2R: Efficiently Isolating Cold Pages in Flash Storages

Minji Kang* Soyee Choi** Gihwan Oh** Sang-Won Lee**
*Samsung Electronics Co. Hwasung, Korea
minji10.kang@samsung.com
**Sungkyunkwan University, Suwon 16419, Korea
{ithdli, wurikiji, swlee}@skku.edu

ABSTRACT

Given skewed writes common in databases, the conventional *1R-Greedy* FTL incurs huge write amplification, most of which is contributed by cold pages amounting to 80% of data. Since *1R-Greedy* manages all flash blocks in *one region* at no type distinction, cold pages will be mixed with non-cold ones in the same blocks, spread across blocks over time, and thus repeatedly relocated upon garbage collections.

In this paper, we propose "*two region*" FTL (2R in short); with two flash regions of normal and cold, it focuses on isolating cold pages into cold region and thus preventing their repetitive relocations. 2R has two versions. The optimized version, *2R-FIFO*, can further prevent the problem of false cold pages in the basic version, *2R-Greedy*, by taking FIFO instead of Greedy as its victim selection policy. 2R is unique in that all its design decisions, including page placement and migration between regions, cold page identification, victim block selection, and space allocation among regions, capitalize on workload characteristics such as write skews, temporal locality, and frozen pages. Thanks to the principled approach, 2R is, unlike the existing hot/cold separation FTLs, a statistics-free and practical solution which can efficiently and effectively separate cold pages using only two regions.

Experimental results using a real OpenSSD as well as trace-driven simulations confirm that *2R-FIFO* can, compared to *1R-Greedy*, halve the write amplification in two OLTP benchmarks, TPC-C and LinkBench, thereby doubling IO performance and transaction throughput. In particular, as flash storage becomes nearly full, *2R-FIFO* starts outperforming *1R-Greedy* by 4x or more.

PVLDB Reference Format:
Minji Kang, Soyee Choi, Gihwan Oh, Sang-Won Lee. 2R: Efficiently Isolating Cold Pages in Flash Storages. *PVLDB*, 13(11): 2004-2017, 2020.
DOI: https://doi.org/10.14778/3407790.3407805

*Work done while in Sungkyunkwan University

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407805

1. INTRODUCTION

For the last decade, flash memory storages have been relentlessly replacing harddisks as the main storage across all sectors of computing, including smartphones, mobile devices, desktops, stand-alone servers, data centers, and even the cloud computing. In particular, the OLTP database market has been leading this trend because of the superiority of flash storages such as high random IOPS/$ and low power consumption [17].

At the core of any flash storage is the flash translation layer (FTL) [30], a firmware module between the host and the flash memory chips. Since overwrites are not allowed in flash memory chips and thus the copy-on-write approaches are taken, FTL has to manage the mapping between the data page's logical address and its current physical address in flash memory. In addition, when all flash blocks are in use, FTL has to reclaim new space by relocating valid data pages and erasing flash blocks. Note that the relocation of data pages during garbage collections amplifies the writes to flash memory. The write amplification is a fundamental and intrinsic issue in flash storages. Among numerous FTL schemes, the page-mapping FTL is the most popular mainly for performance reasons. The existing page-mapping FTL is characterized to manage all flash blocks under *one region* at no distinction of block type and to merge a single block chosen as a victim according to the *greedy* policy upon garbage collections. Throughout this paper, we call this de-facto standard FTL as *1R-Greedy*.

Meanwhile, *1R-Greedy* will incur excessive write amplification under skewed and random writes common in data-intensive applications. As a concrete example, the write amplification factor (WAF) has increased from one to five when the TPC-C benchmark is run on a real OpenSSD with *1R-Greedy* as its FTL. The ever-increasing WAF in turn halved the SSD's IOPS (IO per second) and the benchmark's transaction throughput. The amplified writes will also shorten the lifespan of flash storages. Nevertheless, *1R-Greedy* has been, since its inception almost two decades ago, used across almost every workload with inertia at no fundamental change. Note that the astonishing IO performance improvement in flash storages over the last decade is due to the architectural changes [11, 18, 26], not to the advances in FTL technique itself.

Then, why does *1R-Greedy* amplify writes so excessively under OLTP workloads? Most of its write amplification is, as analyzed in Section 3, contributed by *cold pages* accounting for 80% of all data but only for 20% of all writes. In *1R-Greedy* cold pages are mixed with non-cold ones in the

same flash blocks, spread across flash blocks over time and thus repeatedly relocated upon garbage collections.

To address the problem of cold page spreading in *1R-Greedy*, we propose "*two region*" FTL (hereafter 2R); with two regions of normal and cold flash blocks, it aims at *isolating cold pages* into cold region and thus minimizing their relocations. The main technical contributions of this paper are summarized as follows:

- We observe three characteristics of the write patterns in OLTP applications: write skewness, strong temporal locality, and frozen pages. They play a crucial role in making all design decisions of 2R(**Section 2**).
- We motivate that the cold page spreading is the main culprit for high WAF in *1R-Greedy*. Due to the *single-block-merge* mechanism in *1R-Greedy*, cold pages will be colocated with non-cold pages in same flash blocks, spread over flash blocks of one type, and then repeatedly relocated upon garbage collections. (**Section 3**)
- We propose two versions of 2R. On garbage collections, the basic version *2R-Greedy* merges multiple blocks at once from the same region and relocates their valid pages to a cold block. This *multi-block-merge* mechanism aims to disallow cold pages to mix with non-cold ones in same flash block. The cold pages isolated in cold region will be rarely relocated. (**Section 4.2**)
- The Greedy policy in *2R-Greedy* might allow non-cold pages to enter the cold region, severely offsetting its benefits. To prevent the problem of *false cold pages*, the optimized version *2R-FIFO* takes FIFO as its victim selection policy. *2R-FIFO* enhances the basic FIFO policy further with two optimizations: *selective merge policy* and *second chance policy*. (**Section 4.3**)

Meanwhile, 2R might not be such a novel considering there exist many hot/cold separation FTLs [12, 33, 36]. However, 2R differs from those schemes in that all its design decisions, including page placement and migration between regions, cold page identification, victim block selection, and space allocation among regions, are based on three workload characteristics mentioned above. Owing to the principled approach, 2R can, unlike the existing FTLs, isolate cold pages correctly in a statistics-free manner.

2. BACKGROUND

In this section, we review how *1R-Greedy* works and explain several related concepts. Also, we describe a few intriguing write characteristics in OLTP workloads, which are essential to 2R design.

2.1 1R-Greedy FTL

Page-Mapping FTL An FTL is responsible for several key functionalities such as address mapping, GC and wear level management [30]. Because overwrites are not allowed in flash memory, a new data page write should be handled in an out of place manner (*i.e.*, log-structured) - the old version of the page will be marked as invalid and new version will be stored in a new clean page. Thus, FTL has to manage the ever-changing address mapping between each page's logical address at the file system layer and its physical address in flash memory chips. Since the address mapping scheme is critical to the performance and lifespan of flash storages, numerous address mapping schemes such as block-mapping, page-mapping ,and hybrid-mapping [30] have been proposed since the inception of FTL techniques. Among them, the page-mapping FTL scheme is taken by most flash storage mainly for performance reason at the cost of memory resource for managing the logical-to-physical mapping at the page granularity [16, 18, 30].

Garbage Collection When clean space for new writes runs out, FTL has to reclaim new clean space by calling the garbage collection (GC hereafter) procedure. Upon GC, a victim block is chosen among all flash blocks in use, all valid pages in the victim are copybacked to a clean block B (*i.e.*, read out from the victim block and written back to the clean block), and then the victim is erased and returned as a free block. After GC, new writes from the host will be appended to the remaining clean pages in B. Since only a single block is merged upon each GC, we call the merge mechanism as a *single-block-merge*. However, it has an undesirable consequence: *cold pages can colocate with non-cold ones in the same flash blocks*. While pages relocated from a GC are presumably cold according to the temporal locality, pages newly written from the host are not. In this way, pages with different coldnesses will colocate in one flash block.

WAF The GC procedure causes write amplification because it has to relocate all valid pages from the victim to the new block. Informally, *write amplification factor (WAF)* represents the ratio of the number of physical writes to flash memory over the number of logical writes from the host. Let us denote the total number of pages in a flash block, the number of valid and invalid pages in a victim block as N, V, I, respectively. During GC, V pages have to be copied to another clean block B. After GC, the block B now has I (that is, N - V) free pages. To write I logical pages from the host, $I + V$ (*i.e.*, N) physical pages have to be written. Hence, WAF can be defined as follows:

$$WAF = \frac{I+V}{I} = 1 + \frac{V}{I}$$

Note that the last term in the above equation, V/I, represents the fraction of additional physical writes over the logical writes from the host. Ideal WAF will be one when the value of V/I becomes zero. The above equation indicates that minimizing V is crucial to reducing write amplification.

