in size (e.g., 4 MB), and written to the disk. Upon completion of a backup, the system generates a sequence of fingerprints, called a recipe, where each fingerprint corresponds to a chunk in the backup stream. During restoration, files are reconstructed by sequentially retrieving chunks based on the fingerprints recorded in the recipe. Despite the poor random I/O performance of hard disk drives (HDDs), their cost-effectiveness makes them the popular storage medium for backup systems [2]–[4]. This paper focuses on HDD-based backup systems.

A critical limitation of deduplication-based backup systems is chunk fragmentation, wherein a backup's chunks are dispersed across multiple containers (via references) rather than consolidated in a single or few containers (as the original backup stream would before replacing duplicate chunks with references) [5], [6]. Thus, fragmentation degrades restore performance: the more dispersed a backup's chunks, the poorer the restore performance. To address this problem, prior work proposes rewriting [7], a technique that selectively stores duplicate chunks as unique copies to consolidate the current backup's physical layout into fewer containers. Rewriting operates at the container granularity, targeting sparse containers (those with minimal chunks from the current backup). By rewriting these chunks as unique, the backup eliminates dependencies on sparse containers, improving spatial locality. To minimize rewriting overhead, the selection of sparse containers prioritizes on those containing the fewest chunks referenced by the current backup.

While deduplication eliminates duplicate chunks, it ignores redundancy among similar but non-duplicate chunks. Delta compression can remove such redundant data [8]. To delta-compress a chunk (denoted as A), a similar chunk (denoted as B) that is already stored in the system is first identified as a base chunk. By removing redundant data shared between A and B from the target chunk A, delta compression encodes A into a delta chunk much smaller than A. The system stores this delta chunk instead of the original chunk A, thereby reducing storage overhead. The delta file can later be decoded with the base chunk to restore the original data. Prior to delta compression, a chunk's similar counterpart, the base chunk, must be identified, typically by calculating and comparing their sketches (i.e., weak hashes) [9]. Given that deduplication and delta compression reduce redundancy in complementary ways, recent works have combined both techniques in backup

This research was supported by the National Natural Science Foundation of China No. 62262042 and No. 62572222, and the Jiangxi Provincial Natural Science Foundation No. 20224BAB202017.

systems to maximize redundancy elimination: deduplication removes duplicates first, followed by delta compression to eliminate redundancy among similar, non-duplicate chunks [3], [10], [11].

Hybrid backup systems that combine deduplication and delta compression still require rewriting techniques to reduce fragmentation. However, existing rewriting techniques, designed for deduplication-based systems, only consider deduplication-derived references when identifying sparse containers, ignoring delta compression-induced references and thus reducing defragmentation effectiveness. Our analysis in Section III identifies two challenges in incorporating delta compression-induced references.

Challenge 1: the difficulty to identify sparse containers with the fewest referenced chunks. Compared to the simple reference relationships introduced by deduplication, those introduced by delta compression are more complex and sometimes mutually exclusive. Two scenarios illustrate this: (1) a chunk may have multiple similar chunks stored in different containers, but only one can be selected as the base chunk for delta compression. Selecting different base chunks may lead to different containers being marked as sparse, ultimately resulting in different rewriting decisions. (2) to minimize referenced chunks in certain containers, some duplicate chunks might need to forgo deduplication in favor of delta compression. These factors make it challenging to select the optimal sparse containers.

Challenge 2: the high computational overhead of computing sketches. As noted above, identifying sparse containers may require forgoing deduplication for certain duplicate chunks and applying delta compression instead. This means that the system must compute sketches for all chunks, including duplicates, in the incoming backup to find their similar chunks. This process incurs significant computational overhead. However, we observe that the sketches for duplicate chunks are precomputed, as every duplicate chunk has a previously stored copy whose sketch has already been calculated. By reusing these precomputed sketches, the system can eliminate redundant calculations and drastically reduce computational overhead.

Motivated by the above analysis, we propose Hybrid-Rewrite, a rewriting framework for hybrid backup systems that integrate both deduplication and delta compression. Hybrid-Rewrite incorporates two key techniques to identify sparse containers with the fewest referenced chunks and address the aforementioned challenges:

- **Sketch Echoing.** This technique builds on a typical deduplication strategy that organizes chunks into containers and prefetches container metadata during deduplication to accelerate duplicate detection. Specifically, it stores sketches alongside chunk metadata within containers. When a container is accessed for prefetching metadata during deduplication, sketches are also prefetched. The prefetched sketches are then delivered to duplicate chunks when they are identified via fingerprint matching, which eliminates redundant sketch computations (Challenge 2).
- **Greedy Reference Locking.** Instead of directly targeting containers with the fewest referenced chunks, which is difficult (Challenge 1), this technique identifies containers with the most referenced chunks, a task that is relatively straightforward. It iteratively identifies containers with the most referenced chunks by aggregating all possible references, invalidates conflicting references, and dynamically updates these references until the sparse containers emerge.

Experimental results using real-world datasets demonstrate that, compared to a naive hybrid approach that directly adapts existing rewriting methods, Hybrid-Rewrite achieves a compression ratio 1.42 to 1.8 times higher and improves restore performance by up to 1.81 times. Additionally, sketch echoing reduces sketch computations by 58.15% to 96.95%.

