CLOCK-DNV: A Write Buffer Algorithm for Flash Storage Devices of Consumer Electronics

Dong Hyun Kang, Se Jun Han, Young-Chang Kim, and Young Ik Eom

Abstract—**Today, flash storage devices have become a standard storage in consumer electronics devices, such as smartphones, smart tablets, and smart TVs, due to their attractive features. Since flash storage helps to cut down on system response for customers, consumer devices based on the flash storage are gradually increasing. Unfortunately, the flash storage of consumer electronics devices suffers from random write requests because applications running on the consumer device simultaneously issue a lot of write operations to store their persistent data.**

This paper proposes a novel write buffer algorithm, called Clock with DRAM and NVM hybrid write buffer (CLOCK-DNV), that reshapes random write requests into sequential ones. In order to maximize the number of consecutive write requests, CLOCK-DNV takes the benefits of NVM media with the dirty page padding mechanism. For extensive evaluation, the proposed algorithm is implemented on a flash storage simulator, which is widely used for performance analysis, and is compared with two write buffer algorithms, FAB and CBM. The evaluation results clearly show that CLOCK-DNV maintains higher hit ratios than other write buffer algorithms. Moreover, CLOCK-DNV efficiently reduces the number of write requests issued to the underlying flash memory by up to 56% compared with the state-of-the-art algorithm, CBM, while extending the endurance of flash storage by up to 56%.

Index Terms— **Cache storage, Smart devices, Smart homes, Flash memories, Nonvolatile memory.**

Manuscript received January 1, 2017; accepted February 28, 2017. Date of publication April 12, 2017. This work was supported by Institute for Information & communica-tions Technology Promotion(IITP) grant funded by the Korea govern-ment(MSIP) (No.R0126-15-1082, Management of Developing ICBMS(IoT, Cloud, Bigdata, Mobile, Security) Core Technologies and Development of Exascale Cloud Storage Technology) and Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015M3C4A7065696*). (Corresponding author: Young Ik Eom.)*

Dong Hyun Kang is with Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon, South Korea (e-mail: kkangsu@skku.edu).

Se Jun Han is with Samsung Electronics, 129, Samsung-ro, Yeong-tong-gu, Suwon-si, Gyeonggi-do, South Korea (e-mail: sejun.han@samsung.com).

Young-Chang Kim is with Electronics and Telecommunications Research Institute, Daejeon, South Korea (e-mail: zerowin@etri.re.kr).

Young Ik Eom is with Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon, South Korea (e-mail: yieom@skku.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCE.2017.014700

I. Introduction

NAND flash storage has been widely used in portable consumer devices, such as MP3 players, tablet PCs, and smartphones, because it provides attractive features, such as small size, shock resistance, high performance, and low power consumption. Recently, there has been considerable interest in employing a large-capacity flash storage as a primary storage of smart TVs to efficiently support the special functionalities, including web browser and home cloud server [1], [2]. However, since the flash storage does not allow in-place update, it suffers from the *garbage collection* (GC) overhead at the *flash translation layer* (FTL). Unfortunately, random write requests exacerbate such GC overhead by increasing the number of extra writes during the GC. Moreover, random write requests negatively impact the limited endurance of flash storage by frequently triggering the GC operation [3]-[5].

In order to solve the above storage issues, early work has focused on the DRAM write buffer inside the flash storage [6]-[11] because it is one way to reshape the random write requests into sequential ones. Some research groups reduced the GC overhead with flash-friendly write buffer algorithm that groups multiple pages inside the write buffer into a block based on their logical block addresses (LBA) and evicts consecutive pages belonging to a victim block simultaneously [8]-[11]. However, those algorithms would drop the cache hit ratio of the write buffer because those blocks that have a large number of pages would be selected as victim blocks to issue them sequentially.

Over the last few years, some researches proposed utilizing *non-volatile memory* (NVM) as a write buffer inside flash storage [12], [13] because emerging NVM has DRAM-like features, including byte-addressability and high performance, as well as it has storage-like features, including non-volatility. However, recent research trends in NVM technology indicate that *hybrid write buffer architecture*, which is composed of DRAM and NVM, takes a positive performance and cost benefits compared with the write buffer built on DRAM or NVM alone [12]-[16]. For example, some researchers focused on the hybrid write buffer architecture and proposed the cooperative buffer management (CBM) scheme [12] that employs NVM as a write buffer to reduce the GC cost. Unfortunately, CBM is inefficient in write or read intensive workloads because it was designed to use space on DRAM for a read cache and to use space on NVM for a write buffer.

