Amnesic Cache Management for Non-Volatile Memory

Dongwoo Kang*, Seungjae Baek*, Jongmoo Choi*, Donghee Lee†, Sam H. Noh ‡ and Onur Mutlu §
*Dept. of Software, Dankook University, South Korea
Email: {kangdw, baeksj, choijm}@dankook.ac.kr
†School of Computer Science, University of Seoul, South Korea
Email: dhl_express@uos.ac.kr
‡School of Comp. & Info. Eng., Hongik University, South Korea
Email: samhnoh@hongik.ac.kr
§Dept. of Electrical and Computer Engineering, Carnegie Mellon University, USA
Email: onur@cmu.edu

***Abstract*—One characteristic of non-volatile memory (NVM) is that, even though it supports non-volatility, its retention capability is limited. To handle this issue, previous studies have focused on refreshing or advanced error correction code (ECC). In this paper, we take a different approach that makes use of the limited retention capability to our advantage. Specifically, we employ NVM as a file cache and devise a new scheme called *amnesic* cache management (ACM). The scheme is motivated by our observation that most data in a cache are evicted within a short time period after they have been entered into the cache, implying that they can be written with the relaxed retention capability. This retention relaxation can enhance the overall cache performance in terms of latency and energy since the data retention capability is proportional to the write latency. In addition, to prevent the retention relaxation from degrading the hit ratio, we estimate the future reference intervals based on the inter-reference gap (IRG) model and manage data adaptively. Experimental results with real-world workloads show that our scheme can reduce write latency by up to 40% (30% on average) and save energy consumption by up to 49% (37% on average) compared with the conventional LRU based cache management scheme.**

I. Introduction

The emergence of NVM technologies such as phase-change memory (PCM) and spin transfer torque RAM (STT-RAM) that provides both byte-addressability and non-volatility are bringing about new opportunities in designing the memory hierarchy [1], [2]. One characteristic of NVM, however, is that even though NVM supports non-volatility, non-volatility is sustained for only a certain time period, which we refer to as the retention capability of the device. For instance, PCM represents different data states using different resistances, and the retention capability of PCM becomes limited due to a phenomenon known as the *resistance drift* [3], [4], which results in the states (or target bands, in PCM jargon) representing bits to collide with neighboring states. Hence, if the resistance drift is left unattended, data is eventually lost. Such limits on retention capabilities are also observed in STT-RAM due to the *thermal stability* of the magnetic tunnel junction (MTJ) [5] and in NAND flash memory due to the *charge loss* in the floating gate [6]. Another characteristic related to limited retention capability is its relation with write speed and control of this retention capability. Specifically, the write latency to an NVM is proportional to the retention capability of NVM. That is, write latency increases with longer retention capability and vice versa. Take the PCM example once again. As resistance drifts are the reason for data loss, one way to mitigate this loss is to allocate larger margins between states so that it becomes more robust to the resistance drift. To do so, however, requires narrowing the width of a state (the target band), which in turn requires more iterations to write a cell as this requires finer control, eventually making writes longer. In contrast, wider target bands makes data vulnerable to retention error, but has a positive effect of shortening the write speed. Specifically, we can improve the write speed by 1.7x by reducing the retention capability from 10^7 to 10^4 seconds [7].

Again, the same tradeoff is also observed in STT-RAM where high thermal stability makes the cell more tolerable to random bit-flips, while making it more difficult to write [5]. Similarly, in NAND flash, retention capability can be enforced at the cost of more fine-grained control upon writing and more complex error correction code (ECC) [6]. This tradeoff sets new challenges to system architects in various aspects of performance, reliability, and energy consumption.

In this paper, we exploit the tradeoff between retention capability and write latency to design a novel cache management scheme where NVM is used as a file cache. Whereas traditional cache management schemes focus on selecting victim cache blocks through replacement policies, we introduce the amnesic notion, that is, ability to forget, into the cache management scheme. While replacement based management can be regarded as 1) writing data with unlimited retention time and 2) evicting data that are predicted as not being re-referenced in the near future, our amnesic approach can be viewed as 1) predicting the interval to be re-referenced in the future and writing data with the corresponding retention capability and 2) forgetting them if they are not re-referenced within the interval, thereby making space for new data to be moved in.

A key metric in traditional cache performance analysis is the hit ratio. The basic assumption behind traditional cache studies is that write latency is constant. This study differs in that we propose to enhance cache performance by reducing the

978-1-4673-7619-8/15/$31.00

write latency using the amnestic notion, even though it may hurt the hit ratio. However, even in terms of the hit ratio, it is possible that the amnesic notion can provide benefits. The key issue in enhancing the hit ratio is appropriately forgetting the most unwanted block, the victim block, so that space can be made for the new block to come in. To forget at appropriate points in time, we make use of the inter-reference gap (IRG) model [8].

We argue that, the amnesic approach is a more viable approach for devices that have limited retention capability such as NVM, and we show through experiments that this approach brings about performance and energy benefits. Experimental results show that our approach improves write performance by 30% on average and saves energy consumption by 37% on average compared with the conventional LRU based cache management scheme.

The rest of this paper is organized as follows. In Section II, we present the background information regarding this study including NVM usages and characteristics. Then, we explain the structure of an NVM cache considered in this study and discuss cache behaviours in Section III. Our proposal and evaluation results are described in detail in Section IV and V, respectively. Section VI surveys related work. Finally, we summarize and conclude with future work in Section VII.

II. Background

In this section, we first explore how NVM affects the memory hierarchy in modern computer systems. Then, we discuss the notion of limited retention capability and its relation with write latency in detail.

A. NVM in memory hierarchy

Emerging NVM technologies such as PCM [3], STT-RAM [9], RRAM(Resistive RAM) [10] and NV-DIMM (Non-volatile Dual In-line Memory Module) [11] provides various features in terms of interfaces, density, performance, durability, reliability and power-savings. These features allow system architects to employ NVM usefully at various levels of the memory hierarchy.

One feature of NVM is byte-addressability. Also, NVM is more superior to DRAM with respect to scalability and density enabling NVM to be utilized in large-scale main memory systems [12]–[15]. For instance, 20-nm PCM prototypes have already been demonstrated and 8-nm PCM is projected to be available soon [7]. However, PCM suffers from high latency, especially write latency, compared to DRAM. Hence, several studies have proposed the use of hybrid memory consisting of NVM and DRAM to reap the merits of both DRAM's fast latency and NVM's scalability at the same time [16]–[18].

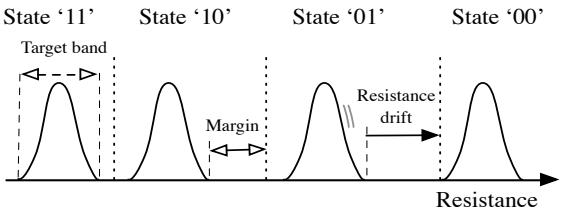
Another feature of NVM is non-volatility. NVM shows better performance compared to traditional storage media such as NAND flash memory and disks. Hence, it is a favorable choice for high-performance storage systems [19]–[23]. In addition, data can be kept permanently, while being accessed in the same manner as data structures in main memory. This allows NVM to be utilized as an efficient persistent store [24]–[27].

When we go one step further, we can build a new type of unified memory that can be used for both main memory and storage at the same time [2], [7], [28]–[31]. Such integration allows features such as 1) accessing files through the load/store instructions instead of the heavy block-oriented interfaces, 2) moving data between main memory and storage without actual copying, 3) whole system persistency, and 4) instant execution and booting.

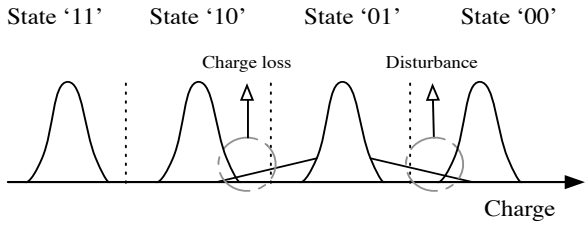
NVM can also be utilized as a cache. Employing NVM as a CPU cache allows us to achieve the density and power-saving advantages [32]–[34]. When we utilize it as a file cache, we can accelerate not only performance but also durability due to the non-volatility of NVM [35]–[38]. Just as flash memory is actively being adopted as a cache in today's storage systems [39], [40], NVM technologies are expected to gain more attention as a cost-effective cache candidate in the near future [41].

