aerospace

MDPI

Article

Delay-D: Research on the Lifespan and Performance of Storage Devices in Unmanned Aerial Vehicles

Donghyun Kang

Department of Computer Engineering, College of IT Convergence, Gachon University, Seongnam-si 13120, Republic of Korea; donghyun@gachon.ac.kr

Abstract: Despite the technological achievements of unmanned aerial vehicles (UAVs) growing in academia and industry, there is a lack of studies on the storage devices in UAVs. However, this is an important aspect because the storage devices in UAVs have a limited lifespan and performance and are rarely replaced due to a system-on-chip architecture. In this paper, we study how UAVs impact the lifespan and performance of the underlying storage device while capturing images during overflight. We also propose a new lifespan and performance-saving mechanism, called `Delay-D`, which is designed at the kernel level to efficiently utilize the features of NAND flash-based storage devices. To confirm the effectiveness of `Delay-D`, we implement a simulator that replays realistic write patterns on UAVs and evaluate quantitative experiments in two different experimental environments. In our evaluation, `Delay-D` demonstrates the dramatic extension possibility of the lifespan by reducing the number of extra writes inside the storage device and improving the overall performance by up to 2.1× on the commercial NVMe SSD.

Keywords: unmanned aerial vehicles; NAND flash-based storage devices; lifespan; performance

check for updates

Citation: Kang, D. Delay-D: Research on the Lifespan and Performance of Storage Devices in Unmanned Aerial Vehicles. *Aerospace* **2024**, *11*, 47. https://doi.org/10.3390/aerospace11010047

Academic Editor: Bosko Rasuo

Received: 5 December 2023
Revised: 30 December 2023
Accepted: 31 December 2023
Published: 31 December 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Unmanned aerial vehicles (UAVs) are rapidly evolving due to the demand spike in various industries, such as national defense, disaster management, and agriculture [1–3]. Recent advances in hardware technologies of UAVs, such as cameras, GPS, telemetry, storage, and sensor modules, significantly accelerate their widespread deployment. The capabilities of UAVs have garnered considerable research interest, as they provide valuable outputs (e.g., images of the overflight, flight logs of UAVs, and real-time video) [3]. Many researchers and engineers have focused on the usage scenarios of UAVs and how to utilize the real-time data collected from them [2,4]. For example, some researchers have concentrated on synchronizing the positions of multiple UAVs based on captured image information and azimuth elements within the UAVs [5]. Scientists are beginning to utilize information from UAVs to provide more intelligent services based on deep learning (DL) technologies [6–8]. With DL technologies, common trends include automatically detecting and counting the number of objects (i.e., humans, cars, ships, and animals) based on image datasets composed of a large number of images captured from UAVs [7,8]. One researcher explored the challenges associated with the semantic segmentation of UAV imagery based on images captured from UAVs [6].

Meanwhile, all commercial UAVs have a limited lifespan due to some hardware resources, such as batteries and storage devices; even the most expensive devices have a limited lifespan. The lifespan of the battery on UAVs is a well-known issue, and much effort has been made to extend it [9]. However, it is not widely recognized that the limited number of writes to NAND flash memory limits the lifespan of storage devices (e.g., eMMC, UFS, and SD cards) built into UAVs [10–13]. The aforementioned technology advancements are worsening the lifespan of storage devices over time as the flight time of UAVs increases. The reason for this is that increased flying time leads to more data being written on a UAV. It can

process and record more data to guarantee system reliability (i.e., logs) and to store a larger amount of images in its primary storage device. Some commercial UAVs provide their primary storage device (i.e., eMMC and UFS) in the form of a system-on-chip, meaning we cannot replace it with a new device even at the end of the storage lifespan [14,15]. Unfortunately, the shorter lifespan may negatively impact the performance of storage devices due to the physical characteristics of NAND flash memory [10–13].

In this paper, we focus on the lifespan and performance issues of storage devices on UAVs. In NAND flash-based storage devices, the write patterns directly influence not only lifespan but also performance; sequential writes are well-known to be beneficial for lifespan and performance due to their lightweight overheads of internal behaviors. Unfortunately, monitoring the internal architectures and behaviors in real-time on commercial storage devices is not feasible due to their security property protection (we describe this in more detail in the next section). We use FEMU [16] based on QEMU, which emulates all kinds of features in NAND flash-based storage devices using virtualization technology and provides various internal parameters. We then determine that UAVs may have side effects when recording their images during overflight in terms of the lifespan and performance of the storage devices.

This paper presents delayed discard (`Delay-D`), a new lifespan and performance-saving mechanism, which focuses on the write patterns issued from the applications of UAVs. `Delay-D` enables the *discard* command, which is an extra command used to postpone the lifespan and enhance performance on NAND-based storage devices, and it tunes the command to cooperate with the *allocation mechanism* on the file system. As a result, `Delay-D` can address the aforementioned issue of lifespan with the key design goal: *stably guaranteeing the lifespan and performance of the storage devices on UAVs*. For evaluation, we implemented `Delay-D` on the Ext4 file system, which is widely used as a default file system on UAVs [15]. We evaluated `Delay-D` based on simulation tools that generate write operations similar to writing images of overflight on UAVs. We performed the evaluation on the FEMU-emulated device and two real-world storage devices to clearly confirm the effectiveness of `Delay-D`. `Delay-D` shows stable performance as well as improved lifespan compared with traditional mechanisms.

In summary, this paper makes the following contributions:

- We first dive into the write patterns that can be generated during UAV flights and understand the side effects in terms of lifespan and performance of NAND-based storage devices. We particularly focus on two different write patterns for storing images of overflight on UAVs.
- We design and implement `Delay-D` as the first lifespan-saving mechanism for storage devices on UAVs. `Delay-D` specifically enables the discard commands with almost zero cost by replacing the commands with write operations and locally tunes the timing and granularity of the commands.
- We perform a comprehensive evaluation using one emulator (i.e., FEMU) and two real-world storage devices to confirm the performance benefit of `Delay-D`, and compare how the lifespan can be affected by the write patterns based on the emulator.

The rest of this paper is organized as follows. Section 2 briefly introduces the features of NAND-based storage devices and presents an in-depth analysis of write patterns that can be generated on UAVs and their side effects. Section 3 proposes our lifespan-saving and high-performance `DISCARD` mechanism, called `Delay-D` in detail. In Section 4, we experimentally compare the lifespan and performance of `Delay-D` with traditional ones under realistic usage scenarios of UAVs. Finally, we survey related works in Section 5; Section 6 concludes this work.

