Received April 1, 2021, accepted April 15, 2021, date of publication April 20, 2021, date of current version April 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3074179

Characterization of Android Memory References and Implication to Hybrid Memory Management

SOYOON LEE AND HYOKYUNG BAHN, (Member, IEEE)
Department of Computer Engineering, Ewha University, Seoul 120-750, South Korea

Corresponding author: Hyokyung Bahn (bahn@ewha.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIP) under Grant 2019R1A2C1009275, and in part by the ICT Research and Development Program of MSIP/IITP, developing system software technologies for emerging new memory that adaptively learn workload characteristics, under Grant 2019-0-00074.

ABSTRACT In this article, we analyze Android applications' memory reference behaviors, and observe that smartphone memory accesses are different from traditional computer systems with respect to the following five aspects: 1) A limited number of hot pages account for a majority of memory writes, and these hot pages have similar logical addresses regardless of application types; 2) The identities of these hot pages are shared library, linker, and stack regions; 3) The memory access behaviors of hot pages do not change significantly as time progresses even after applications finish their launching; 4) The skewness of memory write accesses in Android is extremely stronger than that of desktop systems; 5) In predicting re-reference likelihood of hot pages, temporal locality is better than reference frequency. Based on these observations, we present a new smartphone memory management scheme for DRAM-NVM hybrid memory. Adopting NVM is effective in power-saving of smartphones, but NVM has weaknesses in write operations. Thus, we aim to identify write-intensive pages and place them on DRAM. Unlike previous studies, we prevent migration of pages between DRAM and NVM, which eliminates unnecessary NVM write traffic that accounts for 32-42% of total write traffic. By judiciously managing the admission of hot pages in DRAM, our scheme reduces the write traffic to NVM by 42% on average without performance degradations.

INDEX TERMS Android, smartphone, application, memory reference, NVM, write operation, hybrid memory.

I. INTRODUCTION

With the recent proliferation of mobile applications as well as the advances in software platform technologies, smartphones have become indispensable devices in our daily lives [1]. More and more people are working with their smartphones, and various applications including social network services, online games, video streaming, and location-based services, emerge every day [2]–[4]. In reality, the hardware specification of the current smartphone device has reached to that of a desktop or a laptop computer system [5]. For example, Google Pixel 4a, the most recent version of the Android reference phone, consists of Qualcomm Snapdragon 730G, Octa-core CPU of 2 $\times$ 2.2 GHz Kryo 470 Gold & 6 $\times$ 1.8 GHz Kryo 470 Silver, Adreno 618 GPU, 6 GB LPDDR4X memory, and 128GB UFS 2.1 storage, which is sufficient to perform multitasking [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

A smartphone is not a personal entertainment device any longer, and some official works like video conferencing, stock trading, or social broadcasting, are being performed on smartphones or tablets. Also, desktop applications are increasingly compatible with smartphones by utilizing external I/O devices.

To accommodate more and more applications, the memory capacity of smartphones keeps increasing, which also increases the power consumption significantly. As a battery-based device, energy consumption is one of the major concerns in the design of smartphone systems. It is reported that main memory accounts for the dominant portion of the total system energy in smartphones due to the ever growing size of DRAM to accommodate more applications [7]. Due to its volatile characteristics, DRAM needs consistent recharge of power to maintain its data even though no read/write operation is being performed. This recharge of power, which we call the refresh operation, accounts for a significant portion of memory power consumption as the size of DRAM increases [8], [9].

Non-volatile memory (NVM) technologies have caught interest as an attempt to reduce the power consumption of DRAM. NVM such as PCM (phase change memory) and STT-MRAM (spin-transfer torque magnetic random access memory) is a byte-addressable memory medium like DRAM but it spends less power as it is non-volatile and thus does not need refresh operations [9]–[11], [23]. NVM also has better scalability than DRAM [24], [25].

Despite these prominent features of NVM (i.e., zero refresh power and high density), NVM has critical weaknesses in write operations that make the total substitution of DRAM memory difficult. First, the number of write operations allowed for each NVM cell is limited. For example, the current write endurance of a PCM cell is known to be 10^7 to 10^8 [9], [12]. If the number of write operations performed on a PCM cell exceeds this limit, its lifespan ends and the memory space of the cell cannot be used any longer. The second drawback is that the write latency of NVM is longer than that of DRAM [11], [13], [14].

Nevertheless, NVM's prospect is still bright. Recent researches have proposed various techniques to overcome the limitations of NVM, and their results indicate that NVM will be able to reduce the energy consumption of main memory systems significantly while retaining reasonable performance when it is adopted as main memory [11], [13]–[16]. One of the ways to cope with the write latency and the endurance problems of NVM is making use of a small size DRAM along with NVM [11], [14]. This DRAM hides the slow write latency of NVM and also increases the lifetime of NVM by absorbing frequent write operations.

Fig. 1 presents such an architecture by managing DRAM and NVM together under a single physical address space. Though DRAM and NVM hybrid memory systems have already been studied [11], [17], this article shows that previous solutions are not efficient for smartphone memory systems due to some distinct memory access characteristics in Android applications.

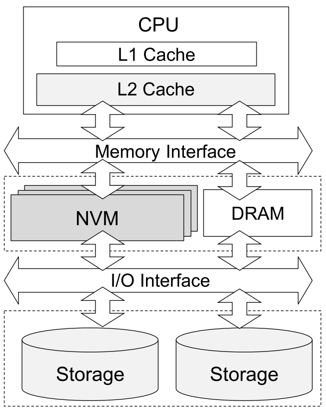


FIGURE 1. DRAM and NVM hybrid memory architecture.

