Microprocessors and Microsystems 50 (2017) 222–236

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

An effective pre-store/pre-load method exploiting intra-request idle time of NAND flash-based storage devices

Jin-Young Kim[a,b], Tae-Hee You[b], Hyeokjun Seo[b], Sungroh Yoon[c], Jean-Luc Gaudiot[d], Eui-Young Chung[b,*]

[a] *Flash Design Team, Device Solution Division, Samsung Electronics, Gyeonggi-Do 445-701, Korea*
[b] *Department of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea*
[c] *Department of Electrical and Computer Engineering, Seoul National University, Seoul 151-744, Korea*
[d] *Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697-2625, USA*

ARTICLE INFO

Article history:
Received 27 November 2015
Revised 24 December 2016
Accepted 12 March 2017
Available online 18 March 2017

Keywords:
Storage
Solid-state disk
NAND flash memory
Read/write cache
idle time

ABSTRACT

NAND flash-based storage devices (NFSDs) are widely employed owing to their superior characteristics when compared to hard disk drives. However, NAND flash memory (NFM) still exhibits drawbacks, such as a limited lifetime and an erase-before-write requirement. Along with effective software management, the implementation of a cache buffer is one of the most common solutions to overcome these limitations. However, the read/write performance becomes saturated primarily because the eviction overhead caused by limited DRAM capacity significantly impacts overall NFSD performance. This paper therefore proposes a method that hides the eviction overhead and overcomes the saturation of the read/write performance. The proposed method exploits the new intra-request idle time (IRIT) in NFSD and employs a new data management scheme. In addition, the new pre-store eviction scheme stores dirty page data in the cache to NFMs in advance. This reduces the eviction overhead by maintaining a sufficient number of clean pages in the cache. Further, the new pre-load insertion scheme improves the read performance by frequently loading data that needs to be read into the cache in advance. Unlike previous methods with large migration overhead, our scheme does not cause any eviction/insertion overhead because it actually exploits the IRIT to its advantage. We verified the effectiveness of our method, by integrating it into two cache management strategies which were then compared. Our proposed method reduced read latency by 43% in read-intensive traces, reduced write latency by 40% in write-intensive traces, and reduced read/write latency by 21% and 20%, respectively, on average compared to NFSD with a conventional write cache buffer.

1. Introduction

Along with the increased number of various portable devices, NAND flash-based storage devices (NFSDs) such as the solid-state drive (SSD), universal flash storage (UFS), and embedded multimedia card (eMMC) [1] have been increasingly widely employed [2,3]. NAND flash memories (NFM) have many advantages, such as nonvolatility, small form factor, low power consumption and high performance. However, problems caused by physical limitations, such as limited program/erase cycles and an erase-before-write requirement, still need to be resolved. These issues are typically observed when short write requests with randomly distributed addresses are applied to an NFSD. For example, even if only a part of one page of NFM would need to be written, the write latency would remain equal to the time required to access an entire page. Moreover, if the target page has already been written and contains data to be modified, the time required to read the page would have to be added to the write latency.

Consequently, most modern NFSDs are equipped with dynamic random access memory (DRAM) as a cache in an attempt to limit performance degradation. In the past, the cache in storage systems originally functioned as a speed-matching buffer between hard disk drives and their hosts [4]. However, an NFSD cache plays a similar role, but its impact is greater because NFMs cannot support in-place updates. In other words, the cache not only temporarily stores host data before transferring them to NFMs, but also reduces the number of updates forwarded to NFMs along with the erase operations, thereby causing long access latencies.

From the perspective of data management, an NFSD cache is different from the cache commonly used in main memory systems.

[*] Corresponding author.
E-mail address: eychung@yonsei.ac.kr (E.-Y. Chung).

http://dx.doi.org/10.1016/j.micpro.2017.03.007
0141-9331/© 2017 Elsevier B.V. All rights reserved.

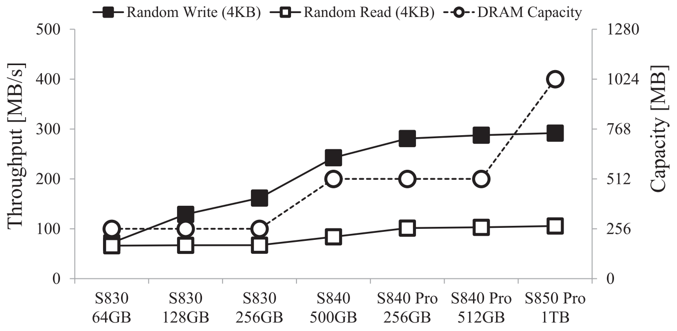


Fig. 1. Performance of SSDs in product generation (□, ■: throughput, ○: capacity).

It focuses on managing write data rather than read data because of the long write latency of NFM. Hence, an NFSD cache is usually used as a write buffer that only stores data in response to write requests.

Fig. 1 shows the performance evaluation of solid-state drives (SSDs) according to their generations. The values were collected using SSD products manufactured by a leading company over a three-year period [1,5]. Because DRAM is used as a write buffer, the write performance of SSDs for random data improved fourfold during that period. However, the write performance subsequently showed no further improvement. At the same time, the read performance did not significantly improve.

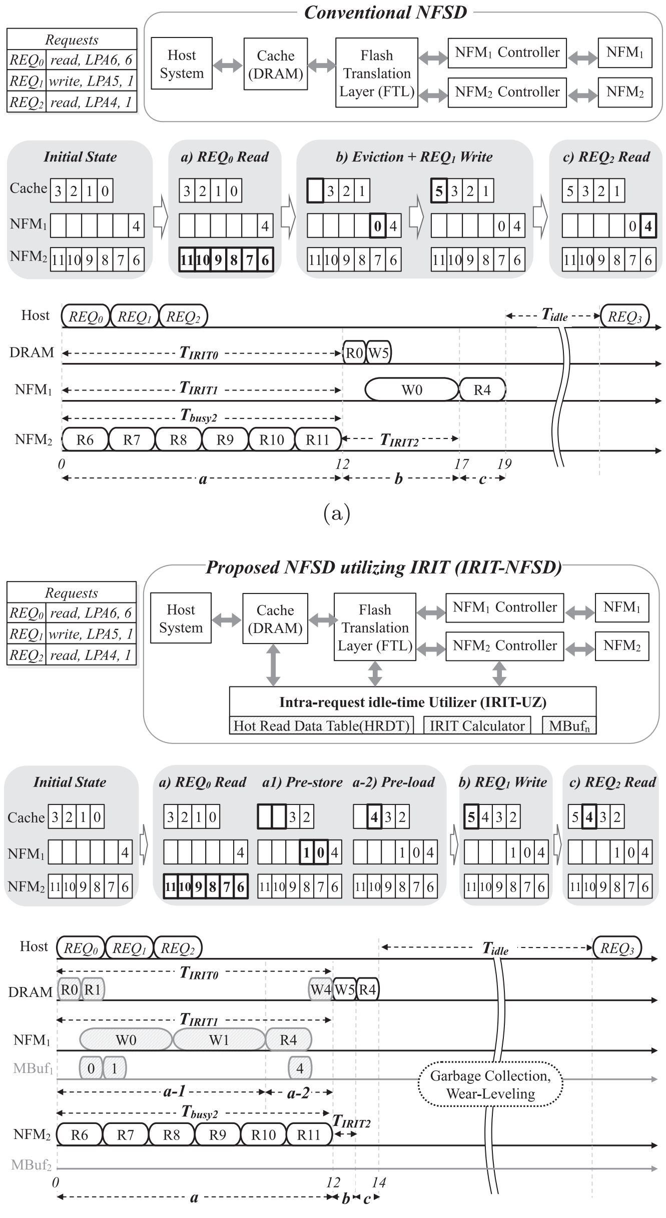
In terms of write performance, even if DRAM is used as a write buffer, it is limited in capacity, and this eventually causes data to be evicted from DRAM to NFM, consequently degrading the write performance. In Fig. 1, the saturated write performance implies that NFSDs are still adversely affected by eviction overhead regardless of the continuously increasing capacity of DRAM. Many studies aimed to improve the write performance by lessening the number of evictions [6–14]; however, NFSDs continued to experience potential write performance saturation since the eviction overhead is not hidden. Accordingly, an effective method for hiding eviction overhead is sorely needed.

In terms of read performance, a cache that is used as a write buffer has only a marginal chance of improving the read performance unless data cached by writing is again requested by a read action within a short interval. A data management method that loads missed read data to a cache (known as *read-allocation*) was used in several approaches [6–14]. However, the method serves to improve write performance by reducing write eviction overhead using secured clean pages. If read-allocation is used for read performance improvement, it should be performed carefully because it can actually increase the eviction overhead.

In this paper, we propose a new idle time utilization method for the purpose of hiding eviction/insertion overhead and overcoming saturated read/write performance. Our contribution is threefold:

First, we exploit novel unutilized idle time in NFSD (called *intra-request idle time (IRIT)*) and utilize it to hide the overhead caused by data migration between an NFSD cache and NFMs. In most NFSDs consisting of multiple NFMs, some components should be idle while a component is operating. For the idle components, we propose a new idle time duration as an IRIT (see Fig. 2), which differs from common idle times in that all components within an NFSD are idle without any request from the host. Moreover, unlike the common idle times that utilize operations, such as merge operations [15–17], garbage collection [18–22], and wear-leveling, we propose using IRIT to migrate data from one component to another (*e.g.,* NFM to a cache and vice versa).

Second, we propose a new eviction scheme termed *pre-store*. In our proposed NFSD, the IRIT of all components is exactly computed on the basis of command and latency information of NFMs and DRAM, and the pre-store scheme migrates the dirty page data in the cache to NFMs in advance during IRIT. The scheme reduces eviction overhead by maintaining a sufficient number of clean pages in the cache and thereby prevents performance degradation.



(a)

(b)

Fig. 2. Timing flow of (a) a conventional NFSD and (b) the proposed NFSD utilizing IRIT.

Third, we propose a new insertion scheme termed *pre-load*, which also utilizes IRIT. Unlike previous methods that perform read-allocation, the proposed pre-load scheme selects data, termed *hot read data* as the data are read into the cache. It loads the selected hot read data into the cache in advance by utilizing IRIT. The pre-loaded read data increases the efficiency of the read data and improves the read performance.

Our proposed method can be independently applied to most cache management schemes. We quantitatively confirmed the effectiveness of our scheme by integrating it into two different cache management methods, which were then compared.

The remainder of this paper is organized as follows: in Sections 2 and 3, we present our motivation and proposed methods, respectively. We show our experimental results in Section 4 and our conclusions in Section 5.

2. Motivation and I/O pattern analysis

2.1. Example of IRIT utilization

Fig. 2 shows an example to demonstrate the inherent benefit behind using IRIT. One is a conventional NFSD; the other is our proposed NFSD which utilizes IRIT (IRIT-NFSD). The smallest box represents a page. A DRAM cache and each NFM have four pages and six pages, respectively. We define a request as {type, logical page address (LPA), data length} and assume that three requests {read, LPA6, 6}, {write, LPA5, 1} and {read, LPA4, 1} are sequentially issued to the two NFSDs. The results of these requests demonstrate why IRIT-NFSD is superior to the conventional NFSD.

