Performance Evaluation 68 (2011) 1193–1206

Contents lists available at SciVerse ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

ShiftFlash: Make flash-based storage more resilient and robust

Ping Huang [a], Ke Zhou [a,*], Chunling Wu [b]

[a] School of Computer Science & Technology, Huazhong University of Science & Technology, Key Laboratory of Data Storage Systems, Ministry of Education of China, Wuhan National Lab for Optoelectronics, China

[b] Guilin Normal College, Guilin, China

ARTICLE INFO

Article history:
Available online 3 August 2011

Keywords:
CDP
Flash-storage
Recovery
SSD
Time-shifting

ABSTRACT

Flash based storage technology has been steadily gaining more and more popularity during the past decades due to its unique merits over conventional disk counterparts and has been projected to revolutionize the entire storage hierarchy. Though it is well-known that flash storage is physically more reliable than hard disk drives within its limited lifespan, neither of them provide sophisticated built-in mechanisms guarding against non-physical failures, such as virus attacks and unintentional errors. One of the unique characteristics of flash is "no in-place overwrites", which would cause a large amount of *superseded* pages/data to remain in the flash until they are selected to be erased by a garbage collection process. Leveraging this idiosyncrasy, we propose ShiftFlash, which provides flash based storage with time-shifting functionality to make it more robust and resilient. By monitoring and recording the modifications of the FTL mapping table, ShiftFlash enables flash state to be reverted to any point-in-time (PiT) in the past. It is implemented within SSD devices and needs minimal support from the upper layer. The trace-driven simulation results of a range of different workloads show that ShiftFlash only introduces marginal overheads with respect to several principal performance metrics, somewhere between 6% and 11%, compared with the original non-shifting flash. ShiftFlash also outperforms other time-shifting schemes by a large extent in many respects.

1. Introduction

Over the past decades, Flash Memory based Solid-state Drives (SSDs) have been widely used in mobile devices and laptops [1,2] and are becoming more and more popular even in large-scale data centers [3,4]. It's been said that flash technology has the potential to fundamentally change the current storage hierarchy. SSD has enjoyed its amazing success for its salient features which are commonly cited as extremely low random read access latency, high shock resistance, reasonable high reliability, low power consumption, non-volatility and small form size. Those superior merits mainly result from the absence of the mechanical moving components which dominate the high access latency of hard disk drives (HDDs). But, on the other hand, it also suffers from several inherent limitations imposed by the physical properties of its chip components, including notorious small random write performance, limited life cycles. The vast majority of existing research work was aiming at mitigating the limitations in order to make it work better and explore possible usage spaces of SSDs. For example, a variety of data layouts and FTL algorithms and implementations [5–13] have been proposed and judiciously evaluated to overcome those limitations. SSDs have also been deployed to be integrated into the storage hierarchy, either as an added tier like read/write cache layer [14–16] or as a component of hybrid storage architecture [17–19].

* Corresponding author.
E-mail addresses: pinghp.hust@gmail.com (P. Huang), k.zhou@hust.edu.cn (K. Zhou), wcl0203107@21cn.com (C. Wu).

0166-5316/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2011.07.010

Flash based storage (e.g. SSD) is mainly considered to be more reliable compared with HDDs for its lack of mechanical moving parts. Since in HDDs, when the disk spindle is moving quickly, it is much more subject to vibration or shock and is more prone to be erroneous. Rather than using magnetic material to store bit information which is the case with rotating disks, flash uses transistors and a floating gate [20] to keep information. The trapped charge states of the floating gate are distinguished as two states, corresponding to 0 and 1, respectively. Due to such different underlying storage mechanisms, flash storage can provide a better guard against physical failures and thus is guaranteed to be more reliable. Put another way, it's more physically reliable. However, such reliability alone is sometimes not adequate in certain scenarios, e.g. in the event of software errors, unintentional mistakes, malicious operations, virus attacks and the like. Under such circumstances, Continuous Data Protection (CDP) [21] technology is highly desirable. CDP is a paradigm in backup and recovery and a means of reliability guarantees. The key idea of CDP is to continuously capture and keep the happening I/Os to the protected targets. It allows the storage state to be potentially reverted to any point-in-time in the past [22] by restoring appropriate previously kept history data. The CDP function can be realized at different levels of the data path, e.g. at file system level [23–25] or at block level [26].

SSDs based on flash differ themselves from HDDs in many important aspects. One of the most distinctive characteristics is that they do not support "in-place overwrites". Another one is that they exhibit "erase-before-write" idiosyncrasy. Usually, there is a mapping table in the SSD controller which maps logical page numbers (LPN) to physical page numbers (PPN) of the underlying storage chips. Handling every write request, flash would first check whether it is a new write, if so, it allocates a new erased page to store the data, and then updates the mapping table to reflect the logical page's new location; otherwise, it writes the data to a newly allocated page, updates the corresponding mapping entry and at last marks the previous PPN as invalid (therefore called *superseded* pages [6]). The *superseded* pages are reclaimed by erasure operations afterwards. Thus, overwrites may cause significant *superseded* pages [27] to linger for a period of time whose length depends on the reclaiming policy. Those *superseded* pages represent the old versions of the data, and thus can be potentially deployed to implement time-shifting functionality.

Leveraging this potential opportunity, we propose ShiftFlash in this paper, which is a kind of flash with added CDP functionality. The rationale behind ShiftFlash is to make it guard against logical errors without incurring noticeable overheads to the original architecture by exploiting the inherent opportunity. We believe that as the deployment of flash based SSDs are steadily getting more widespread [28] and the vital importance of data and applications, especially business continuity, to enterprizes remains unchangeable, ShiftFlash would potentially perform an important role in future storage systems and data protection regimes. Our main contribution is that we developed a novel flash based SSD architecture implementing an expensive data protection scheme in a lightweight manner. We hope ShiftFlash would be of some help to system architects, especially hardware designers and implementers in manufacturing robust and resilient flash storage.

The remainder of this paper is organized as follows: Section 2 gives some background knowledge about both flash storage and CDP technology. Section 3 deals with the implementation details of ShiftFlash. Section 4 presents the evaluation methodology and results. A discussion about ShiftFlash is presented in Section 5. Section 6 overviews related works. Section 7 ends this paper with a concluding remark.

2. Background

2.1. Flash and SSDs background

Flash chips are a kind of non-volatile storage media and are the basic building blocks of SSDs. Flash memory is further categorized into NOR and NAND according to the architecture of the underlying transistors and floating gates [29,30]. A flash memory package consists of one or more dies (chips), each of which is segmented into units of planes. Each plane typically contains thousands (e.g. 2048) of blocks, which are further partitioned into a number (e.g. 64–128) of fix-sized pages. In addition to the data area, each page also contains a spare area that is used to store auxiliary information such as Error Correction Code (ECC) and its corresponding reversely mapped logical page number. Flash supports three types of operations which are read, write/program and erase. Read and write operations are performed in units of page and can be completed in tens or hundreds of microseconds, while erase operations are relatively much more expensive and can only be performed at block granularity, taking several milliseconds [6].

