International Journal of Engineering & Technology, 7 (4.31) (2018) 381-385

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Optimising the Space Utilisation in Real-Time Flash Translation Layer Mapping Scheme

Mohd Bazli Ab. Karim[1]*, Amir Rizaan Rahiman[2], Rohaya Latip[3], Hamidah Ibrahim[4]

[1] *Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia*
**Corresponding author E-mail: bazli.karim@gmail.com*

Abstract

Solid-State Disk (SSD) is a semiconductor storage device and it has become a preferred choice for many storage sub-systems solutions to replace the classical hard drives due to its high performance and durability. Moreover, NAND flash memory has become cheaper in costs. However, this flash memory type has its own limitations due to its erase-before-write operations nature. This limitation will cause the memory to wear faster and consuming higher cost when initiating the cleaning process. To overcome the limitation, an address mapping in NAND flash memory namely Flash Translation Layer (FTL) plays important role in handling I/O operations. Several studies on the FTL have been carried out to manage the IO operations in NAND flash device efficiently. This paper proposed an optimized address-mapping scheme called Optimized Real-Time Flash Translation Layer (ORFTL). In order to increase the NAND flash space utilization, the proposed scheme reduces idle buffer blocks and reassigns the blocks as new Logical Block Addressing (LBA) in order to optimize blocks in flash memory for more space utilization. In addition, the scheme introduces a pool of buffer blocks with the same bandwidth throughput size of IO interface that connects the SSD to the host system in order to guarantee available free spaces to serve write operations. By optimizing both types of blocks, the proposed scheme has shown significant increases in the NAND flash memory space utilization as compared to the existing FTL schemes.

Keywords: *Semiconductor; Solid-state Disk; Flash Translation Layer; Erase-before-write.*

1. Introduction

Solid-State Disk (SSD), a semiconductor storage device has become the primary solution of the storage subsystem by replacing the classical hard drives in computer systems. This is due to NAND flash memory in the SSD has dropped in costs. Furthermore, it offers high performance and durability as compared to the hard drives since it does not suffer from the mechanical latencies as well as high tolerable to the shock and the vibration. Therefore, flash memory is suitable for wide range of uses ranging from portable to mobile devices even for primary storage solutions in datacenters [1], [2].

However, NAND flash memory has its own disadvantages. Due to the nature of erase-before-write characteristic, the memory could wear out easily and leads to its end lifetime as compared to the SATA hard drives. To increase its lifetime, many researchers and storage device manufacturers have come up with many solutions in order to manage the read and the write operations efficiently. One of the solutions is so-called Flash Translation Layer (FTL). The FTL consists of three major inter-related components (address translator, garbage collector and wear leveler) that play important roles to ensure the memory's lifetime get an increase. The idea behind all these components is to delay the erase operation (hereafter-called garbage collection) as much as possible due to the erase-before-write nature. In NAND flash memory, the garbage collection refers to the time costly operation that can disrupt the performance of the device. This is because the process needs to reclaim the invalid pages reside in a block to be erased. To do that, it has to count the valid pages that still exist, copy them to another free block and perform the erase operation. While the garbage collection initiated, targeted blocks will not be available until the process complete, thus increase the latency of write operations. To improve this latency, buffer blocks are used to guarantee the free spaces to serve the write operations [3], [4]. This will incur the cost of an SSD since additional flash memories are required to provide the buffer mechanism to improve the performance.

This paper proposed an address-mapping scheme named Optimized Real-Time Flash Translation Layer (ORFTL). It is a hybrid-FTL address-mapping scheme that reduces the static random-access memory (SRAM) size dependency in storing the mapping information between the Logical Block Address (LBA) and the Physical Block Address (PBA). Unlike existing hybrid-FTL types, by taking advantage of the limitation bandwidth throughput of IO interface, the proposed scheme reduces idle buffer blocks and reassigns the blocks as new LBAs in order to optimize the physical blocks in flash memory for more space utilization. Although all idle blocks have been reduced, the scheme introduces a pool of buffer blocks in order to guarantee available free spaces to serve write operations when the garbage collection process initiated on the existing block. The pool size of the buffer blocks is optimized as same as bandwidth throughput size of IO interface that connects the SSD to the host system. From the conducted simulation results, the proposed scheme has shown approximately 16% increases of space utilization as compared to the existing scheme when the idle buffer blocks are fully utilized for data block purpose. Results also shown that the space utilization of flash memory in various sizes of SSD devices ranging from 200GB to 1200GB has consistently increased compared to the existing scheme since the percentage of space capacity used for a pool of buffer blocks has decreased in various sizes of SSDs. The remainder of this paper has been organized as follows. The background and the related works of the FTL are discussed in Section II. Section III discusses the proposed work. Section IV discusses the simulation and results of the proposed scheme. Section V presents the summary and the future work.