For the purpose of evaluating the performance of the proposed method, we assumed latencies for DRAM read, DRAM write, NFM read, and NFM write of 1, 1, 2, and 5, respectively. In Fig. 2, the subfigures *a*, *b*, and *c*, respectively, represent the processing period of REQ_0, REQ_1 and REQ_2, respectively. For REQ_0 and REQ_2, the conventional NFSD processes R6-R11 and R4 from NFM_1 and NFM_2, and W5 for REQ_1 are processed after flushing LPA0 from DRAM to NFM_1, of which the total time is 19. The data flushing of DRAM is managed by a Least Recently Used (LRU) policy. On the other hand, in IRIT-NFSD, only R6-R11 for REQ_0 is processed from NFM; W5 and R4 for REQ_1 and REQ_2 are processed from DRAM without accessing NFMs. Thus, the total time of IRIT-NFSD is 14, which is shorter than that of the conventional NFSD. This performance improvement is due to the fact that the pre-store/pre-load operation utilizes IRIT, as demonstrated in detail in the next sections.

2.2. Definition of intra-request idle time

In Fig. 2, T_{busy_n} and T_{idle} represent the busy time of each component (the time during which a component operates) and common idle time. T_{idle} is frequently referred to as the *"idle time"*, which is the time of the state during which all components within the NFSD, including cache and NFMs, are idle with no request from the host. In other words, it is the inter-request idle time between one request and the next request. Utilizing the idle time has been proposed in commodity products as well as in many research efforts aimed at enhancing the performance [18–26]. Mostly, garbage collection and wear-leveling are performed during the idle time since they are quite time consuming (10–1000 ms [23,24]). However, it is very difficult to predict the idle time because an NFSD does not know when it can the next request to arrive. Thus, many methods that predetermine a specific time-out [23] or predict idle time duration [24] have been proposed; nevertheless, time delay caused by the improper use of *"idle time,"* is inevitable.

In this paper, intra-request idle time (IRIT) is newly defined (T_{IRIT_n} of Fig. 2). This is the duration of the idle time experienced by the individual components of which an NFSD is composed when an NFSD serves data requested from a host. Although most of the individual IRIT intervals are very short (from 10 ns to 10 ms), the accumulated time is not negligible considering its frequent occurrences in most NFSDs with multiple NFMs. In the example in Fig. 2, the performance improvement of IRIT-NFSD is attributed to IRIT utilization. During the *a* period shown in Fig. 2, the conventional NFSD performs only one host request (REQ_0). On the other hand, IRIT-NFSD performs several operations, including a host request and two migration requests for pre-storing/pre-loading, which reduces the latencies of subsequent requests. More specifically, it is assumed that REQ_1 should be handled after an eviction operation, IRIT-NFSD secures clean pages in advance through pre-storing while handling REQ_0 (*a-1* period). The clean pages help the next request (REQ_1) to be handled without eviction. As for REQ_1, REQ_2 is already assisted by the pre-load that migrates hot (frequently referenced) read data to the cache in advance during an *a-2* period. Consequently, the proposed IRIT-NFSD reduces the execution time of REQ_1 and REQ_2.

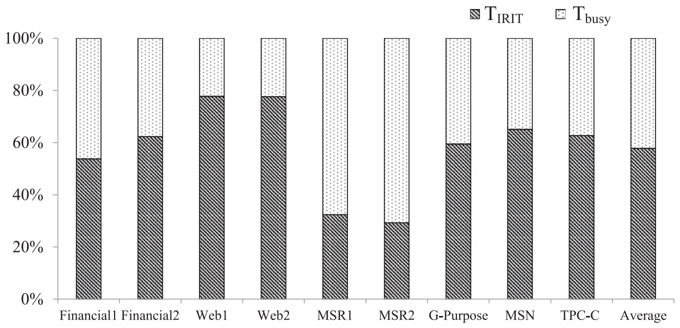


Fig. 3. Ratio of busy time vs. intra-request idle time in an NFSD.

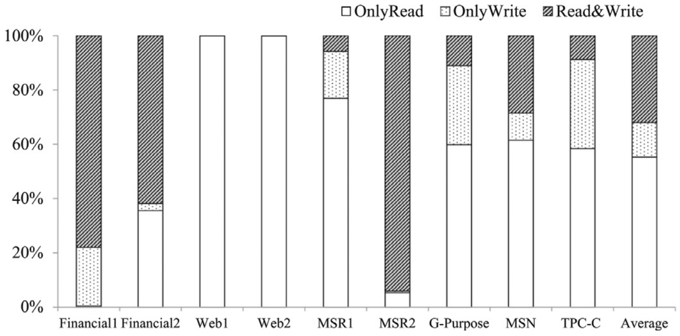


Fig. 4. The ratio of the request operation depending on the addresses.

Although Fig. 2 shows only a simple example, a real NFSD would exhibit a much larger amount of IRIT. We confirmed the amount of IRIT in a quantitative manner by extracting the ratio of IRIT to busy time in an NFSD with four channels, as shown in Fig. 3, where T_{busy} and T_{IRIT} represent the sum of the busy times and IRITs of all components. As depicted in the figure, the amount of IRIT is up to 50% of the total serviceable time, which means that the amount of data processed by an NFSD can be doubled if this IRIT is ideally utilized. However, to the best of our knowledge, this fact has not yet been exploited.

Unlike *"idle time"*, IRIT can easily be calculated based on information of only the logical address and data length of the inserted host request. Thus, IRIT can be used with no time overhead through exact calculation using the IRIT calculator shown in Fig. 2b. However, utilizing the IRIT in an NFSD is limited by the busy status of the cache because the cache plays the role of both the source and destination, transferring data to-and-from multiple NFMs. Hence, we introduce a small buffer (MBuf in Fig. 2b), which enables each component to individually exploit its IRIT by subdividing operating times. Details of MBuf are given in Section 3.2.

2.3. Hot read data detection for pre-loading

Observing the I/O pattern characteristics is one of the most important aspects of this work. Two interesting properties of traces were found by analyzing traces gathered from various computing systems; the first is shown in Fig. 4 (see Table 4 for the details of the traces used).

Fig. 4 represents the ratio of the number of read only, write only, and overlapped addresses to the overall addresses given to the NFSD. For clarity, addresses with no access are excluded from Fig. 4. For other types, we found that only a small portion

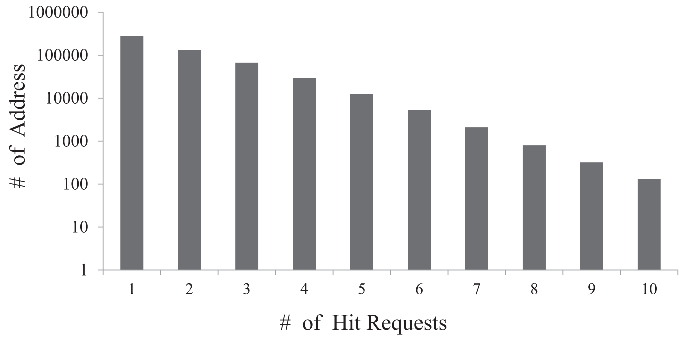


Fig. 5. The number of read requests accessing the same address (trace: MSN).

of the total address space was write-only, whereas a large portion was read-only or both. This indicates that utilization of the write cache buffer without considering read data may present significant hindrances to performance improvement.

The second property we found is depicted in Fig. 5. We measured how frequently read requests accessed the same address using a trace known as *MSN*. In Fig. 5, the x-axis signifies the number of read requests accessing the same address, whereas the y-axis represents the number of addresses accessed by the requests.

If the number of read requests that repeatedly require the same address is large, the graph would be biased to the right and a large hit ratio would be achieved when the cache performs read-allocation. However, as shown in Fig. 5, the graph is biased to the left, which signifies that the data written by read-allocation causes only a marginal read hit on the cache.

Since a read-allocation operation usually causes additional eviction overhead (as mentioned in the previous section), it is quite crucial to accurately select hot read data, which are supposed to be located on the right side of Fig. 5. We paid particular attention to selecting which data is to be inserted and pre-loaded into the cache in order to minimize the losses due to frequent eviction operations that are observed in conventional read-allocation schemes. Eventually, we observe that the proposed scheme is superior in that it fully utilizes the cache with minimal loss.

3. Proposed method

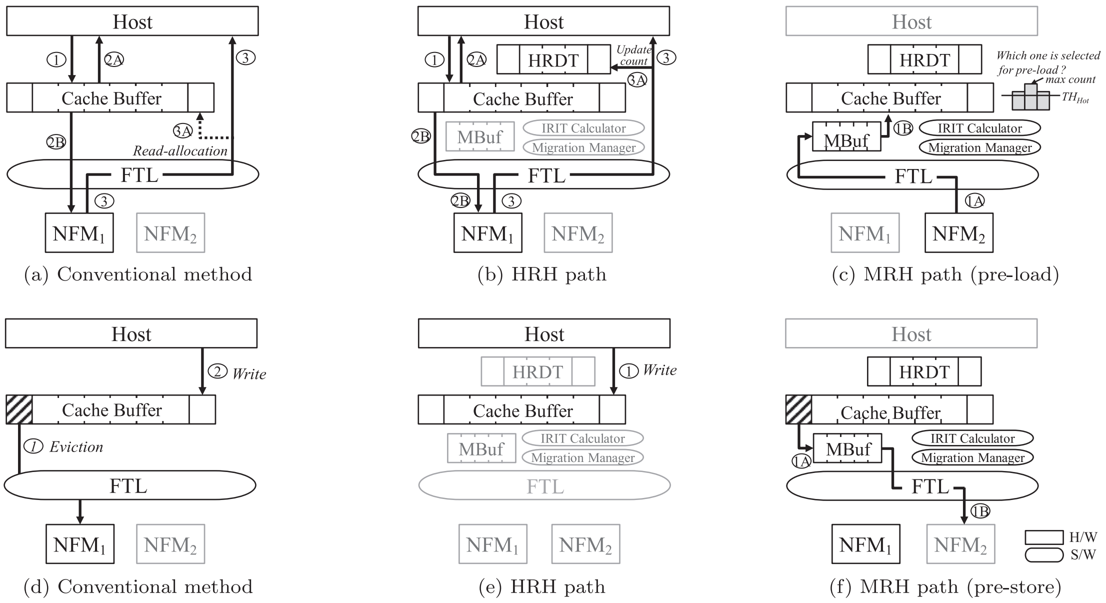
3.1. Overview

In this paper, we propose a new concept referred to as *pre-store/pre-load*. In short, pre-store/pre-load is an eviction/insertion scheme between the cache and NFMs which makes use of the IRIT. Pre-store stores the dirty page data to NFM in advance by exploiting IRIT to maintain a sufficient number of clean pages in the cache. Pre-load detects hot read data (the data frequently read in the cache or to be read in the near future) and loads the detected hot read data to a cache in advance by utilizing IRIT. Our NFSD requires several new modules to implement the proposed method: a hot read data table (HRDT), an IRIT calculator, a migration manager, and a migration buffer (MBuf).

We classify the requests handled in NFSD into two types: the first is sent from the host (the *host request*); the other is generated internally to migrate data between the cache and NFMs (the *migration request*). On the basis of these requests, our method has two operation paths named *Host Request Handling* (HRH) and *Migration Request Handling* (MRH), which are independently executed in parallel. The HRH path is not significantly different from conventional cache operations in terms of handling a host request. On the other hand, the MRH path is a unique operation that utilizes IRIT for improved read/write performance.