B. Tradeoff between retention capability and write speed

All memory types can be spread along a spectrum in regards to retention capability starting from the least retentive to the most. At one extreme, there is the DRAM whose retention time is small, needing, in general, to be refreshed every 64ms. At the other extreme, there is the hard disk whose retention time is larger than 10^{15} seconds, which, in practical terms, can be seen as having infinite retention. NVM and flash memory sit between the two extremes having some limited retention capabilities, which can be controlled along a particular range through the write process.



(a) Data loss due to resistance drift in PCM



(b) Data loss due to charge loss and disturbance in NAND

Fig. 1: States in 2-bit MLC PCM and NAND

An important characteristic related to retention capability is its relation to write latency. Let us elaborate on this relation using Figure 1(a) and (b) that shows the states of 2-bit MLC PCM and NAND, respectively. PCM utilizes the trait of chalcogenide glass, which can be in either the amorphous or the crystalline phase [3]. There are a range of resistances between the two phases. By dividing this range accordingly, PCM represents two states for SLC and four states for MLC.

The mechanism is similar to the NAND case, where states can be differentiated according to the number of charges in the floating gate [6].

Each state in a PCM cell has a target band, a region of resistances that corresponds to valid bits. The resistance in a PCM cell has a tendency to increase with time, and this is known as the *resistance drift*. Hence, when the resistance drifts up to the boundary of the next region, the state can be incorrectly represented leading to data loss [4]. A similar data loss phenomenon is also observed in NAND where retention errors occur due to charge loss over time and read/write disturbances [6].

To alleviate this problem, PCM allots a margin between target bands. Wider margins makes it more tolerant to the resistance drift. However, this also makes the width of the target bands narrower. To write to PCM, an iterative mechanism that alters the resistance of a cell by $\triangle R$ at each iteration is employed. Hence, narrowing target bands requires more precise control over the iterative mechanism. Ultimately, this demands smaller $\triangle R$ resulting in a slowdown of the write latency.

Such tradeoff between the retention capability and write speed, that is, higher retention increasing write latency and vice versa, has been observed and exploited in a previous study. Specifically, Liu et al. uses a model to demonstrate that 1.7x write speedup can be obtained by reducing the retention capability of PCM from 10^7 to 10^4 [7]. They also uncover several quantitative data related to this tradeoff.

NAND flash also exhibits this tradeoff [42]. Writes in NAND flash make use of the incremental step pulse programming (ISPP) mechanism. The mechanism increases the threshold voltage (V_{th}) of a NAND cell step-by-step with a certain voltage increment ($\triangle V_{th}$). The amount of voltage increment in each step affects the write latency and retention time. Specifically, larger increments makes writing faster since less steps are required during ISPP. In contrast, larger increments reduces retention time since it widens the threshold voltage distribution minimizing the margin for tolerating retention errors. Note that STT-RAM also shows a similar tradeoff [5].

In this study, we focus on PCM since PCM is a mature technology and its adoption to real systems are imminent [41]. Also, the availability of quantitative data about the tradeoff in PCM provided by Liu et al. [7] allows us to evaluate the schemes that we propose from diverse viewpoints. We emphasize that our proposal can be applied not only to PCM, but also to NAND and STT-RAM.

III. Cache Architecture and Motivation

In this section, we first describe the NVM cache architecture considered in this paper. Then, we discuss several cache behaviors such as the mean caching time and distribution of intervals between consecutive references that serve as the motivation behind this study.

A. NVM cache architecture

Figure 2 illustrates a conceptual structure of a system equipped with an NVM cache. It consists of three layers: application, NVM cache, and storage. The NVM cache can

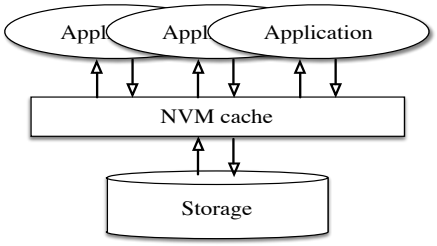


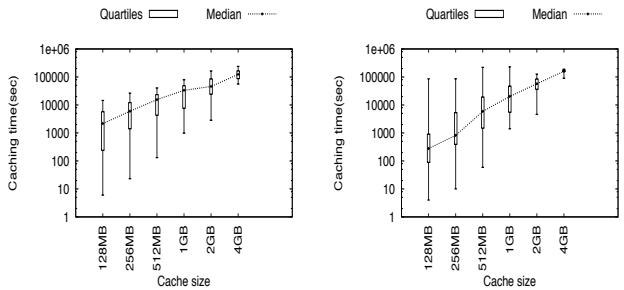
Fig. 2: NVM cache architecture.

be materialized in various forms such as a buffer cache in host systems [35], [36], [38], a server-side cache [41], or a cache within storage systems [43].

Employing an NVM cache provides performance improvements by handling requests on the spot instead of fetching/destaging data from/to a storage system. The hit ratio is an important measure for caches, and the replacement policy plays a key role in the resulting hit ratio. The LRU (Least Recently Used) policy is commonly used since it tends to keep data with temporal locality in the cache. A delayed write mechanism that writes dirty data back to the storage periodically or when they are replaced can be integrated with LRU to enhance performance further. One concern of this mechanism is that some written data could be lost if a sudden power failure occurs. However, the non-volatile nature of NVM relieves this concern and allows this mechanism to be applied more aggressively. We use the LRU policy with delayed write, which writes data back to storage when they are replaced or just before the data is lost due to the limited retention capability, as the baseline configuration for the NVM cache.

B. Cache behavior analysis

Our quest is to make use of the various levels of retention capability. Hence, the first thing we want to observe is how much retention capability an NVM cache requires. For this, we analyze real-world trace which is the MSR Cambridge [44] under the baseline configuration and measure the *caching time* of a cache unit, 4KB block, as shown in Figure 3. The trace has I/O information during 7 days. Caching time, here, is defined as the total time a block kept in the cache, specifically, the time difference between the eviction time and the first reference time.



(a) hm_0 workload (b) $proj_3$ workload

Fig. 3: Mean caching time

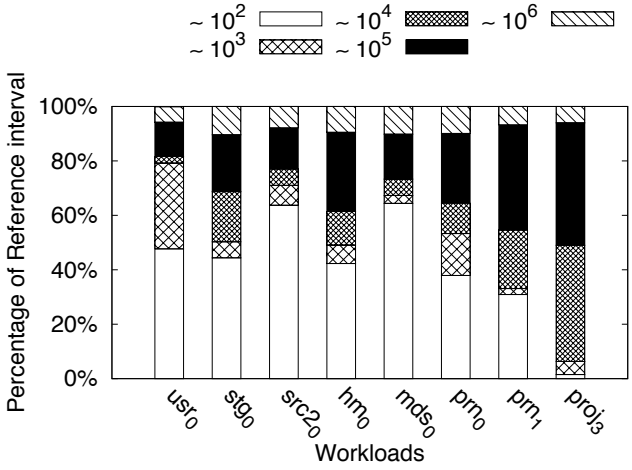


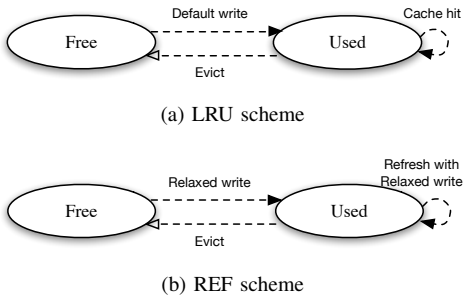
Fig. 4: Proportion of reference intervals

From Figure 3, where we show results for only two of the workloads (working set of each workload is 7.8GB and 8.5GB, repectively), we find that most data are evicted within a limited time period after they enter the cache when cache size is less than each working set. For instance, when the cache size is 1 GB (roughly 12% of the working set), the mean caching time is around 10^4 seconds, while it is less than 10^5 seconds with the cache size of 4 GB. Note that the whole tracing duration of each workload is 7 days, which is 6×10^5 seconds. Similar trend is also observed in other workloads in the MSR Cambridge trace.