Due to the floating gate's charge trapping and detrapping phenomenon, flash suffers from several important technical limitations. One of them is "no in-place overwrites", i.e. overwrites cannot be performed directly in an in-place manner, resulting from the fact that pages can only be written after being erased. In order to overcome this limitation, a software layer FTL is implemented in the flash controller. FTL translates logical block addresses to physical block addresses and makes the flash export block interfaces like traditional disks. Existing FTL algorithms are broadly categorized into block mapping [7,31], page mapping [12] and hybrid mapping [9,11,32] according to the adopted mapping granularity. FTL enables the same logical block address to associate with different physical locations of the flash at different times, masking its peculiarities from upper layer applications and rendering erase operations out of the write path, which improves performance. A variety of FTL schemes and algorithms have been proposed in the research literature [5,8,13]. Thanks to the presence of FTL, overwrites are handled in a "read-modify-write" fashion and are in-directed to be sequentially written in erased blocks at a potential cost of write amplification [33]. Another critical shortcoming is that flash can only sustain a limited number of program/erase

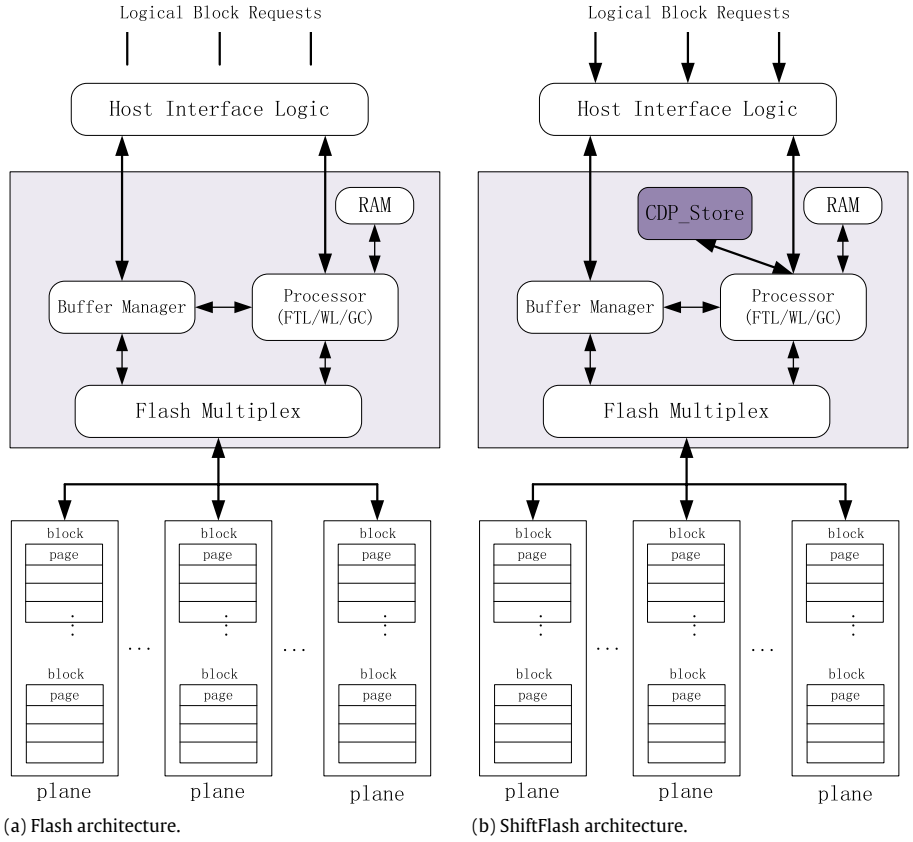


Fig. 1. Flash and ShiftFlash architectures.

(P/E) cycles, typically ranging from 10 to 100 K for MLC NAND and SLC NAND respectively, after which the bit error rates would become unacceptable and the data retention would be unreliable [34]. The block that runs out of its limited cycles is called worn out. In order to prevent the flash from prematurely breaking down due to it being partially worn out, two techniques named wear-leveling and garbage collection have been deployed to improve flash endurance. The main purpose of wear-leveling is to attempt to guarantee that all the blocks of the flash are evenly worn-out. The function of the garbage collection is to periodically recycle the superseded pages through block erasing in order to keep a set of free blocks available at any time.

An SSD is typically comprised of a host interface logic, an array of flash packages, a controller implementing FTL, wear-leveling, garbage collection, a flash multiplexer (mux/demux), buffer manager and peripheral circuitry. The host interface logic is used to accommodate SSDs to host interface connection. The controller containing a processor and additional RAM does sanity checks like ECC and performs address translation, wear-leveling algorithm, garbage collection. The multiplexer decodes received commands, executes low-level flash commands and handles data transport. The buffer manager holds pending requests and satisfies requests [6]. SSDs are often over-provisioned, i.e. the raw capacity is larger than their advertised capacities, in order to provide better performance. Fig. 1(a) shows the logical block diagram of a typical SSD.

2.2. Time-shifting function

The time-shifting function [24], also called Continuous Data Protection [21], is a data protection scheme that can ensure data resiliency and reliability [35] *n* the time dimension. It is widely deployed in organizations and corporations, usually together with snapshots [36] or periodical backups. By continuously (the frequency depends on the preset protection granularity) monitoring the updates of the protected targets and logging the history data, CDP allows the storage state to be rolled back to any point-in-time in the past, i.e. access earlier versions of the data, which is an important necessary property to ensure business continuity. It can be implemented either at file system level or block level using copy-on-write or indirect-on-write techniques [36]. For every coming write request, it would reserve a copy of the data as well as the writing timestamp before overwriting the data. There are two key concepts related to data recovery, which are Recovery Point Objective (RPO) and Recovery Time Objective (RTO). The two metrics are used to characterize the efficacy of a recovery process after failures have occurred. RPOs are the reversible time points available that can be reverted to; they are determined by the protection

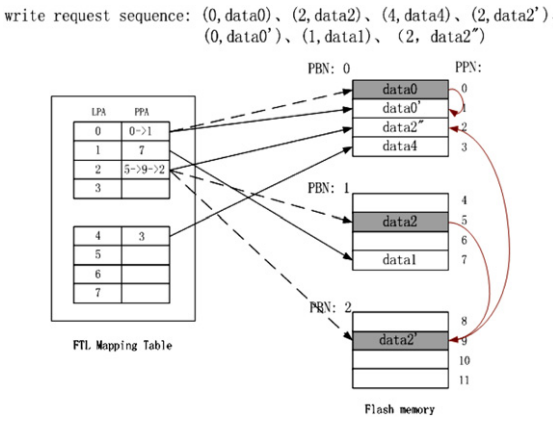


Fig. 2. The evolution of superseded pages generation.

granularity and imply the amount of potential data loss in terms of time length. RTOs represent the time period taken by recovery processes before the system can resume. The ideal value of RTOs is zero, which means recovery processes would be completed instantly. The recovery performance is mainly determined by how the metadata are indexed, searched and where the captured history data are stored, e.g. locally or remotely.

3. ShiftFlash design and implementation

As mentioned previously, the key idea of ShiftFlash is to take advantage of remnant *superseded* pages to make flash time-travelable. The *superseded* pages would remain invalid and unusable until they are erased at some time in the future. However, on the other hand, they could potentially be accessed by preserving old mapping entries. Since time-traveling functionality mandates the preservation of history data, we implement a CDP_Store module in ShiftFlash. Every time a FTL mapping entry is going to be updated, it is in advance backed up to CDP_Store. The CDP_Store can be hosted on a purposely reserved portion of the flash or on an added-on component, like PCM [37] for quick accesses. It provides interfaces for handling recovery commands, storing, retrieving and deleting previous FTL entries. Theoretically, as long as the *superseded* pages remain *superseded,* i.e. they still have not been erased, the flash state can be reverted to a previous point-in-time by restoring the appropriate FTL entries. Compared with the original architecture, ShiftFlash introduces an additional CDP_Store component. The block diagram of ShiftFlash is shown in Fig. 1(b).

3.1. The generation of superseded pages

In a standard page mapping scheme, the FTL mapping table maps each logical page address (LPA) to a physical page address (PPA). For every write request, FTL would first look into the mapping table to check whether it is a new write request or an overwrite request. If it is a new write request, FTL allocates a new physical page to write the data and then adds the new mapping entry into the mapping table, otherwise it takes the following steps: allocates a new physical page, writes the data, atomically updates the corresponding mapping entry to reflect the logical page's new mapped location, and after that invalidates the previously mapped page. Thus, every page overwrite operation would give rise to a *superseded* page.