2. Background

In this section, to easily understand the benefits of our work, we delve deeply into the features of NAND-based storage devices and describe the relationship between the storage devices and UAVs [14,15].

2.1. NAND Flash-Based Storage Devices

NAND-based storage devices, such as eMMC, UFS, SD card, and SSD, are commonly used as the storage media of UAVs due to their high performance and reliability [15]. Unfortunately, these storage devices have some hardware limitations compared to traditional storage devices (i.e., HDDs): flash endurance limits, no in-place updates, and asymmetric read and write latencies [10–13,17]. To hide these limitations, NAND-based storage devices generally adopt a special software layer inside, called the flash translation layer (FTL). This layer is responsible for three crucial operations inside the storage devices: (1) LBA-to-PBA mapping, (2) garbage collection (GC), and (3) wear-leveling.

LBA-to-PBA mapping: This operation allows overwriting data inside the same logical block address (LBA) using a *mapping table* that tracks the location of the physical block address (PBA) corresponding to the location of the LBA. If an overwrite operation at the given LBA is required, FTL first writes data to the new PBA location and then updates the mapping table corresponding to LBA, so as to point to the new PBA location (we call this the remapping mechanism). The old PBA—unmapped after the overwrite—is marked *invalid* and will not be reused before the GC operation.

Garbage Collection: The number of invalid PBAs increases as the device is written by the above remapping mechanism. If there are no valid PBAs inside the storage device, the FTL triggers the garbage collection (GC) operation, which erases one or more blocks to convert invalid PBAs into valid ones. At this point, the remaining valid PBAs in the blocks to be erased must be copied to new PBAs; this process causes significant overhead both in terms of lifespan and performance. Note that a block with no valid PBAs can be simply reused with almost zero overhead. In summary, a single write operation can be amplified to two or more write operations if the single write triggers the GC operation; this is known as the *write amplification factor* (WAF).

Wear-leveling: Unfortunately, the aforementioned write (program)/erase operations (i.e., P/E cycles) cause device-level damage to NAND flash memory cells, which gradually wears out the lifespan of the storage device. Over time, some flash cells may become unusable and unreliable. It is important to maintain an even distribution of P/E cycles across all cells so that they can all survive together until reaching the maximum allowable P/E cycles (i.e., flash endurance limits). In this paper, while we do not include wear-leveling as an explicit design goal, it is necessary to extend the lifespan of NAND-based storage devices.

Meanwhile, operating systems (OSs) provide a special command, `DISCARD`, to enhance the lifespan of NAND-based storage devices. Some researchers and engineers refer to `DISCARD` as the TRIM command from a hardware perspective, but it is essentially the same operation. After a user deletes a file on the file system, the OS issues `DISCARD` commands to inform the underlying storage device that LBAs belonging to the removed file are no longer valid. After this, the FTL invalidates the LBAs specified by the `DISCARD` commands, helping to eliminate unnecessary copy overhead when the next GC operation runs. However, despite such advantages of the `DISCARD` command, most users typically disable the `DISCARD` command. The reason behind this is that it may lead to performance collapse due to *ordering overhead* and *invalidation and queuing overhead*. (1) Since it is possible to reallocate the LBAs belonging to the removed file for a new file, the order between `DISCARD` commands and write operations should be maintained. Otherwise, data in the new file could be unintentionally invalidated on the underlying storage device. (2) The cost of invalidation in the mapping table can sometimes be unexpectedly high if it requires extensive access to the mapping table, or if a queuing delay from the host to the underlying storage device occurs due to outstanding commands.

In summary, many efforts have been devoted to hiding the unique characteristics of NAND flash-based storage devices. However, lifespan issues of storage devices are still open.

2.2. Simulation of Write Patterns on UAVs

The technology of UAVs has been evolving and growing rapidly in recent years [1–3]. In particular, many UAV platforms have been proposed and evaluated, carrying various sensor devices, such as cameras, LIDAR, thermals, etc., in terms of practical flight [1,18]. In various UAV platforms and systems, the storage device is a crucial component because a UAV must record onboard flight logs that include valuable information during flights (e.g., collected sensor values, commands, and signals) [19]. In addition, one of the most utilized functions in UAVs is capturing an image of the overflight to transfer it to the ground control station or to store it in the underlying storage device; this also fundamentally requires storage media. UAVs capture images of overflight through one or more mounted cameras and can store tens of thousands of photos in the underlying storage devices. For example, the VisDrone-Dataset open dataset includes 288 video clips and 10,209 static images [8,20].

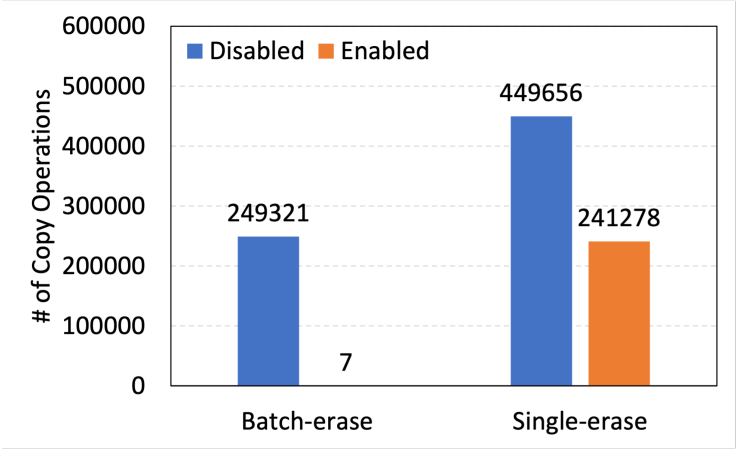
As far as we know, image files stored in the storage device of a UAV are removed after backing them up onto separate dedicated storage devices. In other words, the process of storing files one by one and then removing them all at once is frequently repeated in the storage device of UAVs (we refer to this as **batch-erase** in this paper). If there is no free space available to save a newly captured image during flight, older files may be inevitably removed in the order of capture time to secure free space. In this case, the process of removing one image file and saving one newly captured file may be repeated during the flight of the UAV (we call this **single-erase** in this paper). Note that if the `DISCARD` command has been disabled, the LBAs belonging to the removed image files may negatively affect the lifespan and performance of the storage device for the reasons previously mentioned.