Victim Selection Policy Since the write amplification is proportional to the number of valid pages in victim blocks, the victim should be judiciously selected. The most popular is the greedy policy originating from log-structured file system [30, 33]. It chooses as the victim the flash block with the smallest number of valid pages. This heuristic is known to perform best (that is, to minimize write amplification) for uniform and random writes [21, 19]. The greedy policy is in a sense analogous to the LRU buffer replacement algorithm: the former aims to minimize WAF using the heuristic of choosing a block with the least number of valid pages while the latter does to maximize hit ratio with the heuristic of choosing the least recently used page.

1R-Greedy In addition, in the existing FTLs, all blocks are managed at no distinction of block type; the flash storage space is managed as one region, a pool of flash blocks

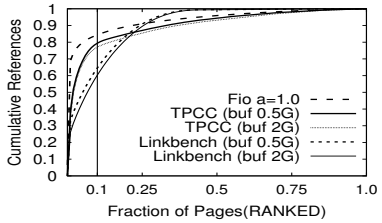


Figure 1: Write Skews in OLTP Workloads

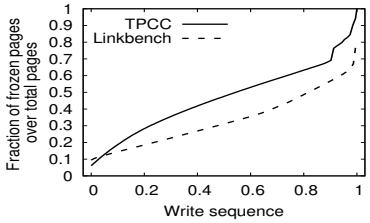


Figure 2: Frozen Pages in OLTP Workloads

of the same type. In summary, the standard FTL can be characterized as managing the address-mapping at the page granularity, choosing victim block using the greedy policy, merging single block per GC, and managing all flash blocks in one region [18], which we call *1R-Greedy* in this paper. Besides commercial SSDs, OpenSSD platform and OpenChannel SSD take *1R-Greedy* as default FTL [1, 2, 10].

Over-provisioning Another critical factor which determines WAF is over-provisioning (OP in short) [3]. Besides the logical capacity exposed to the host OS, most flash storages are equipped with extra physical capacity for better performance and endurance. The OP capacity is the difference between its physical capacity (P) and logical capacity (U), and the OP factor is defined as $(P - U)/U$. With a larger OP area, pages will have longer temporal chances to be invalidated so that victim blocks will have less valid pages. Although the OP factor in SSDs ranges depending on the products and the vendors, the 'factory over-provisioned' factor is quite limited (usually, less than 10%).

2.2 Characteristics in OLTP Write Traces

Understanding workloads is essential to storage system design. But, little work has been conducted on characterizing *write traces* in OLTP workloads, particularly from the perspective of *FTL design*. This section discusses three intriguing characteristics we have observed from realistic write traces: write skew, temporal locality, and frozen page. As explained later, we capitalize on them in designing 2R.

Write Skews It is well known that logical references to tuples at the buffer layer are heavily skewed in OLTP workloads [7, 28]. In addition, the page-level access skew at the buffer layer is virtually same as the tuple level skew [28]. Then, the following question would be whether the write skew still holds for at the block I/O layer.

To answer this question and to quantify the skewness factor in OLTP workloads, we collected the write traces while running TPC-C and LinkBench benchmarks on a commercial RDBMS and the MySQL/InnoDB engine, respectively. In addition, to check the effect of buffer size on the write skewness at the block I/O layer, we collected write traces for two buffer sizes, 0.5GB and 2GB, for each benchmark. For comparison purpose, we also collected a synthetic write trace by running the `fio` tool with the skewness factor (denoted as α) set to 1 (i.e., `random_distribution:zipf=1`). For each write trace, the pages were sorted in the descending order of their write frequency and then the cumulative fraction of the corresponding database versus the cumulative probability of writes was plotted as the x and y axis, respectively. The results are presented in Figure 1.

Consistently across three workloads, as shown in Figure 1, more than 80% of all writes were made to less than 20% of all pages. In other words, while only a small fraction of pages is very hot, the most page is very cold in terms of writes (*i.e.*, infrequently written). We would like to highlight two points from Figure 1. First, Zipfian write distributions are common and strong in realistic OLTP workloads. Second, the write skewness remains quite stable, regardless of database engines and buffer sizes, thanks to the self-similarity property of Zipf's s law [38]. Note that the results in Figure 1 are consistent with the observations in recent research [38] saying that 45 out of 48 real traces show strong Zipfian distributions. In summary, skewed writes are not specific to OLTP workloads but ubiquitous in many applications.

Temporal Locality Though all traces used in Figure 1 are commonly skewed in terms of *per-page write frequency*, two OLTP traces differ from the `FIO` trace in terms of *per-page write interval*. While the write intervals of each page are stationary in FIO, the write intervals of the most page in OLTP traces reveal *strong temporal locality*. That is, in OLTP traces, once a page is written, regardless of its total write count, it tends to be repeatedly and intensively written during a short period and then suddenly not written for a relatively long period. In other words, each page's per-page writes exhibit alternating hot and cold phases and the phase transition is not incremental but *stark*.

The strong temporal locality in per-page writes is attributable to the data processing logic. For instance, while new order entries are inserted to `new_order` and `order_line` tables in the TPC-C benchmark [28], the corresponding data pages will be repeatedly written until they are full of new tuples, and then those pages suddenly become and remain cold quite long until their orders are delivered later.

Frozen Pages It is common in many applications that a large fraction of data pages eventually become *frozen*: most data pages are, once created, actively updated for a while, and then eventually no longer updated but remain ever valid. In this sense, a frozen page is a special type of cold page. Figure 2 illustrates how the fraction of frozen pages change over time in two OLTP traces used for Figure 1. In each workload, the fraction of frozen pages is larger than 40% at the middle point and becomes greater than 60% at the fourth-fifth point. In the case of the TPC-C benchmark [28], most frozen pages comes from three tables of `order_line`, `order`, and `history`, which are used to log all the transactional activities of order-payment-delivery history. In a separate experiment, a similar observation about the frozen pages holds for TPC-E [11]. The existence of frozen pages is not workload-specific. Instead, it would be quite natural in any ever-growing database that most newly

created pages become eventually frozen rather than being permanently updated. In contrast, the FIO trace has no frozen page because every page is continuously updated.

3. MOTIVATIONS

3.1 Write Amplifications in Real SSDs

To understand the effect of write amplifications on IO performance and ultimately on the transaction throughput, we measured both cumulative and running WAFs, IOs per Second (IOPS), and transactions per minute (TPM) while running the TPC-C benchmark using a commercial RDBMS and a commercial SSD. The results are plotted in Figure 3. The SSD with 800GB logical capacity and 10% OP factor is presumably known to use *1R-Greedy* as its FTL. The cumulative WAF was measured every 10 minutes using the S.M.A.R.T interface [23, 34] and the running WAF was derived from the cumulative WAF. With an initial database of 400 GB, the benchmark was run for four days until the growing database filled up the remaining 400GB space.

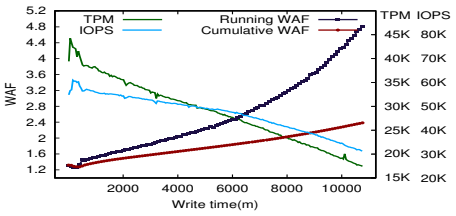


Figure 3: Real WAF on a Commercial SSD (TPC-C)

As shown in Figure 3, While cumulative and running WAF has ever increased over time to 2.4 and 5, respectively, IOPS and TPM has decreased from 60K to 30K and from 40K to 17K, respectively. Because the SSD was initially half-full and thus the remaining SSD capacity played the role of the over-provisioning area for FTL, the running WAF remains somewhat low in the first half period of the experiment. But, as the SSD was filled up with the ever-growing database and thus the over-provisioning area in effect was ever-shrinking, the running WAF continued increasing. In a separate experiment running the same TPC-C benchmark on two SSDs from other vendors, we have also observed that their IO and transaction performance behave almost same to Figure 3. To address the performance drop in SSDs, it is even recommended to use up to 70% of the SSD capacity [22]. In that 30% of the SSD space is traded to sustain the performance, the tuning guide is similar to the *short-stroking* strategy for harddisk [27]. However, considering the price of an SSD is mostly determined by its physical capacity, the trade-off is not an economical option. Hence, to maximize the cost-benefit of SSD, it is compelling to devise a better FTL which can keep WAF low and thus sustain the IO performance high till SSD is mostly full.

Effect of Cold pages on Write Amplification To investigate why the commercial SSD yielded high write amplification for OLTP workloads, we calculated two statistics while replaying the last one-tenth of the TPC-C trace used in Figure 3 against an *1R-Greedy* simulator: the average utilization of every victim block and the average number of copybacks for all pages. First, the average block utilization of victim blocks was about 0.8. This high utilization will, according to the WAF formula in Section 2.1, explain why the running WAF has reached up to 5 in Figure 3. Second, the average number of copybacks of all pages was about 10. In particular, some frozen pages have been relocated even more than 30 times. Recalling that valid pages in victims are presumably cold, cold pages tend to be repeatedly relocated. We conclude from these statistics that cold pages with long write intervals account for most of write amplification in *1R-Greedy*. In particular, frozen pages with infinite write interval, as detailed in Section 5, contribute to more than 80% of write amplifications in OLTP workloads. Under *1R-Greedy*, logically frozen pages are not physically frozen but they are continuously relocated instead.

