DAC: Dedup-Assisted Compression Scheme for Improving Lifetime of NAND Storage Systems

Jisung Park, Sungjin Lee[†], and Jihong Kim
Seoul National University and [†]Inha University
{jspark, jihong}@davinci.snu.ac.kr, [†]sungjin.lee@inha.ac.kr

Abstract*—Thanks to an aggressive scaling of semiconductor devices, the capacity of NAND flash-based solid-state-drives (SSDs) has increased greatly. However, this benefit comes at the expense of a serious degradation of NAND device's lifetime. In order to improve the lifetime of flash-based SSDs, various data reduction techniques, such as deduplication, lossless compression, and delta compression, are rapidly adopted to SSDs. Although each technique has been extensively studied, efficiently combining these techniques in a synergistic fashion was not thoroughly investigated. In this paper, we propose a novel dedup-assisted compression (DAC) scheme that integrates existing data reduction techniques so that potential benefits of individual ones can be maximized while overcoming their inherent limitations. By doing so, DAC greatly reduces the amount of write traffic sent to SSDs. DAC also requires negligible resources by utilizing existing hardware modules. Our evaluation results show that the proposed DAC decreases the amount of written data by up to 30% over a simple integration of deduplication and lossless compression.

I. Introduction

The limited lifetime of NAND flash has been considered a major obstacle for wider use of flash-based solid-state drives (SSDs). With a continued shrinking of semiconductor processes and an adoption of multi-leveling technologies, the capacity of SSDs has been increased greatly. However, the lifetime of SSDs, which is decided by the maximum number of program and erase (P/E) cycles of NAND flash memory cells, has been significantly reduced. For example, SLC NAND manufactured at 5x-nm has 100K P/E cycles, but it drops to 1K P/E cycles with TLC (Triple-Level Cell) NAND at 2x-nm. For recent 3D TLC NAND flash, its P/E cycles are much larger than 2D TLC NAND flash. However, as it scales down, the P/E cycles are expected to get smaller as well.

Since increasing the number of P/E cycles itself is known to be difficult without the fundamental changes of current semiconductor technologies (e.g., new materials), many researchers have focused on developing software/controller-level solutions that can improve the lifetime of SSDs without changing underlying devices. One of popular such solutions is based on a data reduction approach where the amount of data physically written to flash is intelligently reduced. With less amount of written data, SSD lifetimes can be improved by experiencing smaller P/E cycles for a given write traffic. Data deduplication [1], lossless compression [2], and delta compression [3] belong to this solution.

Individual techniques mentioned above exploit different properties of incoming data, so their effects on write traffic reduction are different depending on characteristics of workloads. Data deduplication is effective only when data has high value locality – that is, exactly the same values are frequently written to flash. It fails even if there is a single bit difference between the incoming data and reference data (which is previously written in flash). Lossless compression is not dependent upon value locality, but it works well when incoming data has low entropy. Delta compression compares modified data with original one and writes only differences to flash. Although delta compression is not affected by value locality and data entropy, it can achieve a high compression ratio only when slightly modified data is frequently overwritten.

Since the effectiveness of each technique highly depends on input data, combining multiple data reduction techniques has received a lot of attention in order to achieve high data reduction efficiency over a wide range of data [4]. A simple example is a combination of data deduplication and lossless compression. It first performs deduplication to prevent duplicate data from being written if the same values were already stored. When the deduplication step fails, it applies lossless compression as the second reduction technique. Although this combined technique works better than ones based on a single reduction scheme, this naive combination cannot resolve the inherent limitations of existing techniques, losing some significant opportunities to further reduce incoming write traffic. For example, it does not effectively handle incoming data that has low value locality and high entropy where both deduplication and lossless compression work poorly.

In this paper, we propose a novel integrated data reduction technique, called **d**edup-**a**ssisted **c**ompression (DAC). As its name implies, DAC is based on deduplication and lossless compression, but its approach is totally different from their naive combination. DAC aims at bridging a big gap between deduplication and lossless compression. Unlike deduplication that only considers exactly matched values among blocks and lossless compression that eliminates exactly matched/repeated bit patterns within a block, DAC takes into account data similarity across entire storage space. This enables us to handle a wider spectrum of input workloads, where neither of the individual techniques can effectively deal with.

