Deduplication in SSDs: Model and Quantitative Analysis

Jonghwa Kim
Dankook University, Korea
zcbm4321@dankook.ac.kr

Choonghyun Lee
Massachusetts Institute of Technology, USA
chl@csail.mit.edu

Sangyup Lee
Dankook University, Korea
xiphy@dankook.ac.kr

Ikjoon Son
Dankook University, Korea
ikjoon@dankook.ac.kr

Jongmoo Choi
Dankook University, Korea
choijm@dankook.ac.kr

Sungroh Yoon
Korea University, Korea
sryoon@korea.ac.kr

Hu-ung Lee
Hanyang University, Korea
oihtoto@hanyang.ac.kr

Sooyong Kang
Hanyang University, Korea
sykang@hanyang.ac.kr

Youjip Won
Hanyang University, Korea
yjwon@hanyang.ac.kr

Jaehyuk Cha
Hanyang University, Korea
chajh@hanyang.ac.kr

***Abstract*—In NAND Flash-based SSDs, deduplication can provide an effective resolution of three critical issues: cell lifetime, write performance, and garbage collection overhead. However, deduplication at SSD device level distinguishes itself from the one at enterprise storage systems in many aspects, whose success lies in proper exploitation of underlying very limited hardware resources and workload characteristics of SSDs. In this paper, we develop a novel deduplication framework elaborately tailored for SSDs. We first mathematically develop an analytical model that enables us to calculate the minimum required duplication rate in order to achieve performance gain given deduplication overhead. Then, we explore a number of design choices for implementing deduplication components by hardware or software. As a result, we propose two acceleration techniques: *sampling-based filtering* and *recency-based fingerprint management*. The former selectively applies deduplication based upon sampling and the latter effectively exploits limited controller memory while maximizing the deduplication ratio. We prototype the proposed deduplication framework in three physical hardware platforms and investigate deduplication efficiency according to various CPU capabilities and hardware/software alternatives. Experimental results have shown that we achieve the duplication rate ranging from 4% to 51%, with an average of 17%, for the nine workloads considered in this work. The response time of a write request can be improved by up to 48% with an average of 15%, while the lifespan of SSDs is expected to increase up to 4.1 times with an average of 2.4 times.**

I. Introduction

SSDs are rapidly being integrated into modern computer systems, getting spotlight as potentially next generation storage media due to a high performance, low power, small size and shock resistance. However, SSDs are failing to provide uncompromising reliability of data due to a short lifespan and increased error rate with aging, which is the major roadblock to be accepted as reliable storage systems in data centric computing environments despite of many superb properties [11]. In this paper, we argue that deduplication is a viable solution to enhancing the reliability of SSDs with carefully devised acceleration techniques.

Data deduplication is being widely adopted in various archival storages and data centers due to its contribution to storage space utilization and IO performance by reducing write traffic [21], [30], [37], [35], [34]. Recently, a number of researches from industry [7] as well as from academia [16], [23] have proposed to employ deduplication techniques in SSDs.

In addition to the reduction of write traffic, deduplication in SSDs provides other appealing advantages. First, while conventional storage systems require an additional mapping mechanism to identify the location of duplicate data for deduplication, SSDs already have a mapping table managed by a software layer, called FTL (Flash Translation Layer) [22], and give a chance to implement deduplication without paying any extra mapping management overhead. Second, the space saved by deduplication can be utilized as the over-provisioning area, leading to mitigating the garbage collection overhead of SSDs. It is reported that when garbage collection becomes active, the entire system freezes till it finishes (for a few seconds at least) [4]. This phenomenon is one of the most serious technical problems which modern SSD technology needs to address. Third, the reduction of write traffic and the mitigation of the garbage collection overhead eventually lowers the number of erasures in Flash memory, resulting in the extended cell lifetime. The major driving force in Flash industry is cost per byte. Flash vendors focus their efforts on putting more bits in a cell, known as MLC (Multi Level Cell), and on using finer production process such as 20 nm process. However, this trend deteriorates the write/erase cycle of Flash memory, which decreased from 100,000 to 5,000 or less [23]. Also, the bit error rate of Flash increases sharply with the number of erasures [14], [20]. In these situations, deduplication can be an effective and practical solution to improving the lifespan and reliability of SSDs.

Despite all these benefits, there exist two important technical challenges which need to be addressed properly for deduplication in SSDs. The first one is about the deduplication overhead, especially under the condition of limited resources. In general, commercial SSDs contain low-end CPUs such as ARM7 or ARM9 with small main memory to cut down

978-1-4673-1747-4/12/$31.00

production costs. This environment quite differs from that of servers and archival storages, demanding distinct approaches and techniques in SSDs. The second challenge is about the deduplication ratio. Are there enough duplicate data in SSD workloads?

To investigate these issues, we design a deduplication framework for SSDs. It consists of three components each of which forms an axis of modern deduplication techniques: fingerprint generator, fingerprint manager, and mapping manager. Also, we suggest an analytical model that can estimate the minimum duplication rate for achieving marginal gain in I/O response time. Finally, we propose several acceleration techniques, namely, *SHA1 hardware logic, sampling-based filtering* and *recency-based fingerprint management*.

Our proposed SHA-1 hardware logic and sampling based filtering are devised to address the fingerprint generator overhead. One of the important decisions to be made in designing SSDs is to choose between hardware and software implementations of each building block. We explore two approaches, one is a hardware based implementation, that is the SHA-1 hardware logic, and the other is a software based one, that is the sampling based filtering. Then, we analyze the tradeoffs between the two approaches in terms of performance, reliability and cost.

The recency-based fingerprint management scheme is intended to reduce the fingerprint manager overhead under the limited main memory of SSDs. We examine several SSD workloads with various attributes such as recency, frequency and IRG (Inter-Reference Gap) and find out that duplicate data in SSDs show strong temporal locality. This observation triggers us to design the scheme that maintains the recently generated fingerprints only with simple hash-based fingerprint lookup data structures.

We also discuss how to make an efficient integration of page sharing scheme of deduplication with existing FTLs. The introduction of deduplication in SSDs changes the mapping relation of FTL, from 1-to-1 into n-to-1. This change makes the mapping management complicated, especially for garbage collection to reclaim invalidated pages. Based on the characteristics of SSD workloads, we investigate various implementation choices for n-to-1 mapping managements, including a hardware-assisted management.

The proposed deduplication framework has been implemented on an ARM7-based commercial OpenSSD board [28]. Also, to evaluate the deduplication effects more quantitatively with diverse hardware and software combinations, we make use of two supplementary boards, a Xilinx Virtex6 XC6VLX240T FPGA board [10] and an ARM9-based EZ-X5 embedded board [3]. The Xilinx board is used for implementing the SHA-1 hardware logic and for assessing its performance while the EZ-X5 board is utilized for analyzing the efficiency of the sampling-based filtering on various CPUs.

Experimental results have shown that our proposal can identify 4∼51% of duplicate data with an average of 17%, for the nine workloads which are carefully chosen from Linux and Windows environments. The overhead of the SHA-1 hardware logic is around 80us, leading to improving the latency of write requests up to 48% with an average of 15%, compared with the original non-deduplication SSDs. We also have observed that, in SSDs equipped with ARM9 or higher capability CPUs, the sampling-based filtering can provide comparable performance without any extra hardware resources for deduplication. In terms of reliability, deduplication in SSDs can expand the lifespan of SSDs up to 4.1 times with an average of 2.4 times.

