CIDR: A Cost-Effective In-line Data Reduction System for Terabit-per-Second Scale SSD Arrays

Mohammadamin Ajdari
Department of Computer Science and Engineering
POSTECH
Pohang, South Korea
Email: majdari@postech.ac.kr

Pyeongsu Park, Joonsung Kim, Dongup Kwon, Jangwoo Kim
Department of Electrical and Computer Engineering
Seoul National University
Seoul, South Korea
Emails: {pyeongsu, joonsung90, dongup, jangwoo}@snu.ac.kr

***Abstract*—An SSD array, a storage system consisting of multiple SSDs per node, has become a design choice to implement a fast primary storage system, and modern storage architects now aim to achieve terabit-per-second scale performance with the next-generation SSD array. To reduce the storage cost and improve the device endurability, such SSD array must employ data reduction schemes (i.e., deduplication, compression), which provide high data reduction capability at minimum costs. However, existing data reduction schemes do not scale with the fast increasing performance of an SSD array, due to inhibitive amount of CPU resources (e.g., in software-based schemes) or low data reduction ratio (e.g., in SSD device wide deduplication) or being cost ineffective to address workload changes in datacenters (e.g., in ASIC-based acceleration).**

In this paper, we propose CIDR, a novel FPGA-based, cost-effective data reduction system for an SSD array to achieve the terabit-per-second scale storage performance. Our key ideas are as follows. First, we decouple data reduction-related computing tasks from the unscalable host CPUs by offloading them to a scalable array of FPGA boards. Second, we employ a centralized, node-wide metadata management scheme to achieve an SSD array-wide, high data reduction. Third, our FPGA-based reconfiguration adapts to different workload patterns by dynamically balancing the amount of software and hardware tasks running on CPUs and FPGAs, respectively. For evaluation, we built our example CIDR prototype achieving up to 12.8 GB/s (0.1 Tbps) on one FPGA. CIDR outperforms the baseline for a write-only workload by up to 2.47x and a mixed read-write workload by an expected 3.2x, respectively. We showed CIDR's scalability to achieve Tbps-scale performance by measuring a two-FPGA CIDR and projecting the performance impacts for more FPGAs.

***Keywords*-deduplication; compression; FPGA; SSD array;**

I. Introduction

The massive amount of 2.5 quintillion bytes of data being generated every day [1], and the trend for massive big data analytics [2], machine learning [3] and virtual desktop infrastructure (VDI) demand a new generation of storage servers to offer massive performance and capacity in each node. While recent servers based on NVDIMM boost performance, the current generation of NVDIMM [4] cannot meet the capacity requirements. Thus companies have implemented high capacity, high performance SSD arrays. For example, SmartIOPS flash appliance based on fast NVMe SSDs [5] provides 8 Tbps and 500 TB of capacity in a single node [6].

Due to the higher cost of SSDs and their lifetime limited by the number of writes, the storage servers need to use effective data reduction techniques. Among such techniques, many commercial storage servers focus on online (a.k.a inline) *deduplication* and *compression* [7], [8], [9]. Deduplication reduces the total data size by eliminating duplicates among data blocks. Compression further reduces the data footprint by eliminating the redundancies within each remaining data block. Both techniques together show significant storage space saving. For example, combined deduplication and compression enable 60% storage saving for typical user data in the cloud [10] and up to 90% storage saving for virtual machines [11].

Unfortunately, the software-based data reduction does not scale with the increasing performance of SSD arrays, requiring an inhibitive amount of CPU resources. We observe that a 24-core server only provides an average of 4.2 GB/s (33.6 Gbps) data reduction throughput. This means that to achieve Tbps-scale, it requires **23 sockets of the high-end 22-core CPU** in a node. Considering that a typical motherboard supports up to 4 CPU sockets, it is hard for existing single-node Tbps systems to run both server management applications and data reduction.

The existing alternatives of hardware-accelerated data reduction suffer from severely low data reduction capability and/or cannot efficiently address the wide-spectrum of workloads. First, we show that in-SSD accelerated data reduction [12], [13] loses many of deduplication opportunities (up to 90% of duplicates with RAID-0 16 SSDs) as the system distributes duplicates to different SSDs. Furthermore, the intra-SSD compression has a seriously limited applicability. For example, the intra-SDD compression becomes ineffective in compressing data pre-encrypted by the host for any reason (e.g., security). Second, accelerations with unmodifiable ASIC-based custom hardware can be cost ineffective due to their inflexibility to address different workload needs. We show that different workloads have different read,

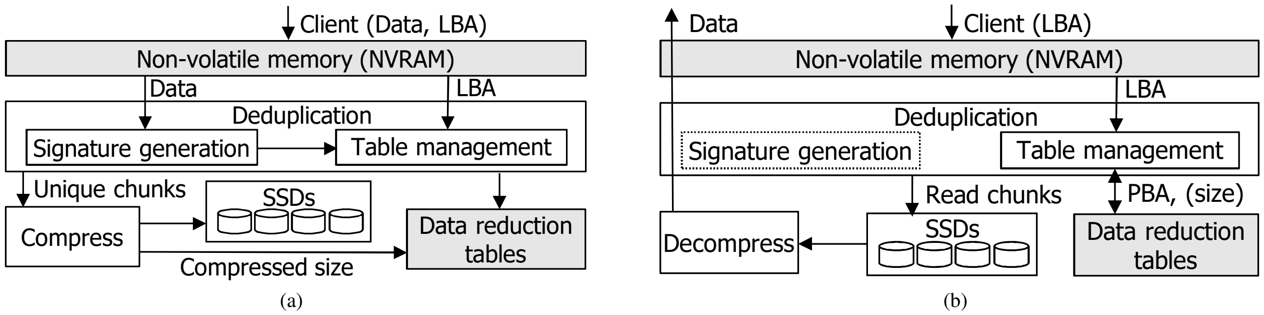


Figure 1. Basic operational steps in a storage server that is capable of data reduction (gray: memory components). (a) write request handling (b) read request handling

write access patterns (e.g., read-write ratio, deduplication ratio, compression ratio) which demand different dynamic acceleration performance. Furthermore, datacenter services frequently evolve over time [14], which enforces tuning the data reduction parameters and algorithms. For example, when a storage server starts to handle a specific type of big data such as genomes, genome-specific compression algorithms [15] can be used for better data reduction.

To overcome the mentioned limitations, we propose CIDR, a novel, **C**ost-effective **I**nline **D**ata **R**eduction system to handle a Tbps-scale SSD array. Our key ideas are as follows. First, we decouple *data reduction-related computing tasks* (i.e., signature generation in deduplication, compression) from the unscalable host CPUs and offload them to a scalable array of FPGA boards. Second, we employ the *centralized, node-wide metadata management* to enable an SSD array-wide, high data deduplication capability. Third, we take advantage of the reconfigurability of an FPGA and the recent ASIC-grade features as technology enablers (e.g., efficient routing, many embedded ASIC logic, large on-chip SRAM cells [16]). We configure and optimize CIDR hardware engines to consume minimal resources to only provide the performance for the average behavior of read-write patterns of a target workload. We then propose additional low-overhead software support modules to efficiently predict the dynamic workload behavior and map it to the static minimal FPGA resources.

We prototyped our design with two FPGA-boards and project the impact for Tbps-scale storage systems with multiple FPGA boards. Our evaluation shows that CIDR outperforms the throughput of the optimized software-based approach by up to 2.47x on write-only workloads. We expect CIDR to be more efficient on mixed half-read, half-write workloads and its throughput improvement reaches 3.2x. In addition, while the performance of the software-based approach is typically limited to four CPU sockets, our scheme scales easily relying on many existing PCIe lanes in a node. Our scheme has less than 14.1% CPU utilization overhead for its software support modules, and a negligible overhead on data reduction ratio.

In summary, we make the following contributions:

- **Problem identification:** We show the major limitations of existing data-reduction approaches to support a Tbps-scale SSD array.
- **A novel FPGA-based architecture:** We propose CIDR, a novel FPGA-based, cost-effective data reduction system for an SSD array to achieve the Tbps-scale storage performance.
- **High performance and scalability:** CIDR outperforms the existing approach by achieving up to almost 100 Gbps on one FPGA and scales with adding more PCIe-based FPGA boards.
- **High data reduction ratio:** CIDR achieves high data-reduction ratio by employing a centralized metadata management scheme and efficiently capturing dynamic workload behaviors.
- **Cost-effectiveness and flexibility:** To minimize the cost of FPGA, CIDR can be reconfigured to dynamically balance the amount of software and hardware tasks running on CPUs and FPGAs, respectively.
- **Prototyping and projection:** We implement a real prototype achieving 160 Gbps performance with two FPGAs, and project our major architecture ideas for 1+ Tbps environment.