In this article, we analyze the memory access behavior of Android applications specially focusing on write operations, and observe that smartphone memory accesses are different from the memory accesses of traditional computer systems. Based on this observation, we present a new smartphone memory management scheme for a DRAM-NVM hybrid memory architecture. Our idea is that a write operation on NVM incurs performance and endurance problems, and thus identifying write-intensive data and placing them on DRAM can resolve the weaknesses of NVM. This is performed by the extensive analysis of memory reference traces extracted from smartphone applications, which is different from the existing data placement schemes that dynamically relocate pages between DRAM and NVM in an online manner [17], [18]. Our analysis shows that online relocations between DRAM and NVM amplify NVM writes significantly in case of smartphone systems, and thus allocation based on a prior analysis is more effective. Our experimental results with various Android applications show that the proposed scheme reduces the write traffic to NVM by 42% on average and up to 87% without performance degradations.

A. CONTRIBUTIONS

The first contribution of this article is the characterization of memory references in Android applications, specially focusing on write operations. Write reference characterization has largely been ignored in past studies, although some studies observe memory writes in desktop systems [17], [19]. This study focuses on characterizing write references of Android generated by a variety of applications, and find out the following important observations that are significantly different from desktop system cases.

1) A limited number of hot pages account for a majority of memory writes in Android applications. Unlike desktop applications, these hot pages have similar logical addresses regardless of application types. These logical addresses can be the main target of DRAM placement in our study.
2) The skewness of memory write accesses in Android is extremely stronger than that of desktop systems. In particular, 10-15% of top ranking data account for 80% of total write accesses in Android, whereas 50-60% of top ranking data account for the same 80% writes in desktop systems.
3) We investigate the identities of the hot pages, and find that they consist of shared library, linker, and stack regions. As such hot pages have similar memory addresses, we can place them on DRAM.
4) The memory access behavior of the hot pages does not change significantly as time progresses even after the completion of the application launching. This implies that the admission of data in DRAM does not need to be changed significantly over time as hot pages tend to be accessed consistently.
5) When comparing the temporal locality and frequency properties in estimating the re-reference likelihood of hot pages, temporal locality is the better estimator irrespective of application types. This result also contradicts the analysis of desktop application cases.

TABLE 1. **Memory access characteristics of Android applications used in this article.**

Traces	Total footprint (MB)	Write footprint (MB)	Ratio of read : write	Number of memory references			
				Data read	Instruction read	Data write	Total count
Angrybirds	76.94	45.72	3.50 : 1	13,387,756	980,312	3,822,479	18,201,717
Mxplayer	79.92	47.31	3.66 : 1	13,851,914	567,456	3,782,347	18,190,547
Youtube	68.64	40.95	4.44 : 1	14,040,959	993,316	3,162,229	18,196,504
Browser	259.86	180.08	4.11 : 1	15,272,935	1,622,628	4,104,436	20,999,999
Facebook	198.66	96.11	5.67 : 1	11,121,174	486,165	2,045,716	13,653,055
Farmstory	53.74	29.45	6.24 : 1	12,675,555	447,297	2,101,818	15,224,670

Based on these findings, our second contribution is the design of a new memory management scheme that we call "Caste" for the hybrid memory architecture in Android. Caste makes use of the aforementioned analysis results of the Android application's memory reference characterization to accurately identify hot pages. By so doing, Caste eliminates most of NVM writes by placing write-intensive pages on DRAM. Unlike previous studies, we prevent online relocation of pages between DRAM and NVM, which completely eliminates unnecessary NVM write traffic that accounts for 32-42% of total write traffic. This article also quantifies the number of hot pages that should be allocated to DRAM for the optimized performances under the given system situation, and suggests an appropriate admission control for DRAM. Our scheme periodically monitors the page fault ratio of DRAM memory, and determines the number of hot pages to be allocated to DRAM based on our page fault ratio model.

B. THE REMAINDER OF THE ARTICLE

The remainder of this article is organized as follows. In Section II, we describe the analysis results of memory access behaviors in Android applications. Section III explains the proposed hybrid memory management scheme for smartphone applications in detail. Section IV presents the performance evaluation results to assess the effectiveness of the proposed scheme. Section V summarizes the related works specially focusing on non-volatile memory and hybrid memory technologies. Finally, Section VI concludes this article.

II. ANALYSIS OF MEMORY ACCESS BEHAVIOR IN ANDROID

This section analyzes the memory reference behavior in Android smartphone applications specially focusing on write operations as we are interested in adopting write-vulnerable NVM as memory media.

To collect memory access traces of Android applications, we add trace collector and analyzer codes to the Valgrind toolset [20]. In particular, we modify the source file cg_sim.c of Cachegrind. We filter out memory accesses if they hit from the cache layers (i.e., fist level, second level, and last level caches) and extract only the memory accesses observable at the main memory layer.

We collect the main memory access traces from 6 Android applications, namely, Facebook a social network service, Angrybirds a game, Youtube an online video-streaming service, Farmstory a networked game, Mxplayer a multimedia player, and Browser an Android web browser. Our trace collector and analyzer extract the total memory footprint, memory footprint by write operations, total number of memory accesses, ratio of read/write operations, access types, etc. Some brief characteristics captured from these traces are listed in Table 1.

