electronics

MDPI

Article

FTRM: A Cache-Based Fault Tolerant Recovery Mechanism for Multi-Channel Flash Devices

Ronnie Mativenga [1], Prince Hamandawana [1], Tae-Sun Chung [1] and Jongik Kim [2,*]

[1] Department of Computer Engineering, Ajou University, Suwon 16499, Korea; ronniematie@ajou.ac.kr (R.M.); phamandawana@ajou.ac.kr (P.H.); tschung@ajou.ac.kr (T.-S.C.)
[2] Division of Computer Science & Engineering, Jeonbuk National University, Jeonju 54896, Korea
* Correspondence: jongik@jbnu.ac.kr

Received: 6 September 2020; Accepted: 23 September 2020; Published: 27 September 2020

Abstract: Flash memory prevalence has reached greater extents with its performance and compactness capabilities. This enables it to be easily adopted as storage media in various portable devices which includes smart watches, cell-phones, drones, and in-vehicle infotainment systems to mention but a few. To support large flash storage in such portable devices, existing flash translation layers (FTLs) employ a cache mapping table (CMT), which contains a small portion of logical page number to physical page number (LPN-PPN) mappings. For robustness, it is of importance to consider the CMT reconstruction mechanisms during system recovery. Currently, existing approaches cannot overcome the performance penalty after experiencing unexpected power failure. This is due to the disregard of the delay caused by inconsistencies between the cached page-mapping entries in RAM and their corresponding mapping pages in flash storage. Furthermore, how to select proper pages for reconstructing the CMT when rebooting a device needs to be revisited. In this study we address these problems and propose a fault tolerant power-failure recovery mechanism (*FTRM*) for flash memory storage systems. Our empirical study shows that *FTRM* is an efficient recovery and robust protocol.

Keywords: cache memory; flash memory; fast boot; fault tolerance; multiple channel

1. Introduction

In computer systems, storage still falls far behind network bandwidth and microprocessor (CPU) speeds that continually double annually [1,2]. The current portable device comes equipped with huge storage capabilities, for example, the recently released Galaxy S10 5G, iPhone XS Max cellphones, and 3D flash memory solid state drive (FSSD) are provisioned with up to 512 GB of flash storage. In these devices, the page-mapping flash translation layer (FTL) provides the best FTL in terms of performance in two-way memory hierarchy systems. However, as the flash storage size scales up, the loading and off-loading of the entire image of mappings from flash to Dynamic Random Access Memory (DRAM) becomes impossible because of limited space and price constraints that comes with DRAM.

To alleviate the problem, a small portion of mapping entries, which are workload dependent, are loaded into the cached mapping table (CMT) in DRAM to take advantage of locality in the particular workload [3,4]. This is commonly referred to as on-demand-based selective-page-mapping scheme. Problems arise when there is a page-update. Updating a page causes an out-of-place write, which changes the physical location of the page. Hence, the mapping entry for the page needs to be updated on both the flash and CMT. However, updating the mapping entry on flash causes another out-of-place write, which degrades the performance of a flash device. To address the problem, existing solutions [3,4] defer the update of the mapping entry on flash until the entry is selected for

eviction from CMT. However, existing solutions suffer from the inconsistency between CMT and mapping entries on flash in case of a sudden failure of a flash device such as power-off. To address the problem, several techniques have been proposed. Map-block [5] algorithm proposed a technique that solves rebooting issues, Per-block [6] proposed using super-blocks to address fast recovery, and log-based approach [7] proposed yet another technique for improving valid and invalidated page location. However, they all suffer from duplicate write operations and lack cache reconstruction techniques that our proposed fault tolerant power-failure recovery mechanism *FTRM* technique offers. The comparison of *FTRM* with existing techniques is summarized in Table 1.

Table 1. A qualitative comparison on existing alternative approaches with our proposed fault tolerant power-failure recovery mechanism (*FTRM*) approach.

Method	Scheme	Translation	Recovery	Cache Reconstruction	Wear Leveling
FTRM	Page Mapping	Fast	Fast	Relevant Pages	Supported
Super-Block	Block Mapping	Slow	N/A	N/A	Not Supported
Translation Block	Page Mapping	Fast	Slow	Random Access	Not Supported

Since an FTL must maintain an up-to-date address mapping table on flash before power-off in order to reconstruct the mapping tables at boot time. During system initialization, FTL should reconstruct the mapping tables in CMT [8]. Therefore, any inconsistencies between page mappings in CMT and flash from the aforementioned page-mapping FTLs and how to reconstruct CMT, is an issue of concern.

In this paper, we propose a fault tolerant fast recovery approach (*FTRM*) that enables on-demand-page-level-mapping FTLs to quickly and efficiently reconstruct consistent up-to-date CMTs in DRAM during boot. Our approach takes advantage of the out-of-band (OOB) region of flash pages for the page mapping table to keep track of the number of accesses on every page through an access counter (*AC*) and a validation flag (*VF*) for tracking page updates (valid or invalidated pages). The *AC* allows us to efficiently separate hot from cold flash pages for quicker CMT loading and offloading while the *VF* is used for pointing to the location of valid data pages to be used for mapping page updates. We also provision an optional Write Counter (WC) that is used to track the number of page program cycles for the purposes of wear leveling. In this scenario, all pages that reach a certain write threshold are moved from a hot region into a cold region of the flash and thus help an even wear of flash pages. This study makes the following contributions:

- We design and implement an efficient fast-wake-up FSSD algorithm, that fuses together with the hot/cold page swapping technique to produce a robust and efficient fault tolerant mechanism for SSDs;
- We further propose a reliable and efficient algorithm for CMT re-construction during system start-up;
- We implement our proposed scheme on the EagleTree simulator whilst emulating Quad-Level-Cell (QLC) based FSSD and then ran our experiments using realistic workloads.

The remainder of this paper is organized as follows: Section 2 discusses the background of flash memory and our motivational goals are discussed in Section 3. Section 4 explains the architectural designs and implementation of our proposed approach. Section 5 presents our experimental setup and evaluation results. We discuss the previous works in Section 6 and then conclude in Section 7.