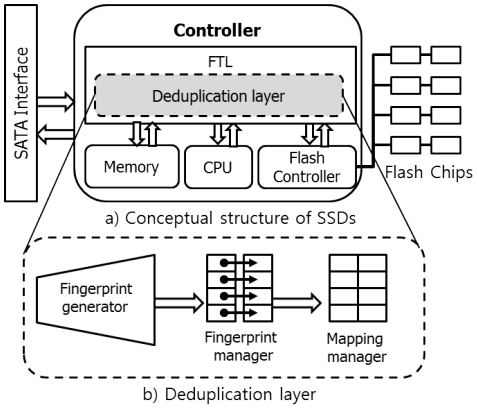


Fig. 1. Deduplication framework in SSDs

The rest of this paper is organized as follows. In the next section, we describe the deduplication framework for SSDs. The analytical model is presented in Section 3. In Section 4, we discuss the design choices of the fingerprint generator and propose the SHA-1 hardware logic and sampling based filtering. The recency-based fingerprint manager and mapping management for deduplication are elaborated in Section 5 and 6, respectively. Performance evaluation results are given in Section 7. Previous studies related to this work are examined in Section 8, and finally, a summary and the conclusion are presented in Section 9.

II. Deduplication Framework

Figure 1 shows the internal structure of SSDs and the deduplication framework designed in this paper. The main components of SSDs are an SATA host interface, an SSD controller, and an array of Flash chips. The SSD controller consists of embedded processors, DRAM (and/or internal SRAM), flash controllers (one for each channel) and ECC/CRC unit.

The basic element of a Flash chip is a cell which can contain one bit (Single-Level Cell) or two or more bits (Multi-Level Cell). A page consists of a fixed number of cells, e.g, 4096 bytes for data and 128 bytes for OOB (Out-of-Band) area [23]. A fixed number of pages form a block, e.g, 128 pages. There are three fundamental operations in NAND Flash memory, namely read, write, and the erase operations. Read and write operations are performed by a unit of page, whereas the erase operation is performed by a unit of block.

Flash memory has several unique characteristics such as the erase-before-write and a limited number of program/erase cycles. To handle these characteristics tactfully, SSDs employ a software layer, called FTL (Flash Translation Layer), which provides the out-of-place update and wear-leveling mecha-

nism. For the out-of-place update, FTL supports an address translation mechanism to map the logical block address (LBA) with physical block address (PBA) and a garbage collection mechanism to reclaim the invalid (freed) space. For the wear-leveling, FTL utilizes various static/dynamic algorithms, trying to distribute the wear out of blocks as evenly as possible.

We design a deduplication layer on FTL. It consists of three components, namely, fingerprint generator, fingerprint manager, and mapping manager as shown in Figure 1 (b). The fingerprint generator creates a hash value, called fingerprint, which summarizes the content of written data. The fingerprint manager manipulates generated fingerprints and conducts fingerprint lookups for detecting deduplication. Finally, the mapping manager deals with the physical locations of duplicate data.

A. Fingerprint Generator

One of the design issues for the fingerprint generator is the size of chunk, that is, the unit for deduplication. There are two approaches to this issue: fixed-sized chunking and variable-sized chunking. The variable-sized chunking can provide an improved deduplication ratio by detecting duplicate data at different offsets [31]. However, the size of write requests observed in SSDs are integral multiples of 512 bytes (usually 4KB) and the requests are re-ordered by various disk scheduling policies, diluting the advantages of the variable-sized chunking. Hence, we use the fixed-sized chunking in this study. We configure 4KB as the default chunk size and analyze the effects of different chunk sizes on the deduplication ratio.

Another design issue is about which cryptographic hash function to be used for deduplication. The SHA-1 and MD-5 are used popularly in existing deduplication systems since they have collision-resistant properties [23]. In this study, we choose the SHA-1 hash function that generates a 160-bit hash value from 4KB data [15]. How to implement the SHA-1 affects greatly the deduplication overhead and we explore two approaches, hardware-based and software-based approaches, which are discussed in Section 4 in details.

B. Fingerprint Manager

The design issue related to the fingerprint manager is how many fingerprints need to be maintained. The traditional archival storages and servers keep all fingerprints for deduplication (a.k.a. full chunk index [30]). However, SSDs have a limited main memory (for instance, the OpenSSD system used in this study has 64MB DRAM). Furthermore, most of this space is already occupied by various data structures such as a mapping table, write buffers, and FTL metadata.

To reflect the limited main memory constraint, we decide to maintain only part of fingerprints that have higher duplication possibility. Now the question is which fingerprints have such possibility. Our analysis of SSD workloads shows that the recency is a good indicator to estimate the possibility, leading us to design a scheme that maintains recently generated fingerprints only. This choice also enables the scheme to be implemented with simple and efficient data structures for fingerprint lookups. More details of the scheme will be elaborated in Section 5.

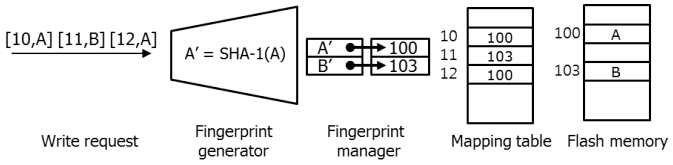


Fig. 2. Deduplication example

C. Mapping Manager

To deal with the physical location of duplicate data, the mapping manager makes use of the mapping table supported by FTL. According to the mapping granularity, the mapping table can be classified into three groups: page-level mapping, block-level mapping and hybrid mapping [29]. Since deduplication requires the mapping capability with the unit of chunk, we design a page-level mapping based FTL with the page size of 4KB.

Figure 2 displays a deduplication example and the interactions among the fingerprint generator, fingerprint manager, and mapping manager. Assume that three write requests, represented as [10, A], [11, B] and [12, A], are arrived in sequence ([x, y] denotes a write request with a logical block address x and content y). Then, the fingerprint generator creates fingerprints, which are passed into the fingerprint manager to find out whether they are duplicates or not (the fingerprints of A and B are denoted as A and B, respectively in Figure 2).

In this example, we do not detect any duplicate for the first two write requests. Hence, we actually program the requests into Flash memory (assume that they are programmed in pages 100 and 103, respectively). After that, the physical block addresses are inserted into both the mapping table and the fingerprint manager. For the third write request, that is [12, A], duplication is detected in the fingerprint manager and only the mapping table is updated without programming.

This example demonstrates that the mapping table used for FTL can be exploited effectively for deduplication. However, when garbage collection is involved, the scenario gets complicated. This issue will be discussed further in Section 6.

III. Model and Implication

In this section, we present an analytical model for estimating the deduplication effect on performance. Also, we discuss the implication of the model, especially in terms of the duplication rate and deduplication overhead.

In the original non-deduplication SSDs, a write request is processed in two steps, namely programming the requested data into Flash memory and updating its mapping information. Therefore, we can formulate the write latency as follows:

$$Write_{latency} = FM_{program} + MAP_{manage} \quad (1)$$

where $FM_{program}$ is the programming time on Flash memory and MAP_{manage} is the updating time of the mapping table.