Fig. 6 compares data handling by a conventional method and the proposed method to explain the data flow in an NFSD equipped with our scheme. The overall operational flow in the proposed method is presented in Fig. 7. Fig. 6a–c compare the data flow of a read operation in the conventional method with that in our proposed method. In Fig. 6a, when a host read request is given (1). the conventional method checks whether the requested data exist in the cache. If they do, it serves the data to the host (2A); otherwise, it brings the read data from an NFM through FTL to the host (2B–3). Depending on the data management policy of the cache, read-allocation may be performed (3A).

Fig. 6b is the same as the conventional method (Fig. 6a) except for the HRDT-related path. In the HRH path, the proposed method updates only the read count of HRDT (3A) and does not perform read-allocation. Instead, hot read data detected by HRDT



(a) Conventional method

(b) HRH path

(c) MRH path (pre-load)

(d) Conventional method

(e) HRH path

(f) MRH path (pre-store)

Fig. 6. Data flow of read/write operation according to each method ((a)–(c): read, (d)–(f): write).

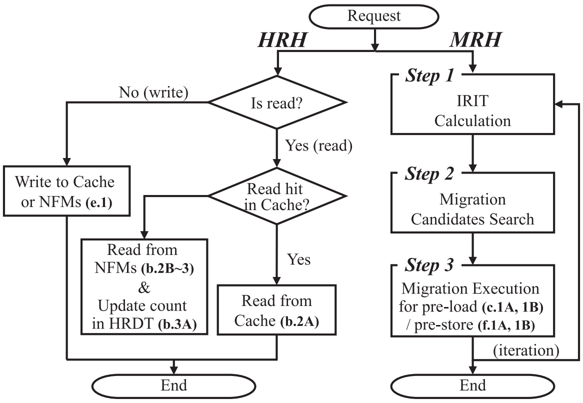

Request
HRH
MRH
No (write)
Is read?
Yes (read)
Write to Cache or NFMs (e.1)
Read hit in Cache?
Yes
Read from NFMs (b.2B~3) & Update count in HRDT (b.3A)
Read from Cache (b.2A)
End
Step 1
IRIT Calculation
Step 2
Migration Candidates Search
Step 3
Migration Execution for pre-load (c.1A, 1B) / pre-store (f.1A, 1B)
(iteration)
End

Fig. 7. Overall operation flow of the proposed NFSD.

are loaded from the NFMs to the cache in advance through the MRH path (Fig. 6c), which is the core of the pre-load operation.

The MRH path consists of three major steps, as shown in Fig. 7. Each of these is explained in Fig. 6c. When a host request is given to NFSD, a command list for NFMs and the cache is generated. In Step 1, the IRIT of all the components is calculated based on the command lists, and the values are given to the migration manager by the IRIT calculator. The migration manager continuously observes the status of the cache, MBuf, and NFMs and detects any data movement for pre-store/pre-load. During Step 2, the migration manager determines when and how the data movement can take place without causing time overhead. Then, in Step 3, the chosen candidate is migrated from one component to another (*e.g.,* from an NFM to the cache or vice versa). Finally, Steps–3 are iteratively run until there no further migration candidates remain, or until IRIT is exhausted.

The write operation differs slightly from the read operation. When a host write request is given, the conventional method (Fig. 6d) writes new data (2) after evicting a dirty page from the cache (1), when the cache buffer is filled with dirty pages. This, however, causes garbage collection and also performance degradation due to the long latency of the program/erase operations of NFM. Our proposed method eliminates the need to perform an eviction when handling a host request (Fig. 6e) by maintaining a sufficient number of clean pages in the cache (Fig. 6f). Similar to pre-load, pre-store does not incur time overheads for migration on account of the MRH path utilizing IRIT.

3.2. Migration buffer

Even though the IRIT of all components is computed by the IRIT calculator, a certain time overhead is inevitable if two components for data migration (the source component and destination component, such as NFM and the cache) do not have sufficient IRIT. In most NFSDs with multiple NFMs and a single cache, utilization of the IRIT in NFSD, especially, depends on the status of the cache. Hence, the proposed NFSD is equipped with a small buffer named *Migration Buffer* (MBuf) between the NFM and the cache to separate their operation domains.

If the cache buffer has insufficient IRIT when migrating hot read data from NFM to the cache without MBuf, the whole system would either have to wait until the migration process is completed or the migration process would have to be delayed until the cache has sufficient IRIT. However, with MBuf, each migration operation (the migration between NFM and MBuf and the migration between MBuf and the cache) can be individually executed. As a result, MBuf enables each component to individually exploit its IRIT, thus improving the utilization of IRIT.

Table 1
Example of IRIT calculation.

	Command	# commands	Latency	T_{busy}	T_{IRIT}
Cache	DRAM READ	5	2	10	30
NFM_1	NFM READ	10	4	40	0
NFM_2	NFM READ	10	2	20	20
NFM_3	NFM READ	10	2	20	20
NFM_4	–	–	–	0	40

Despite a small capacity (see Fig. 20), MBuf is effectively used, which is because IRIT occurs rather frequently as shown in Fig. 3, and data in MBuf is continuously migrated to NFM or the cache during IRIT. We assume that MBuf is composed of SRAM in the NFM controller and uses a single MBuf per channel. Thus, the embedded SRAM is more than ten times faster than when DRAM is used as a cache [27].

3.3. Host request handling (HRH)

The HRH path is slightly different from the conventional method in terms of adding the HRDT. HRDT performs the detection of hot read data for pre-load, which is activated only when a host request is read and not found in the cache, as shown in Fig. 7. HRDT is implemented with a priority queue and records the logical page address (LPA) of the missed read request in the cache and read count. If HRDT already has the LPA, HRDT increases the corresponding read count; otherwise, the LPA is added in HRDT. However, if HRDT is full when adding a new LPA, an LRU LPA is deleted from the HRDT and the new LPA is inserted into the Most Recently Used (MRU) position.

The logical pages whose read counts are larger than a given threshold (TH_{Hot}) are considered migration candidates for pre-load. When the conditions for a pre-load operation are met, the logical page with a maximum read count among the migration candidates is deleted from HRDT and migrated to cache from the NFMs. The migration operation and conditions are described in detail in Section 3.4.

3.4. Migration request handling (MRH)

3.4.1. Step 1: IRIT calculation

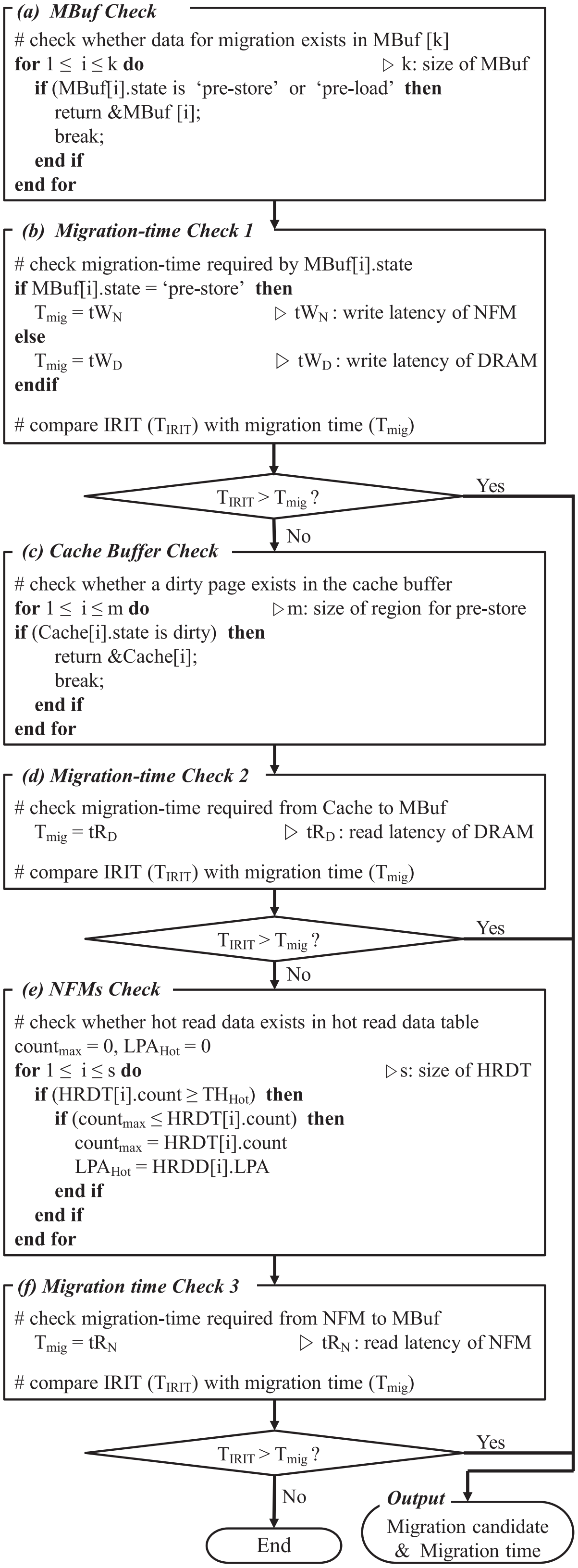
This step can best be explained by an example (Table 1). We assume that an NFSD consists of four NFMs and a single cache and the read latencies of DRAM and NFM are five and ten, respectively.

First, based on information about the commands of NFMs and DRAM, the busy time of each component (T_{busy}) and maximum busy time ($T_{busy_{max}}$) are computed. T_{busy} is computed as the product of the latency of each operation by the number of commands used, where $T_{busy_{max}}$ is the maximum value among the calculated busy times ($T_{busy_{max}}$ is 40 in Table 1). Finally, the IRIT of each component (T_{IRIT}) is calculated by subtracting T_{busy} from $T_{busy_{max}}$. In Table 1, the IRIT of NFM_1 has a maximum busy time of zero, whereas IRIT of NFM_4, which is not performing any operation, is equal to the maximum busy time of 40.

3.4.2. Step 2: Migration candidate search

In this step, we determine which migration candidate (page) is most suitable to perform pre-store/pre-load without time overhead. Fig. 8 shows the iterative search process of Step 2. The process checks whether a candidate for migration exists in each component and the candidate causes time overhead (in the order of MBuf, the cache, and NFMs).

