Compound Mapping and Filter Algorithm for Hybrid SSD Structure

Jin-Young Kim[1]
Department of Computer Engineering
Kangwon National University
Samcheok, South Korea

Se Jin Kwon[2]*
Department of Computer, Media, Industry Engineering
Kangwon National University
Samcheok, South Korea

***Abstract*—With the recent development of byte-unit non-volatile random access memory (RAM), various methods utilizing quad level cell (QLC) not-AND (NAND) flash memory with non-volatile RAM have been proposed. However, tests have shown that these hybrid structures lead to a reduction in the performance of a hybrid solid state disk (SSD) owing to issues regarding space efficiency. This study proposes a compound address method and filter algorithm suitable for the next generation of NAND flash, called hybrid storage media, where QLCs and phase-change memory (PCM) are used together. The filter-mapping algorithm includes a management method that stores data in phase-change memory or flash memory according to the next command, which is accessed when a write command that is half or less than half a page in length is received from the file system. Tests have shown that the compound mapping and filter algorithm reduces the wasted pages by more than half and the number of merge operations is also significantly decreased. This leads to a decrease in the number of delete operations and improves the overall processing speed of the hardware.**

Keywords*—Pram; hybrid architecture; QLC NAND flash memory; algorithm

I. Introduction

There have been rapid changes affecting the memory layer in recent years with the development of byte-unit non-volatile (NV) RAM (phase-change memory). Phase-change memory (PCM) is similar to not-AND (NAND) flash memory, but also includes fast read and write operations, which are characteristics of main memory units. Moreover, its lifespan is approximately 10 times higher than that of NAND flash memory. Key examples of NVRAM include ferroelectric RAM (FeRAM), phase-change memory, and resistive RAM (ReRAM). A hybrid solid state drive (SSD) includes a flash translation layer (FTL), which is a software layer used to efficiently exchange information between hardware components by considering the hardware properties of the phase-change memory and NAND flash memory [1].

The existing hybrid SSD [2, 3] categorizes data into hot and cold data depending on the reading and writing frequency, and then stores high-frequency hot data and metadata into phase-change memory and stores low-frequency cold data into flash memory. When commands are given in duplicate locations it is possible to overwrite the phase-change memory and thus, reduce the number of merge operations in the flash memory to achieve the ultimate goal of improving overall performance. These existing hybrid SSD structures have the disadvantage of a reduction in overall space utilization efficiency because if a write operation delivers less than one page (a write unit) of data from the file system, the entire page will not be filled. To resolve the above issue, this paper proposes a compound mapping and filter algorithm for a hybrid SSD structure. Hybrid filtering refers to an algorithm that differentiates and stores data in the proper memory unit using two types of chips. This filter can be implemented through a buffer, such as a DRAM or resistor. The algorithm conducts two major functions; first, it gathers short write commands and stores them on a single page to improve the space efficiency. When a write command is one half of 8 KB or less (namely, 4 KB or less), where 8 KB is the standard page size in QLC NAND flash memory, the write command information is stored in a hybrid filter to await the next command. If the next command is 4 KB or less and would be written on a different page, both the existing command in the filter and the new command are stored in the phase-change memory. Second, the number of merge operations is reduced through the hybrid filter, which improves the overall performance. When a command is present in the same sector as the command stored in the filter, so that the data must be overwritten, there is no need for a separate operation because the command information is immediately overwritten in the filter, unlike a log block system, in which free blocks must be allocated to the log block to merge the commands. This reduces the number of merge operations, thereby improving the overall performance.

Section 2 analyzes existing studies and analyzes limitations. Section 3 describes the newly proposed filter algorithm and its implementation examples. Finally, Section 4 analyzes the test results and presents future research directions.