Fig. 2 shows the memory access count that occurs on each memory page of the logical address space for the six Android applications we analyzed. In the figure, the red plot shows the write access and the blue plot represents the read access. As shown in the figure, a certain number of limited pages account for a large portion of the memory accesses in Android applications. Another interesting result is that the six applications show similar trends, which implies that hot pages accessed in Android applications have similar logical addresses regardless of application types. Note that this is not the case for traditional desktop applications. For a comparison purpose, we plot the memory access count for some desktop applications, gedit, gqview, and xmms, which are frequently used in the memory access characterization studies [17].

Unlike smartphone cases, the memory access count distributions of desktop applications are different as application types are varied. Due to this reason, memory management studies in traditional computer systems depend on the online behavior of each application's pages such as temporal locality or reference frequency rather than trace analysis results.

To analyze the write access characteristics of Android applications further, we investigate the identities of the hot pages. Fig. 4 shows which regions account for hot write references in Angrybirds. Note that similar trends can be observed for the other applications we analyzed. As shown in the figure, most write accesses in Android applications occur on shared library, linker, and stack regions. Fig. 5 magnifies the shared library region of Fig. 4, and the identities of the hot pages consist of libc, libm, libgui, vm heap, vm bitmap, libandroid, etc. As such hot pages have similar

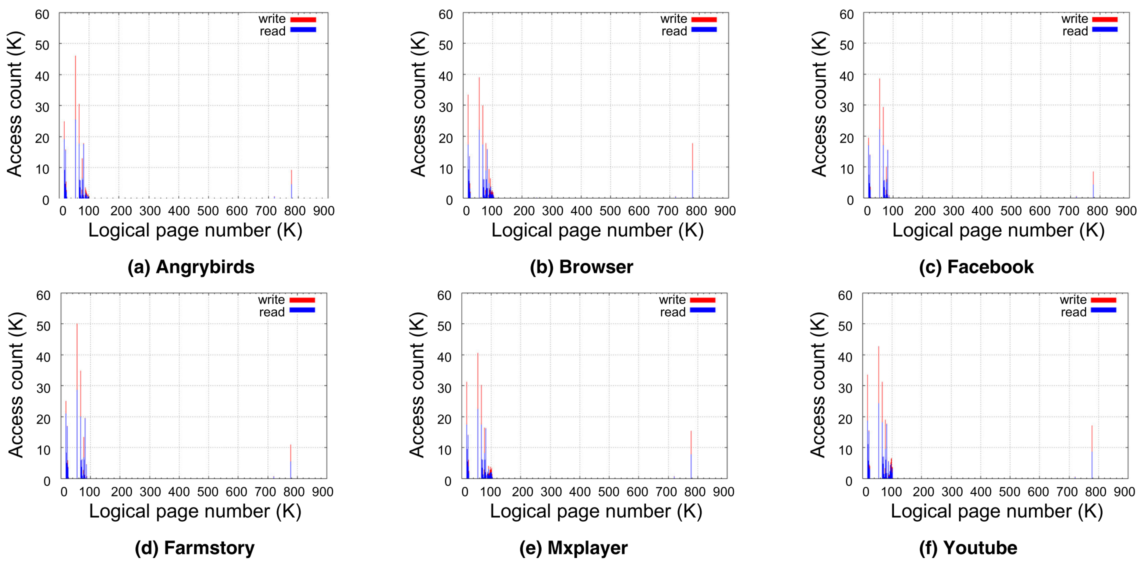


FIGURE 2. Memory access count that occurs on each memory page of the logical address space for Android applications.

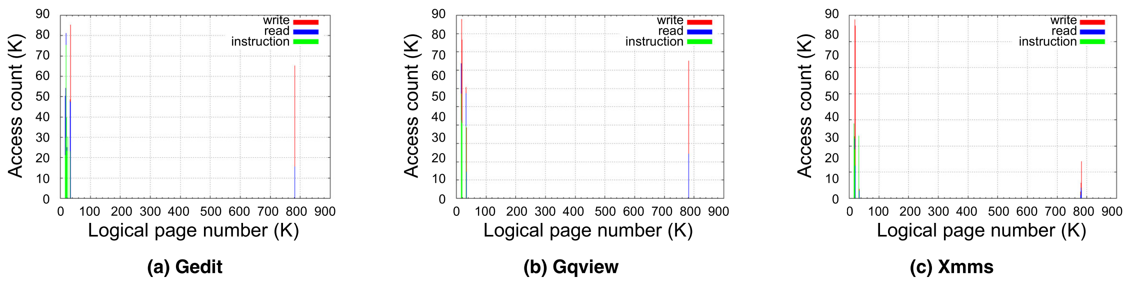


FIGURE 3. Memory access count that occurs on each memory page of the logical address space for desktop applications.

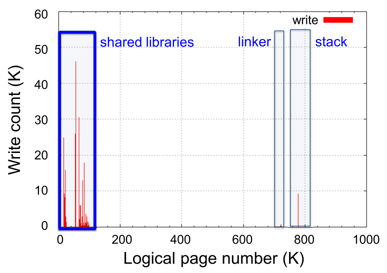


FIGURE 4. Identities of regions that have hot pages.

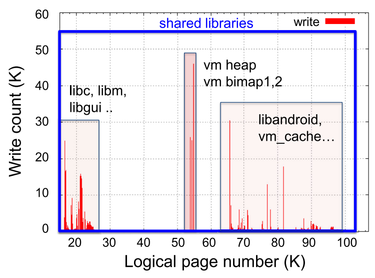


FIGURE 5. Identities of shard library regions that have hot pages.

logical addresses, we can place them on DRAM rather than NVM upon their first accesses.