On the other hand, when we apply deduplication in SSDs, the write latency can be expressed as follows:

$$\begin{aligned} Write_{latency} = & (FP_{generator} + FP_{manage} + MAP_{manage}) \\ & \times Dup_{rate} + (FP_{generator} + FP_{manage} \\ & + MAP_{manage} + FM_{program}) \\ & \times (1 - Dup_{rate}) \end{aligned} \quad (2)$$

where $FP_{generator}$ is the fingerprint creation time, FP_{manage} is the lookup time in the fingerprint manager, and Dup_{rate} is the ratio between the duplicate data and total written data. The equation 2 means that, when a write request is detected as duplicate, it pays the $FP_{generator}$, FP_{manage} and MAP_{manage} overheads. Otherwise, it pays the additional $FM_{program}$ overhead.

From the two equations, we can estimate the expected performance gain of deduplication in SSDs. Specifically, deduplication can yield the performance gain on the condition that equation 2 is smaller than equation 1. The condition can be formulated as follows:

$$Dup_{rate} > \frac{FP_{generator} + FP_{manage}}{FM_{program}} \quad (3)$$

Equation 3 indicates that, when the duplication rate is larger than the ratio of the deduplication overhead (both the fingerprint generation and fingerprint management overheads) to the Flash memory programming overhead, we can enhance the write latency in SSDs. In other words, it suggests the required minimum duplication rate for obtaining the marginal performance gain.

Note that, in SSDs, the write latency actually contains one additional processing time, that is the garbage collection time. During the handling of write requests, FTL triggers garbage collection when the available space goes below a certain threshold value [22]. The garbage collection mechanism consists of three steps: 1) selecting a victim block, 2) copying valid pages of the selected block and updating mapping, 3) erasing the block and making it as a new available block. Hence, the garbage collection time is directly proportional to the average number of valid pages of blocks, which, in turn, has a positive correlation to the storage space utilization [25]. Since deduplication can reduce the utilization, the garbage collection time in equation 2 is smaller than that in equation 1. Therefore, equation 3 also holds if we take into account the garbage collection overhead together.

To grasp the implication of equation 3 more intuitively, we plot Figure 3, presenting the minimum duplication rate under the various deduplication overheads. In the figure, we select four values, 200, 800, 1300, and 2500 us, as the representative program times of Flash memory, reported in previous papers and vendor specifications [22], [28].

From Figure 3, we can observe that the minimum duplication rate decreases as the deduplication overhead decreases or as the program time becomes longer. For instance, in the case when the program time is 1300 us (which is the OpenSSD case used in our experiments), we require more

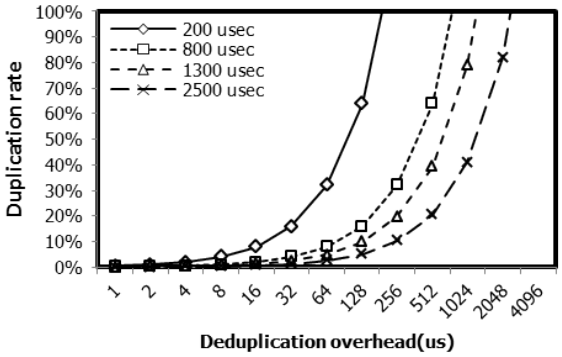


Fig. 3. Minimum duplication rate for achieving performance gain

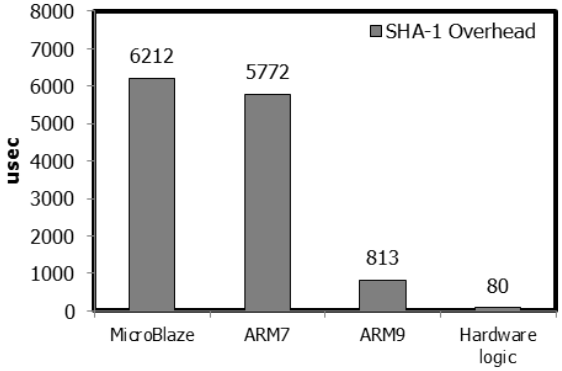


Fig. 4. SHA-1 processing time on various CPUs

than 16% of duplication rate for obtaining the performance gain when the deduplication overhead is 256 us. If we reduce the deduplication overhead from 256 us to 128 us, the required minimum duplication rate becomes 8%. Now the question is how to reduce the fingerprint generation and management overhead.

IV. SHA-1 Hardware Logic and Sampling based Filtering

In this section, we first measure the fingerprint generation overhead on various embedded CPUs, widely equipped in commercial SSDs. Then, we design two acceleration techniques which are respectively hardware-based and software-based techniques.

A. SHA-1 Processing Overhead

To quantify the SHA-1 overhead, we measured the SHA-1 processing time on three embedded CPUs, 150MHz MicroBlaze [10], 175MHz ARM7 [28] and 400MHz ARM9 [3], as shown in Figure 4 (actually, it also contains the SHA-1 processing time on a hardware logic, which will be discussed in the Section 4.2). The results reveal that the SHA-1 processing time is nontrivial, much bigger than our initial expectation. From the analytical model presented in Figure 3, we can find out that applying deduplication on SSDs equipped with ARM 7 or MicroBlaze CPU always degrades the write latency since the required minimum duplication rate for obtaining the marginal gain is higher than 100%.

This observation drives us to look for other acceleration techniques. There is a broad spectrum of feasible techniques, ranging from hardware-based to software-based approaches. In this study, we explore two techniques, SHA-1 hardware logic and sampling-based filtering.

B. Hardware-based Acceleration: SHA-1 Hardware Logic

As hardware-based acceleration, we design a SHA-1 hardware logic on Xilinx Virtex6 XC6VLX240T FPGA [10], as depicted in Figure 5. It consists of five modules: main control unit that governs the logic on the whole, Data I/O Control unit for interfacing the logic with CPU, Dual Port BRAM for storing 4KB data temporary, SHA-1 Core for generating fingerprints using the standard SHA-1 algorithm [15], and hash comparator that examines two fingerprints and returns whether they are the same or not. We use Verilog HDL 2001 for RTL coding [8].

The SHA-1 processing time on the hardware logic is measured as 80 us on average, as presented in Figure 4. With this value, we can have more room for the performance gain by using deduplication as observed in Figure 3. For instance, assuming that the Flash memory program time is 1300 us, the improvement of write latency is expected when the duplication rate is larger than 5%. Note that the hardware logic gives another optimizing chance by conducting the fingerprint generation and other FTL operations such as mapping management and Flash programming in a pipelined style.

C. Software-based Acceleration: Sampling-based Filtering

Although utilizing the SHA-1 hardware logic gives an opportunity to enhance performance, it needs additional hardware resources that increase production costs. Also, from the Figure 3 and 4, we can infer that ARM 9 or higher capability CPUs have a potential to yield performance improvements based only on software approaches. To investigate this potential, we design the sampling-based filtering technique that selectively applies deduplication for write requests according to their duplicate possibilities.