II. Background and Related Work

A. Deduplication-based Backup Systems

Backup systems typically exhibit high data redundancy [12], [13]. To improve storage efficiency, data deduplication is widely employed as a data reduction technique. A deduplication-based backup system identifies duplicate chunks in the backup data stream and replaces them with references, while grouping unique chunks into larger, fixed-sized (e.g., 4 MB) containers for storage [14], [15]. The workflow of such a system typically comprises three core stages: (1) chunking, which divides the backup stream into fixed-sized or variable-sized chunks using chunking algorithms [16]–[18], (2) fingerprinting, which computes cryptographic hash values (e.g., SHA-1) of chunks to generate unique chunk identifiers (also known as fingerprints), and (3) duplicate detection, which identifies duplicate chunks by maintaining a fingerprint index that maps chunk fingerprints to container identifiers and performing fingerprint matching.

Following backup completion, the system generates a recipe, a sequence of fingerprints that preserves the logical order of chunks in the backup stream. During restoration, the system retrieves chunks sequentially according to this recipe. In this process, entire containers containing target chunks are read, one by one as the basic unit, rather than individual chunks, thereby enabling subsequent chunks within the same container to be read concurrently. For HDD-based backup systems, restore performance primarily depends on the number of I/Os required to read containers, with more I/Os resulting in lower restore performance.

B. Fragmentation and Rewriting

Chunk Fragmentation. Deduplication inherently induces chunk fragmentation, a phenomenon where logically sequential chunks become physically distributed across non-contiguous storage containers instead of maintaining spatial locality [2]. This fragmentation arises because duplicate chunks in subsequent backups reference previously stored containers (also known as "old containers") rather than preserving physical adjacency, thereby disrupting the contiguous storage pattern essential for efficient sequential access.

As noted above, during restoration the system reads entire containers to reconstruct files. Although this container-centric I/O pattern facilitates prefetching of sequential chunks within the same container, thereby reducing I/O operations, fragmentation forces the system to read additional containers to retrieve scattered chunks, which significantly degrades restore performance. Restore performance is a critical metric, as higher restore throughput shortens the recovery window and improves overall system availability. For HDD-based systems, where random I/O latency dominates restoration overhead, fragmentation-induced read amplification becomes a major performance bottleneck.

Rewriting. A commonly used approach to reducing fragmentation is rewriting, which stores (rewrites) some duplicate chunks from the current backup alongside unique ones into new containers, thereby reducing references to old containers [2]. Rewriting operates at the container level, where referenced old containers selected for rewriting are referred to as *sparse containers*. For each sparse container, all duplicate chunks in the current backup that reference chunks within it are rewritten into the new containers, eliminating the backup's dependency on the old container and rendering it unreferenced.

Rewriting introduces a trade-off: rewriting more chunks increases redundant data storage, reducing storage efficiency. To minimize rewritten chunks, sparse containers are chosen to contain as few referenced chunks as possible. These containers are typically identified by computing the reuse ratio for each referenced container, defined as: reuse ratio=$\frac{Size\ of\ referenced\ chunks\ within\ the\ container}{Total\ container\ size}$. Containers with lower reuse ratios are prioritized as sparse containers.

Capping [19], a state-of-the-art rewriting approach, partitions the backup stream into non-overlapping segments, each containing a continuous sequence of chunks. Each segment is independently analyzed to identify sparse containers. The number of containers eligible for deduplication within a segment is capped at T, referred to as the capping level. If a segment contains more than T referenced containers, only the top T containers with the highest reuse ratios are retained. The remaining referenced containers that exceed the capping limit are designated as sparse containers.

C. Delta Compression

Delta compression compresses a target chunk (denoted as A) relative to a base chunk (denoted as B). By removing redundant data shared between A and B from A, delta compression encodes A into a compact delta chunk that can later be decoded with the base chunk to reconstruct the original data [8], [20].

The size of the delta generated by delta compression primarily depends on the similarity between the target chunk and the base chunk. The greater the similarity, the more redundant data can be removed, resulting in a smaller delta file. The process of identifying similar chunks is referred to as *similarity detection*, which involves calculating and indexing chunk sketches to locate similar ones [9], [21].

Sketch Calculation. A chunk's sketch consists of one or more weak hashes, referred to as *features*, derived from the chunk. Two chunks are considered similar if they share one or more features. A common method for computing a feature uses a rolling hash function with fixed-size windows (e.g., 48 bytes) over all overlapping regions of the chunk, selecting the maximal hash value as the feature [22], [23]. By applying multiple (e.g., N) different hash functions, multiple features can be generated. However, computing N distinct hash functions is computationally expensive. In practice, a more efficient approach is to compute a single hash value and then apply N linear transformations to it, producing N distinct features.

The computational overhead of sketch calculation is substantial for two reasons. First, the algorithm must examine nearly every byte in the chunk to extract features. Second, multiple features typically need to be computed; for example, the super-feature approach [24] (also known as the N-transform SF) requires computing 12 features per chunk [22], [23].