Fig. 6 shows the distributions of write accesses as the page rankings are varied. In the figure, the x-axis represents the ranking of pages sorted by their total write count and the y-axis represents the number of write accesses on that ranking. As shown in the figure, the write accesses generated by Android applications are extremely skewed. In particular, 10-15% of top ranking pages account for about 80% of total write accesses, implying that write accesses in Android application's memory references mostly result from some hot data. Note that this is different from desktop systems in that 50-60% of top ranking pages usually account for 80% of total write accesses [17].

Fig. 7 shows the logical page numbers that have been accessed as time progresses for each smartphone application. In the figure, the x-axis represents the logical time, which is increased by one for each memory access and the y-axis shows the logical page numbers. Similar to Fig. 2, the red and blue plots represent the write and read operations, respectively. As shown in the figure, memory writes are skewed to a certain number of logical pages and they do not change significantly as time progresses. In the figure, the black vertical line represents the time point that the launch of the application

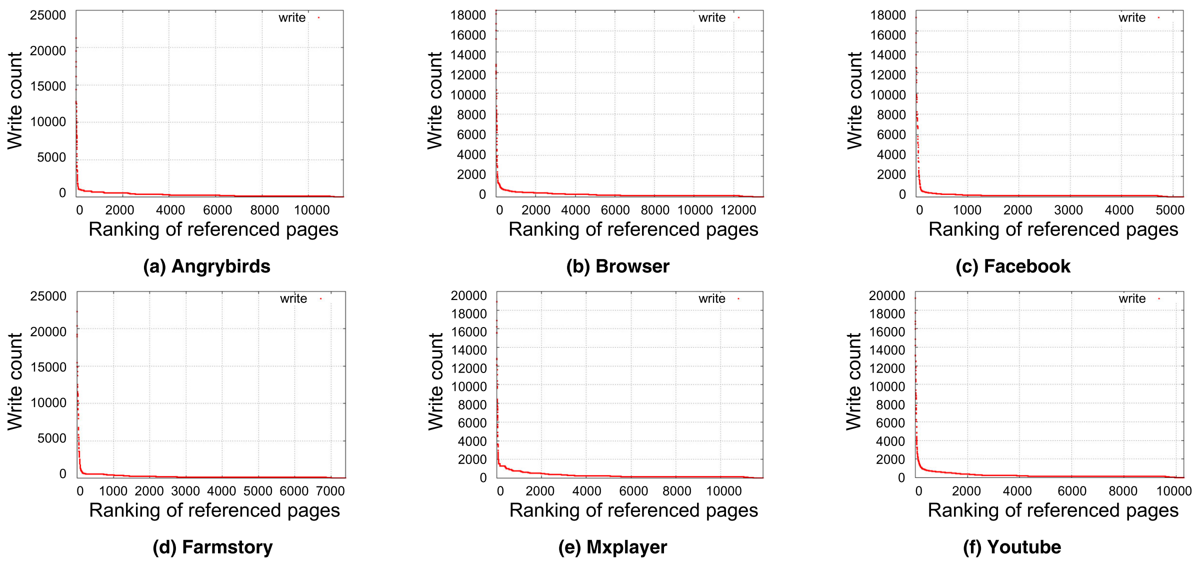


FIGURE 6. Distributions of write accesses as the page rankings are varied for Android applications.

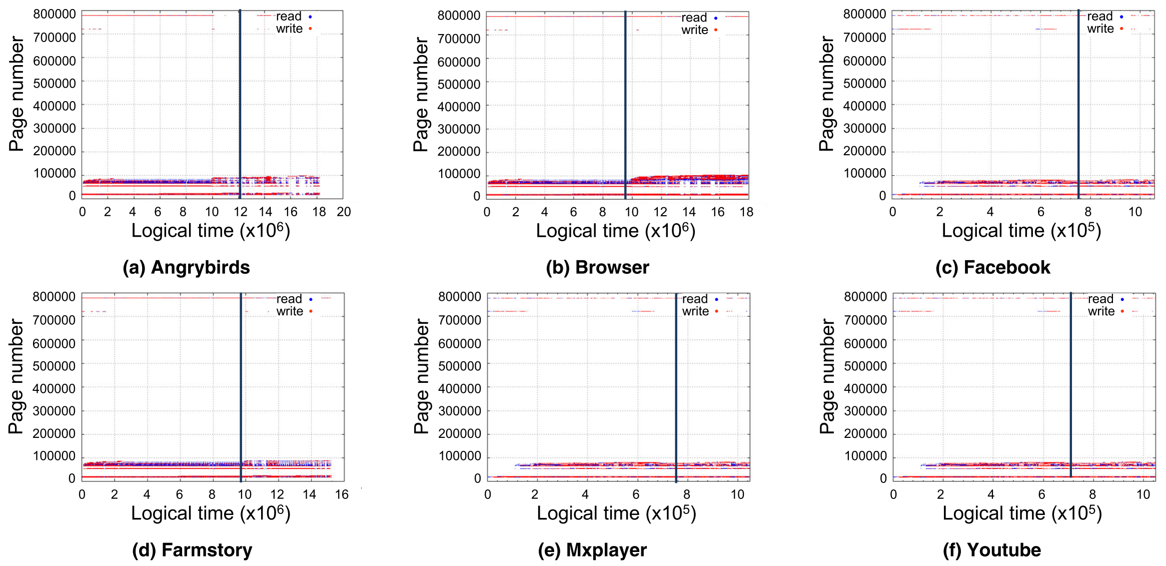


FIGURE 7. Logical page numbers that have been accessed as time progresses for the six Android applications.

has been completed. As we see, the memory access behavior and hot page numbers do not change significantly even after the launch ends.