II. Background

A. Data Reduction Basics

Inline data reduction techniques are applied to write requests before data are stored in an SSD array (Fig. 1a). First, a storage server buffers the client request data and their *logical block address* (LBA) in non-volatile memory. Then it chunks the buffered data and applies deduplication on them. For this purpose, the storage server calculates a signature for each chunk and searches for the same signature in the *data reduction table*. If the signature is found in the table, the chunk is duplicate and its data will not be written to the SSDs. Then, the table management module only assigns a *physical block address* (PBA) to the LBA of the duplicate chunk to show where the same data are already stored. If the chunk is unique, the storage server also compresses it, assigns a new PBA to it, and updates the data reduction tables to include the new signature and the

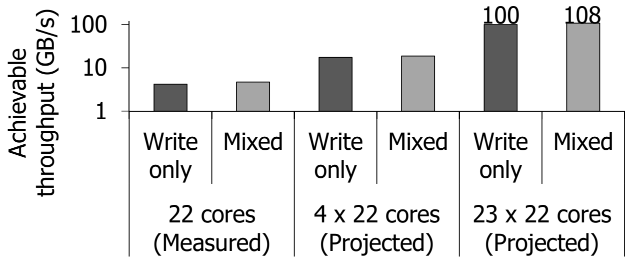


Figure 2. Excessive CPU utilization in the software-based data reduction and its limited throughput scalability

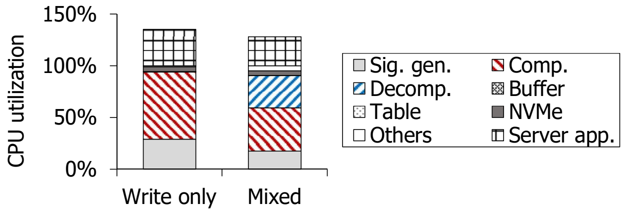


Figure 3. CPU utilization portion of each data reduction step, and the additional CPU-requirement portion for a basic server-side application

LBA-PBA mapping. Lastly, it stores the unique compressed chunk to the SSD array.

Handling read requests requires a maximum of four steps (Fig. 1b). First, the storage server searches requested LBAs from the previously buffered data in the non-volatile memory. If the LBA search results in a buffer hit, the storage server immediately serves the client requested data. If the search results in a buffer miss, it looks up the LBA in the LBA-PBA mapping table. Provided the PBA (and the compressed size), the storage server reads the data from the SSD array, decompresses it and sends it to the client.

B. Basic Design Choices

Non-volatile buffer. Using NVRAM or battery-backed systems to buffer client requests and sending an immediate write completion message is commercially used to hide the backend operation latency from the client, while guaranteeing the data consistency [8], [17].

Data chunking algorithm. Data chunking is categorized into fixed-sized and variable-sized. With fixed-size chunking, all chunks have the same size which simplifies the implementation. Variable-sized chunking increases the duplicate detection capability, but requires computationally demanding operations to detect the chunk boundaries. Both chunking types have been adopted in the industry [8], [18]. In this paper, to ease scalability to Tbps throughputs, we choose the fixed-sized chunking.

Chunk size. Large chunking is usually more efficient than small chunking for many workloads. Table management with large chunks has lower overhead due to accessing a smaller table and less frequently. With 4 KB chunks, data reduction tables are usually hundreds of GBs for the TB-scale capacity data storage [19]. This forces keeping most of the table in a secondary storage which requires performance-sensitive techniques to compensate for it. Large chunks such as 32 KB chunks reduce such overhead by almost an order of magnitude. Real cloud dataset analysis shows that large chunking provides similar data reduction ratio compared to small chunking [10]. Large chunking usually reduces the dedup ratio. However, the compression ratio increases with large chunks which compensates for the reduction of the duplicate detection.

Variety of workloads have large chunk accesses. Real SSD array deployments show that around 74% of data writes have sizes larger than 64 KB [18]. Big data analytics, file-based analytics, and scientific computing workloads are examples with large block writes [20]. In this paper, we assume the 32 KB chunks. However, our architecture is applicable to other chunk sizes.

Signature generation algorithm. Strong hash functions (e.g., SHA-256) are known to have no practical hash collisions in petabytes of data [19]. These functions require only comparing the chunk signatures to determine unique and duplicate chunks. This avoids reading the full data chunks from SSDs and comparing them byte-by-byte. Similar to many existing work [10], [8], [19], we also use strong hashing in this paper.

Compression algorithm. State-of-the-art algorithms such as LZ4 [21] and Intel Gzip from Intel storage acceleration library [22] have been designed considering both compression ratio and throughput. Intel Gzip offers provide better compression ratio and very close throughput to LZ4 [23]. Thus, we use Intel Gzip in the baseline software.

III. Motivation and Design Goals

A. Limitations of SW-based Data Reduction

Software-based data reduction requires an inhibitive number of high-end CPUs to provide Tbps-scale throughput (Fig. 2). Our measurement shows that by saturating 22 CPU cores with optimized deduplication and compression components from Intel ISA-L [22], [24], the CPUs provide only around 4.2 GB/s throughput. The data reduction throughput depends on the data reduction ratio and the ratio of reads and writes. As examples to see the trend, we generated two types of workloads: one with all write requests and 50% deduplication and 50% compression (*Write only*), and another workload with 50% read requests and 50% writes (*Mixed*), which have same amount of deduplication and compression. We conservatively use the linear projection of the throughput assuming high-end CPUs per socket (e.g., a 22-core Intel Xeon E5-4669 v4 [25]). The projection shows an important observation that four 22-core CPUs provide only a fraction of Tbps-scale throughput and the impossible number of 23 sockets in a node is required to meet the desired 100 GB/s of throughput.

Our CPU utilization breakdown shows that three major components of data reduction and general file services compete for CPU resources (Fig. 3). We independently measured

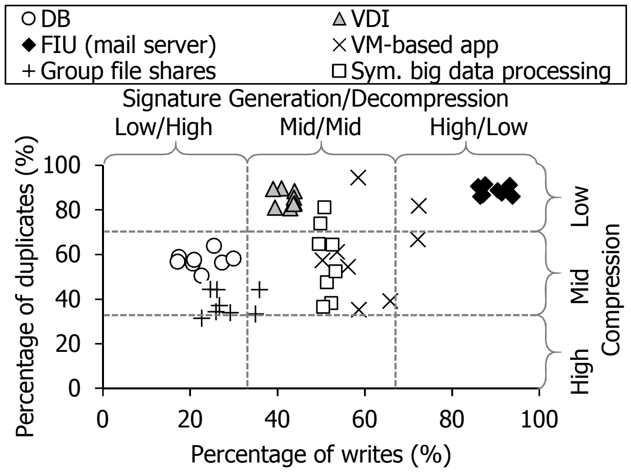


Figure 4. Wide spectrum of workloads with different deduplication, compression, and decompression throughput. The shown workloads are database [26], group file shares [10], VDI [26], VM-based applications [27], [11], [28], symmetric big data processing applications (e.g., Hadoop-based TeraSort), FIU mail server [29].

and added the overhead of data reduction and the network file system (NFS) services. We normalized the CPU utilization of each component to that of the data reduction. On *Write only*, signature generation and compression consume 28.8% and 65.1% of the CPUs, respectively (totally 93.9%). On *Mixed*, decompression is a new overhead with 31.2% utilization. Considering such CPU intensive components, the server cannot even handle general NFS services that demand additional 28%∼35% CPU utilization. Therefore, we choose to offload signature generation, compression, and decompression to a hardware accelerator.

B. Challenges of Hardware Offloading

Efficient coverage of the variety of workloads and environments: Hardware acceleration should satisfy the flexibility requirement to accommodate frequent service changes in a data center (i.e., algorithm and parameter changes) and dynamically adapt to a variety of workload access patterns. Fig. 4 shows the wide spectrum of performance requirements from one workload to another. For example, *database* workloads typically generate only 25% writes (50% of which are duplicates) and 75% reads [26]. This means on average, computational resources should provide decompression at 75% of line-rate throughput, signature generation at 25% and compression and 12.5% of the line-rate. On the other hand, other workloads such as *FIU mail server* have mostly writes (90% of which are duplicates) and only 11% reads [29], [30].

The data center service changes also demand additional change or tuning of the data reduction algorithm. For example, genome-specific compressions (e.g., gdc2 [15]) provide better data reduction than general purpose compressions when the servers mainly handle genome-processing.