3.2 Cold Page Spreading

Then, what is the culprit for the excessive copybacks of cold pages in *1R-Greedy*? The answer is the *single-block-merge* mechanism. The mechanism has, as explained above, an undesirable consequence that cold pages are allowed to colocate with non-cold pages in the same flash blocks. To be worse, the mechanism incurs the phenomenon of *cold page spreading*; cold pages spread across flash blocks as GCs proceed. As cold pages are evenly distributed over flash blocks, the utilizations of victim blocks will increase and thus more valid pages will be copybacked on GCs.

To confirm whether the phenomenon of cold page spreading indeed exists in *1R-Greedy* and, if so, how rapidly and uniformly cold pages spread, we carried out another experiment. While replaying the TPC-C trace used in Figure 3 using *1R-Greedy* simulator, we measured the fraction of cold pages in each of all flash blocks at three-time points in the trace (*i.e.*, initial, middle, and final), sorted all flash blocks in the ascending order of cold page fraction in them, and plot the results in Figure 4(a). At the initial point of the trace most cold pages are stored only on half of all flash blocks (the 'Initial' graph in Figure 4(a)). But, they have gradually spread across all flash blocks over time (the 'Middle' graph in Figure 4(a)), and the fraction of cold pages in every flash block became more than 60% in the end (the 'Final' graph in Figure 4(a)).

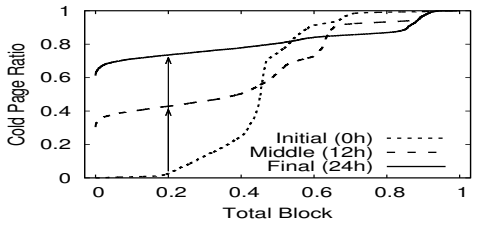
For comparison purpose, we also carried out the same experiment using the FIO trace (α=1.0) and present the result in Figure 4(b). In this experiment, we measured the fraction of cold pages in flash blocks at six points of time. Note that cold page spreading proceeds much faster in FIO than in TPC-C. For instance, cold pages have almost evenly spread already at the early phase of the FIO trace (the 'Initial(0.2h)' graph in Figure 4(b)). Taking into account that the SSD was full of data from the beginning, this fast-spreading of cold pages in FIO is not surprising.

4. DESIGN OF 2R

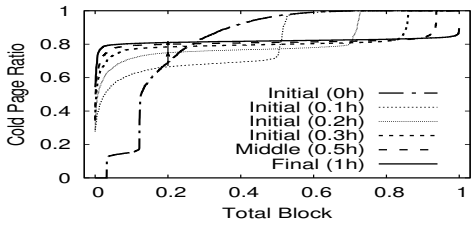
This section presents the design of 2R. It has two versions, *2R-Greedy* and *2R-FIFO*. They differ in the victim selection policy. We will explain the design principles of 2R and describe how each version works.

4.1 Design Principles

The root cause of excessive WAF in *1R-Greedy* is that its *single-block-merge* allows cold pages to colocate with non-cold ones in the same flash blocks. This recognition led us



(a) TPC-C



(b) FIO (α=1.0)

Figure 4: Cold Page Spreading (OP factor = 10%)

to take the first design principle: *"do not mix cold pages with non-cold ones in same flash blocks."* To realize this design principle, we propose "two-region" FTL (2R hereafter). In 2R, flash blocks are distinguished into two types, normal and cold; normal blocks are to store non-cold pages while cold blocks are to store cold pages. Thus, the flash memory space is divided into two regions of normal and cold. The main goal of 2R scheme is to minimize the relocations of cold pages by isolating them into the cold region while keeping non-cold pages in the normal region (that is, bimodal distributions of cold and non-cold pages in two regions).

Though it looks simple, however, designing an FTL with two regions raises several technical issues: which region to place new writes from the host, how to identify cold pages, when and which page to migrate between regions, how to allocate space to each region, and the victim selection policy. In addition, for each issue, there might exist two or more alternatives. Therefore, the design space of FTL with two regions could be very large. For this reason, we take another design principle for 2R: *"all its design decisions should be capitalized on three write characteristics: write skew, temporal locality, and frozen pages."*

These two design principles make 2R *statistics-free* and thus a viable solution for real SSDs. The only statistic 2R needs is the number of valid pages for each block, which commercial SSDs already maintain. In contrast, the existing hot/cold separation FTLs require per-page and/or per-block metadata to classify pages' hotness [12, 33, 36].

4.2 2R-Greedy

We first describe the basic version, *2R-Greedy*. We will explain how it handles such issues as page placement, page migration, cold page identification, and GC. While explaining these issues, the architecture of 2R in Figure 5 and the write and GC procedures in Algorithm 1 will be referred to.

4.2.1 Where To Place New Writes From The Host?

With two regions, a page writes from the host can be placed, depending on the page's hotness, on either normal or cold region. In fact, in the case of existing hot/cold separation FTL schemes supporting multiple regions, a page write will be placed on proper regions depending on the page's hotness which is calculated from its per-page statistics [12, 36]. In contrast, 2R simply places every writes from the host on the normal region ((1) in Figure 5). If the previous version is in cold region, the page now migrates to normal region ((5) in Figure 5). Otherwise, the page remains in normal region ((2) in Figure 5). Note that either type of migration does not cause additional copyback.

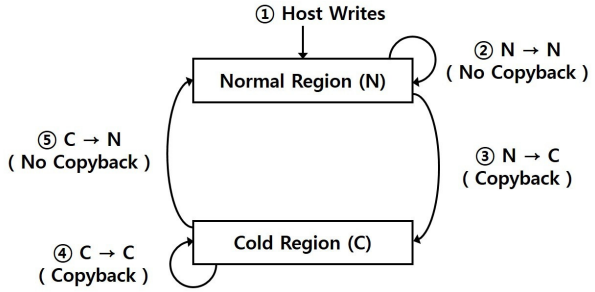


Figure 5: Page Migrations in 2R

This decision on the page placement in 2R is based on a simple heuristic that every newly written page be non-cold at the time of the write. The heuristic is, in turn, based on the write skewness and the temporal locality in OLTP write traces. That is, recalling that more than 80% of all writes are made against hot pages and also that a page is, once written, likely to be re-written soon, the heuristic that any newly written page is in its hot phase would be correct in most case. Although wrong in some cases, this heuristic, unlike the existing solutions [12, 36], does not require any extra overhead to determine page hotness.

Refer to the procedure `Write` in Algorithm 1, where the variable `CurBlk` indicates current normal block where new page writes will be placed. In *2R-Greedy*, all flash blocks are initially used as normal blocks. When all flash blocks are consumed and thus no free space is available for new writes in `CurBlk`, a clean normal block has to be secured by triggering the `GarbageCollect` procedure (line 4 in Algorithm 1).

4.2.2 Cold Page Identification

2R aims at achieving *bimodal distribution* by keeping non-cold pages in normal region while isolating cold pages in cold region. For better bimodality, it is crucial to identify cold pages correctly. The cold page identification in *2R-Greedy* resorts to another heuristic: valid pages in victim blocks will be cold. This heuristic holds true in most cases because a page is unlikely to be written soon, according to the temporal locality, if the page remains intact since its last write or copyback to the victim block. This heuristic will be further supported by two characteristics in OLTP workloads: the stark hot/cold phase transitions in per-page writes and frozen pages. Following the heuristic, all valid pages in victim blocks will be migrated to cold region ((3) or (4) in Figure 5). In summary, the cold page identification in *2R-Greedy* is based solely on temporal locality.

Algorithm 1 *2R-Greedy* Algorithm

```
 1: procedure WRITE (P)
 2:     /* CurBlk: current normal block for new write */
 3:     if no free page in CurBlk then
 4:         CurBlk = GarbageCollect();
 5:     end if
 6:     /* every host write goes to normal region */
 7:     write P to CurBlk;
 8:     mark (if any) P's old version in either region
 9: end procedure
10:
11: procedure GARBAGECOLLECT()
12:     /* victim selection using the Greedy policy */
13:     select N victim blocks from one region
14:     copy valid pages in victims to N-1 cold block(s);
15:     erase N victim blocks;
16:     return a secured FreeBlock;
17: end procedure
```

4.2.3 Garbage Collection and Page Migration

Like *1R-Greedy*, upon GC, *2R-Greedy* also chooses the block with the least number of valid pages as the first victim, regardless of its block type. But, unlike *1R-Greedy* taking *single-block-merge*, *2R-Greedy* merges *two or more blocks from the same region* at once in each GC. By merging multiple blocks from the same region, *2R-Greedy* aims to disallow cold pages from normal victim blocks to mix with colder pages from cold victim blocks.

2R-Greedy chooses multiple blocks (N) from the same region as victims, compacts all valid (and presumably cold) pages in victims into (N-1) cold blocks, and thus secures a completely empty block. The number of blocks to be merged at once, N, will be determined as follows: the total capacity of invalid pages in the blocks should be equal to or greater than the size of one flash block. In this way, *2R-Greedy* intends to strictly separate non-cold pages from cold ones, thus meeting its first design principle: *"do not mix cold pages with non-cold pages in the same flash block."* For the GC algorithm in *2R-Greedy*, refer to the `GarbageCollect` procedure in Algorithm 1.

