An Efficient Sudden-Power-off-Recovery Design with Guaranteed Booting Time for Solid State Drives

Yu-Ming Chang[1,3], Ping-Hsien Lin[1,4], Ye-Jyun Lin[1,5], Tai-Chun Kuo[1,6],
Yuan-Hao Chang[2,10], Yung-Chun Li[1,7], Hsiang-Pang Li[1,8], KC Wang[1,9]

[1]Macronix International Co., Ltd., Emerging System Lab., Hsinchu 300, Taiwan, R.O.C.
[2]Institute of Information Science, Academia Sinica Taipei 115, Taiwan, R.O.C.
{[3]sanchang, [4]pinghsienlin, [5]yejyunlin, [6]tjkuo, [7]monixslee, [8]sbli, [9]kcwang}@mxic.com.tw, [10]johnson@iis.sinica.edu.tw

Abstract

Solid state drives (SSDs) that deliver high-bandwidth and low-latency performance have become the mainstream of storage devices in modern systems. Over the past years, there has been a great deal of researches conducted to improve the SSD performance or reliability with parallel or efficient address translation designs. On the contrary, little work is done for the optimization to guarantee the booting/recovery time of SSDs after any sudden power-off. Motivated by the fact that the fast-growing SSD capacity gradually makes existing scanning and recovering processes become infeasible and unacceptable, we propose an efficient sudden-power-off-recovery design to recover an SSD with guaranteed booting time. The proposed design was implemented on an SSD prototyping platform equipped with in-house NAND flash memories and was evaluated with various benchmarks. The results demonstrate that after sudden power-off, the prototyped SSD can be recovered with a guaranteed and bounded booting time between 80*ms* and 200*ms*.

1. Introduction

Sold state drives (SSDs) that include multiple flash-memory chips have become a popular alternative to replace hard disk drives (HDDs) in recent years, because of their shock resistance, high energy efficiency, and high I/O performance. In order to be compatible with the existing storage interface, i.e., logical block address (LBA), each SSD needs to keep track of the address translation information from LBAs to their corresponding physical addresses in the flash memory. The address translation information is usually maintained/cached in the internal RAM space of the SSD. Once a sudden power-off (or power-loss) occurs, the address translation information would be crashed and need to be recovered by scanning the whole flash-memory space. Such a scanning and recovering process is extremely time-consuming, and is gradually becoming infeasible and unacceptable due to the fast-growing storage capacity. Such an observation motivates us to explore the solution that could efficiently recover the address translation information to make SSDs resilient to sudden power losses and system crashes.

An SSD usually consists of multiple flash-memory chips. Each chip is composed of a large number of blocks. Each block consists of a fixed number of pages, where a block is the basic unit of erase operations and a page is the unit of read/write operations. Each page is divided into a data area and a spare area. The data area is used to store user data, and the spare area is also called out-of-band (OOB) area that maintains the house-keeping information such as the error correction codes (ECC) and information of logical block addresses. Because of the *write-once property*, each page can not be overwritten unless its residing block is erased. A typical solution to overcome this constraint is to adopt the *out-place update* that writes updated data in free pages to improve the write performance. As a result, multiple versions of the same data could coexist in the SSD at the same time. The up-to-date version is called *valid data* and the old versions are considered as *invalid data*. The pages with valid data (resp. invalid data) are referred to as valid pages (resp. invalid pages). Due to the out-place update, in each SSD, a management software, i.e., flash translation layer (FTL), is needed to maintain the address translation information that maps each LBA to its corresponding valid page/data. Note that we refer "address translation information" and "mapping information" interchangeably when there is no ambiguity. In addition, the FTL also includes a garbage collector that is activated to reclaim space of invalid data when there is not enough free space in the SSD. Due to the limited number of program/erase (P/E) cycles of each block, the FTL also includes a wear leveler that is used to prolong the lifetime of the SSD by evenly erasing flash blocks to avoid wearing out any block prematurely.

Due to the importance of address translation information in SSDs, many excellent FTL designs were proposed to resolve the management issue and to achieve a good compromise between read/write performance and RAM space requirement [7,10,13,16]. Based on the on-demand loading/storing address translation information, some proposed to include a update mapping table that only logs the modified mapping information to reduce the overheads on loading/storing address translation information [14]. Another research direction is the garbage collection (GC) that also has significant impact on the performance of SSDs. The simplest solution is the greedy policy that selects the block with the largest number of invalid pages to minimize the overheads on moving valid pages out of the to-be-erased blocks [17], and some others proposed hot-cold swapping and data clustering to improve the GC performance with considering the age of valid data [6,9,11,12]. Meanwhile, to resolve the endurance issue, some researchers focused on different wear leveling designs to extend the lifetime of flash storage devices by moving data around flash blocks to prevent wearing out any block excessively [1,2,5,15]. Furthermore, due to the advances of manufacturing technology, the reliability of flash memory has drawn a lot of attention in recent years. To tackle this issue, some researchers proposed to improve the reliability of flash memory by reducing the write disturbance [4] or including some parity information to improve the capability on correcting error data [3,19]. However, there is little work that focuses on how to efficiently and reliably recover the address translation information of SSDs after sudden power loses, even though the fast-growing capacity of SSDs gradually makes greedy scanning methods infeasible.

