ThinDedup: An I/O Deduplication Scheme that Minimizes Efficiency Loss due to Metadata Writes

Fan Ni[†], Xingbo Wu[⊥], Weijun Li[‡], Song Jiang[†]
[†]*University of Texas at Arlington, Arlington, Texas, USA* [⊥]*University of Illinois at Chicago, Chicago, Illionois, USA*
[‡]*Shenzhen Dapu Microelectronics Co. Ltd, Shenzhen, China*
[†]fan.ni@mavs.uta.edu address, [⊥]wuxb@uic.edu, [‡]liweijun@dputech.com, [†]song.jiang@uta.edu

***Abstract*—I/O deduplication is an important technique for saving I/O bandwidth and storage space for storage systems. However, it requires an additional level of address indirection, and consequently needs to maintain corresponding metadata. To meet requirements on data persistency and consistency, the metadata writing is likely to make deduplication operations much *fatter*, in terms of amount of additional writes on the critical I/O path, than one might expect. In this paper we propose to compress the data and insert metadata into data blocks to reduce metadata writes. Assuming that performance-critical data are usually compressible, we can mostly remove separate writes of metadata out of the critical path of servicing users' requests, and make I/O deduplication much *thinner*. Accordingly the scheme is named *ThinDedup*. In addition to metadata insertion, ThinDedup also uses persistency of data fingerprints to evade enforcement of write order between data and metadata. We have implemented ThinDedup in the Linux kernel as a device mapper target to provide block-level deduplication. Experimental results show, compared to existing deduplication schemes, ThinDedup achieves (much) higher (up to 3X) I/O throughput and lower latency (reduced by up to 88%) without compromising data persistency.**

***Index Terms*—Deduplication, compression, flush, consistency**

I. Introduction

I/O Deduplication has been widely used in storage systems for saving storage space [1]–[4]. With data growth at an explosive rate, deduplication plays an important role at various computing environments, including data centers, portable devices, and cyberphysical systems. Deduplication in primary storage has "cascading benefits across all tiers", contributing to reduction of network and I/O loads to other storage tiers and of space demand on all the tiers [5]. In addition, advantages of inline deduplication are also well recognized [3]. It save disk space and disk bandwidth in the first place.

However, even with these clear benefits inline deduplication is rarely deployed for performance-sensitive primary storage systems in production systems [3]. There are two main concerns from the user side. One is degraded read performance due to compromised locality. When data are deduplicated on hard disks (HDDs), sequentiality of data layout can be disrupted, leaving one or even multiple disk seeks during originally sequential reads. This issue has been well addressed by the iDedup scheme by performing selective deduplication to retain data spatial locality [3]. The other concern, which can be more challenging to tackle, is additional writes for persistency of deduplication metadata.

Deduplication systems have to maintain mappings from logical address space exposed to users of the storage system to the physical address space supported by the storage devices[1]. In particular the mapping is from a *logical block number* (LBN) to a *physical block number* (PBN) for block-level deduplication. To service a synchronous write request, the corresponding address mapping must be persisted onto the disk even when the data is deduplicated. In addition to address mappings, there are other metadata whose persistency can be expensive, including mapping between a data block's fingerprint and its physical address, and a data block's reference count indicating number of logical block addresses mapped to it. Furthermore, frequency of updating the metadata is high. In many deduplication systems, out-of-place block writing is used to enable efficient maintenance of consistency between a block's content and its fingerprint [3], [6], [7]. By arranging the data of the out-of-place writes into a log, slow random accesses can be turned into fast sequential ones. These benefits come at the cost of high metadata maintenance cost – every write causes a new LBA to PBA address mapping.

For high persistency and strong consistency of the system, immediate persistency of the metadata is required, which poses significant challenges to use of inline deduplication in the primary storage. First, the metadata are small compared to the data block size. Writing them through the block interface of disks can introduce significant write amplification. Second, many today's applications prefer to quickly persist user data to minimize chances of losing them in a trend where concern on user experience exceeds that on consumption of resources [8]. This may neutralize the effort of collectively writing metadata through batched service of requests for high I/O efficiency and make metadata I/O even more expensive. Third, metadata may be retained in non-volatile memory without being immediately persisted on the disks. However, this requires special hardware support, such as supercapacitor-backed RAM, which may not be available. It is desired that a general-purpose solution does not assume availability of such supports while still achieving similar performance and persistency. Fourth, to maintain crash consistency between metadata and data, one has to pay extra

[1]In this context the physical address is distinct from the one internal to the storage devices. It refers to a logical address in the linear address space exposed by the device(s)

cost to apply approaches such as journaling, shadowing-based atomic write, or flush-ordered writes. These approaches incur expensive disk flush operations and/or additional writes.

In this work we propose a deduplication scheme, named *ThinDedup*, to address the challenges without requiring any special hardware supports. The idea is to compress the data in data blocks to make room for holding (critical) metadata. Because the metadata are hidden in the data blocks, they can be immediately persisted with data without concerns of amplified write cost, increased latency, and frequent use of flushes. To further reduce use of flushes for keeping data and metadata order, we always store address mapping and signature about the same data block together to enable non-ordered writes of data and metadata. We note that in the workloads of an online primary storage it is more common to have data blocks that can be compressed [9]. In addition, as we will show, to be effective ThinDedup only needs a small percentage of data blocks to be slightly compressible.