2. Preliminaries

Table 2 shows the important acronyms that we are going to use frequently throughout the paper. Non volatile (NAND) flash memory consists of planes that are composed of several dies. As illustrated in Figure 1, each die contains multiple blocks and at this level erase operations are performed. Within a block, there are multiple pages, which are the granularity at which read and write operations are

performed [4]. A page is further divided into data area and a small spare area (typically 1/32 of the data size) called out-of-band (OOB). The OOB of a page is used for storing the page state information and the page's logical page number (LPN). The OOB also contains error correction code (ECC) that is used to check the correctness of the data stored on the page's data area [3,9] as shown in Figure 1. During a read operation, the entire page is read together with the OOB area while a write operation can be done selectively on either the OOB or the data area.

To efficiently mimic traditional the hard disk drive (HDD), SSDs employing flash memory have to appear on a block device to the overlaying file system while at the same time hiding the peculiarities of NAND flash. Therefore, an FTL is used for the translation of read/writes from the file system into the flash. The data structures and mapping tables manipulated by the FTL are the fast but small DRAM. Basically, there are two forms of FTL designs that is based in the contents of their mapping tables, which are the page-level and block-level FTL schemes. The page-level FTL scheme employs a fully associative cache [10] in that the request's LPN can be mapped to any page on the flash. This efficiently utilizes flash blocks even though its downside is the requirement of a huge mapping table to be cached in DRAM. For example, a 512 GB flash memory would require around 1 GB of DRAM space to store the page-level mapping table. This becomes infeasible to have DRAM that can scale with the increasing flash size because of the limited price/MB cost of DRAM [4]. Conversely, block-level FTL schemes translates its logical block number (LBN) to physical block number (PBN) through a set associative mapping table manner [10] with a fixed LPN offset in a block. Even though a block-level mapping table size is greatly reduced by a factor of block-size/page-size as compared to page-level FTL, it suffers from performance degradation. This is due to the fact that, a logical page should be placed onto a particular page on flash which greatly reduces the chances of finding that page [3].

To address the shortcomings of both block-level and page-level FTL schemes, a variety of alternatives like hybrid FTLs [5,11,12] and selective-page-level FTLs [3,4] exist. The hybrid FTLs are a combination of both page-level and block-level FTLs and partitions the flash blocks into data-blocks (i.e., mapped at block-level) and log-blocks (i.e., mapped at page-level). On the other hand, the selective-page-level FTLs takes advantage of a page-level mapping scheme that allows a request to be serviced from any physical page on the flash. Moreover, it exploits the temporal locality workload characteristics to load and unload a portion of mappings from flash to the space limited DRAM called CMT [4]. These schemes also separates flash into data-pages and mapping-pages. The data pages store real data accessed during read/write operations while the mapping pages store the logical-to-physical address mappings. Even though the page mapping FTLs are the most efficient, the selective-page-level-mapping schemes like DFTL [3] and RFTL [4] (i.e. a replication-based DFTL approach), have proven to show good performance. Moreover, they are capable of achieving high I/O parallelism in multichannel Triple-Level-Cell (TLC)-based SSDs owing to their dynamic striping techniques. However, these FTLs suffer from having to maintain a large mapping table in DRAM or on a flash memory array. DFTL, for example, must execute additional read/write operations to read/update the mapping information stored in flash memory.

Table 2. Acronyms used in the article.

ACRONYM	ACRONYM DEFINITION
PPN	Physical Page Number
LPN	Logical Page Number
CMT	Cache Mapping Table
OOB	Out-of-Band
QLC	Quard Level Cells
ECC	Error Correction Code
AC	Access Counter
VF	Validation Flag
WC	Write Counter
FSSD	Flash Memory Solid State Drive
WCTH	Write Counter Threshold

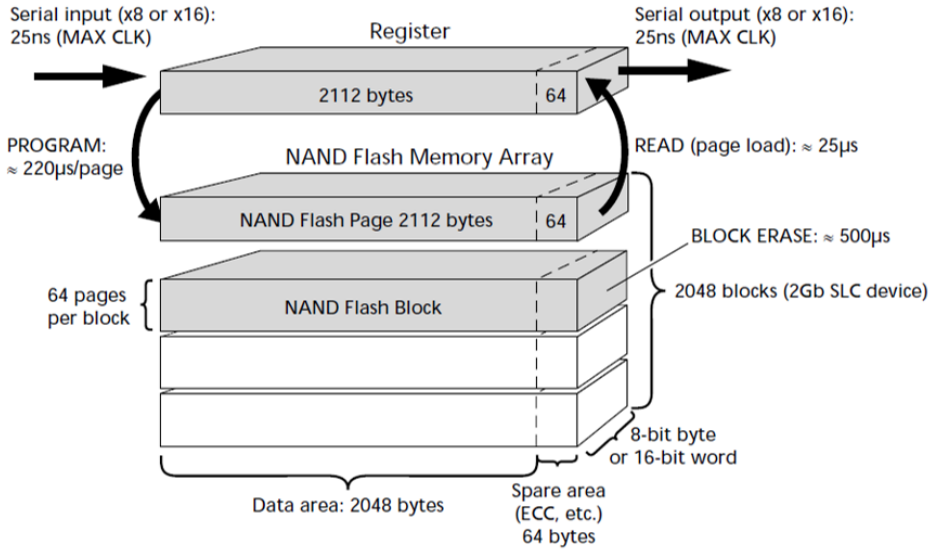

Figure 1. The organizational structure of a NAND flash device. [Source: Micron Technology Inc.].

Thus, a portion of mapping pages are temporarily loaded and offloaded from the mapping table on flash to the CMT depending on the workload locality. Commonly, the flash space is divided into data blocks and mapping blocks, with the data blocks occupying over 80% of the entire space. When a write request arrives, its LPN is used to locate its physical location on flash (PPN). This means that the request's LPN is translated to a PPN or the FTL first checks its corresponding PPN in DRAM for a faster address translation. When located, the PPN then points the controller to the actual entry stored on flash's data page where the actual write operation is done onto a fresh page, i.e., out-of-place-update while the previous page is invalidated. The mapping page in CMT is updated with this new physical data location of the data page on flash while the same mapping page residing on flash's mapping table is neglected so as to reduce the write latency. It is only when this particular mapping page is selected as a victim to be evicted from CMT that its corresponding mapping page on flash is updated. This process is called lazy copying or write-back policy.

Consequently, if the FSSD experiences a sudden power cut, CMT mappings are lost because of the volatile nature of DRAM and during reboot, FTL reconstructs a new CMT by randomly selecting and loading a portion of mappings from the mapping table on flash into the DRAM. This reconstruction disregards whether the candidate mapping pages are up-to-date or not and whether they are hot or cold. This can cause inconsistencies as mapping pages might point a request to an already invalidated physical location of a data page on flash.