II. Previous Studies

A. Information Update Connection

Existing FTL algorithms are categorized into 1:1 [4], 1:N [5, 6], and M:N [7, 8, 9] depending on the number of data blocks that are connected to a single logic block. A data block is where the data are first written, and a log block delays the merge operations as long as possible by recording the overwritten data in different locations according to each algorithm in the event of a store command involving overlapping pages. In the 1:1 connection, if a write command occurs on overlapping pages, a new log block is allocated

* Corresponding author

from the free blocks, and a duplicate sector is recorded in that block to delay the merge operation. However, because only one data block is linked to a single log block, if repeated write commands occur in the same page, merge operations occur much more frequently, thereby reducing the overall performance of the flash memory. In the 1:N connection, a total of N data blocks are linked to one log block. In other words, several data blocks can share a single log block. In addition, because they are generally used an "out-of-place" method that fills the space in any order, the space utilization efficiency is extremely high. However, in the worst-case scenario, data blocks equaling the number of a page will be connected to a single log block, and a significant delay will occur when conducting merge operations. In the M:N connection, this architecture attempts to overcome the disadvantages of a 1:N connection. The main concept is to limit the number of data blocks that can be linked to a single log block.

B. Limitations of Previous Studies

The algorithms for the connection schemes mentioned in Section II.A are difficult to implement in hybrid SSDs, or do not provide optimum efficiency when implemented. Regardless of the algorithm applied, if the size of a write command is eight sectors (4 KB) or less, at least half of the 16 sectors, which is the standard number for a QLC NAND flash memory page, will inevitably be wasted. If a sector mapping method is applied to resolve this phenomenon using only the NAND flash memory, it will require an extremely large memory volume in the main memory device. However, if a hybrid structure comprised of phase-change memory and NAND flash memory is applied, and a phase-change memory of a certain size is mapped based on sector units, a relatively small volume will be required instead.

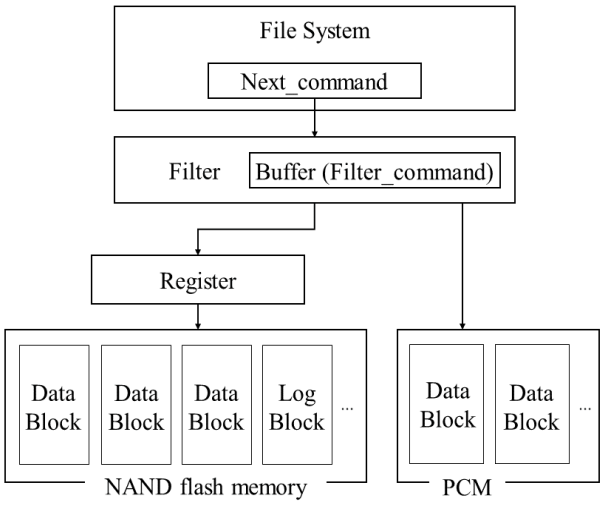

Fig. 1. Overall Structure.

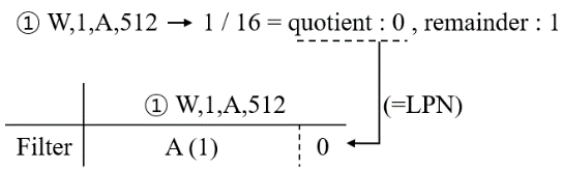

Fig. 2. Command Access and Storage in Filters

Algorithm 1

```
Algorithm 1
1:  if 'Next_command' is less than 4KB in size;
2:    if there is a 'Filter_command' in the filter;
3:      if ther LPN of the 'Filter_command' and the 'Next_command' are the same;
4:        if the LSN of the 'Filter_command' and the 'Next_command' are the same;
5:          In the filter, overwrite the 'Filter_command' with 'Next_command';
6:        else (=LSNs of two commands are not identical)
7:          'Filter_command' and 'Next_command' are stored in flash memory as one page;
8:      else (=LPN of 'Filter_command' and 'Next_command' are not the same);
9:        if the sector mapping table has the same LSN as 'Filter_command';
10:         overwrites 'Filter_command'
            in the data storage location of the same sector in the PCM;
11:       'Next_command' is stored in the filter;
12:     else (=same LSN does not exist in sector mapping table)
13:       the 'Filter_command' is stored in the new space of the PCM;
14:       'Next_command' is stored in the filter;
15:   else (= the filter is empty)
16:     'Next_command' is stored in the filter;
17: else (='Next_command' is larger than 4KB)
18:   'Next_command' is stored in the flase memory;
```