Fig. 2 shows the process of flash handling an exemplar sequence of write requests. The write requests are represented by a series of tuples containing writing logical addresses and written data whose sizes are all assumed to be the same and equal to the flash page size (typically 2 or 4 kB). The left side shows the flash FTL mapping table and the right side shows flash physical pages. In that figure, dotted arrows represent superseded and obsolete mapping relationship, while solid arrows point to valid physical pages. From this figure, we know that the logical addresses' associated physical pages are varying with time due to their being overwritten. *Superseded* pages which are shaded in the figure are generated as a result of overwrite operations. For example, when logical page 0 is written with data 0 for the first time, FTL initially assigns physical page 0 to it. Later on, the fifth write request overwrites logical page 0 with new content data0', causing physical page 0 to be superseded and logical page 0 is updated to mapped to the newly allocated physical page 1. So it is the case with logical page 2, though it incurs two superseded pages which are physical page 5 and page 9 since it has been overwritten twice.

It is worth noting that the figure doesn't depict the real world scenarios totally. In that figure, we assume that each block contains only four pages and the pages are allocated in a pseudo-random manner. For example, the allocated pages for the first two consecutive requests are page 0 and page 5 which come from different blocks (PBN 0 and PBN 1). We keep this simplicity for ease of demonstration. In real world situations, every block typically contains thousands of pages and pages are allocated sequentially within individual blocks for better performance, as it is done with log buffer schemes [11,32]. The reason is that, if the pages are allocated in a pseudo-random way, it would be costly to reclaim them after they are superseded, due to their wide distribution across flash blocks, which is thought to account for the shortcoming of flash's

excruciating slow random writes. On the other hand, sequentially allocating pages is much easier to manage, and more importantly, it can produce better performance thanks to the wide existence of workloads' spatial and temporal locality which can with a high probability cause entire blocks to be invalid and thus erased without incurring additional page copying-back operations [33]. What's more, in the figure we only consider write requests, since read requests make no contributions to the generation of superseded pages.

3.2. ShiftFlash details

3.2.1. Main data structures

In order to keep track of the changing history of mapping entries and support CDP recovery, we have used several data structures and added modifications to some of the existing data structures. Specifically, we use a CDP_entry containing logical block address, corresponding timestamp and some auxiliary information to record every obsolete mapping entry, and all CDP_entries corresponding to the same logical block are linked in the same doubly-linked list. The link list represents the logical block's changing history. We extend each mapping entry to include a timestamp indicating when the mapping entry is created. Handling every overwrite request, a CDP_entry is inserted into the tail of the appropriate list, making elements of the list time-ordered. Every list head contains its corresponding logical page number and some information (e.g. physical page number and timestamp) about this logical page's being written for the first time. All of the lists' heads are organized in an array. There exist several tradeoffs between performance and the usages of SRAM. The addition of timestamps into the modified mapping entries may incur memory overheads, if it's not affordable, the timestamps can alternatively be stored at the metadata regions of the mapped pages, at the introduced cost of reading the metadata regions to get the timestamps every time the pages are superseded. Also, the CDP_Store can alternatively be stored on flash memory. When the protection window (defined later) advances, reclaiming the corresponding space occupied by CDP_Store is relatively easy, since it's been totally written sequentially along the time dimension. In our current prototype implementation, we add timestamps to the in-memory mapping entries and store CDP_Store in memory.

3.2.2. Recovery process

In ShiftFlash the recovery process is straightforward, since every logical page's historical CDP_entries within the permitted protection time window are kept in CDP_Store. Its simple task is to find each logical page's mapped page number that corresponds to the requested recovery time-point and reconstruct the FTL mapping table. In more detail, it scans through the entire head array, and for each logical page, it walks through the list to find the CDP_entry satisfying the condition that its timestamp is equal to or less than but nearest close to the recovery time point. If no such entry is found and its *first-written-time* which is stored in the head of the list is later than the recovery time point, it means that the logical page has not been written yet at the recovery time point and thus need not be restored. Fig. 3 shows the recovery process pseudo-code.

3.2.3. How does it work?

In this section, we show how ShiftFlash works with a demonstrating example. To simplify our presentation, we use updated mapping entries to represent the overwrite requests. For example, $(0, 9, T_1)$ represents that the logical page 0 was overwritten and assigned the new physical page 9 at time T_1. Each mapping table entry contains the logical page number (LPN), the associated physical page number (PPN) and the timestamp. The upper part of Fig. 4 shows how FTL mapping table evolves, while the bottom shows the content of *CDP_Store*. The left-most table of the upper part is the initial FTL mapping table state at time T_0, while the middle and right-most tables correspond to T_1 and T_2 states, respectively. At time T_1, four requests arrive, two of which are overwrites to logical page 0 and 1, causing *CDP_entries* $(2, T_0)$ and $(7, T_0)$ to be added into page 0 and page 1's lists, respectively. The unchanged mapping entry remains as before, e.g. entry $(2, 5, T_0)$ is the same as at time T_0. At time T_2, three overwrites happen to logical page 1,3,5, causing another three *CDP_entries* to be added onto *CDP_Store*. Now, suppose later at some time T $(T > T_2)$ the flash storage is requested to be restored to time $T'(T_0 < T' < T_1)$. Applying the recovery algorithm, we find three *CDP_entries* $(2, T_0)$, $(7, T_0)$ and $(11, T_0)$ which correspond to logical page 0,1,6, respectively. Since their timestamp T_0 is the low-maximum-less-than recovery time T'. Additionally, since logical page 2's list is empty and its *first-written-time* T_0 is smaller than the recovery time, it should be also included in the mapping table. Finally, we have a mapping table which contains four valid logical pages which are page 0,1,2 and 6, whose corresponding mapped physical pages are page 2,7,5 and 11, respectively, shown in Fig. 4.

3.3. Protection window

As time proceeds, *superseded* pages would be accumulated to a large amount, causing the whole capacity to run out. Thus, one potential limitation of ShiftFlash is its limited allowable past-travelable time period, which we define it as *protection window*. Ideally, the longer the protection window, the better it is. However, as we mentioned earlier, CDP is typically deployed combined with other techniques like periodical backups in practice and it is needed only during the time intervals between consecutive backups. Thus, the protection window is not strongly required to be very long in realistic use scenarios. To make the protection window as long as possible, we made two changes to the Garbage Collection (GC) process in the

```
Input(Recovery_time)
Output(table)
Start
   pCDP_entry *tmp;
   Mappingtable table;
   For every ith(0<= i < N) element in the head array
   //N: the total number of pages
        if(head[i].next == NULL && head[i].firsttime <= Recovery_time)
        {
              add the first time written information to table;
              continue;
         }
         else
              continue
        tmp = head[i]->next;
        while( tmp != &head[i])

          if(tmp->time > Recovery_time)
              break; // the page has never been written before Recovery_time
          else if(tmp->time < Recovery_time && (tmp->next == &head[i] ||
             tmp->next->time > Recovery_time))
             add tmp to table
          else if(tmp->time == Recovery_time)
              add tmp to table;
          else
              tmp = tmp->next;
        end while
   End for
End
```

Fig. 3. Pseudo-code of the recovery process.