In this work, an efficient sudden-power-off recovery (SPOR) design is proposed to enhance the reliability of SSDs with the guaranteed booting time without additional hardware support. The proposed design aims at recovering address translation information correctly and efficiently after any (normal or sudden) power-off. In particular, the design is configurable such that the booting (recovery) time can be bounded and guaranteed by adjusting the synchronization frequency of address translation information from RAM to flash memory during the runtime. We must point out that the proposed design only has to read relatively a small number of pages during the system recovery, in contrast to the past recovery design by scanning the whole flash-memory space. The evaluation results demonstrate that the average booting time is 132ms under all investigated benchmarks. Especially, the booting time is bounded between the theoretical worst-case and the best-case values derived from our analysis.

The rest of the paper is organized as follows. Section 2 presents the proposed sudden-power-off-recovery (SPOR) design. Section 3 reports the experiment results. Section 4 is the conclusion.

2. A Sudden Power-off Recovery Design

2.1 Overview

In this section, a *sudden power-off recovery (SPOR) design* is proposed to enhance the reliability of an SSD by enabling the SSD to survive under any sudden power-off without any power-off notification to the controller of the SSD. To be more specific, the SPOR design can efficiently recover the address translation information after sudden power losses. Meanwhile, it is realized with a software implementation so that no additional hardware resource, e.g., super capacitor, is required. To preserve the flexibility, this design is configurable and able to control the expected

978-1-4673-8833-7/16/$31.00 ©2016 IEEE

booting (or recovery) time by adjusting the synchronization frequency for the data stored on RAM and on flash memory. With regard to the compatibility of various SSD designs, the proposed idea is presented based on the most well-know design of flash translation layer, i.e., DFTL [10], which proposed to load address translation information to RAM (e.g., DRAM and SRAM) on demand.

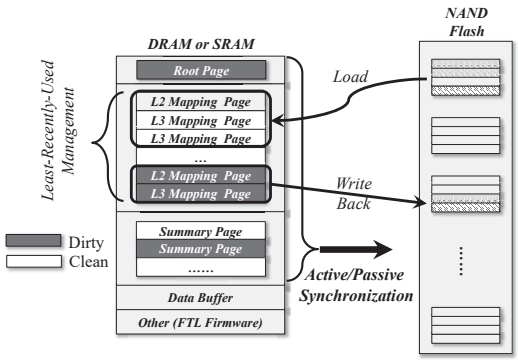

Figure 1. The architecture of the SPOR design for SSDs.

Figure 1 shows the architecture of the proposed SPOR design, which is able to resist sudden power-off to ensure data integrity. Since the access speed of RAM is much faster than flash memory, the frequently-accessed data such as frequently-used metadata should be stored on RAM. In the proposed design, the SRAM contains a *root page*, *L2/L3 mapping pages*, *summary pages*, *data buffer*, and the *FTL firmware* at runtime. The root page and the L2/L3 mapping pages are the address translation information (see Section 2.2) that (1) maps the logical address of a read/write request to the corresponding physical address and (2) are loaded on demand from flash memories. The summary page is used to store the auxiliary data, such as bad/free block tables and valid/invalid counters, for the SSD management. The data buffer stores the data of write requests issued from the host and the data read from flash memories. The FTL firmware stores the code for running the FTL design and the proposed SPOR design.

The data on RAM (e.g., DRAM and SRAM) are called *on-RAM data*, and the data on (NAND) flash memory are called *on-NAND data*. To reduce the hardware cost of RAM, the memory allocated for storing address translation information is limited [1]. Thus, the proposed design shall try to hold only the frequently-accessed mapping pages on RAM by managing them in the least-recently-used (LRU) fashion. Since RAM is volatile and can not preserve any data after the power-off (or power recycling), some on-RAM data must be written back to flash memory timely for data consistency and mapping information recovery. More specifically, the on-RAM data that have been updated and are not consistent with the copies in flash memory are referred to as *dirty data* (marked with gray color in Figure 1) and should be eventually written back to flash memory. On the other hand, the clean data (marked with white color in Figure 1) are consistent with the copies in flash memory so that they can be discarded directly in any circumstances.

