DRACO: A Deduplicating FTL for Tangible Extra Capacity

Bon-Keun Seo, Joonwon Lee, Euiseong Seo

Abstract—The rapid random access of SSDs enables efficient searching of redundant data and their deduplication. However, the space earned from deduplication cannot be used as permanent storage because it must be reclaimed when deduplication is cancelled as a result of an update to the deduplicated data. To overcom this limitation, we propose a novel FTL scheme that enables the gained capacity to be used as permanent storage space for the file system layer. The proposed approach determines the safe amount of gained capacity that can be provided to the upper layer based on the compression rate prediction scheme. It then secures the required space by compressing cold data when capacity overflow occurs from broken deduplication. Our evaluation with a Git repository showed that the file system obtained approximately 69% additional capacity by the proposed scheme.

Index Terms—SSD, flash memory, deduplication, compression, FTL, file systems, over-provisioning

1 Introduction

Althugh the capacity of flash memory solid state disks (SSDs) has rapidly increased because of the improved density of flash memory cells and use of multi-level cells (MLCs), the capacity-to-cost rate of SSDs remains more than ten times higher than that of hard disks. Therefore, the capacity issue continues to be of primary concern to both SSD vendors and consumers.

Most large-volume data being housed in storage devices have a high degree of redundancy; moreover, deduplication of such redundant data by the host-side operating system can secure additional storage space [1]. However, host-side deduplication has major performance drawbacks associated with consuming processor and I/O bus resources.

The rapid random access of SSDs enables the implementation of a deduplication scheme at the flash translation layer (FTL) within SSDs [2], [3]. However, it is technically difficult to provide the space acquired from deduplication to file systems for permanently storing data because the extra capacity must be reclaimed when deduplication is cancelled by an update of the deduplicated data. This issue is more challenging for the SSD-side deduplication scheme than for the host-side counterpart. Therefore, previous SSD-side deduplication schemes have proposed the utilization of this gained capacity as temporary space for garbage collection or wearleveling [2], [3].

To address this limitation, this paper proposes a novel deduplicating FTL scheme named DRACO (deduplication of redundant data and compression of them) that can provide the deduplicated capacity as permanent storage space for the file system level. As shown in Figure 1, DRACO secures necessary space by compressing cold blocks, which contain infrequently accessed data, when capacity overflow occurs as a result of broken deduplication. The compressed data is delivered to the host system after decompression when they are requested. The compressed data are restored when sufficient surplus space is acquired by another round of deduplication.

- *B. Seo is with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Rep. of Korea, 305-701. E-mail: bkseo@calab.kaist.ac.kr*
- *J. Lee and E. Seo are with the College of Information and Communication Engineering, SungKyunKwan University, 440-746. E-mail: joonwon@skku.edu and euiseong@skku.edu*

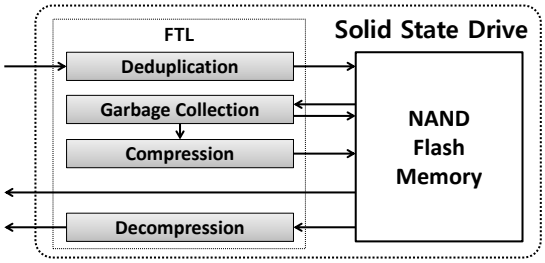

Fig. 1. DRACO FTL Data flows and operations.

The proposed approach requires advance knowledge of the compression rate of the stored data to determine the amount of capacity to provide to the file system. We found that the compression rate of the incoming data stream can be efficiently predicted with byte entropy [4]. The proposed scheme provides a safe amount of gained space considering the error distribution of the compression ratio prediction.

Commodity file systems exist that provide on-line resizing. Such file systems can utilize the provided space for storing data after resizing themselves according to capacity changes. The remaining surplus space, which is not provided to the file system layer, can still be utilized for garbage collection or wear-leveling, as outlined in existing research.