3. Motivation

As pointed out in the previous section, mapping page consistency with valid data page location is of importance during system reboot and is an issue that should be addressed. Figure 2 illustrates the problem of existing solutions. Consider we have three page mappings in the CMT. If there is an update request on LPN 7, we first look up the CMT and translate the address to PPN 10. With the PPN, we can locate the physical page on flash allowing for the write operation to process. This LPN 7 write request invokes an out-of-place update operation to an empty physical data page (i.e., to DPPN 91) while the previous physical page location, DPPN 10 is invalidated as illustrated by steps 1 and 2 in Figure 2. To maintain consistency, the same LPN 7 request entry in CMT is also updated with a new PPN-91 as shown between steps 2 and 4 of the same figure. Furthermore, the corresponding CMT entry (i.e., LPN 7) on the mapping table on flash is also supposed to be updated to DPPN 91 which is not the case. This is done to avoid such extra writes to improve flash performance and endurance. As a result, the least-recently-used (LRU) CMT entry, i.e., cold mapping entry is updated on the flash mapping table before it is evicted from the CMT [4].

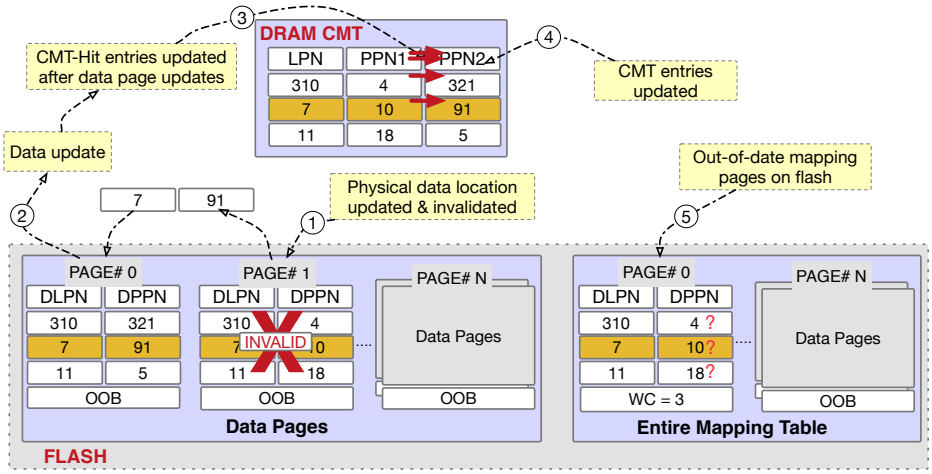

Figure 2. The drawbacks of traditional flash memory solid state drive (FSSD) booting schemes. MLPN/MPPN and DLPN/DPPN refers to the Mapping Pages Logical and Physical Page Numbers and Data Pages Logical and Physical Page Numbers respectively.

During reboot, in Figure 2 (step 5), the outdated mapping pages are reloaded from flash to DRAM for CMT reconstruction causing inconsistencies with current data page locations on the flash resulting in unnecessary page faults/errors. This means that during restart, the system will initialize with an out-dated reconstructed CMT because the mapping entry candidates that were selected and loaded into the CMT were the out-of-date mapping pages residing on flash. Consequently, mapping requests are going to be processed via the outdated CMT and directed to invalidated physical page locations on flash as illustrated in Figure 2. To correct invalidated PPNs, existing approaches scans all mapping pages on the flash but due to DRAM's volatility, an interruption before syncing these CMT entries with the ones on flash is common. Sudden power cuts will result in the loss of up-to-date mapping entry information as only the volatile DRAM-CMT would have been updated whilst the same had not yet been persisted on flash mapping pages.

Furthermore, traditional FSSDs lack an efficient mechanism on how to select candidate mapping entries from the mapping table on flash into DRAM-CMT. Specifically, policies on how the FTL can efficiently select entries for CMT construction and the ones to neglect according to particular workload characteristics are of great concern. Our system proposes a way to select precise mapping entries and a quick boot mechanism to alleviate such issues in traditional FSSDs. As a result, our proposed *FTRM*, for example will always load up-to-date mapping entries into DRAM-CMT thereby reducing page-faults and CMT-miss penalties. Consequently, booting/recovery is enhanced as our fast-boot mechanism uses few pages for CMT reconstruction thereby achieving a faster response time. It also optimizes the I/O parallelism to satisfy the real-time storage and retrieval requirements of current state-of-the-art storage devices.

4. Design and Implementation

This section discusses the design overview and functional goals of our proposed scheme together with how we try to conceal the drawbacks of traditional flash memory devices using our recovery mechanism.

4.1. Design Overview

To overcome the CMT synchronization and reconstruction issues described previously, we propose a fault tolerant and fast recovery approach (*FTRM*). *FTRM* facilitates a quick, efficient, and consistent up-to-date CMT reconstruction in DRAM during boot for SSDs employing on-demand-based selective-page-level-mapping FTLs like DFTL [3] and RFTL [13]. Our approach takes advantage

of the OOB region of flash pages to keep track of the validity of a page via a validation flag (*VF*) and page accesses via an access counter (*AC*). The *AC* allows for an efficient separation of hot from cold mapping pages for a quicker CMT loading and offloading. The optional write counter (*WC*) when enabled allows us to keep track of the program cycles of pages for effective wear-leveling if needed as it keeps a count of all the writes of each page.

The *FTRM* approach aims to improve the overall system recovery time whenever an unexpected power-cut occurs. Furthermore, it facilitates for the overall end-to-end performance improvement of storage media (e.g., SSD) through utilizing the OOB area of physical pages to update mapping entries and to separate hot from cold pages. To ensure data integrity, a regular wake-up procedure or recovery mechanism should be called by the SSD controller each time a sudden power cut occurs. We further design an efficient and reliable way on how to construct the CMT from the mapping pages image from flash. We provide the detailed operations of *VF* and *AC* together with the optional *WC* in the following.