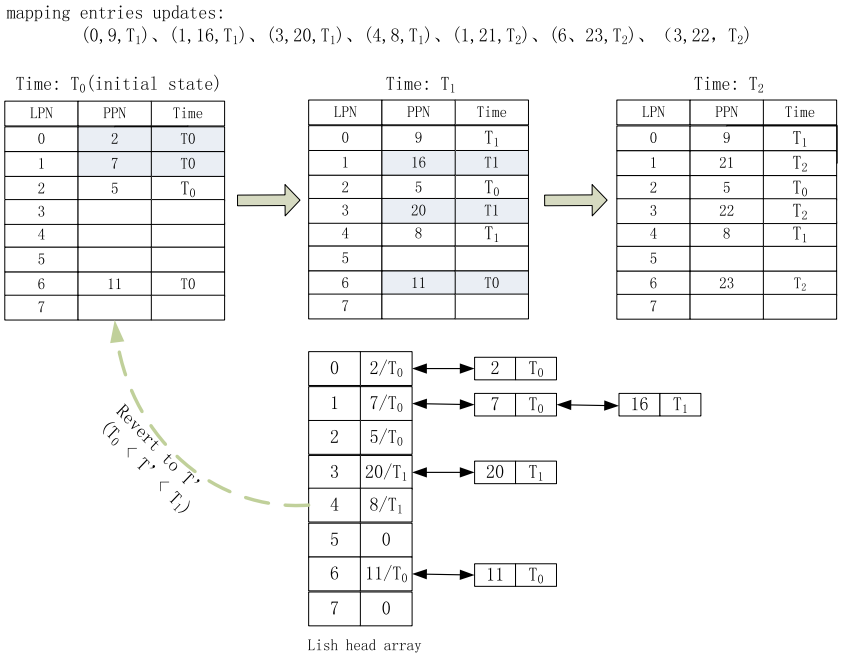


Fig. 4. The evolution of the mapping table and CDP_Store.

original system. One is that, we enlarge the allocation pool to be the whole SSD space, as opposed to original plane-level or chip-level strategy. The other is that we lower the cleaning threshold (i.e. the minimum number of free blocks), which as a result would defer the triggering of GC. However, these two changes would contribute negatively to the application's performance, as well as the GC process, as we will see in Section 4. When the garbage collection process is finally triggered,

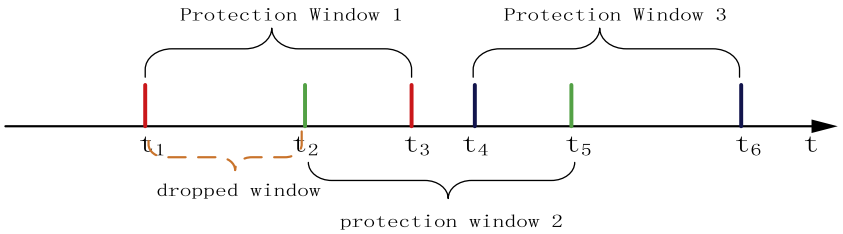


Fig. 5. Advancing protection window in ShiftFlash.

ShiftFlash advances the protection window to make free space. Specifically, it sets a larger lower protection time point (i.e. dropped time point) and releases the *superseded* pages belonging to the dropped time interval, i.e. releasing the oldest pages. As shown in Fig. 5, protection window 1's moving toward protection window 2 would result a dropped interval $[t1, t2]$. The pages that belong to the dropped interval are the candidates to be reclaimed. However, not all of the candidates are allowed to be reclaimed. Considering Fig. 4, if we are going to release all the pages that are before T_1, we cannot release physical page 5. That's because if we released it, then logical page 2 state would have never been correctly recovered. In other words, *superseded* pages cannot be reclaimed if they have not been superseded again after the dropped time point. To work out this dilemma, we snapshot the FTL mapping table state corresponding to the dropped time point (e.g. $t2$ in Fig. 5) as a new initial state, and then update the heads' first-time-written information to the same as the state in snapshot if their first written times are prior to the dropped time point.

3.4. Wear leveling and garbage collection process

There are three main background activities running in flash memory: cleaning, wear leveling and GC, which are closely related to each other. Cleaning is responsible for copying out valid pages from victim blocks before they are erased, i.e., reading out all the valid pages and rewriting them to other locations, at the potential expense of flash-inherent write amplification [33]. The cleaning policy would implicitly choose and determine which blocks are going to be reclaimed. Wear Leveling is a data placement optimization trying to uniformly wear out all blocks to improve the overall effective flash lifetime. The GC process is essentially a two-phase process. The first step is cleaning the valid pages from candidate blocks which are selected by the cleaning policy (e.g. using a greedy approach [6]) to other locations. The second step is to erase those victim blocks to free space. The specific behaviors of the three activities are subject to the workload's access patterns and thus are highly workload sensitive.

In ShiftFlash, the wear leveling mechanism is in principle similar to that of the original flash. The remaining lifetime of every block is dynamically tracked. The cleaning process chooses the blocks with the highest cleaning efficiency which is defined as the ratio of invalid pages to total pages within a block and migrates valid pages contained in them to blocks which have a relatively longer (e.g. beyond average) remaining lifetime. The differences lie in two aspects. One is that as opposed to using intra-plane policy in original flash, ShiftFlash integrates both inter-plane and inter-chip copy-back mechanisms, which is driven by the requirement of longer protection window. It means that valid pages can be copied out from victim blocks to other erased blocks residing in different planes or even chips. The other is that the definition of cleaning efficiency is different from the original one. In ShiftFlash, when calculating cleaning efficiency, all superseded pages (as long as they have not been marked dropped, as discussed later) and currently-mapped pages should be considered as valid, while in original architecture, only the currently-mapped pages are valid. Additionally, in order to avoid portions of the blocks being prematurely worn out due to the hot and cold data accesses, it switches the hot and cold region when access disparities have been observed, by copying the hot data into cold pages and filling cold data into hot pages when conducting cleaning. For example, in ShiftFlash, the blocks which are constantly/repeatedly written with good temporal locality access patterns are more likely to be reused and worn out. Since once those blocks are dropped, those superseded pages within them would correspond to the same logical page due to locality, causing higher cleaning efficiency and with a high probability of being selected again for reuse.

When the overall free space of ShiftFlash reaches the minimum threshold, the Garbage Collection (GC) process is triggered to advance the protection window and reclaim the oldest superseded pages, as mentioned in Section 3.3. Specifically, it examines the metadata regions (they include the active lifespan information, which is the time period between the written time-points of the first and last page) of all blocks to find out the ones that lie in the dropped window, and then marks the contained pages to be dropped. After that, the cleaning process would migrate the valid pages (recall, in ShiftFlash valid pages include no-yet-dropped superseded pages and currently-mapped pages) from those victim blocks, and update the corresponding entries in CDP_Store to remember the actual new locations for them. By contrast, in original flash, it only needs to copy out those currently-mapped pages. At first thought, it is tempting to conclude that ShiftFlash would incur significant additional write amplifications relative to the original non-shifting version. However, as we will see later, write amplification is highly sensitive to access patterns and would not cause too much overhead thanks to the wide locality exhibited by real world applications.

Table 1
Trace characteristics.

Trace	Description	Num of req	Write (%)
Financial1	OLTP	5334,987	76.8
Financial2	OLTP	3699,195	19.3
Cello99	HP-UX OS	443743	70.8
Websearch1	Search app	1055,448	0.12

Table 2
SSD simulation parameters.

Operation type	Latency
Page read	25 μs
Page write	200 μs
Block erase	2 ms
Serial access to register	100 μs

4. Evaluation methodology and results

In this section, we evaluate ShiftFlash using two sets of experiments. One is that we use trace-driven simulations to investigate the performance impacts of ShiftFlash relative to the original architecture, as well as the wear leveling and garbage collection behaviors. The other one is that we compare it with another two CDP schemes in terms of recovery performance, which is probably the most important concern of users facing system failures.

4.1. Experimental setup

We have implemented a page-mapping ShiftFlash in Microsoft's SSD simulator [6] extension to DiskSim [38]. We configure the SSD's size to be 64 GB, with 4 kB page size. We used four enterprize class real-world traces, financial1 [39], financial2 [39], cello99 [40] and Websearch1 [39] to drive the simulator to test ShiftFlash's performance. Additionally, we ran IOmeter [41] several times with different I/O patterns and collected the corresponding block-level traces. Then we used those traces to feed ShiftFlash to study the relationship between its write amplification phenomenon and access patterns. Due to the lack of source code availability, we implemented a TRAP [26] prototype according to the paper and a variation of TRAP which we call TRAP_NO_COM in the iSCSI target to make comparisons among them. TRAP_NO_COM is the same as TRAP except that it logs entire history data rather than only storing the differences between consecutive block updates. The used trace characteristics and SSD parameters are summarized in Tables 1 and 2, respectively. Before each experiment run, we deliberately constructed a full page-mapping FTL to emulate that the entire SSD has been fully used (thus WL and GC processes would be triggered soon) in order to accurately evaluate ShiftFlash. We think using that method would cause the ShiftFlash to be superseded soon, which is essentially the mechanism underlying ShiftFlash.