This paper presents CLOCK-DNV that uses the hybrid write buffer architecture, comprised of DRAM and NVM. The goal of CLOCK-DNV is to obtain high cache hit ratio of the write buffer, high performance, and endurance of flash storage. In order to achieve the above goals, CLOCK-DNV extends the traditional CLOCK algorithm in terms of the *spatial locality* and it takes the benefits of NVM for minimizing the number of write requests issued to the underlying flash memories. Especially, CLOCK-DNV maximizes the number of consecutive pages issued to the flash memory by using the *dirty page padding* mechanism that merges pages on DRAM with those on NVM. For evaluation, CLOCK-DNV is implemented on a *flash storage simulator* [17] and evaluated with various workloads. The evaluation results show that CLOCK-DNV maintains higher hit ratio than other write buffer algorithms. In addition, CLOCK-DNV efficiently reduces the number of write requests and the number of erase operations to the underlying flash memory by up to 56% and 56%, respectively, compared with the state-of-the-art write buffer algorithm, CBM.

The remainder of this paper is organized as follows. Section II describes the background and related work to understand this work in detail. Section III presents the design of CLOCK-DNV write buffer algorithm. Section IV introduces the evaluation setup and presents several evaluation results in terms of cache hit ratio, performance, and endurance. Finally, Section V concludes this paper.

II. Background and Related Work

Over the last few years, *non-volatile memory* (NVM) has received a lot of interests in both industry and academia. This section briefly provides the overview of NVM technology and then introduces the related work of this paper.

A. Emerging Non-volatile Memory

Recently, there are many field studies to utilize NVM technologies, such as spin-torque transfer magneto-resistive memory, phase change memory, and transistor-less cross point. Some researchers already consider NVMs as part of I/O hierarchy. This is because emerging NVMs have both *DRAM-like characteristics* and *storage-like characteristics* as presented in Table I [16]. In addition, NVMs can be employed in various software layers, including page-cache and/or file system layer, for improving the performance of each layer.

TABLE I
Comparison of Memory Medias

Trace	DRAM	NAND	NVM
Non-volatility	No	Yes	Yes
Byte-addressability	Yes	No	Yes
Static Power	Yes	No	No
Read latency	~50 ns	~30 us	~50 ns
Write latency	~50 ns	~300 us	~250 ns
Endurance	N/A	10^4	10^8

Unfortunately, emerging NVM technologies are not yet the available successor of DRAM because of two issues [18]-[20]. First, NVM has asymmetric read and write latency and the write latency of NVM is much slower than that of DRAM [18], [19]. Second, even though the endurance of NVM is higher than that of NAND, its endurance is much lower than that of DRAM media, and therefore, wear-leveling mechanism is required [20]. In order to hide the above issues, some researchers introduced a *hybrid memory architecture*, which is composed of DRAM and NVM, and proposed combining approaches to take the benefits of each memory media. Unfortunately, previous works concentrated on the host side software layers, such as page-cache [14], [15] and/or file system layer. This paper focuses on the storage side ones, such as write buffer inside the flash storage, because this NVM can provide plenty of opportunities to improve overall performance and endurance of the flash storage despite of its low write latency.

B. Write Buffer Algorithm inside a Flash Storage

Today, nearly all consumer electronics devices employ flash storage as their storage media, ranging from portable devices to home appliances. However, the flash storage still suffers from random write requests since such write requests highly increase the expensive GC cost on the flash storage.

There have been many researches on the write buffer algorithm [6]-[13] inside flash storage to reshape random write requests into sequential ones. Previous studies can be classified into two types: DRAM write buffer [6]-[11] and hybrid write buffer that is composed of DRAM and NVM [12], [13].

The DRAM write buffer is well-known and it has been adopted in commercial flash storage. This is because this type temporally stores pages on DRAM write buffer at a block granularity and simultaneously issues stored pages to the underlying flash memory in a sequential manner. FAB was proposed a flash-aware buffer management approach based on the least recently used (LRU) algorithm [9]. This approach gathers pages on the DRAM write buffer into a block based on LBA of each request, and evicts whole pages belonging to a same block at the same time for sequential writes. LB-CLOCK was also designed block-granularity write buffer algorithm by exploiting the traditional CLOCK algorithm [10]. However, these two approaches generate *extra write requests* to reclaim a free space on the write buffer. To reduce the extra write requests, some research groups proposed a combined write buffer algorithm that manages pages on the write buffer at a page-granularity and block-granularity. Some researchers logically partitioned the DRAM write buffer inside a flash storage into page-based list and block-based list [11]. Some researchers introduced a hybrid write buffer tiered DRAM and NVM for a flash storage and proposed a cooperative buffer management (CBM) approach [12] to reduce the GC cost on flash storage. This approach employs DRAM as a read cache and NVM as a write cache to utilize the non-volatility feature of NVM. Also, CBM partitioned the NVM write buffer into page region and block region for sequential write. Unfortunately, when the workload is write-intensive, CBM frequently evicts pages on the NVM write buffer even when there is enough free space inside the DRAM write buffer. As a result, CBM suffers from lower cache hit ratio.