4.2. Operational Process

The schematic diagram of the proposed *FTRM* approach is illustrated in Figure 3. Each page on flash has *WC*, *AC*, and *VF* in its OOB area as depicted in the figure. The *AC*, i.e., step 1 of Figure 3, is incremented each time there is a read/write on that particular page and helps to maintain an even hot/cold page separation that also assists during the CMT construction. The second step, i.e., step 2, of the same figure demonstrates how our *VF* keeps track of the valid and invalidated data pages. Our flash is divided into data pages and mapping pages. Each flash page shows whether a data or a mapping page constitutes a data area and a spare area (OOB) as previously discussed. In Figure 3, *FTRM* keeps track of page access through an *AC* and an optional *WC*. When there is a CMT hit on a read request, the data is fetched from the corresponding data page on flash and its mapping page's *AC* in CMT is updated while the same page on flash is neglected (lazy-copying). The reason being that, if we concurrently update the *AC* (step 1 of Figure 3) in the OOB of the mapping page on flash during this read process, such an extra write operation which will be costly to the ongoing read. Therefore, the *AC*s of mapping pages residing on flash are synced with all the updates that occurred in CMT during a write request operation on the same page and/or when the same page has been targeted as a victim of eviction from the CMT.

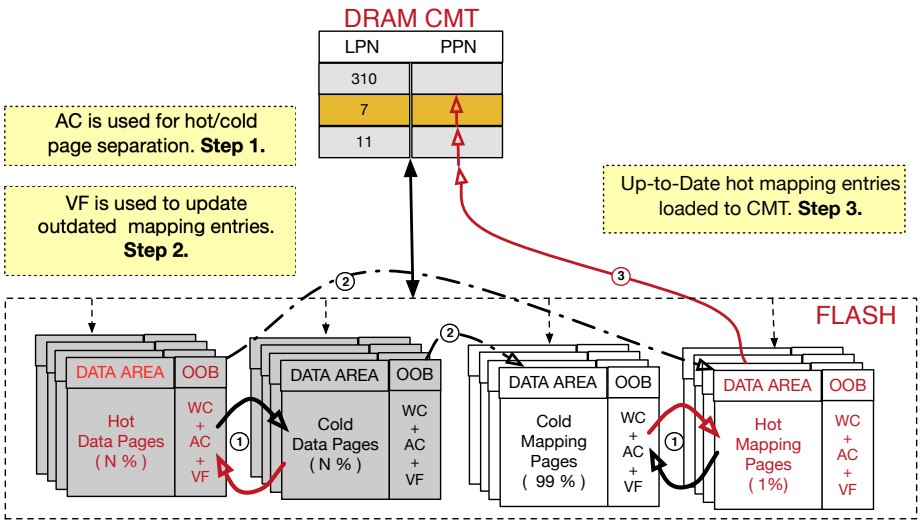

Figure 3. Schematic diagram of the proposed *FTRM* system.

Consequently, we end up having a separation between the frequently accessed mapping pages marked as hot and the rest marked as cold. By doing so, the mapping table on flash becomes divided into hot mapping blocks and cold ones. During reboot, the system selects CMT construction candidates

from this hot flash region. Moreover, the *FTRM* approach keeps track of the valid data pages through the *VF* which is then used to update the corresponding mapping pages residing on flash during reboot in the event of an unexpected power cut as demonstrated by step 2 of Figure 3. Furthermore, the use of the *VF* to update lost CMT mappings and the *AC* for hot and cold mapping-page separation improves the accuracy and loading time of up-to-date +hot CMT entries into DRAM (fast-boot) during system recovery as demonstrated by step 3 of Figure 3. The following subsections will discuss these mentioned processes in more detail.

4.2.1. Hot and Cold Page Identification

Since a flash page consists of 16 (2^4) sectors then, from the 64 Bytes of the OOB region, considering the QLC lifespan, we provisioned our *AC* with 6 bits of the OOB which is around 1% of the total page lifespan, i.e., 5000 program cycles. This can differ depending on the SSD vendor, but in our work we used the SSD structure with 16 sector pages. Our *AC*'s space allocation does not affect the OOB meta-data or ECC that already resides in the OOB but only takes up a small portion of the free space available.

Algorithm 1 outlines hot and cold page identification. For each input page in the request (Line 2), it first checks if the request is a write operation in Line 3. During the write, both corresponding pages' *AC* is updated concurrently with data update operation, i.e., Lines 4–6 in Algorithm 1. During this process, *FTRM* also compares the current page's *AC* value with that set as the threshold, from Lines 7–14 in the algorithm. The page's *AC* has to reach a predefined threshold (TH), i.e., TH + 1 so that it can be marked as hot. In Algorithm 1 Lines 20–23, a mapping page marked as hot remains in that state until it is evicted from CMT where its *AC* is then reset to zero. The reason being that, if an *AC* in DRAM is not written to the OOB of a mapping page on flash whilst the page is cold, we do not need that page during reconstruction. If the page is marked as hot, then its *AC* is already reflected on its OOB (although the *AC* in the cache and that in the OOB may not be consistent). Moreover, our access counter is not affected by wear-leveling operation and does not determine the flash page lifespan. It is the optional *WC* that can be used to determine such. By keeping track of this *AC* and marking pages hot/cold, our page mapping table on the flash array is divided into cold mapping pages on separate blocks with hot mapping pages on the other side.

Conversely, if the request's target page is not a CMT victim, *FTRM* will only update the request entry's CMT *AC* as illustrated in Lines 24 and 25 Algorithm 1. The TH can be set according to developer needs depending on their goal. The pages with a high *AC* are regarded as hot and the rest as cold. There are several ways to keep track of page access. We have chosen to update the *AC* of a page on flash in case that (i) a write operation of the page is issued; (ii) the page is about to be evicted from the CMT; or (iii) the page is in transition from a cold page to a hot one. In other cases, we just update the *AC* of a page in the CMT. This is to maintain the read latency and not add extra delays to it. We remark that the OOB of a page can be updated concurrently with the data area of the same page, that is, updating the *AC* during the page's write operation will not pose any extra latency to the system.

During a CMT-miss, the page mapping for the requested page is fetched from flash and added into the CMT. Our *FTRM* system then sets the *AC* attached to the newly added mapping using the *AC* in the OOB of the requested page after reading the page. In this way, the additional write operation for a read operation is avoided. Since we need an additional OOB write only when a page becomes hot, we can expect that most of the mapping read operations do not need OOB writes. Moreover, we can expect the latency of the additional OOB write to be hidden in the upper layer since it would be rare (depending on how we set the threshold). If we do not write the cached *AC* back to the mapping page's OOB on flash, we lose the up-to-date mapping information. Therefore, if we do not set the *AC* in OOB to zero during CMT eviction, it does not correctly reflect the temporal locality. That is, if we do not reset the *AC* of the page and the page is then reloaded to the CMT, it may start with a high *AC* value. Since the page is the least recently used (LRU) one, we can reflect the temporal locality in our *AC* by abandoning any increments occurred while it is in the CMT. This allows us also to maintain a

low bit *AC* value thus saving already limited OOB space. Thus, the hot and cold pages are separated and the hot ones are then used for CMT reconstruction during system reboot. Since mapping pages constitutes only around 0.2% of total flash space, we can read through the mapping table selectively picking the most recently accessed mapping pages from the hot blocks on flash.