First, the proposed process seeks to avoid an excessive accumulation of pages in MBuf because of its limited capacity. The migration manager selects a page in MBuf as a migration candidate



```
(a) MBuf Check
# check whether data for migration exists in MBuf [k]
for 1 ≤ i ≤ k do                                   ▷ k: size of MBuf
    if (MBuf[i].state is 'pre-store' or 'pre-load' then
        return &MBuf [i];
        break;
    end if
end for

(b) Migration-time Check 1
# check migration-time required by MBuf[i].state
if MBuf[i].state = 'pre-store' then
    $T_{mig} = tW_N$                       ▷ $tW_N$ : write latency of NFM
else
    $T_{mig} = tW_D$                       ▷ $tW_D$ : write latency of DRAM
endif
# compare IRIT ($T_{IRIT}$) with migration time ($T_{mig}$)

(c) Cache Buffer Check
# check whether a dirty page exists in the cache buffer
for 1 ≤ i ≤ m do                    ▷m: size of region for pre-store
if (Cache[i].state is dirty) then
        return &Cache[i];
        break;
    end if
end for

(d) Migration-time Check 2
# check migration-time required from Cache to MBuf
    $T_{mig} = tR_D$                       ▷ $tR_D$ : read latency of DRAM
# compare IRIT ($T_{IRIT}$) with migration time ($T_{mig}$)

(e) NFMs Check
# check whether hot read data exists in hot read data table
$count_{max} = 0$, $LPA_{Hot} = 0$
for 1 ≤ i ≤ s do                                  ▷s: size of HRDT
    if (HRDT[i].count ≥ $TH_{Hot}$) then
        if ($count_{max}$ ≤ HRDT[i].count) then
            $count_{max}$ = HRDT[i].count
            $LPA_{Hot}$ = HRDD[i].LPA
        end if
    end if
end for

(f) Migration time Check 3
# check migration-time required from NFM to MBuf
    $T_{mig} = tR_N$                       ▷ $tR_N$ : read latency of NFM
# compare IRIT ($T_{IRIT}$) with migration time ($T_{mig}$)
```

Fig. 8. Algorithm for searching migration candidate (Step 2 in Fig. 7).

considering the state of the page and the order of insertion into MBuf. As shown in Fig. 8a and b, the page in MBuf is classified as one of two states: "pre-store" (a page to be written to NFM) and "pre-load" (a page to be written to cache). However, if there is no migration candidate in MBuf or the migration time (T_{mig}, the time required to migrate the selected candidate) is larger than the IRIT of the component (T_{IRIT}), our method searches for another migration candidate in the cache or NFMs.

The performance of the cache is more extensively affected by an eviction than by an insertion because of the long write latency of the NFM. Thus, we select an eviction page of the cache as the next migration candidate to hide the longer latency. Fig. 8c and d show the process of checking whether a dirty page satisfying the condition for migration exists in the cache.

If a migration candidate for pre-store is not selected in the cache, our method lastly checks HRDT. As shown in Fig. 8e, among the logical pages whose read counts are larger than a given threshold (TH_{Hot}), the logical page with the maximum read count is selected as a migration candidate for pre-load. If the T_{IRIT} of the selected candidate is larger than T_{mig}, the data migration can be performed, and the candidate ultimately selected is transferred to Step 3.

3.4.3. Step 3: Migration execution

Step 3 is the process of executing the selected candidate. Because it is already confirmed through Step 2 that the selected candidate does not cause time overhead for migration, the migration execution time is hidden from the host. Moreover, because there is an MBuf in each NFM controller, migration operations are independently executed for each channel, which means that the IRIT of all components are utilized.

3.4.4. Iteration process

The procedure from Steps 1 to 3 is iterative, and we calculate IRIT at each iterative cycle to check whether IRIT is sufficient for performing multiple migration operations. In the MRH path, our migration process executes iteratively until IRIT is exhausted or no migration candidate exists.

3.5. Case study

Our pre-store/pre-load scheme utilizing IRIT can be adopted in most cache management methods such as those in [6–14]. We conducted a case study by applying our scheme to two cache management schemes, Clean-first LRU (CFLRU) and Temporal and spatial locality aware CLOCK (TS-CLOCK).

CFLRU [7] is an NFM-aware eviction method. It is a variant of LRU that chooses a clean page as a victim page, rather than a dirty page. Whereas the eviction of a clean page can be performed without NFM operation, dirty pages need to be written to NFM, sometimes causing additional garbage collection.

TS-CLOCK is a variant of the traditional CLOCK algorithm for temporal locality. It exploits spatial locality by using the reference count and two hands [13]. The method minimizes writes to NFMs, by adopting a victim selection algorithm to generate a flash memory-friendly sequential write pattern.

Fig. 9 displays an example of CFLRU adopting our method, and shows how pages of the CF and working regions cooperate when a pre-store/pre-load is performed. CFLRU divides the cache into two regions: a clean-first region and a working region. The working region consists of recently used pages, whereas the clean-first region consists of candidate pages for eviction [7]. For eviction, CFLRU selects a clean page with the highest priority from the clean-first region. To make use of our method in CFLRU, an additional region called *preemptive region (P-region)* must be defined. The P-region is used to generate clean pages in advance. In other words, before a

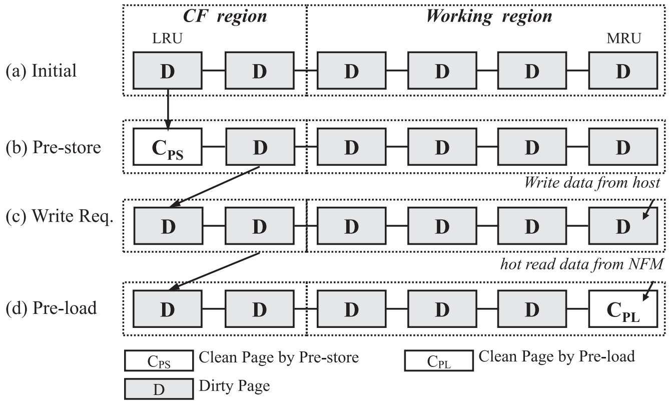


Fig. 9. CFLRU adopting pre-store/pre-load.

Table 2
Timing parameters of DRAM and NFM.

	DRAM [31]	NFM [32]
Read latency	45 ns	75 us
Write latency	45 ns	1300 us
Write bandwidth	1.6 GB/s	6.15 MB/s

Table 3
Specifications of used NFMs.

Category	Value
Capacity	2 GByte/channel
Page size	8 KByte
Number of blocks	1024
Number of page per blocks	256
Number of sectors per page	16

cache requires eviction, dirty pages in the P-region are converted into clean pages in advance from the cache to NFMs through the pre-store of the MRH path. In Fig. 9, we assume that the P-region is the same as the CF region.

(a) We assume that a cache is filled with dirty pages as a result of write requests and they need to be evicted for newly incoming data.
(b) Because there are dirty pages in the P-region, the pre-store scheme needs to migrate data comprising dirty pages to NFMs during IRIT. Pre-store turns a dirty page in the region into a clean page, which continues until either all the IRIT is expended or no dirty page remains in the P-region
(c) If some space for newly incoming data is needed, a clean page in the P-region is selected as a victim. Because the page is cleaned in advance by pre-store, no eviction overhead is generated. Furthermore, the page is closer to the LRU-side of the cache than a page in the working region. Thus, the possibility of the cache exploiting locality is increased.
(d) If hot read data from NFMs must be written to a page for pre-load, the process is exactly the same as (*c*). The process vacates a clean page in the P-region and inserts the hot read data in the MRU position of the cache.

If there is no clean page in either region, the LRU page is evicted. For hot read data, however, the data migration is not activated when there is no clean page in the P-region so as not to cause excessive migration overhead.

4. Experiments

4.1. Experimental set-up

We evaluated the proposed pre-store/pre-load scheme by implementing a trace-driven simulator [24,28,29], which includes a page mapping FTL [30], the storage system configurations, DRAM and NFMs. Table 2 summarizes important timing parameters of DRAM and NFM [31,32]. Our storage system is composed of four NFM channels with a 200 MHz I/O clock [33], DRAM with a capacity of 8 MB and 8GB NFM. The specifications of the NFM are given in Table 3.

To drive the simulator, we prepared several input traces, which are classified into synthetic and real traces in Table 4. We generated various synthetic traces using an in-house trace generator. Our trace generator collected synthetic traces with various localities, by adjusting the sampling size of addresses given to NFSD among the total addresses of 100 MB. Varying the sampling size of addresses means a varying locality, which is used to evaluate performance by varying the locality in Section 4.2. We also used real traces collected from various computer systems. Financial1/2, Web1/2 and MSR1/2, MSN, TPC-C are from [34] and [35], respectively, and G-Purpose was collected from a common PC using DiskMon [36]. They are prepared from various sources ranging from read-intensive (*e.g.,* Web1/2, Financial2) to write-intensive (*e.g.,* Financial1, MRS1). Details of the traces we used are provided in [37].

Before simulation, our simulator fills a 50% region of NFMs with random data. This process induces many additional operations such as garbage collection for the simulation of our method in an environment that closely approximates real situations.

Our experiments compare the following six methods (including our approaches):

- W-LRU: The simplest cache management method, which uses an LRU policy for eviction and inserts only write data into a cache.
- RW-LRU: As with W-LRU, it uses an LRU policy for eviction. However, it inserts not only write data but also read data by performing read-allocation.

Table 4
Used trace information.

Name	Num. of requests	Read ratio (%)	Avg. R-Req length	Avg. W-Req length	Random read (%)	Random write (%)
Synthetic	100,000	Varying	Varying	Varying	Varying	Varying
Financial1	5,334,950	23	5.2	9.5	94	87
Financial2	3,699,195	82	5.6	7.9	94	88
Web1	4,579,810	99	31.6	16.2	0	0
Web2	4,261,706	99	32.3	22.8	0	0
MSR1	1,211,036	12	51.9	15.2	53	72
MSR2	149,864	56	95.0	21.0	22	77
MSN	1,081,879	70	20.0	22.3	0	18
TPC-C	348,645	63	16.3	18.9	0	0
G-Purpose	1,023,244	65	64.0	80.6	58	55

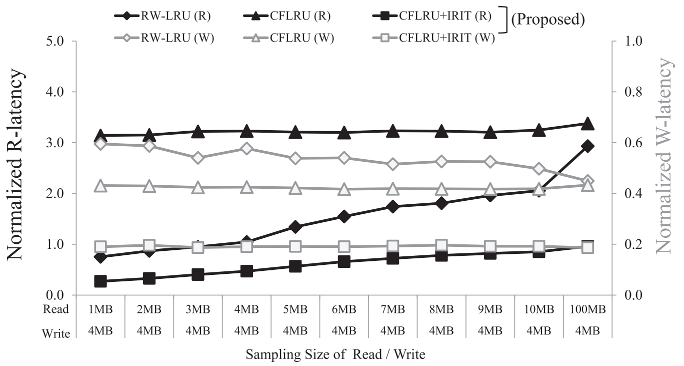


Fig. 10. Average latency by varying read locality normalized to W-LRU (✦, ▲, ■: R-latency, ◊, △, □: W-latency).

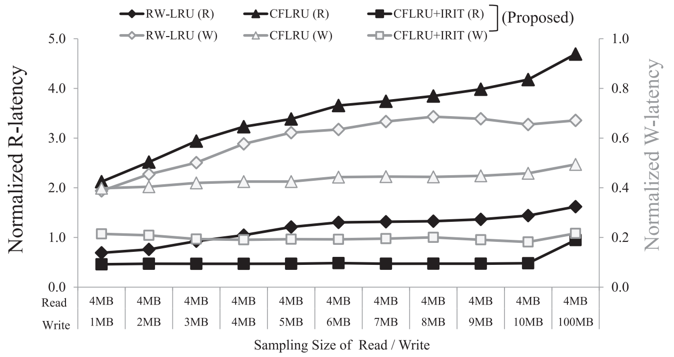


Fig. 11. Average latency by varying write locality normalized to W-LRU (✦, ▲, ■: R-latency, ◊, △, □: W-latency).

- CFLRU: A method proposed in [7], which has the same insertion policy as RW-LRU. However, its eviction policy assigns a higher priority to a clean page than a dirty LRU page.
- TS: A method proposed in [13], which also has the same insertion policy as RW-LRU. Its eviction priority is decided by a CLOCK pointer and a reference count that considers temporal/spatial locality.
- CFLRU+IRIT, TS+IRIT: These methods are an incorporation of our scheme in the methods CFLRU and TS. Basically, they follow the insertion/eviction policies of CFLRU and TS. However, they incur minimal migration overhead because our proposed preload/pre-store scheme secures clean pages in advance by utilizing IRIT.