By exploiting a hash function of a deduplication engine in a finer-grained fashion, DAC locates the most similar pages, not exactly same pages only, in order to overcome the limitation of deduplication. Once the similar page is chosen as a reference for a write request, DAC performs XORing between the requested page and reference page, and resulting data is written to flash after being compressed by a lossless compression engine. Since XORed data of similar pages has extremely low entropy, DAC achieves a much higher compression ratio than merely applying lossless compression. One may think that DAC is similar to delta compression, but this is not the case. Unlike delta compression that takes into account only

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science, ICT and Future Planning (MSIP) (NRF-2015M3C4A7065645). The ICT at Seoul National University provided research facilities for this study. (*Corresponding author: Jihong Kim*)

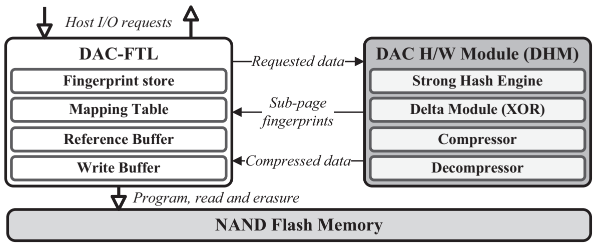


Fig. 1: An organizational overview of our target SSDs.

one original page as a reference, DAC searches for the most similar one in the entire storage, thereby achieving a much better data reduction ratio.

In order to evaluate the effectiveness of the proposed DAC, we develop a new FTL for DAC, which is called DAC-FTL, in a flash emulation environment [6]. We have evaluated our DAC-FTL using various real-world traces, and our experimental results show that DAC-FTL can reduce the amount of written requests up to about 30% over the naive combination of deduplication and lossless compression. Additional resources added to an existing flash controller are negligible.

The rest of this paper is organized as follows. In Section II, we explain an architecture of our target SSD with the DAC scheme. Section III details the dedup-assisted compression algorithm, and in Section IV, we explain FTL design issues for DAC. Section V shows our experimental results, and Section VI concludes with a summary.

II. Overall Architecture of Target SSDs

Fig. 1 depicts an overall architecture of our baseline SSD with the DAC scheme. Similar to recent high-end SSDs employing deduplication and compression, our baseline SSD has a specialized hardware controller, called DAC hardware module (DHM), which is composed of two sub-modules, a strong hash function accelerator and a compression/decompression accelerator. A simple XOR logic is only a new one that is added to the hardware controller. In spite of architectural similarity, DHM works differently from the existing SSD controller in that it is designed to find similar references and to lower data entropy for better compression (see Section III).

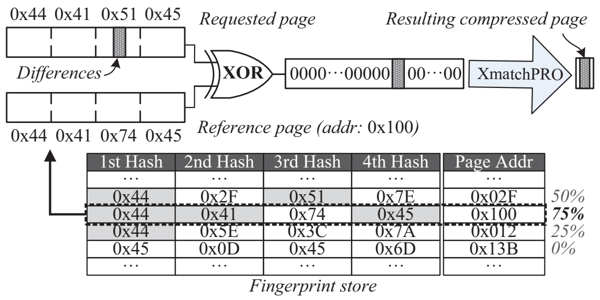
A DAC-aware FTL (DAC-FTL) is a new FTL design for DAC. It handles read and write requests from the host as usual FTLs, but its primary goal is to reduce write traffic by utilizing DHM. DAC-FTL maintains a fingerprint store for efficient search of similar data, in addition to an enhanced mapping table. Two buffers, reference and write buffers, are required for fast read operations and for preventing internal fragmentation, respectively (see Section IV).

III. DAC: Dedup-Assisted Compression

In this section, we describe the mechanism of the proposed dedup-assisted compression algorithm, particularly focusing on its two unique features, (i) reference search and (ii) compression with a reference.

A. Reference Data Search

Existing deduplication techniques aim at finding reference data that are exactly matched to requested data. To this end, a strong hash algorithm is widely used to generate a unique fingerprint for a specific bit pattern with an extremely low collision rate. One important property of the strong hash algorithm is that it creates very different fingerprints with a long distance if bit patterns are similar. This is a desirable characteristic for deduplication and security applications, but it is undesirable for DAC because our goal is to find a similar reference. That is, a fingerprint from a strong hash engine is not a feasible metric to decide similarity among candidates.



1st Hash	2nd Hash	3rd Hash	4th Hash	Page Addr
···	···	···	···	···
0x44	0x2F	0x51	0x7E	0x02F
0x44	0x41	0x74	0x45	0x100
0x44	0x5E	0x3C	0x7A	0x012
0x45	0x0D	0x45	0x6D	0x13B
···	···	···	···	···

Fig. 2: The proposed dedup-assisted compression mechanism.