This implies that once the admission of data in DRAM is determined, it does not need to be changed significantly during the execution of the application as hot pages tend to be accessed consistently.

To reduce the number of writes that occur in NVM, DRAM should absorb as many write accesses as possible. To do this, we need to find a good estimator for future write accesses and evicts those pages not likely to be re-written from DRAM. Temporal locality and reference frequency are well known properties used to estimate the re-reference likelihood of pages [17], [26]. We compare the two properties from the viewpoint of memory write accesses and analyze which is the better estimator.

Fig. 8 shows the effect of temporal locality and frequency on page's write accesses of the six applications. In the figure, the x-axis represents the page ranking with respect to the temporal locality and frequency properties, and the y-axis represents the number of write accesses that occur on the page ranking given in the x-axis. The gray plot and the black plot depict the page rankings based on temporal locality and frequency of page accesses, respectively. As can be seen from the figure, the gray plots are located above the black plots in high rankings.

This implies that the re-reference likelihood of hot pages can be more accurately estimated by temporal locality than frequency of references in Android applications. That is, pages that have been written recently are likely to be re-written in the future. When NVM along with small DRAM is used as main memory, DRAM can absorb most write

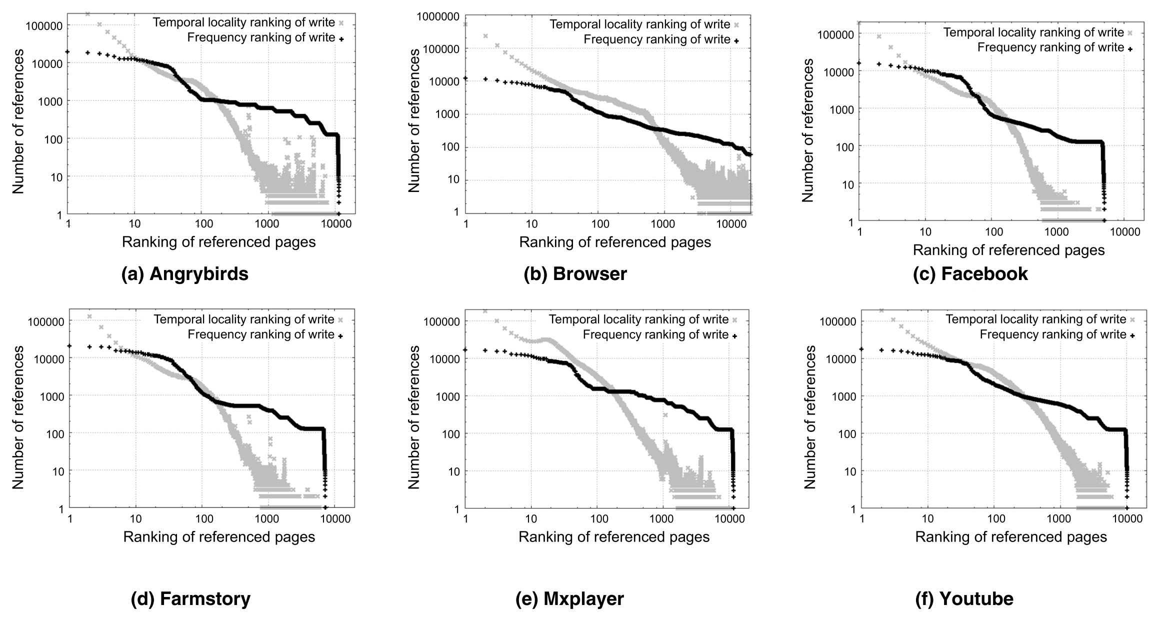


FIGURE 8. Number of writes that occur on the ranking of pages determined by temporal locality and frequency of references in Android.

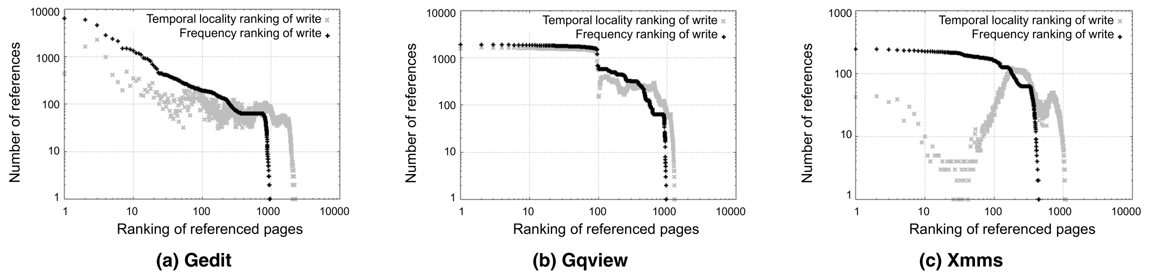


FIGURE 9. Number of writes that occur on the ranking of pages determined by temporal locality and frequency of references in desktop.

accesses by preserving the top ranking pages of temporal locality. Note that this is different from the characteristics of desktop applications, in which frequency based estimation performs better than temporal locality based estimation for the re-reference likelihood of hot pages as shown in Fig. 9 [17]. In summary, temporal locality is a better property than access frequency in estimating the re-reference likelihood of hot pages in Android applications.

III. CASTE: HYBRID MEMORY MANAGEMENT FOR SMARTPHONE APPLICATIONS

In this section, we describe the proposed hybrid memory management scheme for smartphone applications, called Caste.

