StreamDedup: Distributed In-line Deduplication for Disaggregated Storage

JIAYONG LI, ETH Zürich, Switzerland
JONAS DANN, ETH Zürich, Switzerland
ZHENHAO HE*, ETH Zürich, Switzerland
GUSTAVO ALONSO, ETH Zürich, Switzerland
SAI RAHUL CHALAMALASETTI, d-Matrix, USA
DEJAN MILOJICIC, Hewlett Packard Enterprise, USA
LANCE EVANS, Hewlett Packard Enterprise, USA
ALEX VEPRINSKY, Hewlett Packard Enterprise, USA
RUNBIN SHI, Microsoft, USA

Efficient data reduction techniques, including deduplication and compression, are essential in storage systems, affecting performance and longevity. Existing data deduplication approaches often focus on intra-SSD deduplication, missing opportunities for cross-node deduplication, or have scalability issues when aiming for low latency and high throughput data reduction on large-scale, distributed SSD arrays. We propose StreamDedup, a distributed stream accelerator implementing a transparent layer of deduplication as a network-attached, middle-tier service between the compute and storage tiers. StreamDedup manages all aspects of data deduplication and compression and can be seamlessly integrated into existing systems. It is RDMA-enabled and highly scalable, enhancing data processing capacities for large-scale storage systems. Our prototype, deployed on FPGAs, demonstrates that StreamDedup achieves a throughput of 12.7 GB/s on a single node, matching the network bandwidth of disaggregated storage, with a latency of less than 50 µs. Across 10 nodes StreamDedup shows an almost linear increase in throughput with less than 60 µs of latency.

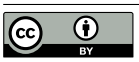
CCS Concepts: • **Hardware → Emerging technologies**; • **Computer systems organization → Reconfigurable computing**; **Cloud computing**; • **Information systems → Storage architectures**.

Additional Key Words and Phrases: Storage Deduplication, Hardware Acceleration, Disaggregated Storage, Compression

ACM Reference Format:
Jiayong Li, Jonas Dann, Zhenhao He, Gustavo Alonso, Sai Rahul Chalamalasetti, Dejan Milojicic, Lance Evans, Alex Veprinsky, and Runbin Shi. 2026. StreamDedup: Distributed In-line Deduplication for Disaggregated Storage. *ACM Trans. Reconfig. Technol. Syst.* 1, 1 (January 2026), 26 pages. https://doi.org/10.1145/3799896

*Now at NVIDIA.

Authors' Contact Information: Jiayong Li, ETH Zürich, Switzerland, jiayong.li@inf.ethz.ch; Jonas Dann, ETH Zürich, Switzerland, jonas.dann@inf.ethz.ch; Zhenhao He, ETH Zürich, Switzerland, zhenhaoh@nvidia.com; Gustavo Alonso, ETH Zürich, Switzerland, alonso@inf.ethz.ch; Sai Rahul Chalamalasetti, d-Matrix, USA, sairahul@d-matrix.ai; Dejan Milojicic, Hewlett Packard Enterprise, USA, dejan.milojicic@hpe.com; Lance Evans, Hewlett Packard Enterprise, USA, lance.evans@hpe.com; Alex Veprinsky, Hewlett Packard Enterprise, USA, alex.veprinsky@hpe.com; Runbin Shi, Microsoft, USA, runbinshi@microsoft.com.



ACM 1936-7414/2026/1-ART
https://doi.org/10.1145/3799896

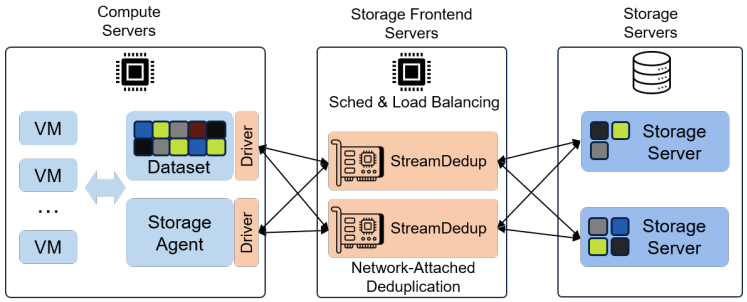


Fig. 1. Multi-tier disaggregated storage system in the cloud with StreamDedup as a network-attached hardware stream processor. (VM: Virtual machine).

1 Introduction

Efficient data reduction techniques, including deduplication and compression, play a crucial role in storage systems, significantly impacting performance and longevity. Microsoft reports that ~ 50% of the data within their primary and secondary storage systems is redundant, creating bottlenecks in storage and network utilization, as well as negatively affecting performance [14, 43]. As a result, efficient, high-performance, system-wide, and inline data deduplication is a crucial strategy to identify and remove duplicate data segments.

Many existing inline deduplication efforts target individual storage nodes or SSDs [8, 10, 16, 18–20, 24, 36, 53, 60, 66, 67]. Such an approach falls short in a disaggregated data center where compute and storage resources operate independently and at a scale far beyond that of a single storage device or node. In multi-tiered environments, storage front-end nodes act as intermediaries between the compute and storage tiers, handling tasks like caching and scheduling (cf. Figure 1). Moreover, this front-end layer may distribute duplicate requests to different storage nodes for load balancing and fault tolerance reasons [5, 6, 15]. Thus, in-SSD or node-level deduplication solutions are unsuitable for cluster-wide deduplication due to their lack of a global view of the entire dataset across various storage nodes [2, 46]. Additionally, recent efforts have highlighted that data deduplication can become compute-bound in multi-threaded CPU environments [17, 38]. To address the computational limitations in CPU solutions, FPGAs [2, 3] and GPUs [4, 26] have been used as accelerators for heavy computations, such as hash computation and compression, with control and table management still handled by CPUs. These approaches incur data movement and control overhead and have scalability issues due to the centralized table management on the CPU. However, low-latency and high-throughput storage deduplication for large storage is not trivial as it presents the following challenges:

(C1) *High Compute Intensity.* Deduplication involves computationally involved tasks, such as hashing during the fingerprinting process to identify duplicates. While node-level or in-SSD deduplication targets the throughput of individual SSDs, system-wide deduplication must handle much higher throughput to a large fleet of storage servers, imposing higher performance requirements.

(C2) *High Memory Intensity.* Deduplication requires heavy random memory accesses to manage metadata of duplicated pages, typically using hash tables, which can become a system bottleneck.

(C3) *High Scalability Requirements.* In a cloud context, the deduplication logic must exhibit excellent scalability in both throughput and management capabilities. This scalability should not

compromise low latency, ensuring the system maintains low latency overhead as it scales to accommodate more storage servers.

Contributions. To address these challenges, we propose StreamDedup, a transparent (i.e., encapsulated so that it is invisible to the user) storage deduplication layer placed in the storage front-end, implemented as a network-attached hardware stream processor. To achieve low latency, StreamDedup operates along the storage I/O path, acting as a bump-in-the-wire accelerator, avoiding unnecessary data movement across PCIe. To address the compute-intensity challenge in (C1), StreamDedup deploys multiple fingerprinting and compression engines for parallel hash generation and page compression that sustain high system-wide throughput. The memory-intensity bottleneck in (C2) is mitigated by dedicated hardware units for parallel, low-latency hash table lookup. StreamDedup is the first system to deploy an in-memory hardware hash table that can be concurrently accessed by a set of lookup engines. To accelerate the fingerprint lookup, StreamDedup also employs a per-bucket Bloom filter to reduce the memory access. To achieve high scalability requirements in (C3), StreamDedup is RDMA-enabled and can be distributed over many nodes. In scaled deployments, StreamDedup builds a distributed hash table across nodes which scales its fingerprint lookup performance and hash table size while keeping a unified view of the data. Lastly, StreamDedup addresses the shortcomings of many storage deduplication systems in related work that cannot handle erase requests by also implementing a complete erase flow.

Key Results. StreamDedup reaches 12.7 GB/s throughput on a single node and scales almost linearly to 125 GB/s on 10 nodes. For write requests, StreamDedup incurs less than 50 µs latency on a single node and less than 60 µs on 10 nodes. For read/erase requests, StreamDedup shows less than 10 µs latency on a single node and less than 16.5 µs latency on 10 nodes. StreamDedup can efficiently deduplicate storage requests with high throughput and low latency overhead at rates comparable to that of commercial systems deployed in the cloud like Microsoft Azure Boost [45].

2 Background

In this section, we introduce disaggregated block storage, storage deduplication, distributed hash tables, and Bloom filters, all of which are used throughout the paper as the basis for the design.

2.1 Disaggregated Block Storage

Storage disaggregation is a widely adopted approach in cloud computing, with prominent examples such as Google Bigtable [7], Amazon's Elastic Block Store [5], Alibaba's Elastic Block Storage [15, 44], and Windows Azure Storage [6]. It separates compute servers from storage servers, which enables independent scaling of both, enhancing cost efficiency and flexibility. In a typical disaggregated block storage deployment, the middle-tier layer is a crucial component, consisting of storage frontend servers handling storage I/O requests and forwarding these to the storage servers [6, 15, 44] (cf. Figure 1). Such middle-tier servers provide maintenance services [28, 68] such as load balancing, caching, replication, fail-over, and snapshot functionality, all critical in a cloud storage system.

In this architecture, storage servers use block storage devices, such as SSDs, to persistently store large volumes of data. Block storage devices store data at the granularity of fixed-size blocks (e.g., 4 KiB), which can be addressed via Logical Block Addresses (LBAs). In SSDs, a firmware component called the Flash Translation Layer (FTL) is responsible for mapping the LBAs of incoming requests to Physical Block Addresses (PBAs) in the flash memory. It also handles wear-leveling, bad block management, and garbage collection, which are critical for maintaining the performance and longevity of the flash memory of SSDs.

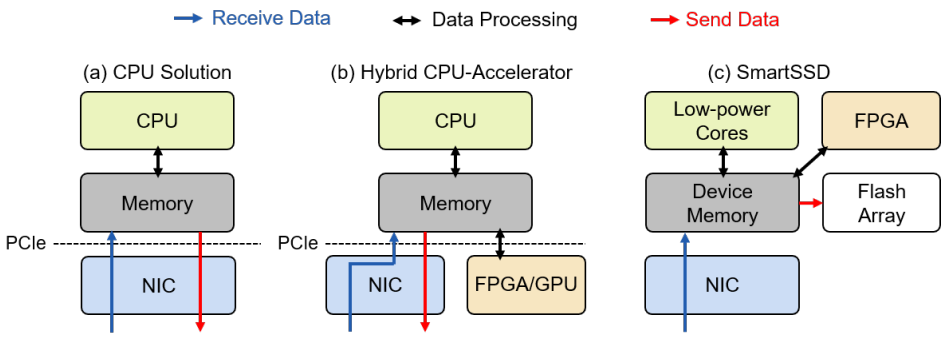


Fig. 2. System architecture of existing solutions.