When normal blocks are merged on GC, every valid page in victim blocks has to migrate to cold region ((3) in Figure 5). Note that this migration causes one additional write. Once moved to cold region, a page will stay there until the new write is made for the page. While staying at cold region, cold pages should not, ideally, experience further relocations. But, cold pages with long write intervals as well as frozen pages might experience repetitive copybacks within cold region ((4) in Figure 5) when their belonging blocks become victim for GC.

Although 2R does not explicitly distinguish page coldness at three or more levels, merging multiple cold blocks will have the effect of isolating very cold pages and frozen pages implicitly into a deeper level. More deeply those cold pages are isolated, the less likely they experience copyback. In particular, frozen pages, after experiencing a few copybacks within cold region, will not be relocated any more; they are now frozen physically as well as logically. In this way, *2R-Greedy* will isolate both frozen pages and very cold pages deeply into cold region and thus minimize their write amplification. That is, with 2R multiple stages of coldness are implicitly classified within cold region by merging cold blocks. For this reason, we argue that two regions are enough to workloads with skewness, strong temporal locality, and frozen pages.

In 2R, the `GarbageCollect` procedure in Algorithm 1 automatically adapts the sizes of normal and cold region. That is, depending on the type of victim blocks, one region grows while the other shrinks. For instance, if N victims are chosen from normal region, cold region grows as cold pages amounting to (N-1) blocks are migrated from normal region. In this way, the space allocation to regions in 2R will adapt to the changing workloads, requiring no tuning. In addition, because the type of victim blocks is highly dependent on the victim selection policy, the victim selection policy is, as illustrated in Section 5, critical to the actual space allocation to each region.

4.3 2R-FIFO

4.3.1 *False Cold Pages in* 2R-Greedy

Though quite effective in isolating cold pages, as demonstrated later, *2R-Greedy* suffers from the problem of *false cold pages*. Informally, a page is defined as *false cold* when soon-to-be-invalidated thus non-cold in reality but, under *2R-Greedy*, misregarded as cold and thus migrated to cold region. In fact, the non-marginal fraction of the pages entering cold region in *2R-Greedy* is false cold. False cold pages will exacerbate the write amplification in two ways. First, each false cold page incurs one copyback from normal to cold region. Second and even worse, they pollute the cold region. As false cold pages are soon invalidated after migrated to cold region, the utilizations of cold blocks will be accordingly lowered. Cold blocks with low utilization are then likely to become victims for GC, which in turn cause even truly cold pages in the blocks to be relocated. To sum, the problem of false cold pages offsets the benefit of cold page isolation in *2R-Greedy*.

The Greedy policy itself is the culprit for false cold pages in *2R-Greedy*. That is, with regard to cold page identification, the policy contradicts the principle of temporal locality in two ways. First of all, since only the number of valid pages in blocks are taken into account in victim selection, normal blocks are more likely to become victim over cold ones. This is especially true with *young* normal blocks in which many non-cold pages happen to be colocated. As a result, non-cold but still-valid pages in such normal blocks will migrate to cold region. Second, because the policy prefers normal blocks as victim and thus the normal region is ever-shrinking, non-cold pages in normal region will have less temporal chance to be invalidated and thus more likely to become false cold pages. This *vicious circle* will make *2R-Greedy* less effective in filtering false cold pages.

4.3.2 2R-FIFO

To prevent the problem of false cold pages, we made three changes to the GC mechanism in *2R-Greedy*: the FIFO victim selection policy, selective merge policy, and the second chance policy. One common goal of these changes is to give non-cold pages in normal region a longer temporal chance to be invalidated and thus to identify cold pages more faithfully to the principle of temporal locality.

FIFO Policy The FIFO policy is used to choose the victim blocks (*i.e.*, *2R-FIFO*). To embody the FIFO policy,

all flash blocks of normal and cold are maintained as a single linked list. Each normal or cold block is appended to the `tail` of the list in the order of its allocation. In addition to the `head`, the FIFO list has three pointers, `cur_scan`, `cur_cold_blk`, and `cur_blk`. The `cur_scan` pointer indicates the block up to which the list is scanned. The `cur_cold_blk` pointer refers to the newest cold block to which the next cold pages will be appended. The `cur_blk` is the normal block at the tail of the list, to which normal writes are made. Once the current normal block, *cur_blk*, is full, *2R-FIFO* starts scanning the list from the `cur_scan` block in a circular order and chooses, as in *2R-Greedy*, multiple victim blocks of the same type per a GC so as to secure a completely clean block. All valid pages in victims will be copied to the `cur_cold_blk` blocks. Then, each victim block will be removed from the list and the `cur_scan` pointer is also adjusted to point to the next candidate victim block in the list. The clean block obtained from the GC is then appended to the tail of the list as normal block and will be used as `cur_blk` to store new writes from the host until it is full. Note that, in *2R-FIFO*, all normal and cold blocks are interleaved in the list in the order of their allocations. The next victim block selection starts from the `cur_scan`, not from the list's head. With the FIFO policy, each non-cold page in normal blocks will now have a more temporal chance to be invalidated until its belonging block becomes the victim.

Selective Merge Policy While scanning blocks in the FIFO list, *2R-FIFO* selectively chooses victim blocks. That is, a block is selected as a victim *only if* its utilization is less than a threshold value (*i.e.*, `BLK_UTIL`), but otherwise skipped. This *selective merge* policy is taken for three reasons. First of all, merging blocks with high utilization will secure only limited space at the cost of relocating numerous valid pages. Second, by skipping blocks with large utilization, valid pages in the blocks will have another chance to be invalidated during the sweep of the circular buffer list. This *second chance* effect will help in identifying truly cold pages. Finally, by merging cold blocks with low utilizations eagerly, the policy can allocate the space trimmed from the cold blocks to normal region. In this way, the space utilization of cold region will improve and also, more importantly, the normal region will shrink at a slower pace than *2R-Greedy*. This virtuous cycle can effectively mitigate the problem of false cold pages. We have empirically found out that *2R-FIFO* performs best with `BLK_UTIL` set to 0.5. With higher threshold value, the copyback overhead of valid pages offsets the benefit of cold page isolation. With lower value, in contrast, *2R-FIFO* suffers from space under-utilization, causing higher write amplification.

Second Chance Policy As `cur_blk` approaches the tail of the FIFO list, the pages at the tail blocks will have less temporal chance to be invalidated because they are recently written or copybacked. Since the pace of scanning the list is faster than that of appending new blocks at the list tail, we decided to give a *second chance* to the blocks near the tail of the list. For this, when reaching a predefined depth of the list, `FIFO_SCAN_DEPTH`, *2R-FIFO* stops scanning the list and instead start scanning from the list head. This second chance mechanism will prevent non-cold pages at the tail of the list from falsely entering cold region.

It is obvious that the performance of *2R-FIFO* will be highly dependent on the parameter `FIFO_SCAN_DEPTH`. To understand the effect of `FIFO_SCAN_DEPTH` on write amplification, we measured the cumulative WAF for the TPC-C trace used in Figure 1 by varying the parameter from 60% to 100%, and presented the result in Figure 6. With the parameter set to 80%, as indicated by the 'Total' line in Figure 6, *2R-FIFO* yields the lowest WAF: the number of copybacked pages in Y-axis is smallest. With higher `FIFO_SCAN_DEPTH`, *2R-FIFO* also will tend to falsely migrate the non-cold pages from the young blocks at FIFO tail to the cold region, thereby causing more non-cold pages to be relocated, as illustrated by the 'HOT' line in Figure 6. With lower `FIFO_SCAN_DEPTH`, in contrast, *2R-FIFO* is likely to choose old blocks at the FIFO head as victim and has to merge cold pages in the blocks, thus causing more cold pages to be relocated, as indicated by the 'COLD' line in Figure 6.

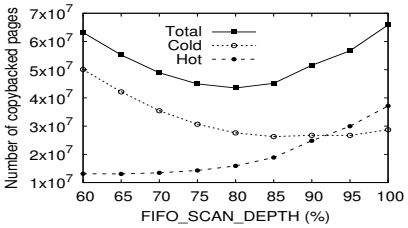


Figure 6: Effect of Varying `FIFO_SCAN_DEPTH` on WAF

4.3.3 Recovery

FTLs should be able to recover from any unexpected failures. For the simplicity of discussion, all the key data structures of *2R-FIFO* are assumed to reside in the battery-backed non-volatile cache [26], including its page-mapping table, three pointers, and the linked FIFO structure. With the help of durable cache, it would be trivial to make the key data structures in *2R-FIFO* consistent and persistent despite of sudden failures. One exception is the FIFO list. When a crash is encountered at any point while manipulating the linked list, it would be time-consuming to recover the broken list. A better solution is to make the update operations for the list atomic. For instance, in order to insert/remove a block to/from the middle of the FIFO list, two pointers have to be updated atomically. The existing solutions for non-blocking linked lists will shed light on this issue [20].