Among accelerators, custom-hardware typically provides the best performance per Watt, but, it is challenging to

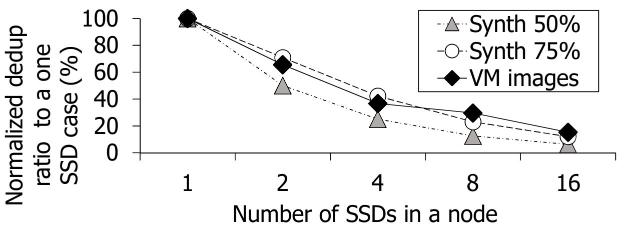


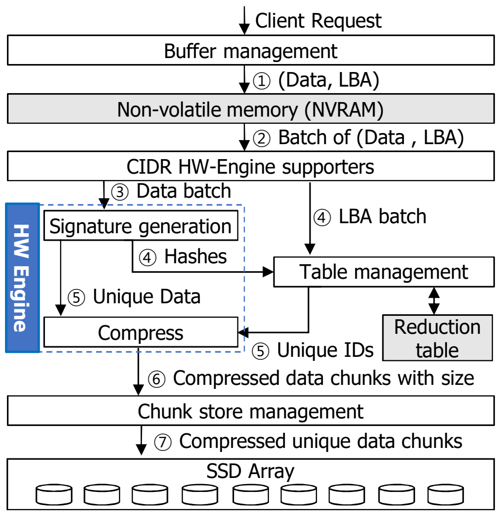
Figure 5. Failure of the scalability of in-SSD accelerated deduplication

provide flexibility and address real-time dynamic workload behaviors. Unmodifiable ASIC-based systems cannot easily adapt to various performance requirements, thus the ASIC designer needs to dedicate lots of hardware resources to overprovision for the worst-case workload requirements. Furthermore, underlying algorithms cannot easily change due to high chip re-manufacturing costs. FPGA is a suitable choice due to its reconfigurable logic fabric. It enables tuning the data reduction logic based on the workload patterns. However, their reconfigurability is usually not fast enough for real-time workload behavior changes. Therefore, with the static FPGA resource allocation, it is still challenging to handle the dynamic real-time variations in a workload.

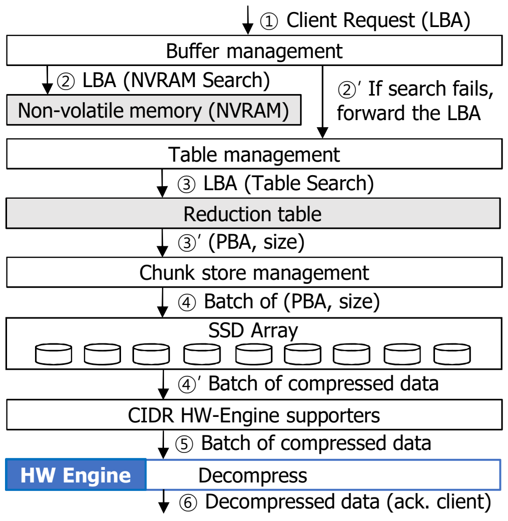
Limitation of acceleration inside an SSD: Offloading the deduplication to each SSD [12], [13], [31] (i.e., in-SSD acceleration) suffers severely from decreasing deduplication opportunities as the number of SSDs in a node increases. Our simulation shows that an array of 16 SSDs set up with RAID-0 configuration misses up to 90% of duplicate opportunities due to the distributed duplicates among SSDs (Fig. 5). We used a real dataset of virtual machine disk images (refer to Section VI) and also modeled synthetic datasets with different dedup-ratios (and random uniform distribution of duplicates in the data). Our used datasets show the consistent trend of degrading deduplication ratio with the increasing number of SSDs.

The in-SSD acceleration of compression [32] also has limited applicability due to its required hardware modification to provide high throughput and also SSDs being the last layer of the IO stack. As an example of low applicability, a storage server with encryption can be considered. As encryption is typically managed by the higher layers on the host, the data encryption happens before sending the data to the SSDs. Such operation randomizes the content of blocks and prevents the in-SSD compression to gain any useful data reduction.

Low-footprint and scalable hardware resources: Designing a scalable Tbps-scale custom hardware accelerator for data reduction is challenging. A naive design results in bottlenecks due to inefficient pipelining or requiring large hardware resources. Tbps-scale requires a large number of signature generation, compression and decompression units on the hardware. Providing data to such units requires careful data network and pipeline designs.



(a)



(b)

Figure 6. Basic version of CIDR system architecture for handling (a) write request (b) read requests.

To ensure scalable implementation, efficient high-throughput pipelining is also critical. The naive approach to use off-chip DRAM to place large pipeline buffers demands many DIMMs and complex DDR controllers. Multiple pipeline stages to handle signature generation and compression make the required bandwidth a multiple of IO line rate. Thus for 1 Tbps line-rate and typical DDR memory providing 0.1 Tbps, multi-Tbps bandwidth pressure on DRAM requires tens of DIMMs and DDR controllers to handle.

C. *Design Goals*

Based on the discussed limitations of the existing work and hardware acceleration challenges, we consider the following design goals:

- **Low CPU utilization.** Minimize CPU utilization by host-decoupled HW acceleration
- **High flexibility.** Provide flexibility to handle datacenter workload evolution and server environment changes efficiently by proposing FPGA-based reconfigurable acceleration
- **High performance and scalability.** Provide high throughput and scalability by using a host- and SSD-decoupled scalable array of FPGAs and by proposing system-wide optimizations
- **Node-wide deduplication.** Guarantee that duplicate detection scales with the number of SSDs in a node by performing a centralized, SSD-decoupled, node-wide data reduction table management.
- **Cost effective.** Maximize overall data reduction ratio cost-effectively by using minimal static FPGA resources based on average workload behaviors and novel techniques to map dynamic workload patterns to static FPGA resources

IV. CIDR: A Cost-effective Inline Data Reduction System

A. *Key Idea*

CIDR consists of three groups of components for high scalability and flexibility. First, an array of PCIe-based, reconfigurable FPGA boards to accelerate high overhead signature generation, compression and decompression (*CIDR HW-Engines*). The second group is the majority of software components that have low-overhead and are exactly same with the baseline (table management, SSD software stack, client request buffer management). Such components are sometimes necessary to run on the host for better data reduction scalability (e.g., table management). Third, we add a set of software modules to support efficient utilization and simplification of the HW-Engines (*CIDR HW-Engine supporters*). The main job of HW-Engine supporters is to minimize dynamic data paths on the HW engines, by predicting the dynamic workload behavior and efficiently scheduling the HW-Engine input data.

B. *Basic System Architecture*

Fig. 6a shows the basic version of CIDR handling client write requests. After buffering enough client requests (LBA, data) in a non-volatile fast memory (①), CIDR HW-Engine supporters make a batch of data chunks (②) and send it to the HW-Engines (③). After signature generation, HW-Engines send the chunk hashes back to the host (④). The table management on the host accesses data reduction tables and determines unique and duplicate chunks. CIDR HW-Engine supporters send the unique chunk IDs to the HW-Engines, so the HW-Engines compress only the unique chunks (⑤). Finally “chunk storage manager” on the

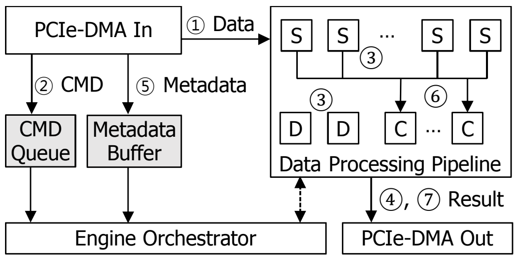


Figure 7. High-level architecture of a single CIDR HW-Engine. (S: Signature Generation, D: Decompress, C: Compress)

host receives the compressed unique chunks from the HW-Engines (⑥), groups and writes them to the SSDs (⑦).

CIDR handles read requests same as the baseline except for the chunk decompression step which is offloaded to the HW-Engines (Fig. 6b). After receiving a client read request LBA (①), CIDR searches the NVRAM buffer (②). If the LBA is not found (②'), CIDR looks up its respective PBA and compressed size mapping in data reduction tables (③, ③'). After collecting a batch of requests to improve SSD throughput utilization (similar to the baseline), CIDR reads the compressed data chunks (④, ④'). Next based on the available number of decompression units on the HW-Engine, HW-Engine supporters make a batch and offload it to the HW-Engines (⑤). Finally the decompressed data are received from the HW-Engines and forwarded to the respective client (⑥).