The technique is motivated by our observation about the characteristics of SSD workloads, represented in Figure 6. We choose nine applications as representative SSD workloads, which will be explained in details in Section 7. In the figure, x-axis is the IRG (Inter-Reference Gap) of duplicate writes while

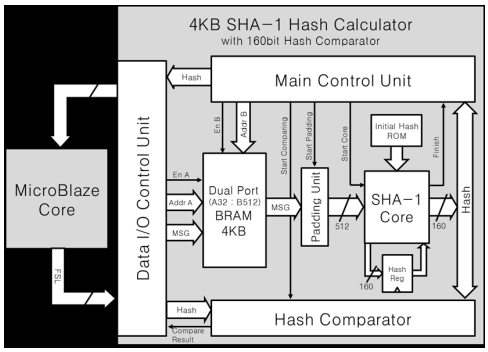


Fig. 5. SHA-1 hardware logic

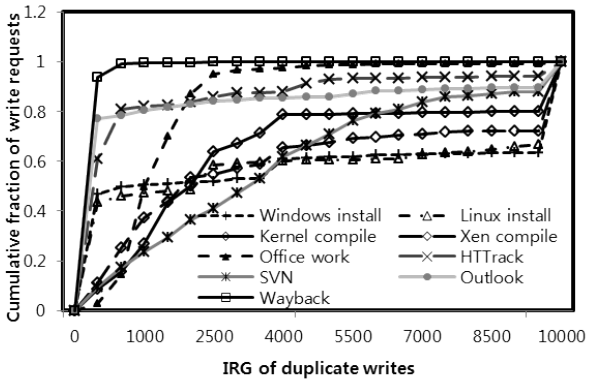


Fig. 6. Characteristics of SSD workloads: Inter-Reference Gap of duplicate writes

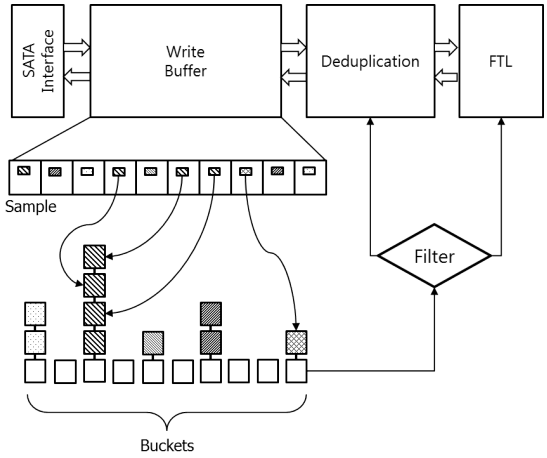


Fig. 7. Details of the sampling based filtering technique

y-axis is the cumulative fraction of the number of writes that have the related IRG. The IRG is defined as the time difference between successive duplicate writes, where time is a virtual time that ticks at each write request [33].

From the figure, we can categorize the applications into two groups. The first group includes windows install, linux install, outlook, HTTrack and wayback. In this group, most of the IRGs of duplicate writes are less than 500 while others are distributed uniformly from 500 to infinite. For instance, almost 95% of the wayback workload and around 80% of outlook and HTTrack workloads are less than 500. In the second group, including kernel compile, xen compile, office and SVN, the fraction of writes increases incrementally as IRG increases. Note that even in this group, more than 60% of IRGs are less than 4,000 and, after that point, the slope becomes almost flat except the SVN workload. This observation drives us to design the sampling-based filtering technique.

Figure 7 demonstrates how the sampling-based filtering technique works. It makes use of a write buffer in SSDs that lies between the SATA interface and FTL. SSDs utilizes a portion of DRAM space as a write buffer for exploiting caching effects to reduce the number of Flash programming operations [26]. In our experimental OpenSSD, the size of

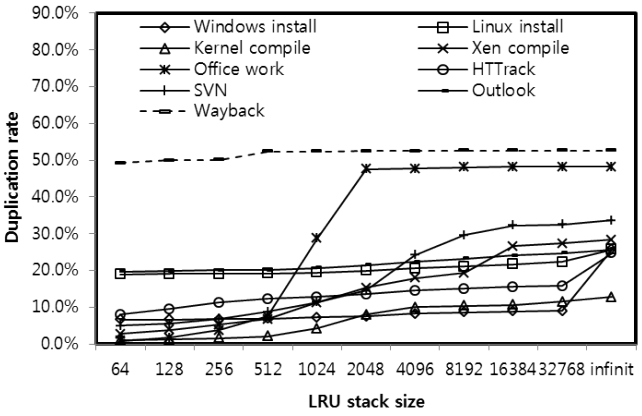


Fig. 8. Characteristics of SSD workloads: Recency and duplication rate

a write buffer is 32MB, maintaining 8,000 numbers of 4KB pending write requests at maximum.

When a new write request is arrived in the write buffer, the technique first samples p-byte data from a randomly selected offset of q. In the current study, we set p and q to 20 and 512 bytes, respectively. Other settings have shown that the results of this technique are insensitive to the values of p and q on the condition that p is larger than 20. Then, it classifies write requests into buckets using p bytes as a hash index, as shown in Figure 7. Hence, the writes that have the same p-byte data go into the same bucket. Finally, when a write request leaves from the write buffer, the technique does not apply deduplication for the writes that are classified into the bucket holding only one request. This decision is based on the observation in Figure 6 that the duplicate writes occur again during the short time intervals. We expect that the technique can reduce the fingerprint generation overhead greatly by filtering out non-duplicate writes while supporting a comparable duplication rate.

V. Recency-based Fingerprint Management

In the previous section, we have discussed two acceleration techniques, one is hardware-based approach and the other is software-based one, for reducing the fingerprint generation overhead. The next question is how to reduce the fingerprint management overhead.

To devise an efficient fingerprint management scheme, we examine the characteristics of SSD workloads with a viewpoint of the LRU stack model [17]. In this model, all written pages are ordered by the last accessed time in the LRU stack and each position of the stack has a stationary and independent access probability. The LRU stack model assumes that the probability of the higher position of the stack is larger than that of the lower position. In other words, a page accessed more recently has a higher probability to be accessed again in the future.

Figure 8 shows the duplication rate under different LRU stack sizes for the nine SSD workloads. In the figure, x-axis is the LRU stack size, which is the number of recently generated fingerprints maintained in the fingerprint manager, and y-axis is the measured duplication rate under the corresponding LRU stack size. It shows that SSD workloads have a strong temporal locality. Especially, for the Linux install, kernel compile, outlook and wayback workload, we can detect most of all duplicate data using the LRU stack size of 64 (in other words, we keep 64 recently generated fingerprints only). For most of the workloads, when the stack size is larger than 2048, we can obtain a duplication rate comparable to the full fingerprints management case.

The observation in Figure 8 guides us to design the recency-based fingerprint management scheme. It maintains recently generated fingerprints only, rather than managing all generated fingerprints. In this study, we configure the number as 2048. Also, considering the CPU/memory constraints of SSDs, we employ efficient data structures for the partial fingerprints managements: a doubly linked list for maintaining LRU orders and two hashes, one using a fingerprint value as a hash key and the other using a physical block address as a hash key. The total DRAM space required for these data structures becomes 2048 entries * 40 bytes per entry (20 bytes for a fingerprint value, 4 bytes for a physical block address, 8 bytes for the LRU list, 8 bytes for two hash lists). Finally, we decide to keep fingerprints on DRAM only, not storing/loading into/from Flash memory during power-off/on sequences.

VI. Effects of Deduplication on FTL

The conventional FTLs maintain a mapping table for translating logical block addresses (LBAs) into physical block addresses (PBAs) as shown in Figure 2. Besides, to lookup LBAs from PBAs during garbage collection, FTLs keep another inverted mapping information for translation between PBAs to LBAs. This information can be managed either by a centralized inverted mapping table or by a distributed manner using the OOB (Out-of-Band) area of each physical page.

Integrating deduplication on FTL raises a new challenge since it changes the mapping relation between LBAs and PBAs, from 1-to-1 to n-to-1. For instance, from Figure 2, we can see that two LBAs (10 and 12) are mapped with one PBA (100). The n-to-1 mapping does not incur any problem during the normal read and write requests handling. However, when garbage collection is involved, the situation becomes complicated. For instance, again from Figure 2, assume that the data A is copied into page 200 during garbage collection. Then, the two entries (10 and 12) related to the copied page need to be identified in the mapping table and their values should be modified as 200. In other words, we need to update all entries associated to copied pages.

