# DLSpace: Optimizing SSD Lifetime via An Efficient Distributed Log Space Allocation Strategy



HUI SUN, Anhui University, China
JIANZHONG HUANG, Huazhong University of Science and Technology, China
XIAO QIN, Auburn University, USA
CHANGSHENG XIE, Huazhong University of Science and Technology, China

Due to limited numbers of program/erase cycles (i.e., P/Es) of NAND Flash, excessive out-of-place update and erase-before-write operations wear out these P/Es during garbage collections, which adversely shorten solid state disk (i.e., SSD) lifetime. The log space in NAND Flash space of an SSD performs as an updated page's buffer, which lowers garbage-collection frequency while reducing consumption of P/Es to extend SSD lifetime. In this article, we propose DLSpace, a novel distributed log space allocation strategy named *d*istributed *l*og *space*, which divides log space into block-level log space and page-level log space to significantly optimize SSD lifetime. DLSpace's log page space is dedicated to data pages in a data block. Such log page space only buffers page-update operations in this data block; thereby the use of log blocks for postponing garbage collection delays. DLSpace is conducive to fully utilizing pages in data and log blocks to avoid erasures of blocks with free pages. Consequently, DLSpace decreases write amplification by reducing excessive valid page-rewrite and block-erase operations under random-write-intensive workloads. We carried out quantitative research on the extension of SSD lifetime by virtue of three metrics (i.e., write amplification, the number of block-erase operations, and the delay time before the first garbage collection occurring). Experimental results reveal that compared with the existing *t*raditional allocation strategy for *l*og *space* (i.e., TLSpace), DLSpace reduces write amplification and the number of erase operations by up to 55.2% and 64.1% to the most extent, respectively. DLSpace also extends TLSpace's delay time of garbage collections by 73.3% to optimize SSD lifetime.

CCS Concepts: • **Information systems → Flash memory**; • **Computer systems organization → Embedded software**; **Secondary storage organization**;

Additional Key Words and Phrases: Distributed log space, traditional log space, lifetime, write amplification, solid state disk

We thank our anonymous reviewers for their valuable feedback and advice. This work is sponsored in part by the National Natural Science Foundation of China under Grant No 61702004 and No. 61572209. Xiao Qin's work is supported by the U.S. National Science Foundation under Grants IIS-1618669, CCF-0845257 (CAREER), CNS-0917137, and the 111 Project under Grant No. B07038. This work is also supported in part by the Anhui University Funding for Doctoral Research (J01003214), the Natural Science Foundation of Anhui Province (1708085QF160), and Key Technology R&D Program of Anhui Province (1704d0802193), and 2017 Opening Foundation of Key Laboratory of Intelligent Computing & Signal Processing, Ministry of Education (Anhui University).

Authors' addresses: H. Sun, Anhui University, 111 jiulong RD, Hefei, Anhui, 230601, China; email: sunhui@ahu.edu.cn; J. Huang (corresponding author), Huazhong University of Science and Technology, 1037 Luoyu RD, Wuhan, Hubei, China; email: hjzh@hust.edu.cn; X. Qin, Auburn University, Auburn; email: xqin@auburn.edu; C. Xie, Huazhong University of Science and Technology, 1037 Luoyu RD, Wuhan, Hubei, China; email: cs_xie@hust.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
1539-9087/2018/12-ART92 $15.00
https://doi.org/10.1145/3284749

**ACM Reference format:**
Hui Sun, Jianzhong Huang, Xiao Qin, and Changsheng Xie. 2018. DLSpace: Optimizing SSD Lifetime via An Efficient Distributed Log Space Allocation Strategy. *ACM Trans. Embed. Comput. Syst.* 17, 6, Article 92 (December 2018), 33 pages.
https://doi.org/10.1145/3284749

## 1 INTRODUCTION

NAND Flash-based solid state disks (i.e., SSDs) increasingly became popular owing to data reliability, high performance, and low energy consumption. SATA3.0, NVMe (non-volatile memory express) (Kim et al. 2016; Xu et al. 2015), next-generation form factor (NGFF or M.2) (Kyrnin 2017), PCI-E, and SSD array eliminate the bottleneck between storage devices and a host with high ultra-fast performance. However, SSD lifetime, which is dependent on its limited write endurance of NAND Flash, imposes a barrier to its widespread applications in enterprise and consumer areas (Grupp et al. 2012; Mohan et al. 2010). Write endurance is intimately related to program/erase cycles (i.e., P/Es) (Wei et al. 2016) in NAND Flash. With the increasing density of cells in NAND Flash (Wang et al. 2012a), the number of available P/Es significantly decreases. NAND Flash space in an SSD is divided into data space and log space (a.k.a., over-provisioning) (Lim et al. 2010; Smith 2012). This research is motivated by the following three shortcomings observed in NAND Flash-based devices:

- Restricted operations (i.e., sequential program (write), out-of-place update and erase-before-write) and a limited number of P/Es due to physical characteristics of floating gate transistors in NAND Flash (Lee et al. 2002).
- Excessive page program and block-erase operations are harmful to NAND Flash write endurance and SSD lifetime (Schroeder et al. 2017).
- Existing log-space allocation strategies may fail to stop extra program and erasure operations, which wears down the limited number of P/Es, especially under random-write-intensive workloads (Sun et al. 2015).

Because of out-of-place update and erase-before-write operations, data in an updated page are rewritten to a free one in another block or the same block after erasure. Write amplification (Hu et al. 2009; Lu et al. 2013), incurred by rewriting a handful of valid pages during garbage collection, widely exists in NAND Flash-based devices. A high write amplification indicates that more data are written in NAND Flash space than that issued from the user I/O load. In NAND Flash space, amounts of the program and erase operations are triggered, and the limited number of P/Es are worn down, which shortens SSD lifetime due to adverse impacts on NAND Flash endurance.

The log space (i.e., over-provisioning), which is reserved for an SSD controller and cannot be accessed by operating systems, buffers updated pages from data space. Log space not only reduces the frequency of excessive program and erase operations but also lowers degradation of P/Es in NAND Flash, which improves its write endurance. In previous work (see, for example, Hu et al. (2009), Lim et al. (2010), Hu and Haas (2010), Frankie et al. (2011), Lee et al. (2012), and Xiang and M (2012)), the traditional log managements can be applied in various mapping schemes such as page-level, block-level, and hybrid-level ones. Most of the log space managements, however, embrace block-level strategies. We refer to this traditional block-level log space allocation strategies as TLSpace, which reveals three weaknesses: (1) merge operations can trigger excessive page program operations and large write amplification (see Figure 2(b) in Section 2); (2) free pages are underutilized in data and log blocks, where the number of P/Es can be worn down during merge operations. Figure 1 presents the usage of data blocks with valid, invalid, and free data pages under

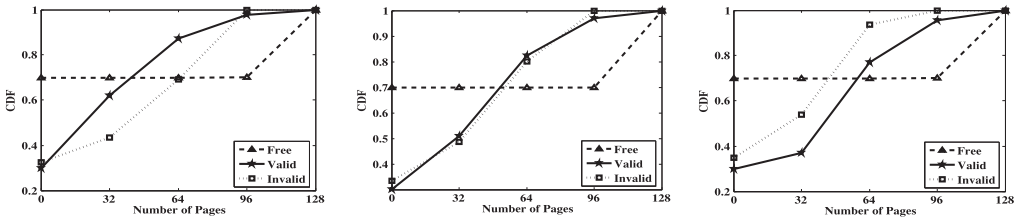


Fig. 1. Cumulative distribution function (i.e., CDF) of blocks with the number of valid, invalid, and free pages based on three sizes of log space under Financial1: (a) 10% log space; (b) 20% log space; (c) 30% log space.

random-write-intensive Finacial1 workload.[1] We observe from Figure 1 that there are an excessive number of free pages in blocks of the SSD. This observation motivates us to propose DLSpace, which aims to manage log space at the block and page levels to fully utilize free pages to optimize garbage collection while enhancing SSD lifetime under the random-write-intensive workloads; and (3) the increased number of garbage-collection operations raise masses of page-rewrite and erase operations. This feature has been illustrated in our prior work (Sun et al. 2015). These drawbacks of TLSpace will impact on NAND Flash write endurance and SSD lifetime. Therefore, we propose a novel log space management called DLSpace, which aims to handle garbage collection and improve SSD lifetime by making full use of log space and unused data space inside SSDs under the random-write-intensive workloads. It should be noted that *merge operation* is considered to be a process that is used in block-level and page-level mapping schemes during garbage collection and not only limited to the hybrid-level mapping policies.

In this article, we propose an efficient distributed log space allocation strategy, referred to as DLSpace, which divides log space into two levels, namely, page-level (i.e., log page space) and block-level (i.e., log block space). Our DLSpace's log page space, which is composed of a portion of data pages in its block, only performs as a buffer for updated pages in this block. Specifically, log page space buffers the page-update operations in the block, thereby delaying the use of log blocks to postpone garbage collection. DLSpace fully utilizes pages in data and log blocks to avoid erasing blocks with free pages. In addition, DLSpace improves write amplification by reducing excessive valid page rewrite and block-erase operations under random-write-intensive workloads. Three main contributions are made in this article:

(1) We design DLSpace, an efficient distributed log space allocation strategy, to improve page program and block erase operations, thereby prolonging SSD lifetime. The management granularity—including block-level and page-level ones—differs from log-space management that is implemented at the block level. DLSpace makes full use of unused pages in flash memory to enhance SSD performance and lifetime during the course of garbage collection. We employ a page-mapping-based flash translation layer to exemplify DLSpace in Section 3.
(2) We quantitatively validate that DLSpace outperforms TLSpace in terms of three metrics, namely, write amplification, delay time before the first garbage collection, and the number of block-erase operations for an SSD. Thanks to the block- and page-level management in

[1]We modify the state-of-the-art SSD simulator, SSDsim (Hu et al. 2011), which is configured using the parameters listed in Table 2. Focusing on the Financial1 workload, we investigate (1) the number of free pages in blocks in flash memory of an SSD and (2) the degree of write amplification improvement under three log-space sizes. We measure three log-space configurations (over-provisioning), namely, 10%, 20%, and 30%.

log space, the DLSpace-based SSD has smaller write amplification than that in TLSpace given the same log space size. Suppressing write amplification increases delay time before the first garbage collection and reduces the number of block-erase operations. Consequently, DLSpace-based SSDs enjoy longer lifetime than TLSpace-based counterparts (see Section 4).

(3) To evaluate our DLSpace, we implement a prototype using real-world SSDs where the TLSpace and DLSpace strategies are deployed. This prototype, which minimizes the impact of buffer and host-side software stacks, facilitates the performance measurements of DLSpace under workloads. The experimental results indicate that DLSpace outperforms TLSpace in terms of all the three metrics (see Section 5).

To summarize, the DLSpace-based log space management is originally proposed in this manuscript. As mentioned above, we observe this phenomenon that there are many free pages in data blocks inside SSD. Then, we propose DLSpace to make full use of these unused and available free pages in data blocks. These pages, which are referred to as page-level log space, distribute in all the data blocks. These page-level log pages locally accept intensive-update I/Os in the data block under a heavy workload. Consequently, DLSpace improves random write performance and reduces the copy-rewrite operations during the garbage collection. The utilization of blocks in log space is postponed, which cuts down the block-erase operations, and decreases write amplification under workloads.

The rest of this article is organized as follows: Section 2 discusses the background and related work. Section 3 presents the detail of DLSpace. TLSpace and DLSpace are quantified in Section 4. We validate the superiority of DLSpace in Section 5. Evaluation environment and experiments based on DLSpace and TLSpace are revealed in Section 6. Section 7 demonstrates further discussion on our work. Conclusion is given in Section 8.

## 2 BACKGROUND AND RELATED WORK

### 2.1 NAND Flash and SSD Basics

NAND Flash (Chen et al. 2009; Gal and Toledo 2005) has three types of operations, namely, read, program (or write), and erase. A page is an atomic unit for read and program, whereas a block is a basic unit for erase operation. A block consists of a fixed number of pages. In NAND Flash, a floating gate transistor (i.e., FGT or cell) is the unit for data bits (Lee et al. 2002).

An FGT consists of a substrate, a floating gate, inter-poly insulation oxide layer, and tunnel insulation oxide layer. The Fowler-Nordheim (i.e., FN) tunneling mechanism for the program and erase operations traps electrons in the tunnel insulation oxide layer (Lenzlinger and Snow 1969). Extra trapping elections make this layer conductive and disable to store data in the cell, thereby resulting in a limited number of P/Es in NAND Flash. NAND Flash memory is classified into three groups, namely, SLC, MLC, and TLC NAND Flash, which have different P/Es (e.g., for 2xnm NAND Flash, SLC: 100k, MLC: 5k–10k, TLC: 0.5k). As the process of NAND Flash scales down, the number of P/Es will decrease. Therefore, excessive program and erase operations can degrade P/Es. Currently, researchers aim to design optimization techniques to reduce excessive operations in NAND Flash space and make full use of available P/Es to improve NAND Flash write endurance and SSD lifetime.

### 2.2 P/Es Optimization

For the past decade, following techniques have mostly been designed to reduce excessive page program and erase operations to perform the P/Es optimization in NAND Flash of an SSD. Techniques aiming to reduce data volume written to an SSD include data de-duplication (Chen et al. 2011; Kim

et al. 2012), data compression (Lee et al. 2011; Wu and He 2012), and performance throttling (Lee et al. 2012).

An efficient flash translation layer (i.e., FTL) (Gupta et al. 2009; Lu et al. 2013; Wei et al. 2011) is applied to make full use of available P/Es in NAND Flash. Popular FTL schemes include mapping policies (e.g., page-level, block-level, and hybrid-level ones), garbage collection (i.e., GC) (Guo et al. 2017; Ha and Kim 2011; Ilias 2010; Park et al. 2016), and wear-leveling (i.e., WL) (D. Jung et al. 2007; Jimenez et al. 2014). GC operation reclaims invalid space in NAND Flash but triggers out-of-place update and copy-rewrite operations. WL makes data distribution across NAND Flash space as even as possible to maximize utilization of P/Es.

Some strategies merge small random writes to large ones using buffer managements (Hu et al. 2010; Huang and Chang 2016; Kim and Ahn 2008; Soundararajan et al. 2010; Wu et al. 2012; Yang and Ren 2011) and garbage collection-aware buffer managements (Hu et al. 2012; Wu et al. 2017) thereby reducing the number of program operations and slowing down the cost of P/Es in NAND Flash.

