LX-SSD: Enhancing the Lifespan of NAND Flash-based Memory via Recycling Invalid Pages

Ke Zhou[*], Shaofu Hu[*†], Ping Huang[*‡], Yuhong Zhao[§]

[*]Huazhong University of Science and Technology, China
[†]College of Computer and Information Technology, China Three Gorges University, China
[‡]National Engineering Laboratory for Next Generation Internet Access System, China
[§]Institute of Information Engineering, Chinese Academy of Sciences, China

Email: k.zhou@hust.edu.cn, hushaofu@hust.edu.cn, templestorager@temple.edu, zhaoyuhong1945@163.com
Corresponding author: Ping Huang

***Abstract*—NAND flash-based SSDs utilize out-of-place update to prevent erase operations from blocking regular requests, in the meanwhile, out-of-place update creates a huge amount of invalid pages. Traditional FTLs perceive these invalid pages as useless data and discard them during garbage collection. But given value locality, certain values are likely to appear afterward, resulting in abundant invalid pages being still useful. In this paper, we propose a content-aware FTL to explore such locality and recycle invalid pages to achieve Lifetime eXtension SSD (LX-SSD), which is hereafter named as Recyclable SSD. Such recycling invalid pages enables an SSD to recycle invalid pages at the page granularity and without involving block erase operation. Since block erasures serve as a criterion for quantifying the lifespan of SSDs, the reduced block erasures could result in lifespan extension. In order to retain high performance, we design a set of acceleration techniques to reduce the overhead of extra computational costs. We also modify the GC algorithm to delay certain useful invalid pages from being erased. We evaluate LX-SSD on real-world workloads, with content information, and the results show that our Recyclable SSD can achieve 10.5%-18.4% write reduction and 40.6%-58.9% reduction in block erasures, which results in 10.5%-18.4% overall reduction in I/O response time, and significant lifespan extension.**

I. INTRODUCTION

Since NAND Flash-based SSDs (solid state drives) have no moving parts, they have unique merits outpacing traditional HDDs (hard disk drives) in high-speed, energy-efficient, access latency and more resistance to physical shock [13]. The multi-/triple-level cells design together with the high-density scaling technology push manufacturers to deploy Flash memory in the data center, PC, and consumer electronic products. But integrating SSDs into commercial systems is still painfully slow. When the density increases, the endurable PE (program-erase) cycles decline substantially from ten thousand to one thousand [29, 36]. For example, the maximum number of PE cycles of the SLC (single-level cell) flash memory fabricated in a 70 nm process is about 10K PE cycles while the 2-bit MLC flash memory fabricated in the 2x nm process decreases to 3K PE cycles. For 3-bit TLC flash memory, this number is only around a thousand cycles [8]. This scaling technology exacerbates the long-standing concerns on SSDs lifespan which is strongly related to the cumulative number of PE cycle. Recent researches mainly focus on reducing host writes through inline deduplication [21, 35] or data compression [10, 27, 48] so as to conserve the PE cycles. In this work, we approach this problem from a different perspective: extending the PE cycle.

Due to physical constraints, NAND flash memory can only be recycled at the granularity of block. The PE cycle of a block is composed of a series of events: (1) programming: free pages in the block are filled with new data; (2) coping: valid pages in the victim block need to be copied into a new active block; (3) erasing: the entire block then can be erased after all valid pages have been copied out. We call this recycle as block recycle, making the entire block reusable through erasure operation, but at the cost of migrating valid pages. During block recycle, all invalid pages in this block are considered to be useless and unable to access. But we argue that those pages actually represent history data, and some of them still contain useful information and can be used rather than naively waiting to be erased in a later block recycle. In order to exploit the useful invalid pages, we propose a page recycle method, recycling at the granularity of page, to act as a supplement to block recycle. Before a block is selected as victim by block recycle, the page recycle can make invalid pages in this block reusable again without doing erase operation. In page recycle, an invalid page is defined as a recyclable page if it shares the identical content with an incoming write request. Normally, a block with *N* physical pages can only serve *N* write requests with a size of 4KB. With the permission of page recycle, the block can serve more write requests in its current PE cycle (invalid pages being recycled), equivalent to extend the PE cycle from N-programs-per-erase to N^+-programs-per-erase.

We propose Recyclable SSD to exploit the recyclable invalid pages. Our design embraces enhancements primarily in the FTL (flash translation layer) with minimal additional hardware changes. When the Recyclable SSD receives a write request from the host system, instead of immediately issuing it to the flash, the Recyclable FTL will firstly figure out whether there is an invalid page that has the same content as the arriving write. If such a match has been found, FTL would not issue this write to the flash: recycle the matched invalid page through simply changing its status, from invalid to valid, then map this write to the recycled page through updating the mapping table. The merit of the Recyclable SSD lies in that it can recycle a page without

erasing and accomplish a write operation without programming, killing two birds with one stone. The Recyclable SSD does not need to change the standard host/device interface for passing any extra information from the upper-level components to the device. All of the design is isolated at the device level and transparent to the user. This guarantees the Recyclable SSD to be an easy drop-in solution, which is highly desirable in practice.

Our contributions in this paper are summarized as follows: (1) We have designed a content-aware FTL to recycle invalid pages at page granularity and made PE cycles endure extra writing; (2) We have identified and characterized salient aspects of value locality in invalid pages, such as value popularity and value version; (3) We have modified the existing garbage collection algorithm to prevent blocks with useful invalid pages from being erased; (4) We implement the Recyclable SSD in the DiskSim [4] simulator and comprehensively evaluate its lifespan extension and performance impact.

The rest of the paper is organized as follows: Section II gives the background of flash memory. Section III analyzes the invalid pages and introduces the characterized value localities. In Section IV, we present the design of the Recyclable SSD in detail and the acceleration methods to minimize introduced overhead. We use real-world workloads to evaluate the performance and effectiveness in Section V and discuss the related work in Section VI. Finally, we conclude our work in Section VII.

II. BACKGROUND

A. NAND Flash Memory

A NAND flash memory package consists of two or several dies, having separate chip enable and ready/busy signals, and supporting interleaving operations. Each die contains multiple planes, and a plane is further divided into thousands of blocks. A flash block is composed of 64-256 pages [26], and each page has a spare area along with the data area. Data read/write are performed at page granularity, in an asymmetric manner. Take a Hynix NAND flash specification for example [7], a typical read operation takes 80us to read a page out of the flash cell, a write operation takes 1.5ms and an erase occurs at the granularity of a block, more expensive than read or write, taking roughly 5ms. Flash memory has critical technical constraints, named Out-of-place overwrite, i.e., a previously used block must be erased before any new pages within it can be reprogrammed. An erase block will wear out after a certain number of erase/program cycles (typically 1K-10K). As a critical component in the SSD design, the FTL is implemented in the SSD controller to emulate a hard disk drive by exposing an array of LPAs (logical page addresses) to the host. In order to address the aforesaid constraints, the FTL designers have developed a number of sophisticated techniques: a mapping table [1] is maintained to track the dynamic mapping between logical page addresses and physical page addresses, and each new write to a logical page will invalidate the previously occupied physical page, and the new content data is redirected to a free page in a clean active block; garbage collection [43] is performed periodically to recycle invalid pages, before a victim block (likely contains a lot of invalid pages) is erased, the remaining valid pages need to be consolidated into another clean block; wear-leveling mechanism is used to track and shuffle hot/cold data writes to be evenly distributed in flash memory, while hot data writes may cause some blocks to wear out earlier than others. In order to facilitate garbage collection and wear-leveling, SSD manufacturers typically include a certain amount of overprovisioned space in addition to the host-usable SSD capacity.

B. Out-of-place Overwrite

SSDs are significantly different from traditional HDDs in aspects of internal structure and storage media. HDD records data by magnetizing a thin film of ferromagnetic material and changes the direction of magnetization to represent binary data bits. This magnetic medium performs write and read operation at the granularity of a sector. A sector can be rewritten by changing the direction of magnetization in the original place, which is called in-place overwrite.