A previous version of this work [21] also employed the *hybrid write buffer* and it focused on both temporal and spatial locality for the overall performance and endurance of the flash storage. Now, the current version enhances the earlier version with the *dirty page padding* mechanism in terms of spatial locality. Also, this paper presents both endurance of the NVM and flash storage with the extensive evaluation results.

III. Design of CLOCK-DNV Algorithm

This paper proposes a novel write buffer algorithm, called Clock with DRAM and NVM hybrid write buffer (CLOCK-DNV), for the flash storage. In order to take the benefits of DRAM and NVM, CLOCK-DNV separately handles the pages on each memory with its own CLOCK algorithm: CLOCK-D for DRAM and CLOCK-NV for NVM. Fig. 1 shows the overview of CLOCK-DNV architecture. As shown in Fig. 1, CLOCK-D manages all pages on DRAM at a *page granularity*, whereas CLOCK-NV handles pages on NVM at a *block granularity* for both the temporal and spatial locality.

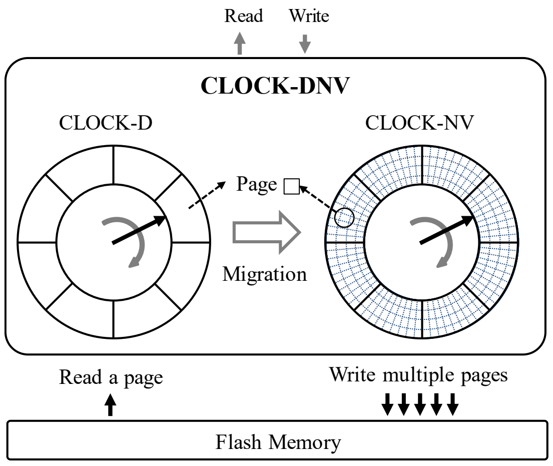

Fig. 1. The overview of CLOCK-DNV architecture.

A. CLOCK-D Replacement Algorithm

CLOCK-D is designed to maintain high hit ratio and high space utilization of DRAM. To achieve the design goals, CLOCK-D follows the basic rules of the traditional CLOCK algorithm and it handles both clean and dirty pages on DRAM. However, unlike traditional CLOCK algorithm, CLOCK-D prefers to keep clean pages over dirty pages on the DRAM write buffer, considering the volatile feature of DRAM (where the scheme is called *dirty first migration*). For the dirty first migration, the reference bit of each page is differently set according to its dirty bit. If a page on DRAM is re-accessed (cache hit), CLOCK-D first checks its dirty bit. If dirty bit of the re-accessed page is 0 (clean page), CLOCK-D changes its reference bit to 1 to give it a second chance. On the other hand, if dirty bit of the page is 1 (dirty page), CLOCK-D does not update its reference bit. Thus, CLOCK-D can maintain clean pages on DRAM for a long time compared with dirty pages.

Fig. 2 shows the pseudo-code of the CLOCK-D algorithm. If there is no free space on DRAM, CLOCK-D first selects a victim page by using its own hand and then reclaims it. If the reference bit of the page pointed by the hand is 1, CLOCK-D changes its reference bit from 1 to 0 for the second chance and then the hand of CLOCK-D is moved to the next page in a circular list. Otherwise, the page pointed by the hand is either evicted in the circular list or migrated to a circular list of CLOCK-NV depending on its dirty bit. If a dirty bit of the victim page is 0 (clean page), CLOCK-D just evicts the page because a read operation on flash memory is much faster than a write operation. On the other hand, if the victim page is dirty, CLOCK-D migrates the page to NVM (CLOCK-NV circular list) for grouping dirty pages into a group.

```
CLOCK-D (circular_list clock, page *p) {
  if (clock.findPage(p)) {    // cache hit
    page *wp = clock.writePage(p);
    if (wp->dirtyBit == 0) {              // clean page
      wp->RefBit = 1;
    }
  } else {                          // cache miss
    if (clock.isFull()) {
      page *v = clock.selectVictimPage();
      if (v->dirtyBit == 0) {             // clean page
        clock.discardPage(v);
      } else {                            // dirty page
        clock.migratePage(v);
      }
    }
    clock.addPage(p);
  }
}
```

Fig. 2. The pseudo code for CLOCK-D algorithm.

B. CLOCK-NV Replacement Algorithm

CLOCK-NV also follows the basic rules of the traditional CLOCK algorithm for the temporal locality. However, since the design goal of CLOCK-NV is to reshape the random write requests issued to the flash memory into sequential ones, CLOCK-NV has two important differences compared with the traditional CLOCK algorithm. First, unlike traditional CLOCK algorithm, CLOCK-NV ignores both the *flush commands* and *force unit access* (FUA) flagged requests from the host because NVM is the persistent region inside the hybrid write buffer. Second, CLOCK-NV manages all pages on the NVM write buffer at a block granularity whose block size is the same as the flash block size and uses a reference bit per block. Therefore, CLOCK-NV has to decide where to place a page among the different blocks and has to track the number of pages in each block.