The size of MBuf, TH_{Hot}, and the ratio of the preemptive region are set to 10 pages, 5, and 0.4, respectively. Further details on the choice of parameters and their definitions are given in Section 4.4.

4.2. Experimental results with synthetic traces

4.2.1. Average latency by varying read locality

Fig. 10 shows the performance of CFLRU+IRIT and the other methods in various read locality conditions. The y-axis on the left and its darker lines correspond to read latency (termed *R-latency*) normalized to that of W-LRU, whereas the y-axis on the right and its lighter lines correspond to write latency (termed *W-latency*). As mentioned above, varying the sampling sizes of the addresses on the x-axis means varying locality.

In terms of R-latency, RW-LRU and CFLRU+IRIT exhibit a similar trend in which R-latency is proportional to read locality. However, the read performance of CFLRU+IRIT is more effective. RW-LRU performs read-allocation that loads all missed read data to a cache, without considering the “hotness” of the read data; thus, the eviction overhead increases. When the read locality is low (when the sampling size of the read is 100 MB in Fig. 10, it is not hot), CFLRU+IRIT inserts a small amount of read data into the cache because the amount of data detected as hot read data is small. This is also confirmed by increasing the read performance gap between RW-LRU and CFLRU+IRIT. On the other hand, the R-latency of CFLRU appears to be independent of the read locality. This is because the read-allocation data residing in the clean pages is evicted by write data because the CFLRU policy does not consider the “hotness” of the read data.

The W-latencies of all schemes appear to be independent of read locality. However, their latencies are all shorter than that of W-LRU owing to clean pages generated by read-allocation or hot read data. CFLRU+IRIT yields the best write performance, owing to the effective read caching by the hot read data and the reduction of migration overhead by IRIT. The write performance of CFLRU is more effective than that of RW-LRU, however, the performance gap between CFLRU and RW-LRU decreases along the y-axis. Because the read locality is lower and the number of read miss events of the cache is increased, the amount of read data allocated to the cache is larger and produces sufficient clean pages. Hence, the write performance is not affected by the eviction policies.

4.2.2. Average latency by varying write locality

Fig. 11 shows the performance trends of all schemes with varying write localities. The exclusion of a low write locality (when the sampling size of writing is 100 MB) does not lead to an increase in the R-latency of CFLRU+IRIT, unlike that of RW-LRU and CFLRU. In case the ratio of read and write localities is similar, the number of clean and dirty pages within the cache is similarly maintained. Hence, when a read-allocation of RW-LRU and CFLRU takes place, the two cache buffer data management methods can load read data to the cache without additional overhead by using the clean pages. However, when the write locality is low and the cache is primarily used for write buffering, eviction of a dirty page for read-allocation is inevitable, thereby causing long R-latency. Conversely, even though the eviction policy of CFLRU+IRIT is similar to that of CFLRU, which selects a clean page as a victim page, the R-latency of CFLRU+IRIT is half that of CFLRU. This is because CFLRU+IRIT not only adopts a conservative policy that loads hot read data only when a clean page exists in the CF region of the cache, but also because it secures clean pages in advance utilizing IRIT. When the sampling size for writing is 100 MB, the read performance of CFLRU+IRIT slightly improves because it cannot also have sufficient clean pages on account of the excessively large amount of write data. However, owing to the conservative policy, CFLRU+IRIT never has a longer R-latency than W-LRU, moreover, neither does it cause a longer W-latency.

In terms of W-latency, the performance of both RW-LRU and CFLRU are improved over that of W-LRU, but are not improved compared to CFLRU+IRIT. The read-allocation of RW-LRU and CFLRU reduces their eviction overhead through clean pages. However, the two schemes are problematic in that they have an absolute space shortage for storing write data compared to W-LRU, which stores only write data. This problem causes poor write performance in the low write locality.

Regardless of varying write locality, the W-latency of CFLRU+IRIT maintains a similar value, which is the smallest figure compared to the others. This is because of the conservative read policy of our scheme, which does not load hot read data to the cache when there are no clean pages in the CF region. It is also caused by pre-store/pre-loads utilizing IRIT, which converts dirty pages generated by write data into clean pages in advance and loads hot read data to the cache without migration overhead.

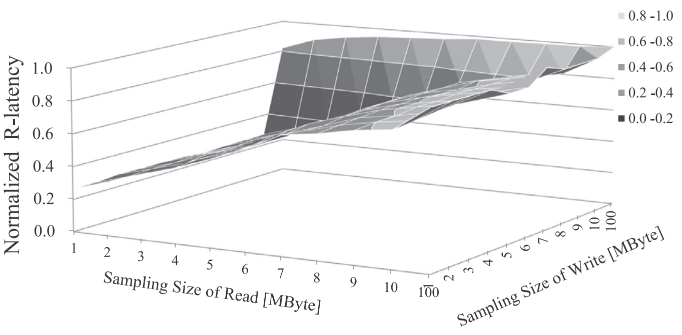


Fig. 12. R-latency distribution of 'CFLRU+IRIT' in varying locality normalized to W-LRU.

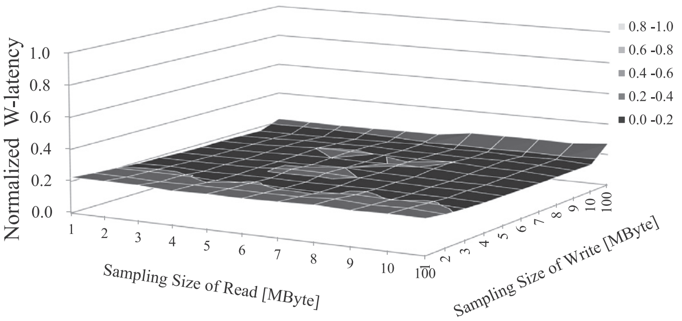


Fig. 13. W-latency distribution of 'CFLRU+IRIT' in varying locality normalized to W-LRU.

4.2.3. Average latency distribution by varying locality

We proved the effectiveness of the proposed method under various locality conditions by measuring the R-latency and W-latency of CFLRU+IRIT by varying both the read locality and write locality simultaneously, shown in Figs. 12 and 13. The numbers on the x-axis and the y-axis of each figure represent the sampling size of the read and write addresses, respectively, and the numbers on the z-axis represent the latencies.

As mentioned earlier, to produce synthetic traces with various localities, our trace generator adjusts the sampling size of addresses given to NFSD among the total address of 100 MB, where varying the sampling size of an address means various localities. For example, if we assume that the capacity of the DRAM cache is 8 MB and the numbers on the x- and y-axes are smaller than 6 MB and 2 MB, respectively, the data of all the addresses given to NFSD can be stored in the DRAM cache and the data can be quickly accessed by using a short DRAM read time. In addition, the case in which the numbers on the x- and y-axes is 4 MB and 4 MB, respectively, is the same as above.

In Fig. 12, the R-latency of CFLRU+IRIT becomes longer as the sampling size of the address increases, but it is not longer than 1.0 normalized to W-LRU. This means that the R-latency of CFLRU+IRIT is shorter than that of W-LRU under all combinations of read/write localities. Moreover, the W-latency of CFLRU+IRIT in Fig. 13 is much shorter than that of W-LRU under the same conditions.

We omit the experimental results of the others except CFLRU+IRIT because of the similarity to those shown in Figs. 10 and 11 and the lack of space. Instead, the minimum, maximum, and average latencies of all schemes are summarized in Table 5. The values in the table represent the latencies of each scheme, and are normalized to those of W-LRU. Therefore, values greater than 1.0 in this table mean that a certain scheme performs worse than W-LRU.

Table 5
Average latencies in varying locality.

	R-latency			W-latency		
	Min	Max	Avg	Min	Max	Avg
W-LRU	1	1	1	1	1	1
RW-LRU	0.6	3.8	1.6	0.3	0.7	0.5
CFLRU	2.1	4.9	3.5	0.4	0.5	0.4
CFLRU+IRIT	0.3	1	0.6	0.2	0.3	0.2

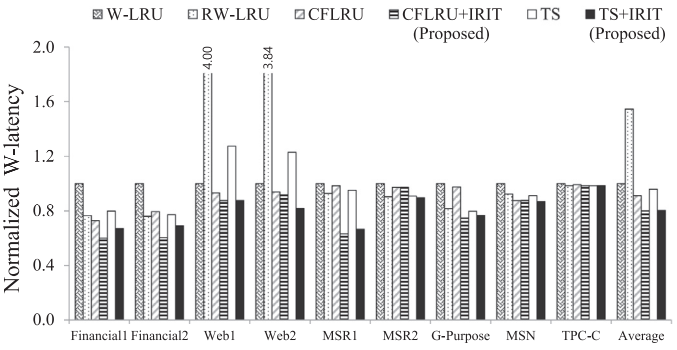


Fig. 14. W-latency normalized to that of W-LRU.

Table 5 shows that CFLRU+IRIT has the shortest R/W-latency among all schemes in all cases, whereas, despite adopting a read-allocation, RW-LRU and CFLRU have longer R-latencies than W-LRU under any condition. The result means that the read-allocation improves the write performance, rather than the read performance.

The experimental results in Sections 4.2.1–4.2.3 indicate that the performance of our method is improved for the following reasons: 1) It reduces eviction overhead through the pre-store scheme utilizing IRIT, which maintains a sufficient number of clean pages within the cache. 2) It adopts a conservative policy that loads hot read data only when a clean page exists in the CF region of cache. 3) It uses the pre-load scheme without migration overhead by migrating hot read data to the cache in advance as much as IRIT allows.

4.3. Real trace experimental results

4.3.1. Average write latency

Fig. 14 shows the degree to which our method improves W-latency. On average, the W-latencies of our method (CFLRU+IRIT and TS+IRIT) are both 20% shorter than that of W-LRU, and are 12% and 16% above those of CFLRU and TS, respectively. Schemes adopting IRIT provide the shortest W-latency in all traces. Especially, our method produces a 40% shorter W-latency in write-intensive traces (Financial1 and MSR1). This is because of clean pages secured in advance through pre-store utilizing IRIT, which reduces migration overhead caused by eviction. On the other hand, RW-LRU, CFLRU and TS have an insufficient number of clean pages in write-intensive traces, which causes the eviction of dirty pages when write data is stored to the cache.

In read-intensive traces (Web1 and Web2), the write performance of our methods outperforms those of CFLRU and TS. The write count caused by data eviction in the methods is the same at zero, however, our methods had a higher write hit in the cache than the others. This can be explained through the hit ratio and average hit count by the loaded read data. This is detailed in Section 4.3.3.