2 Deduplication

To find identical data, the SSD must calculate the content hash keys from incoming data chunks and look up a hash table using these keys. Multi-referenced chunks must be carefully handled to make every reference consistent. These operations incur significant performance drawbacks. We therefore designed the proposed deduplication scheme to be performance oriented.

Fortunately, SSD characteristics enable performance benefits from deduplication. NAND ash memory performs write operations 7 to 8 times slower than read operations. Therefore, reduction of ash memory writes by deduplication can mitigate deduplication overhead. In fact, previous studies have shown that deduplication can yield even better performance in some workloads [3], [2].

2.1 Address mapping

To minimize overhead, DRACO aligns the deduplication unit with the I/O operation unit, which is a ash memory page. A page is typically two to eight KB in size, which is the typical chunk size used by existing deduplication algorithms.

Because duplicated pages can exist anywhere in an SSD, a page-level FTL should be used to appropriately handle the multi-referenced mappings caused by deduplication. Therefore, DRACO employs a variation of the demand-based FTL (DFTL) scheme [5] for the address translation to map a logical block address (LBA) to a flash page address; this is called address mapping. As with DFTL, the mapping table is stored in the flash memory, and a part of the table is cached in DRAM to accelerate address translation.

The deduplication mechanism used in DRACO performs the same as DFTL for processing read requests because there are no changes in the processing path for a read request. Only write requests require additional operations for deduplication.

2.2 Content keys

Hash functions, such as SHA-1 or MD5, which have been commonly used in previous studies, are excessively heavy for embedded processors in an SSD. Calculating a SHA-1 content key from a page takes hundreds of microseconds, nearing the latency of reading a page from flash memory, even when the SSD features a SHA-1 encoding chip. Moreover, the keys in these algorithms do not comfortably fit in the small capacity SDRAM of an SSD. Therefore, deduplication of partial data has been suggested [3]; however, it is obvious that it harms the deduplication ratio. Therefore, byte-by-byte comparison after a weak-but-fast hash match can be more effective than using the aforementioned strong hash functions alone.

SSDs use an error correction code, such as CRC32, to detect and correct bit ips in the ash memory to guarantee data integrity. This code can be used as a content key, although it has higher probability of collision than SHA-1. The hash collision can be resolved by a byte-by-byte comparison of the pages in question. Based on this rationale, DRACO adopts CRC32 values as content keys to minimize the overhead of deduplication.

2.3 Content mapping table

The content mapping table manages the mapping between a content key and the corresponding ash page address. DRACO finds a ash page with a content key using the content mapping table. To expedite this searching operation, a hash structure, which uses the content keys as its hash keys, is applied. A bucket of the hash table is a set of mapping entries with the same hash key. Each bucket is pointed to by an entry in the external hash table.

The size of the content mapping table is as large as the size of the address mapping table of the FTL. This is because all valid ash pages have their own entries in the content mapping table, as they do in the address mapping table. Therefore, it should be stored in ash memory because it is unable to fit in the small DRAM of the SSD. Content keys are evenly dispersed over the table; their access pattern shows little locality because the keys are hashed values. This factor lowers the merit of caching them in DRAM.

When a page with new content arrives, the content mapping table must be updated. Because the table is located in the ash memory, this update produces an additional ash write operation, which takes significant time. To reduce the performance impact from table update operations, DRACO stores the content mapping table in a log-structured way. A bucket is typically much smaller than a ash page, which is the unit of the write request. Consequently, when an update in the bucket occurs, the updated bucket is written together with other updated buckets to comprise a whole page. This can significantly reduce the number of ash write operations caused by content mapping table management.

2.4 In-place garbage collection

Deduplication complicates garbage collection by the FTL because a ash page can be shared by multiple LBAs. Moreover, moving a ash page during garbage collection accompanies remapping of all relevant LBA entries. This is practically impossible because of the space and time complexity of discovering all referencing LBAs for a ash page. For this reason, DRACO adopts in-place garbage collection [6]; in-place garbage collection does not change the address of a flash page.