C. Basic Hardware Microarchitecture

The basic operation of a single CIDR HW-Engine is as follows (Fig. 7). The HW-Engine receives a batch of data chunks through PCIe (①). HW-Engine supporters send a command which goes into the *command queue* (②). After decoding it, the engine orchestrator configures the data processing pipeline to fetch the data chunks into the pipeline (③). When the first stage of the pipeline finishes, the engine orchestrator informs the host to fetch the chunk signatures for the write requests and the decompressed data for read requests (④). At this stage, decompression has fininshed. However, for write requests, after the table management on the host determines unique chunk IDs, they are sent to the metadata buffer on the HW-Engine (⑤). Now the engine orchestrator configures the compression units to fetch the data of unique chunks (⑥). Upon compression completion, the engine orchestrator informs the host to fetch the compressed unique chunks (⑦).

The basic pipelined HW-Engine requires six stages for the data processing pipeline. The first and the last stage are to receive input data from the host and send results to the host, respectively. The intermediate four stages are (1) data buffering for each signature generation unit (2) signature generation (3) buffering for compression units (4) compression. The use of buffering stages before the signature generation and the compression are to minimize the overhead of accessing the data batch buffer on FPGA-board DRAM. Note that the FPGA-board DRAM is the basic choice to handle the large required buffering capacity for parallel processing of many chunks. For example, to achieve 15 GB/s on one FPGA board, 100 independent chunks should be processed at 150 MHz (i.e., over 12.5 MB buffer).

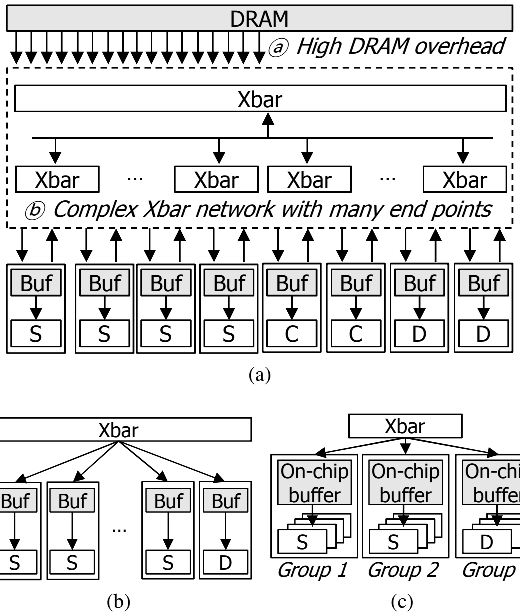


Figure 8. The limitation of the basic pipelined design and the effect of applying first optimization (*balanced on-chip buffering*). (a) Basic pipelined design limitations (b) simplified design before the optimization (c) simplified design after the optimization

D. System optimization 1: Balanced On-chip Buffering

Limitation. The basic pipelined design is not scalable due to heavy centralized buffering and a few required DIMMs (and DDR controllers) on each FPGA-board. The centralized buffering enforces the need for the large distribution crossbars between the DDR controllers and each signature generation, compression and decompression unit (Fig. 8a). Such complexity heavily limits the FPGA implementation for placement and routing, thus limits the design scalability. As explained in Section III-B, the basic pipelining also suffers from the lack of single DIMM bandwidth. Supporting the target pipeline bandwidth requires a multi-DIMM (and multi-DDR controller) design on each board which degrades the scalability and cost-effectiveness.

Solution. To solve the problem of the basic design, we propose an optimization, named *balacned on-chip buffering*. We design it following two key ideas. First, as technology enablers, we use the large SRAM blocks on recent FPGAs [16] to handle large buffering capacity and throughput requirements. Such on-chip memory cells are tens of megabyte and theoretically provide over 1 TB/s bandwidth with distributed units. Second, we target a balance in the granularity of distribution to avoid the overhead of fully distributed buffers and centralized DRAM-like buffering.

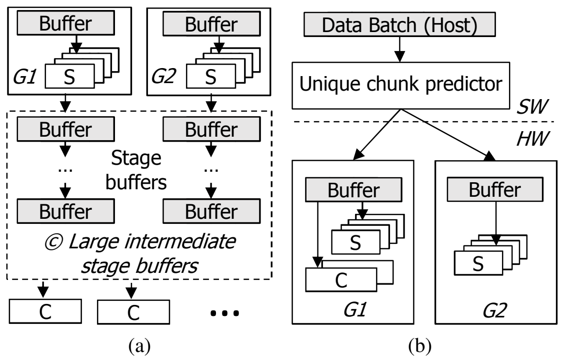


Figure 9. The effect of second optimization (*pre-dedup unique chunk prediction*). (a) before the optimization (b) after the optimization (G1 is for predicted unique chunks)

Both fully distributed buffering and centralized DRAM-like buffering suffer from large data distribution on-chip crossbars and limit scalability. Fully distributed buffering requires a crossbar with many end-points from the PCIe-DMA to the computational units to send/receive data batches. This problem is similar to centralized buffering (large crossbars from the DDR controller to computational units). To overcome such problems, we use physically independent on-chip buffers for *each group of computational units* and add very lightweight arbiters in each group (from Fig. 8b to Fig. 8c).

E. System Optimization 2: Unique Chunk Prediction

Limitation. The basic pipelined design and the design with balanced, on-chip buffering have the requirement to interact with the host table management to determine unique chunk IDs for HW-Engine compression units. While fully on-chip buffering improves this design scalability, it requires additional pipeline stages to ensure no data loss while the host determines unique chunk IDs. Such requirement arises because compression units are supposed to operate on only unique chunks. Therefore, the full batch of data has to stay in the data processing pipeline till the unique ones are selected and fed to compression units. As a result, new pipeline stages and their respective buffering overhead are required between signature generation units and compression units (Fig. 9a). If the host interaction latency for table management and metadata transfers exceeds one pipeline stage latency, more than one pipeline stage should be added the HW-Engines.

Solution. To eliminate the overhead associated with the host and HW-Engine interaction for unique chunk detection, we propose to add a *pre-dedup unique chunk predictor* (Fig. 9b). This new software component predicts which chunks would be unique in a batch before the batch of chunks are sent to the HW-Engines. With little CPU overhead, such prediction information enables integrating the group of compression and signature generation units in the same pipeline stage, thus sharing same buffers. Because compression is only for unique chunks, only selected groups have both signature generation units and compression units. Thus batch making software places the chunks in a batch

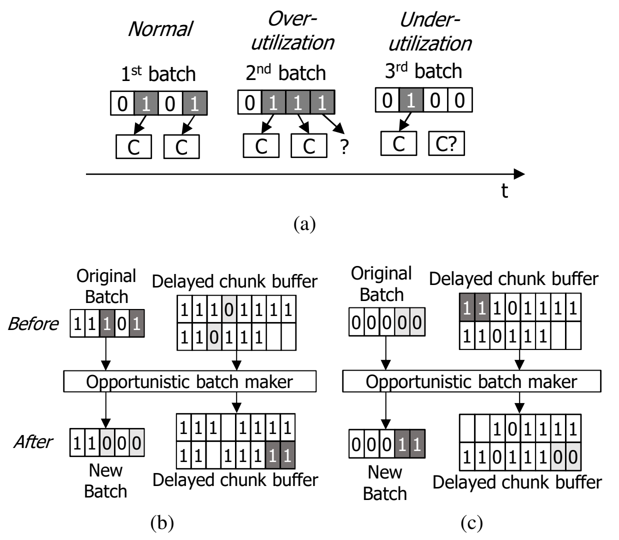


Figure 10. Example for the operation of the *opportunistic batch making* that is present in third optimization (unique: 1, duplicate: 0, C: compression unit) (a) Unique chunk variations between batches (b) Four uniques and 2 compression units ($4 > 2$) (c) All duplicates and 2 compression units ($0 < 2$)

such that the predicted unique chunks are routed to the buffers of the selected groups with compression units.

F. System Optimization 3: Opportunistic Batch Making

Limitation. Previously mentioned optimizations cannot efficiently handle dynamic workload fluctuations on static FPGA computational units (Fig. 10a). In CIDR HW-Engine, the number of computational units is fixed based on the average target workload behavior. For example, the number of compression units is set based on the average required "unique chunk compression". However, as each batch of requests can have different number of unique chunks than average, in some batches, the number of unique chunks exceeds the number of compression units. The easiest way that can handle such scenario sacrifices data reduction ratio by ignoring the excess unique chunks for compression.

Solution. We propose *opportunistic batch making* to significantly mitigate the effect of such fluctuations in a workload and maximize the data reduction ratio (Fig. 10). We observe that the number of required computational units for a workload fluctuates both above and under the average. Thus by collecting the chunks from the "over-utilized" cases and using them in "under-utilized" batches, we can ideally eliminate the dynamic workload problem.