To alleviate the complication, Chen *et al*. proposed a two-level indirect mapping structure and metadata pages [16]. Their method makes use of two mapping tables, primary and secondary mapping tables. For the non-duplicate page, it locates the PBA for a LBA through the primary mapping table, as the conventional FTLs do. However, for the duplicate page, a LBA is mapped into a VBA (Virtual Block Address) through the primary mapping table, which, in turn, is mapped into a PBA through the secondary mapping table. This separation

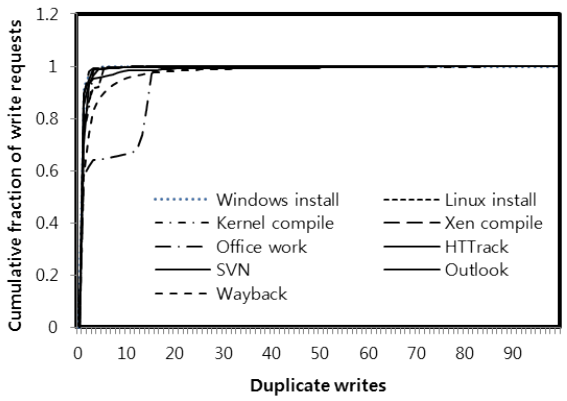


Fig. 9. Characteristics of SSD workloads: Frequency of duplicate writes

enables to update only one entry in the secondary mapping table during garbage collection without searching all entries in the primary mapping table. The metadata pages play a role as the inverted mapping table.

On the other hand, Gupta *et al*. took a different approach [23]. It uses a single-level mapping structure, called LPT (Logical Physical Table), like the conventional FTLs. Also, it employs an inverted mapping table, called iLPT (inverted LPT), that stores translation between a PBA to the list of LBAs that can keep more than one LBA if the PBA contains duplicate data. Using the iLPT, it can identify and update all entries of the LPT that are mapped into the copied page during garbage collection.

There are several tradeoffs between two approaches. The Chen's approach pays one extra lookup operation in the secondary table for duplicate pages during the normal read/write requests handling. On the contrary, the Gupta's approach may perform several mapping updates for a copied page during the garbage collection processing while conducting always one update in the Chen's approach. The worst count of updates is the maximum number of writes on duplicate data. In term of memory footprints, the two approaches require additional DRAM space, one for the secondary mapping table and the other for maintaining two or more LBAs in iLPT, whose size depends on the duplicate rate and the frequency of writes on duplicate data.

To estimate the tradeoffs more quantitatively, we measure the frequency of duplicate writes for the nine SSD workloads, as depicted in Figure 9. In the figure, x-axis represents a PBA that contains duplicate data and y-axis is the frequency, that is the number of writes, on the corresponding duplicate data. The results show that most of writes on duplicate data is less than or equal to 3, meaning that, in most cases, the number of LBAs updated per a PBA during garbage collection is at most 3. This observation leads us to adopt the Gupta's approach in this study, although the Chen's approach also goes well with our proposed deduplication framework. We design our deduplication framework carefully so that it can be integrated with any existing page-level FTLs.

One concern about the page-level FTL is that the sizes of the mapping and inverted mapping tables are too large to fit the limited DRAM space of SSDs. To overcome this obstacle, we can apply the demand-based caching, proposed in [22]. However, caching causes another problem, which is a sudden power-failure recovery. In this case, we can employ a well-known approach such as using a hardware superCap [2] or battery-backed RAM [23]. The caching and power-failure recovery issues are orthogonal to the deduplication issues.

Our framework currently adopts a simple and commonly used algorithm for garbage collection. It triggers garbage collection when available space goes below a certain threshold (GC_threshold). In this experiment, the default value of GC_threshold is set to 80%. When triggered, our algorithm first selects a victim block based on the cost-benefit analysis proposed in [25]. Then, the algorithm copies valid pages of the selected block into other clean pages and updates mapping information. Finally, the algorithm erases the block and converts it as available space.

Here, we would like to discuss that deduplication gives an opportunity to improve the garbage collection efficiency. One method for improving the efficiency is reducing the number of copies of valid pages during garbage collection. To achieve this, valid and invalid pages need to be distributed into different blocks so that garbage collection can select a victim block whose pages are mostly invalid [12]. For this purpose, FTL tries to detect hot and cold data and manages them into different blocks. Data modified frequently is defined as hot data while others as cold data. Hence, most of pages in the block for hot data become invalidated while the block for cold data contains valid pages in most case. Note that duplicate data has a feature that is not invalidated frequently. Hence, the separation of duplicate data from unique data can enhance the garbage collection performance.

In addition, deduplication can be exploited usefully for wear-leveling. Since a Flash memory has a limited number of erase counts, it is important to evenly distribute the wear-out of each block. One of the popularly used wear-leveling algorithms is swapping data in the most erased block with those in the least erased one [14]. The rationale behind this algorithm is that the data in the least erased block is cold data, which prevent the block to be selected as a victim block during garbage collection. When we locate duplicate data, identified by deduplication, on the most erased block, we can improve the wear-leveling efficiency.

Finally, we investigate the feasibility of hardware/software co-design for mapping managements. Deduplication in SSDs requires two different tables, one is the mapping table for LBA to PBA translation and the other is the inverted mapping table for PBA to LBAs translation. Our implementation study has uncovered that maintaining the consistency between the two tables makes the deduplication framework quite complicated for applying locking mechanisms and for considering various exceptional cases for power-failure recovery.

This troublesome drives us to explore an alternative. It is a kind of hardware/software co-design that makes a mapping

table, managed by software, as simple as possible while searching LBAs related to PBA during garbage collection is carried out by hardware such as a memory-searching co-processor. Some commercial SSDs have already equipped such a hardware facility. For instance, OpenSSD provides a hardware accelerator, called as memory utility, that is used for improving common memory operations such as initializing a memory region with a given value or searching a specific value from a memory region [28]. However, the current version of memory utility can cover at most 32KB memory region at a time, which is too small to manage the mapping table. We are currently extending the memory utility that can search several memory regions in parallel and exploit a Bloom filter to skip over uninterested memory regions quickly [13]. We believe that this approach can improve not only memory footprints but also software dependability.

VII. Performance Evaluation

In this section, we first describe the experimental setup and workloads. Then, we present the performance and reliability evaluation results including the duplication rate, write latency, garbage collection overhead and expected lifespan of SSDs.

A. Experimental Environments

We evaluated our proposed deduplication framework on a commercial SSD board, called OpenSSD [28]. It consists of 175MHz ARM7 CPU, 64MB DRAM, SATA 2.0 host interface, and Samsung K9LCG08U1M 8GB MLC NAND Flash packages [6]. The package is composed of multiple chips and each chip is divided into multiple planes. A plane is further divided into blocks which, in turn, divided in pages. The typical read and program times for a page are reported as 400 us and 1300 us, respectively, while the erase time for a block is reported as 2.0 ms [6].

