Performance analysis of buffer management policy considering internal parallelism of solid state drives

Ilhoon Shin and Jundo Kim

Department of Electronic Engineering, Seoul National University of Science and Technology, Gongleung-Dong, Nowon-Gu, Seoul 139–744, South Korea

a) ilhoon.shin@snut.ac.kr

Abstract: A problem with studies to utilize high-capacity RAM inside a SSD as buffer cache for NAND flash memory is their assumption that the NAND is a single chip, when current SSDs parallelise I/O requests through multiple NAND chips. Further, the studies focus on the buffer replacement policy, overlooking the more fundamental question of whether to use the buffer as read/write or write-only buffer. This paper compares the performance of the two buffer types in an SSD environment with internal parallelism using block I/O traces of representative servers and finds the followings. i) Even if the buffer replacement policy is the same, the average response time differs by up to 29.8% depending on the buffer type. Overall, the read/write buffer has a shorter average response time due to a higher buffer hit ratio. ii) However, despite a low buffer hit ratio, the average response time of read requests is reduced by up to 82.5% in the write-only buffer. This is because the read misses are handled bypassing the buffers in the write-only buffers, and thus there is no need to wait until the victim dirty buffer is flushed. iii) The response time is mainly determined by the waiting time, and the long waiting time occurs when evicting a dirty buffer. Therefore, when designing a buffer management policy, it should be considered to flush the tail dirty buffers in advance.

Keywords: solid state drives, NAND flash memory, buffer management, read/write buffer, write-only buffer, internal parallelism

Classification: Circuits and modules for storage

References

[1] A. Ban: Unites States Patent, No. 5,404,485 (1995).
[2] A. Ban: Unites States Patent No. 5,937,425 (1999).
[3] J. Kim, et al.: “A space-efficient flash translation layer for compactflash systems,” IEEE Trans. Consum. Electron. 48 (2002) 366 (DOI: 10.1109/TCE.2002.1010143).
[4] S.-W. Lee, et al.: “A log buffer based flash translation layer using fully associative sector translation,” ACM Trans. Embed. Comput. Syst. 6 (2007) 18
1 Introduction

Compared with hard disks, solid state drives (SSDs) provide high I/O operations per second (IOPS), low energy consumption, silence, and lightness of weight. As a result, SSDs are rapidly replacing hard disks in the laptop, PC, server, and data centre markets. SSDs use NAND flash memory as storage media. In NAND flash memory, once data are written to a cell, new data cannot be written there until the data of the corresponding cell are erased. Therefore, NAND-based storage devices incorporate a flash mapping layer (FTL) to perform an out-of-place update, which writes data to a new location instead of the original location [1, 2, 3, 4]. With the out-of-place update, the location of the data changes on every write, so the FTL maintains the current physical location of each sector through a mapping table between the logical address space and its physical address space. The mapping table is usually kept in RAM to avoid performance degradation. SSDs also embed a high-capacity DRAM for this purpose. The remaining DRAM space can be utilised as buffer cache for NAND flash memory, and various studies have been conducted to improve the overall performance by utilizing this buffer space [5, 6, 7, 8].

Existing buffer-related studies have focused mainly on the question of which data to replace when the buffer space is full to maximise the buffer hit ratio and to reduce the number of I/O requests to NAND flash memory. However, the more
fundamental question, that of whether to use the buffer as a read/write buffer that accommodates both read and write requests or as a write-only buffer that accepts only write requests, has been overlooked. Further, they assume that the underlying NAND is a single chip, and they try to reduce the number of write requests (because a write operation is slower by an order of magnitude than a read operation in NAND flash memory) and to generate a NAND-friendly write pattern. For example, the Clean-First LRU (CFLRU) policy considers a clean buffer that has not been modified as a preferred replacement candidate not to create write requests [5]. The Flash-Aware Buffer (FAB) [6] and the Block Padding LRU (BPLRU) [7] policies manage buffers in NAND block units to generate sequential writes that are known as NAND-friendly [3, 4]. However, modern SSDs parallelise I/O requests with multiple NAND chips and dies [9, 10, 11], and as a result, the response time of write requests is greatly reduced and the random write pattern is no longer NAND-unfriendly [9]. This indicates that the existing buffer replacement policies that are designed for a single NAND chip are likely to be inadequate for modern SSDs.

As a first step of designing an efficient buffer management policy, this paper evaluates the performance of a read/write buffer and a write-only buffer on a SSD simulator that implements internal parallelism. The performance evaluation using block I/O traces of representative servers shows the following facts. First, even if the buffer replacement policy is the same, the average response time differs by up to 29.8% depending on whether the buffer is used as a read/write buffer or a write-only buffer. Therefore, this issue should be considered when designing a buffer management policy together with the replacement issue. Second, although the read/write buffer delivers much higher buffer hit ratio, the average response time of read requests is much shorter in the write-only buffer. Because the read miss is processed bypassing the buffer in the write-only buffer, waiting until the victim dirty buffer is flushed is eliminated. Third, the response time is mainly determined by the waiting time of flushing the victim dirty buffer. Therefore, when designing a buffer management policy, it should be considered to flush the tail dirty buffers in advance before processing the buffer misses.