To investigate the impact on lifespan and performance, we simulated realistic UAV scenarios that trigger batch-erase and single-erase scenarios, respectively. We created 10,000 files, each 1 MiB in size, and then removed them either all at once for the batch-erase scenario or one by one for the single-erase scenario (this process was repeated 50 times). For our experimental study, we monitored both the performance and lifespan of the FEMU-emulated device using the above scenarios (the detailed evaluation setup will be described in Section 4.1).

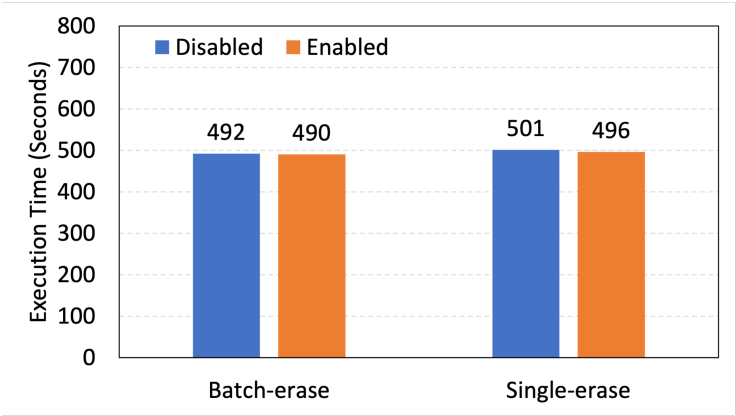
Figure 1 shows the total number of copy operations caused by garbage collection inside the storage device and the total execution time. As shown in Figure 1a, the number of copy operations with `DISCARD` disabled (i.e., Disabled) significantly increases compared to `DISCARD` enabled (i.e., Enabled). In particular, in batch-erase, the data of the LBAs belonging to the already removed files must be unnecessarily copied while running GC operations, which can increase the number of copy operations by up to 30,000 times. Meanwhile, unlike lifespan, the total execution time of the enabled scenario, measured from the start until the end of 50 iterations, shows a result similar to that of the Disabled scenario (see Figure 1b). To compare realistic performance, we additionally conducted the same evaluation on two commercial NAND-based storage devices.

Figure 2 shows the overall performance of two different erase scenarios. Surprisingly, the results measured on real-world storage devices reveal two interesting observations. First, the number of copy operations is barely proportional to its performance; this is intriguing because it surely has a negative impact on the lifespan of the storage device. Second, the SATA SSD reveals the aforementioned overheads of the `DISCARD` command, but this is negligible because it can compensate for the lifespan of NAND-based storage devices.

In summary, the `DISCARD` command can significantly impact the lifespan of storage devices, but it is rarely used in the real world because enabling the command sometimes has a negative impact on overall performance. Importantly, the lifespan of storage devices on UAVs will steadily deteriorate over time due to the above-mentioned erase scenarios, but it is difficult to replace the embedded storage device on UAVs. Such challenges and limitations on UAVs strongly motivate us to design a new `DISCARD` mechanism for UAVs.

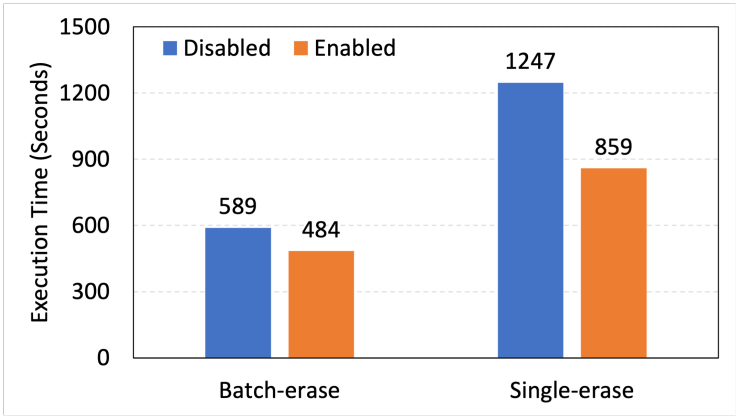


(**a**) Lifespan Impact



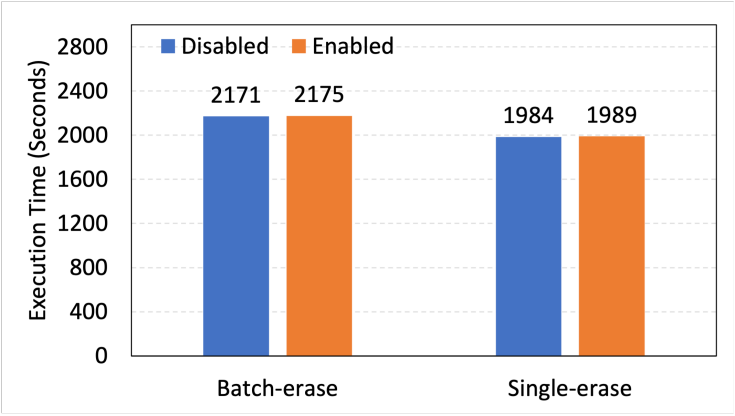
(**b**) Performance Impact

Figure 1. The lifespan and performance impact of the `DISCARD` command on the FEMU-emulated device. Lower is better.



(**a**) NVMe SSD (Samsung 980 PRO)

Figure 2. *Cont.*



(**b**) SATA SSD (Samsung 860 PRO)

Figure 2. The performance impact of the `DISCARD` command on two real-world NAND flash-based storage devices. Lower is better.

3. Design of `Delay-D`

We now propose a novel `DISCARD` mechanism, named delayed discard (`Delay-D`), designed in-kernel to utilize file system behaviors and mitigate queuing delay. Algorithm 1 lists the pseudo-code of major functions of `Delay-D`, which is designed with the following goals in mind:

- It should minimize the overheads caused by the `DISCARD` command to ensure high performance when capturing images during overflight.
- It should prolong the lifespan of the storage device on UAVs by efficiently transferring `DISCARD` commands, thereby reducing the number of internal copy operations during GC operations.
- It should be conceptually simple and transparently support applications on UAVs running on a wide range of storage devices, from traditional eMMC to state-of-the-art SSDs.

3.1. Zero-Cost Invalidation Using Write Operations

The key idea of `Delay-D` is to replace `DISCARD` commands with write operations by employing allocation and reclaim mechanisms in the conventional file system.