A. BASIC IDEA

Fig. 10 shows the overall structure of Caste. The basic idea of Caste is that writes on NVM incur performance and endurance problems, and thus identifying write-intensive pages and placing them on DRAM can resolve the weaknesses of NVM.

As we focus on the write characteristics of pages, we consider the hotness of a page based on write references only. In traditional desktop applications, the hotness of pages changes as workloads evolve. Thus, an efficient hybrid memory management should insert a hot page in DRAM, but moves it to NVM if it becomes cold. However, as analyzed in Section II, memory access behaviors in smartphone applications do not change significantly. That is, it does not happen frequently that a hot page becomes cold or a cold page becomes hot. Thus, once a page is identified as a hot (or a cold) page, we aim to place it on DRAM (or NVM), and do not change its location.

Previous studies on hybrid memory management also report that unnecessary page migrations due to incorrect online prediction degrade the total execution time by 25% on average [22]. Thus, we identify hot pages of an application based on the page rankings determined before the execution of the application. Although the page rankings are determined beforehand, we adjust the number of hot pages that will be placed on DRAM according to the system situation changes. This is necessary because the change of memory situations affects the page fault ratio significantly under the fixed DRAM capacity. Thus, our scheme periodically monitors the page fault ratio of DRAM memory, and determines

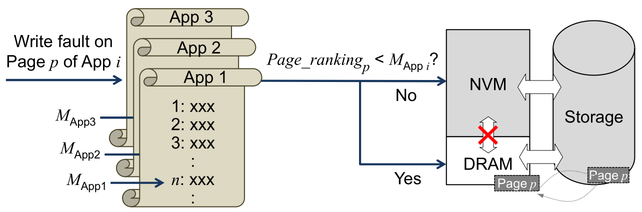


FIGURE 10. Overview of Caste.

the number of hot pages to be placed on DRAM. That is, if the page fault ratio is high, the rankings to be placed on DRAM are decreased, and if the page fault ratio is low, the rankings to be placed on DRAM are increased. This reduces the write traffic to NVM without performance degradations caused by page faults.

B. DETAILS OF CASTE

Caste investigates the write characteristics of each page during the training phase of applications, and determines the page ranking list, which will be maintained. From the next execution of the applications, when a page fault occurs, Caste places the faulted page on DRAM if the page ranking is less than the threshold M, and otherwise places it on NVM. In Caste, moving pages between DRAM and NVM is not allowed. That is, although a page on NVM has been written many times, Caste does not relocate it to DRAM. Similarly, when a page is evicted from DRAM, Caste does not move it to NVM but demotes directly to secondary storage. This is different from previous data placement schemes that allow page swaps between DRAM and NVM; evicted pages from DRAM can be demoted to NVM, and hot pages in NVM can also be promoted to DRAM. The rationale behind this process is to reduce the number of slow storage accesses as well as NVM writes by hierarchical memory management.

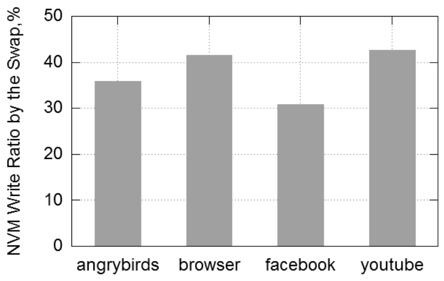
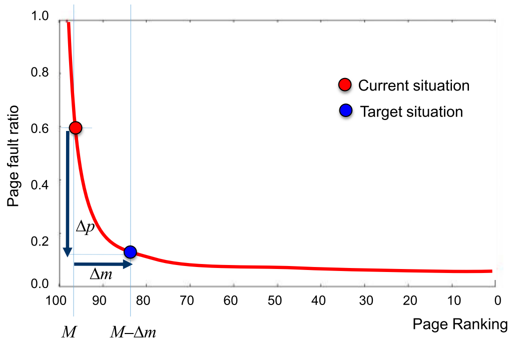
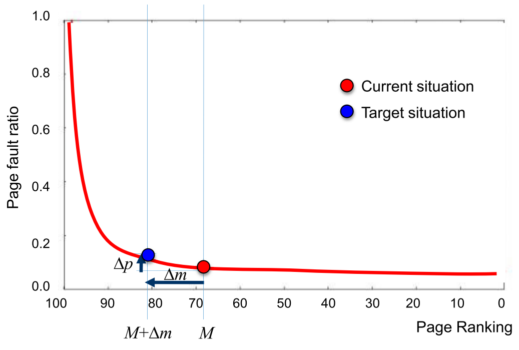


FIGURE 11. The ratio of NVM writes incurred by migrations between DRAM and NVM over total NVM writes.

However, our analysis shows that this incurs too much NVM write traffic in smartphone memory systems. Fig. 11 shows the ratio of writes that occur by NVM-DRAM swap over the total NVM writes when allowing the migration of pages between DRAM and NVM. As shown in the figure, more than 30% of total NVM writes originate from the swaps between DRAM and NVM. This is the reason why Caste is designed not to allow swaps between DRAM and NVM. The only concern Caste may encounter is, then, the performance



(a) High page fault ratio case



(b) Low page fault ratio case

FIGURE 12. Adjustment of the threshold M.

degradation by the increased page fault ratio. To resolve this issue, Caste monitors the page fault ratio of DRAM periodically and adjusts the threshold ranking M that controls the number of pages to be placed on DRAM.