2. Background and Related Work

This section presents the FTL architecture, its overheads and the types of the FTL. Then, we review the related work of the RFTL [3], [4], a well-known work related to the hybrid-FTL type recently.

2.1. FTL Architecture

The architecture of a common NAND flash storage device is shown in Figure 1. One of the important parts that build the storage device is the FTL. The FTL contains three main components namely i) Address Translator (used to manage the logical-physical address-mapping scheme), ii) Garbage Collector, and iii) Wear- Leveler. Another important part of the device is the Memory Technology Device (MTD). Depending on the NAND flash applications (e.g., USD stick, SSD, CompactFlash, MMCs etc.), the MTD may exist in the operating system layer or even in the FTL in order to interact with the memory device and only supports three simple operations; read, write and erase [5].

The address translator component provides an address abstraction between the LBA to the PBA. It hides the low-level system management from the high-level file systems during the IO operations execution. Meanwhile, the garbage collector reclaims the invalid pages reside in the block to be erased due to the erase-before-write characteristic. Due to the characteristic, the number of invalid pages will be increased substantially since there are many IO operations involved. Thus, decreases the available free pages in the device. Therefore, the erase operation is necessary for the device. However, the erase operation will be performed in block unit rather than page unit. Thus, in order to reclaim the invalid pages resides in the block to be cleaned, it calculates the valid pages that still exists, copy them to another block and only then perform the erase operation on the block unit and reclaim it. The wear leveler distributes evenly the erasure counts for all blocks in order to extend the lifetime of the NAND flash device. Without this component, it is possible that a group of blocks in the device facing higher erase counts and wear very quickly and thus reducing the storage capacity of the sub-system.

2.2. Overheads in the FTL

There are three main overheads of the FTL. First, is the erase-before-write. The write operation in NAND flash requires a time-consuming erase operation which degrades the overall IO performance of the device. Thus, the buffer block has been used in order to reduce or avoid the time-consuming erase operations. However, allocating the buffer block without limits will decrease the available space capacity in storing new data. Second is the out-of-place-updating policy in the memory device. There is no physical rewrite operation in any NAND flash memory device. Rewrite operation means the updating request will be stored to a new free page while the previous page will be marked as invalid or garbage. The garbage collector is being used to reclaim all invalid pages in the original block, move or copy the valid pages to new free pages in a new block, before erasing the original block and return it as a free block. This operation causes the latency of write operation in NAND flash device. The third overhead is the durability. High erasure counts on a block could wear the block and lead to its end of the device lifetime. Hence, it reduces the total space capacity of the device from time to time. Moreover, the device lifetime also will be degraded.

2.3. Types of FTL

The FTL comes in three types, i) page-FTL, ii) block-FTL and iii) hybrid-FTL. The page-FTL is a naïve FTL scheme that maps the every logical address request to the available physical address based on the page unit. The page is the basic accessing unit for both read and writes IO operations in the NAND flash device. Therefore, it requires a large size of SRAM to store the mapping information. Meanwhile, the block-FTL maps the logical address to the physical address in a block unit, an accessing unit for the erase IO operation. Although the size of the SRAM is being minimized, this type of FTL requires additional tasks in handling the IO requests. For instance the garbage collection process. It is a time-consuming process and needs to be initiated frequently, which can cause the degradation in the device performance. The hybrid-FTL utilizes both block and page units in mapping the logical sector request to the available physical sector. It is being introduced to overcome the mapping disadvantages of both page-FTL and block-FTL. However, the hybrid-FTL requires a slightly higher amount of the SRAM as compared to block-FTL type. Among these three FTL types, the hybrid-FTL solution is more suitable for SSD due to the SRAM capacity requirement and fewer overheads in performing the compulsory garbage collection. This motivates us to optimize the existing works of the hybrid-FTL in the SSD device.