4.2. Performance metrics

In this section, we explore the user-perceived performance impacts. To measure the performance impacts of ShiftFlash, we ran both SSDsim and ShiftFlash simulator with the four traces and reported four main performance metrics. The four figures in Fig. 6 show the average response time, average read time, average write time and IOPS of SSDsim and ShiftFlash simulator, respectively. We can make several observations from those figures: (1) for all workload traces, ShiftFlash slightly underperforms compared with SSDsim. For example, the average response times of Financial1, Financial2, cello99 and Websearch1 increase by 11.5%, 6.9%, 8.2% and 5.4%, respectively. The IOPS also decreases only by 9.2%, 6.1%, 7.8% and 2.3%, respectively; (2) it seems that the write percentage determines how much the performance drops, i.e. the more write requests there are, the more severe the performance will degrade. As can be seen from Financial1 and Financial2, Financial1 gets a 11.5% response time increase while Financial2 gets a 6.9% response time increase, since Financial1's write percentage is much higher than that of Financial2. That results from the fact that as write requests consume capacity, the garbage collection process would be aggressively triggered and ShiftFlash's reclaiming process is relatively more expensive than SSDSim's, since ShiftFlash should be much more conservative when finding the pages to be claimed. More apparently, Websearch1 gets the least performance degradation, since its write percentage is 0.12%. Actually, Websearch1 is a read-dominate workload and there are almost no overheads caused by CDP functionality in ShiftFlash. What's more, Websearch1's sequentiality and locality also contribute to its relatively better performance. In summary, for all the four workloads, ShiftFlash only introduces trivial overheads, ranging from 6% to 11%, which is generally acceptable in light of its time-shifting functionality. However, for those performance-critically-sensitive applications that can't afford such degradations, the time-shifting function can be flexibly turned off to return it to the original no-shifting architecture.

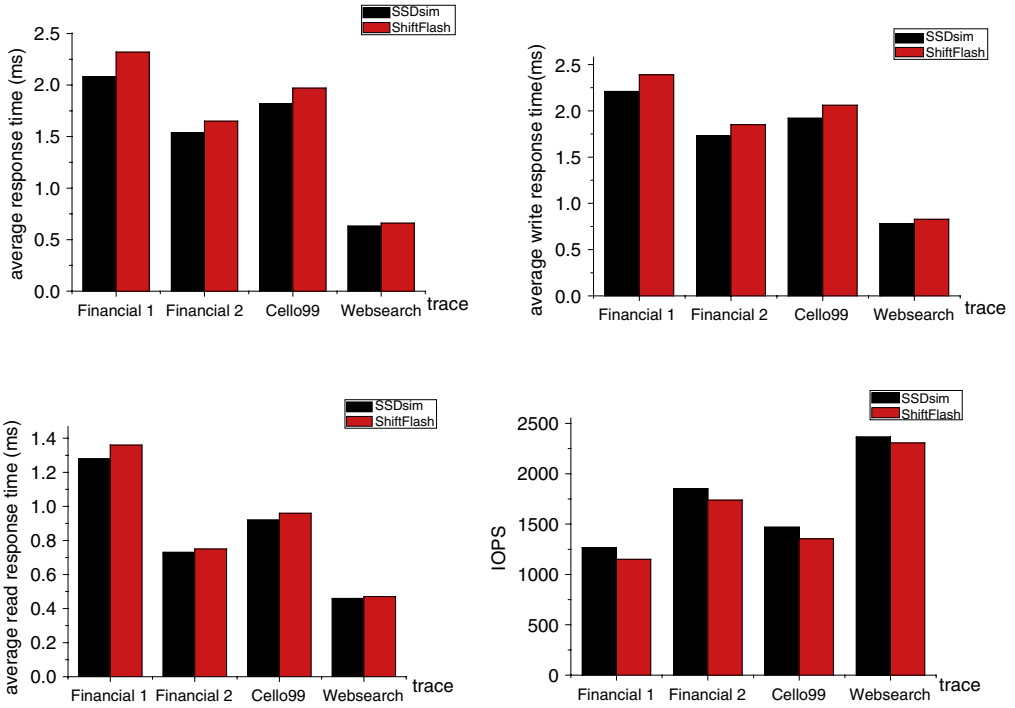


Fig. 6. The average response time, average read response time, average write response time and IOPS of SSDsim and ShiftFlash for different workload traces.

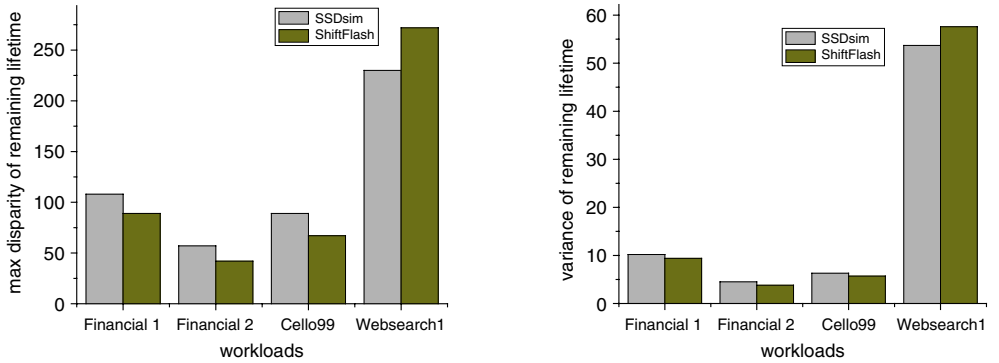


Fig. 7. The left picture shows the max remaining lifetime disparities and the right picture shows the variance for the different traces. Max block lifetime was set to 1000 cycles initially.

4.3. Wear leveling and garbage collection impacts

Obviously, the relative overall performance slowdowns discussed previously are rooted in multiple causes, which primarily include the affected wear leveling and garbage collection processes. In this section, we are going to get a deeper look at the microscopic behaviors of WL and GC in both non-shifting and time-shifting architectures under the same sets of experiments as in the above section, with the purposes of gaining an understanding of the intrinsic relationship between performance impacts and the background activities and revealing their behavior differences.

To evaluate the wear leveling behaviors, we collected and analyzed the remaining lifetime of all the blocks after the accomplishments of the experiments and report two related metrics, i.e., max remaining lifetime disparities and statistic variance of remaining lifetime. Both of the two metrics are good indicators of the block worn-out situations. The figures in Fig. 7 show the two metrics respectively for the four traces. Note that we set the lifetime of each block to be 1000 write–erase cycles before each experiment run. As can be seen from the figures, on the whole, the max disparities and variances for the respective same workloads of ShiftFlash are almost always uniformly smaller than that of SSDsim, demonstrating that ShiftFlash doesn't deteriorate but improves the wear leveling to some extent. For example, Financial1 exhibits max disparities of 108, 89 and variances of 10.2, 9.4 with SSDsim and ShiftFlash, respectively. Nevertheless, this result is not surprising. One possible reason is that ShiftFlash can migrate valid pages from victim blocks to other different planes or chips

Table 3
The number of migrated pages and its average overheads.