2.2 Storage Deduplication

In cloud environments with multi-tenant usage of the underlying storage infrastructure, a substantial proportion of the data stored, such as backup copies, virtual machine images, or user files, often contains multiple copies of the same information [14, 43]. Unlike many existing *offline* deduplication approaches [34, 47, 57], which perform data reduction on secondary storage during idle times, our work focuses on *inline* storage deduplication. This method aims to avoid unnecessary write operations to flash memory by only storing duplicate pages once, thereby conserving space and network bandwidth. Additionally, inline storage deduplication helps extend the lifespan of SSDs, ultimately reducing the costs associated with hardware replacement. Such inline deduplication imposes strict latency and throughput constraints, as it is on the critical path of storage I/O.

Storage deduplication can be implemented at file-level or block-level. File-level deduplication [1, 23] compares entire files and is limited to deduplicating identical files, without addressing redundant parts of data within non-identical files. In contrast, block-level deduplication divides files into smaller segments called blocks, which may be fixed-size or content-defined variable size. This method is more effective for identifying redundancies, especially in large datasets with minor variations. In our study, we use fixed-size 4 KiB blocks, in line with the block size of the SSDs and consistent with methodologies from previous work [2, 3, 50].

The deduplication pipeline typically includes fingerprinting, hash table lookup, and management. During fingerprinting, a hash of each block is computed to serve as a unique identifier. In our work, we focus on strong hashing algorithms, such as SHA-3, to generate these fingerprints and minimize the risk of hash collisions [3, 14, 62]. These fingerprints are then stored in a table to track unique data blocks, along with metadata related to the deduplication process. Finally, data compression is typically employed alongside data deduplication to further reduce the volume of data [2, 3], thus, StreamDedup also includes support for data compression.

2.3 Distributed Hash Tables

A hash table is a common data structure used in the deduplication process as an index for fast fingerprint retrieval [17, 38, 47]. Distributed Hash Tables (DHT) are a specialized data structure designed for decentralized and efficient management of key-value pairs across a distributed set of nodes [11, 42, 46, 69]. In a DHT, nodes are typically logically arranged in a ring, with each node handling a specific range of the hash space. This arrangement facilitates quick resolution of the node responsible for a given hash value, a process also known as consistent hashing.

Each node of the DHT is aware of a part of the other nodes in the system. A routing algorithm efficiently routes the incoming request through the network to the corresponding target node for the hash table operation. For example, for the Chord [56] routing algorithm, each node knows the nodes located in $+1, +2, +4, \ldots, +2^{\lfloor \log_2(n) \rfloor}$ positions relative to itself, where n is the total node count. The routing algorithm will send a given request to the largest node in this list that handles hashes that are smaller than the given hash, known as the closest preceding. Other routing algorithms hold nodes on both sides of the local nodes and send the hash to the nearest node in the list. In routing algorithms like Chord, both the routing table size on each node and the request hop count m scales with the logarithm of the total node count n ($m = \log n$). This is crucial in cloud environments where the latency and bandwidth costs need to be minimized.

2.4 Bloom Filters for Storage System

Bloom filters are a space-efficient data structure that tests whether an item is part of a set, trading accuracy for space efficiency by allowing false positives but not false negatives. They are widely used to reduce unnecessary storage access for non-existing data [7, 50] and to index key-value pairs within storage [12, 40]. Bloom filters alone are insufficient for efficient deduplication, as they can only determine if data definitely does not exist or might be duplicated. Moreover, basic Bloom filters in the form of a bitmap cannot support item deletion, as resetting bits from 1 to 0 could compromise correctness due to shared bits. To overcome this, variants such as counting Bloom filters [35, 41, 52, 59] have been developed. These use counters instead of bits, incrementing with insertions and decrementing with deletions. While these modifications allow for the deletion of elements, they increase memory usage — a counting Bloom filter's 16-bit counter per element uses 15× more memory than a basic Bloom filter.

3 Limitations of Existing Approaches & Motivation

The goal is to design low-latency, high-throughput, and scalable solutions for efficient data reduction in disaggregated storage systems. Existing approaches, with architectures shown in Figure 2, cannot address all of these requirements simultaneously, driving the need for new solutions.

Limitations of Acceleration within SSDs. Current deduplication strategies within SSDs either modify firmware within the Flash Translation Layer (FTL) [8, 10, 16, 18–20, 24, 36, 53, 60, 66, 67] or use near-data processing capabilities with integrated smart-processors [13, 25, 33, 37]. However, these approaches have notable drawbacks in disaggregated storage systems. Internal SSD deduplication can miss up to 90% of potential data reduction opportunities because duplicates are spread across multiple SSDs [3, 46]. This distribution occurs as the data set could be too large to be accommodated by a single node and, in multi-tiered systems, storage front-end servers may redistribute duplicate requests to different nodes for load balancing or scheduling [28, 44, 68].

Limitations of Software Approaches. Recent efforts have highlighted that data deduplication can become compute-bound in multi-threaded CPU environments due to heavy computations in hashing [17, 38]. Figure 3 illustrates our analysis of running strong hashing on high-end multi-core CPUs, revealing low scalability. In response to this computational overhead, various efforts have explored deduplication with non-cryptographic fingerprinting techniques as a trade-off for higher throughput [49, 61, 71]. However, these approaches often sacrifice accuracy in identifying duplicated pages, impacting overall deduplication efficiency.

Limitations of Existing Accelerator Approaches. Recent work has investigated offloading compute intensive task, such as page fingerprint generation, to FPGAs [2, 3] and GPUs [4, 26]. Figure 3 shows that fingerprint generation on FPGAs can provide scalable throughput. However, these accelerator solutions introduce data movement and control overheads for the complete

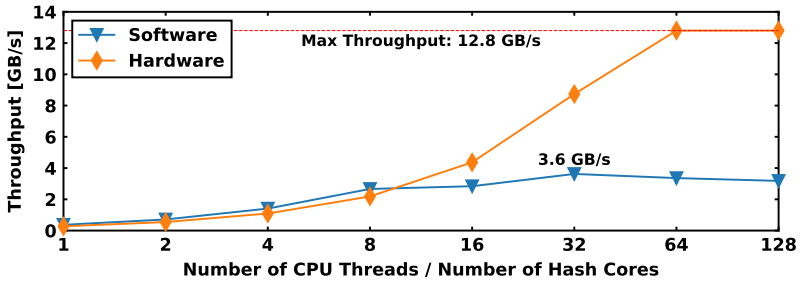


Fig. 3. Throughput of software and hardware solutions for SHA3-256 on 4 KiB pages.

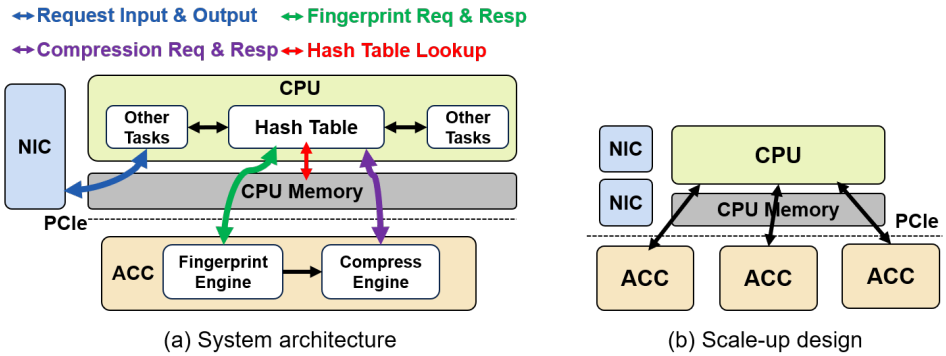


Fig. 4. Existing scale-up architecture [3] for deduplication involves complex data movement and control overheads by the CPU (ACC: Accelerator).

data reduction pipeline due to their system architecture, which increase latency and limit overall scalability. For instance, CIDR [3] offloads fingerprint generation and page compression to hardware while leaving hash table management and other control logic to the CPU host, as shown in Figure 4(a). This setup requires multiple interactions across the PCIe bus, with pages copied to the FPGA for fingerprinting and fingerprints sent back to the host for lookup. The FPGA waits for lookup results, compresses unique pages, and returns them to the host. This design incurs latency overhead and requires complex optimizations on both the hardware and host sides to manage communication overhead. Moreover, scaling out processing capabilities by incorporating more accelerators is constrained by the centralized hash table management, which requires the host CPU and the scale-out accelerators to be on the same PCIe complex, with a limited number of slots, as shown in Figure 4(b). This centralized design can become a bottleneck when serving requests from multiple accelerators, evidenced by evaluations supporting a maximum of only two accelerators. The successor FIDR [2] aims to reduce data movement by enabling direct PCIe peer-to-peer communications between accelerators but still relies on centralized CPU-based hash table management, thus continuing to face latency and scalability limitations as well as consuming valuable CPU cycles. These approaches contrast with those adopted by cloud vendors where storage operations are completely offloaded away from the CPU [45].

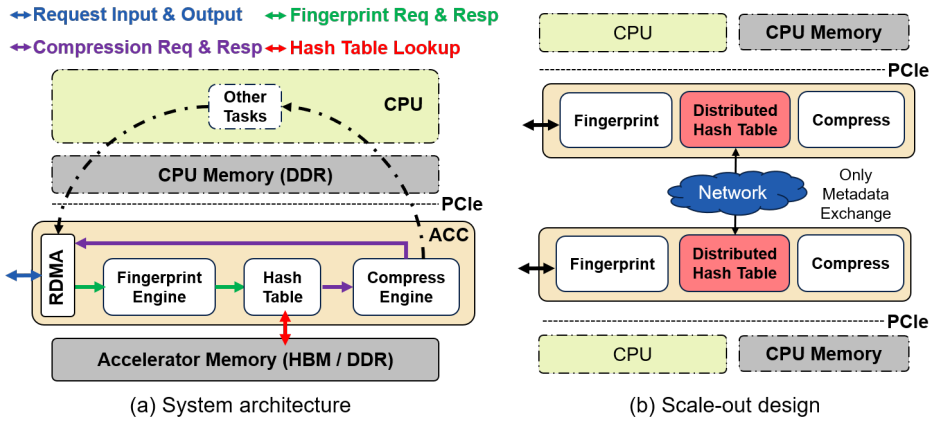


Fig. 5. Scale-out StreamDedup architecture streamlines data paths in accelerators (ACC: Accelerator).

4 StreamDedup Design & Implementation

To address the inefficiencies of existing solutions, we propose StreamDedup. The overall goal of StreamDedup is to achieve *high-throughput, low-latency, and scalable* system-wide data deduplication for a fleet of storage nodes. Figure 5 shows the overall system architecture of StreamDedup.