Chunking, a critical stage in the deduplication workflow, is a known performance bottleneck [17], [18]. Compared to chunking, sketch calculation incurs higher overhead. In addition to hash computations (also required for chunking), sketching requires 12 linear transformations and additional comparisons to select features [25], [26]. Although recent approaches such as Finesse [25], Odess [26], and DeepSketch [21] have introduced optimizations to reduce this cost, the overhead remains significant in practice.

Sketch Indexing. As previously mentioned, two chunks are considered similar if they share one or more matching features. Matching similar chunks typically requires indexing the features of all chunks. To enhance the similarity detection of similar chunks, multiple features of a chunk are often grouped into a super-feature for matching. To further increase the probability of detecting similar chunks, a chunk typically has multiple super-features for comparison. A typical method [24] calculates 12 features per chunk, which are then grouped into three super-features, each containing four features.

Combining Deduplication and Delta Compression. Delta compression complements deduplication by addressing fine-grained redundancies between unique but similar chunks, thereby enhancing storage efficiency when applied to deduplication-based backup systems. Such hybrid systems first eliminate duplicate chunks through deduplication, then search for similar chunks for the remaining unique chunks, and finally perform delta compression on identified similar chunks. Prior work [22], [27] has addressed the memory overhead of sketch indexing, but many challenges remain for systems that integrate deduplication and delta compression.

III. Observations and Motivations

Integrating delta compression into deduplication-based systems introduces dependency on base chunks, requiring prohibitive additional I/O operations to retrieve them and rendering post-deduplication delta compression impractical until very recently. New advances [3], [4], [28] partially mitigate

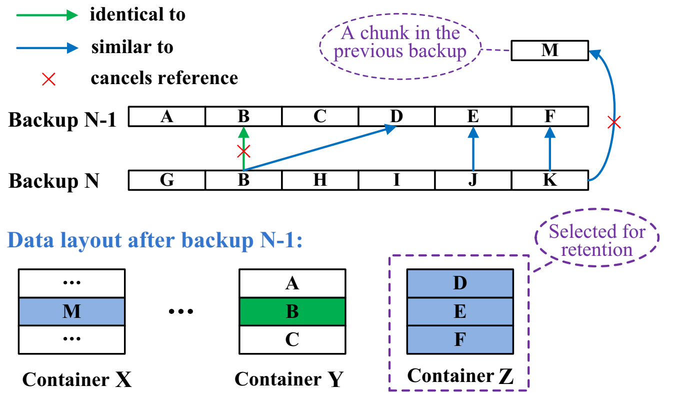

Fig. 1. Consider a system that has ingested $N-1$ prior backups, with Container Z as the most recently written. The system is now processing Backup N, constrained to reference only one prior (old) container. Backup N contains five unique chunks (G, H, I, J, K) and one duplicate chunk (B). Chunk J is similar to chunk E, while chunk K is similar to chunks F and M, with M residing in Container X. Chunk B, originally referencing Container Y via deduplication, is similar to chunk D. All referenced chunks (D, E, F) are stored in container Z, resulting in the highest reuse ratio for Z. Consequently, container Z is prioritized for retention. To mitigate fragmentation, the system should cancel K's reference to M and disable deduplication for B, and instead, applying delta compression against base chunk D.

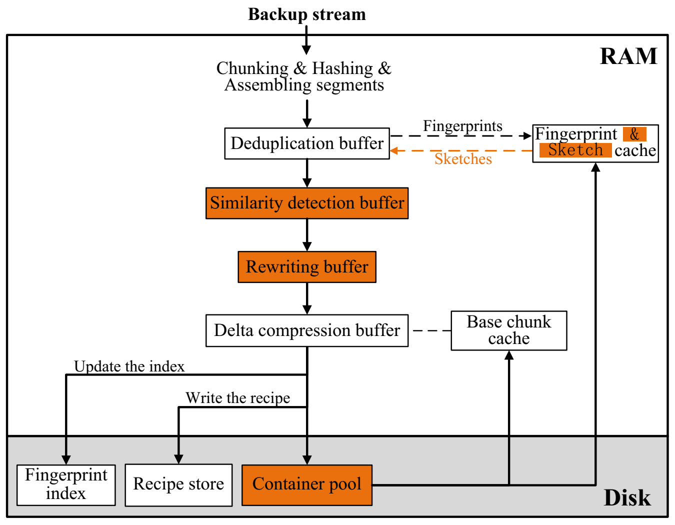

Fig. 2. Hybrid-Rewrite Architecture. Orange-highlighted regions indicate modified data structures and processes in existing hybrid backup systems combining deduplication and delta compression.

I/O bottlenecks, making hybrid backup systems feasible and enabling the development of hybrid rewriting strategies. In this section, we analyze the challenges of applying rewriting to hybrid backup systems that combine deduplication and delta compression and discuss potential solutions.