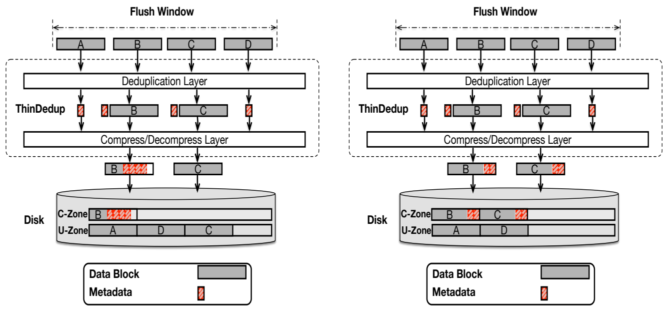
In summary, we made three contributions: 1) We address the metadata issue by hiding only critical metadata in compressed data blocks, and collocating two types of metadata, address mapping and corresponding fingerprint, to minimize use of flushes. 2) We design ThinDedup that can take advantage of compressibility of data blocks with minimally added compression cost and leaving only a relatively small percentage of data blocks compressed. 3) We extensively evaluate ThinDedup and the results show that ThinDedup can provide up to 3× throughput compared to state-of-the-art deduplication schemes. Meanwhile, the latency can be reduced by up to 88% without compromising the throughput.

II. The design of ThinDedup

While insertion of metadata into compressed data blocks is an appealing idea, there are a number of challenges to address to make it truly effective. First, compression consumes CPU cycles, and reading compressed data requires decompression operations. While these costs are negligible compared to the access times of hard disks, they might become a performance issue for faster devices, such as SSDs. The design should ensure that overhead of data block compression and decompression is sufficiently low compared to the I/O time even for SSDs. Second, to reduce (de)compression cost only necessary data blocks are compressed. With compressed and uncompressed data blocks co-existing on the disk, they should be able to be identified with minimal metadata support and overhead, and be managed efficiently. Third, when a data block is deduplicated or is incompressible, its metadata cannot be inserted into the block itself. The design has to make sure that success of metadata insertion does not strongly rely on small deduplication ratio or large percentage of compressible blocks.

A. Window-based Metadata Persistence

In a deduplication system, there are two critical issues that must be addressed in its design, which are persistency and consistency. For persistency, a synchronous write request is acknowledged only when the data is persisted on the disk.



(a) Case I (b) Case II

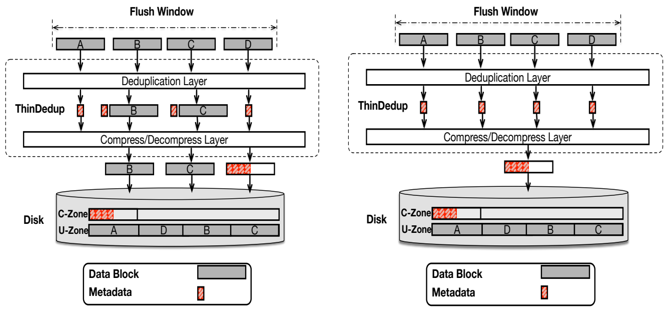
Fig. 1: Cases of serving write requests in a flush window in ThinDedup where explicit metadata persisting is avoided. There are two cases based on the deduplication ratio and the compressibility of blocks in a window. In the figure, four data blocks A, B, C, and D in the same window are issued to ThinDedup system.

For consistency, with a loss of in-memory data structure, possibly due to system crash, at any time, all data and metadata on the disk must remain consistent to each other. While immediate persistency and consistency can be too expensive to achieve, we use window-based batch persistency and fingerprint-assisted consistency in ThinDedup.

In ThinDedup, all incoming write requests are buffered in memory until the total number of data blocks reaches a predefined number threshold or the interval since the first request arrives in the buffer reaches a time threshold. The window size is defined by either of the thresholds, whichever reaches first. At the end of a window, we calculate total amount of metadata (except reference count of physical blocks) to be updated, and select a data block for compression. If the block can be compressed and the extra space made available by the compression is large enough to hold the metadata, all the metadata are inserted into the block (see Fig. 1a). If the extra space is not large enough, only part of them is inserted. Whenever there are still metadata remaining to be inserted, another block is selected for compression. This process is repeated until all metadata are inserted (see Fig. 1b), or all the data blocks have been tried.

After this, all (compressed and uncompressed) blocks in the window are written to the disk without using flushes to enforce a particular order between them for high disk efficiency. If there still exist some metadata that cannot be inserted into data blocks, they will stay in their respective metadata blocks and be written to the disk (see Fig. 2). After submitting the block writes to the disk, ThinDedup issues a flush command to the disk to make sure all the blocks are persisted. Only after the flush is complete, the write requests that had been buffered in the window are acknowledged indicating the data have been safely written.