III. Proposed Method

A. Issue Analysis

Analysis of the traces of existing file system commands available in the UMass Trace Depository [10] indicated that 25% of all traces were not written chronologically. Of these, 24.7% were write commands of one half page size or less. Based on the characteristics of flash memory, if the next page is used after processing a write command, it is impossible to go back to the previously-written page. Pages with wasted sectors after writing fewer than eight sectors (4 KB) accounted for 7% of all write commands. In conclusion, only 26.8% of the volume in all blocks was used, indicating that approximately 73% of the total volume was wasted.

B. Filter Algorithm

To resolve the issues discussed above, a compound mapping and filter algorithm is proposed. The overall structure is shown in Fig. 1. If a command is given to the file system, the command is stored in the appropriate storage space of the PCM and NAND flash memory after passing through the filter algorithm area. The NAND flash memory has data blocks and log blocks. Data passes through the registers before being stored in these blocks. Finally, the PCM contains only data blocks. The general characteristic of flash memory and PCM is that one block consists of four pages and one page consists of sixteen sectors. In this architecture, PCM uses sector mapping and NAND flash memory uses block mapping. The filter algorithm is called a compound mapping because it uses both types of mappings.

For existing 1:N association algorithms, only data blocks and log blocks are used, which places a heavy burden on these two block types. This can result in a significant number of merge operations, shortening a device's lifetime and reducing its performance. To mitigate these issues, we added a new PCM storage space and filter area. The filter area identifies the data according to Algorithm 1 to be described later, and either selects the PCM or flash memory and stores the data in the most suitable device. This will reduce the merging workload and increase storage space efficiency, ultimately improving overall performance.