Many effective solutions were proposed to reduce excessive program and erase operations and lower write amplification by over-provisioning (i.e., OP or log space) (Guan et al. 2017; Hu et al. 2009; Smith 2012). OP is the portion of a NAND Flash space reserved for the SSD controller and cannot be inaccessible to users. TRIM command (i.e., a SATA command) (Frankie et al. 2011, 2012) can expand the OP space when the OS or user erases data.

In addition, data identification strategies (Hsieh et al. 2005; Park and Du 2011) of hot/cold data for flash-memory management, ECC technologies (Jagmohan et al. 2010; Moon and Reddy 2012), page or block reuse strategies (Margaglia et al. 2016; Zhou et al. 2017), new file systems (Lee et al. 2015, 2009; Liu et al. 2016; Min et al. 2012; Zhang et al. 2016), I/O schedulers (Ouyang et al. 2011; Park and Shen 2012; Wang and Huang 2013), file system-aware flash translation layer strategy (Chang et al. 2012), new buffer architectures (Liu et al. 2017), and and so on, attempt to improve NAND Flash write endurance and SSD lifetime by minimizing excessive program and erase operations.

### 2.3 Write Amplification

Write amplification (WA) (Guo et al. 2016; Lee et al. 2017), which is a critical factor to limit the lifetime and performance of SSDs, can be measured based on either NAND Flash space or an entire SSD.

WA is quantified as a ratio of the actual number of program operations per user one (Agarwal and Marrow 2010; Hu et al. 2009; Hu and Haas 2010; Wang et al. 2012b; Xiang and M 2012). We assume that $I$ pages in a block (e.g., B0) with ($V$+$I$) valid pages have been updated. The $I$ pages are set to be invalid after the new content of these pages are updated. $V$ pages are rewritten to another block (e.g., B1) when B0 is reclaimed by garbage collection. In other words, ($V$+$I$) pages are rewritten in the process of updating $I$ pages. Then, we express $WA$ as

$$WA = (V + I)/I. \tag{1}$$

With respect to an entire SSD, write amplification is computed as a ratio of data volume written by SSD controller (e.g., *P_Volume_Data*) to data volume written by the host (i.e., *L_Volume_Data*) (Sun et al. 2013). Thus, we calculate $WA$ as

$$WA = P_Volume_Data/L_Volume_Data = (N_{program}C_{page})/L_Volume_Data, \tag{2}$$

where *P_Volume_Data* is a product of the number $N_{program}$ of program operations and page size $C_{page}$. *L_Volume_Data* depends on workload and can be assessed at the user level.

Without data de-duplication or compression used in an SSD, the ideal value of WA is one, which is beneficial to an SSD. Values larger than one mean that more data volume is written in NAND

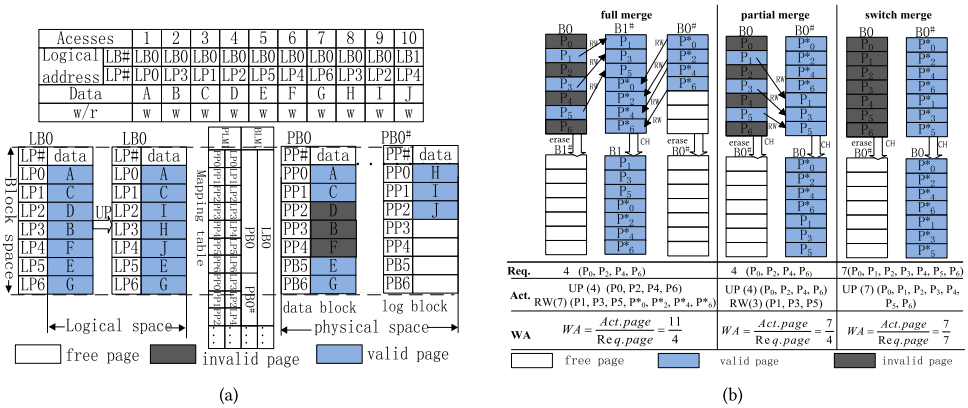


Fig. 2. (a) Traditional log space (block level) in NAND Flash space: LB0: logical data blocks; PB0: physical data block; PB0#: physical log block; UP: page update. (b) Data block and log block merge in TLSpace: B0, B1: data blocks; B0#, B1#: log block; RW: copy rewrite valid page; UP: update page; E: erase block; CH: change the role (e.g., log or data block) of a block; $P_k$,: original valid page; $P^*_k$: updated page; Req.: Required pages written in NAND Flash; Act.: Actual pages written in NAND Flash.

Flash than that issued in workload. Large WA values are harmful to P/Es and adversely affect NAND Flash write endurance and SSD lifetime.

### 2.4 TLSpace Management

NAND Flash space of an SSD is mainly divided into three partitions (i.e., configured space, data space, and log space). The configured space stores parameters and codes of firmware for the SSD; the data space is dedicated to user data; and the log space is reserved for its SSD controller to buffer updated data from the data space to reduce garbage collection frequency. In the TLSpace scheme, the granularity of both data space and log space is a block. According to different mapping schemes, one data block can associate with one or more log blocks and one log block yet associates with one or more data blocks (Kang et al. 2006; Lee et al. 2007).

Figure 2(a) illustrates relationships among logical data blocks, physical data blocks, and physical log blocks in NAND Flash space of a TLSpace-based SSD. A logical data block has the same size as the physical one (e.g., block LB0 and PB0). When pages in a data block (e.g., PB0) are used up, incoming updated pages are rewritten to a log block (e.g., PB0#) associated with PB0. When garbage collection occurs, data blocks are merged with log ones to reclaim invalid pages and generate free blocks. Different merge operations generate a various number of page program and erase operations in NAND Flash space of an SSD.

Three-type merge operations between data and log blocks in TLSpace include full merge, partial merge, and switch merge, respectively. *Full merge* (see Figure 2(b) full merge) copies valid pages from a data block and its associated log block to a new free block and then erases both blocks. Full merge operation triggers extra page rewrites. *Partial merge* (see Figure 2(b) partial merge) copies valid pages in a data block to free ones in a log block, which is then marked as a data one. The number of copy-rewrite operations in partial merge is fewer than that in full merge. *Switch merge* (see Figure 2(b) switch merge) leads to the minimum number of extra program operations. Pages in a data block are all updated to free ones in a log block. There are no valid pages in the data block. The roles of data and log blocks are switched, and the data block is erased. Write amplification is a metric to evaluate the number of extra program operations in the process of merges. For example,

WA values of the full merge, partial merge, and switch merge are 2.75, 1.75, and 1.0, respectively (see Figure 2(b)).

Large write amplification indicates excessive program operations (e.g., full and partial merge in Figure 2(b) based on TLSpace). Data or log blocks are likely to contain free pages while they are erased (see Figure 2(b) full merge). For 3D NAND Flash with large capacity (Lee et al. 2016), it has large-scale log blocks and data blocks in NAND Flash memory. During merge operations, amounts of valid pages in data blocks can incur a large number of rewrite operations, which worsen write amplification and overall system performance. Therefore, P/Es of NAND Flash are cost and SSD lifetime is suffered. To utilize free pages and reduce excessive program and erase operations, we are motivated to propose DLSpace (see details in Section 3), a new allocation strategy for managing log space in large-scale NAND Flash memory such as XLC (Pirovano 2017) or 3D NAND Flash (Kim et al. 2017).

A salient feature of our proposed DLSpace lies in its page- and block-level log management. DLSpace differs from TLSpace in terms of log-space granularity in NAND Flash memory in SSDs. In In Sections 1–3, we present the characteristics of traditional log-space management strategies, highlighting the difference between the conventional schemes and our DLSpace. These existing log-space management techniques have been employed in various flash translation layers or FTLs (see, for example, block-, page-, and hybrid mappings) to improve garbage collection under random-write-intensive workloads. The existing strategies manage block-level log space in NAND Flash (Guan et al. 2017), where the granularity is a block. The space maintaining log and data may be under-utilized in certain cases. To improve the space utilization in SSDs (see Figure 1), we propose DLSpace to manage the log space by seamlessly integrating page- and block-level strategies, in which the granularity of log space is page and block. In these sections, we elaborate the benefits of DLSpace over TLSpace by the virtue of page-level mapping. In Sections 4 and 5, we present the implementation of a real-world SSD platform to evaluate the advantages of DLSpace. Next, we exemplify TLSpace in the platform with page-level mapping, cost–benefit garbage collection, and static wear-leveling. Our DLSpace-based log management strategy can be extended to other SSD devices according to specific firmware structures.

## 3 DLSPACE

This section describes the mechanism and algorithm of DLSpace, which can be applied to SSDs in accordance with page mappings. Let us exemplify the idea of DLSpace through a page mapping scheme.

### 3.1 DLSpace Overview

In a DLSpace-based SSD, logical space is composed of logical blocks (e.g., LB0 and LB1 in Figure 3), each of which has a fixed number of logical pages. A physical space consists of two types of physical blocks, namely, data block (e.g., PB0 and PB1 in Figure 3) and log block (e.g., PB0$^{\#}$ in Figure 3). DLSpace divides pages in a data block into two groups—data pages and log pages (e.g., PL0, PL1 in PB0 in Figure 3). In other words, DLSpace maintains a two-level log space, *block-level log space* and *page-level log space*.

Unlike TLSpace without log pages, DLSpace manages data pages including all pages in a logical block (e.g., page PP0 in the PB0$^{\#}$) and data log pages (e.g., page PL0 in the PB0). Log pages in a physical data block are extra compared with a logical one. But log pages in a physical data block are transparent to user programs when there are available free data pages in the block. When free data pages are exhausted, an updated page in data space is rewritten to a free log page in the block. Note that log pages are only responsible for buffering and handling updated pages in data space of the block. The log pages temperately buffer these updated pages in this data block to (1) delay the

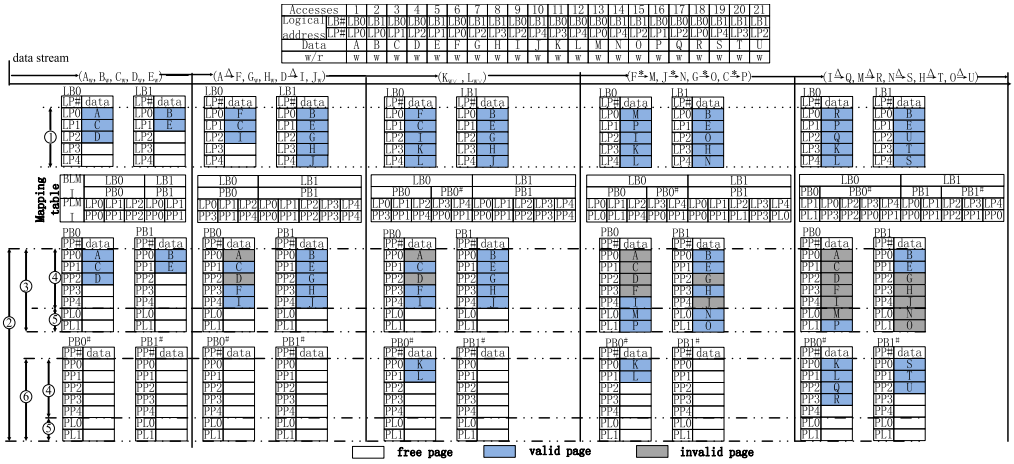


Fig. 3. DLSpace overview and algorithm based on page-level mapping: Six types of space: ① Logical space, ② Physical space, ③ Data block, ④ Data page, ⑤ Log page, ⑥ Log block. Five types of write operations:
(1) Write new data into a free page, e.g., $A_w$, $H_w$, $K_w$, and so on;
(2) Write new data into a free page in log block space, e.g., $K_{w\vee}$ and so on;
(3) Update data into a free page in data page space, e.g., $A \xrightarrow{\Delta} F$, and so on;
(4) Update data into a free page in log page space, e.g., $F \xrightarrow{*} M$, $J \xrightarrow{*} N$, and so on;
(5) Update data into a free page in log block space, e.g., $I \xrightarrow{\Delta} Q$, $N \xrightarrow{\Delta} S$, and so on.

process of selecting a log block from the log block space and (2) to postpone garbage collection. Because a small number of log blocks is more likely to trigger garbage collection. Log pages store updated pages to improve utilization of valid ones in a data block. It avoids erasing blocks with free pages. Moreover, valid pages in data space of a data block are updated to ones in log page space, which is useful to collect invalid pages in given blocks. In the process of garbage collection, excessive valid page rewrites and erase operations are lowered to reduce write amplification even under dominated-random-write workloads. When these log pages are used up, one log block will service incoming requests. The log block can be selected from log space and configured with data page and log page. The updated page is written to a free data page in the log block. In this process, the mapping rule of log pages is identical with that for data ones.

### 3.2 Prototype Implementation

DLSpace is an efficient distributed log space allocation strategy, and its salient feature lies in the page-level log space management. This special log space is composed of a portion of free and unused pages in data blocks. These data pages referred to as the page-level log space buffer the incoming random-updated I/Os in their enclosed blocks.

In DLSpace, the unused free pages in a data block can be fully utilized. Meanwhile, the usage of the block-level log space is postponed, and the process of garbage collection is delayed, which reduces write amplification, and the erase count of a log block; thus, the SSD lifetime can be enhanced as well.

The page-level log management strategy in DLSpace is adopted by using the log pages in a data block. The page-level log space has a fixed number of pages in a data block, for example, eight data pages marking as log pages in a data block (see Section 6.1). If there are free log pages in a data block, then the update pages in the block are first rewritten in the free log pages rather than the

free log block. However, if there is no free log page in the data block, then the update page can be written like that in the conventional page mapping scheduling, e.g., using the free pages in a log block. The reason is that there exists the limited number of log pages in a data block; therefore, the update-intensive workloads will consume these free log pages in the data block. We can statically configure the different log page ratio (i.e., the number of free log pages per block) for different workloads. We present more details of the ratio of log pages in a data block in Section 6.1.

In summary, DLSpace utilizes a portion of log pages in the data block and preserves for these receiving random updated pages in the enclosed block. When the data block including log space is involved in the process of the garbage collection, the dirty data block is reclaimed like that the conventional method in this process. Otherwise, there exists valid data in the log page space; the log page performs the copy-rewrite operation before the data block is erased. To demonstrate the details of the DLSpace, a case of prototype implementation (see Figure 3) is elaborated as follows.

