An Adaptation Layer for Hardware Restrictions of Quadruple-Level Cell Flash Memories

Se Jin Kwon
Department of AI Software
Kangwon National University
Samcheok, South Korea

***Abstract*—In recent years, major flash memory vendors have produced SSDs and fusion memories as substitution for hard disks. However, there has been a lack of studies on access restriction of QLC flash memory, since most researches have targeted small capacity flash memory. As a solution, we propose to implement an adaptation layer between the file system and FTL (Flash Translation Layer). Instead of immediately writing data given from file system to flash memory, the adaptation layer gathers and adjusts data in the unit of a page, and separates random data from sequential data. By implementing the adaptation layer, previous FTL algorithms can be fully applied on the QLC flash memory. According to our experiment, the adaptation layer forms smaller number of pages than the current data gathering algorithm.**

Keywords*—Cache storage; flash memory; SSD; nonvolatile memory

I. Introduction

The capacity of flash memory has been rapidly growing as it has been introduced as a new solution for substituting the hard disks. Current QLC flash memories such as SSDs and fusion memories contain pages that are four to eight times larger than file system's data sector [1]. Due to the large page size of QLC flash memories, it is required to re-access a page to write multiple file system's data sectors within a page. However, the number of partial programming (NOP) within a page is limited to only one to avoid program-disturb errors [2]. Therefore, QLC flash memory uses an internal buffer to gather data in a unit of a page before writing onto the flash memory.

Unfortunately current well-optimized FTLs do not contain data gathering algorithm for NOP restriction, since they are designed for small capacity flash memories [3]. In this paper, we are not concerned with developing efficient mapping algorithm, since the small capacity based FTL algorithms already give various solutions. Instead, we propose to implement an adaptation layer between file system and FTL. It enables the small capacity based FTL algorithms to be fully applied on the QLC flash memory. Instead of immediately writing data given from the file system to the flash memory, the adaptation layer gathers the data sectors and rearranges them suitably for the QLC flash memory.

II. Related Work

Fig. 1 shows the overall architecture of large capacity flash file system. The file system issues write commands along with logical sector numbers and data. In case of small capacity flash memories, the given logical sector number (LSN) is directly converted into a physical sector number of flash memory by the mapping algorithm provided by FTL [4]. However, in case of QLC flash memory, the following problem should be considered.

A. Problem Definition 1

In QLC flash memory, a page size is four to eight times larger than file system's data sector, although the NOP allowed within a page is only one [6]. With restricted NOP, the flash memory does not allow any additional access to a page [7]. Therefore, the QLC flash memory requires a data gathering algorithm which gathers data in the unit of a page before writing onto the flash memory.

The current basic data gathering algorithm [5] is used to gather data with same logical page number (LPN). When the file system issues a write command as "w LSN data: write data in the logical sector (LSN)", the basic data gathering algorithm calculates LPN by dividing LSN with the number of sectors per page. Each write command's data are collected in the buffer until a write command with different LPN appears.

Fig. 2 shows an example of the basic data gathering algorithm. In this figure, we assume there are four sectors within a page and one NOP per page. w 0 A, w 1 B, and w 2 C are gathered into one page, since all of them belong to LPN 0 (=0/4, =1/4, =2/4). However, when w 9 D (LPN 2 =9/4) occurs, the data in the buffer is sealed as a page and is written onto the flash memory. Finally the buffer is flushed and data D is written onto the emptied buffer. Likewise, other write commands are performed.

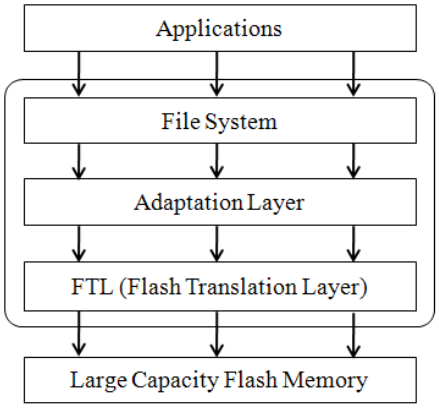

Fig. 1. Architecture of Flash File System.

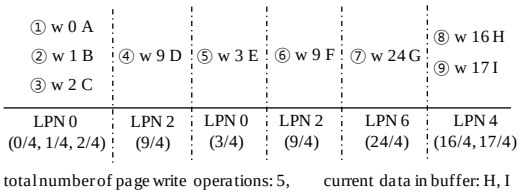

Fig. 2. Basic Data Gathering Algorithm.