4.3.4 MBM-Induced Tail Latency

In 2Rs, merging multiple blocks (*i.e.*, *MBM*) from the same region at once upon garbage collection plays a crucial role in strictly segregating cold pages from non-cold ones. The *MBM* scheme will, however, induce longer tail latency for write operation: a write operation encountering a garbage collection has to wait until multiple blocks are completely merged and erase. In the case of *1R-Greedy*, in contrast, each garbage collection merges only a single victim block so that the tail latency for write operations will be shorter. As the long tail latency issue becomes more critical in modern applications [15], the latency variability resulting from *MBM* has to be properly addressed. The semantics of *MBM*, fortunately, does not enforce the synchronous merge implementation. Instead, merging multiple blocks can be implemented asynchronously. That is, once all valid pages in the first victim block are successfully relocated and the

block is erased, the write operation can proceed while other victim blocks are merged in a background way.

Before closing this section, we would like to stress that *2R-FIFO* is a viable solution for commercial SSDs in two aspects. First, *2R-FIFO* can be implemented with minimal changes to the existing *1R-Greedy*, as detailed in Section 5.3. Second, *2R-FIFO* is efficient in terms of run-time overhead. Upon every GC, for instance, *2R-FIFO* can find the next victim blocks simply by scannng the FIFO list, while *1R-Greedy* and *2R-Greedy* have to compare the numbers of valid pages in all blocks. In addition, unlike other existing FTLs, it can isolate cold pages in a statistics-free manner.

5. PERFORMANCE EVALUATION

5.1 Experimental Setup

We have implemented a simple trace-driven simulator for each of three FTLs and measured WAF while running each simulator using three traces, `FIO`, TPC-C, and LinkBench. WAF is a good performance indicator in comparing different FTL schemes since both IOPS and TPS are, as illustrated in Figure 3, almost inverse-proportional to WAF. For all experiments, the page size was set to 4KB, the flash block was to have 1,024 pages (*i.e.*, 4MB). Besides, while running each trace, the size of flash memory was by default configured to have 10% over-provisioning (OP) in addition to the final database size. The three I/O traces were collected using the `blktrace` tool [8] while running a synthetic `FIO` tool [9], and also running TPC-C and LinkBench. For each OLTP workload, the buffer size was set at 10% of its initial database size. Below are described the three workloads.

FIO `FIO` (Flexible I/O tester) is a synthetic tool that generates I/O patterns as indicated by users [9]. Using the configurable option `random_distribution:zipf=`α in the tool, you can specify the skewness of data access. By default, `FIO` generates uniformly random distribution for random IOs (that is, $\alpha = 0$). We collected a random write trace while writing 90 million of 4KB pages (*i.e.*, 360GB in total) against a fixed database of 8GB using `FIO` with α set to 1.0.

TPC-C The TPC-C trace was obtained while running the benchmark [28] using the Benchmark Factory tool [35] on a popular commercial DBMS. While running the benchmark, the database engine wrote about 105 million of 4K pages (480 GB in total) and the database size had grown from 1GB to 10GB.

LinkBench LinkBench is an open-source benchmark modeling the social graph data at Facebook [7]. This trace was collected while running the LinkBench (version 2.0) tool [6] on MySQL/InnoDB (version 5.7.2). While running the benchmark, the database engine wrote about 300 million of 4K pages (1.2TB in total) and the database size had grown from 2GB to 30GB.

5.2 Performance Analysis

5.2.1 Overall Performance

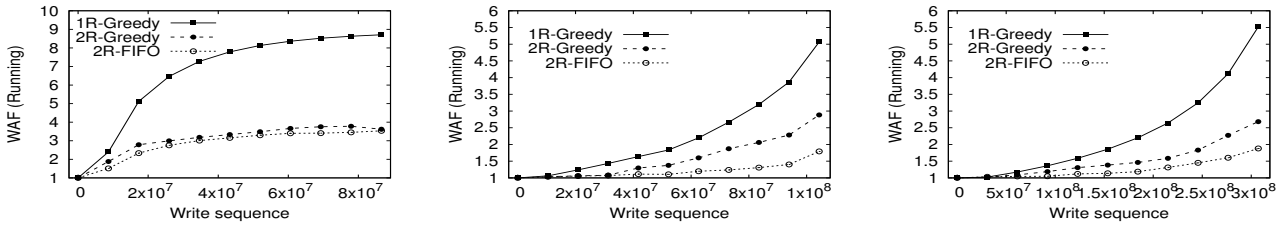
Let us briefly compare the overall performance of three FTLs using Figure 7. While running three I/O traces on three FTLs, we measured the running WAF at every tenth interval point in each trace and present the results in the figure. In the figure, the X-axis represents the sequence of page writes in each trace and the Y-axis does the running WAF which means the ratio of total internal physical writes to flash memory over total logical writes made so far in the trace. As shown in Figure 7, 2R schemes outperform *1R-Greedy* considerably in terms of WAF consistently across all three I/O traces. In particular, it should be noted that the running WAF gaps among three FTLs are ever-growing over time for both TPC-C and LinkBench workload and thus *2R-FIFO* outperforms *1R-Greedy* by more than 2.5 folds in the end. Recalling that the data is growing over time, the effect of isolating cold pages in 2R schemes becomes outstanding as the SSD is filled with more data. It is also noteworthy that the running WAF gap between *2R-FIFO* and *2R-Greedy* is also substantial in both OLTP traces.

5.2.2 Performance Analysis: FIO

***1R-Greedy* vs. 2R schemes:** It is clear from Figure 7(a) that *1R-Greedy* results in high WAF for the FIO trace. In particular, the running WAF spikes at the early phase of the trace and is ever-increasing, though steadily, until the end of the trace. This high WAF in *1R-Greedy* is mainly due to its cold page spreading which was illustrated in Figure 4(b). Recall that cold pages have rapidly spread over all flash blocks in the early stage of the FIO trace and continue spreading till the end of the trace. In contrast, the WAFs in *2R-Greedy* as well as *2R-FIFO* reach the plateau early at the one-fourth point of the trace, then remaining almost stable until the end of the trace. This clearly indicates that for the FIO trace with skewed writes (*i.e.*, α=1.0) 2R schemes can effectively prevent cold pages from spreading instead isolating them into cold region, thus limiting their relocations. In fact, according to a separate analysis, most write reduction of 2Rs over *1R-Greedy* is attributable to cold pages with quite long write intervals.

***2R-Greedy* vs. *2R-FIFO*:** With the FIO trace, as indicated in Figure 7(a), the WAF gap between *2R-FIFO* and *2R-Greedy* is, compared to the one between *2R-FIFO* and *1R-Greedy*, quite small. This is due to the write characteristics in FIO trace: the hotness of each page is statistically determined, its write count is inverse-proportional to its hotness, and, more importantly, each page's write interval is uniform (*i.e.*, no temporal locality in page writes). Therefore, the oldest normal blocks will have the smallest number of valid pages so that both *2R-Greedy* and *2R-FIFO* are likely to choose the same oldest normal blocks as victims. In particular, because the oldest blocks tend to be victim, *2R-Greedy* will not suffer from the problem of false cold pages. Instead, because of its eager cold block merge, *2R-FIFO* can outperform *2R-Greedy*, though marginal.

Effect of Skewness Given highly skewed writes (*e.g.*, $\alpha = 1.0$), 2Rs outperform *1R-Greedy*. To understand the effect of skewness, we collected six write traces using running the `FIO` tool by increasing the skewness factor α from zero (*i.e.*, uniform random) to 1.0 (*i.e.*, highly skewed) at the interval of 0.2, measured the cumulative WAFs while running three FTLs using those traces, and present the result in Figure 8. It is clear from Figure 8 that the benefit of 2Rs over *1R-Greedy* becomes outstanding as the writes are more skewed (*i.e.*, as α increase from 0.6 to 1.0). As the writes are more skewed, 2Rs can identify cold pages more correctly and isolate truly cold pages into cold region while



(a) FIO (α=1.0) (b) TPC-C (c) LinkBench

Figure 7: Running WAF (Over-provisioning = 10%)

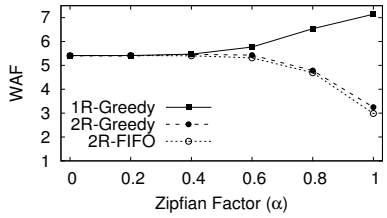


Figure 8: Varying Skewness in FIO

1R-Greedy suffer from the cold page spreading. Meanwhile, given less skewed writes (*i.e.*, $\alpha \leq 0.4$) where the most page has almost similar write interval, 2Rs can hardly distinguish cold pages from non-cold ones so that they perform closely to *1R-Greedy*. It should be noted, however, that 2Rs are not inferior to *1R-Greedy* even for uniform writes (*i.e.*, $\alpha = 0.0$) which is the worst scenario for 2Rs. In this regard, 2Rs are resilient to low skewness. In a separate experiment using the real Cosmos board, we verified that the cumulative WAFs of *1R-Greedy* and *2R-FIFO* are almost the same to those in Figure 8. Meanwhile, the cumulative WAFs of three commercial SSDs are almost same to that of *1R-Greedy* in Figure 8. In summary, we would like to stress that 2Rs are not workload-specific but workload-agnostic.