This design has several efforts on reducing I/O costs. First, persistency with a window of data blocks improves write locality exploited by the I/O scheduler. Second, with such a window of data blocks, it would be relatively easy to find one or more blocks that can be compressed to receive metadata, even when deduplication ratio is high. Third, when data blocks



(a) Case I (b) Case II

Fig. 2: Cases of serving write requests in a flush window in ThinDedup where explicit metadata persisting is needed. There are two cases based on the deduplication ratio and the compressibility of blocks in a window. In the figure, four data blocks A, B, C, and D in the same window are issued to ThinDedup system. In case I, A and D are deduplicated and neither B nor C can be compressed, and in case II, A, B, C and D are deduplicted and no data blocks are written to disk in the window.

are well compressible, we can compress only one or a few blocks for receiving all metadata associating with the writes in the window and leave most blocks in a window uncompressed. This has the potential to significantly reduce the compression cost for higher write performance as well as the decompression cost for higher read performance. Fourth, address mapping and fingerprint about the same data block are stored together, either inserted into a compressed data block or left in a metadata block. Therefore, they will be atomically written to the disk. Usually, a data block has to be written (persisted) before its corresponding address mapping. This is often enforced with a flush between the writes, a soft-update-style approach for crash consistency [10]. However, by storing the data block's fingerprint together with the mapping entry ThinDedup allows them to be persisted in any order without using a flush. This doesn't lead to a consistency issue. Assume that a system crash happens when the mapping info is persisted but the corresponding data block is not yet. As the mapping is stored with the fingerprint, ThinDedup uses the fingerprint to verify the data block at the address pointed to by the mapping and can detect that the data block is not a valid one. Therefore, wrong data will not be returned. Note that the address mappings and fingerprints inserted into data blocks on the disk are not ready for online use as they may not yet be reflected in on-disk data structures regarding the deduplication's metadata. However, for a substantially long time period they are still in memory for serving I/O requests before being forced out of memory. Furthermore, after their persistency with the data blocks, they will incrementally be committed to the data structure.

In a deduplication system a reference count is maintained for each physical block tracking number of logical blocks mapped to the physical block. Once the count is decremented to 0, the physical block can be reclaimed for receiving new data. Whenever a data block is updated, the old physical block's count cannot be decremented before incrementing new physical blocks's count and persistency of new data to prevent data loss. Enforcing such an order for every update needs many flushes. Instead, ThinDedup enforces the consistency requirement in a much larger scale. For every certain (large) number of windows ThinDedup first writes all increased reference counts during the windows to the disk, followed with a flush, and then writes all decreased counts. In this way, the flush's cost can be well amortized. A crash during the process may leave inconsistency between reference counts and address mappings. This is resolved by scanning the address mappings touched during the windows.

B. Zone-based Data Persistence

ThinDedup has two types of data blocks to store on the disk (compressed and uncompressed). However, a data block itself cannot reveal its own type. Therefore, ThinDedup sets up two types of space zone, each for holding blocks of corresponding type. Specifically, *C-Zone* is to hold compressed blocks with metadata inserted in them, and *U-Zone* is to hold uncompressed (regular) data blocks. Figure. 1 and 2 show examples how compressed and uncompressed data blocks are stored in the C-Zone and U-Zone.

A zone holds a large number of blocks (e.g., a 4MB zone can hold 1024 4KB blocks). We maintain a zone bitmap on the disk, where each zone's status is represented by two bits. There are four possible statues: unallocated, U-Zone, uncommitted C-Zone, and committed C-Zone. ThinDedup periodically (usually not during the period when the system is fully loaded) writes the metadata embedded in the compressed blocks in a C-Zone to the well-structured metadata area on the disk. After the commitment, the C-Zone changes its status from 'uncommitted' to 'committed'. Note that this commitment usually does not involve reading the compressed blocks into the memory, as recently used metadata are always retained in the memory for high performance. Because a zone is relatively large, the update of its status is frequent and the cost for updating the status is very small.

The structured metadata area includes two arrays (an array of logical-to-physical block address mapping entries and an array of physical-address-to-reference-count entries) and a B+ tree for indexing fingerprints to their corresponding physical addresses. However, they may not be up to date, as the newest updates may be still only permanently stored in the C-Zones and in the volatile memory. After an expected system crash, the volatile updates are lost. Although the updates are also available in the C-Zone(s), they are not yet well indexed and readily usable. To recover the metadata, ThinDedup needs to scan all compressed blocks that contain not-yet-committed metadata to extract the metadata and commit them to the metadata structure. Because zone statuses are synchronously persisted, this scanning only needs to cover those zones whose statuses are marked as 'uncommitted C-Zones'. Because C-Zones are periodically committed, the zones that have to be scanned during the recovery period is of small number and ThinDedup's impact on the recovery time is minimal.

The allocation of zone space is generally conducted in a manner similar to that of a log-structured file. Whenever a recently allocated zone is filled to its capacity with blocks of the same type (either compressed or not), a new zone of the same type is allocated. Zones are allocated sequentially and the new zone is appended at the end of the log. Similar to a log-structure file system, when serving a write to LBA that has been mapped to a PBA ThinDedup does not overwrite data in the PBA. Instead, it performs an out-of-place write to a new PBA to ease the maintenance of metadata, in particular, the consistency of a data block's content and its fingerprint. Correspondingly, reference count of the original PBA block is decremented by one. When the count is reduced to zero, space occupied by the corresponding block can be reclaimed and reused. The common practice in log-structure file systems for performing garbage collection in a selected segment (equivalent to the zone in ThinDedup) is to copy all live blocks out of the segment and make the entire segment available for new allocations. However, ThinDedup does not take the strategy to minimize metadata maintenance cost. Migration of each live block can lead to a number of updates on metadata, some of which may not be committed yet. These metadata include LBA-to-PBA mappings, PBA-to-reference-count entry, and fingerprint-to-PBA entry. As a major design objective of ThinDedup is to minimize metadata write cost, we leave the live blocks in a zone in place during space reclamation and re-allocate the available space in the zone. A downside of the strategy is that the spatial locality may be compromised for sequential writes. While the impact is small for SSD, it can potentially degrade I/O performance on the hard disk. To this end, ThinDedup first selects zones with a large number of contiguous idle blocks for space reclamation. In the future, we will consider introducing operations for defragmenting scattered idle spaces when the system is not loaded. In the reuse of idle blocks in a zone, there are two issues that have to be addressed effectively.