Fig. 3 shows the pseudo-code of the CLOCK-NV algorithm. If a page is migrated from DRAM to NVM, CLOCK-NV first checks the *logical block address* (LBA) of the page and then places the page to the corresponding block by dividing its LBA by the block size. If the corresponding block does not exist in the circular list of CLOCK-NV, a new block is created in CLOCK-NV and then the page is attached to the created block. Finally, CLOCK-NV updates the reference bit of the block to 1. Since the reference bit is periodically cleared by its own hand, whenever a page belonging to the block is re-accessed, CLOCK-NV sets the reference bit of the block to 1 for the temporal locality. If there is no free space on NVM, CLOCK-NV first finds a victim block and then simultaneously issues all pages in the victim block to the underlying flash memory. Since these multiple page writes can help to reduce the GC cost of the flash storage, CLOCK-NV carefully selects a victim block, which has the largest number of pages among the blocks in the circular list, by traversing whole blocks in CLOCK-NV. If some blocks inside CLOCK-NV have the same number of pages, CLOCK-NV selects a victim block based on the reference bit of each block because it indicates whether the pages in the block have been re-accessed or not recently.

```
CLOCK-NV (circular_list clock, page *mp) {
   if (clock.findPage(mp)) {      // cache hit
      clock.writePage(mp);
      clock.updateBlockRefBit(mp);
   } else {                        // cache miss
      if (clock.isFull()) {
         block *b = clock.selectVictimBlock();
         clock.flushPages(b);
      }
      block b = clock.existBlock(mp);
      if (b == NULL) {
         clock.createBlock(&b);
      }
      clock.addPage(&b, mp);
   }
}
```

Fig. 3. The pseudo code for CLOCK-NV algorithm.

C. Dirty Page Padding Mechanism

To maximize the number of consecutive writes issued to the underlying flash memory, CLOCK-DNV uses the dirty page padding mechanism that pads some dirty pages of CLOCK-D into a block of CLOCK-NV. This mechanism is triggered after selecting a victim block by CLOCK-NV replacement policy. For dirty page padding, CLOCK-DNV first scans a circular list of CLOCK-D to find the auxiliary pages whose block number is same as the victim block. If the auxiliary pages exist on DRAM, CLOCK-DNV pads the pages into the victim block of CLOCK-NV and then issues whole pages of the victim block at the same time for flash-friendly writes. Of course, this mechanism would increase the number of write requests, but it can significantly reduce the expensive GC cost.

Fig. 4 shows an example of dirty page padding mechanism with CLOCK-NV replacement algorithm. In this example, the size of a block is assumed to be four pages. When there is no free space on NVM to accommodate a new page, CLOCK-NV should evict victim pages to reclaim free space on NVM by following a sequence of steps in order. First, CLOCK-NV selects a victim block that has the largest number of pages among the blocks of the circular list as mentioned before. As a result, block 3 is selected as a victim block (Fig. 4a). Second, for the dirty page padding mechanism, CLOCK-DNV scans the circular list of CLOCK-D and finds the auxiliary dirty pages that belong to the block 3. In this example, 2 pages (page 12 and page 15) are padded into the block 3 on-the-fly (Fig. 4b). Finally, consecutive pages that belong to the victim block are issued to the underlying flash memory in a sequential manner. As a result, in this example, CLOCK-DNV flushes 4 pages (page12 – page15) to the flash memory (Fig. 4c) at the same time.

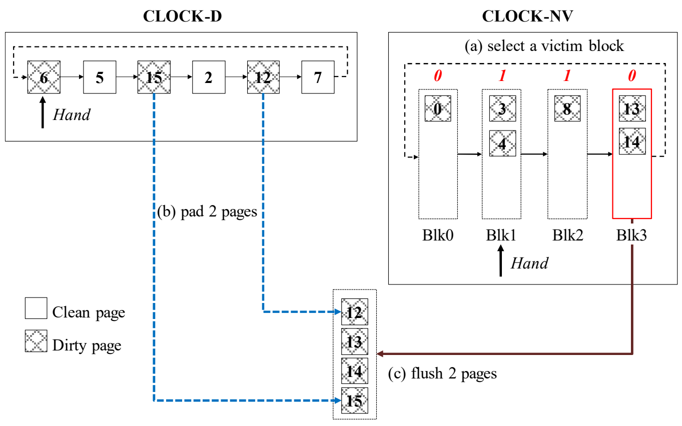

Fig. 4. An example of page padding mechanism. In this example, a block is assumed to have dirty four pages.

IV. Evaluation

This section first describes the evaluation environment and then compares the evaluation results of CLOCK-DNV with those of well-known write buffer algorithms: FAB and CBM.

A. Evaluation Setup

All experiments were performed on a flash storage simulator that emulates SLC NAND flash storage of 32GB capacity with a page-mapped FTL [17]. Table II presents other parameters of the flash storage simulator.