In general, a file system is a crucial component of an OS and is responsible for managing how data is organized in the underlying storage device. For example, when an image of overflight is captured, the file system allocates one or more LBAs based on the size of the captured image and issues write operations to persistently store the data corresponding to these LBAs in the underlying storage device. If a user deletes a file on the storage device, the file system reclaims the LBAs corresponding to the file for future reuse. Finally, the OS transfers `DISCARD` commands, including the reclaimed (or invalid) LBAs, to the storage device (if the command is enabled). Unfortunately, as mentioned before, such mechanisms may cause the ordering overhead if the file system reallocates a reclaimed LBA to new data before issuing a `DISCARD` command corresponding to the reclaimed LBA.

Fortunately, we identified an opportunity in these mechanisms to achieve the same impact as the `DISCARD` command. This is possible because a write operation that includes the reclaimed LBA must be treated like an overwrite operation in NAND-based storage devices. As mentioned in LBA-to-PBA mapping, the FTL inside the device should invalidate the old PBA corresponding to the LBA in the mapping table before writing received data to a new PBA. In the rest of this paper, we refer to this as *zero-cost invalidation* because it is almost free in terms of lifespan. However, to enable zero-cost invalidation, we must overcome the following two challenges:

1. The order and timing in which a `DISCARD` command is transferred to the storage device.
2. The subsequent `DISCARD` command, which can unintentionally invalidate new data.

Algorithm 1: Major functions of `Delay-D`

Input: LBA: logical block address, MAP: mapping table, FD: file descriptor

```
1:  function allocation(size):
2:  while # of allocated LBAs < size do
3:    LBA ← allocate_FreeLBAList()    // It depends on the file system
4:    update_MappingTable(LBA)        // If LBA exists, update its status to inactive
5:  return a set of LBAs
6:
7:  function reclaim(FD):
8:  while get_FileSize(FD) do
9:    LBA ← findLBA(FD)
10:   insert_FreeLBAList(LBA)
11:   insert_MappingTable(LBA)      // LBA is initialized as active status
12:   reduce_FileSize(FD)
13: return
14:
15: deamon transfer_Discard():
16: while pre-defined delay time do
17:   DISCARD ← get_ActiveLBAs(MAP)
18:   /* If LBAs are adjacent, merge them into a single DISCARD command */
19:   DISCARD ← merge_LBAs(DISCARD)
20:   issue_Command(DISCARD)
21:   free_InactiveLBAs(MAP)
```

In order to address the first challenge, `Delay-D` delays issuing `DISCARD` commands as long as possible, allowing new data allocated to the reclaimed LBA to be reflected in the storage device in advance. Importantly, this delay is particularly meaningful in UAVs because the write patterns on UAVs have a high possibility of zero-cost invalidation after performing the single-erase. Since file systems tend to reallocate LBAs removed in the past, `Delay-D` gives the file systems on UAVs more opportunities for zero-cost invalidation by delaying `DISCARD` commands. In other words, the reclaimed LBAs from the single-erase scenario are reallocated as much as possible while storing a new image of overflight, leading to zero-cost invalidation. To tackle the second challenge, `Delay-D` categorizes LBAs required for issuing `DISCARD` commands into two types based on whether LBAs were reallocated or not and omits unnecessary `DISCARD` commands by handling them independently. To this end, `Delay-D` employs an auxiliary in-memory mapping table, which is a tree-structured index, and manages the order of active LBAs to be transferred to the storage device via the `DISCARD` command. After a predefined time for the delay, `Delay-D` transfers a set of `DISCARD` commands consisting of active LBAs and frees the remaining LBAs with inactive status from the mapping table to prevent unintended invalidation. As a result, `Delay-D` can reduce the number of `DISCARD` commands and minimize the overheads caused by enabling the command; it can eliminate the ordering overhead and have a positive impact on the congestion of the queue.

3.2. Sorting and Merging LBAs for `DISCARD` Commands

`Delay-D` is designed to efficiently mitigate queue congestion during data transfer and to capitalize on flash-friendly patterns. As previously mentioned, `Delay-D` can reduce the queue congestion issue caused by outstanding commands (e.g., writes, reads, and `DISCARD`) by enabling zero-cost invalidation. To further alleviate queue congestion, `Delay-D` merges adjacent LBAs before issuing a `DISCARD` command (as shown in Line 19 of Algorithm 1). For example, if a range from LBA 100 to 150 is reclaimed, following the prior reclamation of ranges from LBA 50 to 99 and LBA 151 to 200, then `Delay-D` combines these three ranges into a single `DISCARD` command. It is important to note that `Delay-D` offers unprecedented opportunities to perform wide-range merges by accumulating a sufficient number of

reclaimed LBAs in the in-memory mapping table along with the delay operation. To accelerate the performance of such merge operations, `Delay-D` maintains active LBAs in the mapping table in a sequential way.

Meanwhile, `Delay-D` sequentially transfers a set of merged `DISCARD` commands in the order of the start LBA for flash-friendly invalidation. In general, sequential writing is well-known as a flash-friendly pattern because it significantly reduces the execution time of GC operations. Likewise, when processing the `DISCARD` commands, sequential access to LBAs helps to reduce the invalidation overhead due to the cached mapping table within the storage device. This is because the storage device manages its mapping table based on the locality of access history to accelerate the latency of commands, such as read, write, and `DISCARD`; sequential accesses optimize the spatial locality of the mapping table.

In summary, `Delay-D` is designed to alleviate undesirable drops in lifespan and performance when storing captured images of overflight on UAVs. It offers a simple and straightforward solution by delaying the transfer of the `DISCARD` commands, coupled with some optimizations.

4. Evaluation

For the evaluation, we prototyped `Delay-D` by modifying the kernel code on Linux Kernel 5.10.127. To easily enable `Delay-D`, we implemented `Delay-D` in the file system layer; it can be enabled with the `Delay-D` option when the file system is mounted. This work was conducted using a background daemon that utilizes a tree-structured index to track `DISCARD` commands. It is shared with the journal daemon in the kernel to prevent the invalidation of consistent data. To confirm the effectiveness of `Delay-D`, we experimentally evaluated it with two different `DISCARD` options, using two experimental setups involving different NAND-based storage devices.