As an example, we demonstrate the cases with two compression units (Fig. 10). To determine the utilization of compression units for a given batch, the batch making software component calculates the number of predicted unique chunks from unique chunk prediction and compares with available HW-Engine compression units. If the predicted unique chunks in that batch are more than the compression units, the batch making excludes the excess unique chunks

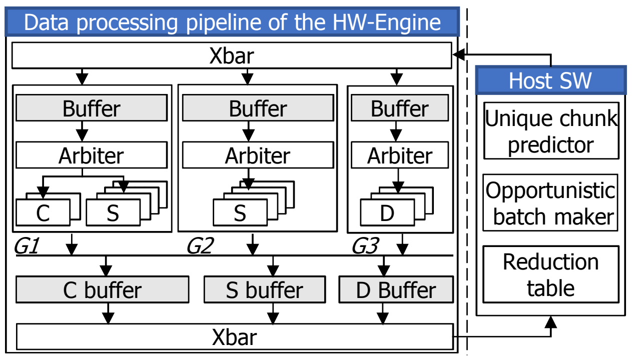


Figure 11. Final CIDR architecture including all of our proposed optimizations. Note: only data processing pipeline on hardware and its required metadata support on software are shown.

from the batch (so delays their processing) and stores them in the *delayed chunk buffer* (Fig. 10b). Then it fetches duplicate chunks from the buffer, if exist, to fill the batch. When the compression units are determined as underutilized (i.e., less number of uniques in the batch), the batch maker swaps some duplicates in that batch with uniques in the delayed chunk buffer (Fig. 10c). In this way, the opportunistic batch making can handle both over- and under-utilization cases effectively.

This delayed batching can happen as long as enough buffer exist for such excess of unique chunks. When the buffer is filled, it no longer delays the batch (it can avoid compression of the chunk or do it on CPU). Fig. 11 shows CIDR after applying aforementioned three optimizations.

G. Single- and Multi-socket Considerations

The recent efforts toward high-bandwidth on-motherboard communication enable CIDR scalability on high-end single socket servers. For example, a high-end single socket AMD EPYC [33] supports 128 GB/s PCIe bandwidth which enables Tbps-scale operation. Next generation (i.e., Gen 4) PCIe potentially doubles this throughput.

To facilitate CIDR scalability on popular multi-socket systems, we also make two design choices. First, we uniformly distribute a group of SSDs, NICs and CIDR HW-Engines across each socket and enable independent data processing on each socket. Second, we ensure high dedup-ratio by centralizing metadata management (i.e., data reduction table accesses and pre-dedup prediction buffer accesses on only a single socket.) For write request handling, only the metadata transfers exist across two sockets (e.g., 32 byte hash value of a 32 KB chunk for the data reduction table management). Such small transfers and the presence of fast inter-socket interconnects (e.g., Intel UPI [34]) make the NUMA overheads negligible. Note that node-wide deduplication makes the read request handling to (sometimes) require fetching data from another socket SSDs. However, we expect such cases to be handled properly because of (1) the read data from the SSDs being in the compressed form, (2) fast intersocket communication (e.g., 60 GB/s with 3 lanes of Intel UPI).

V. Implementation

A. Overview

We implemented the software components of CIDR in the Linux kernel (mostly in the block IO layer) and implemented the HW-Engines on Xilinx VCU1525 FPGA boards. Except the signature generation, compression and decompression which run on CIDR HW-Engines, we reuse the remaining software components (e.g., buffering, table management, NVMe SSD driver, chunk storage management) of the baseline software-based data reduction. In addition to the reused baseline components, we added an FPGA driver to allow communication with the HW-Engine and added two HW-Engine supporters: (a) the pre-dedup unique chunk predictor to simplify the HW-Engine and make CIDR scalable and (b) opportunistic batch maker to send and receive a batch of data between the host and the HW-Engine. In the following sections, we explain the details of the pre-dedup unique chunk predictor implementation, in addition to the HW-Engine details.

B. Pre-dedup Unique Chunk Predictor

CIDR uses a predictor on the CPU to check the uniqueness of every incoming chunk before sending the chunks to the HW-Engines; therefore, the predictor should be lightweight. We use a bloom filter to implement the unique chunk predictor to achieve the high-throughput and lightweight characteristic. The bloom filter is used to test whether an element exists in a set.

To achieve 98% accuracy, and track around 50 TB compressed unique chunks with 32 KB chunks, three billion entries should be used. We implement the chunk predictor (bloom filter) which supports three billion entries. In this configuration, the bloom filter needs six hash functions. We do not execute six different hash functions directly, but use one MD5 hash function and calculate six indices from the MD5 hash value. By using Intel ISA-L for MD5 hashing, the bloom filter achieves 3.5 GB/s per core.

C. HW-Engine Design Considerations

We replicated the signature generation and compression units to provide the desired throughputs, following the target workload characteristics. Due to buffer sharing between signature generation and compression units of a computational group, we set the number of signature generation and compression units in each group to have same throughput. The used compression core (an open-source GZIP core [35]) has the constant throughput of 8 bytes/cycle and each signature generation unit (an open-source SHA-256 core [36]) has 1 byte/cycle. By providing same clock frequency, we grouped each 8 signature generation units with one compression core. To provide data to all computational units, we implemented a simple round-robin arbiter per group.

We carefully selected the buffering memory width and memory type in each group to provide high throughput

Table I
RESOURCE UTILIZATION OF CIDR HW-ENGINE ON ONE VCU9P FPGA

Workload Type (Dedeup ratio, compr ratio)	Write only (50%, -)	Write only (80%, -)	[estimated] Mixed (50%, 50%)	[projected.Gen4 PCIe] write (80%, -)
Aggregate Throughput (GB/s)	10.2	12.8	15.2	25.6
Hash Throughput (GB/s)	10.2 (64 units)	12.8 (80 units)	7.6 (48 units)	25.6 (160 units)
Compression Throughput (GB/s)	5.1 (4 units)	2.5 (2 units)	3.8 (3 units)	5.1 (4 units)
Deompression Throughput (GB/s)	-	-	7.6 (16 units)	-
LUTs	390K (33.00%)	335K (28.34%)	390K (33%)	670K (57%)
Flip Flops	420K (17.76%)	372K (15.72%)	380K (16%)	744K (31%)
on-chip BRAM	1,228 (56.85%)	670 (31.02%)	1100 (50%)	1340 (62%)
on-chip URAM	180 (18.75%)	180 (18.75%)	195 (20%)	360 (38%)
on-chip Power (W)	29.7	28.0	-	-

scalability. We used a new type of on-chip FPGA memory known as URAM. Each URAM is organized in an FPGA as a 288 Kb block, while the older type of memory (also known as BRAM) was organized as 36 Kb. We thus used URAM to satisfy buffering capacity requirements with less replicates. To satisfy throughput requirements (i.e., saturating the x16 PCIe Gen 3 of the VCU1525) we use a few URAM blocks in parallel in each group. For example, with 125 MHz, we used 15 URAM blocks in parallel to provide the required bus width (1024 bits) to reach 16 GB/s throughput.

VI. EVALUATION

A. Environment and Methodology

We evaluated our scheme on a server with dual-socket Intel Xeon E5-2650 v4 CPUs, 256 GB DRAM, two 1 TB Samsung 970 Pro SSDs and two PCIe-attached Xilinx VCU1525 as CIDR HW-Engines. We modified and optimized dm-dedup [30], an open-source deduplication software, to serve as our desired, high performance baseline. We evaluate our scheme on both read and write requests, but we focus specially on the write handling due to their increasing importance in enterprise workloads [27], and their direct impact on the lifetime and the cost of SSDs. We did not have access to any open-source gzip decompression core for the CIDR HW-Engine. Thus, we estimated the HW-Engine resource utilization and its performance to handle read requests using published information of a commercial decompression core [37].

We use two groups of workloads. First, we use Vdbench [38] to generate IO with different desired throughputs, deduplication ratios, and compression ratios. These workloads are suitable to evaluate our scheme throughput, CPU utilization and latency. Second, similar to other data reduction papers, we use a few popular real world datasets and IO traces to analyze dynamic real-world workload behaviors. Such public IO traces usually have small sizes which make them impractical to run on a high-throughput prototype. Furthermore, for privacy reasons, the traces have only encrypted/hashed data content which disables meaningful compression. Thus we run them on our in-house simulator to only evaluate our proposed *opportunistic batch making*. The following are these workloads:

Write-Low: a write-only Vdbench workload with low dedup-ratio and compression ratio (35%, 35%).
Write-Medium: a write-only Vdbench workload with medium dedup-ratio and compression ratio (50%, 50%).
Write-High: a write-only Vdbench workload with high dedup-ratio and high compression ratio (80%, 80%).
Mixed-Medium: a Vdbench workload with the equal number of reads and writes, medium dedup-ratio and medium compression ratio.
VM dataset. A collection of 26 VMware disk images of different linux distributions, total size of 150 GB, 20% dedup-ratio and total deduplication and compression of of 85% at 32 KB chunking.
Kernel sources. a collection of 125 stable Linux kernels of version 3, total size of 68 GB (with 4 KB aligned files), 80% dedup-ratio, and total deduplication and compression ratio of 99%.
Real IO traces. Three-week traces of a mail server [29], [39], and two-week VM traces of a web server [29], [39]. They contain 1.4 TB of writes with 90% dedup-ratio and 28 GB of writes with 43% deduplication at 4 KB chunking respectively.

