UDC 004.087.2

ANALYSIS OF THE NAND FLASH DEVICE GARBAGE COLLECTION ALGORITHMS UNDER LACK OF MEMORY CONDITIONS

I. ZAITSEV, S. ZALIVAKA
Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus

The paper presents the problem of high latency requests in SSD devices with a greedy garbage collection algorithm under low memory conditions. To solve this problem, other existing algorithms (RGA, Random, FIFO) are considered. The algorithms are implemented in the MQSim simulation environment. As a result, it is shown that in lack of memory conditions, the FIFO algorithm can reduce the latency of command executions by an average of four times compared to the greedy algorithm.

Nowadays, solid-state drives (NAND Flash SSD – Solid State Drive) are widely used in many computer systems, along with traditional hard disk drives (HDD – Hard Disk Drive). Solid-state drives have become widespread in comparison to traditional hard drives due to several advantages: faster access time, lower power consumption and compact package. It is also important that due to the lack of moving parts, solid-state drives have better vibration resistance. The disadvantages of solid-state drives include: a higher cost per bit of information storage, a lower throughput and latency during service procedures (e.g. garbage collection, wear leveling, etc.), and a smaller number of erase/write cycles for each block of memory [1].

SSD devices support three basic operations: read, write, and erase. The read and write operations work within the page, while the erase operation works within the block (block is a collection of pages) [2].

Each page in the SSD has one of three possible states: valid, invalid, and free/erased. The write operation can only be applied to a page in a free state. When a write operation is applied to a page, its state is changed from free to valid. Fig. 1 shows a case when data needs to be overwritten, which leads to the latency increase during a write operation. To optimize the write request, the data is written to a free block and the address mapping table (L2P – Logical to Physical Table) is updated. All pages of the old block are marked as invalid. With this optimization, all pages will gradually change their state from free to valid and invalid. The garbage collection procedure is used to restore invalid pages [2].

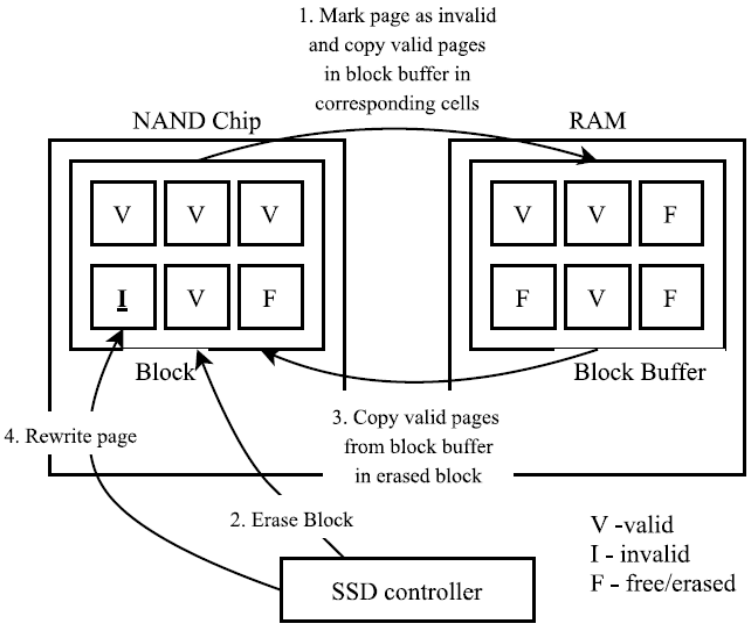

Fig. 1. – Page rewriting process

Garbage collection algorithm have such parameters as a threshold for the garbage collection procedure and a victim block selection policy. The threshold determines the number of free pages when the garbage collection procedure starts (triggered). The victim block selection policy determines which criteria are used to select a block to erase.

Garbage collection leads to significant latency overhead, so when garbage collection is performed, the delay in operational requests increases. In lack of memory conditions, the performance of an SSD device can significantly drop (at least 16 times, according to the modeling results shown in Fig. 2). Therefore, it is necessary to use an algorithm that would reduce the impact of the garbage collector on the device performance in such conditions.

This problem is typical for SSD devices with a greedy garbage collection algorithm. The purpose of the research is to find an algorithm that would reduce command execution latency in lack of memory conditions and compare it to the greedy algorithm performance.