TABLE II
Parameters of Simulator

Parameter	Description
Page size	4 KB
Block size	256 KB (64 pages)
Page read latency	25 us
Page write latency	200 us
Block erase latency	1.5 ms
Overprovisioning ratio	15 %

For evaluation, three write buffer algorithms (FAB [9], CBM [12], and CLOCK-DNV) were separately implemented on the flash storage simulator because of two reasons. First, FAB was designed for DRAM write buffer whereas CBM and CLOCK-DNV were designed for hybrid write buffer architecture, composed of DRAM and NVM. Second, FAB handles all pages on DRAM write buffer at a block-granularity. On the other hand, CBM and CLOCK-DNV manage all pages on the hybrid write buffer at a page granularity and block granularity, simultaneously. The block size of each algorithm is set to the flash block size of the simulator. For hybrid write buffer algorithms, this paper sets the size of DRAM write buffer to be 10% of the total size of hybrid write buffer. Finally, for fair comparison, other configurations of the simulator were set with the same parameters as in CBM [12].

B. *Workloads*

To evaluate CLOCK-DNV over a wide range of applications, three real-world workloads are used as the input of the flash storage simulator: Financial, CAMWEBDEV, and Filebench trace. Financial and CAMWEBDEV trace have their own characteristics in that the Financial trace is a *read-intensive* OLTP workload from UMass Trace Repository, and the CAMWEBDEV trace is a *write-intensive* workload from SNIA. In order to expand the scope of evaluation, this paper also uses a Filebench workload that is collected by running random-rw of the Filebench benchmark. This workload has a similar proportion of read and write operations. Table III presents the characteristics of each workload in detail.

TABLE III
SPECIFICATION OF WORKLOADS

Parameter	Read-Write ratio	Trace size
Financial trace	8.2 : 1.8	1.8 GB
CAMWEBDEV trace	0.8 : 9.2	0.52 GB
Filebench trace	5 : 5	2.81 GB

C. *Cache Hit Ratio*

First, this paper compares the cache hit ratios of the three write buffer algorithms because this is critical for overall performance of the write buffer algorithm.

Fig. 5 shows the cache hit ratios of each algorithm by varying the write buffer size from 16MB to 128MB. As shown in Fig. 5, CLOCK-DNV maintains higher cache hit ratios than other algorithms in most cases. Especially, CLOCK-DNV outperforms FAB by up to 2 times in Financial workload (Fig. 5a). This is because CLOCK-DNV can reclaim free space at a page granularity to avoid the *early-eviction problem*, which evicts all pages in a victim block at the same time.

Meanwhile, CBM reveals considerably lower hit ratios than CLOCK-DNV in the case of write-intensive workload (Fig. 5b). However, this is not surprising because CBM never employs spaces on DRAM for placing any dirty page. As a result, CBM should evict multiple pages in a victim block even though free space exists on DRAM.

In the case of Filebench workload, all algorithms show lower and comparable cache hit ratios than those of the other two workloads. However, CLOCK-DNV outperforms FAB by up to 23% and CBM by up to 15%. These results confirm that CLOCK-DNV fully utilizes free spaces on both DRAM and NVM by sharing their space even though read-write ratio of the workload is 5:5.

D. *Number of Write Requests*

This paper also presents the effects on the number of write requests of CLOCK-DNV by comparing the number of flush requests issued to the underlying flash memory. The flush requests can be composed of clean and dirty pages because FAB and CBM provide *clean page padding*, which pads clean pages to maximize the number of consecutive write requests. Meanwhile, unlike the above algorithms, CLOCK-DNV

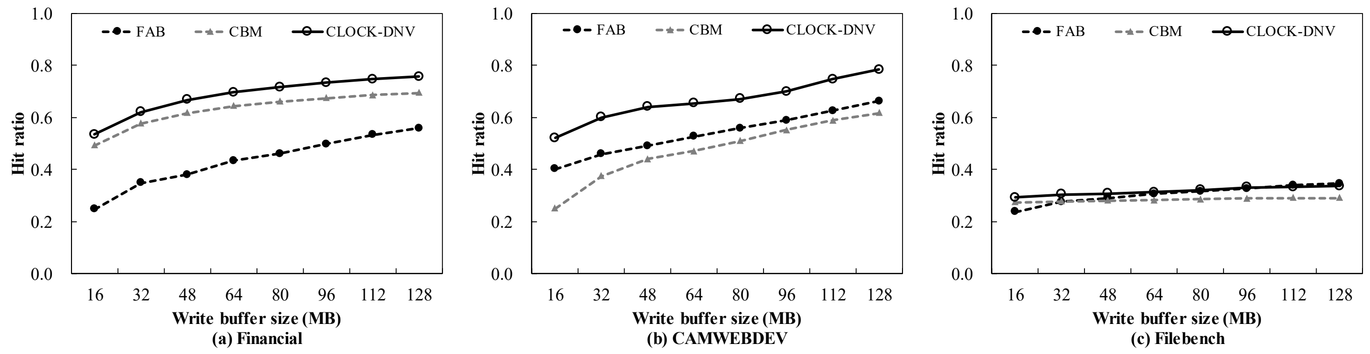

Fig. 5. Cache hit ratios.