Unfortunately, the OpenSSD does not have FPGA logic. So, we utilize a supplementary board, that is a Xilinx Virtex6 XC6VLX240T FPGA board [10]. It consists of 150MHz Xilinx MicroBlaze softcore, 256MB DRAM and FPGA logic with around 250,000 cells. This board is used for implementing the SHA-1 hardware logic and for measuring its overhead. Then, we project the SHA-1 hardware logic overhead on the OpenSSD board similar to that measured on the FPGA board. Hence, all the results reported in this paper are measured on the OpenSSD board while emulating the SHA-1 hardware logic overhead in a time-accurate manner. Currently, we are developing a new in-house SSD platform by integrating NAND Flash packages and SATA 3.0 host interface into the FPGA board.

In addition, we make use of another supplementary board, an ARM9 based EZ-X5 embedded board [3]. It consists of a 400MHz ARM9 CPU, 64MB DRAM, 64 MB NAND Flash memory, 0.5 MB NOR Flash memory, and embedded devices such as LCD, UART and JTAG. This board is used for evaluating the practicality of the sampling-based filtering on ARM 9 and for analyzing tradeoffs of deduplication in terms of performance, reliability, and costs on a various spectrum of CPUs.

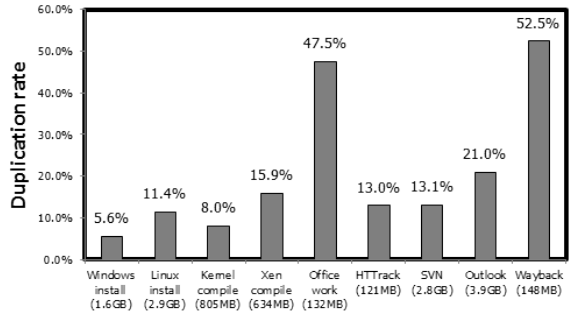


Fig. 10. Duplication rate of SSD workloads

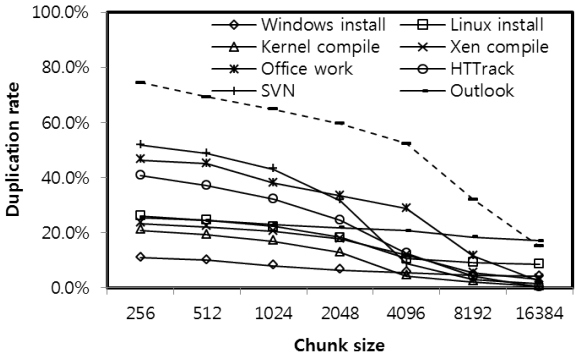
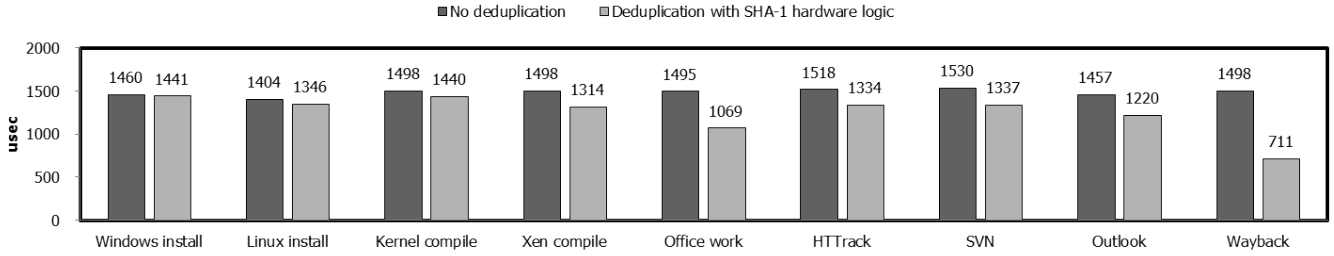


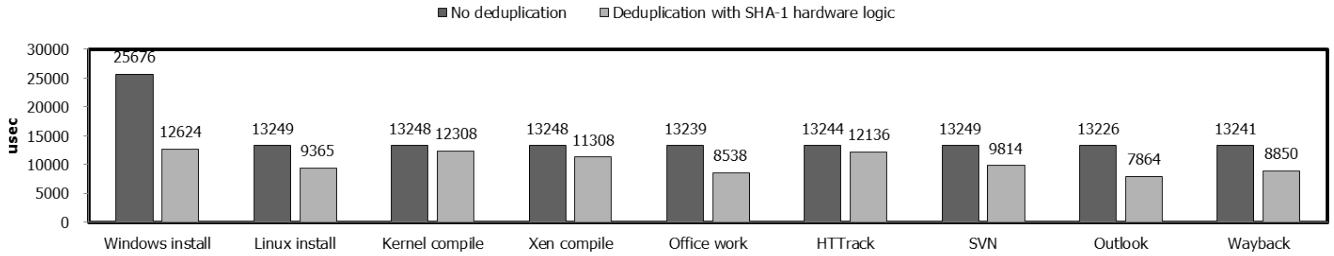
Fig. 11. Effects of Chunk size on Duplication rate

The following nine workloads are used for the experiments.

- Windows install: We install the Microsoft Windows XP Professional Edition. The total size of write requests triggered by this workload is around 1.6GB.
- Linux install: This workload installs Ubuntu 10.10, an operating system based on Debian GNU/Linux distribution, generating roughly 2.9GB writes.
- Kernel compile: We build a new kernel image by compiling the Linux kernel version 2.6.32. The total write size is 805MB.
- Xen compile: The Xen hypervisor is built using the Xen version 4.1.1, issuing 634MB writes.
- Office: We run the Microsoft Excel application while modifying data randomly whose size is roughly 20MB. We also enable the auto save option with the default setting, triggering 132MB writes during the one hour execution.
- Outlook sync: In this workload, we synchronize Gmail accounts used by our research members, randomly selected, with the Microsoft Outlook application. The total write size is 3.9GB.
- HTTrack: It is a backup utility, allowing to download contents from a given WWW site to our local storage [5]. In this workload, we download the contents of our university web site by using HTTrack, generating 121MB writes.
- SVN: The Apache subversion (often abbreviated SVN) is a software version and revision control system [1]. Using



(a) When garbage collection is not invoked during the workloads execution



(b) When garbage collection is invoked during the workloads execution

Fig. 12. Write latency with/without deduplication

the VirtualBox sources, we make a version (contains all sources) and several revisions (contains only the updated sources), which triggers writes with the size of 2.8GB.

- Wayback machine: It is a digital time capsule for archiving versions of web pages across time [9]. We browse the archived pages that are composed of the first page of the Yahoo! web site during the period 1996-2008. The total write size is 148MB.

B. Duplication Rate

Figure 10 shows the duplication rate of the nine workloads, ranging from 4% to 51% with an average of 17%. Among the nine workloads, we can achieve the same duplication rate for each run from the windows install, Linux install, kernel compile and Xen compile workloads, since duplicate data are intrinsic in these workloads. On the contrary, the duplication rate of the office workload varies according to user behaviors. We also tested the case where, after modifying a couple of bytes, we save data with a different filename. Unlike our expectation, the duplication rate is insignificant in this case mainly due to the compression scheme used by the recent Microsoft Office programs. However, we have observed that the auto save function supported by various word processor and spreadsheet programs yields a large amount of duplicate data.

The duplication rate of the HTTrack and outlook workloads depends on the contents of a WWW site and mail server. By testing other sites and servers, we noticed that there exist sizeable duplicate data in general. The wayback machine shows the best duplication rate since it writes not only the modified data but also the unchanged data altogether for archiving. On the other hand, SVN saves modified data only in each revision, resulting in a relatively low duplication rate.