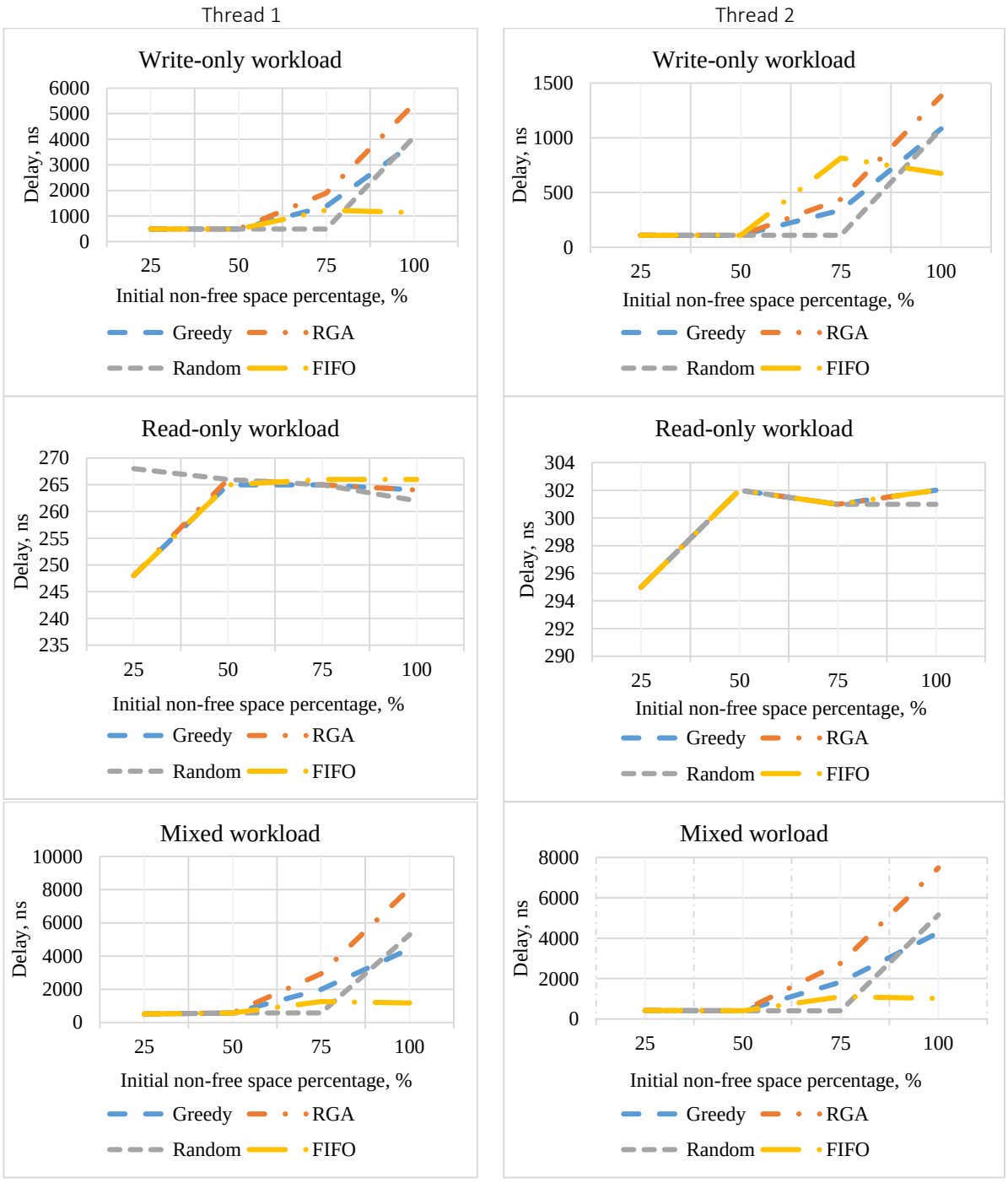

Fig. 2. – Delay values obtained during simulation of considered algorithms under different workloads

There are four garbage collection algorithms analyzed in this work:

1. **Greedy.** The block with the largest number of invalid pages is selected [3].

2. **RGA (Random Greedy Algorithm).** The block with the largest number of invalid pages is selected from a random set of blocks [4].

3. **Random.** A random block is selected [4].