4.1 System Architecture

Our design is based on the following requirements/goals:

StreamDedup is a stream processor designed to manage all aspects of data deduplication. It employs parallel engines for fingerprint generation, compression, and table lookup, all integrated in the same fabric. StreamDedup is also carefully designed to allow highly pipelined and streamlined processing of consecutive requests. Unlike existing approaches that rely on a CPU for hash table management [2, 3], StreamDedup incorporates an optimized hardware-based hash table implementation with techniques like Bloom filters to enhance lookup performance. A primary advantage of this approach is the optimization of data movement, as it eliminates redundant transfers across PCIe.

StreamDedup is scalable as a distributed accelerator to meet the demands of terabyte-scale storage systems. By deploying multiple StreamDedup accelerators collectively, the processing capacity for tasks such as fingerprint generation and compression can be effectively scaled out. Furthermore, StreamDedup supports a distributed hash table architecture across nodes, which aggregates the lookup capability and memory capacity to form a view of a larger, unified hash table capable of managing substantial storage capacities, as shown in Figure 5(b). This distributed hash table design leverages an RDMA stack for efficient inter-node communication and StreamDedup is optimized to reduce the inter-node communication for low-latency distributed hash table lookups. In contrast to existing methods of scaling processing capabilities [2, 3], StreamDedup does not require the collection of accelerators to be confined to a single PCIe root complex which limits scalablity.

StreamDedup can be easily integrated into existing systems. Unlike in-SSD optimizations [25, 33, 37], StreamDedup does not have to be integrated directly into the underlying SSDs but is instead positioned on the I/O path of the storage frontend layer, serving a fleet of SSD arrays. This placement allows StreamDedup to achieve a higher deduplication ratio by having a global view of I/O requests across backend storage nodes. Additionally, performing deduplication at

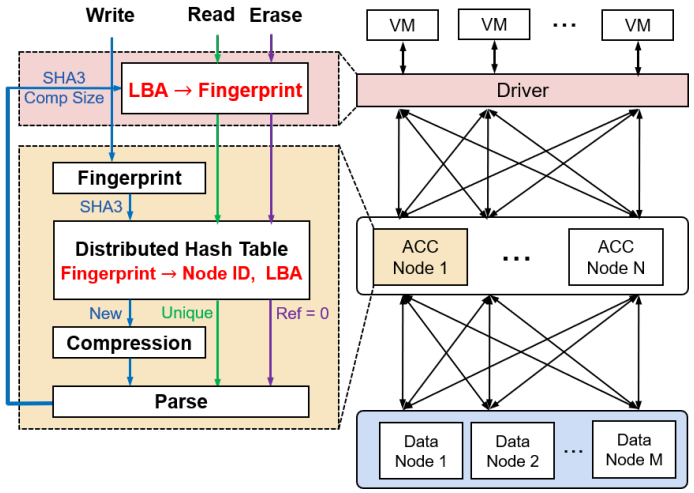


Fig. 6. StreamDedup as a transparent deduplication layer between the compute nodes and storage nodes (VM: Virtual machine; ACC: Accelerator; LBA: Logical block address).

the storage frontend reduces the data volume transmitted over the network, thereby minimizing network traffic. Besides this, StreamDedup is both RDMA-enabled and DMA-enabled, enhancing its flexibility for integration. It can be connected to existing storage frontend servers via PCIe, taking on deduplication tasks previously handled in the request processing pipeline through DMA transfers. Alternatively, StreamDedup can function as an in-network gateway processor, placed prior or after existing frontend servers, directly handling requests from the network and forwarding processed data to the next layer. This approach requires minimal modifications to the existing infrastructure.

4.2 Transparent Deduplication Layer

With StreamDedup, we propose a transparent deduplication layer that can be seamlessly integrated into existing systems with minimal changes to application software and storage hardware, as shown in Figure 6. It consists of a driver running on the host compute nodes, exposing storage APIs to applications, and a dedicated network-attached hardware logic that translates logical block addresses (LBAs) in storage requests to deduplicated LBAs seen by the SSD. Neither the application nor the SSD needs to be aware of the intermediary deduplication logic. This design ensures seamless integration and adaptability within existing storage systems.

For transparency on compute and storage nodes, we use a two-level translation between host-request LBAs and SSD LBAs. The driver on the compute nodes maintains a table mapping each application's LBA to its unique page identifier (fingerprint) and additional metadata. Such tables typically already exist in distributed storage frameworks [5, 15, 44], requiring only the addition of fingerprints. The application LBA to page fingerprint mapping is performed during read and erase operation instead of re-calculating again. The StreamDedup hardware logic contains a hash table which translates the page fingerprint to an SSD LBA.

Write requests, along with their data streams, are directly forwarded to StreamDedup. Within StreamDedup, data is first chunked into fixed-size pages, followed by computing a SHA3-256 hash for each page as their unique identifier. These fingerprints are managed in a hash table that includes metadata like reference counts. When a page is written for the first time, it is assigned a new

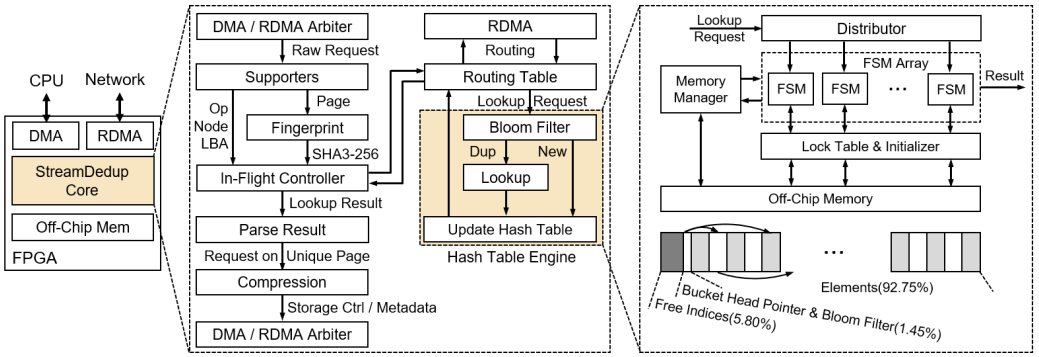


Fig. 7. Overall hardware architecture and memory layout of the StreamDedup core (LBA: Logical block address; FSM: Finite state machine).

entry and a new LBA on the SSD. The page and its write request are then sent to the SSD, and the fingerprint is sent back to the driver to update the mapping table. If a page already exists, we simply update its reference count and avoid sending anything to the SSD.

For read and erase requests, LBAs are translated into page fingerprints by the driver. These fingerprints are sent to the FPGA as part of the request and forwarded to the hash table engine for lookup. Read requests do not change the metadata stored in the hash table and are always forwarded to the SSD to retrieve the page content. For erase requests, the hash table decreases the reference count by one. If the reference count drops to zero, indicating no host references, an erase request is sent to the SSD, facilitating in-line garbage collection.

4.3 StreamDedup Hardware Architecture

We prototype our design on FPGAs and utilize Coyote [31, 51] as the shell[1] to facilitate both DMA and RDMA accesses. The StreamDedup core is dedicated exclusively to deduplication tasks, managing all aspects of the process including data chunking, fingerprinting, compression, metadata management, address translation, and garbage collection. The StreamDedup core can interact with either the local host CPU via DMA or remote nodes through RDMA for request injection and result output, depending on the configuration. Additionally, it supports distributed data deduplication through direct inter-accelerator network communication. Figure 7 illustrates the overall hardware architecture of StreamDedup. In the following sections, we will explore all aspects of the StreamDedup hardware architecture in detail.

4.3.1 Fingerprint and Compression Engines. There are several major components in the StreamDedup core. The supporters are a group of hardware components, including a decoder, arbiters, and request slicing logic. The supporters decode and split the incoming request streams and slice the incoming requests into requests on single pages. The prototype supports the most common commands, i.e., read, write, and erase. Different commands are forwarded to different pipelines and processed in parallel. The write pages are forwarded to the fingerprint engine, which is a set (64×) of SHA3-256 cores [54] deployed in parallel to saturate the throughput of the host or network to make sure the pipeline is not compute-bound. After deduplication, the unique pages are then compressed with a set (6×) of GZip compression cores [65]. A single compression core has a 64 bit wide interface and can sustain 2 GB/s throughput in theory. To make the compression cores work with the rest of the pipeline which is 512 bit wide, we deploy 6 cores in parallel with FIFOs before

[1]An FPGA shell is a set of infrastructure components in hardware providing basic memory and networking abstractions.

and after to convert the interface width and an arbiter that distributes unique pages round robin over the compression cores and collects them in the same order to be forwarded to the DMA / RDMA arbiter. We configured the GZip cores to work on static 4 KiB windows matched to the page size that StreamDedup works on.

4.3.2 Hash Table Engine. Unlike many existing works that only accelerate fingerprint generation and compression [2–4], StreamDedup also incorporates a hardware-based hash table engine, crucial for minimizing unnecessary data movement across the PCIe link. The hash table engine handles parallel fingerprint lookups and metadata management for various storage requests, including read, write, and erase operations. The StreamDedup hash table is structured as an array of buckets, each linked to a list of entries stored in off-chip memory. Figure 7 illustrates the hash table engine's architecture, which features multiple lookup Finite State Machines (FSMs), a memory manager, and lock tables. Additionally, to mitigate the inefficiencies of potentially lengthy linked lists in nearly full hash table buckets, StreamDedup uses a per-bucket Bloom filter to decrease the number of random memory accesses.

Concurrent Lookups with FSMs. Multiple lookup FSMs (the number of FSMs is configurable at compile time) are deployed to handle several lookup requests concurrently, ensuring that the pipeline is not constrained by hash table lookup delays. Each FSM processes one lookup request at a time. Initially, the FSM retrieves the corresponding bucket metadata from off-chip memory and then fetches entries from the linked list while tracking the lookup status. We use the least significant bits of the page fingerprints to determine the bucket index. Each entry, which includes the fingerprint, reference count, and SSD LBA, is padded to 512 bits to match the cache line size in off-chip memory. For existing pages, the FSM adjusts the reference counter in the entry and updates the bucket metadata based on the provided instruction. When a new page is added or the reference counter reaches zero, the FSM will either insert or delete the entry in the linked list. The FSM then returns the lookup results.

Reduce Lookup with Bloom Filter. StreamDedup's hash table design optimizes performance by reducing lookup operations while minimizing memory and hardware costs.