There is another category of hash functions like locality-sensitive hashing that creates fingerprints with a short distance if bit patterns of data are similar. Using such hash functions is beyond the scope of this study because we want to maximally reuse existing hardware modules with minimal changes at the controller level, so as to provide good compatibility with existing SSD controllers. However, the adoption or the development of new hash algorithms suitable for DAC would be interesting future work.

Instead of using or developing new hash functions, DAC makes use of existing strong hash functions for similarity detection. DAC splits a flash page into several sub-pages. For example, if a page size and a sub-page size are assumed to be 16 KB and 4 KB, respectively, there are four sub-pages per page. Using a strong hash function, it then calculates fingerprints of all the sub-pages belonging to a requested page. To find a similar reference, DAC compares four fingerprints with those in a fingerprint store which keeps fingerprints of all the candidates. If there is a candidate page whose fingerprints are all matched, its content is identical to the requested one, showing the best similarity. If nothing is matched, it means that there are no similar references. As a result, based on the number of fingerprints matched, DAC estimates the degree of similarity. As expected, among all the candidates, the most similar one is selected as a reference.

Fig. 2 shows an example of how DAC finds a similar reference. The sub-page fingerprints of a requested page are `0x44`, `0x41`, `0x51`, and `0x45`, respectively. Using those values, DAC searches for the most similar one in the fingerprint store. The page `0x100` is selected as a reference page because it has the largest number of matched sub-pages.

The fingerprint store contains a large number of candidates, so searching for a similar reference by comparing fingerprints would take a very long time. This issue has been intensively investigated by previous deduplication studies, and they showed that reference search can be quickly completed in a constant time. Only the difference between conventional deduplication and DAC is that DAC requires more than one fingerprint comparison. However, the impact of additional fingerprints search on performance is actually negligible due to long I/O latencies of NAND flash.

B. Data Compression with a Reference

Once a similar reference is found, DAC conducts XORing between a reference page and a requested page. This can be done quickly by using hardware-based XOR logics. As depicted in Fig. 2, the reference and the requested pages are very similar, so resulting data mostly has '0', exhibiting extremely low data entropy. Only the exception is a narrow range of region where different bit patterns from the reference are observed. For such data which consists of a series of '0' values, even a simple lossless compression algorithm exploiting run-length encoding can achieve an extremely high compression ratio.

While any lossless algorithms can be used for DAC, we select the XmatchPRO algorithm [5] in this study, which takes advantage of both the run-length and the dictionary-based data encoding. The run-length encoding feature of XmatchPRO is effective to compress successive bit patterns. In the case where similar references are found as shown in Fig. 2, it converts several kilobytes of data to few bytes. XmatchPRO also makes use of the dictionary-based encoding, so it achieves a fairly good compression ratio even for data which has no similar reference. In our observation, XmatchPRO exhibits comparable performance as other LZ-variant algorithms for small size data like 4-16 KB flash pages.

IV. Design of DAC-Aware FTL

A. Request Handling in DAC-FTL

DAC-FTL is responsible for handling incoming read and write requests, performing data compression by communicating with the DHM controller. Fig. 3 shows request handling procedures of DAC-FTL. When a write request arrives, DAC-FTL passes the requested data to DHM to get fingerprints for four sub-pages using the strong hash accelerator of DHM, as described in Section III. If there is no reference page, it merely compresses the requested data without a reference using the DHM's XmatchPRO accelerator. If a similar reference is found, DAC-FTL reads a reference page from a flash array and then delivers it to DHM. If the reference is compressed, DHM decompresses it immediately. Note that decompressing the reference data does not require additional reference reads because DAC-FTL never allows a page compressed with a reference to be used as a reference for other pages. It helps us avoid reading another reference page for the current reference page. By XORing the uncompressed reference page and the requested page, DAC gets the low-entropy data, which is then compressed by DHM.

The size of the compressed data is usually smaller than that of a flash page, so writing it to flash directly causes a fragmentation problem, wasting valuable flash space. For this reason, DAC-FTL temporally keeps the compressed data on a write buffer whose capacity is the same as a flash page. When the write buffer becomes full, DAC-FTL flushes the buffered data to flash at once. It would be possible that the compressed data becomes larger than its original size (e.g., if its data entropy is so high with no reference page). In that case, DAC-FTL writes the original (i.e., uncompressed) data to flash directly.