The basic data gathering algorithm generates a total of five page write operations in Fig. 2, although there are only nine write commands. If each page were fully filled with data, nine write commands should generate only two to three pages since each page can store four write commands. As a solution, we propose to implement an adaptation layer between file system and FTL. The adaptation layer considers the following problem to fully gather data.

B. Problem Definition 2

The adaptation layer is required to separate random data from the gathering page. A file consists of sequential data and random data. The random data refers to file's meta data in which its LSNs are irregularly allocated and its data is frequently updated [3]. The irregular LSN allocation refers to the fact that the random data's LSNs are unlikely to be relevant to nearby LSNs. For example, in Fig. 2, the data corresponding to LSN 9 and LSN 24 are random data. Due to the update of LSN 9 and irregular allocation of LSN 24, the gathering page is sealed as a page whenever write commands with LSN 9 or LSN 24 appear. In order to solve Problem Definition 2, the adaptation layer contains an undefined buffer and two RAM pages: sequential and random.

Definition 1: The undefined buffer is an instant buffer which stores the write command that cannot be immediately decided as random or sequential. The adaptation layer requires maximum of two previous LSNs for decision; therefore, the capacity of undefined buffer is two sectors.

Definition 2: The sequential RAM page (SRP) and random RAM page (RRP) store the write commands that are defined as sequential and random respectively. The size of each RP is one page.

Each write command's LSN and its corresponding data is dynamically inserted into the undefined buffer or one of two RPs depending on the following algorithm.

1) Is the write command sequential to SRP?
2) Is the write command an update?
3) Analyze the write command with other LSNs of the undefined buffer.

First, the adaptation layer checks whether the write command belongs to SRP (<1>). If the write command does not belong to SRP, the adaptation layer checks whether the write command is an update of previous write commands. If the write command is an update, it is inserted into RRP since the update is one of characteristics of random data. However, when the write command does not belong to <1> or <2>, it is analyzed with other LSNs of the undefined buffer. The main role of <3> is to define LSNs with irregular LSN allocation as random data. The data gathering algorithm of adaptation layer is explained in detail in Section III.

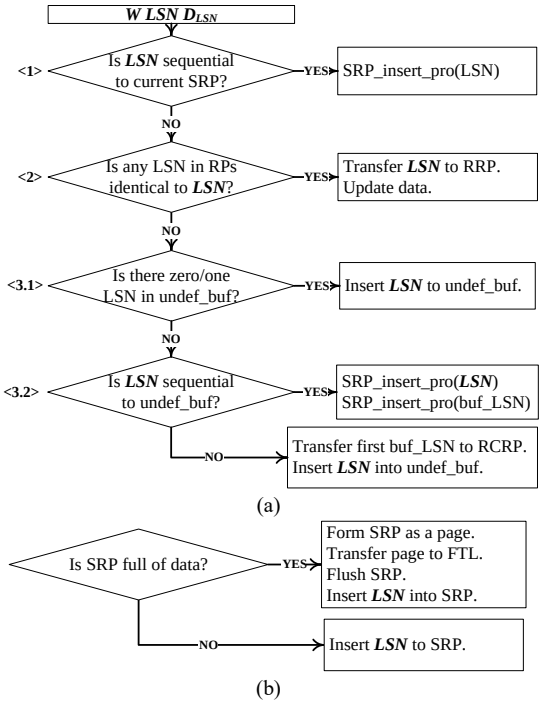

Fig. 3. (a) Data Gathering Algorithm, (b) SRP Insertion Procedure.

III. Data Gathering Algorithm of Adaptation Layer

When the system is initiated, there is no data stored in SRP or RRP. The adaptation layer requires previous LSNs to analyze the pattern of LSNs. Therefore, the adaptation layer simply accumulates the write commands with same LPN until a write command with different LPN appears. Finally the adaptation layer stores the write command with different LPN in the undefined buffer and finishes the initialization.

Example (Fig. 4(a)): For convenient understanding, we assume there are only four sectors per page. We have arbitrarily written LSN and its corresponding data within each sector as (LSN, data). Initially there is no data in SRP or RRP. (0, A), (1, B), and (2, C) are stored in SRP, since all of them belong to LPN 0 (=0/4, = 1/4, =2/4). (9, D) is stored in the undefined buffer since it belongs to LPN 2 (=9/4).