A Bloom filter allows us to bypass linked list lookups when inserting new pages in the average case as a fast path. However, maintaining a Bloom filter for the entire hash table requires extra memory accesses which may lower system performance and standard implementations do not support deletions. Our approach to solve these challenges involves using a per-bucket Bloom filter design with enhanced mechanisms for managing false positives and deletions. Our system employs a 32-bit Bloom filter for each bucket, utilizing three 5-bit hash values derived from the first 15 bits of the fingerprint value. This per-bucket Bloom filter is stored together with the linked list's head pointer in each bucket. This allows the Bloom filter to be transferred together with the bucket's metadata during the hash table lookup process without incurring extra memory access.

When serving requests, if the Bloom filter suggests a page might be present, we perform a linked list lookup. If this results in a false positive, where the page is not found in the list, we reconstruct the Bloom filter. To handle this, our system includes a false positive detection mechanism and prepares a spare Bloom filter during the lookup process. This spare is rebuilt during each linked list lookup and swapped with the original if a false positive is detected. Crucially, this reconstruction occurs in parallel with other operations, ensuring it does not affect overall system performance.

Lock Table. To ensure the correctness of parallel FSM execution, StreamDedup employs a lock table that implements two-phase locking (2PL) for FSM lookup requests. The hardware architecture of the lock table is illustrated in Figure 8. In the linked list lookup scenario, this locking mechanism effectively grants an exclusive lock on an entire bucket for the duration of the lookup process. The FSMs only need to acquire and release the lock once per request. To minimize communication

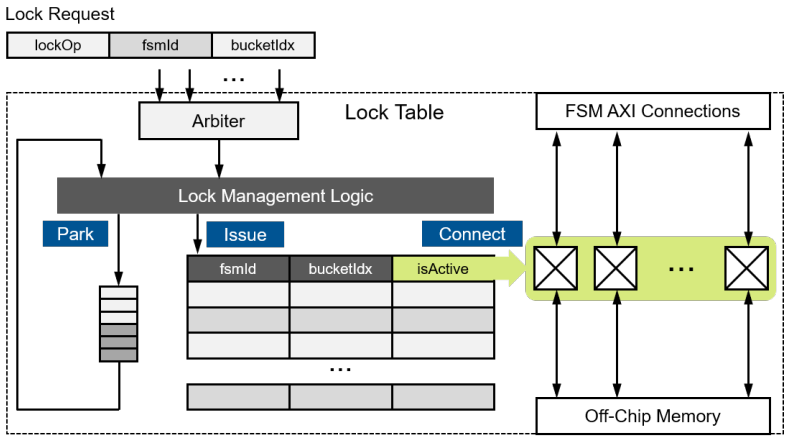


Fig. 8. Lock table architecture (FSM: Finite state machine; AXI: Advanced eXtensible Interface).

overhead between lock acquisition and release, there are no responses from the lock manager back to the FSM. When a request is received, the lock manager determines if the lock can be granted. If another FSM is currently using the bucket, the incoming request is placed in a parking queue. The lock manager then checks each cycle if the request at the top of the parking queue can be served.

Once the lock acquire request is fulfilled, the corresponding entry of each FSM in the lock table is set to record the bucket ID and a flag to indicate that the FSM is holding an active lock. Lock releases are always processed immediately by clearing the corresponding flag in the lock table. The "switch boxes" (indicated by green boxes in Figure 8) between the FSMs and the memory interface permit transactions only when the corresponding FSM holds an active lock. In this design, FSMs notify the lock table when they decide to acquire or release a lock, and the lock table directly controls the FSM's access to external connections to ensure operational correctness.

Memory Manager. To efficiently manage memory space for hash table entries, we deploy a Memory Manager that functions similarly to `malloc` and `free` operations. This system initially stores all empty bucket entry indices in off-chip memory and uses an on-chip buffer as a cache. During an operation, indices are accessed from the on-chip buffer. If the on-chip buffer cannot accommodate further requests to allocate or free up a bucket entry, the Memory Manager then accesses reserved off-chip memory for additional storage or retrieval needs. To optimize performance and mask the latency of accessing off-chip memory, indices are routinely prefetched and stored in the on-chip buffer. StreamDedup maintains a 32-bit index per 512-bit hash table entry. This means the memory manager reserves 6% of the total off-chip memory to store the empty indices. The top 6.25% is used for free indices and hash table metadata (head pointer and Bloom filter), the rest 92.75% of the memory is used to store the hash table entries. For an FPGA with 16 GB memory, 1 GB is used for table-level metadata, and the remaining 15 GB are used to store the hash table entries. In this case, each FPGA can manage metadata for 960 GB of unique pages.

4.3.3 Distributed StreamDedup. StreamDedup aims to enhance scalability in terms of processing capabilities for fingerprint generation and compression, concurrent hash table lookups, and the overall size of manageable storage. While scaling the processing capabilities is straightforward by deploying concurrent accelerators and distributing requests among them, scaling hash table

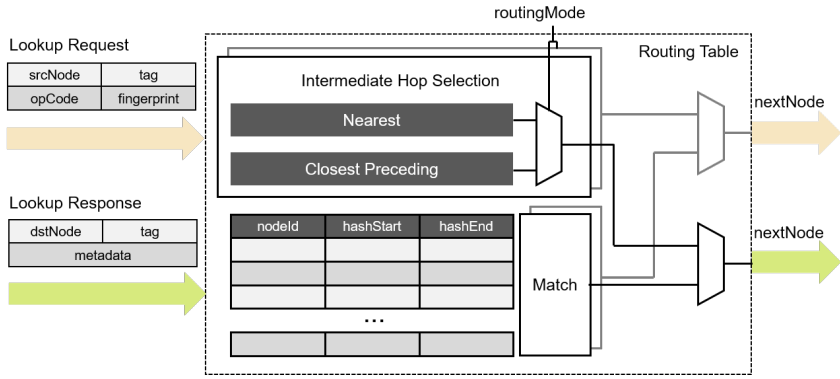


Fig. 9. Routing table architecture.

management consistently across nodes requires a more sophisticated approach. Hence, StreamDedup employs a distributed hash table where the memory space of each node is pooled to form a comprehensive global hash table, enabling efficient local and remote lookups as necessary.

In this setup, when the StreamDedup core receives storage requests, such as write operations, these are processed by the fingerprint engine. Depending on the outcome, StreamDedup determines whether a local or remote lookup is required using a routing table. For remote lookups, StreamDedup leverages an on-chip hardware network stack to facilitate direct connectivity with remote nodes. Critically, instead of transmitting full requests and page data to remote nodes, StreamDedup sends only essential metadata via an remote procedure call (RPC) to trigger a finite state machine at the receiving remote node. This remote node then performs the necessary hash table lookups and updates, subsequently sending back only the updated information. This design reduces data movement which significantly reducing the overhead associated with remote operations.

Routing Table. The routing table is used to find the next node to forward the incoming requests and lookup results both local and from the network. The simplest routing scheme is that each node knows all other nodes and the lookup requests and responses are transferred directly to the target node without intermediate hops. This all-to-all routing scheme reduces the remote lookup latency and network bandwidth usage but poses a significant challenge to meet timing in the placement and routing process when the system scales. The routing table scales linearly to the node count.

As a result, our routing table is designed to be able to handle various advanced routing schemes depending on the users' configuration (Figure 9). In the routing process, we use 16 bits of the SHA3 hash value to decide which node to go to. Each node holds a range in this 16-bit hash, and its routing table stores the node index and the hash value range of itself and a series of remote nodes.

The hash table lookup requests are sent to the routing table to decide where the lookup should be performed. The local requests are first prepended with their source node ID, and then the routing logic decides if the hash value is held by one of the known nodes. If no known nodes have that hash value, the next node will be chosen from the intermediate hop selection logic. By setting the *routingMode* register, the intermediate hop is chosen from the node holding the hash range nearest to the target or the closest preceding node. For example, Chord routing is realized by setting the *routingMode* to high and filling the routing table content by the fingerprint table of each node. To utilize the bidirectional RDMA communication between nodes enabled, we employ a modified version of the Chord algorithm where each node knows nodes located at $\pm 1, \pm 2, \pm 4, \ldots, \pm 2^{\lfloor \log_2(n/2) \rfloor}$,

where n is the total node count. The intermediate hop is the nearest node to the target hash (setting the *routingMode* to low).

The request is forwarded to the remote node's routing table and forwarded to the next node until one node finds that the request should be processed locally. The lookup requests are then sent to the hash table engine to process, and the results are sent back to the routing table with a destination node field that indicates which node to send the response to. This destination node of a response is the same as the source node of a lookup request. The routing table then uses the same strategy on the node index to route the response back. Requests and responses are processed in parallel to maximize throughput and minimize latency.

Concurrent In-flight Remote Lookup. The central part of the StreamDedup in-flight controller is the ring buffer. The requests processed by supporters and SHA3 values from the fingerprint engine are pushed into a ring buffer where they wait for routing and hash table lookup.

As requests are executed in an out-of-order manner, the in-flight controller uses each request's address in the buffer as the unique identifier and keeps it in an extended field called tag throughout the whole routing and lookup process. The logic in the in-flight controller tracks the execution status of all requests in the buffer simultaneously and makes sure the requests are retired in order after the lookup response is written back to the ring buffer. This greatly simplifies the design and debug process since, viewing from outside of the ring buffer, the requests are processed as if the look-ups in the hash table were sequential.

Network Stack. StreamDedup employs an open source RoCE v2-compatible RDMA stack [21] to enable direct network connectivity to the accelerators at 100 Gbps network rate. It supports standard RDMA verbs such as one-sided operations like `WRITE` and two-sided operations like `SEND`.

In the default configuration, the RDMA stack is directly connected to the memory management logic, enabling direct access to off-chip memory locations. We modified this setup so that the StreamDedup core acts as a bump-in-the-wire engine between the memory management logic and the RDMA stack. On the active side, the StreamDedup core issues RDMA commands and prepares the data. On the passive side, it intercepts RDMA network data for further processing. For inter-accelerator network load, we use the two-sided `SEND`.

4.4 StreamDedup Software Stack

The StreamDedup software stack uses a lightweight driver that allows the CPU to issue I/O requests to local and remote FPGAs. The driver maintains an in-memory, per-application LBA-to-fingerprint mapping table. This mapping table optionally also stores metadata such as the compressed size of each page, based on information returned by the FPGA. For each I/O request, the driver translates the request into the format expected by the StreamDedup accelerator. During write operations, the FPGA computes the fingerprint and returns it to the driver, which then updates the mapping table. For read and erase operations, the driver retrieves the corresponding fingerprint from its mapping table and includes it in the request sent to the FPGA. Requests to a locally-attached accelerator are dispatched via DMA, while requests targeting remote accelerators are transmitted over RDMA to leverage direct network access. Once a request arrives at the accelerator, all data processing is executed entirely within the FPGA. The CPU is not involved in the critical path of the deduplication pipeline. This ensures efficient offloading of deduplication operations to StreamDedup, enabling sustained high-throughput and low-latency data processing across both local and distributed environments.