Flash based SSD records data in an array of memory cells made from FG (floating gate transistors), where the different levels of voltage on the FGs represent binary data bits. Using incremental step pulse programming [5], the voltage of cell can be elevated to the desired level, completing a write. But unlike the magnetic medium which can change magnetic direction from either side, the flash cell could not decrease the voltage without block erase. While a series of cells compose a page, read/write are performed at the page granularity and erase occurs at the granularity of a block. A page could not be rewritten in the original place due to the one-way voltage adjusting. In order to avoid the time-consuming erase being triggered, flash memory redirects an overwrite to a new free page in the active block, forming out-of-place overwrite. At the same time, the originally mapped pages are invalidated due to the writes to new free pages. Since the original pages are not erased immediately, massive invalid pages are produced through this out-of-place overwrite and remain in the SSD until the garbage collection recycles them.

III. MOTIVATION AND CHALLENGES

A. Invalid pages are common

The nature of out-of-place overwrite brings about huge amount of invalid pages. A NAND flash page alternates between three states: *free*, *valid*, and *invalid*. A free page becomes valid after it is programmed/written. It then transitions to an invalid page after being overwritten and finally becomes a free page through garbage collection. Normally, the free pages only occupy limited space, typically 5%-15%, which is used to facilitate the garbage collection, and the remaining space is fully filled with valid pages (current data) and invalid pages (out-dated data).

To verify the existence of invalid pages, we have studied 10 disk storage systems installed on 5 machines in the Key Laboratory of Data Storage Systems at Huazhong University of Science & Technology. These disks support 4 office systems, 4 experimental systems, and 2 web servers. Since these disks are HDDs where no invalid page exists, we recorded block I/O request traces on these disks, and load them into a popular SSD extension of the DiskSim [2, 4] to investigate the formation of

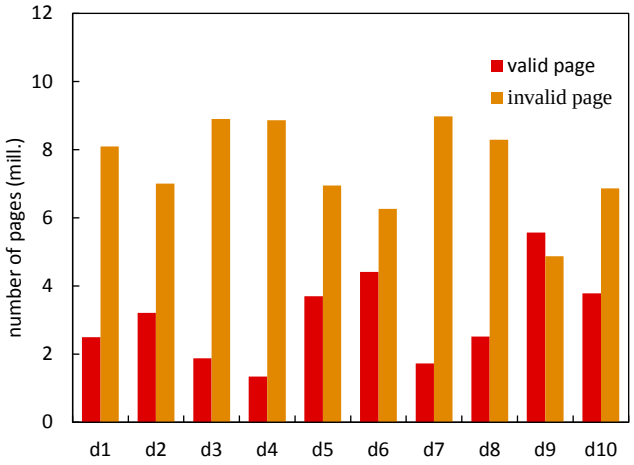

Fig. 1. Comparison of valid and invalid pages formed in 10 different disks. We recorded block I/O request traces on these disks with a duration of 30 days, and observe the formation of valid and invalid pages by running the traces into DiskSim SSD extension simulator.

both valid and invalid pages. The duration of these traces lasts for 30 days, the forepart 10 days are used to age the SSD where all pages are firstly initialized as free. In the aging process, a free page becomes a valid page when it encounters either a read or a write operation. Then we filter out read requests in the other 20 days of the trace, and use the other write requests to observe the formation of valid and invalid pages: a write request to a valid page will overwrite it and results in an invalid page, while a write to a free page generates a valid page. As we can see in Figure 1, the number of invalid pages is 1-4 times the number of valid pages across these 10 disks. Particularly, the offices have exhibited an extremely high invalid page generating rate. Traditionally, these invalid pages are considered to be garbage and useless which will be cleaned later during garbage collection. But we argue that these invalid pages indeed represent historical data, which are simply out of date, but not necessarily useless. In fact, they may still contain useful information.

B. *Invalid pages could be useful*

Data is frequently updated to retain the freshness, such as the weekly or daily updated website, the frequently updated dataset, or the periodically auto-save operation on the edited document. In many scenarios, those frequent updates only modify a small fraction while most of the previous data remains unchanged, which means there are still lots of useful data buried in the previous data that have become invalid pages in an SSD. In order to make the data source more widely representative and more credible, three workloads, including *home*, *mail*, and *web*, from FIU [22], are introduced to observe the appearance of useful data in invalid pages. The *home* workload is a file server supporting developing, testing, experiments and technical writing in FIU's CIS department. The *mail* is collected from the e-mail server of the CS department, and the *web* is FIU's web server workload consisting of webmail proxy and online course management. We select two weeks traces from the 20-29 days workloads, the first week is used to age the SSD, and the second week is used to observe the amount of the useful data. We regard an invalid page as useful data when it shares the identical content

TABLE I. WORKLOAD STATISTICS

Workload	Request (mill.)	Size (GB)	Writes (%)	Unique addr(mill.)
homes	9.3	3.7	93.8	2.4
mail	16.3	4.8	75.3	1.2
web	5.9	2.1	82.5	0.8

with an incoming write request. So we use the recyclable pages (defined in Section I) to represent the useful data. By loading the three traces into DiskSim SSD extension simulator, we find all of them have exhibited adequate recyclable pages, meaning that invalid pages are useful.

As shown in Table I, these workloads are highly write-dominant, with writes ranging from 75.3% to 93.8%. In the duration of two weeks, there are 9.3, 16.3, and 5.9 million requests in *home*, *mail*, and *web* respectively, but only 2.4, 1.2, and 0.8 million addresses appeared, revealing that more than 72.5%-90.2% write requests are overwrites. As we discussed before, these frequently overwrites will generate massive invalid pages: 24.1 GB, 42.2 GB, and 15.5 GB are observed in *home*, *mail*, and *web* respectively during the examined durations. These invalid pages are previous data, but not useless. In fact, we have noticed 2.8GB, 7.4GB, and 2.5GB recyclable pages, meaning that 11.5%-17.6% of invalid pages are recyclable.

C. *Locality Characterization*

Since full scanning in these invalid pages is not feasible, a pressing need of finding localities is arising. According to our observation, all of the three workloads contain a mixture of hot and cold data: a relatively much larger amount of cold data is accessed by a minority of I/O requests, while a small amount of hot data is accessed at a much higher frequency. We characterize two types of value locality to effectively assist us in dividing the invalid pages into highly recyclable and less recyclable.

The first locality is value popularity, a characterization on frequency representing how many times an invalid page has been accessed (number of times being read) before it is invalidated. We observe remarkable value popularity locality in the three workloads from FIU. This locality means that the invalid pages with higher value popularity, more preferred by the host, are more likely to be recyclable invalid pages than those with lower value popularity. It is worth mentioning that in order to minimize the costs of recyclable probability calculation (introduced later in section IV-C), we use 4 bits to record the value popularity and do not distinguish pages being accessed more than 15 times. As we can see in Figure 2, most of the invalid pages have value popularity ranging from 0 to 3, which means they are used to be cold data or less preferred by the host (never been accessed or only accessed by a few times). The over accessed hot data, which have high value popularity, are in the tail and only account for a very small fraction. The concept of hot and cold data also exists in invalid pages. And we can easily strip this large amount of cold invalid pages out by marking each invalid page with value popularity.

In order to find out how sensitive the value popularity will affect the recycle probability, we calculate the recyclable

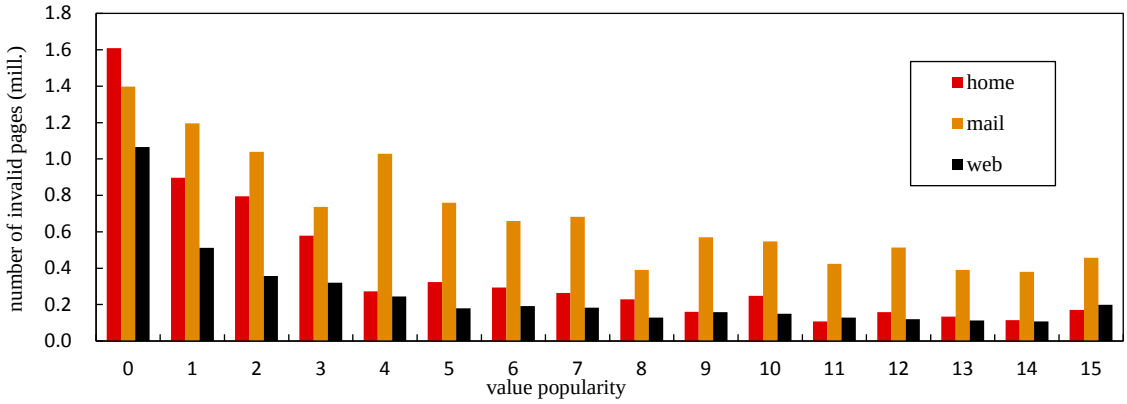

Fig. 2. Statistics of invalid pages under different value popularity. Value popularity presents how many times it has been accessed before it is garbage collected. We divid the invalid pages into 16 groups (0-15) according to their value popularity, and then count the number of invalid pages in each group.