Figure 3 outlines the DLSpace prototype coupled with the page mapping scheme. We incorporate a mapping table including block- (e.g., BLMI) and page-level items (e.g., PLMI) to implement DLSpace, in which two logical blocks (e.g., LB0 and LB1), two physical blocks (e.g., PB0 and PB1), and two log blocks (e.g., $PB0^{\#}$ and $PB1^{\#}$) are considered in this prototype. The number of data and log pages are set to five and two, respectively. In this example, we make one physical block associated with one log one.

In this case, DLSpace processes a special data stream of 21 write I/Os. Given *data stream* ($A_w$, $B_w$, $C_w$, $D_w$, $E_w$), we set the initial number of used pages to three and two in PB0 (i.e., PP0, PP1, PP2 in PB0) and PB1 (i.e., PP0, PP1 in PB1), respectively.

*Data stream_1* ($A \overset{\Delta}{\rightarrow} F$, $G_w$, $H_w$, $D \overset{\Delta}{\rightarrow} I$, $J_w$): By block-level items (i.e., BLMI) in the mapping table, LB0 and LB1 are mapped to PB0 and PB1, respectively. Among page-level items (i.e., PLMI), these relationships of LP2, LP3, and LP4 in LB1 cannot be found, meaning that (LB1, LP2), (LB1, LP3), and (LB1, LP4) are firstly accessed in logical space during $G_w$, $H_w$, and $J_w$. From PLMI of PB1, two used pages and three free pages (i.e., PP2, PP3, and PP4) exist in PB1; thus, LP2, LP3, and LP4 are mapped to PP2, PP3, and PP4, respectively. The data are written in (PB1, LP2), (PB1, LP3), and (PB1, LP4), respectively. The mapping relationships are recorded in PLMI of the mapping table. During *Data stream_1*, data pages in (LB0, LP0) and (LB0, LP2) are updated according to items in the mapping table. In page-level items between LP0 and PB0, there are two free pages; thus, these data pages in (PB0, PP0) and (PB0, PP1) are set invalid. New data in (LB0, LP0) and (LB0, LP2) is rewritten to (PB0, PP3) and (PB0, PP4), respectively. Last, mapping items in PLMI are updated.

*Data stream_2* ($K_{w\vee}$, $L_{w\vee}$): New data in (LB0, LP3) and (LB0, LP4) should be stored in free pages in PB0 according to items in the mapping table. Although there are two free pages in PB0, these free pages hold updated data from data pages in the same block. Therefore, new data are written to free pages in a data or a log block associated with PB0 (e.g., $PB0^{\#}$). Hypothetically, data in (LB0, LP3) and (LB0, LP4) are written in ($PB0^{\#}$, PP0) and ($PB0^{\#}$, PP1), respectively. Mapping items (e.g., (LB0, LP3) to ($PB0^{\#}$, PP0) and (LB0, LP4) to ($PB0^{\#}$, PP1)) are recorded in the mapping table. Note that during $K_{w\vee}$ and $L_{w\vee}$, two free pages exist in log pages; but these pages are used for updated data from data pages of PB0. The new data must be written to free page in another block (e.g., block $PB0^{\#}$).

*Data stream_3* ($F \overset{*}{\rightarrow} M$, $J \overset{*}{\rightarrow} N$, $G \overset{*}{\rightarrow} O$, $C \overset{*}{\rightarrow} P$): Data in (LB0, LP0), (LB1, LP4), (LB1, LP2), and (LB0, LP1) are replaced by new data in logical space, respectively. Using items in the mapping table, data in (PB0, PP0), (PB1, PP4), (PB1, PP2), and (PB0, PP1) should be updated. The updated pages in PB0 or PB1 are respectively rewritten to log pages in its enclosing block when there are free pages in log space of one physical data block. Data in (PB0, PP0), (PB0, PP1), (PB1, PP2), and (PB1, PP4) are set invalid, and new data are rewritten (updated) in (LB0, PL0), (LB1, PL0), (LB1, PL1), and (PB0,

PL0), respectively. These items in PLMI are updated and recorded. In this process, the updated data pages are rewritten in log pages in the same block.

Note that when the user I/Os (i.e., F$\overset{*}{\rightarrow}$M, J$\overset{*}{\rightarrow}$N, G$\overset{*}{\rightarrow}$O, C$\overset{*}{\rightarrow}$P) trigger data update, DLSpace can place the updated data in local page-level log space, such as the update page M and P in block PB0 and updated page N and O in block PB1. The usage of block-level log space for receiving the updated pages is postponed; meanwhile, the updated pages are grouped in a block as soon as possible. In a period, the mapping items for updated pages concentrate in a range of address in a fixed number of blocks rather than distribute in the whole flash memory inside the SSD.

*Data stream_4 (I*$\overset{\Delta}{\rightarrow}$*Q, M*$\overset{\Delta}{\rightarrow}$*R, N*$\overset{\Delta}{\rightarrow}$*S,H*$\overset{\Delta}{\rightarrow}$*T, O*$\overset{\Delta}{\rightarrow}$*U)*: In this case, there are no free pages in log space of a physical data block (i.e., PB0 and PB1). Therefore, new updated data is rewritten in a log block. Data in data pages (e.g., (PB0, PP4) and (PB1, PP3)) and log pages (e.g., (PB0, PL0), (PB1, PL0) and (PB1, PL1)) of physical blocks are rewritten to ones associated with log blocks. In the course of this data stream, the updated data in data or log page are rewritten in log blocks in the context of no log pages in data blocks.

### 3.3 Comparison: DLSpace and TLSpace

Comparing DLSpace (see Figure 3) with TLSpace (see Figure 2(a)), the dispersion of page-mapping items in DLSpace is smaller than that in TLSpace. Update pages are partially grouped in local data blocks and a portion of updated pages are rewritten into the log blocks. The process of updating pages in local page-level log space can make the best use of unused pages in the block and centralize the updated pages in a region, which reduces dispersion of page mapping under intensive-update workloads. The cost of mapping of rewrite pages is much lower than that in TLSpace. Then, the number of updated pages become small and write amplification can be improved during garbage collection. In addition, DLSpace delay the execution of garbage collection and decreases the number of the erased blocks, the details of these metrics are demonstrated in Section 5.

To summarize, we draw the following five conclusions. (1) The sizes of a logical block and a physical block (e.g., LP0 and PB0 in Figure 2(a)) are identical in TLSpace; the size of a logical block is smaller than that of a physical one (e.g., LB0 and PB0 in Figure 3) in DLSpace. (2) TLSpace's log space consists of log blocks; DLSpace's log space is composed of log blocks and log pages (see log pages of PB0 in Figure 3. (3) In TLSpace, when data pages in a physical block are used up, log block (see PB0$^{\#}$ in Figure 2(a)) is immediately employed (see updating page P1 in PB0 in Figure 2(a)) when pages are updated; DLSpace's log block space is lazily utilized and log pages are first applied (see PL0 and PL1 in PB0 in Figure 3). (4) Garbage collection occurs when the number of log blocks decreases to a threshold; therefore, garbage collection, page program, and block-erase operations are more likely to be triggered in TLSpace than that in DLSpace. (5) When log page space is exhausted, log-space operations in DLSpace are similar to those in TLSpace.

## 4 IMPACTS OF TLSPACE VS. DLSPACE ON SSD LIFETIME

Now, we quantitatively study the impacts of two log-space management strategies (i.e. TLSpace and DLSpace) in terms of SSD lifetime. To fair comparisons, these two strategies must be verified using the same log space. For comparisons, we keep the log-space size of TLSpace and DLSpace identical. We initially compare DLSpace and TLSpace on the basic of same log space. Please refer to Table 1 for notation and parameters of an SSD. Space in an SSD is divided into data space, log space, and configured space. Data space is available for users. Log space, which is reserved for the SSD controller, cannot be accessed by the operating system. Configured space is employed to store parameters and firmware codes for an SSD.

Table 1. Notations and Description in This Paper

| Notation | Description |
|---|---|
| $C_{SSD}$ | SSD capacity: $C_{SSD_T}$ for TLSpace; $C_{SSD_D}$ for DLSpace |
| $C_{data}$ | Data space in NAND Flash: $C_{data_T}$ in TLSpace; $C_{data_D}$ in DLSpace |
| $C_{cspace}$ | Configured space in an SSD |
| $C_{parm}$ | Space for parameters in an SSD |
| $C_{log}$ | Log space in an SSD |
| $C_{FW}$ | Space for firmware of an SSD |
| $C_{log-block}$ | Block-level log space in whole log space |
| $C_{log-page}$ | Page-level log space in whole log space |
| $N_{block}$, T | The number of blocks in an SSD |
| U | The number of user blocks in an SSD |
| $N_{page}$ | The number of pages in a physical block: $N_{page_data}$ for data pages; $N_{page_log}$ for log pages in the block |
| $C_{block}$ | A block size |
| $C_{page}$ | A page size |
| V | The number of valid pages in a block |
| I | The number of invalid pages in a block: $I_T$ based on TLSpace; $I_D$ based on DLSpace |
| $\rho$ | The ratio of log space to whole space in an SSD |
| $\chi$ | The percentage of block-level log space for TLSpace |
| $\psi$ | The percentage of block-level log space for DLSpace |
| $\omega_k$ | The percentage of page-level log space in the $k^{th}$ data block for DLSpace |
| $v$ | Variable for the number of valid pages in a block |
| $\widetilde{v}$ | Variable for the mean number of valid pages in a block based on DLSpace: $\widetilde{v}_{data_page}$ for data pages; $\widetilde{v}_{log_page}$ for log pages in the block |
| $x$ | The minimal number of valid pages in a block |
| WA | Write amplification for an SSD: $WA_T$ based on TLSpace; $WA_D$ based on DLSpace |
| FPN | The number of program operations when the first garbage collection is triggered: $FPN_T$ for TLSpace; $FPN_D$ for DLSpace |
| R | Reclaimed space in the process of garbage collection |
| ERN | The number of erase operations for an SSD: $ERN_T$ in TLSpace; $ERN_D$ in DLSpace |
| $\alpha$ | The number of pages out of 128 ones as log space in DLSpace |
| $\beta$ | The number of blocks in 128 ones as log space in DLSpace |
| $z$ | The number of blocks per 128 ones as log space in TLSpace |

Log-space utilization is the ratio between log space and NAND Flash space in the SSD. Note that log-space utilization can be configured by SSD designers. DLSpace and TLSpace are different strategies (see also Section 3). We evaluate the log-space utilization in a tested SSD, which incorporates DLSpace and TLSpace, respectively. We analyze impacts of these two strategies on SSD lifetime. Given log-space parameters, we follow **Theorem 4.1** to quantify log-space capacity of a DLSpace- or TLSpace-powered SSD.

Theorem 4.1. *Let $\chi$ be the percentage of log space in the entire space of a TLSpace-based SSD. Let $\psi$ be the percentage of log block space in the entire space of a DLSpace-based SSD. Let $\omega_k$ be the percentage of log page in the $k^{th}$ data block of DLSpace. Let $N_{block}$ be the total number of blocks in*

*the SSD. When*

$$\chi = \psi + \left[ \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k) \right] \Big/ N_{block}, 0 < \chi < 1,$$

*where TLSpace and DLSpace have the same size of log space in the SSD.*

Proof. SSD capacity can be computed as

$$C_{SSD} = C_{data} + C_{log} + C_{cspace}, \tag{3}$$

where $C_{SSD}$ is the capacity of an SSD, $C_{data}$ is data space for users, $C_{log}$ is the log space in the SSD, and $C_{cspace}$ is the configured space for parameters and codes of firmware for the SSD. $C_{cspace}$ is defined as

$$C_{cspace} = C_{parm} + C_{FW}.$$

Since $C_{cspace} \propto C_{log}$, $C_{SSD}$ is given as

$$C_{SSD} = C_{data} + C_{log} + C_{cspace} \approx C_{data} + C_{log}. \tag{4}$$

Log space is comprised of log block and log page in DLSpace-based SSD, we have

$$C_{log} = C_{log-block} + C_{log-page}. \tag{5}$$

For DLSpace, $\psi$ is the percentage of log block in log space. $\omega_k$ is the percentage of log page in the $k^{th}$ data block. Thus, the capacity of log block space and log page space are respectively estimated as

$$C_{log-block} = \psi N_{block} C_{block}$$

and

$$C_{log-page} = \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k N_{page} C_{page}),$$

where $C_{block}$ and $C_{page}$ are the size of a block and a page in NAND Flash of the SSD, respectively. Consequently, it follows from Equation (4) that

$$\begin{aligned} C_{SSD_D} &\approx C_{data_D} + (C_{log-block} + C_{log-page}) = C_{data_D} \\ &+ (\psi N_{block} C_{block}) + \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k N_{page} C_{page}), \end{aligned} \tag{6}$$

where $C_{SSD_D}$ and $C_{data_D}$ are the capacity of an SSD and the user-data space based on DLSpace, respectively. Let $\omega_k = 0$, meaning that there is only log block space in the TLSpace-based SSD. Let $\chi = \psi$, we can estimate the TLSpace-based SSD capacity as

$$C_{SSD_T} \approx C_{data_T} + C_{log-block} = C_{data_T} + \chi N_{block} C_{block}, \tag{7}$$

where $C_{SSD_T}$ is the TLSpace-based SSD capacity, $C_{data_T}$ is the user-data space, and $\chi$ is the ratio of log block space to the entire SSD space.

To fairly compare the impacts of TLSpace and DLSpace strategies on SSD lifetime, we configure the same user space and log space capacity in an SSD powered by these two strategies, respectively. It follows from Equations (6) and (7) that

$$C_{data_T} + \chi N_{block} C_{block} = C_{data_D} + (\psi N_{block} C_{block}) + \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k N_{page} C_{page}),$$

where we have

$$C_{data_T} = C_{data_D}$$

and

$$C_{block} = N_{page} C_{page}.$$

Since there are $N_{page}$ number of pages in a block; and, we can obtain

$$\chi N_{block} C_{block} = (\psi N_{block} C_{block}) + \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k N_{page} C_{page}). \tag{8}$$

We can simplify Equation (8) as

$$\chi = \psi + \left[ \sum_{k=1}^{\lceil (1-\psi) \rceil N_{block}} (\omega_k) \right] \Big/ N_{block},$$

and, thus, **Theorem 4.1** is proven. □

To study impacts of TLSpace and DLSpace on SSD lifetime, we can apply **Theorem 4.1** to configure the log-space capacity of TLSpace and DLSpace.