B. FPGA Implementation and Deployment

Throughput and resource utilization. CIDR provides high throughput and low FPGA resource utilization due to its workload-adaptable and scalable design. Table I shows maximum throughput and resource utilization of CIDR HW-Engine for a few workloads. A single CIDR HW-Engine provides 10.2 GB/s and 12.8 GB/s for the medium and high deduplication workloads respectively. CIDR HW-Engine achieves this throughput with small FPGA resources as low as 28-33% of LUTs and 31-56% of on-chip BRAMs.

We observe that the throughput limitation comes from the PCIe interconnect bandwidth and the used compression core. Thus, we project the throughput of CIDR for the high deduplication workload and assume faster PCIe lanes (i.e., Gen4 PCIe). In this case, we expect the CIDR HW-Engine to achieve up to 25 GB/s with 60% FPGA resource utilization.

Our estimated throughput and resource utilization of the HW-Engine for 50% mixed read-write workloads also show CIDR's scalability. Our estimation reveals that with only

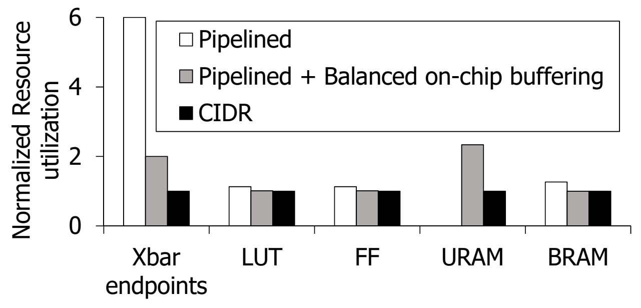


Figure 12. Evaluating our proposed optimizations on CIDR HW-Engine resource utilization.

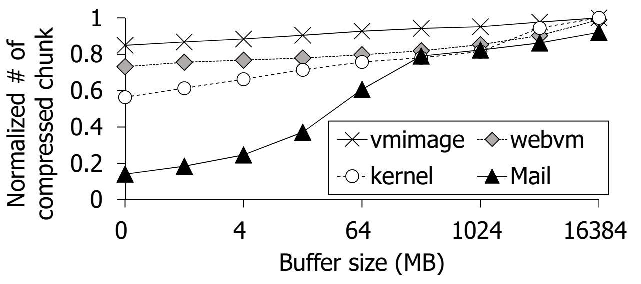


Figure 13. Opportunistic batch making effectiveness vs. the required IO buffering capacity

33% LUT utilization and up to 50% BRAM usage, CIDR is expected to provide 15.2 GB/s aggregate read/write throughput. Considering HW-Engine's low-footprint, higher PCIe bandwidth would further increase HW-Engine's throughput.

Power. The power consumption of CIDR HW-Engines are very low and easily manageable even for 1 Tbps operation. Our conservative simulation using Xilinx Vivado Power Analyzer at maximum temperature of 100°C shows the on-chip power consumption of 28-29.7 W for 0.1 Tbps throughput. Thus, using 5-10 FPGAs to achieve over 1 Tbps performance requires only 140-297 W (similar to one GPU).

C. *Efficacy of System-wide Optimizations*

Our proposed system-wide optimizations proved effective to reduce the HW-Engine footprint and complexity compared to the basic pipelined design. Fig. 12 shows the FPGA resource utilization and on-chip data distribution network complexity (i.e., number of crossbars endpoints) to compare CIDR with two versions of the sub-optimal HW-Engine at 10.2 GB/s. The main benefit of CIDR HW-Engine over the basic pipelined version is that it eliminates DRAM-based buffering, and its associate large on-chip distribution network. Thus CIDR does not require multiple DIMMs and DDR controllers of the basic design and by avoiding the centralized off-chip buffering, it reduces the data distribution network complexity and BRAM usage by 84% and 20% respectively. CIDR also reduces the on-chip buffering requirements (URAM utilization) compared to the basic pipelined + balanced on-chip buffering by as much as 57.1%. This is the effect of eliminating intermediate buffering stage

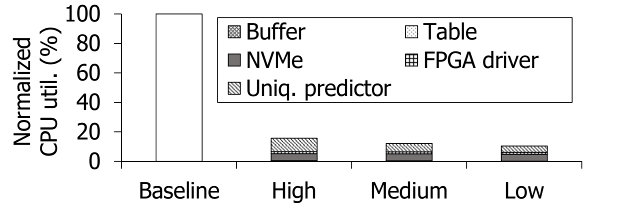


Figure 14. Evaluating the CPU utilization of CIDR on write-only workloads

between signature generation and compression.

CIDR opportunistic batch making can successfully capture the dynamic workload behavior on real workloads, by requiring a small host-side buffer (Fig. 13). Our simulation shows that without opportunistic batch making, the fluctuation of the number of unique chunks in each write batch causes at least loss of unique chunk compression opportunities of 10% in VM and up to 80% in the mail traces, That degrades the data reduction ratio significantly. By using only 1 GB of buffer, our proposed opportunistic batch making captures more than 82% of unique chunk compression opportunities. Higher buffering sizes (e.g., 16 GB) enables opportunistic batch making to capture over 91% unique chunk prediction opportunities. Opportunistic batch making does not need to map 100% of unique chunks to the HW-Engine. If the hardware offloading mitigates the overhead enough, the CPU can also compress the remaining unique chunks. Also note that we conservatively did the simulations by only considering the unique/duplicate status of a chunk and ignored its LBA. In reality, the overwrites (writing to the same LBA) on the buffered requests improves the effectiveness of smaller buffers and strengthens our scheme further.

D. *CPU Utilization*

CIDR hardware offloading reduces the CPU utilization by more than 85% when running at the same throughput with the baseline (Fig. 14). The overhead of the CIDR HW-Engine driver and the HW-Engine supporters (e.g., unique chunk predictor) is totally less than 14.1% CPU utilization for high data reduction and less than 5.4% for low data reduction workload. The 2.5x higher throughput of the high data reduction workload almost proportionally increased the CPU utilization overhead of the unique chunk predictor. Note that other CPU utilization overhead (SSD accesses and table management) is small and similar to the baseline.

E. *Throughput*

We evaluated the throughput of CIDR using three write-only workloads and estimated its performance for a read-mixed workload (Fig. 15). We compared the throughput of CIDR with that of *Baseline* and *High-end*. *High-end* is a CPU-based data reduction system like *Baseline* but adopts a high-end single-socket CPU chip (e.g., Intel Xeon E5-4669 v4 [25]) whose price is similar to the used FPGA. We

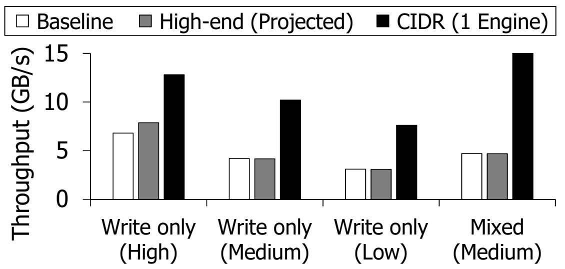


Figure 15. Throughput comparison of CIDR with the projected baseline (High-end)

projected the throughput of *High-end* using the measured *Baseline* throughput and the number of cores assuming the throughput scales linearly. Note that regardless of the workload patterns to access logical addresses, deduplication converts write requests to sequential physical writes and makes the read requests access random physical chunks.

The throughput of CIDR outperforms the software approach on the three write-only workloads. For the low data reduction case (*Low*), the throughput of both the baseline and CIDR is the lowest due to the high compression overhead. As the deduplication opportunity increases from Low to *High*, the compression overhead decreases, so the throughput increases. In short, CIDR shows 1.63x, 2.45x, and 2.47x higher throughput over *High-end* for Write-High, Write-Medium, and Write-Low workloads respectively.

We expect CIDR performance for read-mixed workloads to be even better than write-only workloads. Our estimated throughput of CIDR shows 3.2x higher throughput than the baseline. Such throughput is possible since (1) we eliminate the CPU bottleneck by exploiting larger available parallelism on CIDR HW-Engine, and (2) read request and write request handling require (compressed, uncompressed) batch pair in each host-FPGA transfer direction and balances the use of PCIe bandwidth.