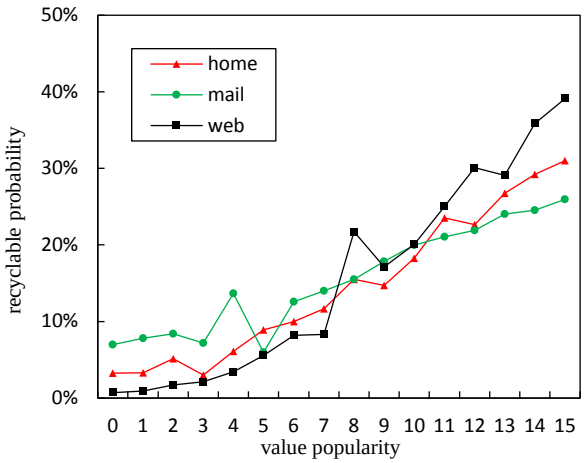

Fig. 3. Recyclable probability is closely related to the value popularity. Assuming that there are *m* invalid pages with value popularity of *x*, *n* of the *m* invalid pages are recyclable pages, we define the recyclable probability of invalid page under value popularity *x* is $\frac{n}{m}$.

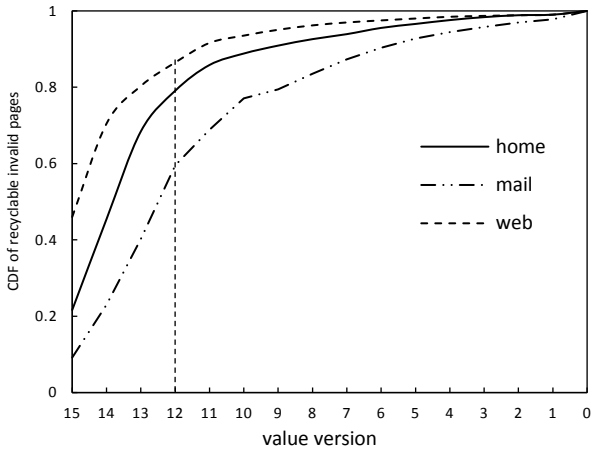

Fig. 4. The CDF of recyclable invalid pages. We count the number of recyclable invalid pages under different value version. Most of the recyclable pages are found with value versions between 15 and 12.

probability under each value popularity based on the statistics. Assuming that there are *m* invalid pages with value popularity of *x*, *n* of the *m* invalid pages are recyclable pages, we define the recyclable probability of invalid page under value popularity *x* is $\frac{n}{m}$. As shown in Figure 3, the recyclable probability under value popularity 0 is particularly low 0.7%-6.9% in *home*, *mail*, and *web*, but it seems to be exponentially increased with larger value popularity, up to 26%-39.1%. These findings show that hot invalid pages, approximately 20.8%-32.9%, with the popularity ranging from 8 to 15, account for about 52.8%-80.2% recyclable invalid pages, implying that recyclable probability is closely related to the value popularity.

The second locality is value version, a characterization on data representing how long an invalid page has been generated. The value version can be obtained from the sequential invalid pages: a series of invalid pages generated by the frequent updates on a certain LPA. We use a linked list to record the generated sequential invalid pages all of which had ever been mapped to the same LPA, and name the list as SIPL (Sequential Invalid Pages List, discussed later in Section IV). Since the length of SIPL depends on the update frequency of LPA, the data that is frequently overwritten will have a longer SIPL. As we have introduced, all of the three workloads exhibit huge overwrites, so we find that the average length of SIPL is 7.3, 12.1 and 6.7 in *home*, *mail*, and *web* respectively. Value version then depends on the position in SIPL, which reflects how recent the invalid page is: the head of SIPL represents the invalid page with the latest version while the tail represents the most out-of-date version. We use 4 bits to denote the value version, and it can represent a range from 0 to 15. The newest invalid page located at the head of the list has the largest version 15 and the version of the rest of the list are consecutively decremented by 1. It is worth mentioning here that 4 bits are sufficient to indicate the version of invalid pages, because when the version drops to less than 0 the recyclable probability is very low, so we do not distinguish the version when it reaches 0. In Figure 4, we find that recyclable invalid pages are mainly distributed at the head

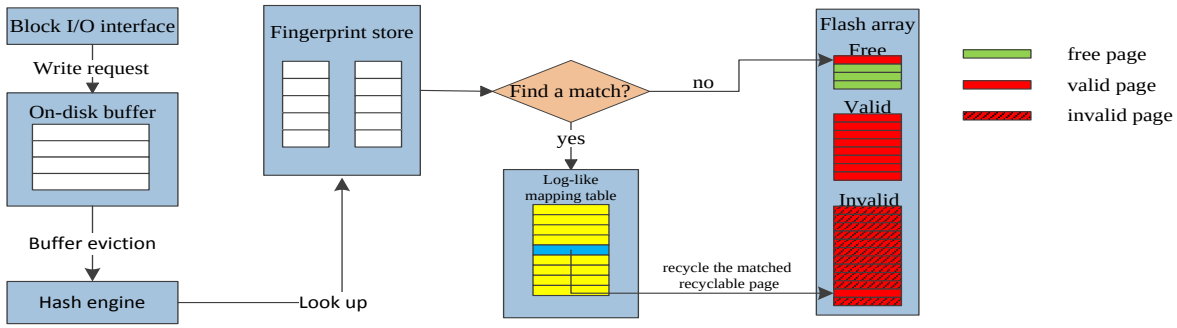

Fig. 5. The Recyclable SSD Overview. A match in fingerprint store will recycle an invalid page while a mismatch will issue the write to a free page.

of SIPL with a larger version number. As a fact, 79.1%, 59.6% and 86.5% of recyclable invalid pages are found under the value version from 15 to 12, in *home*, *mail*, and *web*, respectively. This finding indicates that value version also affects the recycle probability.

Overall, we can obtain the following insights regarding the characterized two localities. First, all these workloads exhibit significant skewness in value locality, i.e., a small fraction of invalid pages contains most recyclable pages. Recycling invalid pages through scanning thus becomes feasible because we can catch most recyclable pages without completely scanning the SSD. Second, the impact on performance can be maintained at a minimum due to the existence of value locality. The performance will be improved since no programming is needed when we recycle invalid pages. The characterized localities make our Recyclable SSD meaningful and feasible to implement by only look-up in a small fraction of invalid pages.

D. Challenges

A number of technique challenges arise when designing the Recyclable SSD:

- The internal device of SSD only has limited RAM capacity. Designing a content-aware FTL necessitates extra metadata which puts additional pressures on the scarce RAM.
- Our design works at the block-layer where FTL only sees a sequence of logical blocks without any attached semantic hints from the host system, which makes it more difficult to identify recyclable invalid pages, and requires more sophisticated design.
- We need to minimize the overhead of lookup operations when scanning invalid pages in order to make the Recyclable SSD still retain high data access performance even when there is little locality among invalid pages.

IV. Design of Recyclable SSD

We develop the FTL mapping unit and the GC of SSD and aim to realize the following several goals: (1) Recycle at page granularity: through studying the plentiful invalid pages and classifying them, we have discovered considerable recyclable invalid pages. Our Recyclable SSD can effectively identify those recyclable pages and recycle them at page granularity before the block recycle, improving the SSD lifespan. (2) Reducing the number of block erasures: the page recycle can recycle the invalid pages without performing erasure operation, which reduces erasure operations significantly, and improves the SSD lifespan substantially. (3) Retaining high performance: scanning operations for identifying recyclable pages are truly time-consuming and will deteriorate the performance if not handled properly. The Recyclable SSD has minimized the impact and exhibit improved performance due to exploiting the phenomenon of value locality. In this section, we present the architecture of the Recyclable SSD firstly and then introduce each of the main components in detail.