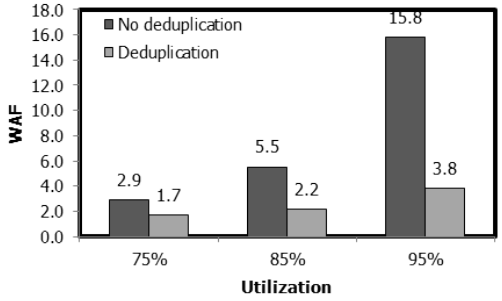
In our proposed deduplication framework, two parameters can affect the duplication rate. One is the number of fingerprints, as already discussed in Figure 8. The other is the chunk size, as presented in Figure 11. In this experiment, we configure the chunk size as 4096. Note that, as the size decreases, we can obtain a higher duplication rate, especially for the office, HTTrack and SVN workloads. It implies that we can expect the enhancement of the deduplication efficiency by using the smaller logical page size, such as fragment, in FTLs.

C. Write Latency

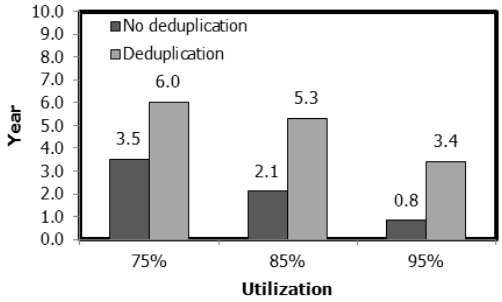
Figure 12(a) shows the improvement of average write latency per each request when deduplication is applied. Deduplication was processed using hardware implementation of SHA-1. Write operation diminishes as much as the duplication rate of Figure 10. Write latency decreases up to 48% with the average of 15% due to the elimination of duplicated data writing. The deduplication performance gain is significant because the overhead of SHA-1 hardware logic is only 80us, which is quite smaller than that of program time. Our proposed analytical model in Figure 3 predicts that the duplication rate should be more than 5% when the overhead is 80us in order to achieve performance gain. This prediction well corresponds with the experimental results.

Figure 12(b) shows the improvement of write latency when garbage collection is considered. In Figure 12(a), write operations were performed on a clear SSD which has all free blocks. In steady state, since there already exist a lot of data in SSDs, garbage collection should be included for reflecting the real world situation. We set 90% of SSD space as occupied by valid data while the rest space as free in this experiment. When we apply deduplication, we can decrease not only the data volume to write but also the number of copied pages during garbage

collection. Also, the reduced space due to deduplication can be exploited usefully as the over-provisioning area, which further decreases the invocation number of garbage collection. For these reasons, the improvement of the average write time by deduplication is even more effective when garbage collection is included during the execution of workloads.



(a) Write Amplification Factor



(b) Expected lifespan

Fig. 13. Expected lifespan with/without deduplication

D. Reliability

The WAF (Write Amplification Factor) is a ratio of the amount of data actually written in Flash memory to the amount of data requested by the host [24]. In SSDs, the WAF is generally larger than 1, due to the additional writes caused by the garbage collection, wear-leveling, and metadata writing. Deduplication can give a chance to reduce the WAF by reducing not only write traffic but also the copied pages during the garbage collection. Figure 13 (a) shows the effects of deduplication on WAF under the three different utilizations, 75, 85 and 95%. It shows that deduplication can reduce WAF significantly, especially under the high utilization.

The reduction of WAF diminishes the number of erase operations, which eventually affects the lifespan of SSDs. Several equations have been proposed to express the relation between the lifespan and WAF [32], [19], [36]. In this paper, using the equation of [32], we estimate the expected lifespan of SSDs with/without deduplication, as shown in Figure 13 (b). The figure shows that deduplication can expand the lifespan up to 4.1 times with an average of 2.4 times, compared with the no deduplication results.

Note that, even though NAND Flash based SSDs provide several advantages including high performance and low energy consumption, a lot of data centers and server vendors hesitate to adopt SSDs as storage systems due to the concerns of reliability and lifetime. Our study demonstrates quantitatively that deduplication is indeed a good solution to overcome the concerns.

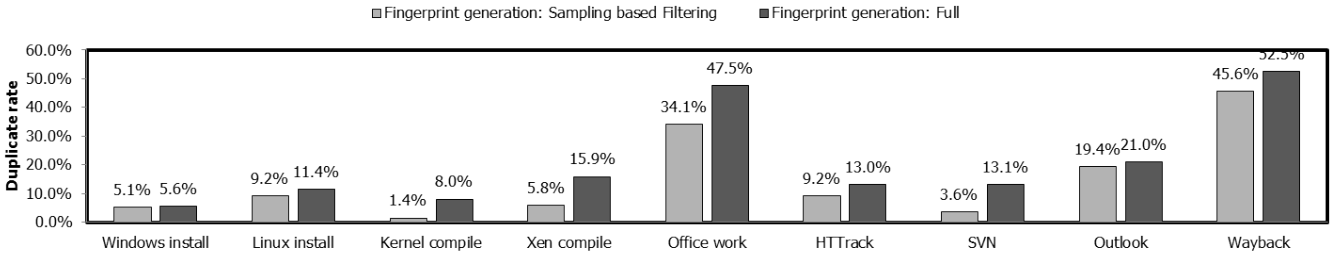
E. Effects of Sampling based Filtering

From Figure 12, we notice that deduplication with the SHA-1 hardware logic can improve the write latency. However, it requires additional hardware resources, which is a viable approach for high-performance oriented SSDs. On the contrary, some SSDs may have a different goal, that is cost-effectiveness to reduce the manufacturing cost. Those SSDs want to employ deduplication to achieve the enhancement of reliability, observed in Figure 13, without additional hardware resources while supporting performance comparable to the non-deduplication scheme. The sampling based filtering technique is proposed for those SSDs.

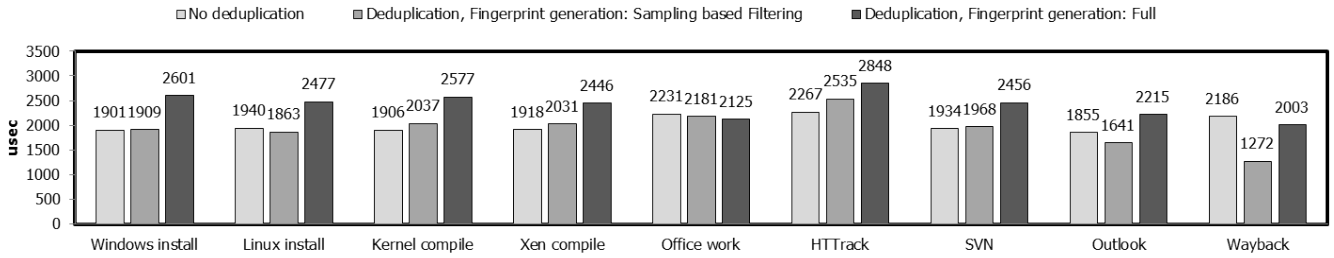
Figure 14 (a) shows the duplication rate under the two conditions: the one is generating fingerprints for all write requests and the other is generating selectively using the sampling-based filtering technique. The former provides a better duplication rate than the latter since the former tries to detect duplication for all writes. However, the results show that the latter still detects roughly 64% of duplicate data, compared with the duplication rate of the former.