The first issue is about efficient maintenance of block status. To determine whether a block is idle, we maintain a bitmap for a zone, each bit for a block. The bitmap needs to be up-to-date so that allocated blocks are not re-allocated until their reference counts reach zero. For correctness and space efficiency the bitmap needs to be updated whenever a block is allocated or a block's reference count turns into zero. The bitmaps are small enough to be kept in memory and keeping the maps in the memory up-to-date is efficient. However, immediately updating them on the disk is not affordable in ThinDedup. In theory, after a loss of up-to-date bitmaps in the memory due to a power failure or system crash, the bitmaps on the disk can be brought up to date by scanning all the block mappings. However, such a recovery process can be too long. To reduce number of zones that have to be scanned in the recovery, we assign each zone a one-bit flag indicating whether the zone's in-memory bitmap is consistent to the one on the disk ('clean') or not ('dirty'). When a clean bitmap is to be updated, ThinDedup changes its status to 'dirty' and synchronously write it to the disk. The following updates on the bitmap will not incur any I/O operation until the bitmap is scheduled to be persisted. Because of spatial locality in the block write and allocation, the number of dirty bitmaps would be limited. When an idle block in a committed C-Zone is re-allocated and receives a new block of data, the zone must change its status to 'uncommitted' to reflect the fact that there are embedded and uncommitted metadata in the zone. While this status change also needs to be persisted, we co-locate the bitmap and the commitment status about a zone and persist them together to save an I/O operation.

When a compressed block is written into a committed C-Zone, its embedded metadata is not yet committed. So the second issue is how to efficiently distinguish blocks whose embedded metadata have been committed from the new written ones that have un-committed embedded metadata to avoid committing them again. To this end, ThinDedup maintains a clock for each type of metadata. The clock ticks when a new metadata entry of the type is generated, and current clock reading becomes a timestamp attached to the entry. When each metadata entry has its unique timestamp, only an entry with a larger timestamp can overwrite one with a smaller timestamp during the commitment. When a C-Zone completes its commitment and changes its status to 'committed', for each type of metadata it records the largest timestamp among all of its blocks as the zone's timestamp for this type. In this way, new metadata can be easily recognized as their timestamps are larger than the zone's. When the zone is committed again, only the new ones are considered for being committed. Because there is a possibility that the metadata entries in the structured metadata area can be newer ones, ThinDedup always compares timestamps during the commitment to make sure that old entries do not overwrite newer ones.

III. Performance Evaluation

We implemented ThinDedup at the generic operating system block device layer as a device mapper target in Linux kernel 4.7.1. The design follows the basic block read/write interfaces of Dmdedup [6]. We extensively conducted experiments on the prototype to evaluate its performance.

A. *Experiment Setup*

In our evaluation, we use 4KB as block size and calculate a 128-bit MD5 value for each block as its fingerprint. The LBA and PBA are 8 bytes, respectively, and an 8-byte timestamp is used to serialize the metadata entries. Thus, the metadata about a block write to be inserted in a compressed data block is 40 Bytes (16B LBA$\rightarrow$PBA+16B fingerprint+8B timestamp), which is only about 1% of the block size. We use a fast compression algorithm (LZ4 [11]) for data (de)compression. On the server used in the evaluation, the algorithm can produce about 700MB/s and 2.45GB/s throughputs for compressing and decompressing 4KB blocks, respectively. Regarding window size, we test windows of different sizes in the evaluation. As write requests are continuously fed into the system, we use number of blocks requested for writing to define window size. Zone size is 4MB. C-Zones start to be committed when there are two or more uncommitted C-Zones in the system.

The experiments are conducted on a server with two Xeon E5-2680v3 2.50GHz CPUs, each has twelve cores and 30MB last-level cache. The server equips with 128GB DDR4 memory, a 320GB Western Digital Caviar Blue SATA disk and a 1TB Crucial/Micron SSD. For the hard disk, its 90th percentile latencies of sequential and random writes of 4KB block are 8.33ms and 13.36ms, respectively. For the SSD device, the latencies are 5.67ms and 5.71ms, respectively.

In the evaluation, we compare ThinDedup with three other block-layer deduplication schemes. Among them, ***ideal*** represents an ideal (but unrealistic) scenario for producing the optimal performance, where metadata are cached in memory and only data blocks are written to the disk. ***OrderedWrites*** is similar to ThinDedup except that its metadata are not inserted into data blocks. Instead, at the end of a window, all dirty data blocks are batch written to the disk, followed by a flush to establish the order, and then all critical metadata (mainly the block mappings) are batch written to the metadata structure on the disk. This scheme represents the upper-bound performance of the OrderMergeDedup [7] scheme, which uses flushes to enforce order and uses I/O delay and merging to exploit write locality. *OrderedWrites* limits the number of flushes for ordering to only one and allows requests within a window to be scheduled freely. However, OrderedWrites does not always ensure metadata consistency on the disk. After a crash, all metadata are restored to a consistent state by scanning the persisted critical metadata. ***Dmdedup*** is an open-source deduplication system using page shadowing to maintain on-disk metadata consistency. It uses Linux's on-disk Copy-on-Write (COW) B-tree implementation to organize the metadata, and each metadata update involves several metadata page write operations and flushes. This scheme can keep the metadata and data always consistent, and does not require a recovery for consistency after a crash.