2.4. Related Work in Hybrid-FTL Types

Several studies on the hybrid-FTL have been reported so far [3], [4], [6]–[15]. Among these, the Real-time Flash Translation Layer (RFTL) is a well known and the better hybrid-FTL solution at this point in time [3], [4]. It has been proposed to evenly distribute the garbage collection time cost and guarantee a near optimum worst-case response time [3], [4]. This is an address-mapping scheme that guarantees physical space to serve IO requests at any window time. The optimized garbage collection strategy was being introduced to enable the RFTL reclaims the available space and serve the IO request simultaneously without interrupting the device operations.

As illustrated in Figure 2, in the RFTL, the block-unit mapping is being used to map the logical block with the three types of physical blocks, namely i) primary, ii) replacement and iii) buffer blocks. The primary block is used to serve handle the normal write requests (writing new data) while the buffer block is used to store the pending writing request if the primary block is fully being occupied. The replacement block is being used to provide the space for reclaiming the primary block. These blocks periodically changing their tasks to guarantee the available free spaces in handling the write operations [3], [4]. Additionally, this FTL solution has improved the worst-case response time as well as the average-case system response time. On the contrary, this solution is quite expensive since the space utilization level that can be optimized is only 33% or one-third of the NAND flash raw capacity. Moreover, the RFTL scheme consumes extra memory space to guarantee access performance in handling the worst-case response time. Although the benchmark results of the block erase counts for the RFTL is ranging from other FTLs [4], but the result is still only capable of one-third of the device space. In other words, the RFTL could spend three times higher block erase counts as compared to the others FTL if the space utilization is taken into consideration.

The RFTL scheme requires additional two data blocks to in order to act as replacement and buffer blocks. To avoid blocked write process due to initiating the garbage collection process, buffer blocks are used to serve the incoming write operations [3], [4]. This would be advantageous if all logical blocks are being used when all the write requests fill-up the IO interface at any given time window. If not, then it would waste the blocks. These buffer blocks can be assigned for the logical block addresses in order to increase the available capacity in the NAND flash device. In addition, although the SSD was meant to become a block device replacement to the SATA hard drive, their performance still cannot be fully utilized due to the limitation of standard IO interface (6 Gb/s or less, depends on SATA types) that connect between the host and the SSD [16]. Due to this reason, in a worst-case scenario when all write requests fill-up this interface at any time window, only some part of logical blocks will serve the request and may require buffer block if garbage collector gets being triggered to reclaim the primary blocks. Meanwhile, the remaining buffer blocks will be just idle or free.

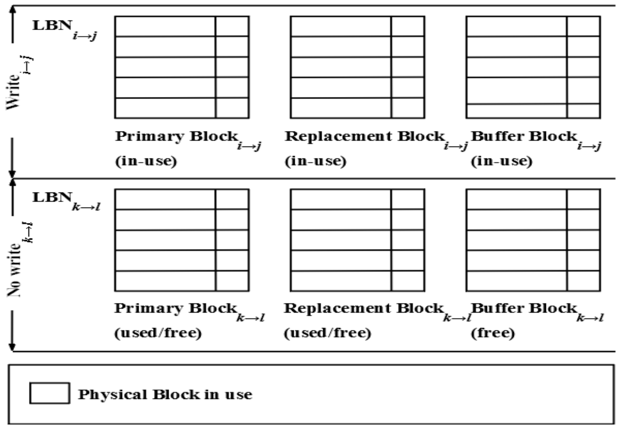
3. Optimized Real-Time Flash Translation Layer (ORFTL)