The merit of the sampling-based filtering is that it can reduce the fingerprint generation overhead by not applying deduplication into write requests that have low duplicate possibility. This is more evident in Figure 14 (b) that describes the write latency under three testing environments, no deduplication, deduplication with the sampling-based filtering and deduplication with the full fingerprint generation. The results show that the sampling-based filtering performs much better than the original full fingerprint generation technique. It shows comparable performance to the non-deduplication scheme even though it creates the SHA-1 hash value in software without hardware resources. Note that, in terms of reliability, it equivalently supports the enhancement of lifespan of Figure 13.

Also note that the results presented in Figure 14 are measured based on ARM 9 CPU. We also conducted the same experiments on ARM 7 CPU. However, on ARM 7, since the overhead of SHA-1 software implementation is too heavy to obtain the performance gain, as already discussed in Figure 4. We find out that, with ARM 7 CPU, deduplication can only enhance the reliability of SSDs. To obtain the performance improvement together, the SHA-1 hardware logic is indispensible. On the other hand, with ARM 9 or higher capability CPUs, deduplication based on SHA-1 software implementation can give both performance and reliability enhancements. The SHA-1 hardware logic can further improve the performance.



(a) Duplication rate with Sampling based Filtering



(b) Write latency with Sampling based Filtering

Fig. 14. Performance evaluation of Sampling based Filtering

VIII. Related Work

Chen *et al*. proposed CAFTL [16] and Gupta *et al*. suggested CA-SSD [23], and those are closely related to our work. CAFTL makes use of the two-level indirect mapping and several acceleration techniques while CA-SSD employs content-addressable mechanisms based on the value locality. Indeed, their work is excellent, inspiring a lot on our work. However, our work differs from their approaches in the following four aspects. First, our work is based on real implementations, using various CPUs, and raising some empirical design and implementation issues. Second, we propose an analytical model that relates the performance gain with the duplication rate and deduplication overhead. Third, we examine the characteristics of SSD workloads with the view of recency, IRG, and frequency, and evaluate their effects on deduplication. Finally, we suggest several acceleration techniques and discuss tradeoffs on various hardware/software combinations. There are other prominent researches for improving the deduplication efficiency and performance. Quinlan and Dorward built a network storage system, called Venti, which identifies duplicate data using SHA-1 and coalesces them to reduce the consumption of storage [34]. Koller and Rangaswami suggested content-based caching, dynamic replica retrieval, and selective duplication that utilize content similarity to improve I/O performance [27]. Zhu *et al*. developed the data domain deduplication file system with the techniques of the summary vector, stream-informed segment layout and locality preserved caching [37].

Lillibridge *et al*. proposed the sparse indexing that avoids the need for a full chunk indexing by using sampling and locality [30]. Guo and Efstathopoulos developed the progressive sampled indexing and grouped markand-sweep for high-performance and scalable deduplication [21]. Debnath *et al*. designed Chunkstash, which manages chunk metadata on Flash memory to speed up the deduplication performance [18].

IX. Conclusions

In this paper, we have designed and implemented a novel deduplication framework on SSDs. We have proposed an analytical model and examined the characteristics of SSD workloads in various viewpoints. We have investigated several acceleration techniques including the SHA-1 hardware logic, sampling-based filtering and recency-based fingerprint management, and have explored their tradeoffs in terms of performance, reliability, and costs. Our observations have shown that deduplication is an effective solution to improving the write latency and lifespan of SSDs.

We are considering three research directions as future work. One direction is exploring a hardware/software co-design for efficient mapping managements such as a parallel memory-searching co-processor. The second direction is integrating compression with deduplication, which can further reduce the utilization of SSDs. The last one is evaluating the effects of deduplication on multi-channels/ways of SSDs.

X. Acknowledgment

This work was supported in part by the IT R&D program of MKE/KEIT No. KI10035202, Development of Core Technologies for Next Generation Hyper MLC NAND Based SSD and by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2009-0085883).

References

[1] "Apache subversion," http://subversion.apache.org.
[2] "Battery or supercap," http://en.wikipedia.org/wiki/Solid-state-drive.
[3] "Ez-x5," http://forum.falinux.com/zbxe/?mid=EZX5.

[4] *http://superuser.com/questions/253961/why-does-my-windows-7-pc-ssd-drive-keep-freezing*.

[5] "Httrack," http://www.httrack.com.

[6] *K9LCG08U1M NAND Flash memory*, www.samsung.com/global/business/semiconductor.

[7] *Sandforce SSDs break TPC-C records*, http://semiaccurate.com/2010/05/03/sandforce-ssds-break-tpc-c-records.

[8] "Verilog 2001," http://www.asic-world.com/verilog/verilog2k.html.

[9] "Wayback machine," http://www.archive.org/web/web.php.

[10] "Xlinix vertex-6 family overview," http://www.xilinx.com.

[11] D. G. Andersen and S. Swanson, "Rethinking flash in the data center," *IEEE Micro*, vol. 30, no. 4, pp. 52–54, Jul. 2010.

[12] S. Baek, J. Choi, S. Ahn, D. Lee, and S. Noh, "Design and implementation of a uniformity-improving page allocation scheme for flash-based storage systems," *Design Automation for Embedded Systems*, vol. 13, no. 1, pp. 5–25, 2009.

[13] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[14] S. Boboila and P. Desnoyers, "Write endurance in flash drives: measurements and analysis," in *Proceedings of the 8th USENIX conference on File and storage technologies*, 2010.

[15] J. Burrows, "Secure hash standard," DTIC Document, Tech. Rep., 1995.

[16] 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 stroage technologies*, 2011.

[17] E. Coffman and P. Denning, "Operating systems theory," 1973.

[18] B. Debnath, S. Sengupta, and J. Li, "Chunkstash: speeding up inline storage deduplication using flash memory," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010.

[19] W. Digital, "Nand evolution and its effects on solid state drive (ssd) useable life," Western Digital, Tech. Rep., 2009.

[20] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: anomalies, observations, and applications," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 24–33.

[21] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, 2011.

[22] A. Gupta, Y. Kim, and B. Urgaonkar, "Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, 2009, pp. 229–240.

[23] 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 stroage technologies*, 2011.

[24] A. Jagmohan, M. Franceschini, and L. Lastras, "Write amplification reduction in nand flash through multi-write coding," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–6.

[25] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," in *Proceedings of the USENIX 1995 Technical Conference Proceedings*, 1995.

[26] H. Kim and S. Ahn, "Bplru: a buffer management scheme for improving random writes in flash storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, 2008, pp. 16:1–16:14.

[27] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *Trans. Storage*, vol. 6, no. 3, pp. 13:1–13:26, Sep. 2010.

[28] S. Lee and J. Kim, *Understanding SSDs with the OpenSSD Platform*, Flashmemory Summit, http://www.openssd-project.org/, 2011.

[29] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, Jul. 2007.

[30] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," in *Proccedings of the 7th conference on File and storage technologies*, 2009, pp. 111–123.

[31] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 174–187.

[32] A. Olson and D. Langlois, "Solid state drives data reliability and lifetime," Tech. Rep., 2008.

[33] V. Phalke and B. Gopinath, "An inter-reference gap model for temporal locality in program behavior," in *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 1995, pp. 291–300.

[34] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *Proceedings of the 1st USENIX conference on File and storage technologies*, 2002.

[35] S. Rhea, R. Cox, and A. Pesterev, "Fast, inexpensive content-addressed storage in foundation," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, 2008, pp. 143–156.

[36] J. Standard, "Solid-state drive requirements and endurance test method (jesd218)," JEDEC, Tech. Rep., 2010.

[37] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, 2008, pp. 18:1–18:14.