	SSDsim	ShiftFlash	Write (%)	Overheads per page
Financial1	14,321	15,728	9.8	1.8 ms/2.5 ms
Financial2	9,873	10,279	4.1	1.6 ms/2.3 ms
Cello99	2,246	2,365	5.3	1.3 ms/2.1 ms
Websearch1	324	412	2.7	0.8 ms/1.1 ms

when conducting the cleaning process, producing better global wear leveling at the cost of higher GC overhead as discussed later. On the contrary, SSDsim can only migrate valid pages to clean blocks in the same plane, which would potentially be accessed repeatedly, subjecting it to more reuse and wearing out. Also note that Websearch1 is the exception and differs in two respects. One is that for both SSDsim and ShiftFlash, it exhibits much higher max disparities and variances. The other is that ShiftFlash no longer gets better wear leveling. We think the reason lies in the fact that Websearch1 is heavily read-dominate. More specifically, the total capacity written by Websearch1 is relatively small and would less likely cause the GC process to be initiated, during which wear leveling is conducted.

To make a comparison of the GC behaviors in SSDsim and ShiftFlash, we report the number of migrated pages and the average GC overhead in terms of time spent during the trace simulating experiments. The statistics are presented in Table 3. Note that the two numbers in each cell of the last column represent the overheads of both SSDsim and ShiftFlash. From the table, we can make two important observations. First, though it's necessary to preserve past data in ShiftFlash as discussed in Section 3.3, we did not see significant write amplifications and the actual write amplifications are application-dependent. We would investigate further the relationship between write amplifications and access patterns in the subsequent section. Secondly, the average page migrating cost in ShiftFlash is more expensive than that of SSDsim. This can be attributed to the following two main reasons. The first reason is that, instead of being migrated to the same plane, valid pages in ShiftFlash can be migrated across chips, which is by nature costly due to the serial chip pins and bus's contention (however, this would produce better wear leveling, as we have discussed before). The second reason is, as we have mentioned we have lowered the cleaning threshold in ShiftFlash, which implicitly would delay the initiation of the GC process and make it slower and harder due to the less available free blocks needed by GC. It's also interesting to note that Websearch1 migrates much fewer pages and spends much less time migrating pages, due to its highly-skewed workload characteristics. Overall, we think the price of GC impacts is not too high to buy the time-shifting functionality, considering the importance of business continuity.

4.4. Write amplifications

To show more clearly the relationship between write amplification and access patterns, we used several IOmeter traces to study the write amplifications in ShiftFlash. We ran IOmeter with different configurations (producing different controllable access patterns) each for 30 min and collected the block-level request traces, then fed those traces to both SSDsim and ShiftFlash, which reports the number of page copy operations at the end of running process. Fig. 8 shows the story. The *x*-axis represents I/O patterns, e.g. 1:1(40%) means that the read/write ratio is 1:1 and the randomness is 40%. From that figure, we know that for workloads having the same randomness, the workloads with more writes would experience more page migrations, e.g. both SSDsim and ShiftFlash would experience more page copy operations with 1:2(40%) than with 1:1(40%). That's because only writes would consume capacity and trigger the GC process and erasure operations. We also note that for fixed read/write ratios, the workloads with higher randomness would cause more copy operations for both SSDsim and ShiftFlash. In other words, ShiftFlash would not experience too many write amplifications for workloads with reasonable locality, which is generally true for realistic applications. That's because good locality would generate a higher cleaning efficiency in ShiftFlash. Furthermore, ShiftFlash would amplify flash's low small random write shortcoming by exaggerating page migrations, which results from the difference in their clean policy, i.e. for SSDsim, as long as the pages that belong to a block are all superseded, they can all be claimed immediately. However, ShiftFlash probably needs to copy some pages out of the block and relocates some of them somewhere prior to erasure operations, as discussed in Section 3.3.

4.5. Recovery comparison

Finally, we compare the recovery performance of ShiftFlash with the well-known stat-of-the-art block level CDP scheme TRAP and its modified version TRAP_NO_COM. Though TRAP was originally implemented in the iSCSI target to verify its efficacy in reducing the storage capacity occupied by CDP history data, our purpose here is just to compare the time taken to finish recovering the same amount of data at the device level from the perspective of users who are experiencing system failures and wanting the systems to resume immediately. We wrote a utility program to write configurable sizes of data to the iSCSI target and logged the block-level traces to feed the ShiftFlash simulator, then reverted the state to the time point before the writing operation. We changed the data size and record the time taken to complete the recovery process. The result is shown in Fig. 9. As shown in that figure, ShiftFlash consistently outperforms both TRAP and TRAP_NO_COM by a large extent and the performance gap increases with the increasing amount of data volume, while TRAP_NO_COM is slightly better than TRAP for an absence of coding and decoding operations. Though the local network bandwidth may

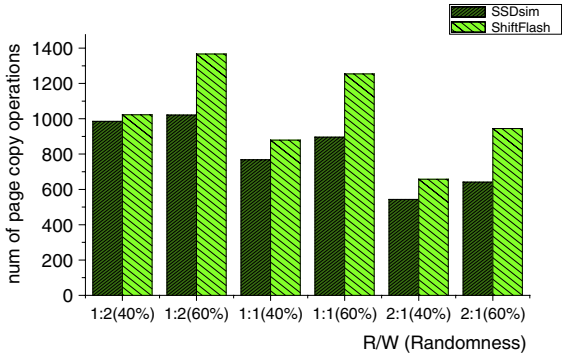


Fig. 8. The page copy operations caused by erase.

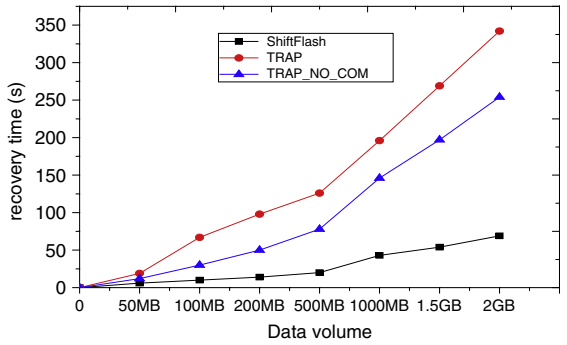


Fig. 9. CDP recovery performance comparison.

bottleneck TRAP and TRAP_NO_COM, the main reason why ShiftFlash outperforms is that ShiftFlash only needs to restore the FTL mapping table without involving any data reading and writing, which is exactly the observation that motivated our work.

5. Discussion

Time-shifting is an important capability to ensure business continuity. It can be implemented at different level along the data path. However, existing hard disk drives-based CDP implementations, either at file system level, like VersiongFS [25] and CVFS [23] or block level like TRAP [26], all cause non-trivial overheads to the original system. The reason is that they should not only keep the metadata changes, but also involve data read and write on the data path. By contrast, ShiftFlash is more lightweight, since it needs only to store the metadata changes without any data transfer expense on the data path thanks to the overwrite-forbidden property. One potential limitation of ShiftFlash may be its relatively small protection window due to *superseded* pages being reclaimed. Theoretically, the permitted length of the protection window is proportional to overall capacity and inversely proportional to consuming speed (write frequency). Even if it is the truth, we anticipate the potential applicability of ShiftFlash based on two observed facts. One is that, as semiconductor techniques advance, SSDs capacities are getting much larger and prices have steadily declined, thus they are able to sustain for a longer time period without reclaiming *superseded* pages imperatively. For example, for one 160 GB SSD, assuming the application continuously writes 20 MBps to it, the protection window can be 2.27 h. In practice, it would be much longer. The other one is that, in reality, CDP typically is used together with other backup techniques, e.g. periodic backups and snapshots. It is reasonably sufficient for CDP just to work during the time interval between consecutive backups, which is typically several hours or one day.