3.1. Motivation

According to the discussion in the previous section, we propose of allocating the buffer blocks based on the maximum throughput of the IO interface, rather than assigning one buffer block for each logical block founds in the device. In this approach, one logical block can be assigned with any buffer blocks from a pool of buffer blocks when the write request being issued from the file system, which gives two optimization opportunities. First, reduce the number of idle buffer blocks assigned to each logical block. Even if the number of writes operation reach the maximum throughput of the device IO interface, the numbers of idle buffer blocks still endure. Second, reassign the number of idle buffer blocks in order to create the more logical block. By considering the current space utilization in the RFTL, we can increase the space utilization for the proposed ORFTL if we reassign the idle buffer blocks for the more logical block. Thus, even in a worst-case scenario of write requests occurs at a given time window, the number of idle buffer blocks still exist due to the maximum throughput of IO interface of the SSD itself.

3.2. Utilizing the Unutilized Buffer Block

Generally, all the IO operation requests (both read and write) are being issued by the file system and will pass through the IO interface (the interface that connects the storage device to the host system). Now, the IO interface communicates at a rate of 6 Gbit/s. At any given time window, when the IO interface full with the write requests, only a small group of logical blocks will be used to serve all write operations. According to Figure 3, when all write$_{i\to j}$ operations are scheduled, depending on the LBN$_{i\to j}$ that has primary block$_{i\to j}$ with invalid pages, the buffer block$_{i\to j}$ will serve the write request, while waiting for the garbage collector to reclaim the primary block$_{i\to j}$. All valid pages in the block then will be copied into the designated replacement block$_{i\to j}$. The role between the primary block and the replacement block will be swapped when the erasure operation completed.



LBN$_{i\to j}$
Write$_{i\to j}$
Primary Block$_{i\to j}$ (in-use)
Replacement Block$_{i\to j}$ (in-use)
Buffer Block$_{i\to j}$ (in-use)
LBN$_{k\to l}$
No write$_{k\to l}$
Primary Block$_{k\to l}$ (used/free)
Replacement Block$_{k\to l}$ (used/free)
Buffer Block$_{k\to l}$ (free)
Physical Block in use

Fig 3: RFTL address mapping scheme – buffer block is used only when write operation is triggered.

Given at any time window wherein a worst-case scenario of all write$_{i\to j}$ operations filled up the interface, logical block$_{k\to l}$ are still idle. The primary block$_{k\to l}$ and replacement block$_{k\to l}$ could still have valid or invalid pages except for the buffer block$_{k\to l}$ where free pages are guaranteed to serve next write operations in case of garbage collector being triggered to reclaim the primary block. To do this, we assign each logical block mapped with two physical blocks, which are primary block and replacement block. However, in order to provide guarantee free pages to serve next write operations, a pool of buffer blocks is considered. The number of buffer blocks in the pool will depend on the capacity of IO interface of SSD. Two equations below describe the difference between both RFTL and ORFTL.

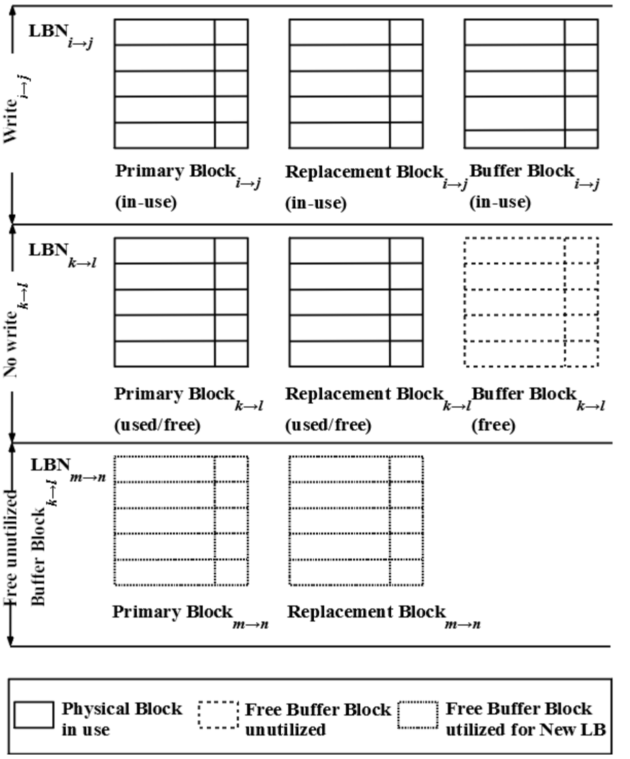
The number of LBA x, for the RFTL is being calculated as follows:

$$f(x)=\sum_{y=i}^{y=j} 3\times(PB)+\sum_{y=k}^{y=l} 3\times(PB)$$

(a) (b)