In general, system architects have preferred NVM with large retention capabilities to achieve better non-volatility. Also, JEDEC recommends the retention capability to be set to 10^7 when we design a system that makes use of the durability of NVM [45]. Therefore, the default retention capability of the write operation in PCM is normally set to 10^7 when PCM is utilized as storage [7].

However, our observation shows that a large portion of data in a cache is evicted before that 10^7 seconds recommended by JEDEC when the case is not unlimited as general system environment cases. This implies that we can improve the write performance by simply applying retention relaxation, that is, writing data with less retention capability. Retention relaxation is even more appealing in a cache as in a cache there are no worries about reliability or data loss as data are backed up in storage. Note that most traditional cache management schemes guarantee the inclusion property, that is, data in cache are also maintained in storage.

One concern of retention relaxation is that it may deteriorate the hit ratio by missing data that are re-referenced after the relaxed retention time. To analyze and quantify this effect, we divide reference intervals into 5 regions as shown in Figure 4 and measure what percentage of references are re-referenced (whether reads or writes) within each interval. This figure shows that, even though roughly 90% of data are re-referenced within the 10^5 second interval, a non-negligible amount of data are also being re-accessed after that time interval. Indeed, retention relaxation can be a double-edged sword. It can enhance write performance by relaxing the retention capability



(a) LRU scheme

(b) REF scheme

Fig. 5: State diagram for LRU and REF schemes

for data repeatedly written in short intervals or for data that are evicted without being re-referenced. However, when data is re-referenced after its retention capability, it will induce a miss, reducing the hit ratio and triggering extra accesses to retrieve the data from storage. To make use of retention relaxation efficiently, we need to differentiate data according to their access intervals and decide how to deal with them such that our goal is met.

IV. Amnesic Cache Management

In this section, we discuss how to overcome the hit ratio reduction while obtaining performance gains from retention relaxation. We first discuss a naive refreshing based scheme. Based on faults found with the naive approach, we present two Amnesic Cache Management (ACM) schemes that we propose that make use of the fact that NVM loses (or forgets, hence amnesic) data after some time.

A. Refresh based cache management

When retention is relaxed, the hit ratio may suffer as data that was cached may become invalid as the retention period expires. One feasible way to resolve this problem is to refresh the cached data. That is, the data cache can be read from and then written back to the cache to replenish the retention capability just before the retention period expires.

Figure 5 shows the state diagram for the traditional LRU-based cache management scheme (LRU) and the refresh-based cache management scheme (REF). In LRU, when a new write request occurs, first, space that is in a free state, if available, is transitioned to the used state. Then, data is written (either fetched from storage or issued by an application) into the allocated space, where writing is done with the default retention capability that maximizes the retention time. In this study, we assume this is 10^7 seconds as is done by Liu et al. [7]. When there is no free space, the space occupied by the LRU block is reclaimed to serve the new request.

The REF scheme works similarly to the LRU scheme, except that it writes data with the relaxed retention capability, such as 10^4 or 10^5. Also, it performs refreshing for data whose retention time is about to expire. This REF scheme can enhance write speed through retention relaxation. For instance, by relaxing retention period from 10^7 to 10^4, we can enhance the write latency by 1.7 times [7]. Also, through refreshing, the same hit ratio as LRU can be maintained.

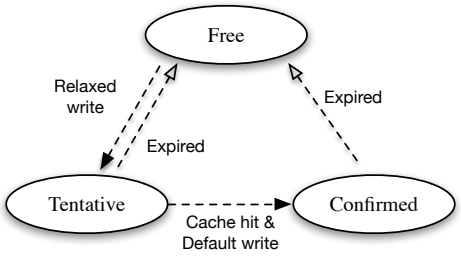


Fig. 6: State diagram for SACM

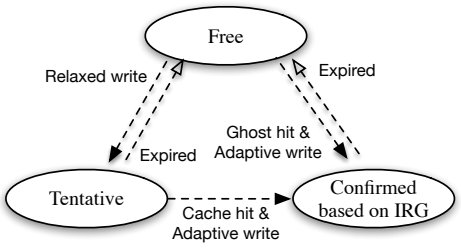


Fig. 7: State diagram for AACM

However, REF raises several concerns. One is the performance degradation due to refreshing though techniques such refreshing in the background could be employed to partially or fully alleviate this degradation. The second concern is the energy consumption due to periodic refreshing. The final concern is the endurance issue. Refreshing increases the actual number of writes to PCM, which eventually leads to shortened PCM lifetime.

Several studies on how to mitigate the refresh overhead such as smart refresh, adaptive refresh and retention-capability aware refresh have been proposed [4], [6], [7], [46]. In this study, we take a completely different approach and propose an amnesic approach, that is, an approach that forgets the contents of the cache for better performance and energy usage. In the following, we propose two versions of the amnesic approach.

B. Simple Amnesic Cache Management

Figure 6 shows the state diagram of the Simple Amnesic Cache Management (SACM) scheme. There are three states in SACM; *free*, *tentative* and *confirmed*. Upon its initial write into the cache, the datum is written with the relaxed write (with retention 10^4 seconds in this study)and is set to the Tentative State (TS). Then, if it is referenced again (read or write) within the retention time, its state is transitioned from TS to the Confirmed State (CS) and is rewritten with the default write (with retention 10^7 seconds in this study). However, if it is not referenced again and the retention time (10^4 seconds) expires, SACM simply forgets the data, and the state is transitioned from TS to the Free State (FS). Data in CS that expires are also forgotten. Note that our scheme satisfies the inclusion property by writing back dirty data into storage just before the retention time expires. Hence, there is no loss of data. Note that, to guarantee the durability, we can employ the write-back flush or write-back persist proposed by Qin et al. [50].

In SACM, the time spent in TS can be considered to be a monitoring period where the value of the data is weighed. If it is not referenced again within the retention time, the data evaporates making new room in the cache. If it is re-referenced, it is considered worthy and moved to CS. This is an important step in SACM. If the monitoring period is too short this will lead to misses even though data may exhibit temporal locality. On the other hand, if it is too large, SACM may waste cache space while maintaining valueless data.

SACM comes with several merits. First, it can enhance write latency by applying retention relaxation for data that are not re-referenced in the cache. Second, by introducing the state CS, it provides enough time for the re-referenced data to be kept in the cache. Finally, it is practical in the sense that it requires only minor hardware modifications to support the two write modes, the relaxed mode and the default mode. Several hardware-level techniques for this supporting have been demonstrated in [7], [47]–[49].

However, there are a couple of issues with SACM. First, the transition from TS to CS causes additional writes. For write requests, they are inevitable, so they are not an issue, which would also be true for the LRU scheme. However, for read requests, the additional writes are all extras that may worsen the endurance of NVM. Even so, we observe that these writes affect little on endurance, which will be discussed later in Section V.

The other issue concerns the default write when transitioning to CS. The question is whether this is the right choice. In Figure 4 we observed that a considerable amount of data are being re-referenced at intervals much shorter than 10^5. This observation leads us to go one step further and design an adaptive scheme, which we discuss in the next section.

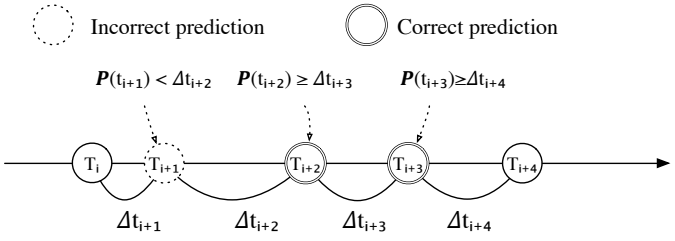
C. Adaptive Amnesic Cache Management

The second scheme that we propose is the Adaptive Amnesic Cache Management (AACM) scheme. Figure 7 shows the state diagram of AACM. AACM has the same three states as in SACM, but with two differences. The first difference is that when transitioning from TS to CS, the write used is now an adaptive write, which we discuss in more detail later, instead of the default write. The other difference is that we introduce a new transition from FS to CS based on the use of a ghost buffer. We elaborate on this further below.