Challenge 1: Dynamic Reference Conflicts in Reuse Ratio Calculation. Rewriting techniques typically calculate the reuse ratio of referenced containers and select those with the lowest ratios as sparse containers. To avoid referencing these sparse containers, duplicate chunks referencing sparse containers are rewritten, and in hybrid systems, delta compression is disabled for chunks with base chunks in sparse containers. However, existing rewriting methods, designed for deduplication-based systems, only consider deduplication-derived references (i.e., duplicate chunks referencing stored copies in old containers) when computing reuse ratios. This approach is insufficient for hybrid systems, as base chunks in old containers must also be included in reuse calculations. Delta compression–induced references introduce two complications that make identifying sparse containers with the lowest reuse ratios more complex.

First, a chunk may have multiple similar chunks in different containers. These similarities create mutually exclusive reference dependencies in reuse ratio calculations, as delta compression requires selecting a single base chunk. Different base chunk selections produce divergent reuse ratios and sparse container candidates, leading to inconsistent rewriting decisions. Second, a duplicate chunk may forgo deduplication and instead be delta-compressed to optimize sparse container selection.

Figure 1 illustrates a simplified example of both cases. In each scenario, reference decisions are mutually exclusive; selecting one invalidates others. Therefore, a fundamental challenge in hybrid systems is resolving these dynamic reference conflicts to accurately identify containers with globally minimal reuse ratios.

While directly identifying containers with minimal reuse ratios is challenging, detecting those with maximal reuse ratios is straightforward by aggregating all possible references. Based on this insight, we can iteratively identify containers with maximal reuse ratios, leaving the remaining referenced containers as those with minimal reuse ratios.

Challenge 2: Overhead of Sketch Computation for Duplicate Chunks. As discussed above, identifying sparse containers with the lowest reuse ratios in hybrid systems may require selectively disabling deduplication for certain duplicate chunks and applying delta compression instead. Our evaluation on four datasets (detailed in Table I) reveals that this conversion is frequently required; on the WEB dataset, up to 12% of duplicate chunks require conversion. Failing to convert these chunks leads to suboptimal sparse container selection, reducing compression gains by 29.4% to 44.3% and degrading restore performance by 4.6% to 44.6% across the datasets.

We observe that sketches for duplicate chunks are redundant, as each duplicate chunk has a previously stored copy with a precomputed sketch. By reusing these precomputed sketches, the system can eliminate redundant calculations, substantially reducing overhead.

IV. Design and Implementation

A. System Architecture

Hybrid-Rewrite aims to accurately identify sparse containers with minimal reuse ratios in hybrid backup systems combining deduplication and delta compression while eliminating redundant sketch computations for duplicate chunks. To achieve this, it integrates the following two key techniques.

- **Sketch Echoing.** Stores and prefetches sketches with chunk metadata, propagating them to duplicate chunks, eliminating the need to recompute sketches for duplicate chunks (see Section IV-B).
- **Greedy Reference Locking.** Iteratively identifies containers with the highest reuse ratios, invalidates conflicting references, and isolates sparse containers (see Section IV-C).

Figure 2 illustrates the architecture of Hybrid-Rewrite. In this system, a backup stream is segmented into chunks, each of which is fingerprinted and subsequently grouped into segments, each containing a continuous sequence of chunks. These segments are processed sequentially through several buffers. In the deduplication buffer, duplicate chunks are identified while their corresponding sketches are obtained. Subsequently, for each chunk, be it duplicate or unique, potential similar chunks are identified in the similarity detection buffer. In the rewriting buffer, sparse containers and fragmented chunks are identified. In the delta compression buffer, the system performs delta compression for unique and fragmented chunks, provided that their corresponding base chunks exist.

During backup, the system maintains an active container for each backup stream. After the incoming backup stream undergo deduplication and delta compression, the system appends unique chunks, rewritten chunks, deltas, and their corresponding metadata into the active container. Once the container reaches its capacity, it is flushed to disk, and a new container is initiated to accommodate subsequent data. Concurrently, file recipes are stored for future restoration, and the on-disk fingerprint index is updated to reflect the fingerprints of the stored chunks and deltas.

B. Sketch Echoing

Sketch echoing builds on a typical deduplication strategy that groups chunks into containers and prefetches container metadata during deduplication to accelerate duplicate detection. We first outline the baseline strategy to contextualize our design and then describe how sketch echoing is implemented on this foundation.

Baseline Deduplication Architecture. A critical performance bottleneck in duplicate detection lies in fingerprint index management. While in-memory index enables low-latency lookups, it sacrifices scalability; disk-resident index, though scalable, introduces high I/O overhead [14]. To balance this trade-off, Zhu et al. [14] proposed a hybrid approach that stores the fingerprint index on HDDs while leveraging locality-based caching to minimize disk I/O during index queries. Their design partitions each container into two sections: a data section, which stores chunk and delta contents, and a metadata section, which records chunk metadata, including fingerprints, offsets, and lengths within the data section. A RAM-resident fingerprint cache accelerates lookups. When a chunk's fingerprint matches an entry in the on-disk index, the system prefetches the metadata of the corresponding container into the cache. Due to spatial locality, subsequent duplicate chunks will likely hit cached fingerprints, thereby reducing disk I/O operations. Additionally, to further minimize unnecessary index queries, Zhu et al. introduced a Bloom filter to pre-filter unique chunks before querying the fingerprint index or cache [14].