The remainder of the paper proceeds as follows: Section 2 explains the internal parallelism of modern SSDs and describes previous work related to buffer management, Section 3 presents the performance evaluation results, and Section 4 draws the conclusions.

2 SSD internal parallelism and buffer management

SSDs connect multiple NAND flash memory chips in parallel via multiple channels. Each NAND chip consists of multiple dies, each of which can independently perform read, write, and erase operations. In addition, each die is composed of multiple planes, and planes belonging to the same die can perform the same kind of operation at the same time. Therefore, SSDs can improve performance by dividing I/O requests into NAND page units, distributing them to multiple chips, dies, and planes and processing them at the same time. Therefore, the processing mechanisms for small random writes and large sequential writes are not essentially different, and they do not differ significantly in performance [9], whereas in a
NAND flash chip, by contrast, a sequential write is more advantageous in performance [3, 4, 6, 7]. Additionally, because both read requests and write requests can be processed in parallel, the throughput of the write requests is similar to that of the read requests [9]. Intensive studies have been conducted to maximise this parallel processing capability of SSDs by evenly distributing the divided sub-requests to chips, dies, and planes, but most of these did not use the internal RAM as buffer cache for flash memory [9, 10, 11].

The internal RAM of SSDs is primarily used to store the mapping table between the logical address space and the physical address space managed by the FTL, but the remaining space can be used as buffer cache for the NAND flash memory. Therefore, many methods have been proposed to improve the buffer hit ratio and to generate NAND-friendly write request patterns by efficiently utilizing this buffer space [5, 6, 7, 8]. CFLRU manages all buffers in a NAND page unit, and when the buffer space is insufficient, the clean buffers (those whose data have not been modified) are preferentially replaced because replacing a dirty buffer causes a NAND write request [5]. FAB manages buffers in both a NAND block and a page unit, and when the buffer space is insufficient, the largest block that has the most pages in buffer is replaced to generate a write request that is as large as possible [6]. BPLRU manages buffers in a NAND block unit, and when the buffer space is insufficient, the least recently used block is replaced [7]. At this time, if there are empty pages in the block, those pages are read from NAND flash memory before issuing the write request, and therefore the write request is always generated in a complete NAND block. CLOCK-DNV manages buffers in a NAND page unit and uses a clock algorithm when replacing the buffer [8]. Its main distinction from the previous methods is that it assumes that SSDs have non-volatile RAM that can be used as a buffer in addition to DRAM.

The existing methods have two main problems. First, they assume that the underlying storage medium is a single NAND flash chip, and the internal parallelism of the SSD is not considered. Assuming that the write operation is much slower than the read operation and that a sequential write pattern is more NAND-friendly than a random write pattern, they try to avoid generating write requests if possible and to generate a large sequential write pattern. However, these assumptions are not valid for SSDs that process I/O requests in parallel. The negative effect of the slow NAND write operation is mitigated by the simultaneous processing of multiple write requests, and a large sequential write uses the same procedure as that used for multiple small random writes; that is, it is divided into multiple sub-requests in NAND page units that are processed simultaneously [9]. As a result, the performance of a sequential write pattern is similar to that of a random write pattern.

Second, the existing methods fix the buffer role as either a read/write buffer or a write-only buffer and focus only on the replacement policy. For example, CFLRU assumes a read/write buffer, and BPLRU and CLOCK-DNV assume a write-only buffer. Before designing a buffer replacement policy, it is necessary to evaluate which of the two buffer types is more suitable for the internal buffer management scheme of SSDs. The above problems necessitate an evaluation of the performance of the read/write buffer and the write-only buffer on the SSD model that utilises internal parallelism.
3 Performance evaluation

We used the SSDSim [10] simulator to model the internal parallelism of SSDs for evaluating the performance of the buffer management policy. The original simulator implements only a write-only buffer that uses an LRU replacement policy. Therefore, we additionally implemented a read/write buffer that uses the LRU replacement policy. Also, the original simulator has a bug that it does not follow the order between processing a buffer miss and flushing a victim dirty buffer. The former must be started after the latter is finished.

The SSD was modelled as follows [11]. It consisted of four NAND flash chips, which were connected to the NAND controller through two parallel channels. That is, two chips shared one channel. Each chip was composed of four dies, and each die was composed of four planes. The NAND block of each plane was 512 KB in size, and the page was 4 KB in size. The latencies of page read, page write, and block erase operations were 500 µs, 900 µs, and 3.5 ms, respectively. It took 25 ns to send one byte over the channel, and the DRAM access time was 15 ns per word. The overprovision rate of the SSD was set to 20%. Page mapping FTL [1, 9, 10, 11] was used, and if the clean block of each plane fell below 10%, garbage collection was performed in the background.