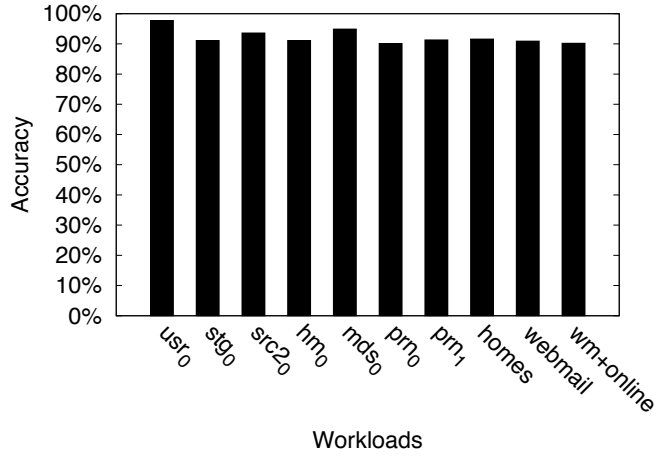
The key idea of AACM is that it estimates the next reference of each data and writes it with the appropriate retention capability adaptively. To estimate the next reference, we make use of the inter-reference gap (IRG) model [8] that has shown that future IRGs can be predicted from past IRGs. IRG is defined as an interval between two consecutive references of a data block.

In this study, we use the first order Markov chain. Specifically, when a data block is re-referenced, we measure the interval between the previous and current references. Then, we assume that the next interval will be the same as the measured interval. Based on this estimation, we write that data with the appropriate retention capability.

Since all data blocks have different IRGs, AACM writes them adaptively with different capabilities, hence the term adaptive write. However, allowing each and every data block to have a different retention capability is not feasible as the NVM hardware will become too complex. Hence, in this study, we take a coarse grain adaptive write approach and divide the retention capability into the six levels where each level is separated by the five threshold shown in Figure 4. Then, the write retention capability of each block is set to the closest upper bound of the IRG among the six levels so that it can guarantee that the data block will be kept in the cache until that IRG. For instance, if the IRG of a data block is 3000, it is written with the retention capability of 10^4 seconds. Note that determining the IRG does not always accompany an actual write on cache. For instance, assume that a data block is written with the retention capability of 10^4 and it is re-referenced after 2000 seconds. Then, the IRG of this block is now set to 2000 seconds. However, as the remaining retention capability of 8000 seconds can still satisfy the next IRG, the adaptive write does not perform the actual write to NVM. The question now, with AACM, is how accurate our IRG-based prediction is. We measure the accuracy of our prediction using the method depicted in Figure 8(a), where accuracy is defined as the number of correct predictions over the total number of predictions. Since we write data with a retention capability that is larger or equal to the estimated interval, we count a prediction as correct if the retention capability selected is larger or equal to the actual interval. For instance, at time T_{i+2}, a prediction is counted as correct if $P(t_{i+2}) \geq \Delta t_{i+3}$ where $P(t_{i+2})$ is the predicted interval at T_{i+2} and Δt_{i+3} is the actual interval.



(a) Accuracy Metric



(b) Accuracy results

Fig. 8: Effectiveness of IRG-based prediction

Figure 8(b) shows the accuracy results for the workloads considered. We find that the IRG-based prediction is quite precise, all being larger than 90%. This implies that our prediction method can differentiate data that are worthy to cache from others using IRGs. Hence, AACM can enhance performance without hurting the hit ratio through the use of adaptive writes. To keep the record of IRG level, the IRG-based prediction only needs 144 bytes for each 4KB block.

Let us now discuss the transition from FS to CS. Recall that the transition from TS to CS only happens upon a re-reference. If a data block has a large IRG that does not survive its retention capability, state transition from TS to CS does not occur. For such data, we integrate a ghost buffer [51], which is a set of metadata managed to monitor the behavior of evicted data, into AACM. When a new request is a hit in the ghost buffer (in FS), we can estimate the IRG of the request, allowing us to transition the data from FS to CS.

D. Cache utilization

Let us now consider the use of a cache in our amnesic approach. The cache space used in ACM is determined by the following equation.

$$U = \alpha \times R \tag{1}$$

where U is the cache size used by data, α is the request arrival rate, and R is the average retention time. This equation tells us that the cache size used increases as the request arrival rate and retention time increases.

Note that in AACM, the retention capability of data in CS is determined by the IRGs. For data in TS, we can derive the proper retention capability from Equation 1. Specifically, if the total cache size is S and the space used by the data in CS is S_C, then the retention capability for the relaxed write that can utilize the cache fully is calculated as $\frac{S-S_C}{\alpha}$. α can be assessed by epoch-based monitoring.

The request arrival rate, however, will fluctuate over time resulting in the cache being under utilized or to overflow. When the cache is under utilized, that is, $U < S$ our schemes selectively refreshes the expired data whose IRG is less than the relaxed retention time. This allows the expired data to be treated as a new request.

When the cache overflows, that is, $U > S$ there will be cached data with retention time remaining, but not enough space to service incoming requests. As this is a typical situation that occurs with traditional caches, we take similar measures and evict a block to make space for the new request. As we are managing the IRGs for the blocks in the cache, we choose the victim block in TS or CS, whose remaining time to the estimated next reference is the longest. Algorithm 1 shows the pseudo code for AACM which consists of two key procedures; i) DO_ACCESS() and ii) AMNESIC(), while the former is triggered by cache accesses, the latter one is invoked every second. The DO_ACCESS() uses two arguments, the requested LBA (Logical Block Address) and a flag, denoted RW, to indicate whether this request is read or write. If the request hits in the cache, AACM predicts the IRG and conducts either the adaptive write operation for the write request or the refresh operation for the read request if the remaining retention capability is shorter than the predicted IRG. Otherwise, the

requested data (either given by an application or fetched from storage) is written into the cache with the retention capability predicted using the ghost buffer or Equation 1. On the other hand, AMNESIC() checks whether there are blocks whose retention time is expired. Then, it invalidates them in the cache while writing them back to the storage if they are dirty.

Algorithm 1 AACM algorithm

```
1: procedure DO_ACCESS(LBA, RW)
2:     if LBA cache hit then
3:         pIRG ← IRG_PREDICT()          ▷ predict and update IRG
4:         if RW is READ then
5:             if pIRG > T_remain then   ▷ T_remain is the remain retention capability
6:                 REFRESH(LBA, pIRG − T_remain))
7:             end if
8:         else                                  ▷ write hit case
9:             ADAPTIVE_WRITE(LBA, pIRG)
10:        end if
11:    else                                      ▷ cache miss case
12:        if free block=0 then
13:            EVICT()                           ▷ subsection IV-D
14:        end if
15:        if RW is READ then
16:            Read from storage
17:        end if
18:        if ghost cache hit then
19:            pIRG ← IRG_PREDICT()
20:        else
21:            pIRG ← proper retention time      ▷ equation 1
22:        end if
23:        ADAPTIVE_WRITE(LBA, pIRG)
24:    end if
25: end procedure

26: procedure REFRESH(LBA, pIRG)
27:    Read from cache
28:    ADAPTIVE_WRITE(LBA, pIRG)
29: end procedure

30: procedure AMNESIC( )                      ▷ run every second
31:    for each list (1 for TS and 6 for CS) do
32:        blocks ← expiration candidate in list
33:        for each block b ∈ blocks do
34:            if b is dirty then
35:                WRITE-BACK(b)
36:            end if
37:            State_b ← FS                  ▷ state of b is free
38:        end for
39:    end for
40: end procedure
```

V. EVALUATION

In this section, we first discuss the experimental environment. Then, we discuss how our proposed schemes affect performance, energy consumption, and endurance, in sequence.