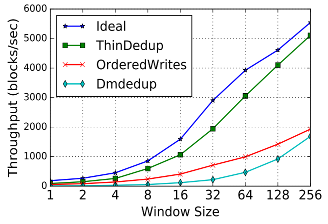
B. Experiment Results with Synthetic Workloads

We generate synthetic block write traces and issue them continuously to each of the deduplication systems. For the traces we test different window sizes, deduplication ratios and compressibility ratios. The deduplication ratio is the ratio of numbers of data blocks written to the disk before and after the deduplication. The compressibility ratio is the ratio of data blocks that can be compressed among all blocks in a window to be written to the disk. A compression ratio of 1.25 means about 20% of the block space can be compressed for holding metadata. We also test random and sequential access patterns. For random access, data blocks' logic address is randomly distributed in a 120GB address space. In the evaluation, the experiments are conducted on both hard disk (HDD) and SSD.

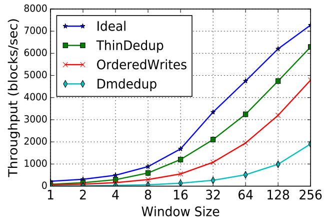
Figure 3 shows the write throughput with the four deduplication schemes using different window sizes with various access patterns and storage devices. As shown in all cases ThinDedup produces much higher throughput than that of OrderedWrites and Dmdedup. In particular, with random accesses ThinDedup has a larger improvement than sequential ones due to the fact that random accesses cause more metadata page writes and ThinDedup avoids most of them by inserting the metadata in compressed data blocks. Generally, using larger window size helps increase relative performance advantage of ThinDedup as it is more likely to find data blocks for compression to avoid expensive explicit metadata writes. However, when the window is too large, for example 256 in Fig. 3, the improvement can become smaller as all benefits from compression have been exploited by ThinDedup while OrderedWrites and Dmdedup favor large windows. For example, ThinDedup achieves the highest improvement when window size is 64, which is about 3X compared to that of OrderedWrites, and the improvement is 2.73X and 2.88X when the window size is 32 and 128 respectively. With sequential access on the SSD, the improvements reduce to 1.9X and 1.77X for 32 and 128 window size, respectively. While data blocks are written sequentially, random access in the logical address space does lead to random access on the disk. Furthermore, sequential access helps reduce amplification of metadata write, as multiple metadata entries are more likely located in the same metadata blocks. This is why even on SSD, which is less sensitive to access pattern, sequential access receives a higher throughput than random access. In comparison, Dmdedup consistently has the lowest throughput, as it involves the largest number of metadata writes and flush operations for strong consistency. Throughput of ThinDedup is close to that of the *Ideal*'s deduplication. This is especially the case with the use of SSD, as our measurements show that ThinDedup removes more than 95% of the metadata and the remaining metadata writes introduces relatively low overhead. On the hard disk, write of the small amount of leftover metadata produces a larger performance loss.

It is obvious that increasing window size can help substantially increase throughput in all the systems. However, it also deteriorates request latency, which measures the period from the time when a request enters a scheduling window to the time when the window of requests are serviced. Figure 4 shows CDF curves of request latency with different window sizes for the four systems on the hard disk. For a given window size, ThinDedup has a latency much lower than OrderedWrites and Dmdedup, and close to that of Ideal. With a larger window size the latency can be significantly increased (though throughput also greatly increases). For example, the 80th percentile latencies of ThinDedup and OrderedWrites with a 32-block window and random access are increased to 3.5X and 3.2X, respectively, when the window size increases to 256. However, Fig. 3a shows that ThinDedup has about the same throughput (about 2000 blocks/sec) at a 32-block window as that of OrderedWrites at a 256-block window. That indicates that ThinDedup can achieve higher throughput without having to significantly increase the window and compromising latency. For sequential access, Fig. 3b shows that ThinDedup at a 32-block window has a throughput (about 2000 blocks/sec) similar to that of OrderedWrites at 64-block window. However, it can have a lower latency, as seen in Figs. 4c and 4d.

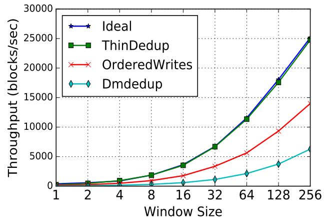
While throughputs of random access on SSD for ThinDedup with a 8-block window and OrderedWrites with a 16-block window are similar (about 2000 blocks/second), we show their latency in Fig. 5. We find that this increase of window size



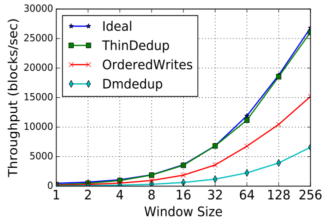
(a) Random Access to HDD



(b) Sequential Access to HDD

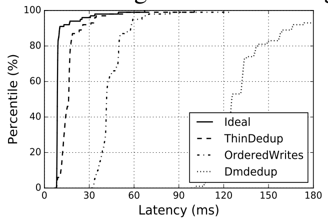


(c) Random Access to SSD

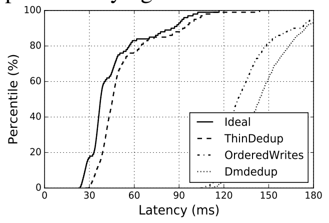


(d) Sequential Access to SSD

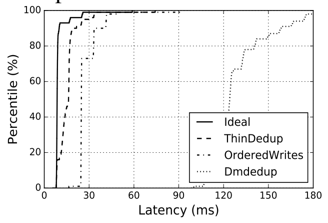
Fig. 3: Write throughput for varying window sizes. The deduplication ratio is 2 and compressibility ratio is 80%.