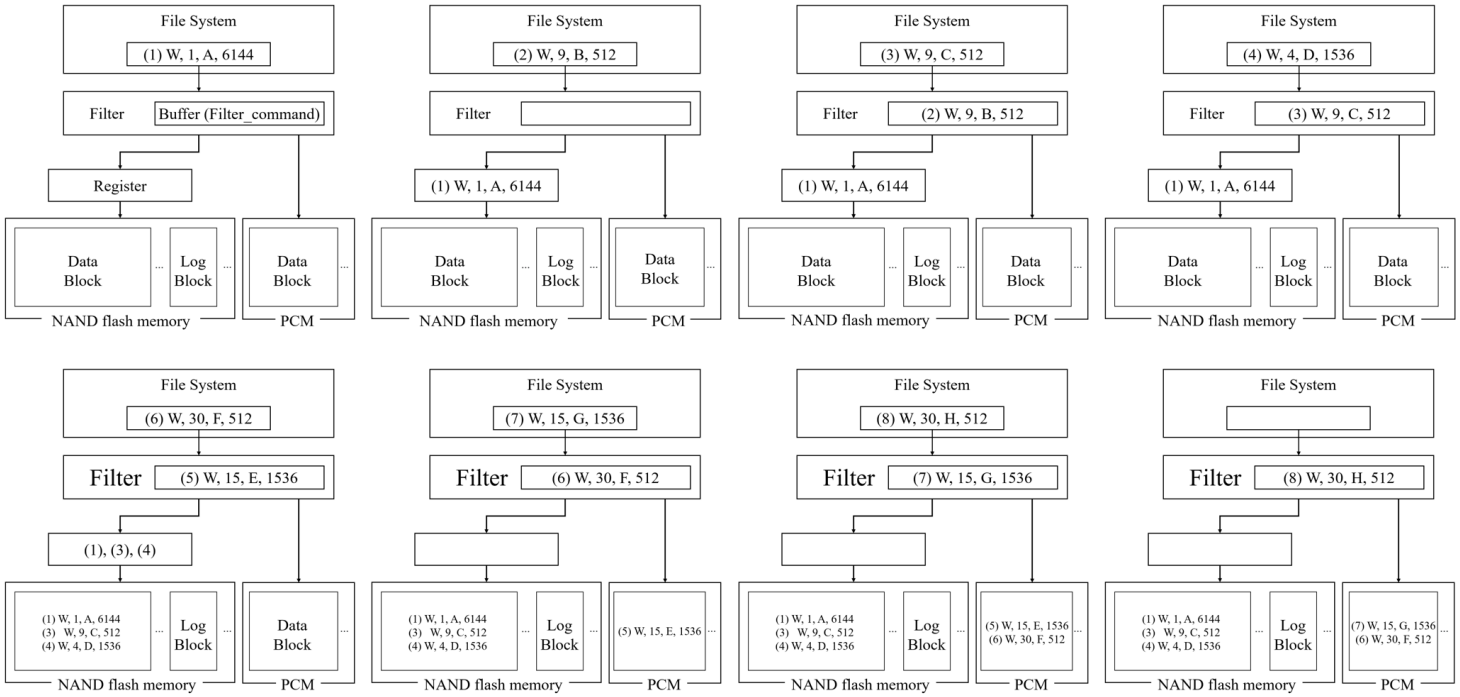

Fig. 3. Algorithm 1 Process with Examples (Proceeding Left to Right on the First Line).

As shown in Fig. 2, a command contains ‘command, logical sector number, data, size’ information. A ‘command’ is a command that runs in the file system as flash memory. ‘W’ means write and ‘R’ means read, but only ‘W’ is used because only write commands are needed. The ‘logical sector number’ is the number of the sector corresponding to the write command. ‘Data’ is the content to be saved and ‘size’ is the capacity of the write command. The unit of size uses bytes by default.

Given the data to process, divide the logical sector number (LSN) by the number of sectors per page to obtain the quotient, where the quotient is called the logical page number (LPN). The LSN and LPN will only reside in the DRAM in the filter area for a short time until the next command is given. The size of the filter in which the instruction can be stored for a short time is equal to the maximum capacity of the filter specified in Algorithm 1.

In order to simplify the filter, it is represented as a single page. In this paper, the LSN is recorded in parentheses for intuitive confirmation. On an actual system, the LSN is not stored in the filter.

For example, in Fig. 2, If LSN 1 is divided by 16, the quotient is 0 and the remainder is 1. Therefore, the LPN is written as 0 and data is written to sector 1 of the filter. This command is the ‘Filter_command’ in Algorithm 1, and a detailed description of this will be provided in the next paragraph. A ‘Filter_command’ will be saved to the PCM or flash memory and stored in the filter as determined by Algorithm 1 when processing the next command.

Algorithm 1 describes the overall processing of the filter and example commands are provided in Fig. 3. We used "OLTP Application I / O", a collection of I / O command information given to storage among the traces provided publicly in UMass Trace Repository [10]. In Algorithm 1, a 'Filter_command' implies that the command is already stored in the filter and a ‘Next_command’ refers to the command that is currently being processed.

The first item to check when given a command to process is the size of the command. If the size of the ‘Next_command’ is less than or equal to the maximum value that can be stored in the filter, verify that the filter has a ‘Filter_command’ already stored.

In the case of an instruction given as ‘(1) W, 1, A, 6144’ in Fig. 3, the size of the instruction is larger than 4096 B, so it does not go through the filter (Algorithm 1, lines 17–18). This is because when a trace is analyzed, very few consecutive write commands that exceed the maximum size of the filter appear in the same sector.

When a scenario occurs in which a command is to be stored in flash memory, the filter collects as many identical page commands into registers as possible before they are stored in flash memory. If the page currently being collected in the register is equal to the LPN of the command or if the register is empty, the command is stored in this register. This includes the processing of the ‘(3) W, 9, C, 512’ and ‘(4) W, 4, D, 1536’commands, for example. However, if the page number that is collected in the register differs from the LPN in the next instruction or if the next instruction causes a register overflow, the data of the existing register is stored in the flash memory before the next instruction is stored in the register.