A. Experimental environment

Our experiments are conducted via trace-driven simulations. We use in-house NVM based cache simulator, which consists of two main components. One is a trace replayer that reads a trace (eg. MSR Cambridge trace) and composes the corresponding I/O requests based on the time recorded in the trace. The other component is a storage emulator which holds a request for latency time. We use two storage emulators for SSD and PCM. The simulator is a time accurate, implying that it responds a request according to the latency parameters of SSD and PCM. Table II summarizes the parameters extracted from previous work [52], [53]. The write latencies reduced by retention relaxation are estimated using the model proposed by Liu et al. [7]. Specifically, by relaxing from 10^7 to 10^6 seconds, we can obtain 1.2x write speedup, while relaxing to 10^5, 10^4, 10^3, and 10^2 yields 1.5x, 1.7x, 1.9x and 2.1x speedup, respectively. For cache management, it makes use of 8 lists, one for free blocks, another one for blocks in the tentative state and other six for the six IRG levels in the confirmed state. We have implemented not only the proposed schemes, SACM and AACM, but also the traditional LRU and REF schemes for comparison purposes. In current implementation, the ghost buffer can maintain information up to 1K blocks.

TABLE II: Experimental parameters

Parameter	PCM	SSD
Read latency	16 us	50 us
Write latency	91.2 us	900 us
Read energy	81.9 nj	14.25 uj
Write energy	4.73 uj	256 uj

We use several real-world workloads such as those from MSR Cambridge [44], the FIU traces [54], and the web search engine [55] as summarized in Table I. The MSR Cambridge traces cover 36 volumes from various servers and we select 10 of them. The webmail, webmail with online (denoted 'wm+online'), and homes of the FIU traces are traces of 21 days of mail, course management activities, and the NFS server, respectively. Finally, the web search workload (denoted 'Websearch3') contains the I/O traces for a web search engine. The workloads used show a spectrum of read-intensive to write intensive workloads. Unless stated otherwise, the results presented are for the cache size set to be 25% of the working set of each workload. We also show results for other cache sizes later in this section.

B. Performance, energy, and endurance

Figure 9 shows the hit ratio results for the various schemes. The results show LRU and REF having the same hit ratio. This is natural as the blocks that they hold in the cache are the same. In SACM, the hit ratio is affected by the separation of the TS and CS states. They result in two different effects. One, it differentiates the less cacheable data from others improving the hit ratio, and two, it decreases the hit ratio due to retention relaxation. The result for SACM is that the hit ratios are comparable to LRU giving and taking a little bit depending on the workload. With AACM, IRG information allows for more accurate management, that is, through retention relaxation more cache space is made available for more cacheable data.

Figure 10 presents the average latency of the considered schemes normalized to that of LRU. In this experiment, we assume that all refreshing time can be hidden by conducting it

TABLE I: Workload characteristics

Workload	Read	Write	Working set	Duration	Description
hm_0	11.0 GB	22.9 GB	7.8 GB	7days	Hardware monitoring
mds_0	3.3 GB	7.8 GB	3.9 GB	7days	Media server
prn_0	13.2 GB	53.6 GB	22.5 GB	7days	Print server
prn_1	181.4 GB	30.8 GB	88.2 GB	7days	Print server
$proj_3$	18.2 GB	2.6 GB	8.5 GB	7days	Project directory
$rsrch_0$	1.4 GB	11.0 GB	1.3 GB	7days	Research projects
$src2_0$	1.4 GB	9.9 GB	1.8 GB	7days	Source control
stg_0	7.4 GB	15.8 GB	8.1 GB	7days	Web staging
ts_0	4.1 GB	11.8 GB	2.2 GB	7days	Terminal server
usr_0	35.4 GB	13.3 GB	4 GB	7days	User home directory
$webmail$	5.4 GB	24.3 GB	1.9 GB	20 days	Web mail
$wm + online$	11.9 GB	42.6 GB	2.1 GB	21 days	Course management
$homes$	15.5 GB	65.3 GB	17.3 GB	21 days	File server
$Websearch3$	62.6 GB	32.5 MB	6.5 GB	3.2 days	Search engine

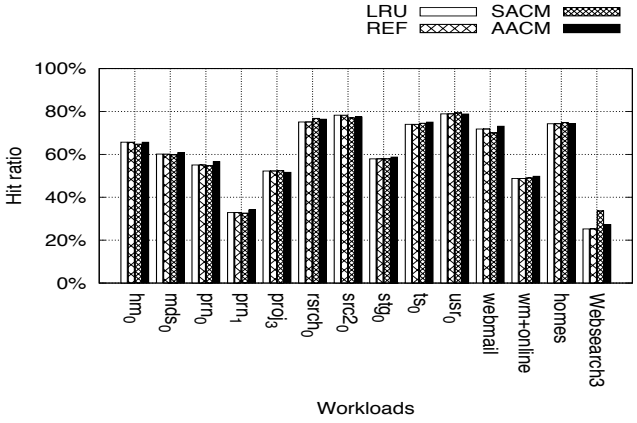


Fig. 9: Hit ratio

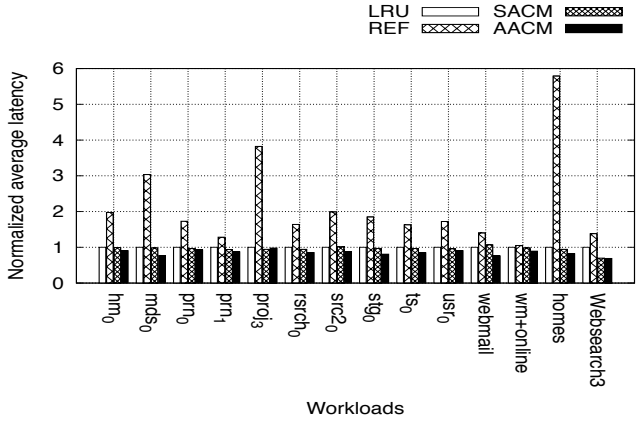


Fig. 11: Normalized average latency (refreshing is visible to user)

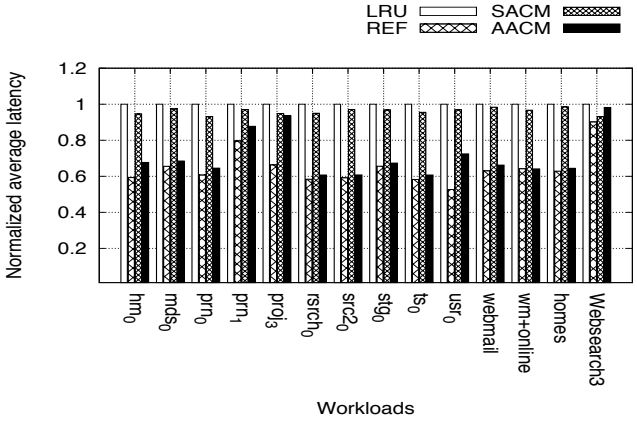
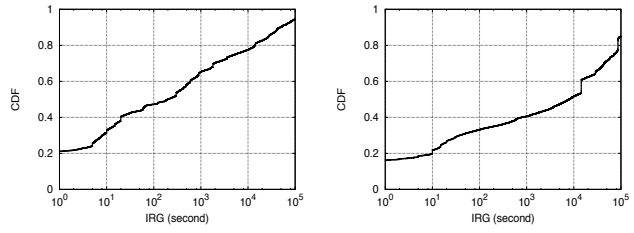


Fig. 10: Normalized average latency (with refresh being done in the background, hence, hidden from user)

completely in background mode. It shows that in comparison with LRU 1) AACM reduces latency by as much as 40% with an average of 30%, 2) REF reduces latency even more by as much as 48% (36% on average), and 3) SACM reduces latency by as much as 7% (4% on average).