6. Related work

Flash and SSDs Related: Over the past decade, flash and SSDs have been extensively studied in the literature to overcome their inherent constraints, primarily focusing on FTL algorithms and write endurance. Agrawal et al. [6] investigated a range of design tradeoffs that are relevant to NAND-flash solid-state storage and developed a SSD simulator that can be seamlessly integrated into DiskSim [38] simulation environment. Gal and Toledo [5] made a survey on the algorithms and data structures used in flash memory. The Block Associative Sector Translation (BAST) [11] scheme exclusively associates a log block with a data block which would potentially cause superseded pages to be widely scattered across the blocks,

incurring too often block merger operations. Fully Associative Sector Translation (FAST) [32] improves BAST by allowing log blocks to be shared by all data blocks, enhancing the utilization of log blocks. The locality-Aware Sector Translation (LAST) scheme [42] overcomes the shortcomings of FAST through providing multiple sequential log blocks to exploit spatial locality in workloads. The superBlock FTL [9] scheme combines consecutive logical blocks into a superblock to utilize block level spatial locality in workloads. DFTL [13] caches part of the mapping table in limited RAM to support page-level mapping. A reconfigurable FTL [8] is proposed to dynamically adjust the associations between data blocks and log blocks according to characteristics of target workloads. Flash write endurance is enhanced by employing sophisticated buffer management [7,43], or using the HDD as cache to absorb small random writes [16], or utilizing CA-FTL [44,45] which aims to diminish the amount of write traffic to flash by applying de-duplication. Hu et al. [33] analyzed write amplification in flash memory. Yang et al. proposes a hybrid array architecture I-CASH [18] composed of HDDs and SSDs. Gordon [3] integrates flash storage into large-scale clusters to achieve high-performance and energy efficiency. Exhaustive analysis of trade-offs between power, energy and performance were made in that paper. Narayanan et al. [46] and Guerra [47] investigated the integration of SSDs to storage system from a viewpoint of financial cost. Empirical studies [34,48] have been conducted to speculate about flash/SSD's internal details which are not officially released publicly. Flash specific file systems [31,49] have also been proposed in order to make the most out of flash storage. Jung [10] proposes a group-based flash wear-leveling algorithm which reduces memory usage by grouping several logically sequential blocks and only managing group summary information.

Of the above related work, CA-FTL is very similar to ShiftFlash in the sense that they also realize additional functionality, which is data de-duplication, within the original architecture. But CA-FTL aims to reduce the write traffic to flash to prolong its lifetime, while ShiftFlash resolves to improve data resiliency and is orthogonal to CA-FTL. Furthermore, ShiftFlash is implemented in a much more lightweight manner through exploiting already existing opportunity. As opposed to ShiftFlash's persisting with superseded pages as long as possible, Wei [27] suggests that superseded pages should be sanitized immediately for the better of data security. They achieve that purpose by adding an extension to FTL which utilizes the fact that flash bits can only be programmed from 1 to 0 s, i.e. from erased states to programmed/written states.

Walk back to the past: Updates to storage states can be monitored and logged at file system level and block level. Versionfs [25] is a user-level file versioning system which is very flexible and highly portable. It can work with any Unix-semantic-compliant file systems. It creates file versions automatically and provides a host of interfaces to configure a variety of policies such as data retention and version storage policies. Ext3cow [24] achieves time-shifting functionality by deploying a copy-on-write scheme. Modifying ext3 file system code while preserving API interfaces, ext3cow is totally transparent to upper layers. Soules et al. [23] have examined two space-efficient file version metadata management schemes in Comprehensive Versioning File System (CVFS). UCDP [50] realizes user-level NFS-specific file versioning by logging NFS requests and responses. Wayback [51] is a user-level file system built on the FUSE framework. Peterson [52] studies how to securely delete earlier versions of individual files. TRAP [26] is a block level continuous data protection architecture with its focus on minimizing the necessary storage capacity for historic versions. Knowing the fact that consecutive updates to the same block exhibit only minimal differences, TRAP compactly stores the XORed result of the two versions, whose content is for the most part zeros. Laden [22] proposed four possible block level CDP architecture alternatives in the controller and studies their respective overheads in terms of space and extra disk I/Os. UVFS [53] is designed to reconstruct file versions from disk block versions maintained by a block level CDP. SWEEPER [54] attacks the problem of quickly identifying the most suitable recovery point for a clean data state in a CDP system by deploying Event monitoring, Checkpoint Indexing, and new search techniques.

7. Conclusion

As the saying goes, every coin has two sides. Flash memory based storage is known to have several limitations resulting from its inherent physical property. Superseded pages are the "negative" results of those limitations, since once superseded, they are invalid and would remain in the flash just as garbage until they are erased again. Interestingly, as is always the case with things in the real world, that "garbage" can also potentially be leveraged to realize special purposes, just like ShiftFlash has demonstrated. While sophisticated and carefully-designed ECC and the removal of mechanical rotating parts are there to make flash memory based storage more reliable, we have explored the flash reliability problem from a distinctive angle. By leveraging the superseded garbage pages in the flash, we have implemented time-shifting functionality at the device level, making it more robust and resilient and the introduced overheads are minimal and acceptable. Due to the numerous advantages of SSDs and their longer realistic lifetime than commonly believed [55], SSDs will get widely deployed. Hopefully, we anticipate that ShiftFlash would play its role in the storage system along with the emerging wide deployment of SSDs.

Acknowledgments

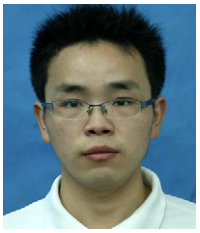
The authors are so grateful to the anonymous reviewers, in particular, our shepherd Prashant Shenoy, for their very constructive and insightful comments and feedback which substantially improved the paper's quality. We also would like to express our sincere gratitude to the whole program committee, particularly the chairs, for their patience in answering our various questions. This work is supported in part by the National Basic Research Program (973 Program) of China under

Grant No. 2011CB302305, the National High Technology Research and Development Program (863 Program) of China under Grant No. 2009AA01A402.