Algorithm 1: Hot and Cold Pages Identification

Input: $Request_{LPN}$ check page access
while $request_{LPN} \neq null$ **do**
  if $request_{LPN} = Write$ **then**
    Increment $request_{AC}$ in DRAM
    Increment $request_{AC}$ on flash page
    `/* Synchronize corresponding flash page AC with the current CMT updates. */`
    if $flash_{OOB}$ $AC > Threshold$ **then**
      $request_{Page} = HOT$
      `/* Mark page as Hot */`
    else
    end
    $flash_{OOB}$ $AC < Threshold$
    $request_{Page} = COLD$
    `/* Mark page as Cold */`
  else
  end
  $request_{LPN}$ = Read
  Increment $request_{AC}$ in DRAM
  `/* Only increament` $request_{AC}$ `on CMT page */`
  if $entry = CMT_{Victim}$ **then**
    flash $OOB_{AC} = 0$
    `/* Reset flash mapping page` OOB_{AC} `counter to zero */`
  else
  end
  entry $\neq CMT_{Victim}$
  Increment $request_{AC}$ in DRAM
end
Output: Hot and Cold Page Separated

4.2.2. Fast Recovery

Our proposed validation flag (*VF*) resides in the OOB area of flash pages where it only occupies a single bit of the OOB space. The *VF* is used to represent the data page state, i.e., when a data page is valid, the flag will indicate 1 and when invalid, a 0 will be reflected. During a write operation, the newly written data page's *VF* is then set to 1 while the previous invalidated page's *VF* is set to 0. Consequently, all those data pages with *VF* = 1 are used for updating the mapping pages on flash memory during system boot whenever up-to-date mappings are lost in DRAM-CMT. This will enable *FTRM* to quickly update our *PPN* for every corresponding *LPN* as indicated in Figure 3. Furthermore, reading a single bit from the OBB becomes really fast and efficient. During system reboot, *FTRM* executes several read operations on specific portions of the flash mapping pages array for CMT construction and uploads these pages into DRAM as illustrated in Algorithm 2. That is, *FTRM* through the validation flag (*VF*), quickly checks whether the physical location of valid data pages corresponds

with the mapping pages on the entire mapping page image on flash as demonstrated by Lines 1–6 of Algorithm 2. The system checks the hot mapping LPNs against the data LPN with $VF = 1$ then syncs the ones that are not up-to-date with the correct information (i.e., Lines 7 in Algorithm 2). It then runs through the marked CTM reconstruction candidate hot mapping pages on flash and loads them to DRAM as shown from Lines 8–11 of Algorithm 2. This quick recovery process is referred to as a fast wake-up and effectively influences flash performance.

Algorithm 2: Fast Recovery

1 **Input:** Request check VF on Data Page
2 **while** $Page_{OOB\text{-}VF} \neq 0$ **do**
3 Check if $LPN \leftarrow PPN$ is correct
4 `/* logical-to-physical address translation of data pages with mapping pages */`
5 $Mapping_{Page} \leftarrow Page_{OOB\text{-}VF}$
6 `/* Locate Corresponding Data Page */`
7 Update $Mapping_{Page}$ on Flash
8 **if** $flash_{OOB\text{-}AC} = HOT$ **then**
9 Select as candidate for CMT construction
10 $CMT \leftarrow Mappings_{Hot}$
11 `/* Load a portion of hot mappings to CMT */`
12 **else**
13 **end**
14 $flash_{OOB\text{-}AC} = COLD$
15 Keep Mapping Page updated
16 `/* Entry is not a candidate for CMT construction */`
17 **end**
18 **Output:** CMT Reconstructed

Unlike in traditional FTLs, our scheme uploads all the page-level mapping information corresponding to hot mappings by first reading the validation flag from the OOB area of the data pages and updating the corresponding mapping entries on flash as previously described. Our *AC* is incremented whenever a data page is accessed be it a read or write operation as-well. The most frequently accessed mapping pages, i.e., hot pages, are then selected and listed as CMT reconstruction candidates and a portion of them is loaded to DRAM-CMT while the rest remain up-to-date on flash as shown in Lines 12–19 of Algorithm 2.

4.2.3. Optional Lifespan Management

By enabling our proposed optional write counter (*WC*) on flash pages, we can easily determine the number of writes each page encounters. We can then use this information for efficient wear-leveling, if needed, so as to improve the overall flash lifespan as demonstrated by Algorithm 3. Here, the system first checks whether the ongoing request is a read or a write. If the request is a write then the system checks whether the *WC* has reached the threshold (*WCTH*) and if so, then the page contents are migrated to a colder page and the previous one is invalidated leading to the update of both CTM and flash mapping entries with the new *PPN* as illustrated by Lines 4–11 of Algorithm 3. If the *WC* of the same page has not yet reached the threshold then it is incremented and current request page contents are written onto a free page while their corresponding mapping page *PPN* is also updated (Lines 12–19 of Algorithm 3). This means that the *WC* is incremented after every successive write operation of that particular page. Conversely, the read operations do not affect our *WC* because read operations do not pose a threat to the flash memory's limited write cycle (lifespan).

Algorithm 3: Page Lifespan Management

```
Input: Request check Write Counter (WC) on Data Page
while request_size ≠ 0 do
    if request = write then
        Check Page Write Count Threshold (WCTH)
        if WC > Threshold (WCTH) then
            PPN++
            /* Migrate Page to next physical address                    */
            WC = 0
            /* RESET writes counter                                     */
            CMT_PPN ← flash_PPN
            /* update mapping page with the new physical data address   */
        else
        end
        WC < WCTH
        WC++
        /* Increment write counter                                      */
        CMT_PPN ← flash_PPN
        /* update mapping page with the new physical data address     */
    else
    end
    request = read
    Maintain current WC + Page Position
    Send contents back to controller
end
Output: Writes Evenly Distributed
```

5. Evaluation

In this Section, we carried out a series of experiments to show the efficacy of our proposed fault tolerant recovery mechanism (*FTRM* for short). To evaluate the effectiveness of our proposed approach, we compared the following:

- *Baseline*: This is the traditional selective-page-mapping FTL (e.g., DFTL);
- *Map-Block scheme* : This is the traditional space efficient FTL and;
- *Our Proposed approach (FTRM)* : This is our proposed fault tolerant, reliable, and fast boot approach.