Now, let us consider the refreshing overhead. Note that refreshing is required for REF to periodically replenish the retention capability, for SACM to transition data from TS to CS when handling read requests, and for AACM to guarantee the estimated intervals in CS. Figure 11 shows the normalized latency including the refreshing overhead. The results show that REF suffers considerably, while SACM and AACM still perform better than LRU though the margin has dwindled. The reason they still perform better is because of the performance gains obtained through retention relaxation even though they pay for the refreshing overhead. Note that, in reality, some refreshing overhead will be hidden while others exposed, yielding performance in between Figure 10 and Figure 11.

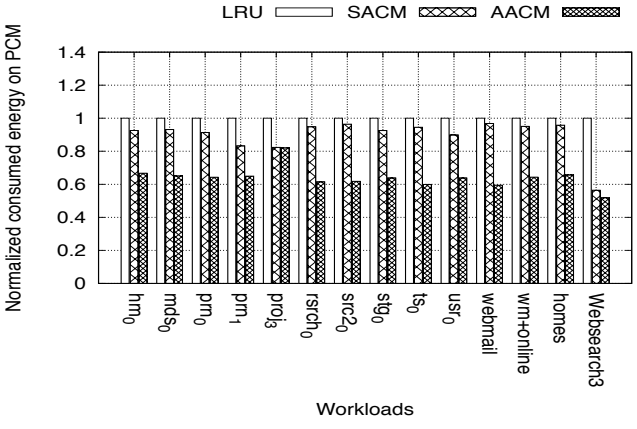
Figure 12 reveals one of the reasons why AACM scheme gains in performance. In the figure, we measure the intervals between two consecutive writes and draw the cumulative distribution of intervals. The results show that 40%$\sim$60% of written data are updated within 10^2 seconds. LRU writes these data with the retention capability of 10^7 seconds. In contrast, AACM writes them with the appropriate relaxed capability



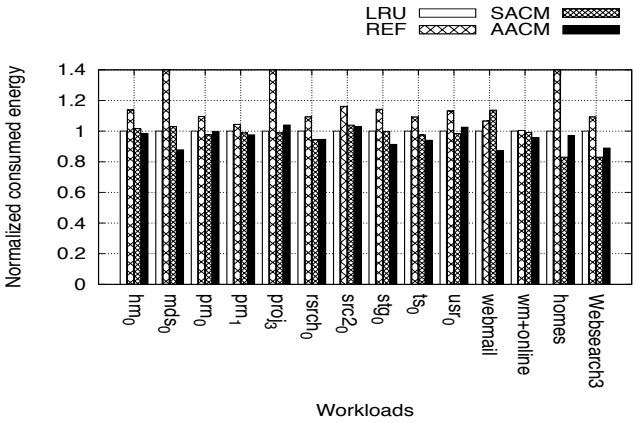
(a) prn_0 workload (b) prn_1 workload

(c) mds_0 workload (d) hm_0 workload

Fig. 12: Distribution of intervals of consecutive writes



(a) PCM cache



(b) Whole storage system

Fig. 13: Energy consumption

based on IRG, resulting in the performance improvement as shown in Figure 10. Note that the performance improvement also comes from the accuracy of the IRG-based prediction shown with Figure 8 in Section IV-C.

Figure 13 shows the energy consumption from two viewpoints, one consumed by the PCM cache only and the other consumed by both the PCM cache and SSD storage. Energy is calculated using the equation, $E = N_{read} * E_{read} + N_{write} * E_{write}$, where N_{read} and N_{write} are the number of reads and writes, respectively, and E_{read} and E_{write} are energy consumed for each read and write, respectively. The number of reads and writes are measured during simulation, while the energy values shown in Table II are used for the default read and write operations. For the relaxed write operation, we adopt the model proposed by Liu et al. [7] that estimates the energy savings by considering the reduction of iterations in the write process due to retention relaxation.

Figure 13(a) shows that the energy consumed relative to using the conventional LRU. Note that, for readability, we do not show the results for REF as they are substantially higher being as much as 9 times higher than LRU. The results show that compared to LRU, AACM and SACM both reduces energy consumption, the savings being on average 37% (and as high as 49%) and 11% for AACM and SACM, respectively. When we consider the whole storage system, Figure 13(b) shows that AACM saves energy by an average of 13%. Energy saving comes from two sources, retention relaxation in PCM and reduction of accesses in SSD, obtained by the increased cache hit ratio.

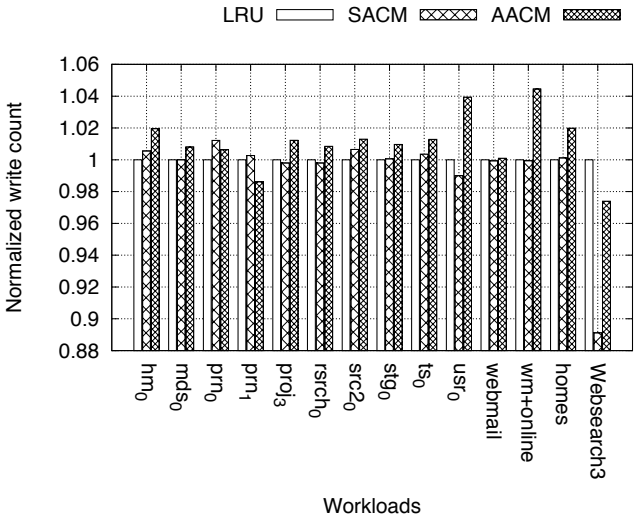
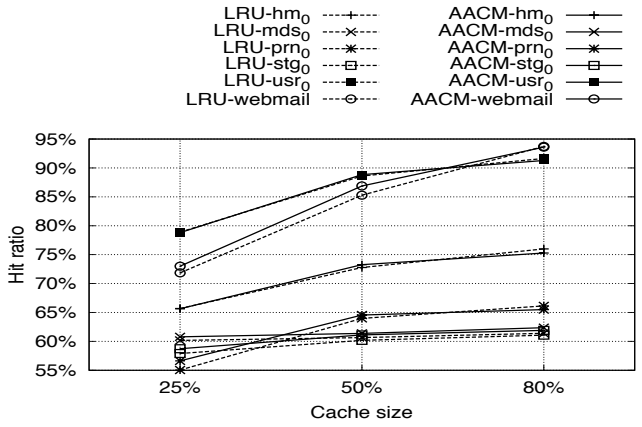


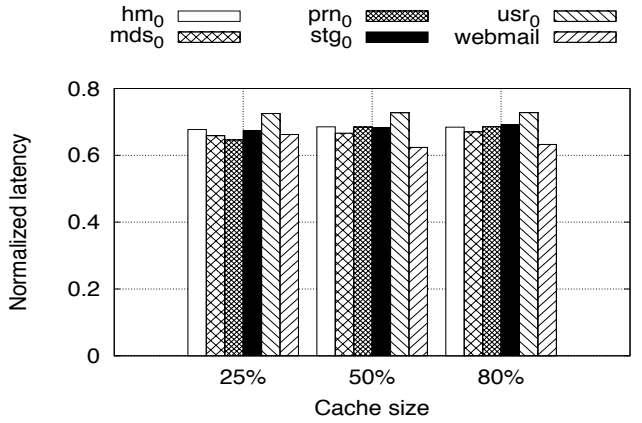
Fig. 14: Endurance

One concern of our scheme is the endurance of PCM since our proposal incurs additional writes. Specifically, SACM requires additional writes when it transitions data from TS to CS, while AACM requires it for guaranteeing the estimated IRGs. However, Figure 14 shows that the additional writes are not significant with SACM showing similar write counts to LRU, while AACM incurs roughly 1% (4% at maximum) more writes compared to LRU. We observe that the enhanced hit ratio compensates for these additional writes. In this figure, we again omit the results for REF that is 5 times higher than LRU. Considering the MLC PCM endurance (10^5 [56]) and the total amount of writes (wm+online), we can estimate that the lifetime of the PCM cache is around 26 years which

is similar to that under the LRU scheme. Another concern of our technique is the data integrity brought by retention relaxation. To address this issue, we can employ an integrity check mechanism such as cyclic redundancy check (CRC), but this is beyond of our scope.