4.1. Experimental Setup

Since the internal behaviors and information of commercial NAND flash-based storage devices are not publicly available, we utilized two experimental setups: (1) a FEMU-emulated device [16] and two commercial devices. We also configured `Delay-D` to transfer `DISCARD` commands after completing each commit transaction of the Ext4 file system (i.e., the delay time of `Delay-D` is almost 5–6 s). In general, the journaling daemon (i.e., JBD2) triggers a commit transaction every 5 s to ensure crash consistency. `Delay-D` avoids unintended invalidation of committed data by issuing `DISCARD` commands after the completion of the commit transaction.

The first experimental setup is built on FEMU [16] which emulates all kinds of features on NAND flash-based storage devices; it provides internal information and shows how the storage device internally works. We configured a 32 GB NVMe SSD with 4 GB of over-provisioning space (12.5%), where the size of each flash page is set to 4 KiB and the latencies for page read, page write, page discard, and block erase are set to 40 µs, 200 µs, 200 µs, and 2 ms, respectively. In an effort to obtain realistic results, we repeated the same evaluation on the second setup, which involved two commercial SSDs: a Samsung 980 Pro (250 GB) with an NVMe interface and a Samsung 860 Pro (256 GB) with a SATA interface [21,22]. Table 1 lists the detailed specifications of these two commercial SSDs.

Table 1. Specifications of two commercial SSDs [21,22].

Type	Samsung 980 Pro	Samsung 860 Pro
Interface	PCIe Gen4	SATA3
Capacity	250 GB	256 GB
Memory	V-NAND 3-bit MLC	V-NAND 2bit MLC
Sequential read speed	Up to 6400 MB/s	Up to 560 MB/s
Sequential write speed	Up to 2700 MB/s	Up to 530 MB/s
Warranty	5-year or 150 TBW	5 Years or 300 TBW

On the other hand, to obtain general results, we adopted the Ext4 file system, which is one of the most common file systems and is widely used in embedded systems including UAVs, IoT devices, etc. We compare `Delay-D` with two different `DISCARD` options on the Ext4 file system:

- **Disabled**: This is the default option of the Ext4 file system, and it never issues a `DISCARD` command, even when a file is removed by a user.
- **Enabled**: This transfers `DISCARD` commands to signal to the underlying storage device that LBAs belonging to the removed file are no longer valid.

Finally, we implemented a simulator to replay two realistic scenarios for UAVs: batch-erase and single-erase. In the batch-erase scenario, 10,000 files, each 1 MiB in size, are sequentially created and then removed all at once; this process of creation and deletion is repeated 50 times. The single-erase scenario also involves creating a total of 10,000 files of 1 MiB size at once, and then removing the files one by one in the order they were created to reclaim free space before creating a new file of 1 MiB size. In this scenario, the deletion and creation processes are repeated 500,000 times. To present effective results in terms of lifespan and performance, we filled the storage devices to 50% utilization with dummy files before running each evaluation. To represent lifespan, we measured the extra writes caused by internal copy operations during garbage collection in FEMU; these extra writes are unintended and negatively affect the lifespan of the storage device. For performance representation, we measured the start and end times of execution when running the batch-erase and single-erase scenarios, respectively.

4.2. FEMU-Emulated Device

To clearly confirm the lifespan that depends on the total number of writes, we first compare `Delay-D` on the FEMU-emulated device [16]. Figure 3 illustrates the number of copy operations caused by extra writes during GC operations.

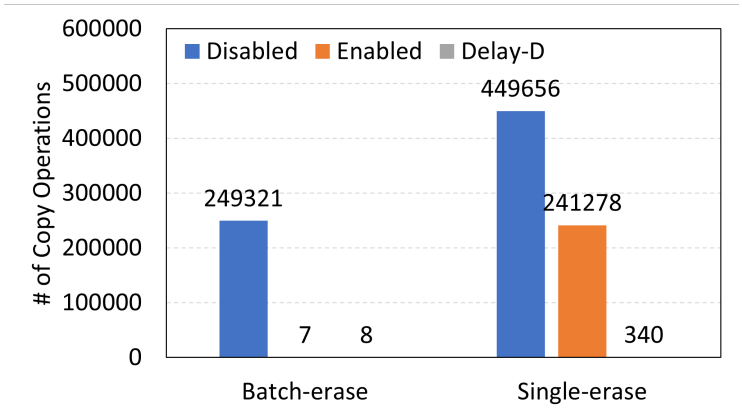
As shown in Figure 3a, the result of `Delay-D` is similar to that of the enabled OPTION in the batch-erase scenario, indicating an optimized result. Meanwhile, in the single-erase scenario, `Delay-D` significantly reduces the number of copy operations by up to 1322× and 709× compared to the Disabled and enabled options, respectively. This reduction is meaningful because the number of write operations can directly impact the lifespan of storage devices. In other words, we can use UAVs equipped with NAND-based storage devices for a long time. To understand such an improvement, we monitored the number of `DISCARD` commands transferred to the FEMU-emulated device.

Table 2 lists the number of issued `DISCARD` commands with the merged counts. As expected, the Disabled option never transfers the `DISCARD` command in both the batch-erase and single-erase scenarios, while the Enabled option issues many `DISCARD` commands. Note that a single `DISCARD` command can include a large range of reclaimed LBAs, thus providing sufficient counts to handle removed files in the evaluation. As shown in Table 2, `Delay-D` significantly reduces the number of `DISCARD` commands by up to 4.1× compared with the Enabled option. This is possible as `Delay-D` merges 3163, 3163, and 3627 `DISCARD` commands for the batch-erase and single-erase scenarios, respectively. Interestingly, even

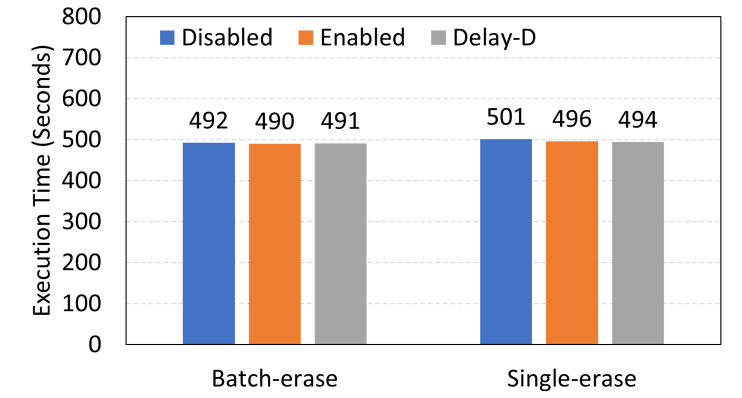
though the number of DISCARD commands is significantly reduced, Delay-D closely matches the performance of the other two options (as seen in Figure 3b). We will describe the reason in the later section in detail.