5.2.3 Performance Analysis: TPC-C and Linkbench

Before analyzing the effect of our 2R schemes on OLTP workloads in detail, let us make two general observations about the results in Figure 7. First, across three FTLs, running WAFs in OLTP traces are quite lower than those in the FIO trace. In particular, even with *1R-Greedy*, the running WAF in each OLTP trace is quite small and increasing slowly in the first half of the trace. This is mainly because each OLTP trace was collected from an initially small but ever-growing database while the FIO trace was from an initially full and fixed database. In the case of OLTP traces, when the database size is relatively small, the remaining unused logical space in flash storages will play the role of over-provisioned area in effect [3]. Second, each FTL shows similar trends in WAF over time for both TPC-C and LinkBench. This is because they are similar in their write characteristics such as skewness, temporal locality, and frozen pages. For this reason, our performance comparison and analysis of three FTLs will focus on the TPC-C trace.

1R-Greedy* vs. *2R-FIFO As Figure 7.(b) illustrates, *2R-FIFO* starts outperforming *1R-Greedy* at the early phase of the TPC-C trace, the running WAF gap between two FTLs is ever-widening over time, and eventually at the last interval in the trace, *2R-FIFO* can reduce the running WAF by 1/3. Recalling that the write performance is inverse to the internal WAF, this result in turn implies that *2R-FIFO* can improve the random write I/O performance over *1R-Greedy* by three folds in principle. This huge performance gain is, as demonstrated in Section 5.3, indeed achievable with our real implementation.

2R-FIFO* vs. *2R-Greedy As shown in Figure 7.(b), *2R-FIFO* starts outperforming *2R-Greedy* considerably in terms of running WAF from the middle of two OLTP traces. In addition, in terms of cumulative WAF at the end of both traces, *2R-FIFO* and *2R-Greedy* achieves the cumulative WAF of 1.23 and 1.61, respectively. Again, there are two sources of WAF reduction in *2R-FIFO* over *2R-Greedy*: false cold pages and frozen pages. First of all, the main problem with *2R-Greedy* is that the write amplification due to non-cold pages (*i.e.*, pages whose write interval is than DB size) is quite large (*i.e.*, about 0.4), which is, to our surprise, even higher than that in *1R-Greedy*. This inefficiency is due to the problem of *false cold pages* in *2R-Greedy*. In contrast, since *2R-FIFO* can prevent pages with short write intervals from falsely entering cold region, it can reduce the write amplification of non-frozen pages, compared to 0.4 in *2R-Greedy*, to 0.1. Second, the write amplifications of frozen pages are approximately 0.2 in *2R-Greedy* and 0.1 in *2R-FIFO*. This improvement is another positive side-effect of preventing false cold pages in *2R-FIFO*. That is, as less false cold pages pollute cold region in *2R-FIFO*, the cold blocks will be less merged and accordingly the frozen pages in cold blocks will be less copybacked.

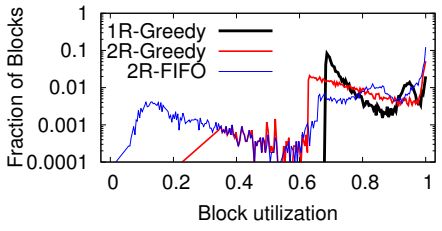


Figure 9: Bimodal Block Utilization (TPC-C)

Bimodality in Block Utilizations As discussed in Section 4, 2R will show the bimodal distribution of block utilizations. To verify the superiority of *2R-FIFO* over *2R-Greedy* in bimodality, we measured the utilization of all flash blocks at the three-fourths point while running the TPC-C

trace for each FTL including *1R-Greedy*, and plotted the result in Figure 9. Note that the y-axis is log-scale. *1R-Greedy* shows no bimodality because it has a single block type: most block tends to linger just above the cleaning point [33].

2R-FIFO is more clear than *2R-Greedy* in terms of the bimodality, which can in turn explain the WAF gap between them. Compared to *2R-Greedy*, *2R-FIFO* will choose normal blocks with quite lower utilization as victim: the average utilization of all normal victim blocks in *2R-FIFO* and in *2R-Greedy* was 0.16 and 0.41, respectively.

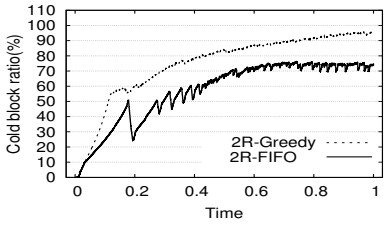


Figure 10: Adaptive Space Allocation (TPC-C)

Adaptive Space Allocation In the TPC-C trace, as the database grows over time, the number of cold and frozen pages also increases. Therefore, the cold region has to accordingly expand to accommodate more cold pages. 2R schemes are novel in that they allocate space adaptively to flash regions, thus requiring no tuning effort. To verify how adaptively 2Rs allocate blocks to cold region, we measured the fraction of cold blocks over all blocks while running the trace and present the result in Figure 10. Note that Y-axis in the figure ranges up to 110% because the OP factor is set to 10%. While both schemes allocate more blocks to cold region over time, the fraction of cold region in *2R-FIFO* is less than that in *2R-Greedy* about by 0.2 across all ranges greater than 0.1 in X-axis. This result indicates that *2R-FIFO* is better than *2R-Greedy* in terms of the space utilization of cold region. *2R-FIFO* can utilize the cold region better because it prevents false cold pages from entering the cold region, and also because it takes the eager cold block merge. With larger normal region, non-cold pages in normal blocks will have more temporal chances to be invalidated before the normal blocks become victims. This is the reason why the average utilization of normal victim blocks in *2R-FIFO* was lower than that in *2R-FIFO*.

Effect of OP Factor on WAF The performance and endurance of flash storages are highly dependent on the over-provisioning (OP) capacity. Irrespective of FTL schemes, WAF will decrease with larger OP area because most non-cold page will have more temporal chance to be invalidated before its belonging blocks become victim for garbage collection. To evaluate the impact of the OP capacity on write amplification, we measured cumulative WAF of each FTL for the TPC-C trace by varying the OP factor from 5% to 50% over the final database size, and the results are presented in Figure 11. We would like to highlight several points from Figure 11. First, *2R-FIFO* outperforms *1R-Greedy* substantially across all OP ranges tested. Second, the WAF gap between *2R-Greedy* and *2R-FIFO* is rapidly narrowing as the OP area is growing. This is because a larger OP area can drastically mitigate the problem of false cold pages in *2R-Greedy*. Third, as the OP factor is reduced from

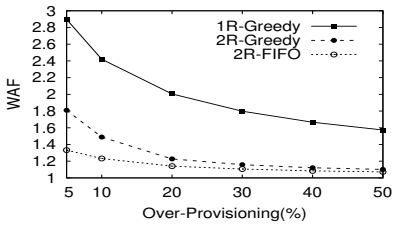


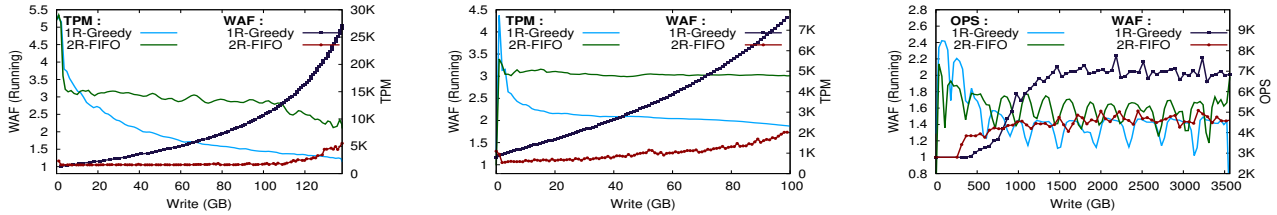
Figure 11: Effect of OP Factor on WAF (TPC-C)

20% to 5%, the WAF increase in *2R-FIFO* is gentle while they are steep in *1R-Greedy* and *2R-Greedy*. This indicates that the benefit of *2R-FIFO* becomes more outstanding with smaller OP factor. Lastly, and most importantly, *2R-FIFO* with only 5% OP factor wins over *1R-Greedy* even with 50% OP factor. Given that the price of an SSD is highly dependent on its physical flash capacity, this indicates that *2R-FIFO* can, compared to *1R-Greedy*, cut the cost of SSDs down by 40% for the TPC-C workload, while providing the same or better performance. This can in turn justify the cost-benefit of *2R-FIFO* over *1R-Greedy*.