Sketch Integration and Echoing. We augment the metadata section by storing sketches alongside fingerprints, offsets, and lengths. When a container's metadata is prefetched, the sketches are also loaded into the cache. In our system, this cache is referred to as the fingerprint and sketch cache since it stores both fingerprints and sketches. Each chunk's fingerprint and sketch are collocated in the metadata section, preserving their association during prefetching. For duplicate chunks identified via fingerprint matches, their sketches are directly inherited from the cached metadata. This eliminates redundant sketch computations for duplicates while maintaining compatibility with existing prefetching mechanisms.

C. Greedy Reference Locking

Existing approaches that integrate deduplication and delta compression, such as PFC-delta and LoopDelta, employ a sequential workflow in which deduplication precedes rewriting, which in turn precedes similarity detection and delta compression. This architecture restricts rewriting to operate only on deduplication-derived references, ignoring references introduced by delta compression. In contrast, our system performs rewriting after both deduplication and similarity detection, ensuring all references (including delta-compressed base chunks) are considered during reuse ratio computation.

During similarity detection, when a chunk has multiple candidate base chunks, Hybrid-Rewrite tracks all candidate base chunks. To identify containers with minimal reuse ratios, we introduce two key modifications to conventional rewriting frameworks:

Modification 1: Comprehensive Reuse Ratio Computation. Existing methods compute reuse ratios only based on deduplication references (i.e., referenced duplicate chunks, hereinafter abbreviated as *DupChunk*). Greedy reference locking expands this by incorporating two delta-specific reference types: referenced duplicate deltas (abbreviated as *DupDeltas*) and base chunks (abbreviated as *BaseChunks*). The revised reuse ratio is defined as:

$$\frac{\sum(DupChunk \; + \; DupDelta \; + \; BaseChunk)}{Total \; container \; size}.$$

Modification 2: Inverse Container Identification. Directly identifying containers with minimal reuse ratios is intractable. To address this, we invert the process by iteratively aggregating all available references in order to identify containers with maximal reuse ratios. Containers not selected through this process are naturally characterized by low reuse.

For clarity, we define *competing references* as mutually exclusive dependencies arising during duplicate detection and similarity detection. A duplicate chunk inherently contains a deduplication-derived reference to a previously stored copy. If this duplicate chunk has one or more similar-chunk candidates identified during similarity detection, all associated references, including its deduplication reference, are classified as competing references. Retaining any one of the competing references invalidates all unretained ones. Similarly, for any chunk with multiple similar chunks identified during similarity detection, all references to candidate base chunks are competing references, as only one may remain after conflict resolution.

Although existing rewriting strategies use varied criteria to identify sparse containers, the implementation of Greedy Reference Locking within these frameworks remains largely consistent. Building on the Capping framework [19], which segments backup streams, limits deduplication to the top T highest-reuse containers per segment (with T being a tunable *capping level*), and designates the remaining containers as sparse, we introduce the following per-segment workflow for greedy reference locking:

- **Greedy Selection:** The reuse ratio of each container is calculated by aggregating all candidate references tracked during duplicate detection and similarity detection, including DupChunks, DupDeltas, and competing similar-chunk candidates. Then, the container with the highest ratio is selected, and all chunks referencing it are locked.
- **Conflict Resolution:** For each locked chunk, competing references are invalidated to ensure referential consistency.
- **Dynamic Recalculation:** The selected container is removed from consideration, and reuse ratios for remaining containers are recomputed after pruning invalidated references.

This three-phase workflow iterates until either T containers are selected or all referenced containers are processed. Containers not selected are classified as sparse, and all their associated references are eliminated through rewriting.

Discussion. Sketch echoing is designed for backup systems that group chunks into containers and prefetch container metadata during duplicate detection. This mechanism avoids introducing additional I/O overheads for sketch retrieval by collocating and piggybacking sketches with metadata. While it eliminates redundant sketch computations for duplicate chunks, identifying containers with globally minimal reuse ratios incurs computational overhead due to: (1) the iterative process to identify maximum-reuse containers, and (2) conflict resolution during reference aggregation.

Optimization strategies, such as batch-selecting high-reuse containers during early iterations to minimize iteration counts, can reduce this overhead. Although batch selection may influence sparse container identification, the probability of such interference diminishes when applied selectively in initial stages. However, this paper primarily focuses on establishing foundational principles for hybrid system rewriting frameworks and proving the concept.

V. Performance Evaluation

A. Evaluation Setup

Experimental Platform. Our evaluations were performed on a machine equipped with a 12-core Intel Xeon Silver 4215R CPU, 32 GB of DRAM, an 8 TB Seagate 7200RPM SATA III HDD, and a 1 TB Samsung 860 PRO SSD. The SSD was used to simulate the user space, while the HDD simulated the backup space.

Evaluated Approaches and System Configurations.