Table 2. The number of DISCARD commands and merged counts on the FEMU-emulated device.

Type	Description	Disabled	Enabled	**Delay-D**
Batch-erase	The number of DISCARD commands	0	4623	1448
	The number of merged commands	0	0	3163
Single-erase	The number of DISCARD commands	0	4660	1127
	The number of merged commands	0	0	3627



(**a**) Lifespan Impact



(**b**) Performance Impact

Figure 3. The lifespan and performance evaluation results on the FEMU-emulated device. Lower is better.

4.3. Commercial NAND Flash-Based Storage Devices

To realistically compare the performance of Delay-D, we repeated the above evaluation on the second experimental setup.

Figures 4 and 5 show how Delay-D enhances performance on two commercial storage devices. As expected, Delay-D significantly improves the overall performance of NVMe SSD by up to 27% and 2.1× for the batch-erase and single-erase scenarios, respectively (see Figure 4). These results are attributable to the zero-cost invalidation and are meaningful for two reasons. First, as shown in Figure 3a, Delay-D can extend the lifespan of the storage device by reducing the number of extra copy operations. As a result, Delay-D can improve both the lifespan and performance of state-of-the-art storage devices. Second, these results come from high-performance storage devices that are considered successors

to SATA SSDs. To understand the performance advantages in detail, we list the number of `DISCARD` commands and merged counts on the NVMe SSD (see Table 3). As shown in Table 3, `Delay-D` mitigates the congestion of the queue during data transfers by reducing the number of issued `DISCARD` commands; `Delay-D` alleviates the congestion by up to 6.2× and 9× for the batch-erase and single-erase scenarios, respectively.

Meanwhile, in SATA SSD, `Delay-D` shows a small performance gap compared to NVMe SSD (see Figure 5). To understand the results, we analyzed the performance metrics by comparing them with the results of FEMU, as they exhibit similar patterns. We discovered that the longer latency of write and synchronous operations in SATA SSD obscures all performance benefits. In other words, the same evaluation takes more time on a SATA SSD compared to an NVMe SSD. Interestingly, Figure 5 shows that `Delay-D` enhances overall performance by up to 4.8% in the batch-erase scenario. To delve into the details, we present the number of `DISCARD` commands and merged counts on the SATA SSD in Table 4. As shown in Table 4, the total number of `DISCARD` commands is slightly larger than the list in Table 3. However, this small difference indicates that the same evaluation was conducted for a longer time due to the longer latency of operations, as mentioned before. Meanwhile, in the single-erase scenario, both Enabled and `Delay-D` show longer execution times than Disabled. The reason is that the SATA interface uses just a single queue between the host and device, and `DISCARD` commands issued by both mechanisms interfere with the foreground I/O operations.

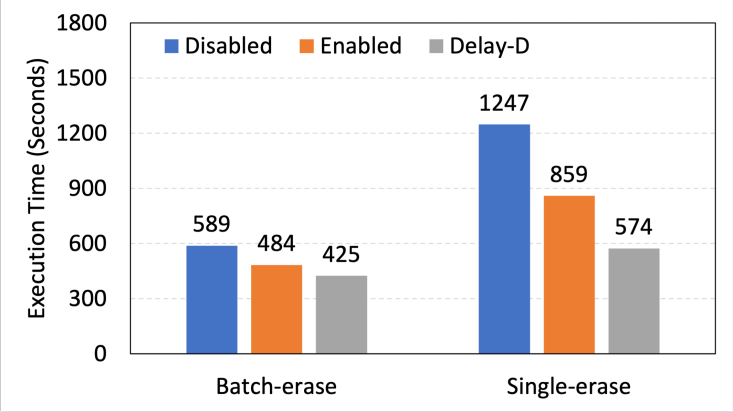


Figure 4. The evaluation results of the overall performance on NVMe SSD. Lower is better.

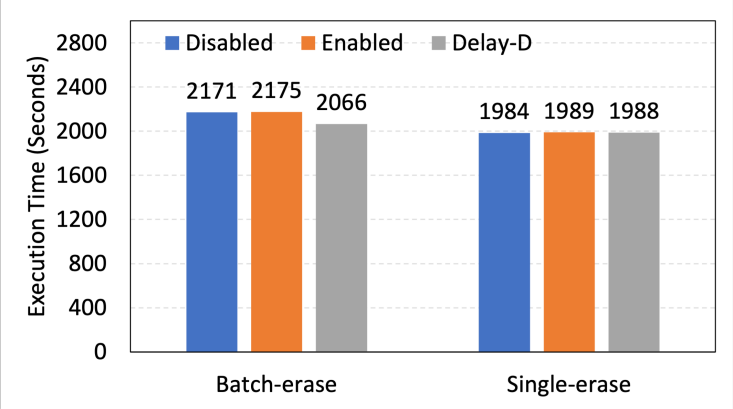


Figure 5. The evaluation results of the overall performance on SATA SSD. Lower is better.

Table 3. The number of `DISCARD` commands and merged counts on NVMe SSD.

Type	Description	Disabled	Enabled	**`Delay-D`**
Batch-erase	The number of `DISCARD` commands	0	3217	515
	The number of merged commands	0	0	2590
Single-erase	The number of `DISCARD` commands	0	8492	943
	The number of merged commands	0	0	7326

Table 4. The number of `DISCARD` commands and merged counts on SATA SSD.

Type	Description	Disabled	Enabled	**`Delay-D`**
Batch-erase	The number of `DISCARD` commands	0	3437	670
	The number of merged commands	0	0	2580
Single-erase	The number of `DISCARD` commands	0	9858	2347
	The number of merged commands	0	0	7341

In summary, `Delay-D` can surely postpone the lifespan of storage devices, but its performance is contingent on the hardware architecture and interface. We believe the overhead of `Delay-D` can be considered lightweight and negligible for two reasons. First, since the technology of storage interfaces is rapidly evolving and becoming more powerful, the interference effect of `DISCARD` commands can be minimized. Second, as shown in Figure 3a, `Delay-D` promises to significantly improve the lifespan of the storage device by reducing write operations, potentially resulting in shorter tail latency. Therefore, we believe the overhead of `Delay-D` is negligible.