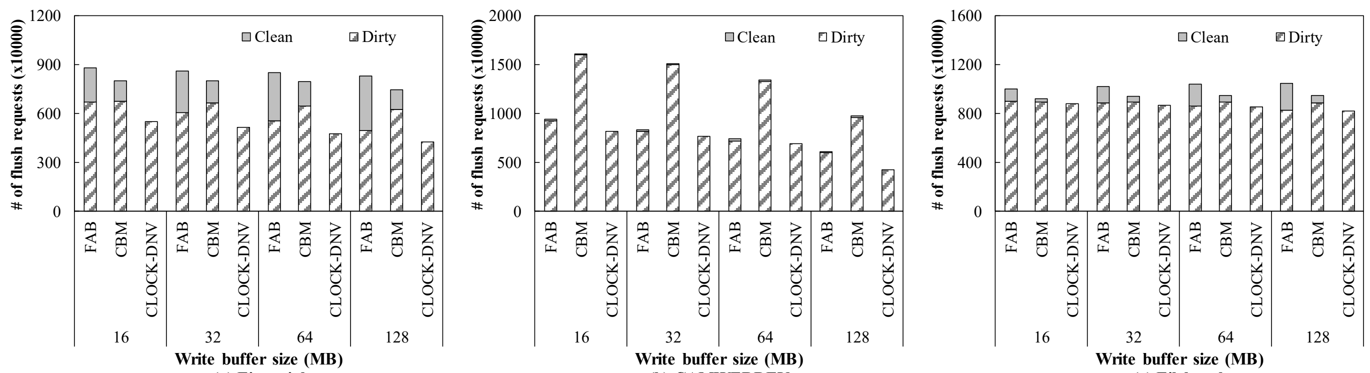

Fig. 6. The number of write requests issued to the underlying flash storage.

utilizes the *dirty page padding* mechanism as mentioned in Section III.

Fig. 6 shows the evaluation results of each algorithm as a bar type. The X axis is in log scale and the Y axis is measured by collecting evicted traces from each write buffer to the underlying flash memory. As shown in Fig. 6, CLOCK-DNV efficiently reduces the number of flush requests issued to flash memory in all cases. Especially, CLOCK-DNV completely eliminates clean page eviction because it pads only dirty pages to the victim block that is selected by CLOCK-NV replacement policy. For Financial workload, CLOCK-DNV issues less number of flush requests by up to 48% and 42% compared with FAB and CBM, respectively. In the case of Filebench workload, CLOCK-DNV outperforms FAB and CBM by up to 21% and 13%, respectively. Meanwhile, CBM reveals the negative performance results in CAMWEBDEV workload. The major reason is that write requests of this workload cannot be placed on the space of DRAM. As a result, CBM dramatically increases the number of flush requests issued to the underlying flash memory by up to 56%, compared with CLOCK-DNV.

E. Endurance of Memory Media

It is hard to demonstrate the effectiveness of CLOCK-DNV without verifying the endurance of memory media because NVM and flash storage have limited lifetime and their endurance is lower than that of DRAM media. Thus, this section shows how many write requests are issued to NVM and also compares the endurance of the flash storage based on the flash storage simulator.

Endurance of NVM: Actually, while the endurance of NVM is an order of magnitude higher than that of NAND flash memory, yet it also suffers from the limited lifetime and it still requires a wear-leveling technique.

Fig. 7 shows the number of write requests issued to NVM and it only compares CLOCK-DNV with CBM because FAB does not employ NVM as its own write buffer. As expected, CLOCK-DNV shows the lowest number of write requests in all cases. Especially, CLOCK-DNV reduces the number of write requests by up to 55% in the case of CAMWEBDEV, which is a write-intensive workload. The major reason for this result is that CLOCK-DNV shares all the space of DRAM and NVM for write requests unlike CBM algorithm, which uses DRAM as a read cache and NVM as a write cache. As a result, all write requests of CLOCK-DNV can be serviced on both DRAM and NVM without frequent eviction operations.

Endurance of Flash Storage: In order to verify the endurance of flash storage, this paper also presents the number of erase operations that is measured from the flash storage simulator. Fig. 8 shows the simulated endurance results of each write buffer algorithm with three different workloads. Unsurprisingly, CLOCK-DNV dramatically improves the endurance of the flash storage. This improvement can be achieved by flushing dirty pages with the dirty page padding mechanism because the number of erase operations is strongly correlated with the write pattern. As shown in Fig. 8, CLOCK-DNV clearly extends the lifetime of the flash storage, compared with FAB algorithm: from 37% to 48% for Financial, from 2% to 28% for CAMWEBDEV, and from 11% to 20% for Filebench workload. In the best case, CLOCK-DNV extends the lifetime of the flash storage by up to 56%, compared with CBM write buffer algorithm (Fig. 8b).