4.5 Discussion: Durability and Recovery

While the current prototype focuses on performance and scalability, durability and recovery mechanisms can be added using established practices in DHT systems. The critical in-memory

data in StreamDedup consists of the driver's LBA-to-fingerprint mappings on the host side and the hash table entries containing reference counts and SSD LBAs on the FPGA. To ensure durability, both components could employ periodic checkpointing and lightweight logging. The driver would periodically persist its mapping table to local non-volatile storage, while the FPGA asynchronously exports snapshots of its hash table metadata to a durable backend using DMA or RDMA. In the event of a failure, recovery begins by restoring the latest checkpoint of both the host and FPGA state, followed by replaying uncommitted I/O records to reach a consistent state. The ordering guarantees required are similar to those in write-ahead logging systems: fingerprint insertions and reference count updates in the FPGA must be durably recorded before the corresponding LBA-to-fingerprint translation is acknowledged to the host driver. For distributed deployments, consistency among nodes can be maintained through lightweight replication of bucket metadata or per-range logs across neighboring nodes, as commonly done in DHT-based systems. These mechanisms provide a practical path to extend StreamDedup with durable metadata and crash-consistent recovery without compromising its low-latency, stream-oriented design.

5 Evaluation

We evaluate StreamDedup performance on a single node, scalability to a cluster of FPGAs, and for real world traces.

Testbed Setup. We run our experiment on a heterogeneous cluster comprising a mix of AMD EPYC CPUs and Alveo U55C FPGAs. Each node contains one CPU and one FPGA connected through PCIe, and all the devices in the cluster are interconnected through a 100 Gbps Cisco Nexus 9336C-FX2 network switch. In the scalability test, we evaluate our design across 10 U55C FPGAs. StreamDedup is deployed with 64 fingerprint cores, 6 compression cores, and 8 lookup FSMs, and is clocked at 200 MHz, unless otherwise specified. The design is synthesized with Vivado 2022.1 in ~4 hours.

CPU (Software) Baseline. For the CPU baseline, we adopt the open-source deduplication software implementation dm-dedup [58] and further optimize it to serve as our CPU baseline. This dm-dedup baseline is the same as that adopted in related work on deduplication with accelerators [2, 3]. The CPU baseline uses a multi-threaded SHA3-256 fingerprint implemented by OpenSSL [48] and a multi-threaded GZip compression algorithm implemented by zlib [39]. The hash table is adapted to user space and keeps the same configuration as dm-dedup and a 10% space overprovision is used to achieve better lookup performance. The number of threads is decided by a grid search to maximize throughput. 32 threads are used for both fingerprint generation and compression in the baseline, and 64 threads for fingerprint generation in the deduplication-only baseline.

Accelerator (Hardware) Baseline. For the FPGA baseline, we adopt CIDR [3], which has a system architecture demonstrated in Figure 4. The CIDR code is not open-source, and we are thus only able to extract throughput for write operations from the paper, while other performance metrics are either not provided or only given as a range. For latency and other metric comparisons, we simulate their performance by maintaining hash table management in the software dm-dedup and offloading only the fingerprint generation and compression to hardware. Our simulated performance, e.g., latency, falls within the range described in their paper.

Workloads. We use synthetic workloads to evaluate our system's throughput and latency. Each workload consists of random requests of the same type (write/erase/read) on all FPGA nodes. We use 32, 768 buckets on each FPGA in the experiment, and all nodes start execution at the same time after they sync with each other. Throughput is obtained by measuring the time used to process 16, 384 pages on each node, and latency is obtained by measuring the average time used to process 16 pages on each node. Our synthetic workloads are summarized here:

- *Write only with various duplication ratios:* All requests are write requests. Some are randomly chosen from the pages that already exist in the hash table, and some are new pages, depending on the duplication ratio.
- *Erase only, all GC:* Pages are randomly erased from the hash table. All pages have a reference count of 1, and are garbage collected (GC) after the erase command.
- *Erase only, no GC:* Pages are randomly erased from the hash table. All pages have a reference count of 2, thus only the reference counter is decremented without sending the erase command to the storage.
- *Read only:* Pages are randomly read from hash table.

Evaluation Methodology. Our evaluation focuses on assessing the latency, throughput, and scalability of our design. Additionally, we assess the performance impact of each component, such as deduplication and compression, on our system and baseline systems. We begin with a single-node evaluation of deduplication alone, where the StreamDedup is configured without the compression cores, followed by an assessment that includes both deduplication and compression. For latency, we measure the additional delay introduced by the StreamDedup accelerators along the storage I/O path. For throughput, we assess the maximum achievable throughput of StreamDedup, assuming that the targeted SSD arrays provide sufficient aggregated bandwidth.

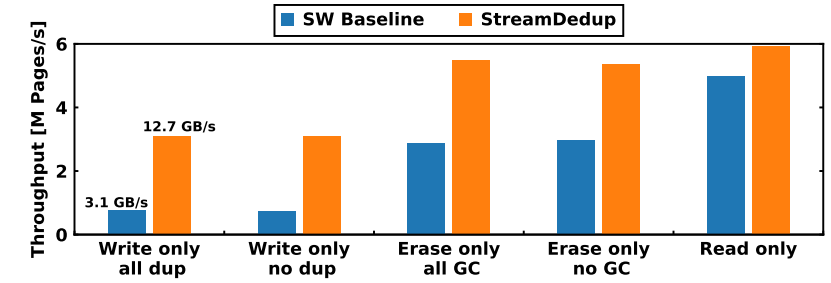
Subsequently, we evaluate our design in a distributed setup. For this, we employ the StreamDedup DMA engine for request injection and output, while using the StreamDedup RDMA stack exclusively for inter-accelerator communication. Such a setting is primarily due to the fact that the FPGA RDMA stack we use can only interoperate with another FPGA RDMA stack, not with commodity NICs. Conducting a fully network-based experiment would require allocating multiple FPGAs solely as load generators thus limiting the scale we could test in the distributed setting. To simulate network bandwidth, we multiplex DMA and RDMA streams and cap the combined throughput at 12.8 GB/s (512-bit data width at 200 MHz), closely matching the 100 Gbps network throughput.

5.1 Deduplication Evaluation

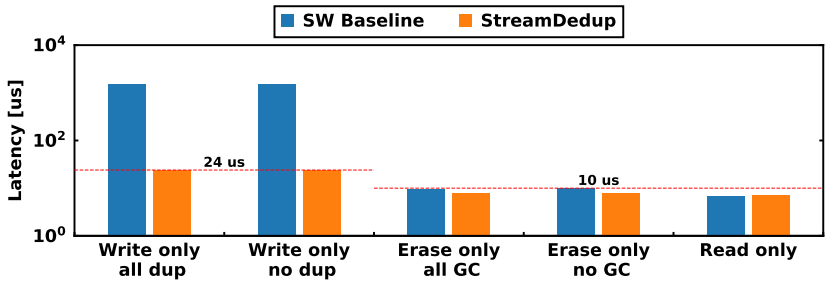
We first evaluate the performance of single-node deduplication under different configurations and workloads.

Throughput. StreamDedup achieves higher throughput for all the workloads when comparing to the CPU baseline due to the parallel and highly pipelined processing of fingerprint and hash table lookup in hardware (cf. Figure 10(a)). For write requests, StreamDedup achieves a throughput of 12.7 GB/s when the hash table is half full, or the duplication ratio is high. This high throughput aligns with our expectations, as the fingerprint engine and hash table engine are specifically designed to match the network bandwidth (100 Gbps) and the DMA engine capacity (12.8 GB/s). In contrast, erase and read requests demonstrate higher throughput in terms of pages processed per second. For erase and read requests, only the hash values (SHA3) need to be transferred to StreamDedup, simplifying the data handling process. Among these, read requests exhibit the highest throughput due to fewer memory accesses required; the lookup FSM does not need to write back any modified hash table elements to memory. Meanwhile, erase requests see slightly higher throughput under all GC conditions compared to no GC workloads because the hash table size reduces after garbage collection, resulting in shorter linked lists for lookups and thus faster processing.

Latency. On the latency side, our system achieves less than 25 µs latency for write workloads. This represents a significant reduction compared to the CPU baseline(64× faster), as shown in Figure 10(b). This improvement is primarily due to the low-latency SHA3 computations performed in hardware, which only require 15.5 µs for a single SHA3 computation. Additionally, hardware allows for efficient and steady multiplexing of input requests across the fingerprinting engines. The



(a) Throughput vs. workload



(b) Latency vs. workload

Fig. 10. Single-node deduplication performance metrics (hash table half full; GC: Garbage collection).

software implementation incurs additional overhead due to context switching in multi-threaded SHA3 computations.

For erase and read workloads, both StreamDedup and the CPU baseline achieve less than 10 µs latency, as these tasks primarily involve hash table access and updates. Despite similar performance in these operations, it is noteworthy that StreamDedup maintains comparable latency while operating at a much lower clock frequency than the CPU.

Hash Table Fullness. Figure 11 shows the write throughput when the hash table is almost full[2]. Comparing this with the write throughput shown in Figure 10(a), we observe that the CPU baseline performance drops significantly when the hash table is almost full, whereas our system still achieves a high throughput of more than 10 GB/s when the hash table is fully occupied and all incoming pages are new (no duplication).

The CPU baseline uses open addressing with linear probing to resolve the hash collisions. This leads to poor insertion performance for new values when the hash table operates near the maximum capacity (even with 10% overprovision) because the probe distance becomes longer and longer. To deliver more consistent performance across different conditions, StreamDedup uses chaining to resolve collisions. The bucket count in StreamDedup's Hash Table Engine is determined by the user-defined parameters, including the total managed space size and average linked list length per bucket, which is 8 by default. The worst-case lookup time can thus be controlled and StreamDedup's worst-case throughput is similar to its max throughput even without the Bloom filter.

[2]Full means > 99% of total capacity is occupied. For write workloads with new pages, the hash table is > 99% full after execution.

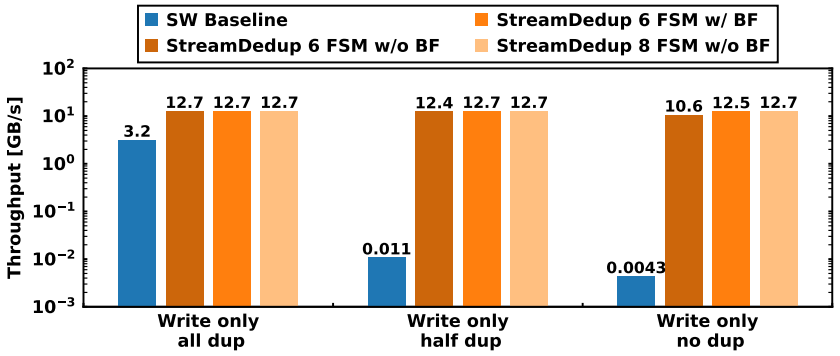


Fig. 11. Single node deduplication throughput vs. Hash Table Engine configuration when the hash table is full (FSM: Finite state machine).