Parameter y is an index number of LBA while PB refers to the physical block being allocated for the LBA. (a) is the number of LBA to be fully utilized in a IO interface, (b) is the number of LBA that are idle when I/O operations take place on (a).

Meanwhile, Figure 4 describes the number of idle buffer block, the buffer block$_{k\to l}$ is being reclaimed from the idle logical block$_{k\to l}$ when in the worst-case scenario of all write requests fill-up the IO interface at any given time window. In return, the reclaimed buffer block$_{k\to l}$ will be reassigned to the new logical block$_{m\to n}$ to increase the space utilization for the proposed ORFTLThis paper proposed an address-mapping scheme named Optimized Real-Time Flash Translation Layer (ORFTL).



LBN$_{i\to j}$
Write$_{i\to j}$
Primary Block$_{i\to j}$ (in-use)
Replacement Block$_{i\to j}$ (in-use)
Buffer Block$_{i\to j}$ (in-use)
LBN$_{k\to l}$
No write$_{k\to l}$
Primary Block$_{k\to l}$ (used/free)
Replacement Block$_{k\to l}$ (used/free)
Buffer Block$_{k\to l}$ (free)
LBN$_{m\to n}$
Free unutilized Buffer Block$_{k\to l}$
Primary Block$_{m\to n}$
Replacement Block$_{m\to n}$
Physical Block in use
Free Buffer Block unutilized
Free Buffer Block utilized for New LB

Fig. 4: Utilizing the free buffer blocks for more space utilization for the ORFTL

Thus, the number of LBA x, for the proposed ORFTL is calculated as follow:

$$f(x)=\sum_{y=i}^{y=j} 3\times(PB)+\sum_{y=k}^{y=l} 2\times(PB)+\frac{\sum_{y=k=m}^{y=l=n} 1\times(PB)}{2}$$

(a) (b) (c)

Part (a) of the above equation is the number of LBA with a pool of buffer blocks to be fully optimized based on IO interface throughput bandwidth size, (b) is the number of LBA without buffer block when the I/O operations take place on (a) and (c) is the reduced buffer blocks taken from (b) and reassign for the new LBA.

4. Simulation and Results

4.1. Performance Metrics

To make the comparison of flash space utilization between the RFTL and the ORFTL, we performed the calculation of the logical block addresses. Table 1 summarizes the performance metric parameters used for the performance comparison. In most of SSD hardware specifications provided by manufacturers (for example in [1], [2], [17]), total physical blocks for flash memory information is not provided. Thus, two assumptions have been made in order to compare flash space utilization between RFTL and ORFTL schemes. First assumption is that user addressable sectors in LBA is calculated based on RFTL address mapping scheme. With this assumption, we calculated the total physical blocks of flash memory in SSD with their capacity ranging from 200GB to 1200GB. From this results, we later calculated the user addressable sectors in LBA with ORFTL address-mapping scheme. The second assumption is that the maximum throughput of IO interface is calculated at 6GB/s or 600MB/s where no latency factor is considered. With this assumption, we calculated the pool size of buffer blocks that will serve write operations when targeted blocks have garbage collection process running to claim invalid pages in the targeted blocks.