SSD lifetime is mainly dependent on write endurance of NAND Flash inside an SSD. Write endurance is determined by the available number of P/Es in NAND Flash. DLSpace enhances P/Es utilization while reduces the excessive program and erase operations to improve SSD lifetime. We employ three metrics (i.e., write amplification, delay for garbage collection occurrence, and the number of erase operations) to quantitatively demonstrate the superiority of DLSpace over TLSpace in Section 5.

## 5 THEORETICAL ANALYSIS OF DLSPACE

In this section, we study the advantages of DLSpace over TLSpace concerning SSD lifetime from the three metrics and write amplification, delay time of garbage collection, and the number of block erases. The definition of these three metrics can be presented as:

- *Write amplification* or WA indicates the relationship between the physical number of program pages and the number of user pages under workloads. A small WA value signifies that excessive program operations are performed in NAND Flash.
- *Delay time for garbage collection* is quantified by the number of programmed pages before the first garbage collection (FPN). A large FPN, which means that garbage collection occurs late, reduces not only garbage collection frequency but also increases the number of invalid pages in a reclaimed block. As a result, the number of rewrite operations for valid page becomes small in the process of garbage collection.
- A small *number of erase operations* (ERN) alleviates the P/Es degradation problem in NAND Flash memory.

SSD lifetime largely depends on the cost of program/erase cycles (P/Es) in NAND flash memory; thus, reducing the number of consumed P/Es improves SSD lifetime. The overarching goal of DLSpace is to enlarge SSD lifetime by cutting back the cost of P/Es in NAND Flash memory. Write amplification is measured as a ratio of the physical number of program pages to the number of user pages under the workload. A high write-amplification value implies that an excessive number of P/Es are consumed, leading to degraded SSD lifetime. FPN is the delay time prior to the first occurrence of garbage collection. Delaying garbage collection tends to increase the number of invalid pages written into flash memory, cutting back the number of block-erase operations. As a result, the SSD delivers high performance for a long period of time. ERN denotes the number of erase operations; a large ERN value imposes detrimental impacts on P/Es and lifetime. Thus,

we conclude that small WAs, large FPNs, and small ERNs indicate a small number of program and erase operations, which in turn are of benefit to improve SSD lifetime.

A qualitative comparison presented in Section 3 shows that, compared with TLSpace, DLSpace has smaller WAs, larger FPNs, and smaller ERNs thanks to DLSpace's promising log-space structure. We quantitatively evaluate improvements of DLSpace over TLSpace using these three metrics. Initially, we prove in Theorem 5.1 that write amplification of DLSpace is lower than that of TLSpace.

Theorem 5.1. *Let $WA_D$ be the value of write amplification based on DLSpace for an SSD under workload with written I/Os. Let $WA_T$ be write amplification value of a TLSpace-based SSD under the same workload. When the log space of the DLSpace-based and TLSpace-based SSDs are identical, the following inequality holds*

$$WA_D \leq WA_T.$$

Proof. During garbage collection, a reclaimed block includes valid pages and invalid pages. Let $N_{page}$ be the number of pages in a block. Let $V$ and $I$ be the number of valid pages and invalid pages in this block, we have

$$N_{page} = V + I.$$

Therefore, we rewrite Equation (1) as

$$WA = (V + I)/I = (N_{page})/(N_{page} - V). \tag{9}$$

To express write amplification of TLSpace-based and DLSpace-based SSDs, respectively, we first prove two auxiliary lemmas.

Lemma 5.2. *Since $\chi$ is the ratio of log space to the total capacity of a TLSpace-based SSD. Then, the following expression for write amplification of the SSD holds*

$$WA_T = 1/2\chi. \tag{10}$$

Proof. In this article, we apply the log block space-based write amplification formula (see expression below) proposed by R. Agarwal (Agarwal and Marrow 2010),

$$WA_T = (1 + \rho)/2\rho,$$

where $\rho$ is the ratio of log space (or over-provisioning) to user space. Ratio $\rho$ is given as

$$\rho = (T - U)/U,$$

where $T$ is the total number of blocks in an SSD and $U$ is the number of user blocks. From Equation (7), $T$ and $U$ are defined as

$$T = N_{block}$$

and

$$U = (1 - \chi)N_{block},$$

where $N_{block}$ is the total number of blocks in the SSD. $\chi$ is the ratio of log blocks to the total blocks based on TLSpace in the SSD. Hence, we have

$$\begin{aligned} WA_T &= (1 + \rho)/2\rho = [1 + (T - U)/U]/2[(T - U)/U] = T/(T - U) \\ &= N_{block}/2[N_{block} - (1 - \chi)N_{block}] = 1/2\chi, \end{aligned}$$

which is the expression of write amplification for a TLSpace-based SSD and, thus, **Lemma 5.2** is proven. □

In TLSpace, the log space is composed of blocks; the log space of a DLSpace-based SSD contains both log block space and log page space.

Lemma 5.3. *Since $\psi$ is the ratio of log block space to the total SSD capacity. $w_k$ is the ratio of log page space to the capacity of the kth physical user block. For all blocks, let the ratio of log page space in every block satisfies*

$$1 > w_1 = w_2 = \cdots = w_k = \cdots = w_{\lceil (1-x)N_{block} \rceil} = \gamma \geq 0,$$

*for simplicity. Thus, the following holds:*

$$WA_D = 1/\{2[\psi + (1-\psi)\gamma] + \gamma[2(1-\psi)(1+\gamma) - 1]\}. \tag{11}$$

Proof. For a user block in the DLSpace-based SSD, the number of pages is given as

$$N_{page} = N_{page_data} + N_{page_log},$$

where $N_{page_data}$ is the total number of pages in a block, $N_{page_data}$ is the number of user pages, $N_{page_log}$ is the number of log pages in the block. Thus, we establish that

$$N_{page_data} = (1-\gamma)N_{page}$$

and

$$N_{page_log} = \gamma N_{page}.$$

Addresses of user data written to NAND Flash space produce a uniform distribution and distribute I.I.D. The uniform distribution is specified as

$$Unif[0, (1-\psi)(1-\gamma)N_{block}N_{page} - 1].$$

According to the random uniform of user write I/Os, the distribution of valid pages (i.e., $f^{(0)}(v)$) in the block reclaimed by garbage collection follows the binomial distribution

$$f^{(0)}(v) = \binom{N_{page}}{v} p^v (1-p)^{N_{page}-v}, \tag{12}$$

where $v$ is the number of valid pages in a block and $p$ is the probability of valid pages in the block. Probability $p$ is expressed as

$$p = (1-\psi)(1-\gamma).$$

Therefore, the mean number of valid pages ($\widetilde{v}$ in a block is given as

$$\widetilde{v} = pN_{page} = (1-\psi)(1-\gamma)N_{page}. \tag{13}$$

Meanwhile, write amplification mainly dependents on the least number of valid pages in a block reclaimed by garbage collection. According to R. Agarwal's work (Agarwal and Marrow 2010), the distribution of valid pages in data space of a block (i.e., $v_{data_page}$ based on DLSpace) converges to a steady-state uniform distribution, which is estimated as

$$v_{data_page} \sim Unif[x, (1-\gamma)N_{page}],$$

where $x$ is the minimal number of valid pages in the block and we obtain

$$x = \{minv : f(N_{page_v} = v) > 0\} \; and \; x \in (0, (1-\gamma)N_{page}).$$

Therefore, we have $(1-\gamma)N_{page} \geq 0$.

In uniform distribution, the mathematical expectation of valid pages in the data space of a block is given as

$$\widetilde{v}_{data_page} = \mathbb{E}(v_{data_page}) = [(1-\gamma)N_{page} + x]/2, \tag{14}$$

where the $\widetilde{v}_{data_page}$ is the mean number of valid pages in a block of a DLSpace-based SSD.

There can be valid pages in log page space in a block. We establish distribution of valid pages in log page space; the mean number of valid pages, $\widetilde{v}_{log_page}$, is computed below,

$$\widetilde{v}_{log_page} = pN_{page_log} = (1-\psi)(1-\gamma)\gamma N_{page}. \tag{15}$$

From Equations (14) and (15), we derive the mean number $\widetilde{v}$ of valid pages in the entire block as

$$\widetilde{v} = \widetilde{v}_{data_page} + \widetilde{v}_{log_page} = [(1-\gamma)N_{page} + x]/2 + (1-\psi)(1-\gamma)\gamma N_{page}. \tag{16}$$

It follows from Equations (13) and (16) that

$$\widetilde{v} = (1-\psi)(1-\gamma)N_{page} = [(1-\gamma)N_{page} + x]/2 + (1-\psi)(1-\gamma)\gamma N_{page}. \tag{17}$$

Then, we can obtain

$$x = (1-\gamma)N_{page}[2(1-\gamma)(1-\psi) - 1]. \tag{18}$$

where $x$ is the minimal number of valid pages in a block and $x > 0$. Write amplification is derived from Equation (1) as

$$WA = (V+I)/I = N_{page}/(N_{page} - x).$$

Therefore, write amplification for the DLSpace-based SSD is estimated as

$$\begin{aligned} WA_D &= N_{page}/\{N_{page} - (1-\gamma)N_{page}[2(1-\gamma)(1-\psi) - 1]\} \\ &= 1/\{2[\psi + (1-\psi)\gamma] + \gamma[2(1-\psi)(1+\gamma) - 1]\}. \end{aligned}$$

Thus, the proof of **Lemma 5.3** is completed. □

The log space capacity of DLSpace equals to that of TLSpace. Using **Lemma 5.3**, the expression in **Theorem 4.1** is rewritten as

$$\chi = \psi + \gamma(1-\psi).$$

Applying **Lemma 5.2**, write amplification for the TLSpace-based SSD can be rewritten as

$$WA_T = 1/2\chi = 1/\{2[\psi + \gamma(1-\psi)]\}. \tag{19}$$

From **Lemma 5.3**, we have

$$WA_D = 1/\{2[\psi + (1-\psi)\gamma] + \gamma[2(1-\psi)(1+\gamma) - 1]\}.$$

In Equation (18), it holds that

$$x = (1-\gamma)N_{page}[2(1-\gamma)(1-\psi) - 1] > 0.$$

From $(1-\gamma)N_{page} \geq 0$, we have

$$2(1-\gamma)(1-\psi) - 1 > 0.$$

From **Lemma 5.3**, $\gamma \geq 0$ and we can obtain that

$$2[\psi + (1-\psi)\gamma] + \gamma[2(1-\psi)(1+\gamma) - 1] \geq 2[\psi + (1-\psi)\gamma]. \tag{20}$$

We conclude from Equations (11), (19), and (20) that

$$WA_D \leq WA_T,$$

and, hence, **Theorem 5.1** is proven. □

Consequently, we prove the superiority of DLSpace over TLSpace in term of SSD write amplification. **Theorem 5.4** afterward proves that DLSpace has a larger FPN value than TLSpace.

Theorem 5.4. *Let $FPN_D$ be the number of page program operations when the first garbage collection occurs for a DLSpace-based SSD under workload. Let $FPN_T$ be the number of page program operations when the first garbage collection occurs for the SSD based on TLSpace under the same workload. As the log-space capacity of DLSpace and TLSpace are identical, the following holds:*

$$FPN_D \geq FPN_T.$$

Proof. We should prove that the larger the average number of invalid pages in a block reclaimed by garbage collection, the bigger value of the average number of data pages written in the block. Using Equation (1), we can obtain

$$WA_D = (V + I)/I = N_{page}/I_D$$

and

$$WA_T = (V + I)/I = N_{page}/I_T,$$

where $I_D$ and $I_T$ is the average number of invalid pages in a block in the DLSpace and TLSpace cases, respectively. It follows that $WA_D \leq WA_T$ (see **Theorem 5.1**); thus, we obtain

$$I_D \geq I_T,$$

which implies that a block of DLSpace has more invalid pages than that of TLSpace. We can draw the conclusion that the average number of page program operations executed before the first garbage collection in DLSpace is larger than that in TLSpace under the same workload. Hence, we have

$$FPN_D \geq FPN_T.$$

This completes the proof of **Theorem 5.4**. □

**Theorem 5.5** demonstrates that DLSpace-based SSDs hold smaller ERNs than those of TLS-based SSDs with the same log-space capacity under the same workload.

Theorem 5.5. *Let $ERN_D$ be the number of erase operations in DLSpace-based SSD under a workload. Let $ERN_T$ be the number of block-erase operations in a TLSpace-based SSD under the same workload. Then, when the log-space size of the DLSpace-based and TLSpace-based are identical, the following inequality holds:*

$$ERN_D \leq ERN_T.$$

Proof. Let

$$R = I \times (ERN) \times C_{page},$$

where $R$ is the reclaimed space during garbage collection. Let $ERN$ be the average number of reclaimed blocks, $I$ is the number of invalid program operations when the first garbage collection begins, and $C_{page}$ is the page size. Assuming that the reclaimed space is the same under workload in DLSpace and TLSpace, we have

$$R_D = I_D \times (ERN_D) \times C_{page} \text{ and } R_T = I_T \times (ERN_T) \times C_{page},$$

where $R_D = R_T$; therefore, the number of blocks being collected in DLSpace-based and TLSpace-based SSDs are expressed as

$$ERN_D = R_D/(I_D \times C_{page})$$

and

$$ERN_T = R_T/(I_T \times C_{page}).$$

From **Theorem 5.4**, it follows that $I_D \geq I_T$; thus, we obtain

$$ERN_D \leq ERN_T.$$

This completes the proof of **Theorem 5.5**. □

## 6 EXPERIMENT AND EVALUATION

In the previous section, we qualitatively prove that DLSpace improves TLSpace in terms of WA, the number of page program operations when the FPN is triggered and the number of block-erase operations under workload (i.e., ERN). Now we implement DLSpace and TLSpace in a real-world SSD and quantitatively compare the WA, FPN, and ERN performance of the two log-space strategies.

Table 2. Parameters Employed in Tested SSD

| Tested SSD | | NAND Flash | |
|---|---|---|---|
| Capacity | 8GB | NAND Flash type | 34nm SLC |
| Data space | 5GB | NAND Flash size | 2GB |
| Log space | 3GB | Page size | 4KB |
| DRAM | NO | Pages per block | 128 |
| Data cache | 32KB | | |

### 6.1 Evaluation Setup and Workloads

Testbed is composed of a host hardware platform and a real-world SSD (i.e., tested SSD). Hardware platform in host is a server with Intel Pentium (R) Dual-Core 3.20GHz CPU, 1GB memory, and 1TB WD HDD. Ubuntu 11.10 is running on the server. We implement both DLSpace and TLSpace in the real-world SSD to evaluate impacts of two-type log-space strategies on SSD lifetime. It is worth noting that the TLSpace-based log space is composed of blocks. Log blocks in the pool of log space are determined by page-level mapping employed in a tested platform. The DLSpace-based log space is made up of block-level log blocks and page-level log pages distributed in the pool of log space and data blocks. We aim to evaluate impacts of TLSpace and DLSpace on *WA*, *FPN*, and *ERN*, which quantify the real-world SSD lifetime. The page-level mapping, cost–benefit garbage collection, and static wear-leveling are applied in tested SSD, parameters of which are listed in Table 2.