(a) Hit ratio



(b) Latency (AACM latency normalized to LRU)

Fig. 15: Performance under different cache sizes (25%, 50%, 80% of working set of each workload)

Now let us turn our focus on the results with different cache sizes. For simplicity, we only consider AACM in discussing these results. Figure 15 shows the hit ratio and latency of LRU and AACM when we increase the cache size so that it can contain 25% (the results presented so far), 50% and 80% of the working set of each workload. In terms of the hit ratio, we find that 1) when the cache size is set to be small, AACM performs better since its ability to forget makes more room for more cacheable data and 2) when the cache size becomes larger, both schemes show comparable performance since LRU also keeps most of the cacheable data. In terms of latency, AACM outperforms LRU due to retention relaxation for all considered cache sizes. From the results, we also expect AACM to perform well for environments where multiple applications with diverse characteristics share the cache space.

Figure 16 shows the proportion of data that was "evicted" from the cache by exceeding the retention capability limit, that is, forgetting. Recall that for our schemes, data can be evicted from the cache through replacement or by forgetting. From this figure, we observe that when the cache size is set

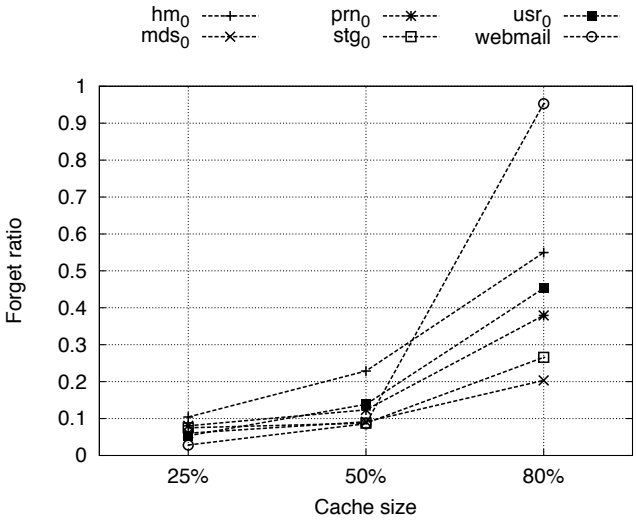


Fig. 16: Proportion of forgetting

to 80% of the working set, 20∼95% (46% on average) of evicted data are due to forgetting. If we consider this finding with Figure 15(a), which shows that LRU and AACM show similar hit ratios, then this tells us that LRU is keeping data in the cache for an unnecessarily long time. AACM forgets these data early, resulting in latency improvements as shown in Figure 15(b). When the cache size is set to 25%, the portion evicted through forgetting is only 5∼10%. However, as cache space is in high demand, this space made available through forgetting allows more worthy data to be brought in to cache resulting in the hit ratio improvement.

VI. Related Work

Previous studies related to our work can be categorized into two groups. The first group of work is about exploiting the tradeoff between the retention capability and write speed while the second group is about using NVM as a file cache. We discuss the two in the following.

Liu et al. propose NVM duet, a novel architecture that unifies working memory and persistent store [7]. It exploits the limited retention capability of PCM to enhance performance by relaxing consistency and durability constraints when PCM is used as working memory while these constraints are guaranteed when it is used for persistent store. Jiang et al. design a write truncate mechanism to decrease the write iterations in PCM, where the retention errors due to the truncation is compensated using the assistance of an extra ECC [47].

Sampson et al. suggest a novel approximate storage that allows errors by reducing the number of programming pulses to improve the performance of PCM [48]. Seong et al. show that PCM is prone to soft errors due to the resistance drift problem [57]. They also propose tri-level-cell PCM, which can lower the errors while enhancing performance.

STT-RAM is considered as an attractive alternative to the conventional on-chip SRAM cache due to its high density, competitive read latency and lower leakage power consumption. However, long write latency is a serious concern and

several previous studies utilize retention relaxation to solve this concern. For instance, Smullen et al. design a reduced-retention STT-RAM cache, which is a hybrid cache with a DRAM-style refresh policy [5]. Sun et al. propose a multi-retention level cache and a dynamic counter-controlled refreshing scheme and employ different levels according to the cache layer [49]. Jog et al. formulate the relationship between retention-time and write-latency and suggest optimal retention-time for efficient cache hierarchy [33].

In the flash memory-based storage domain, retention relaxation has also exploited to boost performance and lifetime. Liu et al. observe that 49∼99% of writes require less-than 1-week retention time [42]. Based on this observation, they design a retention-aware FTL that supports two write modes, retention-relaxed mode and normal mode, and perform periodic reprogramming. Cai et al. suggest a retention-aware error management scheme that makes use of retention relaxation to reduce the ECC overhead and periodic reprogramming (or remapping) to enhance the lifespan of storage [6]. Pan et al. propose a quasi-nonvolatile SSD and scheduling scheme to minimize the refreshing impact on performance [58].

Our work is similar to these previous studies in that retention relaxation is used to enhance performance or lifetime of storage. However, our approach is novel in that we make use of the data loss characteristics, that is, the ability to forget, where as all previous studies rely on refreshing to deal with relaxed retention.

The second group studies related to our work is on those that make use of NVM as a file cache. Kim et al., with real-world traces, demonstrate that PCM-based caching is a viable, cost-effective option for enterprise storage systems [41]. Lee et al. design a novel scheme, called in-place commit, that exploits the non-volatility of NVM [35]. By unioning the buffer cache with journaling layer, it can enhance performance significantly without any loss of reliability.

Fan et al. design a new replacement policy for an NVM cache, called H-ARC (Hierarchical Adaptive Replacement Cache) [36]. It considers not only the conventional factors such as recency and frequency but also NVM-related factors such as dirty and clean for replacement decisions. Liu et al. propose a hash-based caching scheme to improve the random write performance for their PCM-HDD storage architecture [37]. Lee et al. discuss the characteristics of NVM and show that a new metric is required for an NVM cache [38].

All these studies discuss ways to increase the effectiveness of an NVM cache. However, they do not consider the limited retention capability, which is the main focus in our work. One exception is the work by Huang et al. that considers ECC relaxation to reduce the ECC overhead when the SSD is used as a cache [59]. However, to compensate for relaxation, they periodically read data from storage instead of attempting adaptive write or forgetting as we do. To the best of our knowledge, this is the first work that introduces the use of the amnesic notion to balance the retention capability and write performance.

VII. Conclusion

Recently as data becomes bigger and cloud computing prevails, requirement for placing data closer to consumers, such as content delivery network (CDN), also increases rapidly. Applying NVM as a cache is accepted as one promising solution for this requirement. In this paper, we explore new cache management schemes that introduce the amnesic notion to balance the limited retention capability and write speed. Experimental results show that our proposal is effective in terms of performance and energy consumption.

There are two research directions as a future work. One is applying our concept to other resource managements such as approximate computing, retention relaxed storage and zombie memory. The second direction is extending our scheme so that it can reflect another characteristics of NVM such as read/write latency asymmetry and endurance. For instance, we expect that the IRG information can be exploited usefully for wear-leveling in NVM.

Acknowledgment

We would like to thank to our shepherd, Prof. Myoungsoo Jung, and anonymous reviewers for their insightful comments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012R1A2A2A01014233)

References

[1] R. F. Freitas and W. W. Wilcke, "Storage-class memory: the next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4, 2008.

[2] K. Bailey, L. Cede, S. D. Gribble, and H. M. Levy, "Operating system implications of fast, cheap, non-volatile memory," in *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, ser. HotOS, 2011.

[3] O. Zilberberg, S. Weiss, and S. Toledo, "Phase-change memory: An architecture perspective," *ACM Computing Surveys*, vol. 45, no. 3, 2013.

[4] M. Awasthi, M. Shevgoor, K. Sudan, R. Balasubramonian, B. Rajendran, and V. Srinivasan, "Handling PCM resistance drift with device, circuit, architecture, and system solutions," in *Non-Volatile Memories Workshop*, ser. NVMW, 2011.