A. Architecture Overview

Recyclable SSD identifies invalid pages using their fingerprints, which is similar to CAS (content addressable storage) [15, 32]. We employ a collision-free cryptographic hash function (MD5) to compute the hash value as the fingerprint to present each invalid flash page. All of the fingerprints are indexed in a fingerprint store. A write is marked as removable write when its fingerprint is matched in the fingerprint store. The Recyclable SSD can serve those removable writes without performing page programming by recycling the recyclable invalid page, which is simply realized through updating the mapping table and changing page status.

Figure 5 illustrates the process of handling a write request in the Recyclable SSD: First, the write request is temporarily cached in a simple FIFO or SLRU-managed cache. When cache eviction occurs, the evicted write request will be computed to generate a fingerprint by a hash engine for the next scanning. The hash engine [24] can be a dedicated processor or simply a part of the controller logic. Each generated fingerprint is looked up in a fingerprint store, which maintains the fingerprints of highly recyclable invalid pages. A matched finding means that one invalid page is identical in content to the write request. The matched invalid page is recycled by simply changing its status (from invalid to valid), mapping table directs this update write to the recyclable invalid page. Doing so Recyclable FTL is able to filter this unnecessary write since no actual write has to be

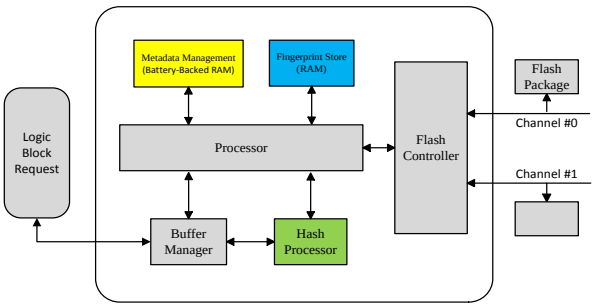

Fig. 6. Components of the Recyclable SSD. We add three extra components: (1) Hash Processor, (2) Battery-Backed Metadata Management and (3) Fingerprint Store.

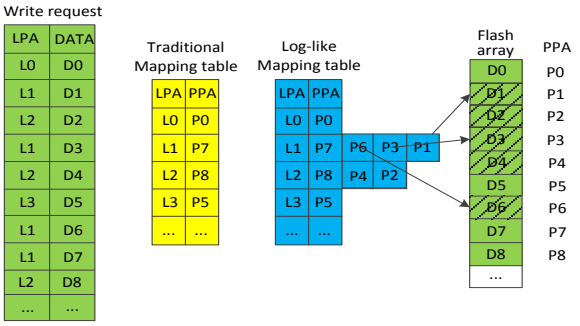

Fig. 7. Log-like mapping table

performed. A mismatch means that there is no invalid page that could be recycled, and therefore the write has to be issued to the flash array in the same way as a regular new write.

In order to make the Recyclable SSD aware of the invalid pages, we add extra components into the SSD architecture as illustrated in Figure 6. (1) Hash Processor: apparently using the byte-by-byte comparison to identify recyclable pages is undesirable, so we employ MD5 to index incoming writes and invalid pages. For each 4KB page, the 16B hash value is generated as its fingerprint and stored in the flash as the page's metadata. We choose fixed-sized hashing for incoming write requests due to the internal basic operation unit in flash being a 4KB page. To minimize the impact on write performance, we utilize a hash processor. The widespread deployment of onboard cryptographic processors in SSDs suggests that the deployment of the hash engine is feasible [18]. (2) Battery-Backed RAM: a traditional FTL will lose the mapping table in the event of a power failure, but the table can be reconstructed in the next time startup. This is mainly contributed by the OOB, a reserved area typically of 64-256B in size, which is dedicated for meta information such as: bad blocks, ECC [16, 30], erase counters, etc. When programming a physical page, the LPA that this page has been mapped to will also be written into the OOB area. The mapping table reconstruction can work properly in traditional SSD where each physical page is mapped only once during its PE cycle. In our Recyclable SSD where the invalid page is recyclable, the reconstruction cannot work anymore, because the OOB only stores the previous LPA while this recycled invalid page may have been mapped to a new LPA. So we employ a persistent battery-backed RAM to maintain the meta-data management. Recently, SSD manufacturers have started providing battery-backed DRAM as a standard feature to deal with a power failure, suggesting using BB-RAM deployment is feasible.

B. *Metadata management*

The existing mapping table holds the PPA of all valid pages and provides us an efficient pattern to recycle invalid pages. As we have introduced, an invalid page is recyclable when it shares the same content with the incoming write request. The recycle of the identified recyclable page is realized in switching the status of the page from invalid to valid. So we can simply achieve this recycle by adding the PPA of the recycled invalid page into the mapping table. But the data structure of existing mapping table cannot fully meet the requirements of the Recyclable SSD. Therefore, we make several enhancements to the metadata management.

1) Log-like mapping table: The mapping table is a basic internal metadata of an SSD, maintaining the mapping of LPA to PPA. Traditional mapping table only contains the mapping of valid pages while invalid pages are treated as useless data waiting for the erase operation within garbage collection. While we need to utilize the invalid pages in our Recyclable SSD, the existing MT (mapping table) is not sufficient anymore. We design an LMT (log-like mapping table), as depicted in Figure 7, which not only contains the mapping records of valid pages but also includes the invalid pages, to track the mapping record of those invalid pages.

Frequent updating of hot data generates series of invalid pages. The LMT uses a linked list to store all the physical page addresses that have ever been mapped to the same logical page address. Physical address is inserted into the head while handling an update request, indicating that the header of the list are newly formed, and removed from the list when recycled by GC. This log-like mapping table doesn't need extra RAM compared with traditional mapping table since its size only depends on the number of physical pages. The only difference is that log-like mapping table is always fully filled while traditional mapping table is not. The historical mapping records can guide us to achieve fast recycling, which we will introduce in Section IV-C.

2) Accessing table: In order to identify the value popularity characterized in Section III-C, we build an extra table AT (accessing table) to record the number of times each physical page has been accessed. The entry in the AT is a key-value pair: (*location*, *times*), where the *location* is a 32-bit physical address and *times* is a value remembering the number of times this physical page has been accessed. All entries are sorted by location in ascending order, and the *times* of each *location* is initially set to 0 when all of the physical pages are free.

The AT table will be updated in the following cases: (1) When the physical page is a valid page, each read operation issued to it will increase its *times* by 1. We define the valid page being accessed more than or equal to 15 times as over-accessed

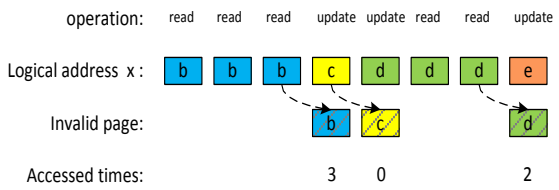

Fig. 8. The accessed times of invalid pages. Invalid page b, c, and d are accessed by 3, 0, and 2 times respectively, representing different value popularities and exhibiting skewed recyclable probability.

pages and the *times* would not increase after it has reached 15. (2) When the physical page becomes an invalid page, its value of *times* will be locked since no read will issue to it anymore. (3) When the GC cleans some invalid pages and turns them into free pages, their *times* will be set to 0 again. We use a battery-backed RAM to guarantee the safety of metadata stored in the LMT table and AT table. These two tables play important roles in identifying the recyclable probability of invalid pages: the value version of an invalid page is calculated based on the position in log-like mapping list which is stored in the LMT table, while the value popularity is obtained from the AT table.

C. Fingerprint Store management

FS (fingerprint store) maintains the information of the recyclable invalid pages. Each entry is a 24B key-value pair: (*fingerprint*, (*location*, *prob*)). The *Fingerprint* is a 16B hash value calculated by MD5, the *location* denotes a 4B physical address, and the *prob* represents the recycle probability, introduced later in this section, occupying 4B. For each 4KB incoming write request, we calculate a 16B hash value as its fingerprint and look it up in the FS to find out which invalid page is recyclable when there is a match.

Since the recyclable invalid pages are skewed distributed, only a small fraction of invalid pages is highly recyclable, which means maintaining fingerprint of all invalid pages into FS is too costly and unnecessary. So we design an algorithm to calculate the recyclable probability of each invalid page in view of the characterized localities: popularity and version. And the FS only stores those invalid pages with higher recycle probability.

1) Recyclable probability calculation: As we introduced in Section III-C, the storage system contains a mixture of hot and cold data. Those logical pages, contain hot data and receive more I/O requests, are updated frequently.