In real traces, in particular, RW-LRU exhibits poor write performance. RW-LRU does not adopt a clean-first policy, even though it has many clean pages such as CFLRU. Specifically, a large amount of read data occupies the storage space of the cache and causes a lack of space for write buffering. Thus, the W-latency of RW-LRU is

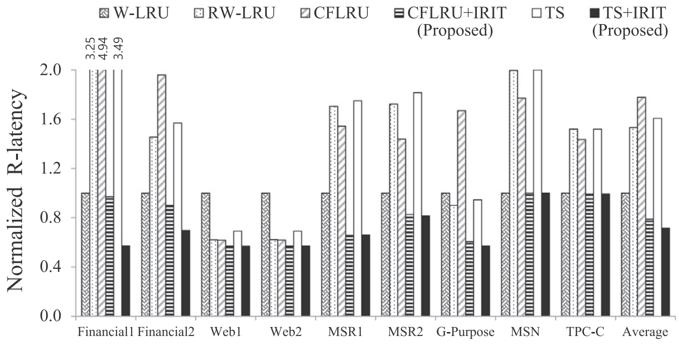


Fig. 15. R-latency normalized to that of W-LRU.

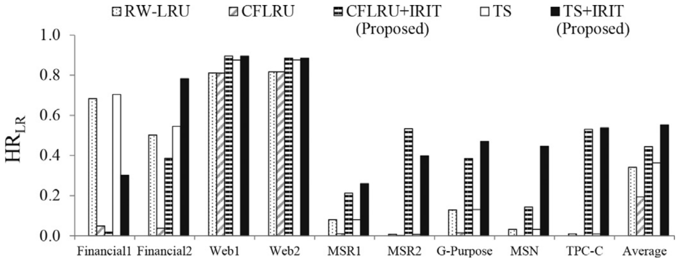


Fig. 16. Hit ratio of loaded read data.

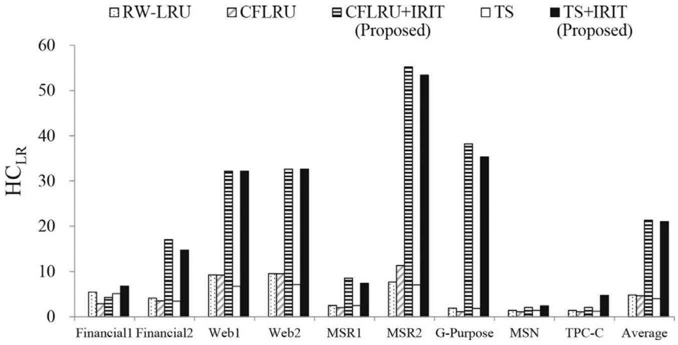


Fig. 17. Average hit count of loaded read data.

longer compared to W-LRU and the W-latencies of other schemes are shorter according to the clean-first policy.

4.3.2. Average read latency

Fig. 15 shows the average read latency of all schemes. On average, CFLRU+IRIT and TS+IRIT have 21% and 28% lower R-latency than W-LRU, respectively, and they outperform CFLRU and TS by 56% and 55%, respectively. Moreover, in read-intensive traces (Web1, Web2), the two methods both produce R-latencies that are 43% shorter than that of W-LRU.

RW-LRU, CFLRU and TS perform read-allocation. However, their major role is not to raise the hit ratio of the read data but to reduce write latency. In most traces, they have longer R-latencies than W-LRU excluding read-allocation. The poor performance has two causes: inefficient read-allocation and the eviction overhead it incurs. Hence, even though the three methods have different eviction policies, their R-latencies are lower than that of W-LRU.

On the other hand, our method secures sufficient clean pages in advance through the pre-store scheme utilizing IRIT and manages loaded read data with its conservative policy that does not induce additional operations and migration overhead for loading read data. Moreover, they improve the quality of the loaded read data by selecting hot read data. The high availability of the loaded read data is described in detail in the next section.

Our experimental results confirm that the proposed method substantially improves read performance for most traces. However, in Financial1 trace, the W-latency of our method decreases by 40% compared to that of W-LRU, whereas the improvement in the R-latency is negligible. This is because Financial1 consists of overlapped addresses for read and write requests, as shown in Fig. 4, and a large amount of data selected as hot read data is already stored in the cache by write buffering.

The R-latencies of our methods barely improve in the MSN and TPC-C traces. The traces are extracted from servers with large capacities and their addresses are usually evenly distributed. Hence, with our methods, the amount of hot read data is small and the hit ratio of the cache is also low. The low read hit count caused by their address distributions is reflected not only in our methods but also in RW-LRU and CFLRU.

4.3.3. Hit ratio and hit count of loaded read data

We define two new metrics to show the relationship between the amount of read data loaded to the cache and the improvement in read performance. They are the hit ratio of loaded read data (HR_{LR}) and the average hit count of loaded read data (HC_{LR}) and are defined as:

$$HR_{LR} = \frac{N_{HR}}{N_{LR}}, \quad HC_{LR} = \frac{C_{HR}}{N_{HR}} \tag{1}$$

where N_{LR}, N_{HR} and C_{HR} indicates the amount of read data loaded to the cache, the amount of read data hit among the loaded data, and the total data access count by the hit read data, respectively. HR_{LR} shows how effectively the read data loaded to the cache are utilized. If the hit rate by loaded hot read data increases, HR_{LR} is close to one. HC_{LR} shows how often the loaded hot read data is requested from the host. Even if $HR_{LR}$is low, the read latency can be improved if HC_{LR} is large.

Fig. 16 shows the value of HR_{LR}, which implies the amount of read data hit among read data cached by read-allocation or hot read data detection. High HR_{LR}s of CFLRU+IRIT and TS+IRIT prove that the effectiveness of the cached data is improved compared to simple read-allocation. HR_{LR}s of CFLRU and TS are lower than that of RW-LRU because they evict clean pages containing cached read data. Moreover, in Financial1/2, RW-LRU outperforms the other schemes, because the traces consist of overlapped addresses for read and write requests and a large number of read-allocations causes an increase in a write hit.

If we consider HC_{LR} shown in Fig. 17, Financial1 is no longer the worst case for our method. Because HC_{LR} presents the cumulative count of cached hit data, the higher value means the quality of cached data has improved. Except for Financial1, the HC_{LR} of our methods is the highest. Although our methods only load a small amount of read data, the data has a high probability of being accessed and of causing large hit counts. HR_{LR}s of our methods are 2.3 times and 1.5 times higher, respectively, compared to that of CFLRU and TS, and their HC_{LR}s are up to 5 times higher than the other methods.

For the reason mentioned above, HC_{LR} and HR_{LR} of our methods are low in Financial1. According to our analysis, write and read addresses of Financial1 access the same addresses quite often (they are occupied by 78% of the total requests). Hence, read hits occur not only from loaded data or read allocated data, but also from write buffered data.

4.3.4. Lifetime of NFMs

The lifetime of NFMs is inversely proportional to the number of erase operations. Fig. 18 shows the erase count (E_C) normalized to that of W-LRU.

On average, the other methods except RW-LRU perform a similar number of erase operations as W-LRU. E_Cs of CFLRU and

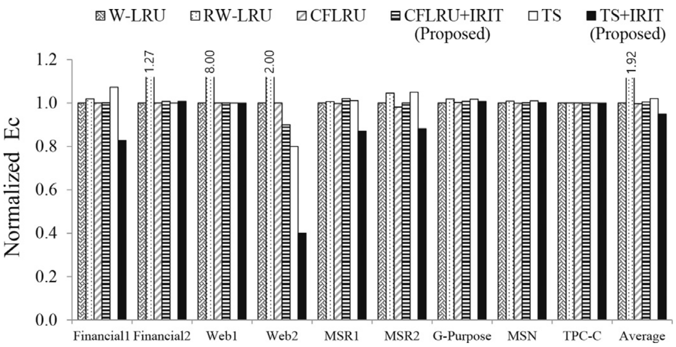


Fig. 18. Erase count normalized to that of W-LRU.

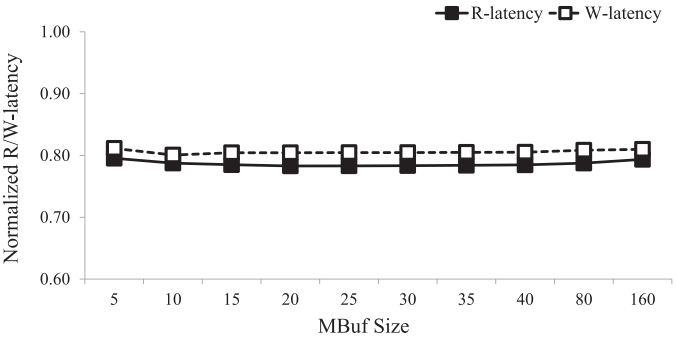


Fig. 20. Average latency by varying MBuf size.

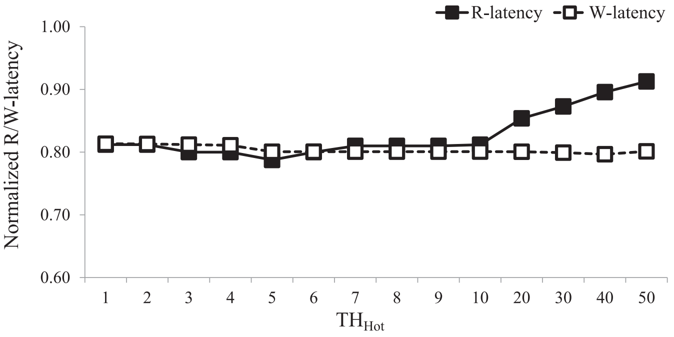


Fig. 19. Average latency by varying TH_{Hot}.

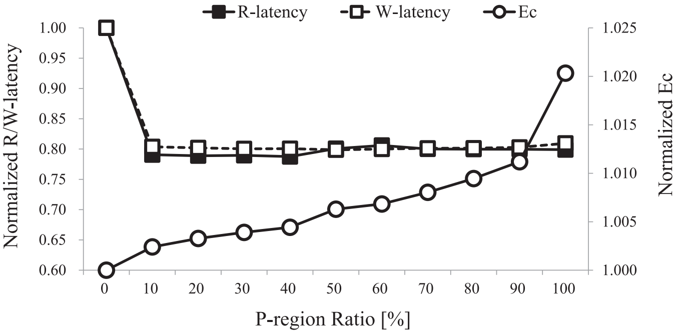


Fig. 21. Average latency and erase count by varying ratio of a preemptive region (□, ■: Latency, ○: Erase Count).

CFLRU+IRIT are 0.2% and 0.5% smaller than that of W-LRU, which is slightly different depending on the characteristics of the traces. CFLRU+IRIT performs more effectively than CFLRU in the read-intensive trace (Web2). It additionally shows the opposite result in the write-intensive trace (MSR1).

On the other hand, E_C of TS+IRIT is smaller than that of TS. The eviction priority of TS is decided by not only a clean page but also a CLOCK pointer and reference, in which a dirty page can be selected as a victim. Hence, securing clean pages utilizing IRIT can be more helpful to TS than to CFLRU.

4.4. Parameter sensitivity and optimization

In this section, we show how the optimal configuration of our method is obtained through an example of CFLRU+IRIT. In addition, we investigate the influence of the change of parameters on the performance, and what the optimal values of the parameters are when the cost and lifetime of the NFMs are taken into consideration. Furthermore, an experiment is performed using real traces, and performance factors are normalized to W-LRU.

4.4.1. TH_{Hot}

TH_{Hot}, the threshold of the HRDT, determines the amount of hot read data pre-loaded to the cache. A low TH_{Hot} value increases the sensitivity of HRDT for read data, therefore, a larger amount of read data regardless of its hotness can be loaded to the cache. A high TH_{Hot} results in only hot data being selected, but it reduces the amount of cached read data and produces no improvement in read performance.