In DAC-FTL, multiple logical pages are packed together to a single physical page. Therefore, a conventional mapping table that maps one logical page to one physical page, i.e., one-to-one mapping, does not work with DAC-FTL. Instead, many-to-one mapping where multiple logical pages point to a physical page is required. Each entry of the mapping table may need to have a physical page number for a logical page, the offset of the logical page on the physical page and the length of compressed data. Maintaining this information in the mapping table requires a large DRAM space. Thus, DAC-FTL keeps all the information about logical pages at the beginning of a physical page where they are stored. This header information is automatically fetched while reading a compressed page, so additional read operations for headers are not necessary.

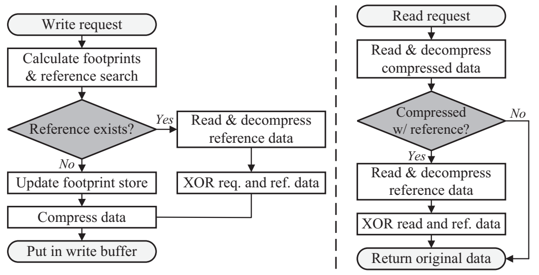


Fig. 3: Request handling procedures of DAC-FTL.

Updating the fingerprint store is the final step of a write process. DAC-FTL adds the fingerprints of the requested data to the fingerprint store *only when* it is compressed without a reference. This is a reasonable choice for the following two reasons; (1) if the requested data has no reference, it means that the contents of that data is unique and similar data was not written before; (2) as we briefly mentioned before, it avoids us to read another reference page when it is selected as a reference page for incoming data in the future.

For a read request, DAC-FTL first reads a physical page from the flash array. If the read data is not compressed, it returns to the host immediately (not shown in Fig. 3). If the data is compressed without a reference, DAC-FTL performs decompression using DHM and delivers uncompressed data to the host. Finally, if the data are compressed with reference data, DAC-FTL reads a reference from flash and decompresses it. Then, the uncompressed requested is XORed with the decompressed reference. The resulting data is sent to the host.

B. Overhead Mitigation in DAC-FTL

DAC-FTL may require a large amount of memory for the fingerprint store. Strong hash functions commonly employed for data deduplication generates tens of bytes hash values per page (e.g., 16 bytes for MD5). If we assume to maintain all the fingerprints for 4 KB pages in 512 GB flash storage, the memory requirement is 2 GB only for fingerprints. Moreover, since DAC-FTL should maintain four fingerprints for each page, the amount of memory for the fingerprint store increases to 8 GB, which is infeasibly large.

DAC-FTL takes two approaches to reduce the memory requirement. Firstly, DAC-FTL uses the first 1/4 of 16 bytes fingerprints from MD5. It may result in more hash collisions, increasing a probability that different data could have the same fingerprint. However, data losses never occur because DAC does not perform data deduplication, always conducting lossless compression even when there is a reference page with all the matched fingerprints. It may decrease the potential

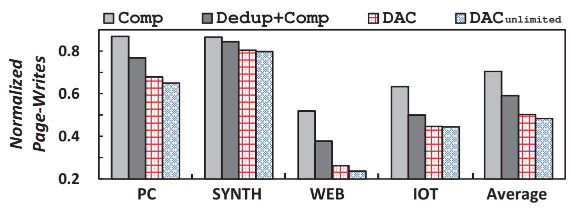


Fig. 4: Comparisons of normalized page-writes for different configurations of data reduction techniques.

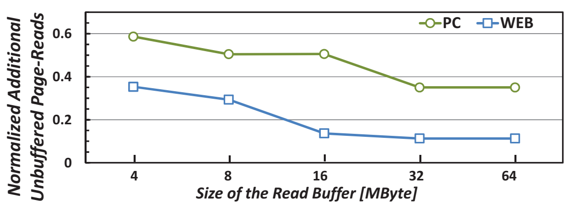


Fig. 5: Comparisons of normalized additional unbuffered page-reads for varying size of read buffer.

benefit of DAC by storing redundant data which can be skipped with deduplication. However, its impact on overall write traffic reduction is negligible since every 4 KB page can be compressed to only 24 Bytes (about 0.5% of the original size) by DAC if it has a reference page whose data is exactly the same as the requested data. In order to further reduce the memory overhead, secondly, DAC-FTL maintains only a limited number of fingerprints in DRAM and manages them in an LRU fashion. This decision is made based on our observation that references to fingerprints have a very skewed distribution with high temporal locality; less than 5% of written pages are being used as reference pages for more than 80% of requested pages. Consequently, its effect on a compression rate is negligible.