Block I/O traces of servers downloaded from Microsoft Research Center (MSRC) (prn0 and proj0) [12] and from the Storage Performance Council (SPC) (fin1 and fin2) [13] were used as input traces. The characteristics of each workload are detailed in Table I.

Table I. Trace information

| Trace | Read request ratio (%) | Avg. read req. size (sectors) | Avg. write req. size (sectors) | Avg. inter-arrival time (ms) |
|-------|------------------------|-------------------------------|-------------------------------|-----------------------------|
| prn0  | 22.21                  | 45.67                         | 19.34                         | 0.11                        |
| proj0 | 5.85                   | 35.68                         | 81.83                         | 0.14                        |
| fin1  | 15.40                  | 4.50                          | 7.45                          | 0.01                        |
| fin2  | 78.46                  | 4.56                          | 5.84                          | 0.01                        |

Fig. 1 shows the buffer miss ratio. The x-axis is the buffer size in MB and the buffer type. ‘RW’ denotes the result for the read/write buffer, and ‘Write’ that for the write-only buffer. The y-axis is the buffer miss ratio. With both buffer roles, LRU was used as the replacement policy. The legend ‘miss with flush’ denotes the buffer miss that evicts a dirty buffer. The legend ‘miss without flush’ denotes the buffer miss that evicts a clean buffer.

The results show that the read/write buffer delivers lower miss ratio than the write-only buffer in overall. In proj0 and prn0, where the buffer miss ratio is very high, the two buffer roles show the similar buffer miss ratios. However, in fin1 and fin2, where the buffer miss ratio is relatively low, the read/write buffer outperforms
the write-only buffer. The total miss count of the read/write buffer is lower by up to 25.2% in fin1 and by up to 58.4% in fin2. The gap is mainly due to the difference of ‘miss without flush’. In the write-only buffer, ‘miss without flush’ corresponds to a buffer miss of a read request. Thus, we can know that the buffer miss ratio of the read requests is very high when using the write-only buffer, especially in fin2 trace. This indicates that the write-only buffer can be inefficient in workloads where a temporal locality of read requests exists. Using the small read/write buffer together with the write-only buffer can lower the miss ratio.

![Fig. 1. Comparison of the buffer miss ratio.](image)

Fig. 2 compares the average response time of read/write requests. The x-axis is the same with Fig. 1, and the y-axis is the response time in µs. The legend ‘waiting time’ indicates the delay time until buffer miss processing starts. The waiting time is caused by dirty buffer replacement, pending or ongoing requests. The legend ‘processing time’ denotes the actual buffer miss processing time.

The results show that in fin1 and fin2 traces the read/write buffer delivers a better response time than the write-only buffer due to its lower buffer miss ratio. The improvements range from 4.0 to 8.4% in fin1 and from 12.1 to 29.8% in fin2. The interesting fact is that in the both buffer roles the long response time is mainly caused by the long waiting time, except in fin2 trace. This is because the ratio of ‘miss with flush’ is relatively very high, as seen in Fig. 1, which means that most buffer miss requests wait until the victim dirty buffer is flushed. In order to improve
the average response time, this waiting time should be shortened by flushing tail dirty buffers in advance and choosing a clean buffer as a victim.

Meanwhile, except for synchronous or direct writes, an application does not wait for a write request to complete after issuing it. That is, the average response time of a read request is more important to application performance than the average response time of a write request. Thus, we compare the average response time of only read requests in Fig. 3. The x-axis and the y-axis are the same with Fig. 2.

The results show that the write-only buffer outperforms the read/write in overall. In all the traces, the response time of the read requests are greatly reduced: by 47.4–52.2% in proj0, by 51.3–82.5% in prn0, by 54.3–64.8% in fin1, and by 11.7–31.9% in fin2. The improvements is mainly achieved by reducing the waiting time. In the write-only buffer, the read miss request is processed bypassing the buffer, and thus the waiting time of the dirty buffer flush is not involved. This again indicates that reducing the waiting time of the buffer miss is important, especially in the read/write buffer. We can shorten the waiting time of the read requests by flushing tail dirty buffers in advance and by preferentially selecting a clean buffer as a victim, especially in case of a read miss.
In this paper, we have analysed the performance of a read/write buffer and a write buffer for an LRU replacement policy in an SSD with internal parallelisation. The results show that even under the same buffer replacement policy, the average response time of I/O requests can differ by up to 29.8% depending on whether the buffer is used as a read/write buffer or a write-only buffer. Therefore, the buffer role as well as the replacement policy should be considered. In addition, the response time was mainly determined by the waiting time of flushing the victim dirty buffer. Therefore, when designing a buffer management policy, it should be considered to flush the tail dirty buffers in advance and to preferentially choose a clean buffer as a victim, especially in case of a read miss. As a future work, we plan to design a new buffer management policy which combines the read/write and the write-only buffer and pre-flushes multiple tail dirty buffers simultaneously to utilize the parallel processing capability of SSDs.

**Acknowledgements**

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2016R1D1A1A09918559).