We employ four full-parallelism levels (i.e. channel-level, chip-level, die-level, and plane-level) in our tested SSD. SSDs with the non-full-parallelism mode are unpopular in the IT industry and; therefore, we overlook the modeling issue of the non-full-parallelism-based SSDs. Nevertheless, our DLSpace is readily to be extended to manage log space of SSDs with the non-full-parallelism mode. We envision our full-parallelism model as a case study demonstrating how to apply the DLSpace to effectively manage log space inside SSDs. In case an SSD's parallelism is different (e.g., non-parallelism mode) from that in our manuscript, it is straightforward to follow the DLSpace approach to develop a new log management strategy for the new SSD parallelism mode. It is arguably true that the intrinsic idea of DLSpace is applicable to any SSD regardless of parallelism modes.

DLSpace—a log-space management strategy—is deployed as a plugin component in flash memory inside SSDs. DLSpace can be implemented in SSDs with various capacities like 8GB, 128GB, and 1TxB. Collaborating with the SoliWare Corporation, we built a cost-effective 8GB-SSD testbed. In this Section, we observe noticeable benefits of DLSpace in the 8GB-tested SSD regarding three performance metrics, i.e. WA, FPN, and ERN.

It is noteworthy that large-capacity SSDs (e.g., 128GB and 1TB) readily perform DLSpace in the testbed. To validate the scalability of DLSpace, we extend the experiment on the 128GB-based SSD platform and evaluate impacts of TLSpace and DLSpace on WA, FPN, and ERN, which quantify the real-world SSD lifetime. Please refer to Appendix A for the more details. We conclude that the advantages of DLSpace do exist in large-capacity SSDs and DLSpace is featured with good scalability.

Five real-world workloads are replayed in our testbed. Table 3 summarizes the average request size, write data volume, and the ratio of write I/Os. Finacial1 and Financial2 (i.e., F1 and F2) are collected from OLTP applications at a financial institution (Bates and McNutt 2007). *Usr* is collected from user home directories. *Proj* is obtained from project directories, and *prn* is collected from print servers. These workloads are available and download from SNIA's website (Microsoft 2007). We

Table 3. Workload Characteristics

| Workload | Avg. req. size Wr/Rd (KB) | Data volume written(GB) | Wr : Rd |
|---|---|---|---|
| Financial1(F1) | 3.8/2.3 | 13.1 | 77 : 23 |
| Financial2(F2) | 3.0/2.3 | 1.8 | 18 : 82 |
| usr | 15.1/52.7 | 55 | 3 : 97 |
| proj | 42.6/18.1 | 140 | 94 : 6 |
| prn | 9.6/22.8 | 45 | 78 : 22 |

deploy IOReplayer, an I/O trace tool and extended from Btreplay (Axboe and Brunelle 2007), to replay all workloads in the testbed. The tested SSD is opened with O_DIRECT and configured a raw block device. The operating system is utilized with CFQ as the I/O scheduler.

**ALGORITHM 1:** Get Values of $\alpha, \beta, and z$ algorithm

**Require:** The number of pages in a block, $N_{page}$;
**Ensure:** The values of $\alpha, \beta, z$;

```
for i ∈ {1, 2, ··· , N_page} do
    for j ∈ {1, 2, ··· , N_page} do
        for k ∈ {1, 2, ··· , N_page} do
            if i == k + j * (1 − k/128) then
                z = i; α = j; β = k
            end if
        end for
    end for
end for
```

Following **Algorithm 1**, multiple sets of $\alpha$, $\beta$, and $z$ and their values can be determined. As **Theorem 1**, we configure the ratio of log-block space and log-page space to set the same log space in tested SSD with two-type strategy. Impacts of DLSpace and TLSpace on the SSD lifetime are evaluated in terms of *WA*, *FPN*, and *ERN*.

In tested SSD, there are 128 pages per block. Therefore, we can obtain

$$N_{page} = 128 \ and \ C_{block} = 128 C_{page}.$$

Let $\omega$ be the same value in every block, and we derive the following statement from **Lemma 5.3**

$$w_1 = w_2 = \cdots = w_k = \cdots = w_{\lceil (1-x)N_{block} \rceil} = \alpha/128,$$
$$\psi = \beta/128, \ and \ \chi = z/128,$$

where $\alpha$ is the number of pages per 128 ones reserved as log pages that compose log page space in DLSpace. Let $\beta$ be the number of blocks per 128 ones reserved as log blocks being as log block space in DLSpace-based SSD. And $z$ is the number of blocks per 128 ones reserved as log space in TLSpace-based SSD. The formula in **Theorem 1** can be rewritten as

$$z/128 = \beta/128 + \left[\lceil (1 - \beta/128) \rceil N_{block} \, (\alpha/128)\right] / N_{block},$$

which can be simplified as

$$z = \beta + \lceil (1 - \beta/128) \rceil \alpha, \tag{21}$$

where $\alpha, \beta, z \in [0, 128]$ and $\alpha, \beta, z \in \mathbb{N}$.

Table 4. Parameters in TLSpace and DLSpace

| Allocation Strategies of Log Space | | Parameters | | |
|---|---|---|---|---|
| | | $\alpha$ | $\beta$ | $z$ |
| TLSpace (T) | | 0 | 53 | 53 |
| DLSpace | DLSpace-1(D-1) | 8 | 48 | 53 |
| | DLSpace-2(D-2) | 28 | 32 | 53 |
| | DLSpace-3(D-3) | 48 | 8 | 53 |

A array of ($\alpha$, $\beta$, and $z$) is defined as a set of parameters. By Equation (21), we can configure amounts of sets, those of which have the same $z$ referred as a series of parameters. The same $z$ means that we can ensure the log space in both DLSpace- and TLSpace-based SSDs is equal. A series with $\alpha = 0$ indicates the log space for TLSpace-based SSD, and others represent log space for DLSpace-based SSD. One series (see Table 4) is selected to study different impacts of TLSpace and DLSpace on *WA*, *FPN*, and *ERN* for tested SSD.

Table 4 lists one case for TLSpace and three cases for DLSpace. We investigate (1) impacts of DLSpace and TLSpace on *WA*, *FPN*, and *ERN* (see DLSpace and TLSpace in Table 4) and (2) impacts of DLSpace on trends of WA, FPN, and ERN (see D-1, D-2, and D-3 in Table 4). The series of $\alpha$, $\beta$, and $z$ are configured in our tested SSD.

We construct a tested platform to qualitatively compare DSLpace and TLSpace under the same hardware configuration. In our prior work, our findings confirmed that large log-space capacities or high over-provisioning rates are likely to lower write amplification. In other words, large-capacity log space improves the WA, FPN, and ERN metrics in both DLSpace- and TLSpace-based log management strategies. To examine the advantage of DLSpace over TLSpace, we configured the same size of log space in DLSpace- and TLSpace-based SSDs.

According to Equation (21), we obtain a set of arrays in the form of ($\alpha$, $\beta$, and $z$) to configure the same log space. Therefore, 37.5% total capacity in the tested SSD is only used to verify our proposed DLSpace. The default configuration with 7.0% log space is also used in the DLSpace-based SSDs.

### 6.2 Experiment Methodology

IOReplayer replays two groups of traces: (1) the quantity of written data in a workload (i.e., F2) is smaller than the user capacity (i.e., 5GB) in the tested SSD and (2) workload (i.e., F1, usr, proj, and prn) has a larger amount of write data than user capacity in the tested SSD.

We modified the firmware of the tested SSD to read its metrics regarding the number of program operations, FPN, and ERN. When IOReplayer completes a trace, the WA, FPN, and ERN values are collected from the tested SSD and recorded to a log file in 10 minutes. Because program and erase operations could be running with the execution of demonic garbage collection. WA value can be obtained based on the number of program operations and user-written data. Secure erase command is employed to clean space in the tested SSD before every test to put it into the same initial state.

In the tested SSD, 32KB space in SRAM rather than DRAM is employed as a buffer to eliminate influence from cache space on WA, FRN, and ERN. This is because large buffer space can merge smaller write I/Os into larger ones and produce interference to the impacts of DLSpace and TLSpace on WA, FPN, and ERN.

### 6.3 Results Evaluation

*6.3.1 Write Amplification.* Write amplification values (see Figure 4(a)) of the tested SSD incorporating DLSpace (i.e., D-1, D-2, or D-3 as listed in Table 4) are smaller than those of the same

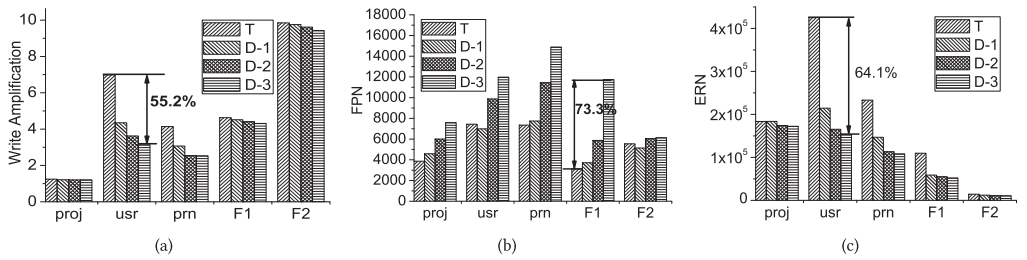


Fig. 4. Three metrics for the tested SSD in TLSpace (T), DLSpace-1 (D-1), DLSpace-2 (D-2), and DLSpace-3 (D-3) cases: (a) WA, (b) FPN, (c) ERN.

SSD employing TLSpace (i.e., T). From Figure 4(a), it is observed that increasing log page space of a data block brings garbage collection frequency down, which in turn reduces write amplification. Thus, the number of program or erase operations decreases, which improves utilization of P/Es and prolongs SSD lifetime.

When it comes to *proj*, write amplification is approximately one thanks to a large number of sequential requests in this workload. As a result, the write-amplification impacts made by both TLSpace and DLSpace are marginal.

For example, results of D-3 reveal that DLSpace reduces write amplification of TLSpace by 4.0%. When we use the *usr* to drive the SSD (e.g., D-3), we observe that DLSpace significantly improves write amplification of TLSpace by 55.2%. DLSpace is superior to TLSpace in D-3 case, because *usr* is dominated by random I/Os. Log pages in the D-3 case can buffer an increased number of updated pages from data space in the block. DLSpace prevents more page rewrite and erase operations in the process of garbage collection and; therefore, write amplification becomes very small.

Under *prn*, write amplification remains unchanged in D-2 and D-3 cases, implying that the log page space set can deal with requests in D-2 case. Write amplification for the tested SSD in D-3 offers 7.0% and 5.0% improvement over the TLSpace counterpart under Financial1 and Financial2, respectively. Due to random requests in this workload, write amplification cannot be reduced and program operations frequently occur, which enlarges P/Es consumption in NAND Flash space and shortens SSD lifetime.

*6.3.2 Number of Programs before First GC.* An FPN value presents the delay time before the first garbage collection is performed in the tested SSD. A large FPN indicates a long garbage collection delay, meaning that the log-space allocation strategy in the tested SSD deals well with write I/Os in workload. Thus, garbage collection frequency and the number of program operations in NAND Flash space decreases, which helps in enlarging the tested SSD lifetime.

Log page space in data blocks of the tested SSD can temporarily accept write I/Os. FPN values (see Figure 4(b)) for the SSD based on DLSpace are larger than those in TLSpace-based SSD in most cases. Under *prn* (78%-write I/Os and the average size of write I/Os is around 9.6KB), FPN of the SSD in D-3 case achieves 51% improvement over that in TLSpace-based SSD. In this case, DLSpace-based SSD outperforms the TLSpace counterpart by a factor of 73.3% in the term of FPN under Financial1 with small requests (e.g., 3.8KB) and 77%-write I/Os. Under other workloads, increasing the number of log pages in data blocks can enlarge the FPN values.

*6.3.3 ERN: Number of Block-erase Operations.* The number of erase operations (i.e., ERN) of the tested SSD mainly depends on random-write data size. Increasing erase operations inevitably decreases the number of available P/Es. It is a wise idea to minimize the number of erase operations to extend SSD lifetime by alleviating the cost of P/Es.

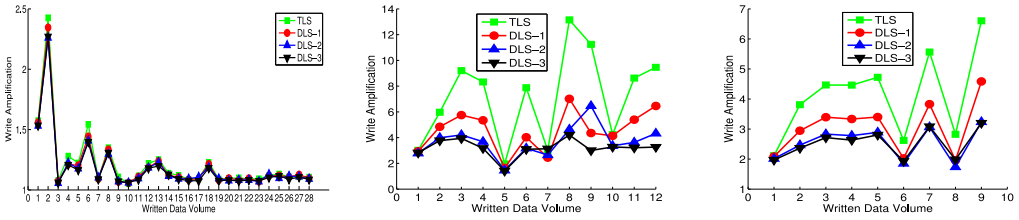


Fig. 5. Write amplification of the tested SSD in T, D-1, D-2, and D-3 cases driven by the three real-world workloads: (a) proj; (b) usr; (c) prn.

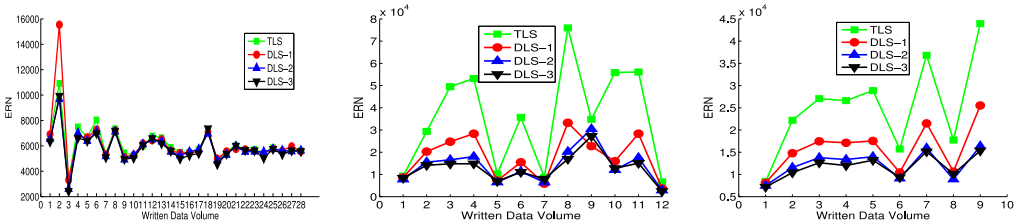


Fig. 6. Number of erase operations (i.e., ERN) of the tested SSD in T, D-1, D-2, and D-3 cases driven by the three real-world workloads: (a) proj; (b) usr; (c) prn.