In Figure 8, a logical address *x* with value *b* receives several requests and finally is updated to value *e*. Three invalid pages (*b*, *c*, *d*) are formed and exhibit skewed recyclable probability: if a page was accessed (read) by the upper layer, the page is more popular for the recycle purpose. For example, the invalid page with value *b* which have been accessed for 3 times has larger recyclable probability than the invalid page *c* which has never been read.

So we firstly define the value popularity of an invalid page based on the number of times it has been accessed. The popularity of an invalid page p is denoted as $pop_{inv}(p)$ and is calculated as:

$$pop_{inv}(p) = (1+\alpha)^{ace_{inv}(p)} \quad (1)$$

In this equation, $ace_{inv}(p)$ represents the accessed times to the invalid page before it is invalidated. The value of $ace_{inv}(p)$ ranges from 0 to 15, where 15 presents over-accessed pages which have been accessed more than or equal to 15 times. The variable α is a growth factor that determines how quickly the popularity grows as ace_{inv} increases. Since the recyclable probability is nonlinearly increased with the incremental accessed times, we use exponential function to simulate the growth. Growth factor is typically set in the range of 0.1 to 0.15 – higher values give more weight to growth rate.

Then, we use $pop_{inv}(p)$ to initialize the recyclable probability when an invalid page is newly formed. The initial recyclable probability of an invalid page p is denoted as $ReProb_{ini}(p)$ and is initialized as:

$$ReProb_{ini}(p) = \frac{pop_{inv}(p)-1}{(1+\alpha)^{15}-1} \quad (2)$$

Since the maximum value of $pop_{inv}(p)$ is $(1+\alpha)^{15}$, we normalize $ReProb_{ini}(p)$ to a range from 0 to 1 using equation (2).

However, it is worth noting that $ReProb_{ini}(p)$ may decline after the initialization, mainly due to the drop of value version which also has notable influence on the recycle probability. The value version of an invalid page p is denoted as $ver_{inv}(p)$ and is calculated as:

$$ver_{inv}(p) = \begin{cases} 15 - pos_{list}(p) & pos_{list}(p) < 15 \\ 0 & pos_{list}(p) \geq 15 \end{cases} \quad (3)$$

The $pos_{list}(p)$ represents the position in the list of sequential invalid pages which have been mapped to the same logical page. We do not distinguish the version of pages in the tails of the list because their recyclable probabilities are very low. So we set $ver_{inv}(p)$ to 0 after $pos_{list}(p)$ reaches 15.

Apparently, $ver_{inv}(p)$ represents how close the invalid page p approaches the fresh page (i.e, the currently valid page). As we have illustrated in Figure 7, list (P6, P3, P1) includes a sequential invalid pages which have been mapped to the same logical page L1 (hot data, updated frequently). The recyclable probability of the invalid page P1 declined twice after P3 and P6 are inserted into the list successively.

We use $ver_{inv}(p)$ to mark the decay of recycle probability. The final recyclable probability of the invalid page p is denoted as $ReProb_{fin}(p)$ and is calculated as:

$$ReProb_{fin}(p) = ReProb_{fin}(p) * (1-\beta)^{(15-ver_{inv}(p))} \quad (4)$$

In this equation, the variable β is a decay factor that determines how quickly $ReProb_{fin}(p)$ declines as $ver_{inv}(p)$

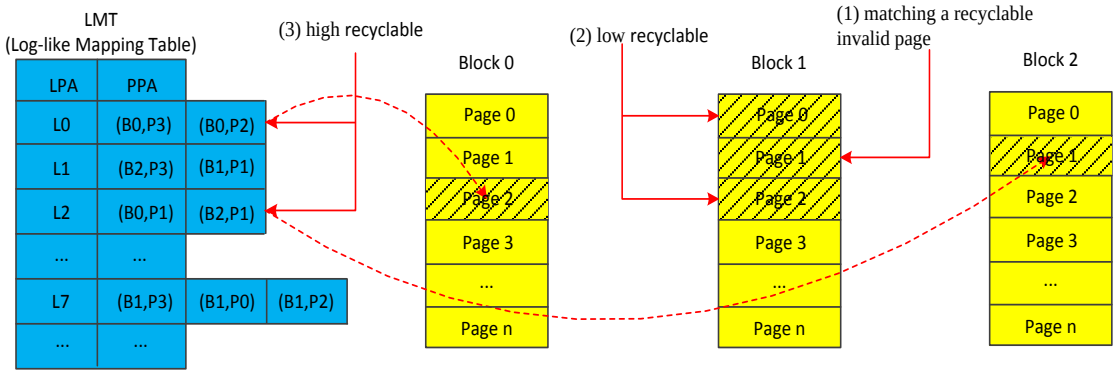

Fig. 9. Spatial locality on logical page address. The invalid pages (B1, P0) and (B1, P2) with close physical addresses are low recyclable pages while the dispersed invalid pages (B0, P2) and (B2, P1) show high recyclable probability.

falls. Decay factor is typically set in the range 0.15 to 0.2 – higher values give more weight to decay rate.

2) Fingerprint Store maintenance: Maintaining FS in the flash array will lead to extra writes and consume flash memory lifespan, so we use an in-memory RAM to store the FS and absorb lookup and update on the flash array. On SSD startup or rebooting, the FS is empty and need to be reconstructed, the Recyclable SSD will scan the invalid pages and use the algorithm, introduced in Recyclable probability calculation, to calculate recycle probability, and only load the metadata (key-value pairs) of invalid pages with higher recyclable probability into the FS.

The loaded entries in FS are arranged in descending order of recyclable probability to facilitate fast lookup. After initialization, FS is updated dynamically. New entries may be inserted while fresh invalid pages are formed, and stale entries may be squeezed out when its recyclable probability reaches the least, and the cleaned invalid pages need to be removed when GC is triggered. As introduced in the log-like mapping table, a new invalid page, inserted at the head of the linked list, will decrease the value version of the subsequent invalid pages. So the FS also has to keep freshness by reordering entries whose value version has been decreased.

We set the capacity of FS to 3MB which can maintain 128K invalid pages' metadata. Although we will lose the chance to identify some recyclable invalid pages, which have been proved quite small by experimental analysis in Section V-B, but it will save us lots of RAM capacity for fingerprint storage and accelerate the lookup operation.

3) Fast loading: In block level HDD deduplication, if we find a duplicated block b_x (with storage address x), then its nearby blocks b_{x-i} to b_{x+i} are very likely to be duplicate too. This is called as spatial locality [17, 37], which refers to the use of blocks or pages within relatively nearby addresses. Since the Recyclable SSD is similar to the deduplication in the sense of searching data items which share the identical content, we can utilize the spatial locality to quickly identify other highly recyclable invalid pages and load it into the fingerprint store, thereby, accelerating the look-up operation in it. But identifying such locality in an SSD is not as easy as in HDD. The SSD has two-level addresses: the mapping table will map contiguous logical addresses to non-contiguous physical addresses. When we match a recyclable invalid page, the adjacent invalid pages are not likely to be recyclable. This is mainly because that spatial locality is only related to the logical address. Fortunately, with the help of LMT table (introduced in Section IV-B), we are able to solve this problem.

As mentioned before, the LMT table maintains the mapping history of each logical address. While matching a recyclable invalid page (B1, P1), Page1 in Block1, we query LMT to find out the logical address L1 that (B1, P1) have been mapped to. As depicted in Figure 9, the surrounding invalid pages (B1, P0) and (B1, P2), which have relatively close physical addresses, have no spatial locality with (B1, P1) and are less recyclable pages, because they have been mapped to L7 which is too far from L1. On the contrary, the dispersed invalid pages (B0, P2) and (B2, P1) are highly recyclable pages, because as we can see in LMT, they have been mapped to L0 and L2 which are close to L1. So every time when we match a recyclable invalid page, e.g., invalid page (B1， P1), we use LMT to quickly locate those invalid pages which have spatial locality with the matched (B1，P1), such as (B0, P2) and (B2, P1), and set $pop_{inv}(p)$ of these pages to the highest value, representing they are likely to be recycled next, and fast load them into the head of FS store.