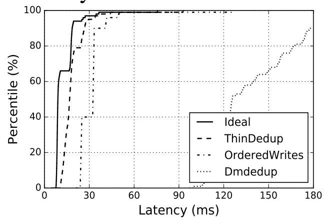
(a) Window size=32, Random



(b) Window size=256, Random

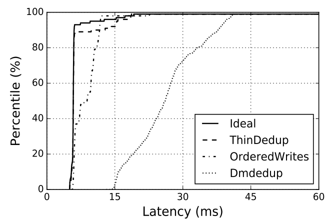


(c) Window size=32, Sequential

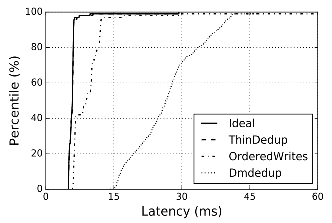


(d) Window size=64, Sequential

Fig. 4: Write latency on the **HDD** with different window sizes. The deduplication ratio is 2 and compressibility ratio is 80%.



(a) Window Size = 8



(b) Window Size = 16

Fig. 5: Random write latency on **SSD** with different window sizes. Deduplication ratio is 2 and compressibility ratio is 80%.

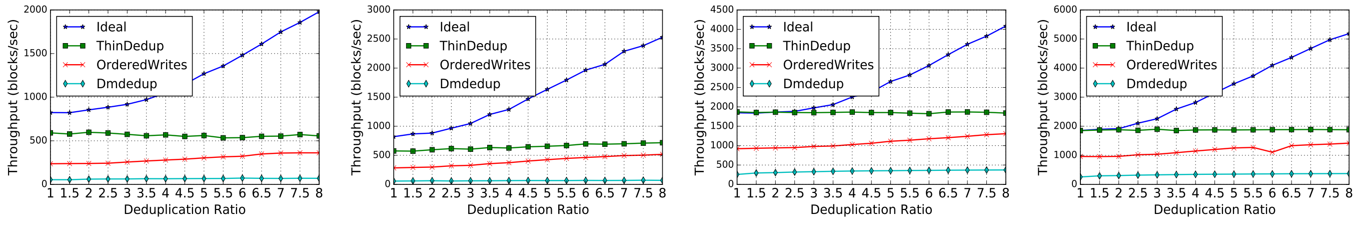
does not significantly impact the latency. SSD has a much higher speed than the hard disk. The I/O time spent for a window of requests is mainly dominated by the flush operations. While increasing window size does not require more flushes, the impact is small. However, because ThinDedup reduces one flush with its fingerprint-assisted consistency for each window, its latency is significantly smaller than OrderedWrites.

To investigate how deduplication ratio affects throughput of the systems, we change the ratio in traces of different access patterns on the hard disk and SSD. The results are shown in Fig. 6. As shown, the Ideal system keeps increasing its throughput with increase of the ratio as more writes of data blocks are removed. For other systems, deduplication of a data block only removes write of data blocks, and metadata remain. For OrderedWrites and Dmdedup, higher deduplication ratio does not help increase throughput. As data blocks are sequentially written, the improvement due to reduction of number of data blocks is limited. While ThinDedup maintains a higher throughput than these two systems, its throughput has minimal increase with the increase of deduplication ratio, and even has small decrease. With a high deduplication ratio, many data blocks in a window are removed, and it is hard to remove metadata by finding compressible data blocks to insert them, and more metadata have to be explicitly written to the disk. Fig. 7 shows the percentage of metadata blocks that have to be explicitly written to the disk due to inability of being inserted into data blocks (compared to number of written metadata blocks with zero insertion). As we can see, if the deduplication ratio is relatively small (a ratio of around 2 is common [12]) or the window size is large, almost all writes of metadata blocks can be removed. Fig. 6 also shows that ThinDedup's throughput is almost the same as that of the Ideal system on the SSD. On an SSD, write of a few leftover metadata blocks in ThinDedup has minimal impact on its performance. Meanwhile, its removal of a flush per window does give its advantage over the other two systems.

To understand the impact of data compressibility, we show ThinDedup's throughput on the hard disk under different data compressibility ratios in Fig. 8. As we can see, with a reasonably large window sizes (16 or 32), as long as the ratio is 40% or higher, or 40% or more of data blocks in a window can be compressed, ThinDedup can receive its full benefit. With a high compressibility ratio (e.g., 80%) and window size (e.g., 32), a high data block compression ratio (e.g, 1.5), and a moderate deduplication ratio (e.g., 2), only about 10% of the data blocks need to be compressed and only a small percentage of zones on the disk are C-Zones (e.g., 10%). Considering the high throughput of the (de)compression operations and the low percentage of C-Zone, the performance degradation caused by the (de)compression operations is negligible.