Unlike data deduplication and lossless compression, DAC-FTL requires to read reference pages for compression and decompression of the requested page. Those additional reads could badly affect overall I/O performance, especially for read requests from host; DAC-FTL requires two additional reference page reads to service a single page request, which doubles read latencies. To mitigate read overheads, DAC-FTL employs a read buffer whose size is 32 MB that caches popular references. Since reference data has high temporal locality mentioned before, many page reads for reference data can be served from a small read buffer.

V. Evaluations

For our evaluation, we implemented DAC-FTL on FlashBench [6], which is a storage emulation environment for flash-based SSDs. The SSD emulator was 512 GB with eight channels with eight ways, and its page size was 4 KB and the number of pages per block was 128. Hardware modules for a strong hash function (MD5) and lossless compression (XmathPRO) were emulated by software in FlashBench.

In order to evaluate the effectiveness of DAC, we used four block I/O traces collected from a high-end PC in various scenarios. All the traces included actual data as well. PC recorded I/O activities in general PC usages (e.g., documenting, installing programs, etc.), and SYNTH was collected from hardware synthesizing procedures. WEB and IOT captured I/O activities while browsing World-Wide-Web and storing data generated from IOT sensor devices, respectively.

We compared five different SSD configurations with different data reduction techniques. For `Dedup` and `Comp`, the FTL performed either deduplication or lossless compression, respectively. In `Dedup+Comp`, deduplication was first performed and lossless compression was applied if deduplication failed. `DAC` compressed requested data as described in Section IV. The size of the fingerprint store was set to keep 0.5% of all the fingerprints. For example, the fingerprint store manages only 5K fingerprints in an LRU fashion for a trace with 1M fingerprints. $\texttt{DAC}_{\texttt{unlimited}}$ was identical to `DAC` except its fingerprint store was unlimited.

Fig. 4 shows the amount of written pages for each trace under five different configurations. Fig. 4 is normalized to `Dedup`. In terms of a data reduction ratio, `DAC` outperforms the other SSD configurations for every trace. Even compared with `Dedup+Comp` that employs both deduplication and lossless compression, `DAC` reduces write traffic by up to 30% and by 15% on average. This result clearly shows that `DAC` finds similar data patterns more efficiently than the naive combination of deduplication and lossless compression, and prevents them from being written to SSDs.

$\texttt{DAC}_{\texttt{unlimited}}$ shows better performance than `DAC`, further reducing the amount of written data by lower than 2% on average. However, this benefit is not so attractive, considering its high memory consumption. `DAC` only requires 0.5% of that $\texttt{DAC}_{\texttt{unlimited}}$ requires for the fingerprint store, but exhibits similar write traffic reductions, achieving excellent lifetime improvement as well.

We finally evaluate the effect of a read buffer on read performance. Fig. 5 shows the number of additional reads for reference pages with different buffer sizes ranging from 4 MB to 64 MB. This graph is normalized by that of `DAC` with no read buffer. As shown in Fig. 5, the use of a read buffer is effective in reducing reference reads; only with a 64-MB reference buffer, extra reads are reduced by more than 60% (for PC) and 90% (for WEB), respectively.

VI. Conclusions

In this study, we proposed a new dedup-assisted compression scheme which significantly enhanced the lifetime of SSDs by effectively integrating individual data reduction techniques. Our proposed scheme maximally reduced the write traffic to SSDs by utilizing deduplication and lossless compression engines with few additional hardware and memory resources. Our experimental results showed that the proposed scheme achieved up to 30% higher data reduction ratio even over a combination of existing data reduction techniques.

References

[1] F. Chen *et al.* CAFTL: A Context-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In *Proc. USENIX Conf. File and Storage Technologies.*, 2011.

[2] S. Lee *et al.* Improving Performance and Lifetime of Solid-State Drives Using Hardware-Accelerated Compression. In *IEEE Trans. Consum. Electron.*, 57(4):1732–1739, 2011.

[3] G. Wu *et al.* Delta-FTL: Improving SSD Lifetime via Exploiting Content Locality. In *Proc. ACM European Conf. Comput. Sys.*, 2012.

[4] S. Lee *et al.* An Integrated Approach for Managing the Lifetime of Flash-Based SSDs. In *Proc. Design, Automation and Test in Europe.*, 2013.

[5] J. L. Nunez *et al.* The X-MatchPRO 100 Mbytes/second FPGA-Based Lossless Data Compressor. In *Proc. Design, Automation and Test in Europe.*, 2000.

[6] S. Lee *et al.* FlashBench: A Workbench for a Rapid Development of Flash-Based Storage Devices. In *Proc. Int. Symp. Rapid Sys. Prototyping.*, 2012.