4. **FIFO (First In – First Out).** A queue-based algorithm, i.e., each victim block is added to the queue when it is fully written. If the drive needs to perform a garbage collection procedure, the victim block is chosen as a result, i.e., the least recently written block [3].

The MQSim simulator is chosen as the environment for the experiment [2]. In this environment, the workload (the sequence of requests of the same or different types) can be reconfigured, the parameters that are responsible for the ratio of write and read requests, and the initial non-free space percentage of the disk can be set. Also, in this environment, it is possible to run simulations using multiple threads to execute requests. The simulation is performed with three types of workload: write commands only, read commands only, and mixed workload (read and write commands). In all cases, the number of requests is 1.2×10^5 and the two-threads mode is set.

Fig. 2 shows the measured delays for each algorithm. The first column shows the results for Thread 1, and the second column shows the results for Thread 2. The first row shows the results for write-only workload, the second row – for read-only workload, and the third row – for mixed workload (for Thread 1 read percentage is 80, for Thread 2 read percentage is 30).

Based on the results shown in Fig. 2, it is possible to make the following conclusions:

1. The read delay is approximately the same for different algorithms under conditions of varying degrees of free disk space.

2. Under write-only and mixed workloads the first three algorithms show almost the same behavior of the delay dependency despite of free disk space changes.

3. The FIFO algorithm can be described by a different behavior and shows the maximum delay value not when the disk is full. Then it shows almost a constant delay value, which is 5-6 times (on average) less than the value shown by other algorithms in lack of memory conditions. Based on the simulation results, FIFO algorithm shows the best performance in both cases, lack of memory conditions (disk is almost full) and free disk space around 25% or less.

Figs. 3 and 4 show the average number of page movements for GC (Garbage collection) and total number of GC executions for write-only and mixed workloads.

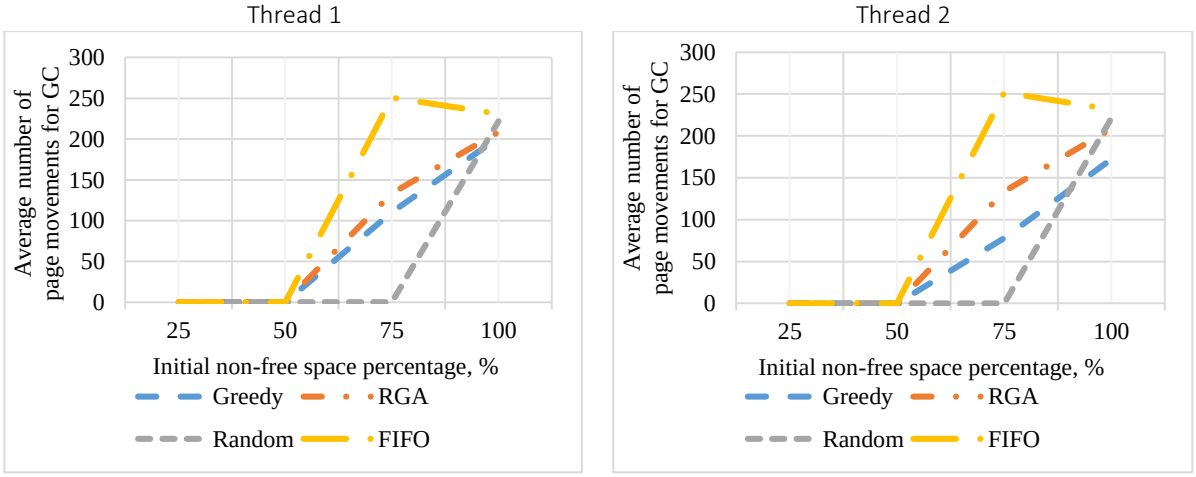

Fig. 3. – Average number of page movements for different garbage collection algorithms

Based on the obtained data, it can be concluded that:

1. The number of page movements and total number of GC executions are correlated to the delay value, i.e., with a decrease of free disk space, the write delay, number of page movements and total number of GC executions will increase. This can be explained by the fact that as the device has less free space, there are fewer free pages and the probability of getting a rewrite request increases, which leads to a search of a free block, the absence of which triggers the garbage collection procedure.

2. The conclusions described above (point 1) does not match the FIFO algorithm. Average number of page movements and total number of GC executions are almost equal after reaching the minimal value (25% of free disk space).