D. Garbage collection

SSD writes generally involve two types: external writes and internal writes. External writes come from the host system, while the internal write is induced by moving valid pages from victim block to a new block during GC (garbage collection). When an external write is issued randomly to the SSD, the pages marked as invalid are scattered throughout the entire SSD, creating many holes in every block. GC (garbage collection) is a fundamental process by which solid state drives can recycle

those holes, always ensuring enough free pages for the incoming writes, and thus retaining high performance. Traditional GC policies, like greedy algorithm [14], is designed to reclaim as much space as possible and thus reduce the write amplification [47] due to migrating valid pages. Others [6, 28, 44] have taken the erasure count, elapsed time or predicted IO into consideration. But most of them may work poorly with our Recyclable SSD due to being unaware of its characteristic. For example, when there is insufficient free space (i.e., below 5%) to flush data, the greedy algorithm selects a block with the most invalid pages as the victim, regardless of the recycle probability. In order to avoid blocks containing most recyclable invalid pages to be selected as the victim, we modify the greedy algorithm via designing a new victim block selection scheme which assigns each invalid page with different weights on the basis of recyclable probability. The **R**e**C**yclable-aware GC, named as RC-GC, calculates a value denoted as $BlkStats(b)$ to represent the dirty status of block *b*:

$$BlkStats(b) = \sum_{i=1}^{n} PageStats(i) * (1 - ReProb_{fin}(i)) \qquad (5)$$

In this equation, *n* is the number of pages within block *b*, $PageStats(i)$ is 1 or 0 when flash page *i* is invalid or valid, and $ReProb_{fin}(i)$ is the metric of recyclable probability introduced in Section IV-C. When free space is not sufficient, the block with the largest $BlkStats$ will be selected as the victim block. The enhancement on victim block selection enables RC-GC to more efficiently carry out garbage collection which would otherwise unnecessarily clean the blocks with recyclable invalid pages.

V. Experimental evaluation

We implement and evaluate the design of the Recyclable SSD based on a series of comprehensive trace-driven simulation experiments. In this section, we present the experimental results to investigate the following questions:

- How many writes can be eliminated?
- How sensitive are recycle process to the RAM capacity of the fingerprint store?
- How does the lifespan of SSD benefit from recycling invalid pages?
- How will average response time benefit from the page recycle?

A. Experimental setup

We have implemented our design in the Microsoft Research SSD extension [2] to the DiskSim [4] simulation platform. The SSD extension provides the simulation of the major components in FTL, such as the indirect mapping, garbage collection, and wear-leveling. We have made several enhancements to the mapping tables to track mapping history of the invalid pages and modified the garbage collection algorithm as discussed in preceding sections. The unmodified version of SSD extension refers to the standard SSD as a *baseline*.

TABLE II. Configurations of the SSD simulator

Description	Confirmation
Page size	4K
Pages per block	64
Blocks per plane	2048
planes per chip	8
number of chip	10
read latency	25us
write latency	200us
erase latency	1.5ms
Over-provisioning	15%
Cleaning threshold	5%

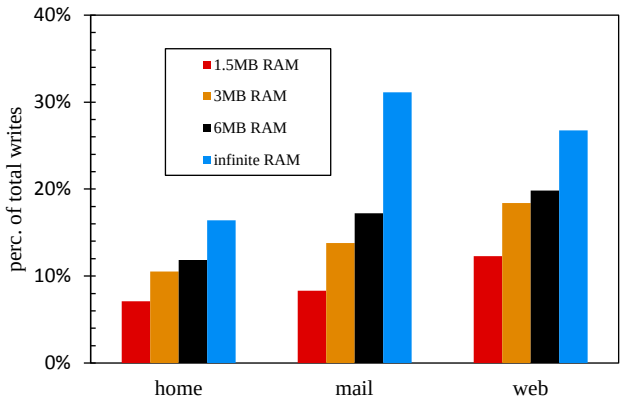

Fig. 10. Eliminated writes

We set simulated SSD with the default configurations from the SSD extension, specified in Table II. Due to alignment constraint of DiskSim, the page size is set to 4KB for all drives. To maintain the parallelism, each chip has 8 planes and each individual plane has its own allocation pool.

In order to verify the performance of the Recyclable SSD, we measure the reduction on write, the number of block erasure operations and the average response time. We use three sets of experiments with different fingerprint store RAM sizes of 1.5MB, 2MB, and 6MB, maintaining 64K, 128K, and 256K entries respectively, to see how the capacity of RAM will affect the results. All the experiments are fed with the workloads collected from FIU, introduced in Section III-B.

B. Write reduction

Denote the total number of received write as *n*, and the number of actual write being flushed into flash memory as *m*. Then the write reduction is calculated as $\frac{n-m}{n}$. Figure 10 shows the write reduction under varying RAM sizes from infinite to 1.5MB. Here "infinite" means that there is sufficient RAM capacity where all removable writes could be removed. The *mail* exhibits highest write reduction (with infinite RAM) 31.1%, while *home* is the lowest 16.4%. The reason is that *mail* has the most update writes, i.e., the average number of update

operations on each logical address is 7.8, resulting in the largest amount of invalid pages. When RAM size is 1.5MB,

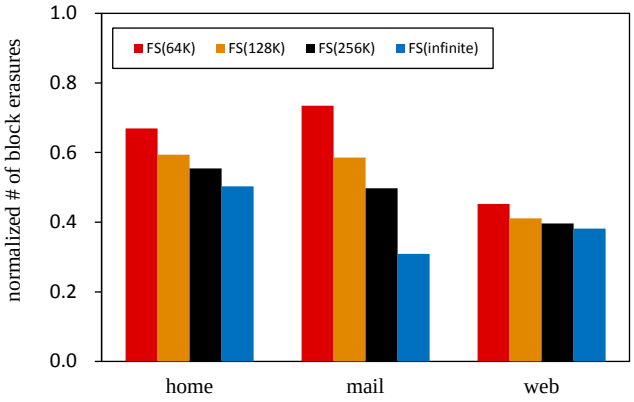

Fig. 11. Normalized number of block erasures

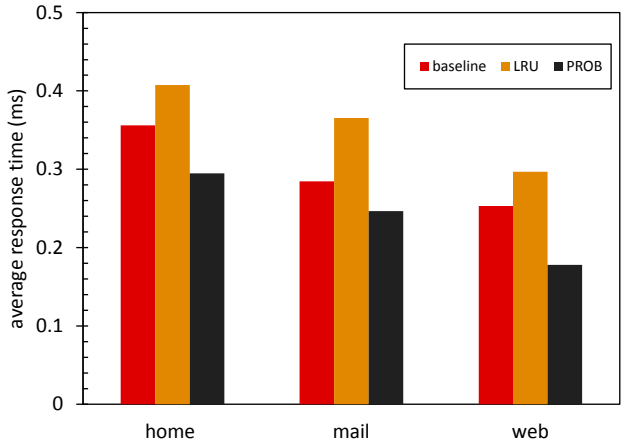

Fig. 12: Average response time

the Recyclable SSD can achieve 7.1%-12.3% write reduction in three workloads, and a moderate 3MB RAM size can lead to 10.5%-18.4% write reduction. Doubling the size to 6MB only achieves a slight increase to 11.8%-19.8%. Therefore we consider a 3MB RAM, which maintains 128K entries as the standard size of FS.

C. Number of block erasures

Recyclable SSD utilizes page recycle to avoid block erasures. We measure the number of block erasures as the metric for the Recyclable SSD's performance on extending lifetime. Figure 11 shows the normalized number of block erasures compared to the baseline. The Recyclable SSD reduces the number of block erasures significantly, the maximum gain on reduction can be achieved when infinite FS containing sufficient entries, reporting 49.7%-69.1% reductions in the three workloads. This reduction mainly comes from two sources: first, the removed writes save the consumption of free pages, and reduce the GC operations which are strongly related to the amount of free pages; second, the recycled invalid pages can recreate valid pages, making the status of blocks healthier, and avoid the block being selected as a victim.

When FS maintains limited entries from 64K to 256K, the Recyclable SSD still can get considerable achievements compared to the maximum gain. For example, attributed to the high value locality, the *web* has exhibited the best results: the normalized number of block erasures under FS of 64K, 128K, and 256K are 45.2%, 41.1%, and 39.6% respectively, approaching the 38.2% maximum gain. But we also note that in the *mail* workload, which has the most invalid pages but least value locality, the normalized number of block erasures varies from 49.7% to 73.4% under FS from 256K to 64K. These findings suggest that the Recyclable SSD is strongly influenced by the value localities. In all the three workloads, the Recyclable SSD can benefit more when FS contains more entries, but the increase rate are very low after the number of entries reaches 128K, suggesting 128K FS is a better choice.