Synchronizing the on-RAM and on-NAND data timely and efficiently is the key issue to the read/write and booting efficiency of SSD. To this end, the proposed SPOR design includes two synchronization mechanisms, i.e., active synchronization and passive synchronization, to synchronize critical on-RAM data actively and passively (see Section 2.3); thus, the data integrity is guaranteed and the full mapping information is recoverable. In addition, we also present the booting/recovery procedure to reconstruct all the on-RAM data including the mapping and summary information to correctly access data of the up-to-date version (see Section 2.4). In particular, the booting time is configurable and bounded with the proposed SPOR design such that the booting cost can be minimized with a given SSD specification.

2.2 Paging-like Address Translation Mechanism

In this section, a *paging-like address translation* mechanism in the proposed SPOR design is presented to illustrate the flow on serving the read/write requests from the host, where any metadata, i.e., mapping information, used in the mechanism should be recoverable after any sudden power-off. It will translate a logical page address (LPA) of a read/write request to a physical address of flash memory in a way that is very similar to the *paging* mechanism used in the memory management of modern OSes. In this way, only a part of mapping pages are required and loaded (or read) to RAM, so that the space efficiency of RAM is improved as the RAM size is limited. To simplify the illustration for the address translation, we only focus on the discussion in looking up the mapping pages to find the physical address of a given logical page address, instead of the management of mapping pages on RAM.

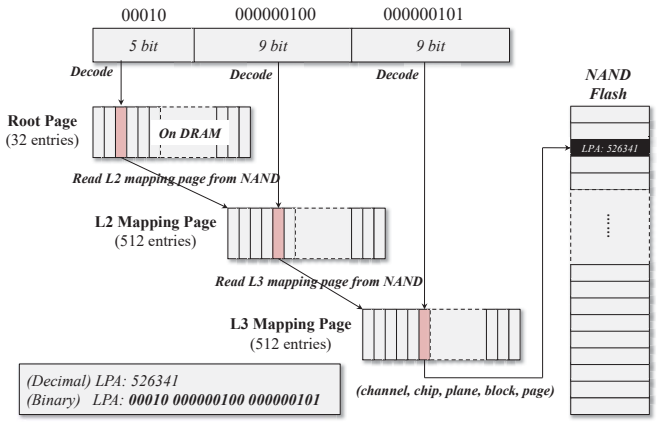

Figure 2. An example of the paging-like address translation.

To clearly illustrate the proposed paging-like address translation mechanism, an example is presented in Figure 2 to explain the flow on looking up a physical address. In the beginning, the logical page address (LPA) of a read/write request will be divided into three segments and presented in binary. The size of these three segments are 5-bit, 9-bit, and 9-bit respectively. In this example, if the LPA of a read/write request is 526,341 and its binary representation is 00010000000100000000101. This LPA will be divided into three segments with the binary values 00010, 000000100, and 000000101 accordingly. Then, the 3^{rd} entry (i.e., entry 00010) in the root page should store the physical address that stores the pointer pointing to the next-level mapping page, i.e., the L2 mapping page, where the root page permanently resides on RAM so the access cost on this page is negligible. Next, as the L2 mapping page is loaded to RAM (or it has been loaded to RAM in the previous address translation), the 5^{th} entry (or entry 000000100) in this mapping should store the physical address that contains the pointer pointing to an L3 mapping page. Finally, as the L3 mapping page is loaded onto RAM, its 6^{th} entry (or entry 000000101) should point to the target page storing data of LPA 526,341.

In the above example, we assume that the page size is 2KB, the size of an entry used to store one physical address is 4B; thus, the maximum number of entries in each page is 512. In this way, the number of required bits to represent the location of each entry in a page is 9. This is why the size of the two segments for indexing L2 and L3 mapping pages are both 9. Since the used number of entries in the root page is 32, the maximum size of the SSD capacity indexed by this example is 16GB ($32 \times 512 \times 512 \times 2KB$). Note that, any selection of the above parameters (e.g., the number of used entries in a mapping page or a root page) should depend on the specification of the considered SSD, and should be compatible with the proposed SPOR design that synchronizes these mapping pages timely and efficiently.

2.3 Active/Passive Synchronization Policy for On-RAM Data

In order to ensure the consistency of the metadata (including the address translation information) between RAM and flash memory, the active and passive synchronization policies are proposed to write the on-RAM data back to the flash memory in different ways so as to avoid crashing any metadata after a sudden power-off occurred. The passive synchronization is activated passively and inevitably when some on-RAM data must be written back to flash memory due to insufficient RAM space, and the active synchronization is used to actively synchronize the on-RAM data back to flash memory on a regular basis. In addition, the frequency of

[1] Although the platform used to implement an SSD prototype has a large memory size, i.e., 2GB, we only use partial memory in the emulation for the cost consideration.