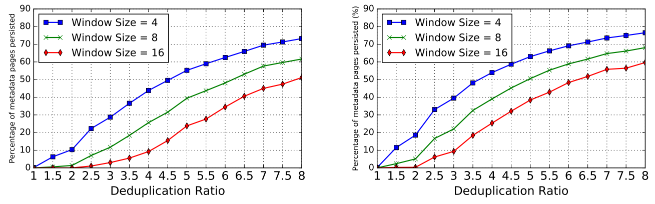
C. *Experiment Results with Real-world Workloads*

To evaluate ThinDedup under realistic workloads, we use publicly available FIU traces, which includes three traces collected on production systems at Florida International University: Web, Mail, and Homes [13]. Each trace covers requests in 21 continuous days. Access pattern in the trace is relatively consistent across the duration of 21 days. However, the deduplication ratio varies substantially across the days. To highlight correlation of the trace characteristic with its running performance, we choose three write-intensive segments from each trace, each segment representing one-day-long accesses of distinct deduplication ratio. In an experiment, the system is warmed up with requests preceding the tested segment of requests. Table I lists the days on which segments are chosen



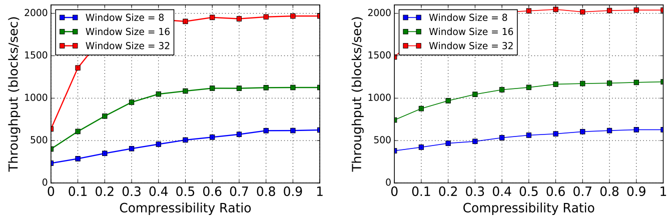
(a) Random write to HDD (b) Sequential write to HDD (c) Random write to SSD (d) Sequential write to SSD

Fig. 6: Throughput under different deduplication ratios. The window size is 8 and compressibility ratio is 80%.



(a) Random access (b) Sequential access

Fig. 7: Percentage of metadata blocks written to disk with different deduplication ratio. The compressibility ratio is 80%.

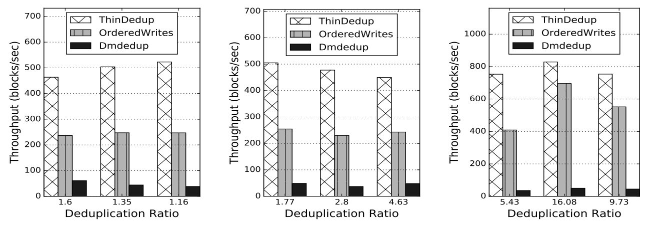


(a) Random Access (b) Sequential Access

Fig. 8: ThinDedup's throughput on HDD with different data compressibility. The deduplication ratio is 2.

TABLE I: Characteristics of FIU traces

Trace	Selected Day	Dedup. Ratio
WebVM	4,8,12	1.60, 1.35, 1.16
Homes	11,12,13	1.77, 2.80 , 4.63
Mail	1, 2, 5	5.43, 16.08, 9.73



(a) WebVM (b) Homes (c) Mail

Fig. 9: Throughput of different trace segments on HDD.

and their deduplication ratios. The traces only contain fingerprints without real data. We assume a compressibility ratio of 80% (80% of blocks are compressible) and a compression ratio of 1.25 (each compressible block can contribute 20% of its space), which is a conservative and reasonable assumption according to [14]. The window size is 8.

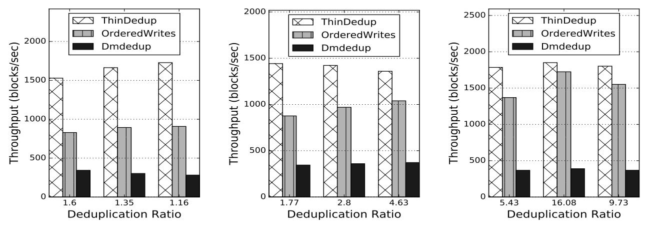
Figures 9 and 10 show the throughput of trace segments on the hard disk and on SSD. Across the test cases ThinDedup produces the highest throughput, while Dmdedup's throughput is much worse than the two systems. Compared to OrderedWrites, ThinDedup improves throughput by 19% to 112% on the hard disk and 7% to 91% on the SSD. Consistent to observations on synthetic workloads, a very high deduplication ratio reduces ThinDedup's advantage (see throughput of the Mail trace with the ratios of 16.08 and 9.73 in Fig. 9c and 10c, respectively). A higher ratio reduces opportunity of finding compressible blocks. As we expected, improvements of ThinDedup on the SSD is smaller than those on the hard disk with the same trace segment as SSD's performance is less sensitive to random writes. However, SSD's performance is still heavily affected by flushes, which ThinDedup manages to reduce. Also, considering the amount of data blocks written to the disk when the deduplication ratio is extremely high (e.g. 16.08), about 27.8% of the metadata writes can be avoided. This is important for SSD, which has limited lifespan. We can also see that a high deduplication ratio (e.g., 16.08 and 9.73 in Fig. 9c and 10c, respectively) does not necessarily lead to substantial improvement of throughput, though many writes of data blocks can be removed. As data blocks are sequentially written, cost of persisting metadata can dominate the I/O cost in the deduplication. This highlights the importance of addressing the metadata issue in an online primary deduplication.

IV. Related Work

Related work includes I/O deduplication, crash consistency maintenance and data compression in storage systems.