- **Dedup:** A standard exact deduplication approach proposed by Zhu et al. [14], which employs an on-disk fingerprint index combined with an in-memory fingerprint cache and a Bloom filter to reduce I/O overhead during fingerprint index queries.
- **Dedup & Capping:** Dedup enhanced with the Capping rewriting scheme [19]. In this configuration, Capping considers only deduplication-derived references when calculating the reuse ratio of containers.
- **Greedy:** A hybrid approach that integrates deduplication and delta compression to eliminate both duplicate and similar redundancy. Greedy represents the theoretical upper bound on compression achievable with both techniques. Due to its extremely high compression ratios on certain datasets (e.g., up to $231\times$ on RDB), which would obscure the improvements brought by Hybrid-Rewrite, we report its results in Table I as a dataset characteristic rather than include it in comparative figures.
- **Greedy & Capping+:** Greedy enhanced with Capping+, which is a modified Capping scheme that accounts for both deduplication-derived and delta compression-induced references, as detailed in Modification 1 of Section IV-C.
- **Greedy & Hybrid-Rewrite:** Greedy enhanced with our proposed Hybrid-Rewrite rewriting scheme.

For the deduplication component across all evaluated approaches, we adopt a Rabin-based chunking algorithm with average, minimum, and maximum chunk sizes set to 8 KB, 2 KB, and 64 KB, respectively, and calculate chunk fingerprints using SHA-1. For the delta compression component, including Hybrid-Rewrite, we use Odess [26] to generate chunk sketches and Xdelta [8] for delta encoding. Odess was configured to generate 12 features, grouped into 3 super-features (each containing 4 features) for matching similar chunks. The segment size was set to 20 MB (slightly less than 20 MB in practice; a segment boundary is declared when adding another chunk would exceed 20 MB). The container size is set to 4 MB, and the base chunk cache is implemented as an LRU cache with a capacity of 128 containers (512 MB). During restoration, the restoration cache is configured as an LRU cache with a capacity of 256 containers (1 GB).

For all approaches that include Capping, the capping level is set to 14 per 20 MB, i.e., at most 14 old containers can be referenced during deduplication and delta compression for every 20 MB of backup data, consistent with [7]. Since our segment size is 20 MB, this setting corresponds to deduplicating against at most 14 old containers per segment.

Performance Metrics. We evaluate our scheme using two metrics. The *compression ratio* measures the total space savings from deduplication and delta compression, defined as $\frac{Size\ of\ input\ data\ stream\ before\ compression}{Size\ of\ input\ data\ stream\ after\ compression}$. A higher com-

TABLE I
WORKLOAD CHARACTERISTICS OF THE FOUR DATASETS USED IN THE PERFORMANCE EVALUATION. MCR DENOTES THE MAXIMUM COMPRESSION RATIO ACHIEVABLE, AS MEASURED BY THE *Greedy* APPROACH.

Name	Size	MCR	Workload descriptions
WEB	365 GB	78	133 days' snapshots of the website:news.sina.com [29], which are collected by *wget*.
CHRO	284 GB	48	100 versions of source codes of chromium project.
LNX	365 GB	45	300 versions of Linux kernel source code.
RDB	1080 GB	231	200 versions of Redis database snapshots.

TABLE II
THE AMOUNT OF SKETCH CALCULATION AVOIDED BY SKETCH ECHOING.

Dataset	Avoided Sketch Calculation (%)
WEB	84.45%
CHRO	62.94%
LNX	58.15%
RDB	96.95%

pression ratio indicates greater space savings. The *speed factor* [19], defined as the average data size restored per container read, quantifies the restore performance. A higher speed factor value corresponds to superior restore performance. To align with prior work [7], [19], all reported speed factors represent the average across the final 20 backups in each experimental sequence.

Evaluated Datasets. Our evaluation utilized four real-world datasets, with characteristics detailed in Table I. Both the CHRO and LNX datasets store each backup as a monolithic TAR archive. The "MCR" column reports the theoretical compression ratio upper bound achieved by *Greedy* approach. We exclude *Greedy* from comparative figures to avoid obscuring the relative benefits of our proposed rewriting scheme.

B. Efficiency of Sketch Echoing

Table II presents the percentage of chunks for which sketch calculation was avoided due to sketch echoing. The results show that sketch echoing reduces sketch computation by 58.15% to 96.95%. The RDB dataset exhibits exceptional performance (96.95%) due to its high proportion of duplicate chunks, enabling near-complete elimination of redundant sketch calculations.

C. Compression Ratio

Figure 3 compares *Greedy & Hybrid-Rewrite*'s compression ratio to three approaches across four datasets. The results reveal that *Greedy & Hybrid-Rewrite* achieves a compression ratio 1.42 to 1.8 times higher than that of *Greedy & Capping+*. This improvement stems from Hybrid-Rewrite's ability to accurately identify sparse containers with the lowest reuse ratios, thereby minimizing rewritten chunks. Additionally, both *Greedy & Hybrid-Rewrite* and *Greedy & Capping+* outperform *Dedup & Capping*, owing to the added redundancy removal from delta compression.