Bloom Filter. To demonstrate the effectiveness of our Bloom filter in identifying new pages and enhancing throughput for write requests, we test different Hash Table Engine configurations for three workloads on a single node (Figure 11).

All configurations achieve a throughput of 12.7 GB/s when all write requests involve duplicated pages. However, as the proportion of new pages in incoming requests increases, more requests proceed to the end of the linked list during the hash table lookup, affecting throughput. For example, the throughput of a configuration with 6 lookup FSMs but without a Bloom filter begins to decline when half of the pages are new, eventually dropping to 10.6 GB/s when there is no duplication among incoming requests.

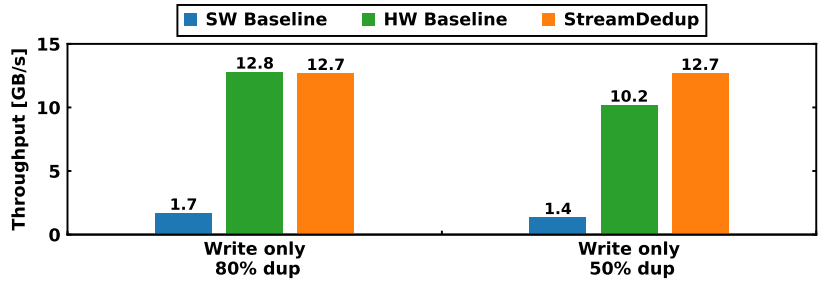
With the help of the Bloom filter, throughput degradation can be mitigated by 10.5× in the worst case and eliminated when less than half of the pages are new. This allows a configuration with 6 lookup FSMs and a Bloom filter to achieve similar throughput as configurations with 8 FSMs. This demonstrates that the Bloom filter design can effectively reduce the need for more FSMs in the Hash Table Engine.

5.2 End-to-end Evaluation

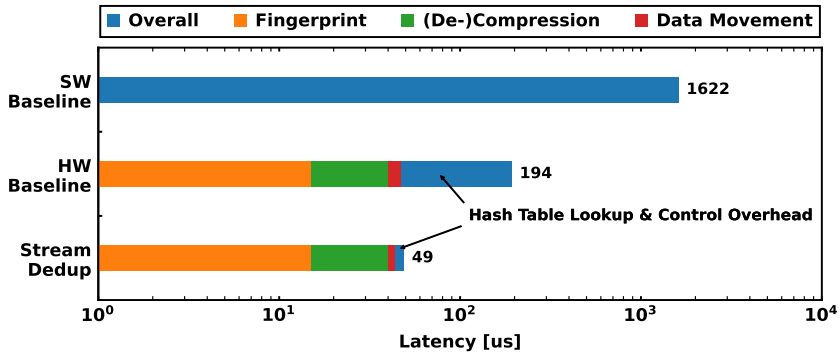
In the following, we evaluate the performance of the end-to-end pipeline including deduplication and compression.

Throughput. Figure 12(a) compares the throughput for write workloads with 80% and 50% duplication ratios among StreamDedup, the CPU baseline, and the hardware baseline. The duplication ratio affects the necessary compression throughput to maintain line-rate, as only non-duplicate pages require compression. Our StreamDedup design achieves stable line-rate throughput in all scenarios, supported by adequate fingerprint and compression engines. Specifically, our deployment of compression cores delivers a total throughput of 11.15 GB/s, preventing any bottleneck in compression.

In contrast, the CPU baseline suffers performance degradation when compression is enabled due to low CPU compression performance and resource contention from multi-threaded processing of fingerprinting and compression tasks. The hardware baseline is limited by its hybrid design, which requires hash table lookups on the CPU and incurs control overhead across PCIe. Additionally, it needs significant on-chip buffering to manage in-flight fingerprint generation while awaiting hash table lookup results. This buffering restricts the number of fingerprint and compression engines due to limited on-chip memory resources, thus constraining throughput.



(a) Write throughput (hash table half full).



(b) Write latency (hash table half full, 50% duplication ratio).

Fig. 12. Latency and throughput of the overall data reduction pipeline with deduplication and compression.

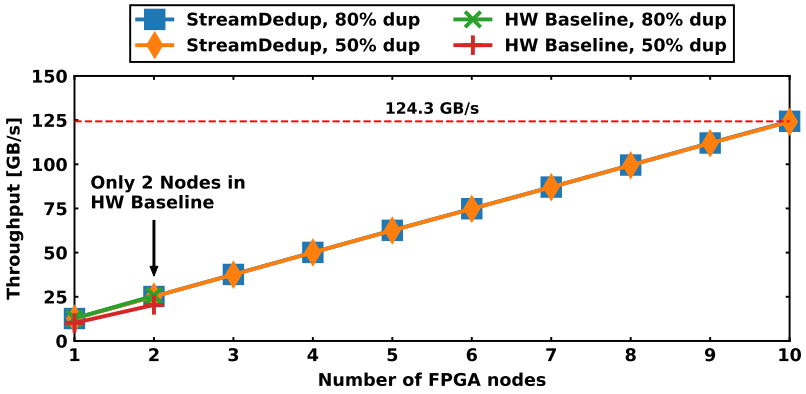
Latency. Figure 12(b) compares the overall latency of write operations between StreamDedup and the baselines. We observe that hardware solutions significantly outperform the CPU baseline, with StreamDedup achieving even lower latency than the hardware baseline. The latency for the hardware baseline is extracted from the paper [3]. This breakdown reveals that the hardware baseline suffers from high latency due to control and hash table lookup processes. A notable cause of increased CPU latency in the hardware baseline is the need for additional optimizations, such as a unique block predictor and opportunistic batching in the CPU software. These techniques, while improving throughput, consume CPU cycles and thus contribute to higher overall latency.

5.3 Scalability

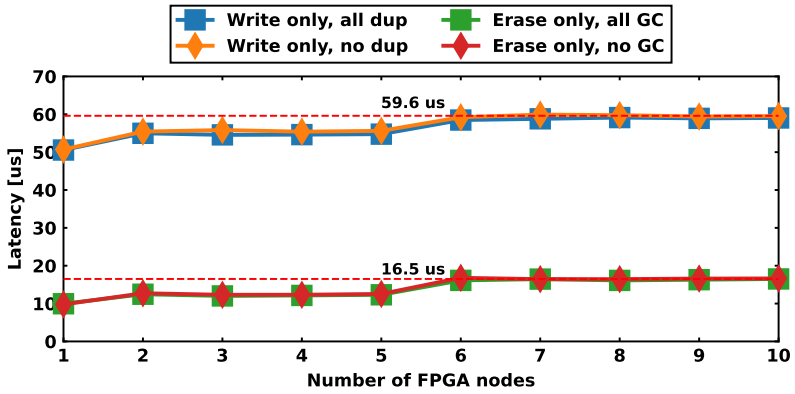
To examine the scalability of StreamDedup, we deploy it on up to 10 nodes and measure the corresponding throughput (Figure 13(a)) and latency (Figure 13(b)).

Throughput. The aggregated throughput of our system scales nearly linearly with increased node count. With 10 nodes, the throughput reaches approximately the theoretical maximum of 125 GB/s. We are able to achieve this due to the combined processing power of the fingerprint engine, compression engine, and hash table engine distributed across multiple nodes. As illustrated in Figure 13(a), the scalability of the hardware baseline is significantly more restricted (only 2 nodes in experiments). Its scalability is limited by the centralized hash table management, requiring all accelerators to be connected to a single PCIe switch root complex.

Latency. StreamDedup shows logarithmic latency scaling with node count. In a 10-node setup, the latency for write requests is smaller than 60 µs, and the latency for erase and read requests is



(a) Aggregated write throughput.



(b) Average write latency.

Fig. 13. Scalability benchmark (hash table full).

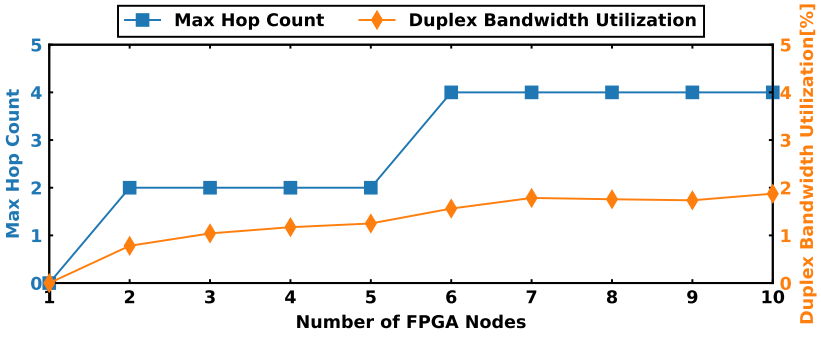


Fig. 14. Inter-accelerator network bandwidth utilization for write operation.

smaller than 16.5 µs. In multi-node cases, the main factor that affects the latency is the network transmission time which scales with the worst-case hop count. In our routing scheme, this scales as the logarithmic of the total node count.

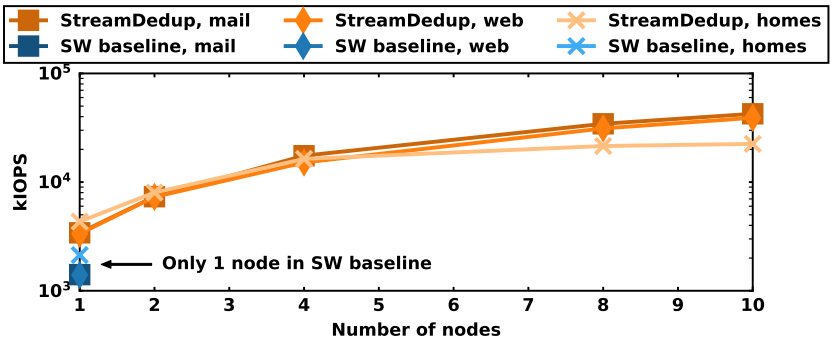


Fig. 15. Real world trace performance.

Network Bandwidth Utilization. The network bandwidth utilized for StreamDedup inter-accelerator communication is minimal, consuming only about 2% of the maximum theoretical network bandwidth of 10 nodes (Figure 14). This efficiency is due to transmitting only a small amount of metadata for remote hash table lookups, without moving the entire requests, including the page itself, through the network.