5.3 Experimental Results using Real SSD

To evaluate the effect of *2R-FIFO* on a real system, we have prototyped *2R-FIFO* on the OpenSSD Cosmos board [5] and compared it with *1R-Greedy*, the default FTL in the board, by running TPC-C, LinkBench, and YCSB benchmarks on each of two FTLs. *2R-FIFO* was implemented with minimal change of 52 lines to the *1R-Greedy* codebase on the board. The OpenSSD Cosmos board employs the HYU Tiger 4 controller based on Dual-Core ARM Cortex-A9 on top of Xilinx Zynq-7000 board, and 32GB MLC Nand flash memory. All the experiments were conducted on a Linux platform with the 4.15 kernel running on an Intel Core i7-4770 3.4GHz processor with 16GB DRAM. The size of over-provisioning area on the board was set to 10%.

TPC-C MySQL/InnoDB engine (version 8.0.2) and the `tpcc-mysql` tool was used to run the TPC-C benchmark. While running the benchmark against each FTL, the initial database of 1.5GB was created and then 16 threads of the benchmark tool concurrently ran until the database grew up to the full capacity of SSD (*i.e.*, 28GB). The database page size was set to 16KB which is the default in the database engine and the buffer size was to 512MB. While running the benchmark for each FTL, the running WAF and TPM were measured at every 10 seconds and plotted in Figure 12(a). In the figure, the X-axis represents the amount written from the host, and the left and the right Y-axis does the running WAF and the transaction throughput, respectively. Note that the throughput and running WAF of *1R-Greedy* in Cosmos board over time change very similarly in those of real commercial SSD in Figure 3. The average TPM of *2R-FIFO* (13,512) is 1.9x better than that of *1R-Greedy* (7,092) and the last 1% TPM of *2R-FIFO* (9,912) surpasses that of *1R-Greedy* (2,432) by 4.07x. The huge throughput gap between *2R-FIFO* and *1R-Greedy* is a direct reflection of WAF reduction. As shown in Figure 12(a), the last 1% WAF of *2R-FIFO* was 1.7 while that of *1R-Greedy* was 5. Finally, it should be noted that the running WAF of *2R-FIFO* has sustained less than 1.1 until the SSD is almost full, which is very close to the ideal WAF of 1.



(a) TPC-C on MySQL (b) LinkBench on MySQL (c) YCSB on RocksDB

Figure 12: OLTP Performance on Cosmos OpenSSD

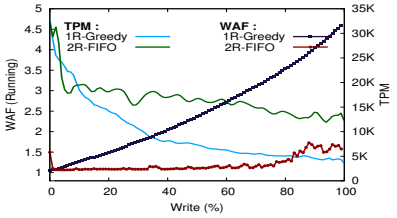


Figure 13: Multi-Tenant TPC-C Performance

LinkBench With the same InnoDB setting as in TPC-C, we measured the running WAF and TPM at every 10 seconds while running LinkBench on each FTL and presented the results in Figure 12(b). In this experiment, the initial database of 5GB was created and then 32 thread of the benchmark tool concurrently ran until the database grew up to the full capacity of SSD (*i.e.*, 28GB). *2R-FIFO* outperforms *2R-Greedy* about by two times in terms of both average and last OPS. Likewise the TPC-C case, the significant throughput gain of *2R-FIFO* over *1R-Greedy* in LinkBench is also due to the WAF gap between two FTLs. The cumulative WAF of *1R-Greedy* and *2R-FIFO* is 2.1 and 1.28, respectively and their running WAFs at the last point of time are 4.3 and 1.8, respectively. Note that the running WAF trends in both *1R-Greedy* and *2R-FIFO* and their WAF gap over time in Figure 12(b) are very close to those in Figure 7(c).

YCSB In addition, to compare the performances of *1R-Greedy* and *2R-FIFO* for NoSQL applications, we measured OPS (operation per second) and WAF while running the update-only YCSB workload [14] with RocksDB [4] on top of each FTL and present the results in Figure 12(c). For this experiment, 32GB MLC Nand flash memory was added to the board so as to store the initial database of 35GB and also to accommodate the space amplification required from RocksDB. While running the benchmark, RocksDB wrote a large number of SSTables whose total size amounts to 350GB.

To our surprise, even for the YCSB benchmark, *2R-FIFO* wins over *1R-Greedy* in terms of average OPS (5,216 vs. 4,685) as well as the running WAF (2 vs. 1.4). This result is counter-intuitive: we expected that the running WAF be very low (*i.e.*, close to 1) even with *1R-Greedy* since the write patterns in RocksDB are mostly sequential (*e.g.*, 32MB-long SSTables) and thus flash-friendly. But, our preliminary analysis reveals the writes from RocksDB are also skewed so that *2R-FIFO* outperforms *1R-Greedy*.

Multi-Tenant Databases With the ever-increasing capacity of SSDs, it would be common for multiple databases to share a single large SSD. In particular, with more applications moving to the cloud, more database instances serviced by cloud vendors will run in multi-tenant environment [29]. To evaluate the effect of *2R-FIFO* on multi-tenant databases, we measured total TPSs of four TPC-C benchmarks concurrently running on a single Cosmos SSD with each of *1R-Greedy* and *2R-FIFO*, and also measured the running WAF of the SSD, and present the results in Figure 13. Each database's size was set to one-fourth of single database used for Figure 12(a). Not surprisingly, the performance gap between *1R-Greedy* and *2R-FIFO* under multi-tenant databases is very close to the gap under single tenant database (Figure 12(a)). Regardless of the number of concurrent tenants, the benefit of *2R-FIFO* will hold as long as the writes are skewed.

6. RELATED WORKS

2R aims at separating cold pages from non-cold ones into the cold region and thus reducing write amplification. In this regard, there exist two types of related work: hot/cold separation FTLs schemes [12, 36] and multi-stream SSDs [25]. In addition, the cold page isolation in 2R is analogous to the hot page filtering in 2Q buffer algorithm [24].

Hot/Cold Separation FTLs Numerous hot/cold separation FTL schemes [12, 36] have been proposed to reduce write amplification by placing pages with different hotness into different regions. Indeed, most hot/cold separation FTL can be traced back to LFS [33] which pioneered the idea of separating hot/cold data and storing them in different segments. In particular, the authors of LFS showed that the victim selection policy is crucial in segregating hot/cold data correctly. Existing hot/cold separation FTLs have extended LFS mainly in two ways. First, unlike LFS, they explicitly divide the flash storage into multiple regions. Second, they determine each page's hotness based on either the update recency [12] or the update frequency [36].

But, existing hot/cold separation FTLs incur both space and computation overheads to maintain per-page hotness metadata and estimate the page hotness on demand. For instance, for each page, the *DAC* scheme [12] keeps track of its region number as well as the last-written-time in the page-mapping table, and calculates each page's hotness on-demand to decide its migration to neighbor region on write or GC. In addition, the cost-benefit victim selection policy used in DAC as well as LFS incur overly run-time overhead. To be specific, to select a victim upon every GC, they

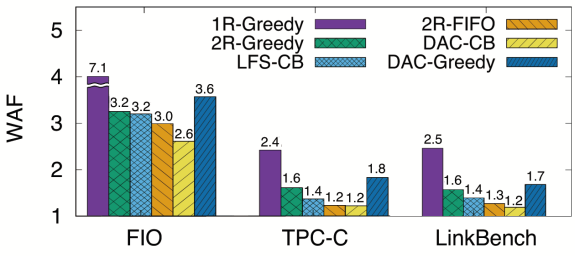


Figure 14: Comparison with other FTLs (OP=10%)

have to apply the formula, $a \times (1-u)/u$, to every block, where a and u represent the block's age and utilization, respectively, and further have to compare all blocks (*e.g.*, 1M blocks) so as to select a block with the largest cost-benefit. Therefore, though quite effective in separating hot/cold data and thus reducing write amplification, those statistics-heavy FTLs have been discarded for commercial SSDs.

To quantitatively compare 2Rs against existing hot/cold separation FTLs, we made simulators for DAC [12] and LFS [33]. Using each simulator, we measured the cumulative WAF while running each of FIO, TPC, and LinkBench traces used for Figure 7, and presented the results in Figure 14. For all three traces, DAC performs best with the cost-benefit policy and four regions [12] and LFS does with the cost-benefit policy [33]. Thus, their results are denoted as `DAC-CB` and `LFS-CB`, respectively, in the table. In addition, for comparison, the WAFs of *1R-Greedy*, *2R-Greedy*, and *2R-FIFO* for the same traces are summarized in the table. Below we briefly analyze the result of LFS and DAC.

`LFS-CB` achieves quite low WAF for each of three traces, which is in between the WAFs of *2R-Greedy* and *2R-FIFO*. This confirms that LFS can effectively segregate hot/cold pages even with a single region. Though, `LFS-CB` underperforms *2R-FIFO* mainly for two reasons. First, the cost-benefit policy in LFS can choose blocks which cost high but benefit marginally as victim. That is, the $a \times (1-u)/u$ formula can in some cases prefer old blocks nearly full of cold pages to young blocks with low utilizations. Note that such blocks are excluded as victim by the selective merge policy in *2R-FIFO*. Second, because it has one write stream, LFS allow pages written from host to colocate pages cleaned from garbage collection in the same flash block.