Table 1: Performance Metrics

SSD Capacity in Gb	200	400	800	1200
User Addressable Sectors in LBA	Information is based on [2] 200GB = 390,721, 968 400GB = 781,422,768 800GB = 1,562,824,368 1200GB = 2,344,255,968			
Physical Blocks per LBA	RFTL = 3, ORFTL = 2			
Max. IO Interface Throughput	6 Gb/s or 600MB/s			
Sector size	512 bytes			
Total sectors of Buffer Blocks based on IO Interface Throughput	1, 171, 875			

4.2. Results

As shown in Figure 5, the proposed scheme shows better improvement than the RFTL scheme. Given that the space capacity of flash memory in SSD ranging from 200GB to 1200GB, RFTL scheme can only provide 33% of space capacity. Meanwhile, ORFTL scheme can offer 16% more than RFTL. This is because the proposed scheme has considered the limitation bandwidth throughput of IO interface and reducing idle buffer blocks and assigned them as additional primary and replacement blocks of new LBAs in order to optimize the physical blocks in flash memory for more space utilization. In another view, result from Figure 5 also shown that in the range of 200GB to 1200GB of space capacity of flash memory in SSD, only small amount of space required and allocated for pool of buffer blocks to serve write requests when targeted blocks has garbage collection in progress. This buffer size is fix across various sizes of SSDs as long as the IO interface is same.

Meanwhile, Figure 6 illustrates the evaluation result for the required amount of space capacity for the buffer blocks in the ORFTL scheme. As can be seen, the required space capacity substantially declined when the capacity of the flash memory in SSDs is increased. This figure concludes that the proposed ORTFL scheme has fully optimized the capacity of the NAND flash device when the role of the assigned buffer blocks for the address-mapping are fully optimized.

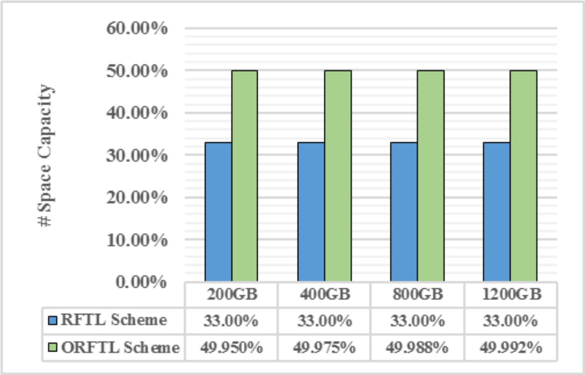


Fig 5: The space utilization requirement in address-mapping scheme.

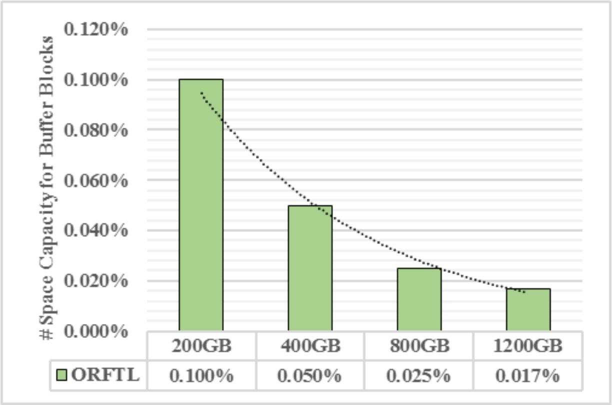


Fig 6: Space capacity used for the buffer blocks.

In Figure 7, it describes the space capacity reclaimed from the optimization of space in RFTL based on SSD capacity ranging from 200GB to 1200GB. As explained in earlier, an SSD consists of additional flash memories are required to provide the buffer mechanism to improve the performance. An SSD with RFTL in place will cost 3 times of amount of flash memory. However, not all buffer blocks will be in-use and remains idle because of the bottleneck of IO interface limitation. By taking advantage of this limitation, ORFTL reclaims these idle blocks and assigned them as additional primary and replacement blocks of new LBAs in order to optimize the physical blocks in flash memory for more space utilization.

As shown, only 6GB space allocated for buffer mechanism across every SSD size and the additional space utilization increase as the SSD size increase.