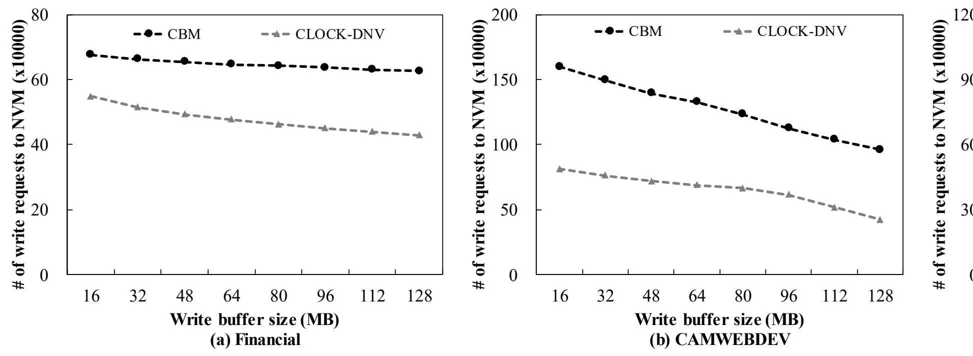

Fig. 7. Comparison of the NVM endurance of CLOCK-DNV with that of CBM.

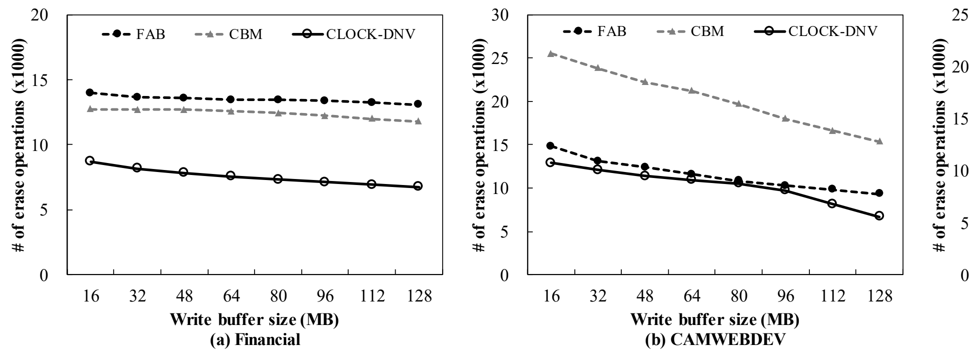

Fig. 8. Comparison of the flash storage endurance of each algorithm with three different workloads.

V. Conclusion

Today, most consumer electronics devices employ flash storage devices as their main storage, but excessive random write requests issued to the underlying flash storage significantly decreases both the performance and endurance of the flash storage. This paper proposed CLOCK-DNV to reshape random write requests into sequential ones by using the *hybrid write buffer architecture*, which is composed of DRAM and NVM media. The key idea of CLOCK-DNV is to share all free space on DRAM and NVM, and to handle the space on DRAM at a page granularity and the space on NVM at a block granularity for utilizing both the temporal and spatial locality. In addition, CLOCK-DNV helps to maximize the number of consecutive write requests by taking advantage of NVM and the *dirty page padding* mechanism. Evaluation results on the flash storage simulator clearly confirm that CLOCK-DNV outperforms previous write buffer algorithms, FAB and CBM, in all cases, while extending the endurance of both NVM and flash storage.

References

[1] M. R. Cabrer, R. P. D. Redondo, A. F. Vilas, J. J. Pazos Arias, and J. G. Duque, "Controlling the smart home from TV," *IEEE Trans. Consumer Electronics*, vol. 52, no. 2, pp. 421-429, Jul. 2006.

[2] D. Lee, C. Min, and Y. I. Eom, "Effective flash-based SSD caching for high performance home cloud server," *IEEE Trans. Consumer Electronics*, vol. 61, no. 2, pp. 215-221, May 2015.

[3] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "SFS: Random write considered harmful in solid state drives," *in Proc. USENIX FAST*, 2012, pp. 309-320.

[4] H. Kim, M. Ryu, and U. Ramachandran, "What is a good buffer cache replacement scheme for mobile flash storage?," *in Proc. ACM SIGMETRICS*, 2012, pp. 235-246.

[5] D. H. Kang, C. Min, and Y. I. Eom, "An efficient buffer replacement algorithm for NAND flash storage devices," *in Proc. IEEE MASCOTS*, 2014, pp. 239-248.

[6] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," *in Proc. ACM CASES*, 2006, pp. 234-241.

[7] H. Jung, H. Shim, S. Park, and J. Cha, "LRU-WSR: Integration of LRU and write sequence reordering for flash memory," *IEEE Trans. Consumer Electronics*, vol. 54, no. 3, pp. 1215-1223, Aug. 2008.