`DAC-CB` performs best across three traces in terms of WAF. With the help of per-page update recency metadata, `DAC-CB` with four regions can place or migrate pages to the proper region. Also, it can choose right victim blocks using the cost-benefit policy. Therefore, as intended, `DAC-CB` can effectively cluster data pages, according to their hotness, into different regions. Interestingly, when the Greedy policy is instead used, `DAC-Greedy` underperforms even *2R-Greedy* for all traces. But, recall that the run-time overhead for managing per-page and/or per-block hotness information in `DAC-CB` is unacceptable to resource-scarce SSDs. In contrast, *2R-FIFO* is a viable solution for SSDs because it performs close to `DAC-CB` while it works in a statistics-free manner.

Multi-Stream SSDs A novel interface for flash storages, Multi-Stream SSD (MS-SSD), was recently proposed and standardized [25, 37], which allows applications to place pages with different lifetimes to different streams (*i.e.*, different flash blocks). Both MS-SSD and 2R aim to reduce write amplification by separating pages with different hotness into different flash blocks. When properly hinted by applications, this interface is effective in reducing write amplification [25]. With MS-SSD, however, an application has to determine the number of streams to be used in advance and further has to designate the proper stream-id for every write call. In particular, given the stark and irregular hot/cold phase transition in per-page writes in OLTP workloads, any database storage engine itself could not, upon writing a dirty page to the storage, predict its hotness correctly and thus assign the right stream to the page. For this reason, it is a daunting task to leverage MS-SSD for OLTP workloads, and the performance gain is quite limited with an enormous tuning effort [13] or even storage engine modification [32]. In contrast, without any user hint or any complicated tuning effort, 2R can isolate cold pages into cold region and at the same time performs nearly optimally.

2Q Buffer Replacement Algorithm The goal of FTL is similar to that of the buffer replacement algorithm: an FTL aims at minimizing write amplification with the limited over-provisioning space while a buffer replacement algorithm does at maximizing hit ratio with the limited buffer size. Most buffer replacement algorithm capitalizes on temporal locality in data accesses to identify hot pages and keep them buffered. Among numerous algorithms, 2Q [24] is known to perform as well as LRU-2 [31] but to have constant time overhead and require no tuning. In its simplified version, the buffer space is divided into two buffers, A1 and Am. On the first reference to a page, 2Q places it in A1 queue. If the page is re-referenced during its Al residency, then it is regarded hot and thus moved to Am, the main buffer. The key idea of 2Q is to filter hot pages using A1 buffer according to the temporal locality. Likewise 2Q, 2R identifies cold pages from normal region according to the temporal locality. Thus, it is also a practical and low-overhead solution. But, one key design consideration in 2R FTL is how to prevent cold pages from mixing with non-cold ones upon garbage collections. 2Q is free from such complicated issue.

7. CONCLUSION

In this paper, we showed that given highly skewed writes common in OLTP workloads, cold pages are spread over flash blocks and thus cause excessive write amplification under the existing *1R-Greedy* scheme. To remedy this problem, we proposed two-region FTL, 2R, which focuses on isolating cold pages into the cold region and thus minimizing their write amplification. In particular, since all its design decisions are based on the write characteristics in OLTP workloads, 2R is very practical; unlike the existing solutions, it does not require per-page statistics, any tuning effort, or any user hint. Also, experimental results using a real SSD confirm that the optimized version, *2R-FIFO*, can achieve almost optimal WAF (*i.e.*, 1.2 or less) for OLTP workloads.

Acknowledgements

We would like to thank the anonymous PVLDB reviewers for their valuable comments. This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) (No.2015-0-00314) and in part by the National Research Foundation of Korea (NRF) (No.2018R1A2B2005502).

8. REFERENCES

[1] OpenSSD Project. http://www.openssd-project.org/wiki/The_OpenSSD_Project, 2019.

[2] OpenSSD Project. http://lightnvm.io/liblightnvm/, 2019.

[3] Over-Provisioning. https://en.wikipedia.org/wiki/Write_amplification, 2019.

[4] Rocksdb: A persistent key-value store for fast storage environments. https://rocksdb.org, 2019.

[5] The OpenSSD Project. http://www.openssd.io, 2019.

[6] T. G. Armstrong. Facebook Graph Benchmark. https://github.com/facebookarchive/linkbench, 2013.

[7] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. LinkBench: A Database Benchmark Based on the Facebook Social Graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1185–1196. ACM, 2013.

[8] J. Axboe. blktrace:Block layer IO tracing tool. http://git.kernel.org/cgit/linux/kernel/git/axboe/blktrace.git, 2006.

[9] J. Axboe. FIO (Flexible IO Tester). http://git.kernel.dk/?p=fio.git;a=summary, 2006.

[10] M. Bjørling, J. Gonzalez, and P. Bonnet. LightNVM: The Linux Open-Channel SSD Subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, pages 359–374. USENIX Association, Feb. 2017.

[11] S. Chen, A. Ailamaki, M. Athanassoulis, P. B. Gibbons, R. Johnson, I. Pandis, and R. Stoica. Tpc-e vs. tpc-c: Characterizing the new tpc-e benchmark via an i/o comparison study. *SIGMOD Rec.*, 39(3), Feb. 2011.

[12] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using Data Clustering to Improve Cleaning Performance for Flash Memory. *Software Practice and Experience*, 29(3):267–290, Mar. 1999.

[13] S. Choi, H. Park, and S. W. Lee. Don't write all data pages in one stream. In *Proceedings of the 21th International Conference on Extending Database Technology*, EDBT 2018, pages 654–657, 2018.

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, 2010.

[15] J. Dean and L. A. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, Feb. 2013.

[16] P. Desnoyers. Analytic Models of SSD Write Performance. *ACM Transactions on Storage*, 10(2):8:1–8:25, Mar. 2014.

[17] J. Gray and B. Fitzgerald. Flash Disk Opportunity for Server Applications. *Queue*, 6(4):18–23, July 2008.

[18] 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*, ASPLOS XIV, pages 229–240, 2009.

[19] R. Haas and X.-Y. Hu. The fundamental limit of flash random write performance: Understanding, analysis and performance modelling. *IBM Research Report (RZ3771)*, Mar. 2010.

[20] T. L. Harris. A Pragmatic Implementation of Non-Blocking Linked-Lists. In *Proceedings of the 15th International Conference on Distributed Computing*, DISC '01, pages 300–314, 2001.

[21] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write Amplification Analysis in Flash-based Solid State Drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 10:1–10:9. ACM, 2009.

[22] M. Huc. Why SSD Performance Slows Down As It Becomes Full. https://pureinfotech.com/, Jan. 2020.

[23] Intel. Solid-State Drives in Server Storage Applications. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ssd-server-storage-applications-paper.pdf, 2014.

[24] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 439–450. Morgan Kaufmann Publishers Inc., 1994.

[25] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho. The Multi-streamed Solid-State Drive. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14. USENIX Association, 2014.

[26] W.-H. Kang, S.-W. Lee, B. Moon, Y.-S. Kee, and M. Oh. Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 529–540. ACM, 2014.

[27] S.-W. Lee, B. Moon, and C. Park. Advances in Flash Memory SSD Technology for Enterprise Database Applications. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 863–870, 2009.

[28] S. T. Leutenegger and D. Dias. A Modeling Study of the TPC-C Benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 22–31. ACM, 1993.

[29] F. Li. Cloud-Native Database Systems at Alibaba: Opportunities and Challenges. *PVLDB*, 12(12):1942–1945, 2019.

[30] Ma, Dongzhe and Feng, Jianhua and Li, Guoliang. A Survey of Address Translation Technologies for Flash Memories. *ACM Computing Survey*, 46(3):36:1–36:39, Jan. 2014.

[31] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 297–306. ACM, 1993.

[32] H.-W. Park, S. Choi, M. An, and S.-W. Lee. Freezing Frozen Pages with Multi-Stream SSDs. In *Proceedings*

of the 15th International Workshop on Data Management on New Hardware, DaMoN'19, 2019.

[33] M. Rosenblum and J. K. Ousterhout. The Design and Implementation of a Log-structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

[34] Samsung. Samsung Solid State Drive White Paper. `http://www.samsung.com/global/business/semiconductor/minisite/SSD/us/html/whitepaper/whitepaper.html`, 2013.

[35] Q. Software. Benchmark Factory for Databases. `http://www.quest.com/benchmark-factory/`, 2018.

[36] R. Stoica and A. Ailamaki. Improving Flash Write Performance by Using Update Frequency. *PVLDB*, 6(9):733–744, 2013.

[37] William Martin(T10 Technical Editor). SCSI Block Commands - 4 (SBC-4) (Working Draft Revision 9): 4.34 Stream Control. `http://www.t10.org/cgi-bin/ac.pl?t=f&f=sbc4r09.pdf`, November 2015.

[38] Y. Yang and J. Zhu. Write Skew and Zipf Distribution: Evidence and Implications. *ACM Transactions on Storage*, 12(4):21:1–21:19, June 2016.