The in-place garbage collector copies valid pages from an erasure block to a clean block; it then discards invalid ones before erasing the block. To this end, it must be able to determine whether a page is valid. The DFTL uses a reverse mapping recorded in the out-of-band (OOB) area to check the validity of a page. However, DRACO can no longer use the reverse mapping because a page can be referenced by multiple LBAs. A straightforward solution, which keeps track of the reference counts for each page with a data structure located in DRAM, is impractical because that data structure would be too large.

Therefore, to check the validity of a ash page, DRACO maintains a bitmap in DRAM. When a page is allocated to an LBA, the corresponding bit in the bitmap is set to mark the page as valid. However, when a mapping between an LBA and a ash page is broken, the bit for the ash page is not touched because the page may be valid by mapping from other LBAs. To identify invalid pages, DRACO periodically reconstructs the validity bitmap with garbage collection. Initially, the bitmap is fully cleared. DRACO scans the address mapping table and sets the bit for a page of the mapping entry. When the scanning is complete, DRACO switches the new bitmap with the former one. In turn, the in-place garbage collector can identify invalid pages based on this current bitmap. If the page size is 4 KB, the validity bitmap will take only 0.003% of the SSD capacity.

3 Compression

The garbage collector in DRACO begins compressing pages when the number of free pages falls below the predefined threshold, C_{op}. Additionally, a decompression threshold exists that is larger than C_{op}. If the number of free pages is above C_{op}, the decompression of the compressed pages begins.

The garbage collector should choose victim erasure blocks that minimize the performance impact and maximize the acquired free space. To achieve this, both compression ratio prediction and hot-cold page separation are applied for victim selection.

The advance knowledge of data size after compression not only helps in selecting a victim block for compression, it also enables safe capacity management, as introduced in Section 4.

Because actual compression of data requires unacceptably heavy computation, we developed a solution based on byte

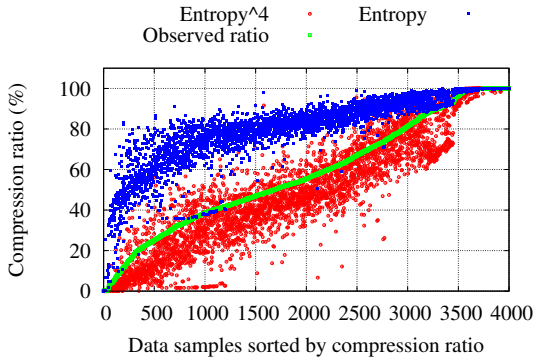

Fig. 2. Comparison between predicted and actual compression ratio.

entropy H [4]. Byte entropy of a data block has been used to approximate the compression ratio of data when using Huffman encoding. The byte entropy calculation routine runs very quickly in $O(n)$ time, which is well suited to the low-power embedded processor inside the SSD. However, in terms of the popular zip algorithm, the correlation between byte entropy and compression ratio becomes low because modern compression algorithms typically build a dictionary of frequently appearing strings to further reduce the size of the compressed data.

Based on several experiments, we empirically determined that H^4 shows a strong correlation to the zip algorithm algorithm compression ratio. Figure 2 shows the relationship obtained from a sample file set of diverse file types including text, multimedia, executables, and database files. The actual compression ratio of a sample was an average of 5.3% larger than H^4, and the standard deviation of the differences was 9.32%. Based on this observation, DRACO uses H^4 for prediction of the compression ratio of an erasure block.

Reading a compressed page requires immediate decompression of the page during the request handling, which also degrades read latency. Therefore, the victim selection must be able to classify cold pages, which are infrequently access pages, from hot pages.

DRACO keeps track of access counts for every erasure block. The access count unit is not a page; it is an erasure block. This is because the block-level access count will drastically compact the memory requirement for the access counters. Moreover, a significant spatial locality typically exists among pages in the same erasure block. Thus, pages in the same block are considered as sharing the same access frequency. The access count is halved with each iteration of garbage collection to favor the most recent accesses. A page is identified as a cold one when the access count of the page is below a threshold, which is easily determined by the average access count of all blocks.