[5] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proceedings of the 17th IEEE Symposium on High Performance Computer Architecture*, ser. HPCA, 2011.

[6] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash correct-and refresh: Retention-aware error management for increased flash memory lifetime," in *Proceedings of the 30th IEEE International Conference on Computer Design*, ser. ICCD, 2012.

[7] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang, "NVM duet: Unified working memory and persistent store architecture," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, 2014.

[8] V. Phalke and B. Gopinath, "An inter-reference gap model for temporal locality in program behavior," in *Proceedings of the ACM SIGMETRCS joint international conference on Measurement and modeling on computer systems*, ser. SIGMETRICS, 1995.

[9] W. Zhao, E. Belhaire, Q. Mistral, C. Chappert, V. Javerliac, B. Dieny, and E. Nicolle, "Macro-model of spin-transfer torque based magnetic tunnel junction device for hybrid magnetic-CMOS design," in *Proceedings of the International Behavioral Modeling and Simulation Workshop*, 2006.

[10] R. Degraeve, A. Fantini, S. Clima, B. Govoreanu, L. Goux, Y. Y. Chen, D. Wouters, P. Roussel, G. Kar, G. Pourtois, S. Cosemans, J. Kittl, G. Groeseneken, M. Jurczak, and L. Altimime, "Dynamic hourglass model for SET and RESET in HfO2 RRAM," in *Proceedings of the Symposium on VLSI Technology*, 2012.

[11] Viking Technology, "Understanding non-volatile memory technology whitepaper," 2012, http://www.vikingtechnology.com/uploads/nv_whitepaper.pdf.

[12] E. Kultursay, M. Kandemir, S. A., and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *IEEE International Symposium on Performance Analysis of Systems and Software*, ser. ISPASS, 2013.

[13] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA, 2009.

[14] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using PCM technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA, 2009.

[15] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA, 2009.

[16] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: a hybrid pram and dram main memory system," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC, 2009.

[17] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS, 2011.

[18] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories," in *IEEE 30th International Conference on Computer Design*, ser. ICCD, 2012.

[19] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP, 2009.

[20] X. Wu and A. L. N. Reddy, "SCMFS:a file system for storage class memory," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC, 2011.

[21] S. Pelley, T. F. Wenisch, B. T. Gold, and B. Bridge, "Storage management in the NVRAM era," *VLDB Endowment*, vol. 7, no. 2, 2013.

[22] A. Wang, P. Reiher, G. Popek, and G. Kuenning, "Conquest: Better performance through a disk/persistent-ram hybrid file system," in *Proceedings of the 2002 USENIX Annual Technical Conference*, ser. ATC, 2002.

[23] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2010.

[24] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "Nv-heaps: making persistent objects fast and safe with next-generation, non-volatile memories," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS, 2011.

[25] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: lightweight persistent memory," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS, 2011.

[26] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2013.

[27] S. Venkataraman, N. Tolia, P. Ranganathan, and R. H. Campbell, "Consistent and durable data structures for non-volatile byte-addressable memory," in *Proceedings of the 9th USENIX conference on File and stroage technologies*, ser. FAST, 2011.

[28] J.-Y. Jung and S. Cho, "Memorage: Emerging persistent ram based malleable main memory and storage architecture," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS, 2013.

[29] S. Baek, J. Choi, D. Lee, and S. H. Noh, "Energy-efficient and high-performance software architecture for storage class memory," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 3, 2013.

[30] S. Oikawa, "Integrating memory management with a file system on a NVM," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC, 2013.

[31] D. Narayanan and O. Hodson, "Whole-system persistence," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, 2012.

[32] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili, "An energy efficient cache design using STT RAM," in *ACM/IEEE International Symposium on Low-Power Electronics and Design*, ser. ISLPED, 2010.

[33] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and D. C. R., "Cache revive: Architecting volatile STT-RAM caches for enhanced performance in cmps," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC, 2012.

[34] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE, 2010.

[35] E. Lee, H. Bahn, and S. H. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, ser. FAST, 2013.

[36] Z. Fan, D. H. C. Du, and D. Voigt, "H-ARC: A non-volatile memory based cache policy for solid state drives," in *IEEE 30th Symposium on Mass Storage Systems and Technologies*, ser. MSST, 2014.

[37] Z. Liu, B. Wang, P. Carpenter, J. S. Li, Dongand Vetter, and W. Yu, "PCM-based durable write cache for fast disk I/O," in *IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS, 2012.

[38] K. Lee, I. Doh, J. Choi, D. Lee, and S. H. Noh, "H-ARC: A non-volatile memory based cache policy for solid state drives," in *Proceedings of Advances in Computer Science and Technology*, ser. ACTA, 2007.

[39] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and C. E. Schrock, "Janus: Optimal flash provisioning for cloud storage workloads," in *Proceedings of the 2013 USENIX Annual Technical Conference*, ser. ATC, 2013.

[40] D. A. Holland, E. Angelino, G. Wald, and M. I. Seltzer, "Flash caching on the storage client," in *Proceedings of the 2013 USENIX Annual Technical Conference*, ser. ATC, 2013.

[41] H. Kim, S. Seshadri, C. L. Dickey, and L. Chui, "Evaluating PCM for enterprise storage systems: A study of caching and tiering approach," in *Proceedings of the 12th USENIX conference on File and stroage technologies*, ser. FAST, 2014.

[42] R.-s. Lui, C. Yang, and W. Wu, "Optimizing NAND flash-based SSDs via retention relaxation," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST, 2012.

[43] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha, "Performance trade-offs in using nvram write buffer for flash memory-based storage devices," *IEEE Transactions on Computers*, vol. 6, no. 58, 2009.

[44] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage*, vol. 3, no. 4, 2008.

[45] JESD218A:, "Solid-state drive (SSD) requirements and endurance test method," 2011, http://www.jedec.org/standards-documents/docs/jesd218a.

[46] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh"," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA, 2012.

[47] L. Jiang, B. Zhao, Y. Zhang, Y. Jun, and B. R. Childers, "Improving write operations in MLC phase change memory," in *Proceedings of the 18th IEEE Symposium on High Performance Computer Architecture*, ser. HPCA, 2012.

[48] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2013.

[49] Z. Sun, X. Bi, H. H. Li, W. F. Wong, O. Z. L., X. Zhu, and W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh

scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2011.

[50] D. Qin, A. D. Brown, and A. Goel, "Reliable writeback for client-side flash caches," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 451–462. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/qin

[51] R. H. Patternson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proceedings of the ACM SIGOPS 15nd symposium on Operating systems principles*, ser. SOSP, 1995.

[52] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, 2012.

[53] B. Yoo, Y. Won, S. Cho, S. Kang, J. Choi, and S. Yoon, "SSD characterization: From energy consumption's perspective," in *Proceedings of the USENIX Hot Storage*, 2011.

[54] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "SRCMap: energy proportional storage using dynamic consolidation," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST, 2010.

[55] UMASS trace, http://traces.cs.umass.edu/index.php/Storage/Storage.

[56] F. Bedeschi, R. Fackenthal, C. Resta, E. M. Donze, M. Jagasivamani, E. C. Buda, F. Pellizzer, D. W. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande, "A Bipolar-Selected Phase Change Memory Featuring Multi-Level Cell Storage," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 1, pp. 217–227, 2009.

[57] N. H. Seong, S. Yeo, and H.-H. S. Lee, "Tri-level-cell phase change memory:toward an efficient and reliable memory system," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA, 2013.

[58] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile ssd: Trading flash memory nonvolatility to improve storage system performance for enterprise applications," in *Proceedings of the 18th IEEE Symposium on High Performance Computer Architecture*, ser. HPCA, 2012.

[59] P. Huang, P. Subedi, X. He, S. He, and K. Zhou, "FlexECC: Partially relaxing ecc of mlc ssd for better cache performance," in *Proceedings of the 2014 USENIX Annual Technical Conference*, ser. ATC, 2014.