If a scenario occurs that saves a command to a filter, such as processing the ‘(2) W, 9, B, 512’ command, the command can be saved to the filter immediately if the filter is empty (lines 1 and 15–16).

If a ‘Next_command’ must be saved in the filter (lines 3–7), but the filter is not empty (a ‘Filter_command’ is already stored in the filter), compare the LPNs of both commands (line 3). If the LPN is the same, the LSNs are compared again. If the LSNs are the same, the filter is overwritten (lines 4–5). In the figure, the command ‘(3) W, 9, C, 512’ would overwrite command ‘(2) W, 9, B, 512’.

If the LSNs are not the same, two write commands, such as the ‘(3) W, 9, C, 512’ and ‘(4) W, 4, D, 1536’ commands, are stored in the flash memory on the same page and the filter state changes to empty (lines 6–7).

When the ‘(6) W, 30, F, 512’ command is given as the ‘Next command’, the LPNs of the ‘Filter command’ and ‘Next command’ are different (line 8). Therefore, the sector mapping table is referred to and the PCM checks whether this is the same sector as the ‘Filter command’. In the current situation, because the PCM is empty, there is no identical sector, so the ‘Filter command’ is stored in the PCM and the ‘Next command’ is stored in the filter (lines 12–14).

When the command ‘(7) W, 15, G, 1536’ is given, the PCM is not empty but there is no same sector for the filter command, so data ‘F’ corresponding to the ‘Filter command’ is newly saved in the PCM. However, if the ‘(8) W, 30, H, 512’ command is given, the PCM overwrites existing data 'E' with filter command data ‘G’ because this is the same sector as that of the filter command. If the algorithm used were the 1:N association, it would have already used a significant amount of log block space due to overwriting.

C. Example Execution and Limitations

Fig. 4(a) and (b) show the results of the 1:N algorithm and the proposed filter algorithm performed on the same command. The command is one of the "OLTP Application I / O” commands publicly available from the UMass Trace Repository used for performance evaluations [10].

In the results analysis of Fig. 4, when running the compound mapping and filter algorithm, three pages were used for NAND flash memory and 55 sectors were used for PCM, resulting in 52,682 B. However, using 1:N concatenation, 10 pages were used for the data block and 13 pages for the 188,416 B log block. As a result, the space utilization efficiency of the filter algorithm is three times higher than for the 1:N association algorithm. In addition, the 1:N association algorithm wastes approximately 17 times more space than the filter algorithm.

Compared to the 1:N connection algorithm, not only does the compound mapping algorithm conduct much fewer merge operations, its use of partial sector mapping greatly improves the space utilization. Fig. 4(a) shows the processing of a command by this method. At a glance, the space utilization efficiency and the data storage density are much higher than the conventional 1:N association algorithm shown in Fig. 4(b).

Use of the conventional 1:N association algorithm results in many page allocations, as shown in Fig. 4(b). However, the amount of data that is actually stored in this space is very small, resulting in wasted capacity and lower space utilization efficiency.

The filter algorithm can store data on a sector-by-sector basis, and data of less than half a page (4 KB) can be algorithmically executed in the phase-change memory, where it is possible to overwrite data immediately when a write command occurs for the same position, and data are managed using sector-by-sector mappings. Complex flash memory mapping can be accomplished through a block-mapping application. With this approach, redundant sectors, which account for 79% of all traces, can be effectively managed.

However, from a cost point of view, there is a limit to the capacity of the phase change memory because this memory is expensive. Therefore, a small amount of space should be allocated to maximize the cost efficiency of the phase change memory. A minimum amount of space should also be allocated for the merge operator because if data should be stored in the phase change memory, but its space is not sufficient or the amount of invalid data that can be overwritten is too low, a merge operation will be performed. That is, since the size of the phase change memory is small, the number of merging operations increases. Therefore, the cost of the merge operator should be minimized.

As limitations, the volume used in the sector-mapping table is relatively large, and the cost of the phase-change memory is high. Therefore, a means to reduce the size of the mapping table and at the same time the amount of phase-change memory that provides the greatest efficiency for each NAND flash memory capacity should be sought. It is also necessary to consider a more efficient method of merging and to check detailed conditions on how to exchange information between the flash memory and PCM.