F. Throughput Scalability

By using the *Write-Medium* workload and measuring CIDR throughput with up to two FPGAs, we observed almost linear scalability (Fig. 16). Such scaling is possible because of (1) the independent operation of each HW-Engine, (2) enough PCIe bandwidth to each HW-Engine on independent PCIe slots, and (3) the negligible effect of NUMA inter-socket communication for the dedup table management and the pre-dedup unique chunk predictor.

CIDR shows higher throughput scalability over the high-end baseline with the similar price (Fig. 16). The scalability of *High-end* is limited by both its low per-chip performance and the small number of sockets that a single motherboard supports. In contrast, CIDR provides higher throughput per chip and relies on typical characteristics of high-performance servers (many available PCIe lanes and 2U-3U server form-factor as in [40]). Based on our two FPGA setup, the best we can do to model next generation systems with

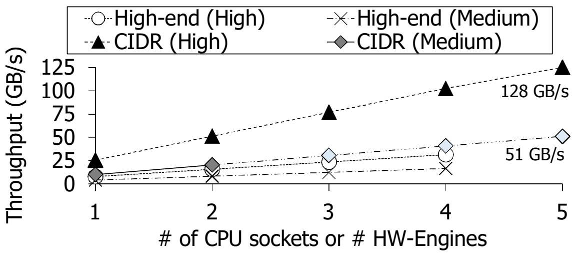


Figure 16. Throughput scalability of CIDR

more FPGAs is using linear projection. Thus we expect that CIDR achieves 40.8 GB/s for *Write-Medium* workload with four HW-Engines, while *High-end* is bounded to 16.6 GB/s. Following Table I and assuming upcoming PCIe Gen-4, the difference of CIDR performance and the baseline becomes more significant. We expect that CIDR achieves over 100 GB/s for *Write-High* with only four FPGAs while the software-counterpart is limited to around 30 GB/s. Note that in extreme cases, contention on DRAM/PCIe may make our projection become sub-linear.

G. Read Request Latency

CIDR provides consistent low read latency at a high throughput environment. We estimated the read request latency of a 32 KB compressed chunk by separately measuring each step involved in read request handling. CIDR read handling requires reading a batch of chunks from SSDs and transferring it to CIDR HW-Engines for decompression. This increases the read latency from 124 us in the baseline to ~318 us in CIDR. However, CIDR still matches the sub-milisecond latency requirement that is the common industry standard [41]

VII. Related Work

A. Improving data reduction software

Optimizing for write throughput. Many existing works on deduplication have assumed hard disks [42], [19], [43], [44], [45], [46], [47]. These works usually use small chunking, which requires a large deduplication table to be in a disk and a small portion cached in DRAM. Thus their goal is to minimize slow disk accesses through improving the hit-rate of the table cache, doing partial deduplication, or using SSD-based table [43]. As an example, Wildani et. al [19] proposed statistical techniques to group IO accesses for better table prefetching and higher DRAM cache hit-rate. Our work is orthogonal, and we can adopt such techniques if the table size becomes very large.

Some studies targeted very high throughput deduplication on software. For example, sampling in deduplication has been shown to enable up to 6 GB/s throughput [44]. Their system is for the backup environment which has the high content locality, and allows sampling with small deduction of the dedup-ratio. However, we target the primary storage which is known to have low locality [10]. Systems such as

P-dedup [48] used software pipelining to improve throughput. Intel ISA-L [22], [24] is another attempt to improve the throughput of data reduction. We integrated such library in our baseline and CIDR outperformed it by up to 2.47x.

Optimizing for read latency. Different techniques have been proposed to mitigate the overhead of data reduction on client reads. El-shimi et. al [10] proposed using deduplication and compression on cold data. iDedup [49] used inline deduplication but only deduplicated sequentially adjacent blocks to increase the sequentiality of reads accesses. Both of such systems were HDD-based and optimized for random reads. However, these problem are not severe with SSDs.

B. Hardware acceleration

Many studies proposed hardware acceleration for general-purpose compute-intensive algorithms, while CIDR efficiently integrates such acceleration cores for a scalable data reduction. For example, FPGA-accelerated variable sized chunking [50] and SHA-256 hashing [36] can be considered. FPGA-based or ASIC-based compression has been also widely studied [51], [52], [53], [54].

A few recent works proposed hardware acceleration for deduplication [55], [56], [57], [9], delta-compression [58], and combined deduplication and compression [59]. Some use a GPU as the accelerator for only variable sized chunking [56] or only SHA1 signature generation [57]. Some use custom hardware (FPGA or ASIC) for data reduction acceleration [55], [9], [58], [59]. As an example,Katayama et. al [59] uses FPGA to accelerate both deduplication and compression. But they target faster WAN communication and allocate FPGA resources for the fixed 50 Gbps rate of signature generation, compression and 25 Gbps of decompression. Overall, such existing works have low target throughputs and do not need to consider the challenges for a Tbps-scale design.

Some recent studies proposed the platforms to accelerate near-storage processing [60], [61], [62]. For example, Nallatech provides FPGA boards with U2-connection in a U2 SSD array [60]. While they have similarities with our scheme in their deployment, Nallatech has not disclosed their architecture details or any optimizations to integrate acceleration of deduplication and compression. BlueDBM[61] is an orthogonal platform with a custom-built flash pool and attached FPGA chips which we can use to improve CIDR performance.

VIII. Discussion

For certain environments, some of the limitations of in-SSD deduplication and compression can be resolved. Such environments may use content-based routing (instead of RAID) to improve dedup-ratio or ignore data encryption to improve compression ratio. Because each SSD provides high throughput in a Tbps-scale system, it has to rely on an FPGA/ASIC for data reduction. Such close coupling of one FPGA to each SSD results in many FPGAs, and also disables system-wide optimizations. Thus host- and SSD-decoupled acceleration in CIDR is a key to scalability.

The open-source hardware compression core that we used was not optimized for the compression ratio. Thus the software (Intel Gzip) provides slightly higher compression ratio for the same data. This is only a limitation of the current prototype. The use of more optimized gzip compression cores such as [51] provides higher compression ratio (even with lower FPGA resource utilization).

Acknowledgment

This work was partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) (NRF-2015M3C4A7065647, NRF-2017R1A2B3011038), Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2015-0-00590, No. 1711073912), and Creative Pioneering Researchers Program through Seoul National University. We also appreciate the support from Samsung Advanced Institute of Technology (SAIT), and Automation and Systems Research Institute (ASRI), Inter-university Semiconductor Research Center (ISRC) at Seoul National University.

IX. Conclusion

We proposed and prototyped CIDR, a cost-effective inline data reduction system to be scalable for Tbps-scale SSD arrays. CIDR FPGA-based engine is flexible to adapt to different algorithms and workload patterns. CIDR achieves scalability by novel techniques to minimize the HW-Engine footprint and map the dynamic workload behavior to minimal static FPGA resources.

References

[1] Jacobson, Ralph, “2.5 quintillion bytes of data created every day.How does CPG and Retail manage it?.” https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/.

[2] Mehta, Yash, “Four big data trends to prepare for 2018.” https://www.kdnuggets.com/2018/01/four-big-data-trends-2018.html.

[3] Moor Insights and Strategy, “Storage-optimized machine learning: Impact of storage on machine learning.” https://www.hitachivantara.com/en-us/pdf/datasheet/vsp-with-internal-nas-modules-datasheet.pdf, 2018.

[4] Alcorn , Paul , “Intel Displays 512GB Optane DC Persistent Memory DIMMs.” https://www.tomshardware.com/news/intel-optane-persistent-memory-dimms,37150.html, 2018.

[5] SmartIOPS, “World’s Fastest SSDs.” http://www.smartiops.com/worlds-fastest-ssds/, Feb. 2018.

[6] SmartIOPS, "Flash Summit 2016 Product Video." http://www.smartiops.com/.

[7] PureStorage, "PureStorage Purity Reduce." https://www.purestorage.com/products/purity/purity-reduce.html.

[8] Deepstorage.net, "Storage efficiency imperative: an in-depth review of storage efficiency technologies and the solidfire approach." http://www.deepstorage.net/NEW/reports/SolidFireStorageEfficiency.pdf, July 2012.

[9] Chris M. Evans, "HPE 3PAR Adaptive Data reduction: A competitive comparison of array-based data reduction." https://www.hpe.com/h20195/v2/getpdf.aspx/4AA6-6256ENW.pdf, Jan 2017.

[10] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, "Primary data deduplication-large scale study and system design.," in *USENIX ATC*, 2012.

[11] C. Constantinescu, J. Glider, and D. Chambliss, "Mixing deduplication and compression on active data sets," in *Data Compression Conference (DCC), 2011*, pp. 393–402, IEEE, 2011.