Notably, on the RDB dataset, *Dedup* achieves the highest compression ratio, as shown in Figure 3. However, *Greedy*

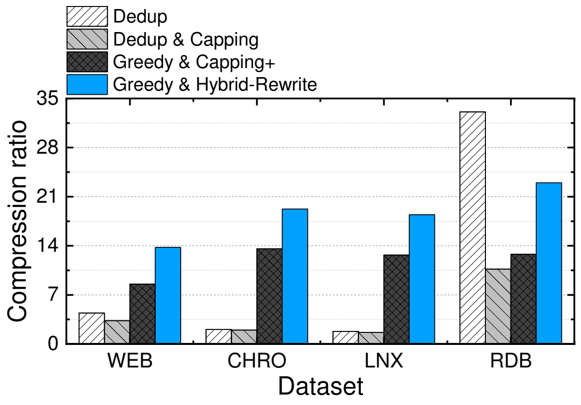

Fig. 3. Compression ratio of the four approaches on the four datasets.

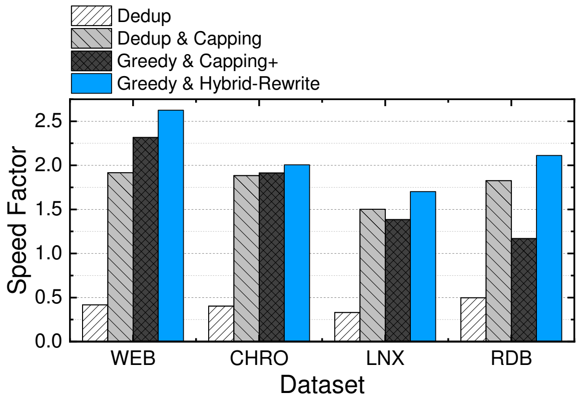

Fig. 4. Speed factor of the four approaches on the four datasets.

actually attains a higher compression ratio than *Dedup* (see Table I). To enhance restore performance, Capping (or Capping+) prevents some redundancy from being eliminated, resulting in a reduced compression ratio. When the compression ratio is already high, small changes in redundancy can significantly affect the ratio. For the RDB dataset, deduplication alone yields a compression ratio of 33. Thus, Capping's reduction of a small amount of redundancy leads to a noticeable decrease in compression ratio, which, as discussed in the next subsection, translates into improved restore performance.

D. Speed Factor

Figure 4 depicts the speed factor of the four approaches across the four datasets. The results demonstrate that *Greedy & Hybrid-Rewrite* achieves speed factors 1.05 to 1.82 times higher than those of *Greedy & Capping+*. This enhancement arises because the speed factor depends on both the number of referenced old containers and the number of newly written containers (both must be read into memory to supply chunks or deltas during restoration). Fewer rewritten chunks in *Greedy & Hybrid-Rewrite* result in fewer newly written containers, yielding a higher speed factor. Moreover, *Greedy & Hybrid-Rewrite* also outperforms *Dedup & Capping* in speed factor.

Overall, by accurately identifying sparse containers with the lowest reuse ratios, Hybrid-Rewrite outperforms direct extensions of Capping (i.e., Capping+) in both compression ratio and restore performance.

VI. Conclusion

In this paper, we present Hybrid-Rewrite, a rewriting framework for backup systems that integrate deduplication and delta compression. By employing sketch echoing and greedy reference locking, Hybrid-Rewrite effectively addresses two key challenges: the high computational demands of sketch generation and the dynamic reference conflicts during sparse container identification. Experimental evaluations on real-world workloads reveal that Hybrid-Rewrite outperforms a straightforward hybrid extension of existing rewriting schemes in both compression ratio and restore performance, and sketch echoing alone reduces sketch computations by up to 96.95%.

References

[1] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[2] X. Zou, J. Yuan, P. Shilane, W. Xia, H. Zhang, and X. Wang, "The dilemma between deduplication and locality: Can both be achieved?" in *the 19th USENIX Conference on File and Storage Technologies (FAST'21)*. USENIX Association, February 23 - 25 2021, pp. 171–185.

[3] X. Zou, W. Xia, P. Shilane, H. Zhang, and X. Wang, "Building a high-performance fine-grained deduplication framework for backup storage with high deduplication ratio," in *the 2022 USENIX Annual Technical Conference (ATC'22)*. Carlsbad, CA, USA: USENIX Association, July 11 - 13 2022, pp. 19–36.

[4] Y. Zhang, H. Jiang, D. Feng, N. Jiang, T. Qiu, and W. Huang, "Loopdelta: Embedding locality-aware opportunistic delta compression in inline deduplication for highly efficient data reduction," in *the 2023 conference on USENIX Annual Technical Conference (ATC'23)*. Boston, MA, USA: USENIX Association, July 10 - 12 2023, pp. 133–148.

[5] Y. Nam, G. Lu, N. Park, W. Xiao, and D. H. Du, "Chunk fragmentation level: An effective indicator for read performance degradation in deduplication storage," in *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*. Banff, Canada: IEEE Computer Society Press, September 02 - 04 2011, pp. 581–586.

[6] Z. Cao, S. Liu, F. Wu, G. Wang, B. Li, and D. H. Du, "Sliding look-back window assisted data chunk rewriting for improving deduplication restore performance," in *the 17th USENIX Conference on File and Storage Technologies (FAST'19)*. Boston, MA, USA: USENIX Association, February 25 - 28 2019, pp. 129–142.

[7] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, J. Liu, and et al, "Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 855–868, 2015.