The FTL compresses together a group of victim pages into a chunk to maximize the compression ratio [7]. A chunk is stored in contiguous flash memory pages. When a flash page is compressed, the address of the page must be appropriately changed to point to the location in the compressed chunk. Because the compression of a page inevitably changes its address, remapping of all relevant address translation entries must be followed. The issue of discovering all referencing LBAs for a compressed page again arises. Therefore, the address remapping is postponed until the garbage collection operation.

To efficiently identify and locate a compressed page, the addressing scheme should be modified accordingly. The location of a compressed page, which belongs to a chunk, can be expressed as the tuple of the page address of the chunk; it is the offset of the page in the chunk, and the compressed size of the page. This tuple is encoded and packed into a 64-bit entry of the address mapping table. The FTL can therefore identify whether the target page is compressed and locate it without additional data structures.

An in-memory table for remapping is used by the garbage collector to identify the page to remap during garbage collection. The entry of this table is created at each compression operation. The garbage collector removes an entry from the table after the remapping for the respective entry is complete.

The decompression delay during read operations of compressed pages can be fairly compensated by the reduced I/O latency because the number of actual pages to read is decreased by the compression [7].

4 Capacity Management

The capacity of the DRACO SSD frequently changes according to the duplicity of the stored data. Accordingly, the operating system should reflect the change to effectively utilize the acquired capacity. Some file systems or volume management systems, such as the logical volume manager (LVM), provide dynamic resizing of file system. Operating systems currently support sufficient functionality for dynamically resizing of the file systems. Based on these features, we propose the dynamic capacity management function for host-side operating systems.

The proposed method uses the device driver in the operating system to monitor capacity changes of the underlying SSD. The device driver invokes the resize routine when a capacity change occurs. A special command to efficiently monitor the capacity of the SSD should be added to the storage interface, such as SATA. The file system mounting routine registers a callback function, which enables the device driver to invoke for resizing.

The provided capacity of the SSD should remain in a safe range to avoid a catastrophic capacity shortage. The safe amount of capacity is defined as the maximum capacity that DRACO can accommodate by compressing all stored data. This amount evidently depends on the compression ratio of the stored data.

Assuming that the stored data is not at all compressed, the maximum available capacity of an SSD, C_{free}, can be defined as $C_{clean} + C_{invalid} + C_{uncomp} \times (1 - r_{comp})$ where C_{clean}, $C_{invalid}$, C_{uncomp} and r_{comp} are the number of clean pages, invalidated pages, valid pages and estimated average compression ratio, respectively.

Because the error of the compression ratio estimation follows a normal distribution, r_{comp} is recalculated with consideration of the allowable confidence level, which determines the probability of capacity shortage occurrence in case all data must be compressed. From the samples in Figure 2, we determined that $r_{comp} = avg(H^4) + 0.053 + 2.58 \times 0.0932$ with a confidence level of 99%.

Capacity shortage occurs when C_{free} becomes zero while the actual compresstion ratio after compressing all stored data is greater than r_{comp}. This occurs in extremely rare cases. However, such cases would be resolved by offloading to other storage devices, which is an ongoing research issue.

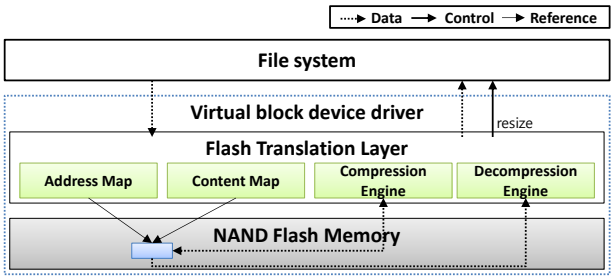

Fig. 3. DRACO FTL architecture.