[12] F. Chen, T. Luo, and X. Zhang, "Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives.," in *FAST*, 2011.

[13] J. Kim, C. Lee, S. Lee, I. Son, J. Choi, S. Yoon, H.-u. Lee, S. Kang, Y. Won, and J. Cha, "Deduplication in ssds: Mo del and quantitative analysis," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pp. 1–12, IEEE, 2012.

[14] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, pp. 13–24, IEEE Press, June 2014.

[15] S. Deorowicz, A. Danek, and M. Niemiec, "Gdc 2: Compression of large collections of genomes," *Scientific reports*, vol. 5, p. 11565, 2015.

[16] Xilinx, "Virtex Ultrascale+ FPGAs." https://www.xilinx.com/support/documentation/product-briefs/virtex-ultrascale-plus-product-brief.pdf.

[17] Hewlett Packard Enterprise, "HPE 3PAR StoreServ Architecture." https://www.hpe.com/h20195/v2/getpdf.aspx/4aa3-3516enw.pdf.

[18] Sergey Zhuravlev, "An analysis of IO size modalities on Pure Storage FlashArrays." https://blog.purestorage.com/an-analysis-of-io-size-modalities-on-pure-storage-flasharrays/, 2016.

[19] A. Wildani, E. L. Miller, and O. Rodeh, "Hands: A heuristically arranged non-backup inline deduplication system," in *ICDE*, 2013.

[20] "I/O type of example applications for IBM Spectrum Scale block size setting." https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.3/com.ibm.spectrum.scale.v4r23.doc/bl1ins_fsblksz.htm.

[21] "LZ4 Compression repository." https://github.com/lz4/lz4.

[22] Intel, "Reposity of Intel Storage Acceleration Library." https://github.com/01org/isa-l.

[23] Intel, "Storage acceleration with I-SAL," 2017.

[24] Intel, "Repository of Intel Storage Acceleration Library for cryptographic hashes." https://github.com/01org/isa-l_crypto.

[25] Intel, "Intel Xeon Processor E5-4669 v4." https://ark.intel.com/products/93805/Intel-Xeon-Processor-E5-4669-v4-55M-Cache-2_20-GHz.

[26] Lydiksen, Lou, "PureStorage Real Workload Models." https://blog.purestorage.com/modeling-io-size-mixes-with-vdbench.

[27] R. Birke, M. Bjoerkqvist, L. Y. Chen, E. Smirni, and T. Engbersen, "(big) data in a virtualized world: volume, velocity, and variety in cloud datacenters," in *Proceedings of the 12th USENIX conference on File and Storage Technologies*, pp. 177–189, USENIX Association, 2014.

[28] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, p. 7, ACM, 2009.

[29] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," in *File and Storage Technologies (FAST), 8th Usenix Conference on*, Usenix, 2010.

[30] V. Tarasov, D. Jain, G. Kuenning, S. Mandal, K. Palanisami, P. Shilane, S. Trehan, and E. Zadok, "Dmdedup: Device mapper target for data deduplication," in *Ottawa Linux Symp.*, 2014.

[31] Z. G. Chen, N. Xiao, F. Liu, Y. X. Xing, and Z. Sun, "Using fpga to accelerate deduplication on high-performance ssd," in *Materials Science and Intelligent Technologies Applications*, vol. 1042, pp. 212–217, Trans Tech Publications, 11 2014.

[32] S. Lee, J. Park, K. Fleming, J. Kim, *et al.*, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Transactions on consumer electronics*, vol. 57, no. 4, 2011.

[33] AMD, "AMD EPYC 7000 series." https://www.amd.com/en/products/epyc-7000-series.

[34] Microway, "Performance characteristics of common transports and buses." https://www.microway.com/knowledge-center-articles/performance-characteristics-of-common-transports-buses/.

[35] Xilinx, "Open-source GZIP hardware core." https://github.com/Xilinx/Applications/tree/master/GZip.

[36] J. Doin, "Open-source SHA-256 hardware core." http://opencores.org/project,sha256_hash_core.

[37] CAST Inc., “ZipAccel Decompression core.” http://www.cast-inc.com/ip-cores/data/zipaccel-d/cast-zipaccel-d-x.pdf.

[38] Oracle, “Vdbench.” https://www.oracle.com/technetwork/server-storage/vdbench-downloads-1901681.html.

[39] Storage Networking Industry Association. IOTTA trace repository, “FIU Traces.” http://iotta.snia.org/, 2008.

[40] PureStorage, “FlashArray//X10. Enterprise-grade all-flash storage for IT generalist.” https://www.purestorage.com/content/dam/purestorage/pdf/datasheets/ps_ds3p_entry-level-storage_03.pdf.

[41] Degrenand, Jean-Luc, “Putting FlashArray//X to the Test with Epic Workloads.” https://www.microway.com/knowledge-center-articles/performance-characteristics-of-common-transports-buses/, 2017.

[42] B. Zhu, K. Li, and R. H. Patterson, “Avoiding the disk bottleneck in the data domain deduplication file system.,” in *Fast*, vol. 8, pp. 1–14, 2008.

[43] B. K. Debnath, S. Sengupta, and J. Li, “Chunkstash: Speeding up inline storage deduplication using flash memory.,” in *USENIX annual technical conference*, pp. 1–16, 2010.

[44] F. Guo and P. Efstathopoulos, “Building a high-performance deduplication system.,” in *USENIX annual technical conference*, 2011.

[45] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, “Sparse indexing: Large scale, inline deduplication using sampling and locality.,” in *Fast*, vol. 9, pp. 111–123, 2009.

[46] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, “Extreme binning: Scalable, parallel deduplication for chunk-based file backup,” in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS’09. IEEE International Symposium on*, pp. 1–9, IEEE, 2009.

[47] W. Xia, H. Jiang, D. Feng, and Y. Hua, “Similarity and locality based indexing for high performance data deduplication,” *IEEE transactions on computers*, vol. 64, no. 4, pp. 1162–1176, 2015.

[48] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Z. Wang, “P-dedupe: Exploiting parallelism in data deduplication system,” in *Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on*, pp. 338–347, IEEE, 2012.

[49] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, “idedup: latency-aware, inline data deduplication for primary storage.,” in *FAST*, 2012.

[50] D. Li, Q. Yang, Q. Wang, C. Guyot, A. Narasimha, D. Vucinic, and Z. Bandic, “A parallel and pipelined architecture for accelerating fingerprint computation in high throughput data storages,” in *FCCM*, 2015.

[51] J. Fowers, J.-Y. Kim, D. Burger, and S. Hauck, “A scalable high-bandwidth architecture for lossless compression on fpgas,” in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pp. 52–59, IEEE, 2015.

[52] M. S. Abdelfattah, A. Hagiescu, and D. Singh, “Gzip on a chip: High performance lossless data compression on fpgas using opencl,” in *Proceedings of the International Workshop on OpenCL 2013 & 2014*, p. 4, ACM, 2014.

[53] Aha product group, “AHA378.” http://www.aha.com/data-compression/.

[54] B. Sukhwani, B. Abali, B. Brezzo, and S. Asaad, “High-throughput, lossless data compresion on fpgas,” in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pp. 113–116, IEEE, 2011.

[55] M. Ajdari, P. Park, D. Kwon, J. Kim, and J. Kim, “A scalable hw-based inline deduplication for ssd arrays,” *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 47–50, 2018.

[56] P. Bhatotia, R. Rodrigues, and A. Verma, “Shredder: Gpu-accelerated incremental storage and computation.,” in *FAST*, 2012.

[57] J. Ma, R. J. Stones, Y. Ma, J. Wang, J. Ren, G. Wang, and X. Liu, “Lazy exact deduplication,” *ACM Transactions on Storage (TOS)*, vol. 13, no. 2, p. 11, 2017.

[58] D. Li, Q. Wang, C. Guyot, A. Narasimha, D. Vucinic, Z. Bandic, and Q. Yang, “Hardware accelerator for similarity based data dedupe,” in *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pp. 224–232, IEEE, 2015.

[59] K. Katayama, H. Matsumura, H. Kameyama, S. Sazawa, and Y. Watanabe, “An fpga-accelerated high-throughput data optimization system for high-speed transfer via wide area network,” in *Field Programmable Technology (ICFPT), 2017 International Conference on*, pp. 211–214, IEEE, 2017.

[60] Nallatech, “Nallatech 250-U2 -Proxy In-Line Accelerator (PIA).” https://www.nallatech.com/store/fpga-accelerated-computing/pcie-accelerator-cards/250-u2/.

[61] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, *et al.*, “Bluedbm: An appliance for big data analytics,” in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 1–13, IEEE, 2015.

[62] IBM, “CAPI Flash Adaptor.” https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.capi/capi_flash_adapter.htm.