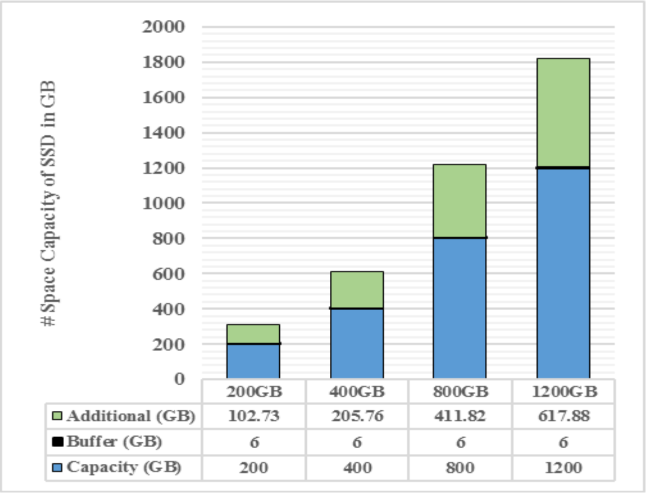


Fig 7: Space capacity of flash memory in SSD with ORFTL.

5. Conclusion

In this paper, we had proposed an optimized address-mapping scheme called the ORFTL. The scheme has shown significant improvement in space utilization as compared to the current scheme. By modifying the role of the assigned buffer block, the proposed scheme reduces the number of idle buffer blocks and utilized for these blocks for new logical block addresses. Moreover, the proposed scheme takes the advantage in utilizing the capacity of IO interface that connects the SSD to the host system. In the future, we will exploit the usage of the proposed address-mapping scheme with the garbage collector and wear leveller.

Acknowledgement

This work is supported by the Geran Putra (GP) GP/2018/9596300 of Universiti Putra Malaysia, Malaysia.

References

[1] "Intel Solid-State Drive DC S3700." Intel Corporation, Oct-2012.

[2] "Intel Solid-State Drive DC S3710 Series." Intel Corporation, Sep-2015.

[3] Zhiwei Qin, Yi Wang, Duo Liu, and Zili Shao, "Real-time Flash Translation Layer for NAND Flash Memory Storage Systems," in *Proceeding RTAS '12 Proceedings of the 2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, Beijing, China, 2012, pp. 35–44.

[4] Yi Wang *et al.*, "A Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 1, pp. 17–29, Jan. 2016.

[5] Jeffrey B. Layton, "Anatomy of SSDs," *Linux Magazine*, 27-Oct-2009.

[6] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, 2002.

[7] Chin-Hsien Wu and Tei-Wei Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," in *Proceedings of the 2006 International Conference on Computer-Aided Design*, San Jose, CA, USA, 2006, pp. 601–606.

[8] Sang-Won Lee, Won-Kyoung Choi, and Dong-Joo Park, "FAST: An Efficient Flash Translation Layer for Flash Memory," in *Zhou X. et al. (eds) Emerging Directions in Embedded and Ubiquitous Computing. EUC 2006. Lecture Notes in Computer Science.*, vol. 4097, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 879–887.

[9] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2006, pp. 161–170.

[10] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim, "LAST: Locality-aware Sector Translation for NAND Flash Memory-based Storage Systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, pp. 36–42, Oct. 2008.

[11] Hyunjin Cho, Dongkun Shin, and Young Ik Eom, "KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems," presented at the DATE'09, 2009, pp. 393 –398.

[12] Yi Wang, Duo Liu, Meng Wang, Zhiwei Qin, Zili Shao, and Yong Guan, "RNFTL: A Reuse-Aware NAND Flash Translation Layer for Flash Memory," presented at the LCTES'10, 2010, pp. 163–172.

[13] Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan, "MNFTL: An Efficient Flash Translation Layer for MLC NAND Flash Memory Storage Systems," in *Proceedings of the 48th Design Automation Conference (DAC)*, New York, NY, USA, 2011, pp. 17–22.

[14] Jalil Boukhobza, Pierre Olivier, and Stéphane Rubini, "A Scalable and Highly Configurable Cache-Aware Hybrid Flash Translation Layer," *Comput. 2014*, vol. 3, pp. 36–57, Mar. 2014.

[15] Sungjin Lee, Ming Liu, Sangwoo Jun, and Shuotao Xu, "Application-Managed Flash," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, Santa Clara, CA, USA, 2016, pp. 339–353.

[16] Myoungsoo Jung, "Exploring Design Challenges in Getting Solid State Drives Closer to CPU," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1103–1115, Apr. 2016.

[17] "M500 2.5-Inch SATA NAND Flash SSD." Micron Technology, 2013.