The ERN value is minimized under Financial2 (see Figure 4(c)), which exhibits 1.8GB written data. ERN drops with the increasing number of log pages in data blocks (for example, T, D-1, D-2, and D-3). The sequence *proj* with 94%-write I/Os (i.e., 140GB write data) makes the TLSpace-based SSD's ERN close to that of the DLSpace-based SSD. Unlike other workloads, the *usr* is comprised of 55GB random-written data that increases the number of erase operations (see Figure 4(c)). The number of log pages in data blocks heavily affects ERN. In the D-3 case, we increase the number of log pages in data blocks to enable the tested SSD to efficiently handle random write I/Os and reduce copy-rewrite and block-erase operations. In this case, both garbage collection frequency and ERN are pushed down. Consequently, DLSpace reduces ERN of TLSpace by up to 64.1% in this case. Under *prn*, log pages in data blocks still properly deal with random I/Os. Meanwhile, ERN decreases when the number of log pages increases.

*6.3.4 Variations of WA and ERN.* Now we study the impact of workloads on the lifetime variations of the tested SSD powered by TLSpace and DLSpace. In this group of experiments, we focus on the impacts of write I/O size on both WA and ERN for the tested SSD.

In this group of experiments, we study the performance trends of WA (see Figure 5) and the ERN (see Figure 6) under write-intensive workloads. To put the performance trends of WA and ERN under a microscope, we conduct a series of tests (a.k.a., test points), each of which is focused on the result of write I/Os processing 5GB data. In doing so, we intuitively demonstrated the performance trends of write amplification and the number of erase operations. We tested five workload conditions: Financial1, Financial2, usr, prn, and proj, the data sizes of which are 13.1GB, 1.8GB, 55GB, 140GB, and 45GB, respectively. Because each test point if focused on write I/Os of 5GB, we evaluated usr, prn, and proj that handle large datasets.

Therefore, we reply the *proj*, *usr*, and *prn* traces characterized with write intensive. We intentionally split a trace into a few smaller ones with 5GB-written data volume. In Figure 5(a), the horizontal axis has a scale in 5GB-written-data sub-trace in *proj*. These sub-traces are sequentially

replayed by IOReplayer. Ten minutes after finishing each sub-trace, WA and ERN values are measured. The total number of sub-traces in the *proj, usr, and prn* traces are 28, 11, and 9, respectively.

In *proj, usr, or prn*, WA and ERN for test-SSD based on TLSpace and DLSpace has the same values in the first sub-test. Thanks to the initial state in the first test, where test-SSD has enough data space and log space for the write date in the trace. However, this experimental results present differences in subsequent cases. Many log pages in log space make the trend of WA or ERN smooth, which means excessive page program and block-erase operations hardly frequently occur. Therefore, utilization of P/Es is improved and lifetime is prolonged for the test-SSD based on DLSpace.

Under the *proj* with sequential write I/Os, the WA and ERN values keep around 1.2 and 6000, respectively (see Figures 5(a) and 6(a)). WA reaches 2.5 when the second 5GB-subtrace is replayed, because write data size of the sub-trace exceeds the SSD capacity and page rewrite and block-erase operations are triggered in garbage collection.

The subsequent WA values present marginal changes. Experiment confirms that WA or ERN of the SSD does not significantly change under sequential-requests-intensive workloads.

Given random-write-intensive workloads (e.g., *usr* and *prn*), DLSpace (e.g., D-1, D-2, and D-3) noticeably reduces WA and ERN of the TLSpace-based SSD (see Figure 5(b) and (c) and Figure 6(b) and (c)). This trend is reasonable, because log page space in data blocks postpones garbage collection by buffering random I/Os. DLSpace reduces page copy-write and block-erase operations to slow down WA. Thus, the WA tendency is much smooth with more log pages configured in log space (Figure 5(b) and (c)). ERN of the tested SSD (see Figure 6(b) and (c)) has the same trend.

### 6.4 Evaluation Summary

We summarize the performance above analysis of DLSpace and TLSpace as follows.

- Under sequential workloads or in an initial state (SNIA 2012) of the tested SSD, DLSpace maintains similar performance as that of TLSpace due to infrequent copy-rewrite and block-erase operations as well as sufficient data space.
- Under random workloads or after the initial state, DLSpace is significantly superior to the conventional TLSpace. DLSpace substantially reduces WA and ERN of TLSpace by 55.2% and 64.1%, respectively. DLSpace increases FPN of TLSpace by up to about 73.3%.
- Compared with TLSpace, our DLSpace exhibits smooth WA and ERN trends. When the number of log pages in DLSpace is large, further increasing the number of log pages only leads to a marginal increment in WA and ERN of SSDs.

Our experimental results confirm that DLSpace not only reduces the number of page program and block-erase operations but also keeps the low frequency of garbage collection. In doing so, DLSpace is conducive to slowing down the P/Es degradation in NAND Flash and prolonging SSD lifetime.

## 7 FURTHER DISCUSSION

In this article, we proposed a log-space distributed allocation strategy called DLSpace to extend SSD lifetime. With the new design of log-page space in place, DLSpace postpones garbage collection, avoids erasing log block with free pages, and lowers write amplification. Thus, DLSpace substantially improves the utilization of log blocks and its mechanism is seamlessly integrated with a page-mapping scheme. Besides, we have a solid plan to integrate DLSpace with block-level or hybrid-level mapping schemes in SSDs in the follow-up work.

In our experiments (see Section 6.1), a handful series of parameters $\alpha$, $\beta$, and $z$ can be obtained. We chose partial series in our test, because all the ones have similar configurations for DLSpace and TLSpace. The experiment aims to study the various impacts of DLSpace and TLSpace on WA, FPN,

and ERN; therefore, one sample series listed in Table 4 is sufficient to demonstrate the strengths of DLSpace. DLSpace's strength cannot be fully achieved under sequence-write-intensive workloads or in the initial state of an SSD. Sequence-write-intensive workloads rarely trigger garbage collection, thereby making excessive program and erase operations hardly occur. In the SSD's initial state, there is sufficiently enough data space and log space to accommodate write requests. We conclude that DLSpace is superior to TLSpace either under random-request-dominated workloads and after the initial state of SSDs.

Buffer management and over-provisioning (a.k.a., log-space management) are two commonly adopted methods to improve garbage collection and SSD lifetime. In this study, we pay attention on the implementation of log-space management DLSpace for SSDs. To evaluate the benefits of DLSpace-based log-space management, we make use of a small portion of buffer inside a tested SSD (see Table 2). Evidence shows that increasing the buffer space can further improve the performance metrics in our experiments. Nevertheless, the impact of buffer management on SSD lifetime is out the scope of this study.

We quantitatively evaluate our DLSpace in a real-world SSD manufactured by the SoliWare Corporation. Due to the cost of constructing a customized SSD, we built an SSD with a small capacity to assess DLSpace-based log-space management in terms of write amplification, garbage collection, and block erase under the five workloads. Nevertheless, we do not imply by any means that DLSpace is limited to small SSDs. It is arguably true that real-world SSD with large data space and log space can also enjoy DLSpace, because the benefits of log space management (e.g., TLSpace and DLSpace) do exist in SSDs regardless of their capacities. But the advantage of DLSpace over TLSpace also shows as long as the same size of log space configured inside SSDs. To make a fair comparison, we test the advantage of DLSpace over TLSpace by configuring the same size of log space inside the SSD.

## 8 CONCLUSION AND FUTURE WORK

In an SSD, log space buffers updated pages in data space. The existing log-space allocation strategies like TLSpace are designed for blocks, where free pages cannot be fully utilized during the course of page update operations. Page rewrite and erase operations trigger garbage collections and enlarge write amplification under random-request-dominated workloads, thereby degrading the available P/Es in NAND Flash and shortening SSD lifetime.

To address the aforementioned problems, we proposed an efficient distributed log space allocation strategy, referred as DLSpace, to extend SSD lifetime. Unlike TLSpace, DLSpace divides log space into two levels, i.e., log block and log page space. Log page space is composed of partial pages in a data block. It stores updated pages from data space to postpone garbage collection and utilizes free pages in the block. In other words, DLSpace gives a high priority to log-page space by utilizing log pages before any log block is used. Compared with the existing TLSpace, DLSpace is conducive to (1) preventing excessive valid page rewrite and block-erase operations thanks to delayed garbage collection, (2) reducing the number of erase operations, and (3) lowering write amplification. Then, DLSpace improves P/E utilization in NAND Flash while extending SSD lifetime.

Theoretical analysis is proposed to show that DLSpace outperforms TLSpace in terms of WA, FPN, and ERN. Experiments confirm that DLSpace reduces write amplification and the number of erase operations of by 55.2% and 64.1%, respectively compared with TLSpace. It delays TLSpace's garbage-collection starting time by 73.3% to the most extent.

We expect that a large buffer (e.g., DRAM on board) will lead to a significant improvement in terms of WA, the number of page program operations before the first garbage collection (i.e., FRN), and the ERN. In a future study, we plan to incorporate buffer management strategies with a large capacity into SSDs to investigate the impact of buffer management on WA, FRN, and ERN of SSDs.

# APPENDIX

## A DLSPACE SCALABILITY

In Section 6, we quantitatively evaluate that DLSpace improves TLSpace in terms of WA, the number of page program operations when the FPN is triggered and the ERN under five types of workloads. To validate the scalability of DLSpace, we implement DLSpace on a large-capacity tested SSD. First, the scalability of DLSpace is defined as how the SSD lifetime varies with the capacity size of the tested SSD for the same DLSpace-based log space allocation strategy. Note that we study the SSD lifetime impacted by DLSpace and TLSpace from the three metrics, WA, FPN, and ERN.

We conduct extended experiments concerning the above three metrics. In this section, we present the experimental setup and workloads for the scalability evaluation in Section A.1 Experimental methodology in Section A.2 adopts the same one in Section 6.2. We analyze the results of scalability evaluation in Section A.3. Section A.4 summarizes our extended experiments for the scalability of DLSpace in a large-capacity tested SSD.

### A.1 Experimental Setup and Workload

The testbed is equipped with the same overview of hardware and software (see Section 6.1) in the scalability evaluation. The new tested SSD is a 128GB-based real-world SSD. We implement both DLSpace and TLSpace in a large-capacity SSD to evaluate DLSpace scalability regarding WA, FPN, and ERN. These metrics based on these two log-space strategies affect the large-capacity SSD lifetime. The page-level mapping, cost–benefit garbage collection, and static wear-leveling are also applied to the new SSD, see its parameters in Table 5. Note that four full-parallelism levels (i.e., channel-level, chip-level, die-level, and plane-level) deploy deployed in the new tested SSD as well.

In Table 3, five real-world workloads (i.e. *Financial1 (F1), Finacial2 (F2), usr, prn, and proj*) are replayed on our new testbed. IOReplayer is used to replay these workloads on the 128GB-based SSD. The new large-capacity SSD is opened with *O_DIRECT* and configured a raw block device.

According to Equation (21) and Algorithm 1, we use the new configuration parameters in Table 5 for the TLSpace- and DLSpace-based SSD. From the experimental results, WA, FPN, ERN, and the changes of these metrics present the following features. (1) DLSpace is significantly superior to the conventional TLSpace. (2) The optimization of these metrics can be enhanced when the number of log pages in DLSpace increases. For example, the maximum number of log pages in the DLSpace-3 (i.e. D-3) case can buffer more updated pages from data space in the block than that in T, D-1, and D-2. The tested SSD incorporating D-3 has better metrics of SSD lifetime. Refer to Section 6.3 for details about impacts of T, D-1, D-2, and D-3 on WA, ERN, and FPN. Without loss of generality, we select the log-space parameters (TLSpace, i.e. T) and DLSpace-1 (i.e. D-1) in Table 4 to study various impacts of TLSpace and DLSpace on WA, FPN, and ERN on the 128GB-based large-capacity SSD.

Table 5. Configuration Parameters of the Tested SSD for DLSpace Scalability Evaluation

| Tested SSD | | NAND Flash | |
|---|---|---|---|
| Capacity | 128GB | NAND Flash type | 25nm MLC |
| Data space | 75GB | NAND Flash size | 16GB |
| Log space | 53GB | Page size | 8KB |
| DRAM | NO | Pages per block | 256 |
| Data cache | 32KB | | |

Table 6. Parameters of TLSpace and DLSpace for DLSpace Scalability Evaluation

| Allocation Strategies of Log Space | | Parameters | | |
|---|---|---|---|---|
| | | $\alpha$ | $\beta$ | $z$ |
| TLSpace | TLSpace-I($T_I$) | 0 | 53 | 53 |
| | TLSpace-S($T_S$) | 0 | 53 | 53 |
| DLSpace | DLSpace-1-I($D-1_I$) | 8 | 48 | 53 |
| | DLSpace-1-S($D-1_S$) | 8 | 48 | 53 |

Thus, we find that the advantages of DLSpace over TLSpace on large-capacity tested SSD validate the good scalability of DLSpace.

In our prior work (Sun et al. 2015), we find that the performance of large-capacity SSD is different in the initial state and the steady state. Therefore, we configure the tested SSD with two states (i.e. initial state and steady state) to evaluate DLSpace scalability from the above metrics using the large-capacity SSD. The initial state that is defined as the factory state makes SSD high performance. The steady-state SSD evolves into a steady performance state relative to the workloads. To place the tested SSD in the steady state (steady for short), the 8KB-align and sequential-write access is configured to distribute data across the available space of NAND Flashes as unevenly as possibly for about four hours.

### A.2 Experimental Methodology

The experimental methodology for DLSpace scalability is the one used in Section 6.2. Please refer to Section 6.2 for more details.

### A.3 Results Evaluation

Our experiments aim to present the scalability of the proposed DLSpace, meaning that the tested SSD with DLSpace exhibits its advantages of *WA, FPN, ERN* and the change trend of these metrics whether on the low- or large-capacity tested SSD platform. In Table 6, four cases are employed in our experiments to study the above metrics: (1) TLSpace-based SSD in the initial state (baseline, $T_I$); (2) TLSpace-SSD in the steady state (i.e., $T_S$); (3) DLSpace-1-based SSD in the initial state ($D-1_I$); (4) DLSpace-1-based SSD in the steady state ($D-1_S$).

*A.3.1 Write Amplification.* A large-capacity SSD presents different performance in the initial state and the steady state. We perform two-state-based experiments in the scalability evaluation.