In Fig. 19, if the value of TH_{Hot} is one, the overall trends show that smaller values of TH_{Hot} result in a shorter R-latency because a larger amount of read data is loaded into the cache and the probability of a read hit increases. However, a selection of excessively small TH_{Hot} may cause pollution of the cache by pre-loading data that are not sufficiently hot, as well as overhead by a large number of migration operations. Based on the data in Fig. 19, we choose the value of TH_{Hot} as five, which leads to the shortest average latency of all traces.

4.4.2. MBuf size

MBuf is a small buffer between the NFMs and the cache. It isolates their operating domains for efficient data migration. If a large MBuf is required to improve the performance, its overhead cost cannot be ignored because it is composed of SRAM in the NFM controller. However, as shown in Fig. 20, MBuf, with a size of only ten pages of NFM, causes the performance to plateau. This implies that MBuf is effectively utilized despite the small buffer size because the proposed method independently exploits the IRIT of each component.

4.4.3. Preemptive region size

The size of a preemptive region (P-region) (Section 3.5) is one of the most important parameters requiring optimization. Our method transforms dirty pages into clean pages until there are either no dirty pages in the region or until IRIT is exhausted. The process is closely related to both the performance and lifetime of NFMs; the relationship between them is depicted in Fig. 21.

The x-axis represents the size of the P-region in terms of the ratio to the total size of the cache, whereas the y-axes on the left and the right represent the average latency and E_C, respectively. The performance is improved even if only 10% of the cache is used as the P-region. Furthermore, if the portion of cache that is used is excessively large, the performance fluctuates because of massive eviction during IRIT, which incurs a lack of IRIT to execute other operations on the cache. Moreover, a larger P-region incurs additional E_C because of a greater number of page eviction operations, which can reduce the lifetime of the NFM.

As shown in Fig. 21, until the P-region is 40% occupied, R/W-latencies continuously decrease and E_C increases with a gentle slope. Once the occupancy exceeds 40%, the R-latency increases and E_C also increases with a steep slope. Thus, we select 40% as

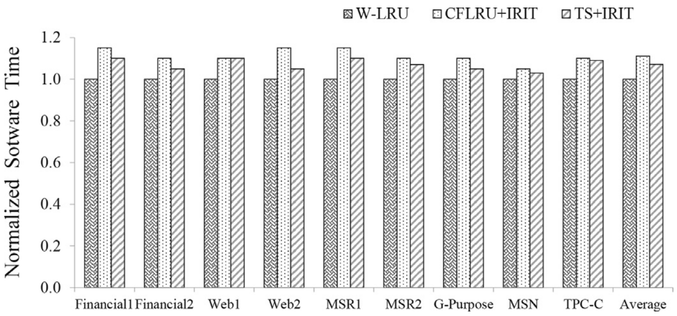


Fig. 22. Software execution time normalized to that of W-LRU.

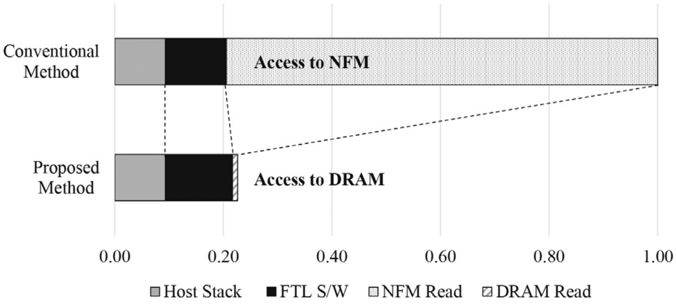


Fig. 23. Overall execution time normalized to that of the conventional method.

the P-region. This value reduces the lifetime by 0.4%; nevertheless, it helps improve performance by 21% on average and at most 43%.

4.5. Cost overhead

Considering practical system design, we analyzed both the hardware and software cost for our implementation. In the proposed NFSD, a migration buffer (MBuf) and host read data table (HRDT) were added on the hardware-side, and an IRIT calculator and migration manager were added on the software-side.

4.5.1. Hardware cost

In Section 4.4, we describe an experiment we conducted to determine the parameter sensitivity and size optimization, for which the size of MBuf (S_{MBuf}) and HRDT (S_{HRDT}) were selected as 10 and 1000, respectively. When the memory capacity required for storing one page of data and one page address is 4 KB and 4B, respectively, the hardware cost overhead can be computed as follows :

$$S_{MBuf} = 10\,pages = 10 \times (4\text{ KB} + 4B) \approx 40\text{ KB} \tag{2}$$

$$S_{HRDT} = 1000\,tables = 1000 \times (4B) \approx 4\text{ KB} \tag{3}$$

$$H/W Overhead = S_{MBuf} + S_{HRDT} = 44\text{ KB} \tag{4}$$

where MBuf needs to store both the data and address because of the need to transfer the exact data during IRIT from NFMs to the cache or vice versa; however, HRDT does not need to store data only because of checking the hot read address. Considering the high cost of SRAM, the cost overhead of 44 KB cannot be ignored. However, most NFSDs already adopt 1MB SRAM for data management and 128 KB SRAM called TCM (Tightly Coupled Memory) for performance improvement in NFM controllers and the system bus. Therefore, we think that our additional hardware cost is affordable for a performance improvement of up to 40% by the proposed techniques.

4.5.2. Software cost

Fig. 22 shows the software execution time of the proposed methods and W-LRU, normalized to that of W-LRU. The measured values represent the total software execution time of the flash translation layer (FTL) after handling the host command. Quantitatively speaking, the proposed methods require a 10% longer software execution time on average than W-LRU.

The increase is caused by two factors, the first of which is the time added for performing the IRIT calculation. However, the process of the IRIT calculation is very simple and does not require much software execution time owing to using the command list which has already been generated by the NFM controller. The second factor is the time added to search for the migration candidate. The searching process requires us to determine which migration candidate is most suitable; thus, the process checks whether a candidate for migration exists in each component and the candidate causes time overhead in the order of MBuf, DRAM cache, and NFMs. The process runs iteratively until IRIT is exhausted or no further migration candidate exists. In this process, there is no choice to spend additional software execution time, which is one of the most important processes in our proposed method.

However, the increased values are not reflected in direct proportion to the overall execution time. Fig. 23 shows the overall execution time of an NFSD when a read request is required from the host. The overall execution time includes the host command process time, read time of NFM or DRAM, and aforementioned FTL software time. We assume that the conventional method accesses NFM because the read data is not stored in DRAM and the proposed method accesses DRAM because in this case hot read data is stored in DRAM.

In the conventional method, the overall execution time is dominated by NFM read time (the ratio is the same as that of a commercial SSD); therefore, a slight increase in software execution time does not greatly affect the overall performance of the NFSD. On the other hand, the proposed method dramatically reduces the overall execution time as a result of transferring hot read data from NFM to the DRAM cache, despite the increased software time of approximately 10%. The validity of this analysis was well proved by the result of read performance improvement as shown in Fig. 15.

4.6. Energy consumption

In a general NFSD, the system power is dominated by memory devices such as NFM and DRAM rather than by the controller, and the power of the memory devices is related to the number of read/write operations and the execution speed. In this section, we prove the effectiveness of the proposed techniques in terms of the energy consumption by measuring the operation count of NFMs and DRAM cache in each NFSD and analyze their energy consumption.

This comparison excludes the three methods (RW-LRU, CFLRU, TS), which incur too many operations of NFMs and the DRAM cache due to inefficient read-allocation and eviction overhead, and even have a much longer read latency than that of W-LRU as shown in Fig. 15. In this analysis, therefore, we compare W-LRU as a baseline and only two methods (CFLRU+IRIT, TS+IRIT) using our proposed techniques. The performance of these two methods is outstanding compared to that of W-LRU.

As shown in Fig. 24, our proposed methods incur a larger number of read/write operation counts compared to W-LRU due to additional read data movement from NFMs to the DRAM cache, by a maximum of 60% and 8% on average, respectively. However, despite the concern over power, our methods consume less energy than W-LRU in Fig. 25, which is attributable to the shorter latency than that of W-LRU.

In our methods, the read operations finish earlier than those of W-LRU due to frequent DRAM data access cached by the effi-

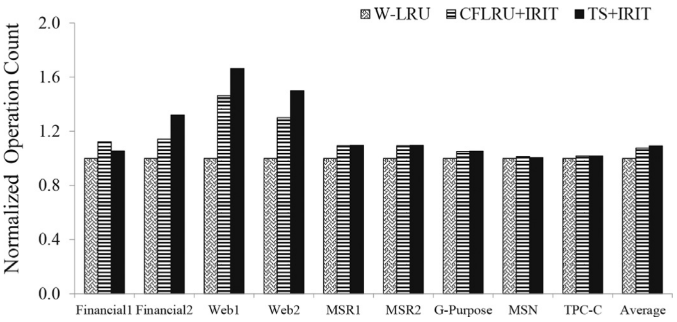


Fig. 24. Operation counts of NFMs and the cache in each NFSD (NFM read/program/erase, Cache read/write).

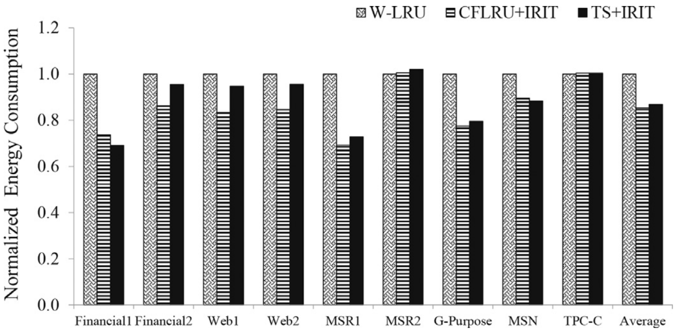


Fig. 25. Energy consumption normalized to that of W-LRU.

cient hot read data movement (in Fig. 15), and the write operations finish earlier due to the hidden eviction time overhead as a result of using IRIT (in Fig. 14). Thus, our method is characterized by shorter periods of power consumption, and the overall system energy consumption becomes smaller despite the higher operation count. Quantitatively speaking, the proposed methods consume 14% less energy. In summary, the effectiveness of our techniques is proved in terms of energy consumption as well as performance.

5. Conclusion

In this paper, we proposed a new data management method that utilizes a new idle time scheme. The scheme reduces eviction overhead and is helpful for saturated read/write performance. Our novel method exploits unutilized idle time (*i.e.*, intra-request idle time). Further, we proposed a data management scheme that utilizes idle time (*i.e.*, pre-store/pre-load). The experimental results showed that the proposed approach reduces read latency by up to 43% for read-intensive applications and write latency by up to 40% in write-intensive applications, and reduces read latency and write latency by an average of 21% and 20%, respectively, compared to NFSD with a write cache buffer. In particular, read latency is reduced by 56% compared to CFLRU which performs read-allocation. In the future, our migration scheme utilizing IRIT could be extended to improve other important factors of NFSDs, such as SLC and MLC management of NFMs and wear-leveling management among several NFMs.

Acknowledgments

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2016R1A2B4011799), by the ICT R&D program of MSIP/IITP[2016(R7177-16-0233), Developement of Application Program Optimization Tools for High Performance Computing Systems], by Ministry of Science, ICT & Future Planning [No.2016M3A7B4911115], by NSF Grant XPS-1439165, and by Samsung Electronics Company, Ltd., Hwasung, Korea.