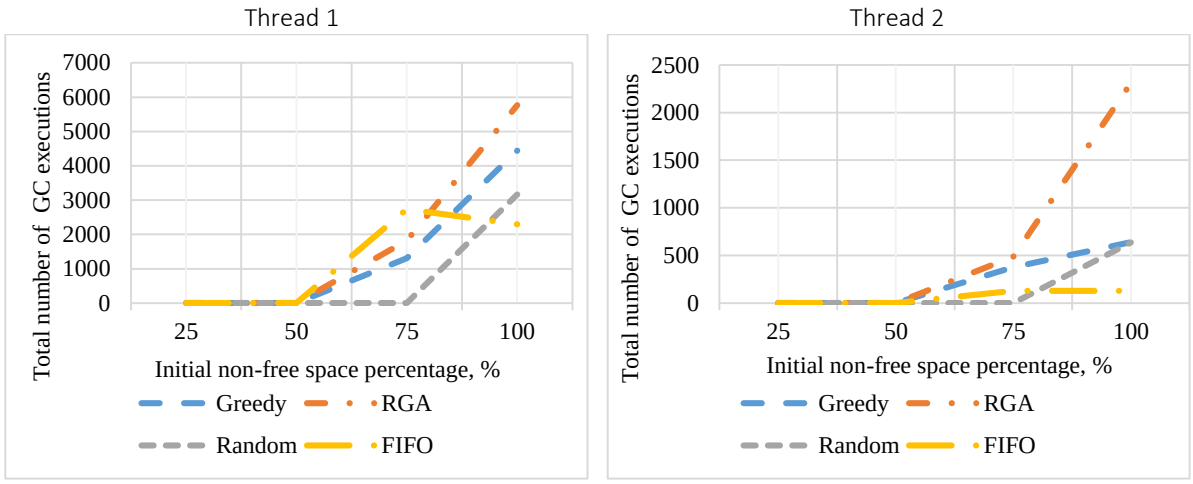

Fig. 4. – Total number of GC executions for different algorithms

Fig. 5 shows the total number of WL (Wear Leveling) executions for whole device.

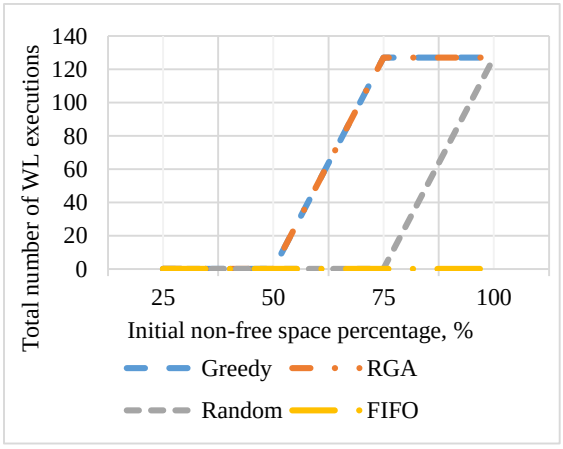

Fig. 5. – Total number of WL executions for different algorithms

Based on Fig. 5, the following conclusions can be drawn:

1. For the FIFO algorithm, wear leveling is not applied, because the victim block for garbage collection is selected from the head of the queue. In this case, the blocks will wear out equally, as least recently written blocks will be selected as victim blocks. Thus, the FIFO algorithm provides a built-in algorithmic level of uniform distribution of wear out levels for the blocks.

2. The other algorithms show approximately the same performance.

During the experiment various algorithms (Greedy, RGA, Random, FIFO) have been compared under lack of memory conditions using the MQSim simulation environment. The results of the experiment have shown that the FIFO algorithm is four times more efficient in terms of requests latency comparing to the greedy algorithm. Usage of the FIFO algorithm reduces the latency of requests by reducing the number of the garbage collection procedure calls and ensures an almost uniform page wear leveling in the device.

REFERENCES

1. Kim Y., Taurus B., Gupta A., FlashSim: A Simulator for NAND Flash-based Solid-State Drives.
2. Tavakkol A., Gómez-Luna J., Sadrosadati M., MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices.
3. Houdt B. V. Performance of garbage collection algorithms for ash-based solid state drives with hot/cold data.
4. Houdt B. V. A Mean Field Model for a Class of Garbage Algorithms in Flash-based Solid State Driver.