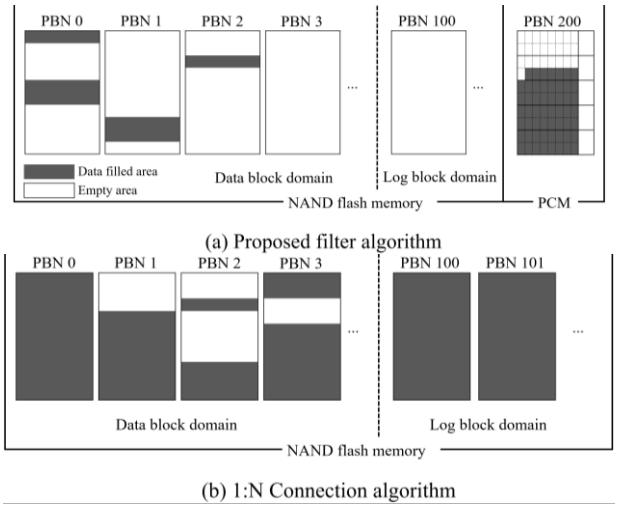

Fig. 4. Comparison of Space Utilization.

TABLE I. ESTABLISHMENT OF TEST HYPOTHESIS

Variable	*Value*
Number of blocks used in the test	2,048
Number of pages per block	64
Number of sectors per page	16
Sector size (Byte)	512

IV. TEST RESULTS AND DIRECTIONS FOR FUTURE RESEARCH

A. Test Results

This section compares and analyzes the efficiency of the proposed algorithm to a compound mapping filter algorithm that uses 1:N linking sectors based on traces, and measures the number of operations performed and the time needed to achieve the results. The consumed time for flash memory is assumed in the simulation by referring to a technical note provided by Micron Technology [11]. The time required for a random write per sector is 55 μs, and the time required for a block erasure is 500 μs. Also, trace analysis indicates that the average size of one write operation is 3584 B.

We used the "OLTP Application I / O", a collection of I / O storage command information for traces provided publicly by the UMass Trace Repository [10]. We analyzed the read / write command and the corresponding sector number and size, and conducted performance evaluations based on this command. Because QLC is still in the development stage, it was not possible to provide a hardware performance evaluation, so the evaluation was performed based on software coding.

As common characteristics for the two algorithms, one block is composed of 64 pages, and each page consists of 16 sectors, as shown in Table I. In both cases, the data block and log block domains use NAND flash memory. The 1:N connection algorithm uses a 1 GB data domain and a 10 MB log block domain; the filter algorithm was set up using a data domain of 1 GB, log block of 5 MB, and filter domain of 5 MB, where the filter domain used dynamic, static, or parameter RAM (DRAM, SRAM, or PRAM).

Table II shows the results of analyzing 378,914 write commands on a single chip. Here, a merge operation refers to merges between the data domain and log blocks in the NAND flash memory. For the 1:N connection algorithm, 1,112,738 write operations were required. This represents approximately 7,789,166 sectors (55 microseconds per sector), or approximately 428.4 seconds in total. On the other hand, the filter algorithm required 200.9 seconds because 522,012 sectors were involved. Therefore, using the filter algorithm, it is possible to reduce the number of write operations and their associated time by 46% compared with the conventional method. Erase operations also yielded significant differences. In the 1:N chain algorithm, 546 block deletion operations and 273 merge operations were performed. However, the filter algorithm applied only 34 block deletions and 17 merge operations. These numbers indicate that when using the filter algorithm, the numbers of delete and merge operations are reduced by 93% compared to the 1:N connections algorithm.