References

[1] F. Chen, S. Jiang, X. Zhang, SmartSaver: turning flash drive into a disk energy saver for mobile computers, in: Proceedings of ISLPED'06, October 2006.
[2] M.L. Chiang, P.C.H. Lee, R.C. Chang, Cleaning policies in mobile computers using flash memory, The Journal of Systems and Software 48 (3) (1999) 213–231.
[3] A.M. Caulfield, L.M. Grupp, S. Swanson, Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications, ACM SIGPLAN Notices 44 (3) (2009) 217–228.
[4] D. Gantenbein, Faster servers, services with flashstore, 2011. http://research.microsoft.com/en-us/news/features/flashstore-021411.aspx.
[5] E. Gal, S. Toledo, Algorithms and data structures for flash memories, in: ACM Computing Survey'05, vol. 37 (2), 2005, pp. 138–163.
[6] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, Design tradeoffs for SSD performance, in: Proceedings of USENIX Annual Technical Conference, 2008, pp. 57–70.
[7] H. Kim, S. Ahn, BPLRU: a buffer management scheme for improving random writes in flash storage, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08, San Jose, CA, February 2008.
[8] C. Park, W. Cheon, J. Kang, A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications, ACM Transactions in Embedded Computing Systems 7 (4) (2008).
[9] J. Kang, H. Jo, J. Kim, J. Lee, A superblock-based flash translation layer for NAND flash memory, in: Proceedings of the 6th ACM & IEEE International Conference on Embedded Software, EMSOFT'06, New York, USA, 2006, pp. 161–170.
[10] D. Jung, Y.-H. Chae, H. Jo, A group-based wear-leveling algorithm for large-capacity flash memory storage systems, in: Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES'07, September 2007, pp. 160–164.
[11] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, A space-efficient flash translation layer for compactflash systems, IEEE Transactions on Consumer Electronics 48 (2) (2002) 366–375.
[12] A. Birrell, M. Isard, C. Thacker, T. Wobber, A design for high-performance flash disks. Technical Report MSR-TR-2005-176, Microsoft Research, December 2005.
[13] A. Gupta, Y. Kim, B. Urgaonkar, DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings, in: Proceedings of ASPLOS'09, Washington, DC, March 2009.
[14] J. Matthews, S. Trika, D. Hensgen, R. Coulson, K. Grimsrud, IntelRturbo memory: nonvolatile disk caches in the storage hierarchy of mainstream computer systems, Transactions on Storage 4 (2) (2008) 1–24.
[15] Microsoft Corporation, Microsoft Windows ReadyBoost. http://www.microsoft.com/windows/windows-vista/features/readyboost.aspx.
[16] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, T. Wobber, Extending SSD lifetimes with disk based write caches, in: Proceedings of the 8th USENIX Conference on File and Storage Technologies, FAST'10, San Jose, CA, February 2010.
[17] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement, in: Proceedings of HPCA'10, Bangalore, India, January 2010.
[18] J. Ren, Q. Yang, I-CASH: intelligently coupled array of SSDs and HDDs, in: Proceedings of HPCA'2011, February 2011.
[19] Y. Kim, A. Gupta, B. Urgaonkar, MixedStore: an enterprise-scale storage system combining solid-state and hard disk drives, Technical Report CSE 08-017, Department of Computer Science and Engineering, The Pennsylvania State University, September 2008.
[20] P. Pavan, R. Bez, P. Olivo, E. Zanoni, Flash memory cells—an overview, Proceedings of the IEEE 85 (1997).
[21] Storage Networking Industry Association. An overview of today's continuous data protection (CDP) solutions, 2006. http://www.snseurope.com/supplements/snia-1-4.pdf.
[22] G. Laden, P. Ta-Shma, E. Yaffe, Architectures for controller based CDP, in: Proceedings of the 5th USENIX Conference on File and Storage Technologies, FAST'07, 2007.
[23] C.A.N. Soules, G.R. Goodson, J.D. Strunk, G.R. Ganger, Metadata efficiency in versioning file systems, in: Proceedings of 2nd USENIX Conference on File and Storage Technologies, FAST'03, March 2003.
[24] Z.N.J. Peterson, R. Burns, Ext3cow: a time-shifting file system for regulatory compliance, ACM Transactions on Storage 1 (2) (2005).
[25] K.M. Reddy, C.P. Wright, A. Himmer, E. Zadok, A versatile and user-oriented versioning file system, in: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, FAST'04, 2004.
[26] Q. Yang, W. Xiao, J. Ren, TRAP-array: a disk array architecture providing timely recovery to any point-in-time, in: Proceedings of the 33rd Annual International Symposium on Computer Architecture, ISCA'06, June 2006, pp. 289–300.
[27] M. Wei, L.M. Grupp, F.E. Spada, S. Swanson, Reliably erasing data from flash-based solid state drives, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, February 2011.
[28] T. Claburn, Google plans to use Intel SSD storage in servers, http://www.informationweek.com/news/storage/systems/showArticle.jhtml?articleID=207602745.
[29] Solid State Storage Initiative. NAND Flash Solid State Storage for the Enterprise, An In-Depth Look at Reliability, 2009.
[30] *M*-Systems. Two technologies compared: NOR vs. NAND, White Paper, 2003.
[31] A. Ban, Flash file system. United States Patent, No. 5, 404, 485, April 1995.
[32] S. Lee, D. Park, T. Chung, D. Lee, S. Park, H. Song, A log buffer-based flash translation layer using fully-associative sector translation, ACM Transactions on Embedded Computing Systems 6 (3) (2007).
[33] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, R. Pletka, Write amplification analysis in flash-based solid state drives, in: Proceedings of SYSTOR'2009, 2009.
[34] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, J.K. Wolf, Characterizing flash memory: anomalies, observations and applications, in: Proceedings of MICRO'09, New York, USA, 2009.
[35] Z. Yang, Y.F. Dai, AutoProc: an automatic proactive replication scheme for P2P storage, Science China Information Sciences 54 (2011) 1–10. doi:10.1007/s11432-011-4260-5.
[36] D. Hitz, J. Lau, M. Malcolm, File system design for an NFS file server appliance, in: Proceedings of the Winter 1994 USENIX Conference. San Francisco, CA, January 1994.
[37] Numonyx White Paper. The basics of phase change memory (PCM) technology. http://www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf.
[38] J. Bucy, J. Schindler, S. Schlosser, G. Ganger, DiskSim 4.0, 2010. http://www.pdl.cmu.edu/DiskSim.
[39] Available at: http://traces.cs.umass.edu/index.php/Storage.
[40] HP Labs, Tools and Traces. http://tesla.hpl.hp.com/.
[41] IOMeter. http://www.iometer.org.
[42] S. Lee, D. Shin, Y. Kim, J. Kim, LAST: locality-aware sector translation for NAND flash memory-based storage systems, in: Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability, SPEED2008, Feburary 2008.
[43] J. Hu, H. Jiang, L. Tian, L. Xu, PUD-LRU: an erase-efficient write buffer management algorithm for flash memory SSD, in: Proceedings of the 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2010, August, 2010.
[44] A. Gupta, R. Pisolkar, B. Urgaonkar, A. Sivasubramaniam, Leveraging value locality in optimizing NAND flash-based SSDs, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, February 2011.

[45] F. Chen, T. Luo, X. Zhang, CAFTL: a content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, February 2011.
[46] D. Naraynan, E. Thereska, E. Donnelly, Migrating enterprise storage to SSDs: analysis of tradeoffs, in: Proceedings of EuroSys'09. Nuremberg, Germany, March 2009.
[47] J. Guerra, H. Pucha, J. Glider, W. Belluomini, R. Rangaswami, Cost effective storage using extent based dynamic tiering, in: Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST'11, February 2011.
[48] F. Chen, D.A. Koufaty, X. Zhang, Understanding intrinsic characteristics and system implications of flash memory based solid state drives, in: Proceedings of SIGMETRICS/Performance'09, Seattle, WA, June 2009.
[49] W.K. Josephson, L.A. Bongo, D. Flynn, DFS: a file system for virtualized flash storage, in: Proc. of the 8th USENIX Conference on File and Storage Technologies, FAST'10, February 2010.
[50] N. Zhu, T. Chiueh, Portable and efficient continuous data protection for network file servers, in: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN'07, 2007.
[51] B. Cornell, P.A. Dinda, F.E. Bustamante, Wayback: a user level versioning file system for Linux. in: USENIX Annual Technical Conference, 2004.
[52] Z.N.J. Peterson, R. Burns, J. Herring, Secure deletion for a versioning file system, in: Proc. of the 4th USENIX Conference on File and Storage Technologies, FAST'05, December, 2005.
[53] M. Lu, T. Chiueh, File Versioning for block-level continuous data protection, in: Proceedings of the 29th IEEE ICDCS, ICDCS'2009, 2009.
[54] A. Verma, K. Voruganti, R. Routray, R. Jain, SWEEPER: an efficient disaster recovery point identification mechanism, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08, San Jose, CA, February 2010, 2008.
[55] V. Mohan, T. Siddiqua, S. Gurumurthi, R. Stan, How I learned to stop worrying and love flash endurance, in: Proceedings of HotStorage'10, Boston, MA, June 2010.



Ping Huang was born in 1984. Currently, he is a Ph.D. student majoring in computer architecture at HuaZhong University of Science and Technology (HUST), China. His research interests include distributed storage systems, operating systems, file systems and emerging storage technologies, etc. (Email:pinghp.hust@gmail.com)

Ke Zhou was born in 1974 in Xiangtan, Hunan, China. He received the Ph.D. degree from the College of Computer Science and Technology, HuaZhong University of Science and Technology (HUST) in 2003. Currently, he is a Professor of the College of Computer Science and Technology at HUST. His main research interests include computer architecture, network storage systems, parallel I/O, storage security and network data behavior theory. (Corresponding Author, Email:k.zhou@hust.edu.cn)

Chunling Wu was born in 1981 in Guilin, China. She received her B.S and M.S degrees from Guilin University of Electronic Technology in 2006 and 2009, respectively. Currently, she is a lecturer at Guilin Normal College. Her research interests include parallel computer architecture, operating systems and embedded systems. (Email:wcl0203107@21cn.com)