5.4 Real World Traces

To show StreamDedup performance under a realistic workload, we replay three real world traces that were collected by Florida International University (FIU) [29, 30]. The traces represent the storage I/O of their mail, web, and homes environments over a period of 21 days. Figure 15 shows the performance in thousands of I/O operations (kIOPS) on a logarithmical scale for the software baseline and 1 to 10 nodes running StreamDedup. Each trace is striped over the available nodes until the hash table is full. Since the traces don't capture the environments state at its starting point, reads to pages in the traces that do not have a recorded write in the trace exist. Therefore, we prepare the hash table by initializing these pages before executing the benchmark.

For the software baseline, the performance for the mail and web traces behaves similarly but the system achieves ~ 50% higher throughput for the homes trace. This is because homes pages are 512 Bytes instead of 4 KiB (for mail and web) in size. Thus, the 64 threads used for hashing need to move and process less data. For StreamDedup, we observe linear performance scaling for all three traces. However, we do not observe the same performance improvement as with the software baseline for the homes trace because hashing is not the bottleneck of the system.

5.5 Resource Utilization

The resource utilization of StreamDedup Core is around 40% (cf. Table 1), leaving enough space for other functionality or cope with more network/host bandwidth. The fingerprint engine and compression engine take up more than 90% of the cores resources as they consist of SHA3 and GZip core groups to saturate the input bandwidth.

5.6 Discussion

The primary benefits of StreamDedup manifest in both throughput and bandwidth efficiency rather than just microsecond-level latency. Our single-node prototype sustains line-rate throughput at 12.8 GB/s (512-bit per clock cycle at 200 MHz) and achieves an average end-to-end latency of ~12 µs per write request, which is substantially below typical SSD access latencies of 80–100 µs. Hence, StreamDedup does not limit storage access. Instead, it preserves microsecond-level responsiveness

Table 1. Resource utilization on one Alveo U55C FPGA.

Unit	LUTs	Registers	BRAM
Overall	720.4K (55.2%)	1229K (47.1%)	1255 (62.2%)
RDMA	62.08K (4.76%)	143.9K (5.52%)	48 (2.36%)
StreamDedup	525.2K (40.3%)	848.6K (32.6%)	1022 (50.7%)
Fingerprint (64x SHA3)	296.2K (22.7%)	585.8K (22.5%)	192 (9.52%)
In-Flight Controller	516 (0.04%)	565 (0.02%)	9 (0.45%)
Routing Table	6346 (0.49%)	5344 (0.20%)	0 (0.00%)
HT (6x FSM w/ BF)	13.09K (1.00%)	25.57K (0.98%)	1 (0.05%)
Compression (6x GZip)	204.0K (15.6%)	217.8K (8.35%)	732 (36.3%)

LUTs: Look-up tables; BRAM: Block RAM; HT: Hash table; FSM: Finite state machine

while reducing data movement over PCIe and the network through in-line duplicate detection and compression. The resulting bandwidth savings improve effective link utilization across compute and storage tiers and enable near-constant throughput across duplication ratios, with latency dominated by the device and interconnect rather than the deduplication logic.

Across workloads, throughput remains near line-rate under high duplication ratios and degrades gracefully as the fraction of new pages grows. This stability arises from bounded linked-list lengths and per-bucket Bloom filters, which mitigate collision-induced stalls. In multi-node setups, latency scales logarithmically with node count, primarily reflecting network hop delays, while inter-accelerator traffic remains metadata-bound, consuming only about 2% of available bandwidth.

Although evaluated in an in-network configuration, StreamDedup's architecture generalizes to multiple deployment models. A *CPU-only* system offers software flexibility but suffers from limited throughput and high overheads. A *PCIe-attached accelerator* places deduplication logic inline on the host I/O path, providing a balance between performance and integration cost, while a *PCIe peer-to-peer* model allows direct data exchange with SSDs or NICs, eliminating unnecessary CPU mediation. The *in-network* model used in this work extends these concepts to the cluster level, enabling global deduplication visibility across storage nodes and minimizing host-side data transfers. These configurations form a spectrum between flexibility and efficiency, and StreamDedup's modular interface permits deployment under any of them with minimal modification.

6 Related Work

Storage deduplication has been extensively studied [27, 63]. There are three main directions for storage deduplication inside SSDs, with CPUs, and with accelerators.

Deduplication in SSDs. Various deduplication techniques have been explored within the SSD internal structure. One approach is modifying the firmware within the Flash Translation Layer (FTL) of SSDs which has been thoroughly explored by related work [8, 10, 16, 18–20, 24, 36, 53, 60, 66, 67]. For example, Remap-SSD [70] proposes to remap the duplicated write addresses to existing entries in the address table within the FTL firmware and augment existing SSDs with a small component of non-volatile RAM to solve the data consistency challenges introduced by remapping addresses. More recently, SmartSSDs emerged as conventional SSDs integrated with a low-power processor or FPGA integrated into the existing FTL components to provide near-data processing capabilities [13, 25, 33, 37]. Thus another approach is, to configure the embedded FPGA as an accelerator for hash computation and, together with the embedded core, provide deduplication functionality [9]. However, these techniques working on the storage device itself are limited to intra-SSD deduplication and cannot achieve cluster-wide deduplication across multiple storage

devices. Notably, the approaches in Remap-SSD and SmartSSDs require fundamentally different hardware, necessitating a costly update of the entire storage backend.

Deduplication with CPUs. There is extensive related work on storage deduplication with CPUs [55, 58, 64]. Recent studies have indicated that data deduplication is compute-bound in multi-threaded CPU environments [17, 38]. To address this computational challenge, several approaches have investigated the use of non-cryptographic fingerprinting techniques for deduplication as a compromise to achieve higher throughput [49, 61, 71]. Nevertheless, these methods reduce the accuracy in detecting duplicated pages, negatively affecting overall deduplication efficiency. They also consume valuable CPU cycles and might induce unnecessary data copying, making them less competitive in practice.

Deduplication with Accelerators. CIDR [3] is a storage deduplication system that offloads fingerprint generation and page compression to an FPGA while retaining hash table management and other control logic with the CPU host. This configuration necessitates multiple interactions across the PCIe bus. Pages are transferred to the FPGA for fingerprinting, and the resulting fingerprints are sent back to the host for lookup. The FPGA waits for lookup results, compresses only the unique pages, and sends them back to the host. The successor FIDR [2] aims to mitigate data movement overhead by leveraging direct PCIe peer-to-peer communications between hardware accelerators. However, it is still built on top of centralized CPU-based hash table management, thus continuing to face latency and scalability limitations. Additionally, neither CIDR nor FIDR mentions how erase requests are supported and the workloads only contain write and read requests. STYX [22] offloads memory deduplication (ksm) and compression (zswap) from the operating system kernel to a SmartNIC, i.e., NVIDIA BlueField-2. The pages to be examined are copied to the SmartNIC memory via RDMA and copied back to CPU memory after processing. However, the memory copy between the host and the SmartNIC dominates the overall execution and is a major bottleneck for large data movements. Multes++ [32] implements hardware deduplication logic for key-value pairs in Parquet files on one FPGA. The requests arrive through a 10Gbps network, then the SHA-256 values are computed by 9 parallel SHA cores. An in-memory Cuckoo hash table is used to index the hash values, and each key in the hash table stores the pointer to the keys stored in the NVM.

7 Conclusions

StreamDedup is an innovative FPGA-based deduplication system that relocates the deduplication logic to an independent middleware layer, and leverages FPGA reconfigurability to provide a full hardware accelerated solution. It combines hardware hash calculation, metadata management, and Bloom filter techniques to deliver a high performance system, reducing latency by an order of magnitude compared to traditional software-based methods. This translates to a substantial improvement in data storage efficiency, a critical need in today's data-driven world. Our approach offers a drop-in solution that can be integrated into existing storage systems with minimal effort due to its compatible interfaces, ensuring versatility and adaptability. It also works at line rate for modern networks, easily supporting 100 Gbps transmissions, in contrast to previous work.

References

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*. USENIX Association, 2002.

[2] Mohammadamin Ajdari, Wonsik Lee, Pyeongsu Park, Joonsung Kim, and Jangwoo Kim. FIDR: A scalable storage system for fine-grain inline data reduction with efficient memory handling. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*, pages 239–252. ACM, 2019.

[3] Mohammadamin Ajdari, Pyeongsu Park, Joonsung Kim, Dongup Kwon, and Jangwoo Kim. CIDR: A cost-effective in-line data reduction system for terabit-per-second scale SSD arrays. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*, pages 28–41. IEEE, 2019.

[4] Samer Al-Kiswany, Abdullah Gharaibeh, Elizeu Santos-Neto, George Yuan, and Matei Ripeanu. StoreGPU: Exploiting graphics processing units to accelerate distributed storage systems. In *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC-17 2008), 23-27 June 2008, Boston, MA, USA*, pages 165–174. ACM, 2008.

[5] Amazon. Amazon Elastic Block Store. https://aws.amazon.com/ebs/, 2024. Accessed: 2024-12-08.

[6] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. Windows Azure Storage: A highly available cloud storage service with strong consistency. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 143–157. ACM, 2011.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008.

[8] Feng Chen, Tian Luo, and Xiaodong Zhang. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*, pages 77–90. USENIX, 2011.

[9] Zheng Guo Chen, Nong Xiao, Fang Liu, Yu Xuan Xing, and Zhen Sun. Using FPGA to accelerate deduplication on high-performance SSD. *Materials Science and Intelligent Technologies Applications*, 1042:212–217, 2014.

[10] Zhengguo Chen, Zhiguang Chen, Nong Xiao, and Fang Liu. NF-Dedupe: A novel no-fingerprint deduplication scheme for flash-based SSDs. In *2015 IEEE Symposium on Computers and Communication, ISCC 2015, Larnaca, Cyprus, July 6-9, 2015*, pages 588–594. IEEE, 2015.

[11] Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. Distributed data deduplication. *Proc. VLDB Endow.*, 9(11):864–875, 2016.

[12] Biplob K. Debnath, Sudipta Sengupta, Jin Li, David J. Lilja, and David Hung-Chang Du. BloomFlash: Bloom filter on flash-based storage. In *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*, pages 635–644. IEEE, 2011.

[13] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. Query processing on smart SSDs: Opportunities and challenges. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1221–1230. ACM, 2013.

[14] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Oltean, Jin Li, and Sudipta Sengupta. Primary data deduplication-large scale study and system design. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, page 26, USA, 2012. USENIX Association.

[15] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 519–533. USENIX Association, 2021.

[16] Ramin Gholami Taghizadeh, Reza Gholami Taghizadeh, Fahimeh Khakpash, Mohammadreza Binesh Marvasti, and Seyyed Amir Asghari. CA-Dedupe: content-aware deduplication in SSDs. *The Journal of Supercomputing*, 76(11):8901–8921, 2020.