[8] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," *in Proc. USENIX FAST*, 2008, pp. 239-252.

[9] H. Jo, J.-U. Kang, S.-Y. Park, and J.-S. Kim, "FAB: Flash-aware buffer management policy for portable media players," *IEEE Trans. Consumer Electronics*, vol. 52, no. 2, pp. 485-493, May 2006.

[10] B. Debnath, S. Subramanya, D. Du, and D. J. Lilja, "Large block clock (LB-CLOCK): A write caching algorithm for solid state disks," *in Proc. IEEE MASCOTS*, 2009, pp. 1-9.

[11] G. Wu, B. Eckart, and X. He, "BPAC: An adaptive write buffer management scheme for flash-based solid state drives," *in Proc. IEEE MSST*, 2010, pp. 1-6.

[12] Q. Wei, C. Chen, and J. Yang, "CBM: A cooperative buffer management for SSD," *in Proc. IEEE MSST*, 2014, pp. 1-12.

[13] E. Lee, J. Kim, H. Bahn, and S. H. Noh, "Reducing write amplification of flash storage through cooperative data management with NVM," *in Proc. IEEE MSST*, 2016, pp. 1-6.

[14] H. G. Lee, "High-performance NAND and PRAM hybrid storage design for consumer electronics," *IEEE Trans. Consumer Electronics*, vol. 56, no. 1, pp. 112-118, Feb. 2010.

[15] Y.-J. Lin, C.-L. Yang, H.-p. Li, and C.-Y. M. Wang, "A buffer cache architecture for smartphones with hybrid DRAM/PCM memory," *in Proc. IEEE NVMSA*, 2015, pp. 1-6.

[16] D. H. Kang and Y. I. Eom, "FSLRU: A page cache algorithm for Mobile devices with hybrid memory architecture," *IEEE Trans. Consumer Electronics*, vol. 62, no. 2, pp. 136-143, May 2016.

[17] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual," Parallel Data Lab., Tech. Rep. CMU-PDL-08-101, May 2008.

[18] H. Chung, B. H. Jeong, B. Min, Y. Choi, B.-H. Cho, J. Shin, J. Kim, S. Jung, J.-M. Park, Q. Wang, Y.-J. Lee, S. Cha, D. Kwon, S. Kim, S. Kim, Y. Rho, M.-H. Park, J. Kim. I, Song, S. Jun, J. Lee, K. Kim, K.-W. Lim, W.-R. Chung, C. Choi, H. Cho, I. Shin, W. Jun, S. Hwang, K.-W. Song, K. Lee, S.-W. Chang, W.-Y. Cho, J.-H. Yoo, and Y.-H. Jun, "A58nm 1.8V 1Gb PRAM with 6.4MB/s program BW," *in Proc. IEEE ISSCC*, 2011, pp. 500-502.

[19] T. Lee, D. Kim, H. Park, S. Yoo, and S. Lee, "FPGA-based prototyping systems for emerging memory technologies," *in Proc. IEEE RSP*, 2014, pp. 115-120.

[20] S. Im and D. Shin, "Differentiated space allocation for wear leveling on phase-change memory-based storage device," *IEEE Trans. Consumer Electronics*, vol. 60, no. 1, pp. 45-51, Feb. 2014.

[21] S. J. Han, D. H. Kang, and Y. I. Eom, "Hybrid write buffer algorithm for improving performance and endurance of NAND flash storages," *in Proc. IEEE ICCE*, 2016, pp. 83-84.

Dong Hyun Kang received the B.S. degree in Computer Engineering from Korea Polytechnic University, Korea in 2007 and the M.S. degree in College of Information and Communication Engineering from Sungkyunkwan University, Korea in 2010. He is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Sungkyunkwan University. His research interests include file and storage systems, operating systems, and embedded systems.

Se Jun Han received the B.S. degree in Computer Engineering from Chungnam National University, Korea in 2005 and the M.S degree in College of Information and Communication Engineering from Sungkyunkwan University, Korea in 2015. He is currently a senior software engineer for Samsung Electronics in Korea. His current research interests include embedded operating system, storage system.

Young-Chang Kim received his B.S., M.S., and Ph.D. degree in Computer Engineering from Chonbuk National University, Korea in 2001, 2003, and 2009, respectively. He is currently a Senior Researcher in the Electronics and Telecommunications Research Institute, Daejeon, Korea. His research interests include storage systems, high performance computing, and database management system.

Young Ik Eom received his B.S., M.S., and Ph.D. degrees in Computer Science and Statistics from Seoul National University, Korea in 1983, 1985, and 1991, respectively. From 1986 to 1993, he was an associate professor at Dankook University in Korea. He was also a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine, from Sep. 2000 to Aug. 2001. Since 1993, he has been a professor at Sungkyunkwan University in Korea. His research interests include virtualization, operating systems, cloud systems, and UI/UX systems.