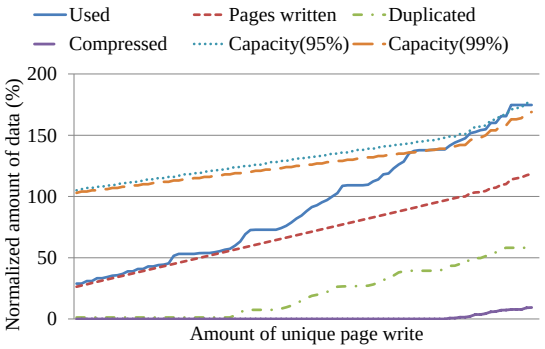

Fig. 4. Capacity changes in DRACO

The file system resizing overhead can be reduced by redesigning the file system. Most file system resize routines can easily expand their capacity by appending the additional capacity to the existing space. Shrinking the file system, however, is a much heavier process because out-of-bound data should be migrated. This can be remedied by the modification of file systems such that it becomes aware of the extra space and manages the acquired space considering the shrinking operation. We have earmarked this issue for future research.

5 Evaluation

We implemented an SSD simulator as a virtual block device in the Linux kernel. The simulator is composed of the logical units, as shown in Figure 3. The simulated FTL implements the deduplication and selective compression mechanism of DRACO. The virtual block device driver provides the capacity management interface to the file system.

We conducted an experiment in which a number of the Linux kernel source Git repository, which contains numerous Linux kernel versions, was copied to the simulated SSD. The Linux source tree contains approximately 8% of duplicated data; 35% of flash pages were deduplicated between each version of linux source trees.

The provided capacity of the simulated SSD increased as data was duplicated with respect to the capacity formula in Section 4. We plotted the expected capacity in the confidence level of both 95% and 99%, as shown in Figure 4.

In this experiment, the file system size continually grew according to the available SSD capacity. DRACO began compressing data when the capacity shortage occurred. The expected capacity tended to grow quickly because the compression ratio was conservatively estimated; accordingly, the compression operation supplied more free space than expected. At the end of the experiment, it was possible to provide 69% greater capacity than the original SSD capacity with a confidence level of 99%.

To analyze the performance impact by DRACO, we conducted the same experiment with the DFTL simulator. The SSD capacity for the original DFTL was twice that of the one used in the DRACO experiment. The comparison showed that DRACO is 5% faster than the DFTL because DRACO successfully reduced many flash memory writes by deduplication.

6 Conclusion

In previous research, a few deduplicating FTLs that exploit the rapid random access and powerful embedded systems of SSDs have been proposed. However, the extra capacity gained from deduplication cannot be used for file systems because a capacity shortage may occur due to the copy-on-write operations induced by an update of deduplicated data. Therefore, to date, the acquired capacity has been used only for internal operations of SSDs.

To address this issue, we have proposed a novel FTL that can provide the extra capacity acquired from deduplication to the upper layer file systems by resolving the capacity overow issue. The proposed FTL provides a safe amount of extra capacity based on the dynamic compression rate prediction of the stored data. It secures space required for resolving capacity overow by compressing cold data.

The proposed scheme is expected to provide a substantial amount of extra capacity for the file system layer with negligible performance overhead. However, capacity shortage may still occur in the proposed scheme when almost all data must be compressed while the compression rate prediction error simultaneously exceeds the predefined confidence interval. This would be an extremely rare case; nevertheless, we are conducting research on capacity ofoading to remote storage devices, which is expected to eliminate this risk.

References

[1] D. T. Meyer and W. J. Bolosky, "A Study of Practical Deduplication," *ACM Transactions on Storage*, vol. 7, no. 4, pp. 1–20, Jan. 2012.

[2] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging Value Locality in Optimizing NAND Flash-based SSDs," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies*. San Jose, CA: USENIX Association, 2011, pp. 91–103.

[3] 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 *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, ser. FAST'11. San Jose, CA: USENIX Association, 2011, pp. 77–90.

[4] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[5] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *ASPLOS '09: Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, 2009, pp. 229–240.

[6] Y. Zhang, L. P. Arulraj, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "De-indirection for Flash-based SSDs with Nameless Writes," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*. San Jose, CA: USENIX Association, Feb. 2012, pp. 1–16.

[7] Y. Park and J.-s. Kim, "zFTL: Power-Efficient Data Compression Support for NAND Flash-based Consumer Electronics Devices," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1148–1156, Aug. 2011.