The write commands subsequent to the initialization follow the data gathering algorithm as shown in Fig. 3. Fig. 3 is a detailed view of the data gathering algorithm aforementioned in Section II. Each procedure of Section II corresponds to the procedure of Section III respectively except <3>, which is described in two parts (<3.1> and <3.2>) in Fig. 3. When the file system issues a write command, the adaptation layer checks whether the LSN is sequential to the SRP as shown in <1> of Fig. 3(a). The write command's LSN is decided as the sequential data when the differential between the write commands' LSN and the last LSN of SRP equals to the differential between two immediate last LSNs of SRP.

Example (Fig. 4(b)): w 3 E is sequential to SRP, because both result of LSN 3-LSN 2 and LSN 2-LSN 1 equals to one.

When a LSN is sequential to the SRP, the adaptation layer searches an empty sector within the SRP as explained in Fig. 3(b). If the SRP contains an empty sector, the LSN and its corresponding data can be directly inserted into the SRP. On the other hand, if the SRP is full of data, the data in SRP is sealed as a page, and it is sent to the FTL to be written onto the flash memory. Finally the SRP is flushed, and the LSN and its corresponding data are written to the SRP.

When a LSN does not belong to <1>, the adaptation layer checks whether the write command's LSN has previously appeared in the undefined buffer or RPs as explained in <2> of Fig. 3(a). If an identical LSN exists, the LSN's corresponding data is defined as the random data because one of random data's characteristics is the frequent update as mentioned in Section II. Therefore, the LSN is transferred to the RRP, and its corresponding data is updated.

Example (Fig. 4(c)): When w 9 F is issued from the file system, the adaptation layer searches for LSN 9. Due to (9, D), LSN 9 and its data F are written to the RRP and old data D is deleted.

If the write command's LSN does not belong to <1> or <2>, the adaptation analyzes the pattern of trace by comparing the write command's LSN with other undefined LSNs. Our algorithm requires two previous undefined LSNs for analysis; therefore, the LSN is temporarily stored in the undefined buffer (undef_buf) when there are less than two undefined LSNs as explained in <3.1> of Fig. 3(a).

Example (Fig. 4(d)): w 24 G does not belong to <1>, because LSN 24-LSN 3 does not equal to LSN 3-LSN 2. It does not belong to <2>, because there is no identical LSN in the undefined buffer or RPs. Thus, w 24 G must be compared to other undefined write commands. Unfortunately there is less than two LSNs in the undefined buffer so (24, G) is just stored in the undefined buffer. With same reason, next write command, w 16 H, is also stored into the undefined buffer.

When the undefined buffer contains two undefined LSNs, the adaptation layer checks whether the write command's LSN is sequential to them or not as shown in <3.2> of Fig. 3(a). The write command's LSN is considered as sequential, if the differential between the write commands' LSN and the last LSN of undefined buffer equals to the differential between two immediate last LSNs of undefined buffer. If the LSN is sequential to the undefined buffer, the undefined LSNs and write command's LSN are inserted into the SRP.

On the other hand, if the write command's LSN is not sequential to the undefined buffer, the first LSN and data of the undefined buffer is transferred to RRP, and the write command's LSN is newly inserted into the undefined buffer.

Example (Fig. 4(e)): When w 17 I is issued from the file system, the undefined buffer contains two LSNs: (24, G) and (16, H). The adaptation layer checks whether w 17 I is sequential to the undefined buffer or not. w 17 I is not sequential to the undefined buffer, because LSN 17-LSN 16 does not equal to LSN 16-LSN 24. In this case, (24, G) is transferred into RRP, and LSN 17 and its corresponding data I is newly inserted into the undefined buffer.

In Fig. 4(e), we have defined (24, G) as random data, even though LSN 24 has not appeared before. The adaptation layer defines the first LSN of the undefined buffer as random data due to the characteristic of irregular LSN allocation. The first undefined LSN is not sequential to the SRP and it does not have any chance of being a portion of sequential data in future. For example, (24, G) is not sequential to the SRP, and it is not sequential to next two commands: (16, H) and (17, I). On the other hand, second undefined LSN, LSN 16, remains in the undefined buffer, since it still has chance of being a portion of sequential data depending on the next write command (w 18 J).