In Figure 7(a), write amplification of the tested SSD incorporating DLSpace (i.e., D-1 in the initial state ($D-1_I$) and D-1 in the steady state ($D-1_S$)) presents the smaller values than those of the same SSD equipped with the TLSpace (i.e., TI in the initial state and Ts in the steady state). Experimental results validate the scalability of DLSpace, which results from that the large-capacity DLSpace-based SSD outperforms the TLSpace counterpart in the term of WA under five workloads in two states.

In the initial state, write amplification is very small, because there is a sufficient amount of available space for write I/Os to incur only infrequent garbage collection when data is updated under workloads. The values of $D1_I$ plotted in Figure 7(a) are very low than that of $T_I$ (see *usr* for example). The result is attributed to the fact that the tested SSD takes advantage of DLSpace-based log space management strategy. To place the tested SSDs in the steady state, the values of the tested SSD are higher than that in the initial steady. The reason for this trend is that the lack of sufficient data space for written data can aggravate write amplification. In this case, the capacity of log

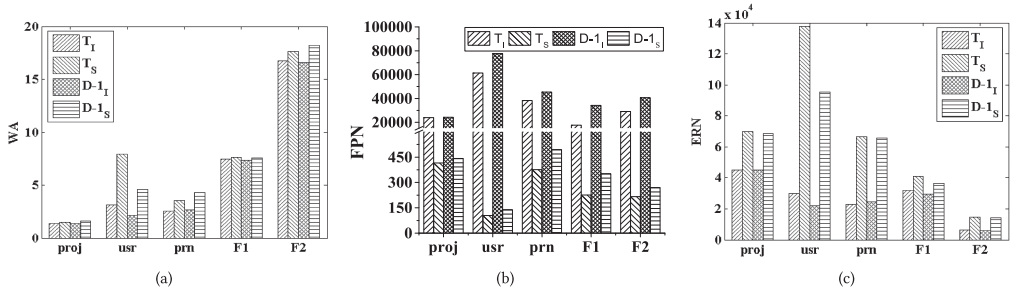


Fig. 7. Three metrics of large-capacity tested SSD for scalability evaluation in TLSpace-I ($T_I$), TLSpace-S ($T_S$), DLSpace-1-I ($D-1_I$), and DLSpace-1-S ($D-I_S$) cases: (a) WA, (b) FPN, and (c) ERN. *T* in the initial state ($T_I$), *T* in the steady state ($T_S$), *D-1* in the initial state ($D-1_I$), and *D-1* in the steady state ($D-1_S$).

space (i.e., the inclusion of extra storage capacity in NAND Flashes that is reserved only for SSD controllers) is an important factor affecting the write amplification for SSDs with small data space. With the same log space size, the DLSpace reduces write amplification (such as the write amplification in $T_s$ and $D-1_S$ under *usr* and *prn*, respectively). DLSpace makes full use of the block-level log space and page-level log page in a data block and brings garbage collection frequency down; thereby reducing write amplification, improving utilization of P/Es and prolonging SSD lifetime.

In Section 6.3.1, we have analyzed the impacts of five workloads on write amplification, which depends on I/O characteristics of workloads. In *proj*, a great many sequential requests make write amplification of approximately one in both TLSpace and DLSpace. The results of D1I and D1S reveal that DLSpace reduces write amplification of TLSpace as well. When the *usr* is loaded on the SSD in steady state (e.g., $T_S$ and $D-1_S$), we observe that DLSpace significantly improves write amplification of TLSpace by 40.8%. In this case, the random-write I/Os in *usr* trigger smaller write amplification in DLSpace than that in TLSpace.

*A.3.2 Number of Programs before First GC.* As mentioned in Section 5, an FPN value means that the delay time before the first garbage collection is performed in the tested SSD. Due to the available space for write I/Os, the steady-state-based SSD presents significantly smaller FPN values than that in the initial state.

In the initial state, the various FPN values result from the twofold reasons. (1) There is a great deal of space for user I/Os demand and more I/Os will be written in the SSD prior to reaching the threshold of garbage collection operation. (2) The DLSpace provides the page-level log space and efficiently lessens the write traffic under random-write workloads. Then, FPN values in *F1, F2, usr,* and *prn* verify the advantage of DLSpace over TLSpace (see Figure 7(b)). Because the sequential I/Os dominate *proj*, DLSpace has similar values to TLSpace.

In the steady state for the large-capacity SSD, increasing the number of log pages in data blocks enlarges the FPN values. In the context of insufficient user space, the page-level log space in a data block can temporarily accept the updated I/Os and the usage of log blocks is postponed. Experimental results indicate that FPN values (see Figure 7(b)) of the DLSpace-based SSD are larger than those in TLSpace-based SSD in most cases. FPN of the SSD in $D-1_S$ case achieves 55.8%, 25.0%, 32.7%, 31.4%, and 2.2% improvement over that in TLSpace-based SSD under *F1, F2, usr, prn, and proj* workloads, respectively.

In the large-capacity SSD, the scalability of DSpace is well validated by the large FPN comparing with TLSpace in the initial and the steady. Results in this test indicate that DLSpace deals well with write traffics under workloads. Thus, DLSpsce scalability helps in enlarging lifetime of the large-capacity SSD.

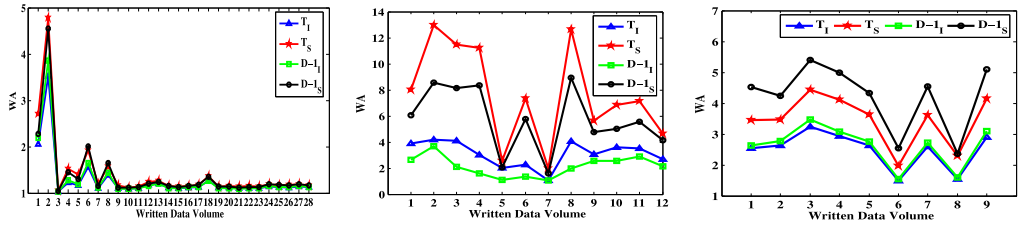


Fig. 8. WA trend of the tested SSD for DLSpace Scalability Evaluation in TLSpace-I ($T_I$), TLSpace-S ($T_S$), DLSpace-1-I ($D-1_I$), and DLSpace-1-S ($D-I_S$) cases driven by the three real-world workloads: (a) proj, (b) usr and (c) prn, *T* in the steady state ($T_S$), *D-1* in the initial state ($D-1_I$), and *D-1* in the steady state ($D-1_S$).

*A.3.3 Number of Block-erase Operations.* The larger the ERN value is, the fewer the number of available P/Es is in NAND Flash memory. ERN is one of important metrics that are used to evaluate log space allocation strategies under workloads. The ERN tightly relating to DLSpace has a significant impact on the scalability of DLSpace (see Figure 7(c)). In most workloads, the ERN values are determined by the amount of the random-write data size. It is a wise idea to keep the scalability of DLSpace to perform a small number of erase operations, which will alleviate the cost of P/Es and improve the SSD lifetime.

The initial-state SSD with much sufficient user space presents the lower number of ERN values than that in the steady state. Meanwhile, The DLSpace-based log space management reduces the number of erase operations under workload, such as ERN values in the $T_I$ and $D-1_I$, respectively. Note that the *F2* drops the ERN values because of its small written data size (i.e. 1.8GB). Moreover, the proj contains a great number of sequential requests and makes the TLSpace-based SSD's ERN close to that of the DLSpace-based SSD whether in the initial or the steady state. The *usr* dominated random I/Os in the 55GB written data enlarges the ERN values in the two states. However, the good scalability of DLSpace can keep the lower ERN in the $D-1_I$ and $D-1_S$ than that in the $T_I$ and $T_S$. Consequently, DLSpace reduces ERN of TLSpace by up to 60.0% and 42.0% in the initial state and the steady state, respectively. In this case, both garbage collection frequency and ERN are pushed down by DLSpace, comparing with TLSpace.

*A.3.4 Variations of WA and ERN.* In Section 6, we quantitatively analyze the scalability of DLSpace in terms of WA, FPN, and ERN in tested SSD. These metrics verify the DLSpace scalability from the perspectives of mean performance values in the tested SSD powered by TLSpace and DLSpace. In this Section, we study the lifetime variations of the tested SSD to study the scalability of DLSpace under workloads. In large-capacity SSD, the efficient scalability of DLSpace will make the trend of WA and ERN much smoother on the DLSpace-based SSD comparing to TLSpace.

Similarly to experiments in Section 6.3.4, we evaluate the trends of WA (see Figure 8) and the ERN (see Figure 9) under *usr, prn, and proj* workload in two states (i.e. the initial and the steady states). The method in Section 6.2 is employed to put the trends of WA and ERN under a microscope. A series of tests (i.e. test points) one of which has 5GB-write data divide a large-dataset trace (i.e. usr, prn, and proj) into 28, 11, and nine sub-traces, respectively. Data sizes of *usr, prn, and proj* are 55GB, 140GB, and 45GB, respectively. In Figure 8(a), the horizontal axis has a scale in 5GB-written-data sub-trace in *proj*. IOReplayer sequentially replays these sub-traces on the tested SSD. When a sub-trace completes, the IOReplayer records the values of WA and ERN on the tested SSD after ten minutes.

In Figure 8 and Figure 9, we observe that WA and ERN have the similar trend on the TLSpace and DLSpace-based tested SSD in the initial or the steady state. Thanks to the initial state, where

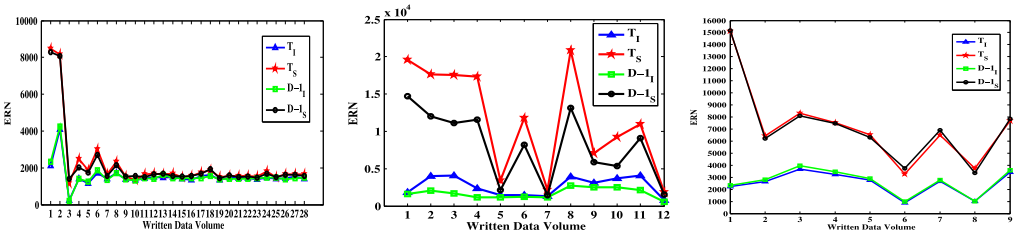


Fig. 9. ERN trend of the tested SSD for DLSpace Scalability Evaluation in TLSpace-I ($T_I$), TLSpace-S ($T_S$), DLSpace-1-I ($D-1_I$), and DLSpace-1-S ($D-I_S$) cases driven by the three real-world workloads: (a) proj, (b) usr, and (c) prn.

TLSpace- and DLSpace-based test-SSD has enough data space and log space for the write data in the trace. In the steady state, the average value at each point of sub-trace presents much higher than that in the initial state, which results from twofold. First, the user space of the tested SSD in this state is smaller than that in the initial state. Second, the access I/Os in *proj* follow sequential mode. However, the DLSpace-based tested SSD has lower values of WA and ERN than that TLSpace-based counterpart in two states. Many log pages in log space make the trend of WA or ERN smooth; thus, the good scalability of the DLSpace confirms that WA or ERN of the SSD does not significantly change under sequential-requests-intensive workloads.

In contrast, DLSpace (e.g., $D-1_S$ and $D-1_I$) noticeably reduces WA and ERN of the TLSpace-based SSD (see Figure 8(b) and (c) and Figure 9(b) and (c)) under random-write-intensive workloads (e.g., *usr* and *prn*) in two states. The reason for this trend is attributed to the log page space in data blocks, which postpones garbage collection by buffering random I/Os in the data blocks and delaying the utilization of log blocks. Thus, DLSpace reduces page copy-write and block-erase operations to slow down WA. It is reasonable that WA tendency is much smoother with more log pages configured in log space (Figures 8(b) and (c)). ERN of the tested SSD (see Figures 9(b) and (c)) has the same trend. We can see that the DLSpace in the large-capacity SSD still has the good scalability to make the trends of WA and ERN values smooth as soon as possible.

### A.4 Summary Conclusion

As we mentioned above, the scalability of DLSpace depends on how the SSD lifetime varies with the capacity of the tested SSD. In our work, we quantitatively analyzed SSD lifetime regarding *WA*, *FPN*, and *ERN*. A large-capacity SSD presents different performance in the initial state and the steady state. Therefore, we performed extensive experiments on the impacts of DLSpace on WA, FPN, and ERN on a large-capacity tested SSD in the two states. Experimental results reveal that:

(1) DLSpace maintains similar performance to that of TLSpace under sequential workloads or in an initial state. This phenomenon is attributed to infrequent copy-rewrite and block-erase operations as well as sufficient data space.
(2) DLSpace is significantly superior to the conventional TLSpace under random workloads in the initial state or the steady state. The reason is that the page-level log space in the data blocks accepts random write I/Os in the located blocks and improves SSD performance.
(3) Our DLSpace still exhibits smooth WA and ERN trends comparing with TLSpace due to a large number of log pages in DLSpace.

We can see that the noticeable advantages of DLSpace in the improvement of SSD lifetime from the three metrics (i.e., WA, FPN, and ERN) and the smooth trends of WA and ERN on a large-capacity SSD. Thus, DLSpace has the good scalability on low- and large-capacity tested SSDs.

## ACKNOWLEDGMENTS

We thank the SoliWare Corporation providing assistance in designing the real-world SSD testbed.

## REFERENCES

R. Agarwal and M. Marrow. Dec. 2010. A closed-form expression for write amplification in NAND flash. In *Proceedings of the IEEE Global Communications Conf. Exhibition & Industry Forum (GLOBECOM'10)*. IEEE, 1846–1850.

J. Axboe and A. D. Brunelle. 2007. Blktrace User Guide. http://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html.

K. Bates and B. McNutt. 2007. I/O and Search Engine I/O, UMass Trace Repository.

Yuan-Hao Chang, Po-Liang Wu, Tei-Wei Kuo, and Shih-Hao Hung. 2012. An adaptive file-system-oriented FTL mechanism for flash-memory storage systems. *ACM Trans. Embed. Comput. Syst.* 11, 1 (2012), 9.

F. Chen, D. A. Koufaty, and X. Zhang. 2009. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*. ACM, New York, NY, 181–192.

F. Chen, T. Luo, and X. Zhang. Feb. 2011. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state driver. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. USENIX Association, Berkeley, CA, 77–90.

D. Jung, Y. H. Chae, H. Jo, J. S. Kim, and J. Lee. Sept. 2007. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In *Proceedings of the 2007 International Conference Compilers, Architecture, and Synthesis for Embedded Systems (CASES'07)*. ACM, New York, NY, 160–164.

T. Frankie, G. Hughes, and K. Kreutz-Delgado. Aug. 2011. SSD trim commands considerably improve overprovisioning. In *Proceedings of the Flash Memory Summit 2011*. 1–19.