5. Related Work

5.1. Previous Research Trends and Efforts on UAVs

UAVs are evolving rapidly due to their diverse applications and advancements in technology. Therefore, numerous previous studies and commercial products have been designed for UAVs in recent years [1–5,23]. For example, Lee et al. introduced the portability issues of tightly coupled software on UAVs and proposed a new software structure for UAV flight control software [23]. Abbas et al. surveyed the usage of UAVs in smart cities [24]. Meanwhile, many research efforts have been designed to utilize images of overflight on UAVs. For example, Du et al. proposed a synchronizing scheme that manages the positions of multiple UAVs based on captured images [5]. Some scientists are focusing on deep learning technologies to provide intelligent services [6–8]. In particular, common research trends involving UAVs or their data revolve around automatically detecting objects (i.e., humans, cars, ships, or animals), based on the captured image datasets from UAVs [7,8]. To accomplish this, UAVs should store huge amounts of image data on the underlying storage device with flight logs, which may lead to a rapid decrease in the endurance of NAND-based storage devices. Nowadays, some engineers are focusing on high-performance storage devices, like SD Express, to capture high-resolution drone overflight images, and are attempting to redesign some functionalities to attain faster performance [25]. We believe that this paper extends the trajectory of this research in that `Delay-D` targets the storage devices on UAVs, improving the performance and lifespan of storage devices.

5.2. Limitations of NAND Flash-Based Storage Devices

To our knowledge, NAND flash-based storage devices have long been used to improve performance in modern computer systems, from servers to desktops. Almost all embedded systems, including UAVs, IoT devices, and smartphones, adopt NAND flash-based storage devices (e.g., eMMC, UFS, and SD Cards) as their primary storage media. Unfortunately, storage devices have some limitations, e.g., asymmetric read/write latency, performance collapse caused by garbage collection, and a limited lifespan. Many researchers have focused on the lifespan of storage devices because it directly affects their performance.

For example, some researchers attempted to extend lifespan by manipulating different operation voltages and times for erase operations [11]. Jimenez et al. enhanced the longevity of storage devices by placing more stress on strong flash cells compared to the weakest ones [26]. Lee et al. proposed the throttling technique to solve unpredictable lifetime problems in enterprise server environments [10]. Meanwhile, some research groups have proposed algorithms for buffers to temporarily reduce the number of writes through improved caching efficiency [27–30].

5.3. Emulators of NAND Flash-Based Storage Devices

Finally, many researchers have focused on designing SSD emulators that can provide various internal data and behaviors in detail [16,31,32]. Yoo et al. designed a VSSIM emulator to study the internal architecture of SSD in 2013, and Zuolo et al. enhanced VSSIM to deliver detailed information, such as performance breakdown curves [31,32]. In 2018, FEMU was based on virtualization; it is widely used because it presents very similar performance metrics to commercial SSDs [16]. In other words, it is the latest and most stable version among the proposed emulators. Therefore, we employed FEMU as the base emulator in this paper.

6. Conclusions

UAVs are commonly used to capture overflight images. We initiated a study on the impact of the image capture process on UAVs in terms of the lifespan and performance of NAND flash-based storage devices. Surprisingly, we discovered that the write pattern resulting from image capture can be detrimental to the underlying storage device. In general, since commercial UAVs seldom replace their primary storage devices, this pattern tends to worsen the lifespan and performance of storage devices on UAVs over time. In this paper, we presented a new lifespan and performance-saving mechanism, called `Delay-D`, which significantly reduces the internal writes caused by the copy operations during GC operations, enhancing overall performance alongside some optimization schemes. Importantly, `Delay-D` can provide file systems with more opportunities to save in the context of write patterns on UAVs. In order to confirm the effectiveness of `Delay-D`, we conducted a series of experiments with two different experiment environments: FEMU-emulated devices and two different types of SSDs. As a result, we can confirm the dramatic reduction in the internal extra writes during GC operations, by up to 1322×. We believe that such a reduction can postpone the lifespan of the storage devices. In addition, `Delay-D` improves the overall performance by up to 2.1× on commercial NVMe SSDs. In summary, `Delay-D` opens up possibilities to support high-resolution video capturing on UAVs with high-performance and long-lasting storage devices.

Meanwhile, it is hard to port `Delay-D` in real UAVs because manufacturers of commercial UAVs do not open-source their operating systems. However, in future work, we will try to adopt `Delay-D` on real UAVs because we believe we anticipate that lifespan and performance issues with UAVs will become more significant over time. Also, we believe it will be worthwhile to delve into the storage patterns in detail when capturing images during overflights and removing them, as will be explored in our future research.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2023-00251730).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

UAV	unmanned aerial vehicle
DD	delayed discard
GPS	global positioning system
DL	deep learning
eMMC	embedded multimedia card
UFS	universal flash storage
SD card	secure digital card
SSD	solid-state drive
HDD	hard disk drive
GC	garbage collection
FTL	flash translation layer
LBA	logical block address
PBA	physical block address
WAF	write amplification factor
OS	operating system