References

[1] Samsung Electronics, 1955. http://www.samsung.com/semiconductor/products/flash-storage/.
[2] R. Micheloni, A. Marelli, K. Eshghi, Inside Solid State Drives (SSDs), vol. 37, Springer Science & Business Media, 2012.
[3] G. Lawton, Improved flash memory grows in popularity, Computer 39 (1) (2006) 16–18, doi:10.1109/MC.2006.22.
[4] B. Jacob, S. Ng, D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann, 2010.
[5] Anandtech Inc., 1997. http://www.anandtech.com/bench/product/533?vs=1252/.
[6] O. Kwon, H. Bahn, K. Koh, FARS: a page replacement algorithm for nand flash memory based embedded systems, Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology (CIT'08), IEEE, 2008, pp. 218–223, doi:10.1109/CIT.2008.4594677.
[7] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, J. Lee, CFLRU: a replacement algorithm for flash memory, in: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, ACM, 2006, pp. 234–241, doi:10.1145/1176760.1176789.
[8] Y.-S. Yoo, H. Lee, Y. Ryu, H. Bahn, Page replacement algorithms for NAND flash memory storages, in: Proceedings of the 2007 International Conference on Computational Science and its Applications (ICCSA'07), Springer, 2007, pp. 201–212.
[9] J. Park, H. Lee, S. Hyun, K. Koh, H. Bahn, A cost-aware page replacement algorithm for nand flash based mobile embedded systems, in: Proceedings of the 7th ACM International Conference on Embedded Software (EMSOFT '09), ACM, 2009, pp. 315–324, doi:10.1145/1629335.1629377.
[10] Z. Li, P. Jin, X. Su, K. Cui, L. Yue, CCF-LRU: a new buffer replacement algorithm for flash memory, IEEE Trans. Consum. Electron. 55 (3) (2009) 1351–1359, doi:10.1109/TCE.2009.5277999.
[11] H. Lee, H. Bahn, K.G. Shin, Page replacement for write references in NAND flash based virtual memory systems, J. Comput. Sci. Eng. 8 (3) (2014) 157–172.
[12] P. Jin, Y. Ou, T. Härder, Z. Li, AD-LRU: an efficient buffer replacement algorithm for flash-based databases, Data Knowl. Eng. 72 (2012) 83–102, doi:10.1016/j.datak.2011.09.007.
[13] D.H. Kang, C. Min, Y.I. Eom, TS-CLOCK: temporal and spatial locality aware buffer replacement algorithm for nand flash storages, in: Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems, ACM, 2014, pp. 581–582, doi:10.1145/2591971.2592028.
[14] H. Lee, H. Bahn, Characterizing virtual memory write references for efficient page replacement in NAND flash memory, in: IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'09), IEEE, 2009, pp. 1–10, doi:10.1109/MASCOT.2009.5366768.
[15] I. Corporation, Understanding the Flash Translation Layer (FTL) Specification, 1998.
[16] A. Gupta, Y. Kim, B. Urgaonkar, DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-Level Address Mappings, vol. 44, ACM, 2009.
[17] S.-W. Lee, W.-K. Choi, D.-J. Park, FAST: an efficient flash translation layer for flash memory, in: Emerging Directions in Embedded and Ubiquitous Computing, Springer, 2006, pp. 879–887.
[18] J. Lee, Y. Kim, G.M. Shipman, S. Oral, F. Wang, J. Kim, A semi-preemptive garbage collector for solid state drives, in: 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2011, pp. 12–21, doi:10.1109/ISPASS.2011.5762711.
[19] Y. Kim, J. Lee, S. Oral, D. Dillow, F. Wang, G.M. Shipman, et al., Coordinating garbage collection for arrays of solid-state drives, IEEE Trans. Comput. 63 (4) (2014) 888–901, doi:10.1109/TC.2012.256.
[20] M. Jung, R. Prabhakar, M.T. Kandemir, Taking garbage collection overheads off the critical path in SSDs, in: Middleware 2012, Springer, 2012, pp. 164–186.
[21] J. Lee, Y. Kim, G.M. Shipman, S. Oral, J. Kim, Preemptible i/o scheduling of garbage collection for solid state drives, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 32 (2) (2013) 247–260, doi:10.1109/TCAD.2012.2227479.
[22] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, T.-W. Kuo, Garbage collection and wear leveling for flash memory: past and future, Proceedings of 2014 International Conference on Smart Computing (SMARTCOMP), IEEE, 2014, pp. 66–73, doi:10.1109/SMARTCOMP.2014.7043841.
[23] S. Lee, J. Kim, in: Improving Performance and Capacity of Flash Storage Devices by Exploiting Heterogeneity of MLC Flash Memory, vol. 63, IEEE, 2014, pp. 2445–2458, doi:10.1109/TC.2013.120.
[24] S.-H. Park, D.-G. Kim, K. Bang, H.-J. Lee, S. Yoo, E.-Y. Chung, An adaptive idle-time exploiting method for low latency NAND flash-based storage devices, IEEE Trans. Comput. 63 (5) (2014) 1085–1096, doi:10.1109/TC.2012.281.
[25] N. Mi, A. Riska, Q. Zhang, E. Smirni, E. Riedel, Efficient management of idleness in storage systems, ACM Trans. Storage 5 (2) (2009a) 4, doi:10.1145/1534912.1534913.
[26] N. Mi, A. Riska, X. Li, E. Smirni, E. Riedel, Restrained utilization of idleness for transparent scheduling of background tasks, in: Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance'09), vol. 37, ACM, 2009b, pp. 205–216, doi:10.1145/1555349.1555373.

[27] J. Hutchby, M. Garner, Assessment of the potential & maturity of selected emerging research memory technologies, in: Workshop & ERD/ERM Working Group Meeting, 2010. http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2010ITRS/2010Update/ToPost/ERD_ERM_2010FINALReportMemoryAssessment_ITRS.pdf.

[28] S.-H. Park, S.-H. Ha, K. Bang, E.-Y. Chung, Design and analysis of flash translation layers for multi-channel NAND flash-based storage devices, IEEE Trans. Consum. Electron. 55 (3) (2009) 1392–1400, doi:10.1109/TCE.2009.5278005.

[29] J.-Y. Kim, S.-H. Park, H. Seo, K.-W. Song, S. Yoon, E.-Y. Chung, Nand flash memory with multiple page sizes for high-performance storage devices, IEEE Transactions on very Large Scale Integration (VLSI) Systems 24 (2) (2016), doi:10.1109/TVLSI.2015.2409055.

[30] H.-J. Kim, S.-G. Lee, A new flash memory management for flash storage system, in: Proceedings of the Twenty-Third Annual International Computer Software and Applications Conference (COMPSAC'99), IEEE, 1999, pp. 284–289.

[31] J. Standard, Low power double data rate 2 (LPDDR2), JESD209-2E, April 8 (2011).

[32] Micron Technology, NAND Flash Memory Datasheet, MT29F64G08CBAA, 0000.

[33] H. Semiconductor, et al., Open NAND Flash Interface Specification, Technical Report, Technical Report Revision 1.0, 2006. http://www.onfi.org/.

[34] UMass Trace Repository, 0000 http://traces.cs.umass.edu/.

[35] Storage Networking Industry Association, 2011. http://iotta.snia.org.

[36] DiskMon for Windows v2.01, 2010. http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx/.

[37] Supplement Data, 2015. http://dtl.yonsei.ac.kr/down/PreStorePreLoad.pdf.

Jin-Young Kim received the B.S. and M.S. degrees in semiconductor science from Dongguk University, Seoul, Korea, in 2003 and 2005, respectively, and the Ph.D. degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2017. In 2005, she joined Samsung Electronics Co., where she was involved in circuit design for next-generation DRAM, 1-transistor DRAM, 4F2 DRAM, PRAM and NAND flash Memory. Her current interests include system architecture and VLSI design using flash memory and emerging memories.

Tae-Hee You received the B.S. degree in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2009. He is currently a Ph.D. candidate in Yonsei University. His research interests include high performance system architecture and VLSI design with the special emphasis on NVM memory applications.

Hyeokjun Seo received the B.S. degree in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2008. He is currently a Ph.D. candidate in Yonsei University. His research interests include system architecture and VLSI design based on flash memory applications.

Sungroh Yoon received the B.S. degree in electrical engineering from Seoul National University, Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, CA, in 2002 and 2006, respectively. From 2006 to 2007, he was with Intel Corporation, Santa Clara, CA. Previously, he held research positions with Stanford University, CA, and Synopsys, Inc., Mountain View, CA. Dr. Yoon was an assistant professor with the School of Electrical Engineering, Korea University from 2007 to 2012. Currently, he is an assistant professor with the Department of Electrical and Computer Engineering and also with the Inter-disciplinary Program in Bioinformatics, Seoul National University, Korea. His research interests include biomedical computation and embedded systems. He is a senior member of the IEEE.

Jean-Luc Gaudiot received the Diplôme d'Ingénieur from the École Supérieure d'Ingénieurs en Electrotechnique et Electronique, Paris, France, in 1976, and the MS and PhD degrees in computer science from the University of California, Los Angeles, in 1977 and 1982, respectively. He is currently a professor and chair of the Electrical and Computer Engineering Department at the University of California, Irvine. Prior to joining UCI in January 2002, he was a professor of Electrical Engineering at the University of Southern California since 1982, where he served as and the director of the Computer Engineering Division for three years. He has also done microprocessor systems design at Teledyne Controls, Santa Monica, California (1979-1980) and research in innovative architectures at the TRW Technology Research Center, El Segundo, California (1980–1982). He consults for a number of companies involved in the design of high- performance computer architectures. His research interests include multithreaded architectures, fault-tolerant multiprocessors, and implementation of reconfigurable architectures. He has published more than 170 journal and conference papers. His research has been sponsored by US National Science Foundation (NSF), US Department of Energy (DOE), and US Defense Advanced Research Projects Agency (DARPA), as well as a number of industrial organizations. In January 2006, he became the first editor-in-chief of IEEE Computer Architecture Letters, a new publication of the IEEE Computer Society, which he helped found to the end of facilitating short, fast turnaround of fundamental ideas in the computer architecture domain. From 1999 to 2002, he was the editor-in- chief of the IEEE Transactions on Computers. In June 2001, he was elected chair of the IEEE Technical Committee on Computer Architecture, and reelected in June 2003 for a second two-year term. He has also chaired the IFIP Working Group 10.3 (Concurrent Systems). He is one of three founders of PACT, the ACM/IEEE/IFIP Conference on Parallel Architectures and Compilation Techniques, and served as its first program chair in 1993, and again in 1995. He has also served as a program chair of the 1993 Symposium on Parallel and Distributed Processing, HPCA-5 (1999 High Performance Computer Architecture), the 16th Symposium on Computer Architecture and High Performance Computing (Foz do Iguaçu, Brazil), the 2004 ACM International Conference on Computing Frontiers, and the 2005 International Parallel and Distributed Processing Symposium. He was elevated to the rank of AAAS Fellow in 2007. He is a member of the ACM and ACM SIGARCH. In 1999, he became a fellow of the IEEE.

Eui-Young Chung received the B.S. and the M.S. degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2002. From 1990 to 2005, he was a Principal Engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. He is currently a professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul, Korea. His research interests include system architecture, bio-computing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications and flash memory applications.