We further carefully considered the most effective and most efficient way to update our access counter (*AC*) on the mapping page OOB area without causing effect to the read or write operation on flash in-order to keep CMT and flash mapping page *AC* synchronized. This led us to run comparison experiments on the following *FTRM* proposed ways:

- $FTRM_{Base}$: With this approach, the *OOB-AC* of a mapping page is updated whenever a corresponding read/write request operation is ongoing;
- $FTRM_{Victim}$: Here the mapping page's *OOB-AC* is updated when there is a corresponding write operation to the same page or when the same entry has been targeted for CMT eviction and;
- $FTRM_{Write}$: This approach considers the update of *OOB-AC* only when there is a write request corresponding to the same mapping page on flash and resting the *AC* whenever the same page's corresponding entry has been marked as a CMT victim as previously discussed.

5.1. Experimental Setup

By carefully considering the above-mentioned facts, we calculated the number of mapping reads to evaluate the reconstruction efficiency as our comparison matrix that each approach undergoes during system start-up. The fewer the mapping read operations means the more efficient or faster the approach is. Fault tolerance was evaluated based on the number of page faults (as a comparison matrix for checking the robustness of the systems) together with the number of write operations and the total time taken. Therefore, the fewer or absence of such page faults indicates the robustness of the system and consequently the accuracy in reconstructing CMT.

We implemented our proposed approaches on EagleTree Simulator [14]. Our setup was composed of a 128 GB multichannel SSD of 8 KB page sizes, that is a QLC FSSD. Table 3 summarizes the SSD size and access settings that were used during our experiments. The I/O scheduler uses First-In-First-Out (FIFO) and the wear-leveling threshold ratio was set to 0.1% for cell lifespan which can also be varied according to designer needs. We then used realistic workloads [15,16] to run the experiments and they are comprised of MSNFS with 85% read I/Os, Financial1 with 50% write dominant trace, MSR-ts_0 with 82% write traces, and then RADIUS comprising 91% write dominant workload characteristics which are also presented on Table 4.

Table 3. SSD (solid state drive) operational settings.

Parameter	Access Time
DRAM (ns)	100
Page Read Delay	25 µs
Page Write Delay	200 µs
Bus Data Delay	100 µs
Block Erase Delay	1500 µs
Parameter	**Size**
SSD channels (#)	8
Page size	8 KB
Bock size	256 pages
Plane size	2048 blocks
Die size	1 plane
Chip size	4 dies

Table 4. Read/write ratio of workload benchmark traces.

Benchmark	Writes	Reads	Write Size (KB)	Read Size (KB)
MSNFS	15%	85%	10.06	14.38
Finacial1	54%	46%	2.91	2.48
RADIUS	91%	9%	7.20	6.58
MSR-ts_0	82%	18%	8.01	13.68

5.2. Experimental Results

We exposed our *FTRM* approaches to various operational scenarios which had both read and write dominant realistic workloads so that we could evaluate their efficacy. In Figure 4a, $FTRM_{Write}$ and $FTRM_{Victim}$ show more than 90% of the I/Os having a response time below 100 µs while $FTRM_{Base}$ has only below 35% of I/Os with the same response time for mapping reads. This is due to the extra OOB mapping write operation that $FTRM_{Base}$ undergoes during a mapping read so as to continually keep the flash mapping page *AC* up-to-date or in sync with the CMT entry. As a result, the mapping reads ends up taking as long as the write operation unlike its counterparts $FTRM_{Victim}$ and $FTRM_{Write}$ which update their *ACs* only during CMT eviction and or during a write operation. Conversely, when the workload becomes write intensive as witnessed from Figure 5b, we see $FTRM_{Base}$'s number of mapping writes closing in as it manages to process 90% of I/Os with a response time between 250–300 µs which is very close to $FTRM_{Write}$ and $FTRM_{Victim}$'s mapping write performances. The reason being that, because of the write intensity environment, both approaches have more mapping writes than reads. Therefore, all the *FTRM* approaches can have the opportunity to update/synchronize the OOB of

their corresponding mapping pages on flash with those in CMT concurrently with the ongoing data area writes.

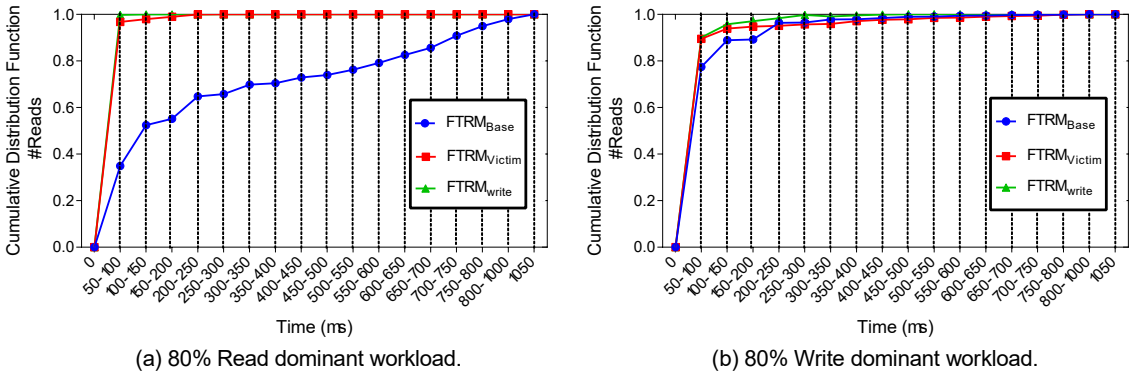

Figure 4. Comparison of Cumulative Distribution Function (CDF) for Mapping Reads I/Os between *FTRM* alternatives.