D. Average response time

While the lifetime is extended, the average response time of the Recyclable SSD also benefit a lot from write and block erasure reductions. The write reduction can decrease the number of page programming operations while the block erasure reduction can decrease the number of erase operations. Since programming and erasing are the most time-consuming operations in NAND flash memory, these decreasing can improve the write response time substantially. But we also note that calculating fingerprints affects performance, so we suggest to use a dedicated hashing engine [12] with 934MHz processor frequency, to reduce hashing latency. We simulate the hashing unit by modeling 32μs [11] latency of calculating and searching fingerprint into SSD extension. Therefore, in our experiments, the hashing overhead is merely observable.

In order to find out the effectiveness of recyclable probability calculation, we use two FS maintenance schemes: LRU and PROB. The LRU scheme manages the FS as a queue with least-recently-used eviction policy, while the PROB maintains the FS as an ordered queue which is sorted by the recycle probability. The capacity of FS is set to the optimal size 3MB.

As shown in Figure 12, under the LRU scheme, which is unaware of the recyclable probability, a large amount of recyclable invalid pages is missed, and the overhead of searching and generating fingerprint outpaces the benefits of page recycle. The average response time is then increased by 14.4%, 28.5%, and 17.3% in *home*, *mail*, and *web* respectively compared with the baseline. But the average response time under PROB scheme in all the three workloads are improved. The *web* workload, which has the most value locality, shows a 29.7% improvements in response times. The *home* and *mail* workloads also exhibit 17.2% and 13.3% improvements. These improvements prove that calculating recyclable probability can effectively keep those high recyclable invalid pages in FS and make the Recyclable SSD retain high access performance.

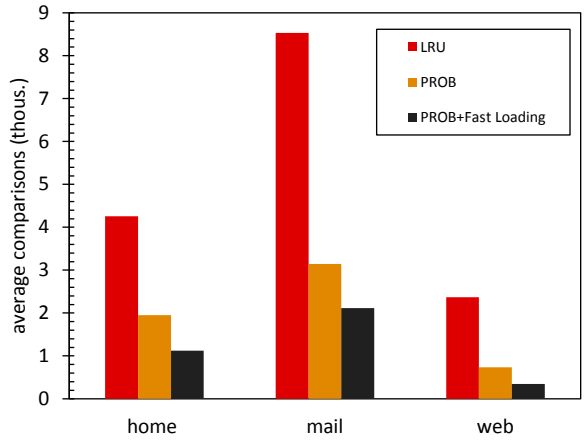

Fig. 13. Reduction of fingerprint comparisons

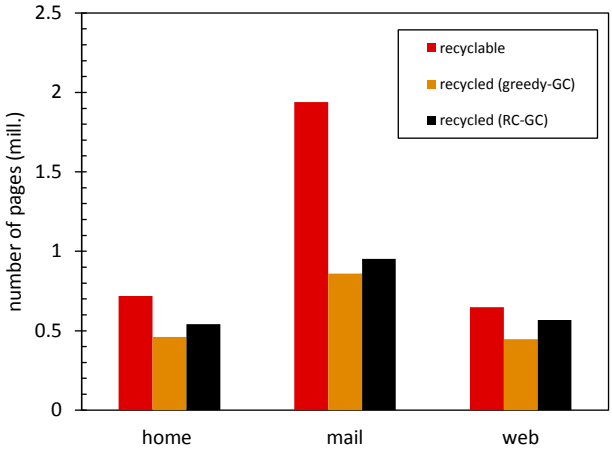

Fig. 14. Comparison of greedy-GC and RC-GC on the recycled invalid pages

E. Benefit of fast loading

The fast loading discovers the spatial locality existing in invalid pages and accelerates the speed of lookup in FS. Figure 13 shows the average number of fingerprint comparisons of LRU, PROB, and PROB plus Fast Loading. We configure the FS to the optimal size 3MB with 128K entries. Due to the lack of locality awareness, the LRU has the most fingerprint comparisons. The PROB which uses value popularity and value version to exploit value locality has decreased the average comparisons to around 1K times. The fast loading, utilizing spatial locality, can further achieve 32.8%-52.6% reduction compared with PROB. In web, for example, the PROB plus fast loading scheme can effectively minimize the average number of fingerprint comparisons to 0.3K times. Thus we apply the fast loading to speed up the lookups of FS.

F. Benefit of RC-GC

The reason of missing recyclable pages comes from two aspects. First, the limited FS capacity will lose the chance to recycle some recyclable pages; second, the GC may erase the block which still contains recyclable pages. Our RC-GC gives each invalid flash pages in the victim block with a different weight based on the recyclable probability and can effectively avoid blocks with high recyclable invalid pages being selected as the victim.

As shown in Figure 14, the greedy-GC can assist Recyclable SSD to recycle 64.1%, 44.4%, and 68.8% of recyclable pages, in *home*, *mail*, and *web*. But with the support of RC-GC, the Recyclable SSD can gain more, recycling 49.2%-87.5% of recyclable pages. Comparing with greedy-GC, the RC-GC can assist the Recyclable SSD to achieve a 10.8%-27.1% increase in the number of recycled invalid pages.

VI. RELATED WORK

NAND flash is a write-once memory, new data cannot be programmed into a used flash page without erasure. Flash cell reprogramming is strictly limited by the constraint that voltage level V_{th} can only be raised. Several recent studies propose the use of WOM codes to enable invalid pages to be used for extra writing [9, 25, 31, 41], and reduce the block erasures significantly. Other researchers propose shiftflash [3] which marks each invalid pages with time stamps and provides the SSD with time-shift function through retrieving the invalid pages.

There are extensive researches focusing on Flash Translation Layer, and most of them bring about improved performance or extended lifetime [20, 23, 33, 40, 45, 46]. Exploring localities is one approach to designing new FTL and overcoming flash memory inherent constraints. Temporal locality [34, 39] has been utilized by buffering writes in SSD to eliminate duplicated writes and extend lifetime. Aayush et al. exploits value locality and implements CA-SSD [21] to remove the duplicated writes inline and reduce write response time substantially. CAFTL [35] also utilizes inline deduplication to remove duplicated writes, but mainly focusing on enhancing the lifespan and saving space. CA-FTL is similar to our Recyclable SSD in the sense that they all utilize CAS to address flash pages. It performs deduplication only in the range of valid pages: reduce the write traffic through removing the duplicated writes in-line and coalescing duplicated pages out-of-line. Space is saved but at the cost of weakened redundancy. Our Recyclable SSD aims at the invalid pages, which is orthogonal to CA-FTL and can be combined with it to achieve further potential benefits. The Delta-FTL[42] exploits the content locality between the write data and its corresponding old version in the flash, instead of writing a brand new data it only stores the compressed delta in flash.

Other researchers attempt to extend the lifetime by reducing the write amplification [19, 38]. Eunji Lee et al. present a CDM (Cooperative Data Management) scheme to manage data in flash storage and nonvolatile memory (NVM), and allows some valid pages to be invalidated if they have a copy in NVM. This scheme can eliminate the copy-out operation, reducing write amplification substantially [38]. Similarly, Guangxia Xu et al. propose a swap-aware garbage collection algorithm NSAGC [19]. It considers valid flash pages which have modified copies in the

main memory as the latent invalid flash pages and skip the copy of these latent invalid pages to reduce write amplification of GC.

VII. Conclusion

Our study is the first attempt to address the benefit from recycling invalid pages without reprogramming or cleaning. We present the Recyclable SSD to extend SSD lifetime while slightly improving performance. The characterized localities in invalid pages minimize the overhead of identifying recyclable invalid pages, ensuring that the implementation of the Recyclable SSD is feasible. Using several real-world workloads, we have conducted an extensive evaluation on the Recyclable SSD. Results have shown that the number of block erasures can be decreased up to 58.9%, resulting in significant lifetime extension, while the overall I/O response time can be improved by up to 18.4%. Our design is applicable to existing flash architectures, requiring only minor adjustments within the FTL without interface modifications. Most related works for extending SSD lifetime are orthogonal to the design of the Recyclable SSD and can be augmented with recycling invalid pages.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their valuable comments that greatly improved this paper. Our research is supported in part by the National Natural Science Foundation of China under Grants 61232004 and 61502189，and the National Key Research and Development Program of China (No.2016YFB0800402).