When the page fault ratio is high, Caste decreases M to restrict pages to be placed on DRAM as the DRAM capacity is not sufficient to accommodate current workloads. In this case, Caste focuses on the reduction of page faults rather than NVM writes. In contrast, when the page fault ratio is sufficiently low, Caste increases M to accommodate more pages on DRAM. This eventually leads to the reduction of NVM writes.

To model the expected page fault ratio, we utilize the Belady's lifetime function as it is known that the function accurately approximates the hit ratio for given memory sizes [21]. Fig. 12 shows the two cases of adjusting M based on this model. In Fig. 12(a), as the current page fault ratio is high, Caste decreases M to accommodate less pages on DRAM. Inversely, when the page fault ratio is sufficiently low, Caste increases M to place more pages on DRAM as shown in Fig. 12(b). Fig. 13 shows how our model fits the page fault ratio well for the given page rankings in the workloads we considered. As shown in the figure, our model well fits in case of Facebook and Farmstory, but there are some gaps between the actual page fault ratio and the value estimated by our model as the page ranking threshold decreases in case of Angrybirds and Browser. However, this does not matter as we are interested in the boundary condition for

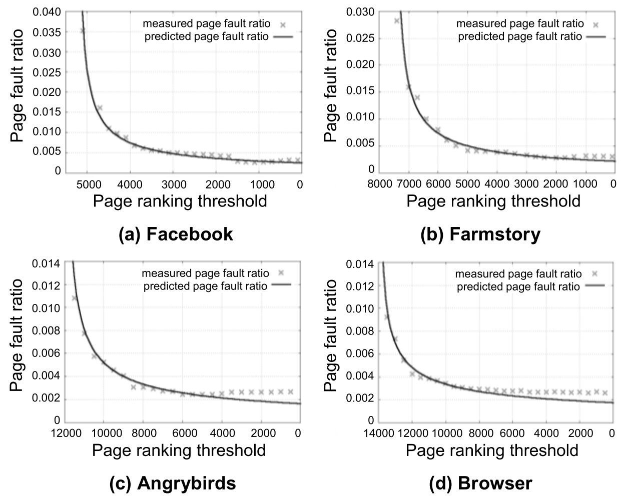


FIGURE 13. Measured page fault ratio versus the page fault ratio estimated by our model.

determining page rankings for DRAM admission, which is far from the tail of the graphs.

Although we adjust the number of pages to place on DRAM, an eviction policy is necessary as the size of DRAM is limited. That is, adding a new hot page to DRAM requires the eviction of an existing page from DRAM if there is no free space in DRAM. As we observed that temporal locality is better than frequency in estimating the re-reference likelihood of hot pages, we use the second-chance algorithm for the eviction of pages in DRAM. The second-chance algorithm is a popular algorithm that reflects the temporal locality of references.

Now, let us explain the details of the Caste algorithm based on the pseudo-code depicted in Algorithm 1. When a page fault occurs, Caste places the faulted page on DRAM if the ranking of the page is less than the threshold M of the application, and otherwise places the page on NVM. If DRAM is full, then a victim page is selected and flushed to storage. To find a replacement victim in DRAM, Caste monitors whether a page has recently been accessed or not by making use of each page's reference bit. As Caste uses the second-chance replacement algorithm, it sequentially scans through DRAM resident pages, and evicts the first page that has the reference bit of 0. For every page with the reference bit of 1 in the course of the scan, Caste clears the bit to 0 instead of evicting the page. As the reference bit of a page is set to 1 upon every page access by the paging unit hardware, a page which is not accessed until the next scan is certain to be evicted. If NVM is full, then Caste selects a victim in NVM by making use of the second-chance algorithm similar to replacement in DRAM. That is, Caste sequentially scans through pages in NVM, and clears the reference bit of the pages. If a page whose reference bit is already cleared is found, Caste evicts this page. If the page has been modified after entering NVM, it is flushed to storage before evicted from NVM.

If the current epoch ends, Caste updates the page ranking threshold M that determines the number of hot pages to be

Algorithm 1

```
procedure caste(page $P$, app $A$)
    if page_ranking ($P$) $< M_A$ then
        insert(DRAM, $P$);
    else
        insert(NVM, $P$);
    end if
    if current epoch ends then
        update_threshold($M_A$);
    end if
end procedure
procedure insert(memory_type $Mem$, page $P$)
    while no free page in $Mem$ do
        $Q \leftarrow$ page pointed by clock-hand$_{Mem}$;
        if reference_bit($Q$) is 0 then
            evict $Q$ from $Mem$;
            if modified_bit($Q$) is 1 then
                flush $Q$ to storage;
            end if
        else
            clock-hand$_{Mem}$ advances to next;
        end if
    end while
    insert $P$ to $Mem$;
end procedure
procedure update_threshold(int $M$)
    if page_fault_rate($M$) - page_fault_rate($M$ - $\Delta M$) $> \mu$
    then
        decrease $M$ by $\Delta M$;
    else if page_fault_rate($M + \Delta M$) - page_fault_rate
    ($M$) $< \mu$ then
        increase $M$ by $\Delta M$;
    end if
    // Page_fault_rate function is estimated by Belady's life-
time model
end procedure
```

placed in DRAM. This is based on our Page_fault_rate model estimated by the Belady's lifetime function. Specifically, if the estimated page fault ratio is not degraded significantly although the threshold M is increased by ΔM, the new threshold for the next epoch is set to $M + \Delta M$. Otherwise, if the estimated page fault ratio is significantly improved if the threshold M is decreased by ΔM, the new threshold is set to $M - \Delta M$.