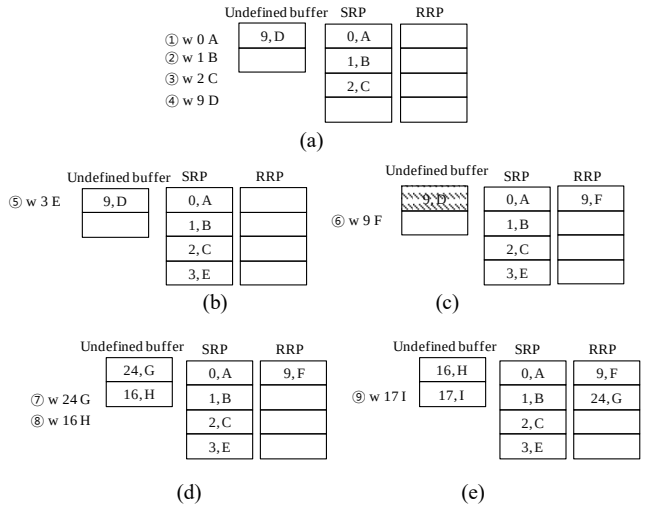

Fig. 4. (a) An Example of Initialization, (b) An Example of <1> of Fig. 3(a), (c) An Example of <2>, (d) An Example of <3.1>, (e) An Example of <3.2>.

TABLE I. NUMBER OF PAGES GENERATED BY PBM AND APRA

trace	total input	Basic data gathering algorithm (χ)	Adaptation layer (y)	difference (χ-y)
A	12,363,602	1,558,096	1,535,448	22,648
B	15,084,489	1,938,608	1,903,344	35,264
C	40,220,118	20,561,211	17,225,971	3,335,240
D	42,558,072	25,370,754	21,004,710	4,366,044
E	4,717	1,808	627	1,181
F	5,110	1,737	428	1,309
G	69,575	6,928	4,993	1,935
H	18,899	3,334	1,673	1,661

IV. Performance Evaluation

In this section, we have implemented our adaptation layer and compared it to current basic data gathering algorithm. We have analyzed the number of pages formed by each algorithm with the traces retrieved from various devices. Both algorithms are simulated on 256 Gbyte SSD, which consists of eight sectors per page and one NOP per page.

According to Table I, the adaptation layer forms smaller number of pages than current data gathering in overall environments. The adaptation layer has reduced over twenty thousand page write operations, and has reduced approximately one thousand page write operations in embedded devices. As we expected, separating the random data from the gathering page has fully filled pages with data, thus significantly reducing total number of page write operations. On the other hand, the basic data gathering algorithm formed many pages with empty sectors, because the page is likely to be sealed as a page whenever the random data interferes.

V. Conclusion

In this paper, we have dealt with the NOP restriction property of QLC flash memory. We have proposed to implement an adaptation layer between file system and FTL. It gathers and adjusts data in a unit of page so that small capacity based FTLs can be implemented on FTL without considering the NOP restriction. Furthermore, it separates random data from the gathering page, in order to reduce the number of page write operations. According to our experiment, the adaptation layer forms smaller number of pages than the current basic data gathering algorithm.

References

[1] MICRON Electronics, “Cache Programming Operations,” MICRON Electronics Technical Notes, 2022.

[2] Li-Pin Chang, “A Hybrid Approach to NAND-Flash-Based Solid-State Disks,” IEEE Transactions on Computers, 2010.

[3] Samsung Electronics, “QLC SSD,” 2022.

[4] Tatsuo Shiozawa, Hirotsugu Kajihara, Tatsuro Endo, and Kazuhiro Hiwada, “Emerging Usage and Evaluation of Low Latency FLASH,” 2020 IEEE International Memory Workshop (IMW), 2020.

[5] Mamoru Fukuchi, Shun Suzuki, Kyosuke Maeda, Chihiro Matsui, and Ken Takeuchi, “BER Evaluation System Considering Device Characteristics of TLC and QLC NAND Flash Memories in Hybrid SSDs with Real Storage Workloads,” 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021.

[6] Yoshiki Takai, Mamoru Fukuchi, Reika Kinoshita, Chihiro Matsui, and Ken Takeuchi, “Analysis on Heterogeneous SSD Configuration with Quadruple-Level Cell (QLC) NAND Flash Memory,” 2019 IEEE 11th International Memory Workshop (IMW), 2019.

[7] R. Mativenga, J.-Y. Paik, J. Lee, T. S. Chung, and Y. Kim, “RFTL: Improving performance of selective caching-based page-level FTL through replication,” Cluster Comput., vol. 22, no. 1, pp. 1–17, 2019.