References

1. Choi, J.; Kim, H.M.; Hwang, H.J.; Kim, Y.D.; Kim, C.O. Modular Reinforcement Learning for Autonomous UAV Flight Control. *Drones* **2023**, *7*, 418. [CrossRef]
2. Nikhil, N.; Shreyas, S.M.; Vyshnavi, G.; Yadav, S. Unmanned Aerial Vehicles (UAV) in Disaster Management Applications. In Proceedings of the 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 20–22 August 2020; pp. 140–148.
3. The Capability of Unmanned Aerial Vehicles. Available online: https://www.oilspillresponse.com/knowledge-hub/surveillance/the-capability-of-unmanned-aerial-vehicles-uav/ (accessed on 23 November 2023).
4. Ameli, Z.; Aremanda, Y.; Friess, W.A.; Landis, E.N. Impact of UAV Hardware Options on Bridge Inspection Mission Capabilities. *Drones* **2022**, *6*, 64. [CrossRef]
5. Du, M.; Zou, H.; Wang, T.; Zhu, K. A Cooperative Target Localization Method Based on UAV Aerial Images. *Aerospace* **2023**, *10*, 943. [CrossRef]
6. Song, A. Deep Learning-Based Semantic Segmentation of Urban Areas Using Heterogeneous Unmanned Aerial Vehicle Datasets. *Aerospace* **2023**, *10*, 880. [CrossRef]
7. Tan, L.; Liu, Y. Object Detection and Counting in UAV Images Based on Deep Learning. In Proceedings of the 2023 IEEE 7th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 4–6 March 2023; pp. 1616–1620.
8. Zhu, P.; Wen, L.; Du, D.; Bian, X.; Fan, H.; Hu, Q.; Ling, H. Detection and Tracking Meet Drones Challenge. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 7380–7399. [CrossRef] [PubMed]
9. Telli, K.; Kraa, O.; Himeur, Y.; Ouamane, A.; Boumehraz, M.; Atalla, S.; Mansoor, W. A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs). *Systems* **2023**, *11*, 400. [CrossRef]
10. Lee, S.; Kim, T.; Kim, K.; Kim, J. Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST '12), San Jose, CA, USA, 15–17 February 2012; pp. 1–14.
11. Jeong, J.; Hahn, S.S.; Lee, S.; Kim, J. Lifetime Improvement of NAND Flash-based Storage Systems Using Dynamic Program and Erase Scaling. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST '14), Santa Clara, CA, USA, 17–20 February 2014; pp. 61–74.
12. Park, J.; Jeong, J.; Lee, S.; Song, Y.; Kim, J. Improving Performance and Lifetime of NAND Storage Systems Using Relaxed Program Sequence. In Proceedings of the Design Automation Conference (DAC'16), Austin, TX, USA, 5–9 June 2016; pp. 1–6.
13. Hong, D.; Kim, M.; Cho, G.; Lee, D.; Kim, J. GuardedErase: Extending SSD Lifetimes by Protecting Weak Wordlines. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST '22), Santa Clara, CA, USA, 22–24 February 2022; pp. 133–146.
14. Bekkouche, O.; Samdanis, K.; Bagaa, M.; Taleb, T. A Service-Based Architecture for Enabling UAV Enhanced Network Services. *IEEE Netw.* **2020**, *34*, 328–335. [CrossRef]
15. Siewert, S.; Rocha, K.; Butcher, T.; Pederson, T. Comparison of Common Instrument Stack Architectures for Small UAS and CubeSats. In Proceedings of the 2021 IEEE Aerospace Conference, Big Sky, MT, USA, 6–13 March 2021; pp. 1–17.
16. Li, H.; Hao, M.; Tong, M.H.; Sundararaman, S.; Bjørling, M.; Gunawi, H.S. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST 18), Oakland, CA, USA, 12–15 February 2018; pp. 83–90.

17. Li, J.; Zhao, K.; Zhang, X.; Ma, J.; Zhao, M.; Zhang, T. How Much Can Data Compressibility Help to Improve NAND Flash Memory Lifetime? In Proceedings of the USENIX Conference on File and Storage Technologies (FAST '15), Santa Clara, CA, USA, 16–19 February 2015; pp. 227–240.
18. Zhang, Z.; Zhu, L. A Review on Unmanned Aerial Vehicle Remote Sensing: Platforms, Sensors, Data Processing Methods, and Applications. *Drones* **2022**, *7*, 398. [CrossRef]
19. Khatiri, S.; Panichella, S.; Tonella, P. Simulation-based Test Case Generation for Unmanned Aerial Vehicles in the Neighborhood of Real Flights. In Proceedings of the 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), Dublin, Ireland, 16–20 April 2023; pp. 1–12.
20. VisDrone-Dataset. Available online: https://github.com/VisDrone/VisDrone-Dataset (accessed on 23 November 2023).
21. Specifications of Samsung 860 Pro. Available online: https://semiconductor.samsung.com/consumer-storage/internal-ssd/860pro/ (accessed on 24 December 2023).
22. Specifications of Samsung 980 Pro. Available online: https://semiconductor.samsung.com/consumer-storage/internal-ssd/980pro/ (accessed on 24 December 2023).
23. Jargalsaikhan, T.; Lee, K.; Jun, Y.K.; Lee, S. Architectural Process for Flight Control Software of Unmanned Aerial Vehicle with Module-Level Portability. *Aerospace* **2022**, *9*, 62. [CrossRef]
24. Abbas, N.; Abbas, Z.; Liu, X.; Khan, S.S.; Foster, E.D.; Larkin, S. A Survey: Future Smart Cities Based on Advance Control of Unmanned Aerial Vehicles. *Appl. Sci.* **2023**, *13*, 9881. [CrossRef]
25. SD Express Takes Drones to New Heights. Available online: https://www.sdcard.org/press/thoughtleadership/sd-express-takes-drones-to-new-heights-2/ (accessed on 30 December 2023).
26. Jimenez, X.; Novo, D.; Ienne, P. Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST '14), Santa Clara, CA, USA, 17–20 February 2014; pp. 47–59.
27. Kang, D.H.; Min, C.; Eom, Y.I. An Efficient Buffer Replacement Algorithm for NAND Flash Storage Devices. In Proceedings of the International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'14), Paris, France, 9–11 September 2014; pp. 239–248.
28. Kim, H.; Ryu, M.; Ramachandran, U. What is a good buffer cache replacement scheme for mobile flash storage? *ACM Sigmetrics Perform. Eval. Rev.* **2012**, *20*, 235–246. [CrossRef]
29. Jo, H.; Kang, J.U.; Park, S.Y.; Kim, J.S.; Lee, J. FAB: Flash-aware buffer management policy for portable media players. *IEEE Trans. Consum. Electron.* **2006**, *52*, 485–493.
30. Kim, H.; Ahn, S. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'08), San Jose, CA, USA, 26–29 February 2008; pp. 239–252.
31. Zuolo, L.; Zambelli, C.; Micheloni, R.; Indaco, M.; Carlo, S.D.; Prinetto, P.; Bertozzi, D.; Olivo, P. SSDExplorer: A Virtual Platform for Performance/Reliability-Oriented Fine-Grained Design Space Exploration of Solid State Drives. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1627–1638. [CrossRef]
32. Yoo, J.; Won, Y.; Hwang, J.; Kang, S.; Choi, J.; Yoon, S.; Cha, J. VSSIM: Virtual machine based SSD simulator. In Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), Long Beach, CA, USA, 6–10 May 2013; pp. 1–14.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.