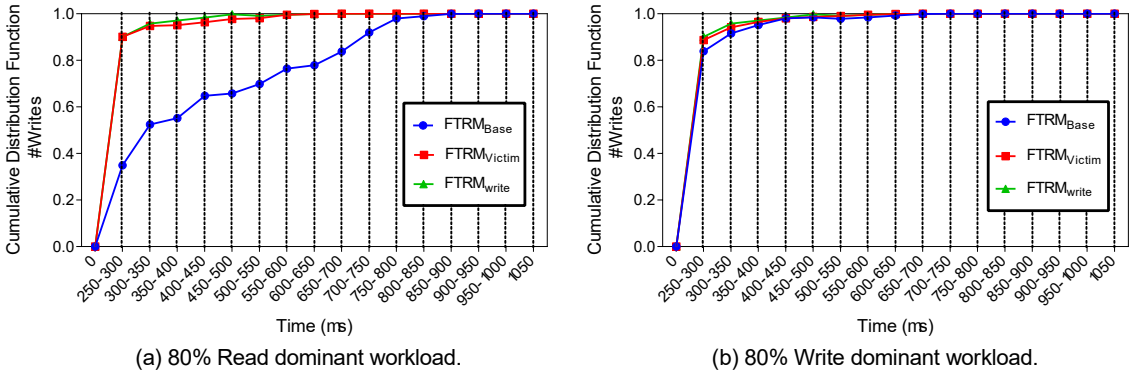

Figure 5. Comparison of Cumulative Distribution Function (CDF) for Mapping Writes I/Os between *FTRM* alternatives.

The outcomes from analyzing a suitable *FTRM* approach from the above-mentioned trio made use to opt for the $FTRM_{Write}$ approach. This is because of the fact that, even though $FTRM_{Base}$ offers the best CMT reconstruction results, being better than both its counterparts, it degrades the mapping read time, consequently reducing end-to-end system performance considerably. Furthermore, these induced extra write operations in-turn poses a great threat to page accesses by other ongoing internal operation, thereby increasing access conflicts. In the following sections, we evaluated our proposed $FTRM_{Write}$ approach against the already existing ones (Baseline and Map-Block). Note that the following *FTRM* approach is a simplified name referring to *FTRM*.

5.2.1. CMT Construction

Figure 6a shows mapping read count comparison between our Baseline, Map-block, and our proposed *FTRM* approach under RADIUS, Financial1, and MSNFS realistic traces. Interestingly, *FTRM* outperforms both Map-block and Baseline by an average of 38% under all traces because it utilizes the *AC* from the data pages' OOB area to separate hot/cold mappings. The system then easily selects CMT construction candidates only from the hot updated mapping pages. Conversely, Map-block approach experiences more reads than the rest because it has to scan/read through all the physical pages scattered across flash chips to able to select the CMT reconstruction entries and then reconstruct.

5.2.2. Fault Tolerance

The results from Figure 6b evidenced that our proposed approach (*FTRM*) can improve a mapping page consistency by over 65% on average, compared to traditional FTLs. This is because *FTRM* updates mapping pages by referencing the *VF* on their corresponding data pages' OOB region on flash regardless of unexpected system power loss. Consequently, not only does this improve performance time (Figure 6c) but also reduces the percentage number of page faults when the CMT starts to process incoming requests soon after construction, thus the reduced number of page faults witnessed from *FTRM*.

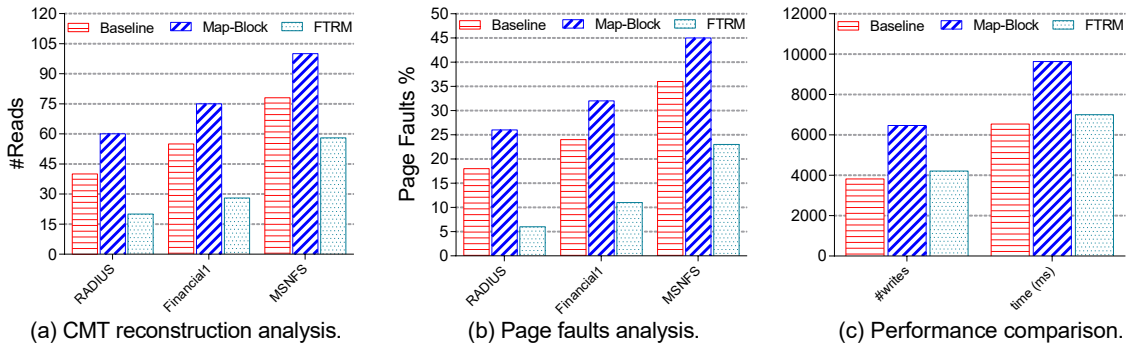

Figure 6. Reliability and efficiency comparison between Baseline, Map-Block, and our proposed *FTRM* approach. (**a**) Reconstruction analysis using mapping reads count; (**b**) percentage number of page faults after system reboot; and (**c**) performance comparison through writes and time analysis.

To evaluate *FTRM*'s performance, we measured and recorded the write counts and the total elapsed time in milliseconds for each approach using the RADIUS workload. We used these traces because they are 91% write dominant while comprising of small-random-write requests. Such types of requests reduce flash performance speed because of the out-of-place-update nature of flash memory. Figure 6c shows the writes count with elapsed time comparison results and we evidenced that the number of writes are directly proportional to time taken. For example, the Map-Block approach experienced over 6450 writes in 9500 ms of time while it took *FTRM* around 7000 ms to process 4200 writes. Baseline has the least writes because it has no recovery mechanism, whilst Map-Block experiences the most writes since it has to duplicate writes to another block. Furthermore, *FTRM's* performance is tightly close to the baseline's because the *WC* on the OOB area of data pages is updated concurrently with the data area during a write operation.

6. Related Works

As discussed in the previous section, power recovery and fault tolerance is of concern in flash memory apart from GC and the slow write/reads operations. Several research works have proposed some fast-wake-up algorithms [3–8]. To resolve rebooting issues, a fast-wake-up method [17] and a map block approach [5] were proposed. These allow all mapping entries to be loaded into DRAM and during boot, the system simply scans all these mappings and loads them into DRAM. The Map block approach has the advantage of 100% guaranteed cache-hits for all incoming requests but such block-mapping schemes have lower performance when compared to page-mapping schemes. This is because page-mapping FTLs facilitates parallelism in current multichannel SSDs. Furthermore, they induce dynamic stripping mechanism thereby speeding up writes as compared to block-mapping which is restricted to a static stripping of physical pages on flash. If a page can be placed anywhere on flash, then the result is lesser access conflicts and faster physical page allocation during writes.