[8] J. P. MacDonald, "File system support for delta compression," Master's thesis, University of California, Berkeley, Berkeley, CA, May 2000.

[9] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Combinatorial pattern matching*. Springer, 2000, pp. 1–10.

[10] Y. Zhang, H. Jiang, M. Shi, C. Wang, N. Jiang, and X. Wu, "A high-performance post-deduplication delta compression scheme for packed datasets," in *IEEE 39th International Conference on Computer Design (ICCD'21)*. IEEE, October 24 - 27 2021, pp. 464–471.

[11] Y. Zhang, W. Zeng, H. Jiang, D. Feng, Z. Xu, S. He, M. Zhang, and D. Wu, "An efficient delta compression framework seamlessly integrated into inline deduplication," *ACM Transactions on Storage*, 2025.

[12] D. Meyer and W. Bolosky, "A study of practical deduplication," in *the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. San Jose, CA, USA: USENIX Association, February 15 - 17 2011, pp. 229–241.

[13] G. Amvrosiadis and M. Bhadkamkar, "Identifying trends in enterprise data protection systems," in *the 2015 conference on USENIX Annual Technical Conference*. Santa Clara, CA: USENIX Association, July 08 - 10 2015, pp. 151–164.

[14] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. San Jose, CA, USA: USENIX Association, February 26 - 29 2008, pp. 269–282.

[15] Y. Allu, F. Douglis, M. Kamat, R. Prabhakar, P. Shilane, and R. Ugale, "Can't we all get along? redesigning protection storage for modern workloads," in *the 2018 conference on USENIX Annual Technical Conference (ATC'18*. Boston, MA, USA: USENIX Association, July 11 - 13 2018, pp. 705–718.

[16] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, "FastCDC: A fast and efficient content-defined chunking approach for data deduplication," in *the 2016 conference on USENIX Annual Technical Conference (ATC'16)*. Denver, CO: USENIX Association, June 15 - 17 2016, pp. 101–114.

[17] F. Ni and S. Jiang, "RapidCDC: Leveraging duplicate locality to accelerate chunking in CDC-based deduplication systems," in *the 10th ACM Symposium on Cloud Computing (SoCC'19)*. Santa Cruz, CA, USA: ACM Association, 2019, pp. 220–232.

[18] X. Jin, H. Liu, C. Ye, X. Liao, H. Jin, and Y. Zhang, "Accelerating content-defined chunking for data deduplication based on speculative jump," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 9, pp. 2568–2579, 2023.

[19] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *the 11th USENIX Conference on File and Storage Technologies (FAST)*. San Jose, CA, USA: USENIX Association, February 12 - 15 2013, pp. 183–197.

[20] D. Trendafilov, N. Memon, and T. Suel, "zdelta: An efficient delta compression tool," 2002.

[21] J. Park, J. Kim, Y. Kim, S. Lee, and O. Mutlu, "Deepsketch: A new machine learning-based reference search technique for post-deduplication delta compression," in *the 20th USENIX Conference on File and Storage Technologies (FAST'22)*. Santa Clara, CA, USA: USENIX Association, February 22 - 24 2022, pp. 247–264.

[22] P. Shilane, M. Huang, G. Wallace, and et al, "WAN optimized replication of backup datasets using stream-informed delta compression," in *the Tenth USENIX Conference on File and Storage Technologies (FAST'12)*. San Jose, CA, USA: USENIX Association, February 14 - 17 2012, pp. 1–14.

[23] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, "Migratory compression: Coarse-grained data reordering to improve compressibility," in *the 12th USENIX Conference on File and Storage Technologies (FAST'14)*. Santa Clara, CA, USA: USENIX Association, February 17 - 20 2014, pp. 257–271.

[24] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences (SEQUENCES'97)*. Washington, DC, USA: IEEE, June 13 1997, pp. 21–29.

[25] Y. Zhang, W. Xia, D. Feng, H. Jiang, Y. Hua, and Q. Wang, "Finesse: fine-grained feature locality based fast resemblance detection for post-deduplication delta compression," in *the 17th USENIX Conference on File and Storage Technologies (FAST'19)*. Boston, MA, USA: USENIX Association, February 25 - 28 2019, pp. 121–128.

[26] X. Zou, C. Deng, W. Xia, P. Shilane, H. Tan, H. Zhang, and X. Wang, "Odess: Speeding up resemblance detection for redundancy elimination by fast content-defined sampling," in *the 37th International Conference on Data Engineering (ICDE'21)*. IEEE, April 19 - 22 2021, pp. 480–491.

[27] P. Shilane, G. Wallace, M. Huang, and W. Hsu, "Delta Compressed and Deduplicated Storage Using Stream-Informed Locality," in *the 4th USENIX conference on Hot Topics in Storage and File Systems*. Boston, MA, USA: USENIX Association, June 13 - 14 2012, pp. 201–214.

[28] Y. Zhang, H. Jiang, C. Wang, W. Huang, M. Chen, Y. Zhang, and L. Zhang, "Applying delta compression to packed datasets for efficient data reduction," *IEEE Transactions on Computers*, vol. 73, no. 1, pp. 73–85, JAN 2024.

[29] "Sina news," http://news.sina.com.cn/.