T. Frankie, G. Hughes, and K. Kreutz-Delgado. Mar. 2012. A mathematical model of the TRIM command in NAND-flash SSDs. In *Proceedings of the 50th Annual Southeast Regional Conference*. ACM, New York, NY, 59–64.

E. Gal and S. Toledo. 2005. Algorithms and data structures for flash memories. *ACM Comput. Surv.* 37, 2 (2005), 138–163.

L. M. Grupp, J. D. Davis, and S. Swanson. Feb. 2012. The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, San Jose, CA, 1–8.

Yong Guan, Guohui Wang, Chenlin Ma, Renhai Chen, Yi Wang, and Zili Shao. 2017. A block-level log-block management scheme for MLC NAND flash memory storage systems. *IEEE Trans. Comput.* 66, 9 (2017), 1464–1477.

Jiayang Guo, Yiming Hu, Bo Mao, and Suzhen Wu. 2017. Parallelism and garbage collection aware I/O scheduler with improved SSD performance. In *Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS'17)*. IEEE, Orlando, FL, 1184–1193.

Jie Guo, Chuhan Min, Tao Cai, and Yiran Chen. 2016. A design to reduce write amplification in object-based NAND flash devices. In *Proceedings of the 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS'16)*. IEEE, Pittsburgh, PA, 1–10.

A. Gupta, Y. Kim, and B. Urgaonkar. Mar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th International Conf. Architectural Support for Programming Languages and Operating System (ASPLOS'09)*. ACM, New York, NY, 229–240.

K. Ha and J. Kim. 2011. A program context-aware data separation technique for reducing garbage collection overhead in NAND flash memory. In *Proceedings of the 7th the International Workshop on Storage Network Architecture and Parallel I/O (SNAPI'11)*. IEEE, Los Alamitos, 1–10.

J. Hsieh, L. Chang, and T. Kuo. 2005. Efficient on-line identification of hot data for flash-memory management. In *Proceedings of the 2005 ACM Symposium on Applied Computing*. ACM, New York, NY, 838–842.

J. Hu, H. Jiang, L. Tian, and L. Xu. 2010. PUD-LRU: An erase-efficient write buffer management algorithm for flash memory SSD. In *Proceedings of the International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'10)*. IEEE, Los Alamitos, CA, 69–78.

Jian Hu, Hong Jiang, Lei Tian, and Lei Xu. 2012. GC-ARM: Garbage collection-aware RAM management for flash based solid state drives. In *Proceedings of the IEEE 7th International Conference on Networking, Architecture, and Storage*. IEEE, Los Alamitos, CA, 134–143.

X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. May 2009. Write amplification analysis in flash-based solid state drives. In *Proceedings of the The Israeli Experimental Systems Conference (SYSTOR'09)*. ACM, New York, NY, 1–9.

X. Hu and R. Haas. Mar. 2010. *The Fundamental Limit of Flash Random Write Performance: Understanding, Analysis and Performance Modelling*. Technical Report. IBM Research Report, RZ 3771.

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the International Conference on Supercomputing*. ACM, New York, NY, 96–107.

Sheng Min Huang and Li Pin Chang. 2016. Exploiting page correlations for write buffering in page-mapping multichannel SSDs. *ACM Trans. Embed. Comput. Syst.* 15, 1 (2016), 1–25.

I. Ilias. May 2010. *Performance of the Greedy Garbage-Collection Scheme in Flash-based Solid-State Drives.* Technical Report. IBM Research Report, RZ 3769, IBM.

A. Jagmohan, M. Franceschini, and L. Lastras. May 2010. Write amplification reduction in NAND flash through multi-write coding. In *Proceedings of the IEEE 26th Symposium Mass Storage Systems and Technologies (MSST'10)*. IEEE, Los Alamitos, CA, 1–6.

Xavier Jimenez, David Novo, and Paolo Ienne. 2014. Wear unleveling: Improving NAND flash lifetime by balancing page endurance. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'14)*, Vol. 14. USENIX Association, Santa Clara, CA, 47–59.

J. Kang, H. Jo, J. S. Kim, and J. Lee. 2006. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the ACM/IEEE 6th International Conf. Embedded Software (EMSOFT'06)*. ACM, New York, NY, 161–170.

J. Kim, C. Lee, S. Lee, I. Son, J. Choi, S. Yoon, H. Lee, S. Kang, Y. Won, and J. Cha. Apr. 2012. Deduplication in SSDs: Model and quantitative analysis. In *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST'12)*. IEEE, Los Alamitos, CA, 1–12.

H. Kim and S. Ahn. Feb. 2008. BPLRU: A buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. USENIX Association, San Jose, CA, USA., 1–14.

Hyunsuk Kim, Su-Jin Ahn, Yu Gyun Shin, Kyupil Lee, and Eunseung Jung. 2017. Evolution of NAND flash memory: From 2D to 3D as a storage market leader. In *Proceedings of the 2017 IEEE International Conference Memory Workshop (IMW'17)*. IEEE, Los Alamitos, CA, 1–4.

Hyeong-Jun Kim, Young-Sik Lee, and Jin-Soo Kim. 2016. NVMeDirect: A user-space I/O framework for application-specific optimization on NVMe SSDs. In *Proceedings of the HotStorage*. USENIX Association, 41–45.

Mark Kyrnin. July 2017. How the M.2 SSD Is Going to Make Your PC Even Faster. https://www.lifewire.com/what-is-m2-833453.

C. Lee, D. Sim, J. Hwang, and S. Cho. 2015. F2FS: A New File System for Flash Storage. 273–286.

Eunji Lee, Julie Kim, Hyokyung Bahn, Sunjin Lee, and Sam H Noh. 2017. Reducing write amplification of flash storage through cooperative data management with NVM. *ACM Trans. Stor.* 13, 2 (2017), 12.

J. Lee, S. Hur, and J. Choi. 2002. Effects of floating-gate interference on NAND flash memory cell operation. *IEEE Electron. Device Lett.* 23, 5 (2002), 264–266.

Jaeduk Lee, Jaehoon Jang, Junhee Lim, Yu Gyun Shin, Kyupil Lee, and Eunseung Jung. 2016. A new ruler on the storage market: 3D-NAND flash for high-density memory and its technology evolutions and challenges on the future. In *Proceedings of the 2016 IEEE International Electron Devices Meeting (IEDM'16)*. IEEE, Los Alamitos, CA, 11.2.1–11.2.4.

S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim. 2009. FlexFS: A flexible flash file system for MLC NAND flash memory. In *Proceedings of the Conference on USENIX Annual Technical Conference*. USENIX Association, San Jose, CA, 1–14.

S. Lee, T. Kim, K. Kim, and J. Kim. Feb. 2012. Lifetime management of flash-based SSDs using recovery-aware dynamic throttling. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, San Jose, CA, 1–14.

S. Lee, J. Park, K. Fleming, J. Kim, and Arvind. 2011. Improving performance and lifetime of solid-state drives using hardware-accelerated compression. *IEEE Trans. Cons. Electron.* 57, 4 (2011), 1732–1739.

S. W. Lee, D. J. Park, T. S. Chung, D. H. Lee, S. Park, and H. J. Song. 2007. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.* 6, 3 (2007), 18.

M. Lenzlinger and E. H. Snow. 1969. Fowler-nordheim tunneling into thermally grown SiO2. *J. Appl. Physics* 40, 1 (1969), 278–283.

S. Lim, S. Lee, and B. Moon. May 2010. FASTer FTL for enterprise-class flash memory SSDs. In *Proceedings of the International 6th Workshop on Storage Network Architecture and Parallel I/Os (SNAPI'10)*. IEEE, Los Alamitos, CA, 3–12.

Duo Liu, Kan Zhong, Tianzheng Wang, Yi Wang, Zili Shao, Edwin Hsing-Mean Sha, and Jingling Xue. 2017. Durable address translation in PCM-based flash storage systems. *IEEE Trans. Parallel Distrib. Syst.* 28, 2 (2017), 475–490.

Xin Liu, Ying Lu, Yutong Lu, Chunjia Wu, and Jieting Wu. 2016. masFS: File system based on memory and SSD in compute nodes for high performance computers. In *Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS'16)*. IEEE, Los Alamitos, CA, 569–576.

Y. Lu, J. Shu, W. Zheng, and S. Li. 2013. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*, Vol. 5. USENIX Association, San Jose, CA, 257–270.

Fabio Margaglia, Gala Yadgar, Eitan Yaakobi, Yue Li, Assaf Schuster, and Andre Brinkmann. 2016. The devil is in the details: Implementing flash page reuse with WOM codes. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'16)*. USENIX Association, 95–109.

Microsoft. 2007. Microsoft Enterprise Traces. Retrieve from http://iotta.snia.org/traces/list/BlockIO.

C. Min, K. Kim, H. Cho, S. Lee, and Y. Eom. 2012. SFS: Random write considered harmful in solid state drives. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, 139–155.

V. Mohan, T. Siddiqua, S. Gurumurthi, and M. R. Stan. June 2010. How I learned to stop worrying and love flash endurance. In *Proceedings of the 2nd USENIX Conference Hot Topics in Storage and File Systems (Hotstorage'10)*. USENIX Association, 1–5.

S. Moon and A. L. N. Reddy. 2012. Write amplification due to ECC on flash memory or leave those bit errors alone. In *Proceedings of the IEEE 28th Symposium Mass Storage Systems and Technologies (MSST'12)*. IEEE, Los Alamitos, CA, 1–6.

X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, and D. K Panda. 2011. Beyond block I/O: Rethinking traditional storage primitives. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA'11)*. IEEE, Los Alamitos, CA, 301–311.

D. Park and D. H.C. Du. 2011. Hot data identification for flash-based storage systems using multiple bloom filters. In *Proceedings of the 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*. IEEE, IEEE, Los Alamitos, CA, 1–11.

Hyunchan Park, Seehwan Yoo, Cheol-Ho Hong, and Chuck Yoo. 2016. Storage SLA guarantee with novel SSD I/O scheduler in virtualized data centers. *IEEE Trans. Parallel Distrib. Syst.* 27, 8 (2016), 2422–2434.

S. Park and K. Shen. 2012. Fios: A fair, efficient flash i/o scheduler. In *Proceedings of the of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, Los Alamitos, CA, 1–15.

Agostino Pirovano. 2017. *Physics and Technology of Emerging Non-Volatile Memories*. Springer, Cham, Switzerland. 27–46.

Bianca Schroeder, Arif Merchant, and Raghav Lagisetty. 2017. Reliability of NAND-based SSDs: What field studies tell us. *Proc. IEEE* 105, 9 (2017), 1751–1769.

K. Smith. Aug. 2012. Understanding SSD over provisioning. In *Proceedings of the Flash Memory Summit 2012*. 1–16.

SNIA. May 2012. *Solid State Storage (SSS) Performance Test Specification (PTS) Client Version 1.1*. SNIA.

G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. 2010. Extending SSD lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*. USENIX Association, 101–114.

Hui Sun, Xiao Qin, Hong Jiang, Jianzhong Huang, and Changsheng Xie. 2015. RB-Explorer: An accurate & practical approach to write amplification measurement for SSDs. *IEEE Trans. Comput.* 64, 4 (2015), 1133–1148.

H. Sun, X. Qin, and C. Xie. 2013. Measuring and analyzing write amplification characteristics of solid state disks. In *Proceedings of the International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'13)*. IEEE, Los Alamitos, CA, 212–221.

H. Wang and P. Huang. 2013. A novel I/O scheduler for SSD with improved performance and lifetime. In *Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST'13)*. IEEE, Los Alamitos, CA, 5–9.

W. Wang, Z. Wang, K. Ni, Y. Zhao, J. Ma, and L. Pan. 2012b. A simplified model of write amplification for solid state drives adopting page-level address translation mechanism. *Adv. Mater. Res.* 383 (2012), 2156–2160.

Yi Wang, Luis Angel D. Bathen, Zili Shao, and Nikil D. Dutt. 2012a. 3D-FlashMap: A physical-location-aware block mapping strategy for 3D NAND flash memory. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, Dresden, Germany, 1307–1312.

Debao Wei, Libao Deng, Liyan Qiao, Peng Zhang, and Xiyuan Peng. 2016. PEVA: A page endurance variance aware strategy for the lifetime extension of NAND flash. *IEEE Trans. VLSI Syst.* 24, 5 (2016), 1749–1760.

Q. Wei, B. Gong, S. Pathak, B. Veeravalli, L. Zeng, and K. Okada. May 2011. WAFTL: A workload adaptive flash translation layer with data partition. In *Proceedings of the IEEE 27th Symposium Mass Storage Systems and Technologies (MSST'11)*. IEEE, Los Alamitos, CA, 1–12.

G. Wu and X. He. Apr. 2012. Delta-FTL: Improving ssd lifetime via exploiting content locality. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12)*. ACM, New York, NY, 253–266.

G. Wu, X. He, and B. Eckart. 2012. An adaptive write buffer management scheme for flash-based SSDs. *ACM Trans. Stor.* 8, 1 (2012), 1.

Suzhen Wu, Bo Mao, Yanping Lin, and Hong Jiang. 2017. Improving performance for flash-based storage systems through GC-Aware cache management. *IEEE Trans. Parallel Distrib. Syst.* 28, 10 (2017), 2852–2865.

L. Xiang and K. Brian M. Jan. 2012. An improved analytic expression for write amplification in NAND flash. In *Proceedings of the International Conference Computing, Networking and Communications (ICNC'12)*. IEEE, Los Alamitos, CA, 497–501.

Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. 2015. Performance analysis of NVMe SSDs and their implication on real world databases. In *Proceedings of the 8th ACM International Systems and Storage Conference*. ACM, New York, NY, 6:1–6:11.

Q. Yang and J. Ren. Feb. 2011. I-CASH: Intelligently coupled array of SSD and HDD. In *Proceedings of the IEEE 17th Symp. High Performance Computer Architecture (HPCA'11)*. IEEE, Los Alamitos, CA, 278–289.

Jiacheng Zhang, Jiwu Shu, and Youyou Lu. 2016. ParaFS: A log-structured file system to exploit the internal parallelism of flash devices. In *Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 87–100.

Ke Zhou, Shaofu Hu, Ping Huang, and Yuhong Zhao. 2017. LX-SSD: Enhancing the lifespan of NAND flash-based memory via recycling invalid pages. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST'17)*. IEEE, 1–13.

Received October 2017; revised June 2018; accepted October 2018