TABLE II. COMPARISON OF ALGORITHM

	Filter Algorithm	*1:N Connection Algorithm*
Write operations (number)	522,012	1,112,738
Write operations (seconds)	200.9	428.4
Delete operations (number)	34	546
Delete operations (seconds)	0.017	0.273
Merge operations (number)	17	273

B. Directions for Future Research

This study assumed that the filter will use DRAM or SRAM. However, such memory types are relatively expensive compared to PRAM, and hence the memory volume must be reduced as much as possible for greater cost efficiency. Therefore, a method that uses PRAM should be considered. PRAM has a slower access speed compared to DRAM or SRAM, and hence an algorithm that uses a two- or four-step pipeline technique must be designed to improve the speed.

To the two-step pipeline, two filters (Filter 1 and Filter 2) composed of eight connected sectors operate within the phase-change memory. After reading the write command in Filter 1, the write command is also read in Filter 2. Because differences in the delay time may occur depending on the input command, the filter that finishes its operation first will read the new command and process it according to the algorithm.

To further elaborate, if the domain in a phase-change memory uses a filter, and phase-change memory is used for storage, the filter is converted into the data domain immediately, and the eight sectors that are connected out of the extra domains in the phase-change memory will be used as a new filter. The algorithm described in this paper requires two operations when data are stored in the NAND flash memory or phase-change memory because of data passage through the filter. However, when PRAM is used, the filter is incorporated in the phase-change memory architecture, so only one write operation is needed to store data in the phase-change memory.

ACKNOWLEDGMENT

This work was supported by Basic Science Research through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A3B04031440). This study was also supported by a 2018 Research Grant from Kangwon National University (No. 000000000).

REFERENCES

[1] Ahmed Izzat Alsalibi, Sparsh Mittal, Mohammed Azmi Al - Betar, Putra Bin Sumari "A survey of techniques for architecting SLC/MLC/TLC hybrid Flash memory ‐ based SSDs," Practice and Experience, e4420, ISSN 1532-0626, 2018.

[2] Jung Sik Park, Hi-seok Kim, Ki-Seok Chung, and Tea Hee Han, "PRAM and NAND Flash Hybrid Architecture based on Hot Data

Detection," 2nd International Conference on Mechanical and Electronics Engineering ICMEE, 1, pp. 93-97, 2010.

[3] Jin Kyu Kim, Hyung Gyu Lee, Shinho Choi, and Kyoung Il Bahng, "A PRAM and NAND Flash Hybrid Architecture for High-Performance Embedded Storage Subsystems," EMSOFT '08 Proceedings of the 8th ACM international conference on Embedded software, pp. 31-40, 2008.

[4] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min and Yookun Cho, "A space-efficient flash translation layer for CompactFlash systems," IEEE Transactions on Consumer Electronics, 48(2), pp. 366-375, 2002.

[5] Sang-Won Lee, Dong-Joo Park, Tae-sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song, "A log buffer-based flash translation layer using fully-associative sector translation," Embedded Computing Systems (TECS), 6(3), pp. 1-27, 2007.

[6] Usman Anwar, Se Jin Kwon, and Tae-Sun Chung,"SAF: States Aware Fully Associative FTL for Multitasking Environment," Computer and Information Science 2015, 614, pp. 1-11, (2016)

[7] Dawoon Jung, Jeong-Uk Kang, Heeseung Jo, Jinsoo Kim, and Joonwon Lee, "Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme." ACM Transactions on Embedded Computing Systems, 9(4), pp. 40:1-40:41, 2010.

[8] Jeehong Kim,l authorDong Hyun Kang, Byungmin HaHyunjin Cho, and Young Ik Eom, "MAST: Multi-Level Associated Sector Translation for NAND Flash Memory-Based Storage System," Computer Science and its Applications, 330, pp. 817-822, 2015.

[9] Hyunjin Cho, Dongkun Shin, and Young Ik Eom, "KAST: K-Associative Sector Translation for NAND flash memory in real-time systems," DATE '09 Proceedings of the Conference on Design, pp. 507-51, 2009.

[10] UMass Trace Repository, "OLTP Application I/O" "http://traces.cs.umass.edu/index.php/storage/storage/"

[11] Micron Technology, Inc "NAND Flash 101: An Introduction to NAND Flash and How to Design it into Your Next Product", "https://user.eng.umd.edu/~blj/CS-590.26/micron-tn2919.pdf"