References

[1] Agrawal, N., et al., "design tradeoff for ssd performance ", in *USENIX 2008 Annual Technical Conference*, 2008.

[2] "SSD extension for DiskSim simulation environment", *https://www.microsoft.com/en-us/download/details.aspx?id=52332*;

[3] Huang, P., K. Zhou, and C. Wu, "ShiftFlash: Make flash-based storage more resilient and robust", *Perform. Eval.*, 2011, 68(11): p. 1193-1206.

[4] John Bucy, J.S., Steve Schlosser, Greg Ganger, "The DiskSim Simulation Environment (v4.0)", *http://www.pdl.cmu.edu/DiskSim/*;

[5] Kang-Deog, S., et al., "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme", *IEEE Journal of Solid-State Circuits*, 1995, 30(11): p. 1149-1156.

[6] Chiang, M.L. and R.C. Chang, "Cleaning policies in mobile computers using flash memory", *J. Syst. Softw.*, 1999, 48(3): p. 213-231.

[7] "Hynix Semiconductor H27UBG8T2CTR-BC Datasheet", *http://www.datasheetspdf.com/*;

[8] "SLC, MLC and TLC NAND flash in Micron", *https://www.micron.com/products/nand-flash*;

[9] Berman, A. and Y. Birk, "Retired-page utilization in write-once memory — A coding perspective", in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, 2013.

[10] Jiangpeng Li, K.Z., and Xuebin Zhang, "How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime", in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015.

[11] HELION, "Fast hashing cores", *http://www.heliontech.com/fast_hash.htm*;

[12] "ARM Cortex R4", *https://www.arm.com/zh/products/processors/cortex-r/cortex-r4.php*, 2014;

[13] Deng, Y., "What is the future of disk drives, death or rebirth?", *ACM Computing Surveys (CSUR)*, 2011, 43(3): p. 23.

[14] M. Wu, a.W.Z., "eNVy: a non-volatile, main memory storage system", in *Proceedings of the 6th International Conference on Architecture Support for Programming Languages and Operating Systems*, 1994.

[15] Nath, P., B. Urgaonkar, and A. Sivasubramaniam, "Evaluating the usefulness of content addressable storage for high-performance data intensive applications", in *Proceedings of the 17th international symposium on High performance distributed computing*. 2008, ACM: Boston, MA, USA.

[16] Huang, Z., et al., "XI-Code: A Family of Practical Lowest Density MDS Array Codes of Distance 4", *IEEE Transactions on Communications*, 2016, 64(7): p. 2707-2718.

[17] A González, C.A., M Valero, "A data cache with multiple caching strategies tuned to different types of locality", in *ACM International Conference on Supercomputing 25th* 2014.

[18] pureSilicon, "1TB SSD with hardware based encryption", *http://www.marketwired.com/press-release/puresilicon-debuts-worlds-first-1tb-25-inch-ssd-most-compact-ssd-per-gb-1248837.htm*;

[19] Xu, G.X., M.M. Wang, and Y.B. Liu, "Swap-aware Garbage Collection Algorithm for NAND Flash-based Consumer Electronics", *IEEE Transactions on Consumer Electronics*, 2014, 60(1): p. 60-65.

[20] Lee, S., et al., "Lifetime management of flash-based SSDs using recovery-aware dynamic throttling", in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. 2012, USENIX Association: San Jose, CA.

[21] Aayush Gupta, R.P., Bhuvan Urgaonkar, and Anand Sivasubramaniam, "Leveraging Value Locality in Optimizing NAND Flash-based SSDs", *9th USENIX Conference on File and Storage Technologies*. 2011.

[22] Koller, R. and R. Rangaswami, "I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance", *Acm Transactions on Storage*, 2010, 6(3).

[23] Huang, P., et al., "An aggressive worn-out flash block management scheme to alleviate SSD performance degradation", in *Proceedings of the Ninth European Conference on Computer Systems*, 2014.

[24] Zhou, K., et al., "Deep self-taught hashing for image retrieval", in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015.

[25] Gala Yadgar, E.Y., and Assaf Schuster, "Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes", in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015.

[26] Desnoyers, P., "What systems researchers need to know about NAND flash", in *Proceedings of the 5th USENIX conference on Hot Topics in Storage and File Systems*. 2013, USENIX Association: San Jose, CA.

[27] Zhang, X., et al., "Reducing Solid-State Storage Device Write Stress through Opportunistic In-place Delta Compression", in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. 2016, USENIX Association: Santa Clara, CA. p. 111-124.

[28] Tsao, C.W., et al., "Efficient Victim Block Selection for Flash Storage Devices", *IEEE Transactions on Computers*, 2015, 64(12): p. 3444-3460.

[29] Liu, R.-S., et al., "DuraCache: a durable SSD cache using MLC NAND flash", in *Proceedings of the 50th Annual Design Automation Conference*. 2013, ACM: Austin, Texas.

[30] Motwani, R., Z. Kwok, and S. Nelson, "Low density parity check (LDPC) codes and the need for stronger ECC", *Flash Memory Summit*, 2011: p. 41-50.

[31] Margaglia, F., et al., "The Devil Is in the Details: Implementing Flash Page Reuse with WOM Codes", in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. 2016, USENIX Association: Santa Clara, CA. p. 95-109.

[32] Ungureanu, C., et al., "HydraFS: a high-throughput file system for the HYDRAstor content-addressable storage system", in *Proceedings of the 8th USENIX conference on File and storage technologies*. 2010, USENIX Association: San Jose, California.

[33] Dong, G.Q., Y.Y. Pan, and T. Zhang, "Using Lifetime-Aware Progressive Programming to Improve SLC NAND Flash Memory Write Endurance", *IEEE Transactions on Very Large Scale Integration (Vlsi) Systems*, 2014, 22(6): p. 1270-1280.

[34] Gokul Soundararajan, V.P., Mahesh Balakrishnan, Ted Wobber, "Extending SSD Lifetimes with Disk-Based Write Caches", in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST 10)*, 2010.

[35] Feng Chen, T.L., Xiaodong Zhang, "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives", in *9th USENIX Conference on File and Storage Technologies*. 2011.

[36] Fukuzumi, Y., et al., "Optimal Integration and Characteristics of Vertical Array Devices for Ultra-High Density, Bit-Cost Scalable Flash Memory", in *2007 IEEE International Electron Devices Meeting*, 2007.

[37] Min, J., D. Yoon, and Y. Won, *"*Efficient Deduplication Techniques for Modern Backup Operation", *IEEE Transactions on Computers*, 2011, 60(6): p. 824-840.

[38] Eunji Lee, J.K., Hyokyung Bahn, Sam H. Noh "Reducing Write Amplification of Flash Storage ", in *32nd International Conference on Massive Storage Systems and Technology*, 2016.

[39] Kim, H. 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, USENIX Association: San Jose, California.

[40] Im, S. and D. Shin, *"*ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer", *J. Syst. Archit.*, 2010, 56(12): p. 641-653.

[41] Grupp, L.M., et al., "Characterizing flash memory: Anomalies, observations, and applications", in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[42] Guanying Wu, X.H., "Delta-FTL:improving SSD lifetime via exploiting content locality*"*, in *7th ACM european conference on Computer Systems (EuroSys 12)*. 2012.

[43] Van Houdt, B., *"*Performance of garbage collection algorithms for flash-based solid state drives with hot/cold data", *Performance Evaluation*, 2013, 70(10): p. 692-703.

[44] Lin, M.W. and S.Y. Chen, *"*Efficient and Intelligent Garbage Collection Policy for NAND Flash-based Consumer Electronics", *IEEE Transactions on Consumer Electronics*, 2013, 59(3): p. 538-543.

[45] Huang, P., et al., "FlexECC: Partially Relaxing ECC of MLC SSD for Better Cache Performance", in *USENIX Annual Technical Conference*, 2014.

[46] Lee, S. and J. Kim, *"*Effective Lifetime-Aware Dynamic Throttling for NAND Flash-Based SSDs", *IEEE Transactions on Computers*, 2016, 65(4): p. 1075-1089.

[47] Hu, X.-Y., et al., "Write amplification analysis in flash-based solid state drives*"*, in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. 2009, ACM: Haifa, Israel.

[48] Wei, J., et al., *"*Efficiently representing membershipfor variable large data sets", *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(4): p. 960-970.