A per-block mapping method using a super-block FTL that employs a hybrid address translation scheme for fast recovery was also proposed [6]. It stores the mapping entries on the OOB region of

each physical page and therefore unlike the Map block method, per-block method does not require any physical blocks to store mapping entries. Moreover, it has a space advantage since the data area of physical pages will all be dedicated to data entries. However, this slows down the time to load or construct the mapping table since it has to go through several read operations on the OOB area of all the physical pages scattered across the flash. Log-based recovery schemes [7] records valid and previously invalidated mappings onto a separate log block for recovery usage but they have performance issues for example, caused by the duplicate write operations.

To alleviate the above issues, Translation Block methods employing On-Demand-based selective-page-mapping schemes like [1,3] were proposed but such approaches cannot apply the Map-block method because of its demand for a huge mapping table to be loaded in DRAM. Therefore, only 1% of the hot page-level mapping table is loaded into DRAM and 99% remains in the mapping blocks on flash. Consequently, recovering from power loss is of concern in such systems, therefore our proposed approach endeavors to improve and guarantee data consistency and performance coupled with faster reboot if sudden power loss occurs. We achieve this by requiring a few read operation to reconstruct the mapping table in DRAM while keeping in consideration the OOB area for updates check and hot/cold page separation.

7. Conclusions

In this study, we proposed an efficient, fault tolerant and fast cached mapping table (CMT) recovery scheme (*FTRM*) for flash memory, coupled with an effective hot/cold page separation algorithm. The design goal of *FTRM* was to guarantee fast and safe CMT recovery in DRAM with negligible performance degradation. To achieve this, *FTRM* keeps track of flash page accesses and updates by using an access counter *AC* and a validation flag *VF* implemented in the OOB area of each page. Whenever a mapping page is accessed during a read/write operation, that page access is recorded to an *AC* which is later used to separate the hot from cold mapping pages on flash. After an unexpected sudden power cut, during reboot, all data pages with $VF = 1$ are used for updating corresponding out-of-date mapping pages on flash. Then, some of the hot mappings initially separated by *AC* are loaded into DRAM as CMT entries. Even though our proposed approach is implemented on flash memory, it can also be applied to various kinds of FTL algorithms and designs without affecting the traditional file system. Our proposed *FTRM* approach, as evidenced from evaluation results, has superior performance and robustness when compared to the traditional FSSD-based FTLs.

Author Contributions: Conceptualization, R.M.; Data curation, R.M.; Formal analysis, R.M.; Investigation, R.M. and P.H.; Methodology, R.M.; Project administration, R.M.; Software, R.M.; Supervision, T.-S.C. and J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korea government (MSIT) (No. 2019R1F1A1059795 and 2019R1F1A1058548).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mativenga, R.; Hamandawana, P.; Kwon, S.J.; Chung, T. EDDAPS: An Efficient Data Distribution Approach for PCM-Based SSD. In Proceedings of the 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast, UK, 10–13 September 2018; pp. 158–159. [CrossRef]
2. Chang, H.S.; Chang, Y.H.; Kuan, Y.H.; Huang, X.Z.; Kuo, T.W.; Li, H.P. Pattern-Aware Write-Back Strategy to Minimize Energy Consumption of PCM-Based Storage Systems. In Proceedings of the 5th Non-Volatile Memory Systems and Applications Symposium, NVMSA 2016, Daegu, Korean, 17–19 August 2016.
3. Gupta, A.; Kim, Y.; Urgaonkar, B. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, 2009, ASPLOS XIV, Washington, DC, USA, 9–11 March 2009; pp. 229–240.

4. Mativenga, R.; Paik, J.Y.; Kim, Y.; Lee, J.; Chung, T.S. RFTL: Improving Performance of Selective Caching-Based Page-Level FTL Through Replication. *Clust. Comput.* **2018**. [CrossRef]
5. Kim, J.; Kim, J.M.; Noh, S.H.; Min, S.L.; Cho, Y. A Space-efficient Flash Translation Layer for CompactFlash Systems. *IEEE Trans. Consum. Electron.* **2002**, *48*, 366–375. [CrossRef]
6. Jung, D.; Kang, J.U.; Jo, H.; Kim, J.S.; Lee, J. Superblock FTL: A Superblock-based Flash Translation Layer with a Hybrid Address Translation Scheme. *ACM Trans. Embed. Comput. Syst.* **2010**, *9*, 40:1–40:41. [CrossRef]
7. Chung, T.S.; Lee, M.; Ryu, Y.; Lee, K. PORCE: An Efficient Power off Recovery Scheme for Flash Memory. *J. Syst. Arch.* **2008**, *54*, 935–943. [CrossRef]
8. Ramanan, K.; Williams, J. A Low Power Fast Wakeup Flash Memory System for Embedded SOCs. In Proceedings of the 2017 IEEE International Conference on IC Design and Technology (ICICDT), Austin, TX, USA, 23–25 May 2017; pp. 1–4. [CrossRef]
9. Choudhuri, S.; Givargis, T. Real-Time Access Guarantees for NAND Flash Using Partial Block Cleaning. In *Software Technologies for Embedded and Ubiquitous Systems*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 138–149.
10. Hennessy, J.L.; Patterson, D.A. *Computer Architecture: A Quantitative Approach*, 6th ed.; Morgan Kaufmann: Cambridge, MA, USA, 2019.
11. Lee, S.W.; Park, D.J.; Chung, T.S.; Lee, D.H.; Park, S.; Song, H.J. A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation. *ACM Trans. Embed. Comput. Syst.* **2007**, *6*, 18. [CrossRef]
12. Chung, T.S.; Park, D.J.; Park, S.; Lee, D.H.; Lee, S.W.; Song, H.J. System Software for Flash Memory: A Survey. In *Embedded and Ubiquitous Computing*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 394–404.
13. Mativenga, R.; Hamandawana, P.; Kwon, S.J.; Chung, T. ExTENDS: Efficient Data Placement and Management for Next Generation PCM-Based Storage Systems. *IEEE Access* **2019**, *7*, 148718–148730. [CrossRef]
14. Dayan, N.; Svendsen, M.K.; Bjørling, M.; Bonnet, P.; Bouganim, L. EagleTree: Exploring the Design Space of SSD-based Algorithms. *Proc. VLDB Endow.* **2013**, *6*, 1290–1293. [CrossRef]
15. SNIA IOTTA Repository. Available online: http://iotta.snia.org (accessed on 1 February 2015).
16. UMass Trace Repository. Available online: http://traces.cs.umass.edu/index.php (accessed on 1 February 2015).
17. Lasser, M. Method for Fast Wake-Up of a Flash Memory System. U.S. Patent No. 6,510,488, 21 January 2003.