[17] Fanglu Guo and Petros Efstathopoulos. Building a high-performance deduplication system. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, Portland, OR, June 2011. USENIX Association.

[18] Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Leveraging value locality in optimizing NAND flash-based SSDs. FAST'11, page 7, USA, 2011. USENIX Association.

[19] Jin-Yong Ha, Young-Sik Lee, and Jin-Soo Kim. Deduplication with block-level content-aware chunking for solid state drives (SSDs). In *2013 IEEE 10th International Conference on High Performance Computing and Communications &2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1982–1989, 2013.

[20] Kyuhwa Han, Hyukjoong Kim, and Dongkun Shin. WAL-SSD: Address remapping-based write-ahead-logging solid-state disks. *IEEE Transactions on Computers*, 69(2):260–273, 2020.

[21] Maximilian Jakob Heer, Benjamin Ramhorst, Yu Zhu, Luhao Liu, Zhiyi Hu, Jonas Dann, and Gustavo Alonso. RoCE BALBOA: Service-enhanced data center RDMA for SmartNICs. *CoRR*, abs/2507.20412, 2025.

[22] Houxiang Ji, Mark Mansi, Yan Sun, Yifan Yuan, Jinghan Huang, Reese Kuper, Michael M. Swift, and Nam Sung Kim. STYX: Exploiting SmartNIC capability to reduce datacenter memory tax. In *2023 USENIX Annual Technical Conference*

(USENIX ATC 23), pages 619–633, Boston, MA, July 2023. USENIX Association.

[23] Keren Jin and Ethan L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, New York, NY, USA, 2009. Association for Computing Machinery.

[24] Yanqin Jin, Hung-Wei Tseng, Yannis Papakonstantinou, and Steven Swanson. Improving SSD lifetime with byte-addressable metadata. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '17, page 374–384, New York, NY, USA, 2017. Association for Computing Machinery.

[25] Yangwook Kang, Yang-suk Kee, Ethan L. Miller, and Chanik Park. Enabling cost-effective data processing with smart SSD. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2013.

[26] Chulmin Kim, Ki-Woong Park, and Kyu Ho Park. Ghost: GPGPU-offloaded high performance storage I/O deduplication for primary storage system. In *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '12, page 17–26, New York, NY, USA, 2012. Association for Computing Machinery.

[27] Jonghwa Kim, Choonghyun Lee, Sangyup Lee, Ikjoon Son, Jongmoo Choi, Sungroh Yoon, Hu-ung Lee, Sooyong Kang, Youjip Won, and Jaehyuk Cha. Deduplication in SSDs: Model and quantitative analysis. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2012.

[28] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[29] Ricardo Koller and Raju Rangaswami. FIU IODedup traces (SNIA IOTTA trace 402). In Geoff Kuenning, editor, *SNIA IOTTA Trace Repository*. Storage Networking Industry Association, November 2008.

[30] Ricardo Koller and Raju Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Transactions on Storage (TOS)*, 6(3):1–26, 2010.

[31] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. Do OS abstractions make sense on FPGAs? In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 991–1010. USENIX Association, November 2020.

[32] Lucas Kuhring, Eva Garcia, and Zsolt István. Specialize in Moderation—Building application-aware storage services using FPGAs in the datacenter. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, Renton, WA, July 2019. USENIX Association.

[33] Jaewook Kwak, Sangjin Lee, Kibin Park, Jinwoo Jeong, and Yong Ho Song. Cosmos+ OpenSSD: Rapid prototype for flash storage systems. *ACM Trans. Storage*, 16(3), jul 2020.

[34] Hyungjoon Kwon, Yonghyeon Cho, Awais Khan, Yeohyeon Park, and Youngjae Kim. DENOVA: Deduplication extended nova file system. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1360–1371, 2022.

[35] Minseok Kwon, Vijay Shankar, Salvatore Pontarelli, and Pedro Reviriego. A fingerprint-based bloom filter with deletion capabilities. In *2019 European Conference on Networks and Communications (EuCNC)*, pages 453–458, 2019.

[36] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286, Santa Clara, CA, February 2015. USENIX Association.

[37] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. SmartSSD: FPGA-accelerated near-storage data analytics on SSD. *IEEE Computer Architecture Letters*, 19(2):110–113, 2020.

[38] Chuanyi Liu, Yibo Xue, Dapeng Ju, and Dongsheng Wang. A novel optimization method to improve de-duplication storage system performance. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 228–235, 2009.

[39] Jean loup Gailly and Mark Adler. zlib: A massively spiffy yet delicately unobtrusive compression library. https://github.com/madler/zlib. Accessed: 2024-12-08.

[40] Guanlin Lu, Young Jin Nam, and David H. C. Du. Bloomstore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11, 2012.

[41] Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. Optimizing bloom filter: Challenges, solutions, and comparisons. *IEEE Communications Surveys &Tutorials*, 21(2):1912–1949, 2019.

[42] Shengmei Luo, Guangyan Zhang, Chengwen Wu, Samee U. Khan, and Keqin Li. Boafft: Distributed deduplication for big data storage in the cloud. *IEEE Transactions on Cloud Computing*, 8(4):1199–1211, 2020.

[43] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *9th USENIX Conference on File and Storage Technologies (FAST 11)*, San Jose, CA, February 2011. USENIX Association.

[44] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. From luna to solar: the evolutions of the compute-to-storage networks in Alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022*

Conference, SIGCOMM '22, page 753–766, New York, NY, USA, 2022. Association for Computing Machinery.
[45] Microsoft. Microsoft Azure Boost. https://learn.microsoft.com/en-us/azure/azure-boost/overview, 2024. Accessed: 2024-12-08.
[46] Myoungwon Oh, Sejin Park, Jungyeon Yoon, Sangjae Kim, Kang-won Lee, Sage Weil, Heon Y. Yeom, and Myoungsoo Jung. Design of global data deduplication for a scale-out distributed storage system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1063–1073, 2018.
[47] João Paulo and José Pereira. A survey and classification of storage deduplication systems. *ACM Comput. Surv.*, 47(1), jun 2014.
[48] The OpenSSL Project. OpenSSL: Cryptography and SSL/TLS toolkit. https://github.com/openssl/openssl, 2024. Accessed: 2024-12-08.
[49] Jiansheng Qiu, Yanqi Pan, Wen Xia, Xiaojia Huang, Wenjun Wu, Xiangyu Zou, Shiyi Li, and Yu Hua. Light-Dedup: A light-weight inline deduplication framework for Non-Volatile memory file systems. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 101–116, Boston, MA, July 2023. USENIX Association.
[50] Sean Quinlan and Sean Dorward. Venti: A new approach to archival data storage. In *Conference on File and Storage Technologies (FAST 02)*, Monterey, CA, January 2002. USENIX Association.
[51] Benjamin Ramhorst, Dario Korolija, Maximilian Jakob Heer, Jonas Dann, Luhao Liu, and Gustavo Alonso. Coyote v2: Raising the level of abstraction for data center FPGAs. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSP 2025, Lotte Hotel World, Seoul, Republic of Korea, October 13-16, 2025*, pages 639–654. ACM, 2025.
[52] Christian Esteve Rothenberg, Carlos A. B. Macapuna, Fabio L. Verdi, and Mauricio F. Magalhaes. The deletable bloom filter: A new member of the bloom family, 2010.
[53] Bon-Keun Seo, Seungryoul Maeng, Joonwon Lee, and Euiseong Seo. DRACO: A deduplicating FTL for tangible extra capacity. *IEEE Computer Architecture Letters*, 14(2):123–126, 2015.
[54] SpinalHDL. SpinalCrypto. https://github.com/SpinalHDL/SpinalCrypto, 2024. Accessed: 2024-12-08.
[55] Kiran Srinivasan, Timothy Bisson, Garth R. Goodson, and Kaladhar Voruganti. iDedup: latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, page 24. USENIX Association, 2012.
[56] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA*, pages 149–160. ACM, 2001.
[57] Yan Tang, Jianwei Yin, Shuiguang Deng, and Ying Li. DIODE: Dynamic inline-offline de duplication providing efficient space-saving and read/write performance for primary storage systems. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 481–486, 2016.
[58] Vasily Tarasov, Deepak Kumar Jain, Geoffrey H. Kuenning, Harvey Mudd College, Sonam Mandal, Karthikeyani Palanisami, Philip Shilane, Sagar Trehan, and Erez Zadok. Dmdedup : Device mapper target for data deduplication. 2014.
[59] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys &Tutorials*, 14(1):131–155, 2012.
[60] Yoshihiro Tsuchiya and Takashi Watanabe. DBLK: Deduplication for primary block storage. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5, 2011.
[61] Chundong Wang, Qingsong Wei, Jun Yang, Cheng Chen, Yechao Yang, and Mingdi Xue. NV-Dedup: High-performance inline deduplication for non-volatile memory. *IEEE Transactions on Computers*, 67(5):658–671, 2018.
[62] Avani Wildani, Ethan L. Miller, and Ohad Rodeh. HANDS: A heuristically arranged non-backup in-line deduplication system. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 446–457, 2013.
[63] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016.
[64] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. P-Dedupe: Exploiting parallelism in data deduplication system. In *Seventh IEEE International Conference on Networking, Architecture, and Storage, NAS 2012, Xiamen, China, June 28-30, 2012*, pages 338–347. IEEE Computer Society, 2012.
[65] Xilinx. Vitis_Libraries. https://github.com/Xilinx/Vitis_Libraries, 2024. Accessed: 2024-12-08.
[66] Zhichao Yan, Hong Jiang, Song Jiang, Yujuan Tan, and Hao Luo. SES-Dedup: a case for low-cost ECC-based SSD deduplication. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 292–298, 2019.
[67] Binqi Zhang, Chen Wang, Bing Bing Zhou, and Albert Y. Zomaya. Inline data deduplication for SSD-based distributed storage. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 593–600, 2015.

[68] Jie Zhang, Hongjing Huang, Lingjun Zhu, Shu Ma, Dazhong Rong, Yijun Hou, Mo Sun, Chaojie Gu, Peng Cheng, Chao Shi, and Zeke Wang. SmartDS: Middle-tier-centric SmartNIC enabling application-aware message split for disaggregated block storage. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[69] Yang Zhang, Yongwei Wu, and Guangwen Yang. Droplet: A distributed solution of data deduplication. In *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 114–121, 2012.

[70] You Zhou, Qiulin Wu, Fei Wu, Hong Jiang, Jian Zhou, and Changsheng Xie. Remap-SSD: Safely and efficiently exploiting SSD address remapping to eliminate duplicate writes. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 187–202. USENIX Association, February 2021.

[71] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 442–454, 2018.