*a) **Deduplication in storage systems**:* Many previous studies focus on deduplication in backup and archival storage systems [4], [15]–[17] instead of primary storage systems where improving deduplication ratio for space saving is the main goal [17], [18]. Moreover, as data has backups during the deduplication operation, consistency is usually not a major concern for offline deduplication. While this is not the case for inline deduplication in primary storage system [3], [5], [19], [20], where performance is a key concern, many efforts have been made to minimize the performance overhead. As deduplication causes fragmentation, the read performance may be degraded as sequential accesses are turned into random ones. Srinivasan et al. [3] try to minimize fragmentation caused by deduplication and improve the read performance of hard-disk-based primary storage systems. This work is complimentary ours as ThinDedup addresses issues also on primary storage. ThinDedup is to improve performance with more efficient management of metadata persistency and consistency.

How to handle deduplication metadata efficiently is of great importance for good performance. DBLK [20] uses multi-layer bloom filter to reduce metadata retrieval cost and cannot help much with the efficiency of metadata persistency. Meister et



(a) WebVM (b) Homes (c) Mail

Fig. 10: Throughput of different trace segments on the SSD.

al. [19] propose to store metadata on a faster storage device (SSD) while data are still stored on the hard disk. However, this does not address the write amplification issue due to writing small metadata through block interface. The benefit of the design diminishes when data blocks are also stored on SSDs. ThinDedup minimizes performance loss due to frequent metadata writes without requiring any special hardware.

b) ***Data compression in storage systems:*** Data compression technique has been widely used in storage systems to save space and improve I/O performance. in [21], process data that are to be written back are compressed for saving memory space and staying in memory longer. Makatos et al. [22] use online I/O compression to improve I/O performance for SSD-based cache. Li et al. [23] combines compression and deduplication to make a space-efficient SSD cache for primary storage. ThinDedup uses the data compression for a different purpose. The space saved is to store small but performance-critical metadata rather than more data. Wu et al. [24] propose the Selfie technique to compress data block to make room for storing address mapping in the virtual disk system. Selfie only compresses a data block to store its own metadata, and if the block cannot be compressed, it has to give up. Thindedup can insert any metadata in a window to a compressed block in a window, and it is very like to find a block for compression.

V. Conclusion

In this paper we describe ThinDedup, an efficient deduplication scheme designed to minimize the performance loss due to metadata persistency. By embedding metadata into the compressed data blocks, ThinDedup removes most of metadata persisting operations out of the critical path. It also uses window-based batch persistency to amortize the high flush overhead. To provide crash consistency, ThinDedup stores fingerprint together with block mapping to remove requirement on the write ordering. Experiments with micro benchmarks and real-world workloads demonstrate that compared to other state-of-the-art approaches, ThinDedup provides up to 3X throughput. Meanwhile, the write latency can be reduced by up to 88% without compromising the throughput.

Acknowledgment

We are grateful to reviewers of the paper for their constructive comments, which helps to improve the papers quality. This work was mainly supported by US National Science Foundation under CNS 1527076. In addition, Weijun Li was supported by Shenzhen Peacock Plan (KQTD20150917164 53118).

References

[1] 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*, vol. 11, 2011.

[2] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage." in *FAST*, vol. 2, 2002, pp. 89–101.

[3] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "idedup: latency-aware, inline data deduplication for primary storage." in *FAST*, vol. 12, 2012, pp. 1–14.

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

[5] L. DuBois, M. Amaldas, and E. Sheppard, "Key considerations as deduplication evolves into primary storage," *White Paper*, vol. 223310, 2011.

[6] 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 *2014 Ottawa Linux Symposium*, 2014.

[7] Z. Chen and K. Shen, "Ordermergededup: Efficient, failure-consistent deduplication on flash," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016, pp. 291–299.

[8] T. Harter, C. Dragga, M. Vaughn, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "A file is not a file: understanding the i/o behavior of apple desktop applications," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 3, p. 10, 2012.

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

[10] G. R. Ganger, M. K. McKusick, C. A. Soules, and Y. N. Patt, "Soft updates: a solution to the metadata update problem in file systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 2, pp. 127–153, 2000.

[11] Y. Collet, "lz4," http://cyan4973.github.io/lz4/, 2015.

[12] M. Lu, D. Chambliss, J. Glider, and C. Constantinescu, "Insights for data reduction in primary storage: a practical analysis," in *Proceedings of the 5th Annual International Systems and Storage Conference*. ACM, 2012, p. 17.

[13] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 13, 2010.

[14] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems." in *FAST*, vol. 4, 2012, p. 500.

[15] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters." in *FAST*, vol. 11, 2011, pp. 15–29.

[16] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system." in *USENIX Annual Technical Conference*, 2011.

[17] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage." in *FAST*, vol. 2, 2002, pp. 89–101.

[18] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35. ACM, 2001, pp. 174–187.

[19] D. Meister and A. Brinkmann, "dedupv1: Improving deduplication throughput using solid state drives (ssd)," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2010, pp. 1–6.

[20] Y. Tsuchiya and T. Watanabe, "Dblk: Deduplication for primary block storage," in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2011, pp. 1–5.

[21] S. Jennings, "Transparent memory compression in linux," 2013.

[22] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Using transparent compression to improve ssd-based i/o caches," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 1–14.

[23] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: a capacity-optimized ssd cache for primary storage," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 501–512.

[24] X. Wu, Z. Shao, and S. Jiang, "Selfie: co-locating metadata and data to enable fast virtual block devices," in *Proceedings of the 8th ACM International Systems and Storage Conference*. ACM, 2015, p. 2.