C. ANDROID MEMORY MANAGEMENT AND CASTE

Android memory management is performed by LMK (low memory killer) and kswapd (kernel swap daemon). When free memory space is below the LMK-threshold, Android triggers LMK to make free memory. LMK selects the lowest priority application and kills it, thereby freeing all pages belonging to that application [41]. The priorities of applications are determined based on the time of the last use. Also, as the bottom core part of Android consists of the Linux kernel, kswapd can be activated when there is not enough free memory. Kswapd frees a certain number of page frames by flushing their contents to secondary storage. The default reclamation algorithm of kswapd maintains two page lists, the active list and the inactive list, and evicts pages not used recently in the inactive list [17]. Although the eviction granularity is different (i.e., application-level in LMK and page-level in kswapd), both LMK and kswapd commonly evict pages that are not used recently when there is not enough available memory. That is, they make use of online activities to predict which pages are not likely to be re-used in the near future.

Unlike the aforementioned replacement issue, Android memory management does not consider the placement issue. This is because it does not need to consider the placement of pages among free memory frames under the homogeneous DRAM architecture. However, as the memory system consists of heterogeneous media (i.e., DRAM and NVM), page placement becomes an important issue. From this point of view, we make the following observations that can be exploited in efficient page placement of Android memory. First, unlike desktop applications, memory references in smartphones are concentrated on pages of a certain logical addresses and this is consistent regardless of application types. Second, the memory reference behaviors of these hot pages do not change significantly as time progresses even after applications finish their launching. Based on these observations, we analyzed how to efficiently manage hybrid memory in smartphones and found that determining the type of memory (i.e. DRAM or NVM) for placing each page and preventing migrations between DRAM and NVM is efficient. This is differentiated from previous studies that mostly focus on the replacement algorithm, which determines an eviction victim based on the online behavior of page references, whereas we focus on the placement of pages between DRAM and NVM based on the prior knowledge analyzed from Android's memory reference behavior.

IV. EXPERIMENTAL RESULTS

We perform trace-driven simulations to assess the effectiveness of Caste. For a comparison purpose, we additionally simulate a scheme that adds a page to DRAM if a write operation on that page occurs first time and page migration between DRAM and NVM is allowed. We call this scheme *migration-allowed*, which is a typical scheme used in hybrid memory systems [11], [17], [27], [28]. In the experiments, the ratio of DRAM is varied to 10%, 15%, and 20% of the total main memory capacity. Accordingly, the NVM size is set to 90%, 85%, and 80% of the main memory, respectively. The page size is set to 4KB, which is the default setting of Android.

Fig. 14 shows the NVM write traffic of Caste relative to the migration-allowed scheme for each workload as the DRAM size is varied. As shown in the figure, Caste reduces the NVM write traffic significantly for a variety of workloads and DRAM sizes. Specifically, the reduced write traffic of Caste is an average of 42% and up to 87%. This is mainly because Caste eliminates NVM writes caused by data migration between DRAM and NVM, which accounts for the majority of total NVM write traffic. In Browser, Mxplayer, and Farmstory, the reduced write traffic by Caste is over 50% regardless of the DRAM size. The write traffic of Facebook is increased when the DRAM size is less than 20%. This is because the number of hot pages in Facebook is relatively large, and thus the DRAM size of 10% or 15% is not enough for maintaining these hot pages. Based on this observation, we need to set the size of DRAM required for Caste to at least 20% of the total memory capacity. From now on, we use the default DRAM size as 20%, which can minimize the refresh energy of DRAM without performance penalties. In such configurations, Caste can reduce NVM writes for all cases and the average reduction in write traffic is 62% in comparison with the migration-allowed scheme.

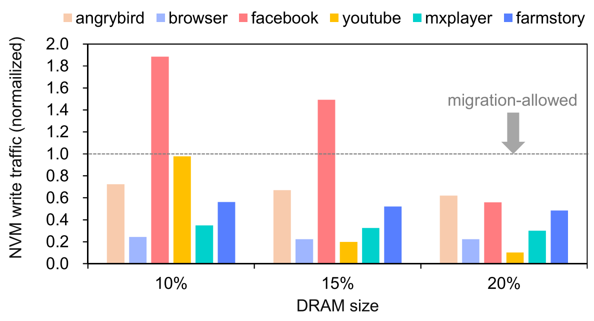


FIGURE 14. Relative NVM write traffic to Caste in comparison with the migration-allowed scheme.

Fig. 15 shows the total page fault ratio normalized to that of the migration-allowed scheme as the DRAM size and the workloads are varied. Although placing evicted pages to storage is likely to incur a lot of page faults, Fig. 15 shows that the page fault ratio is not increased when the DRAM size is set to 20%. Moreover, in some applications like Facebook and Youtube, the page fault ratio is even improved under the 20% DRAM size. This is because Caste places pages based on the exact write characteristics obtained by trace analysis. Moreover, Caste does not degrade the page fault ratio as it controls the number of incoming pages adaptively according to system situations. When the DRAM size is 10% or 15%, the page fault ratio is degraded because it is not sufficient to accommodate all hot pages in DRAM memory. Based on this result, we can also ensure that the DRAM size for Caste should be set to 20% of the total memory capacity.

Fig. 16 compares Caste and the migration-allowed scheme with respect to the lifetime of NVM. Although we do not limit the target of Caste to specific NVM media, we focus on our experiments under PCM as it is vulnerable to write operations, and its endurance cycle is shorter than other NVM media. Table 2 lists the DRAM and NVM characteristics we experimented for the lifetime and the energy issue of our experiments [17].

For the simulation of the NVM lifetime, we sequentially execute the six workloads repeatedly until the write limit

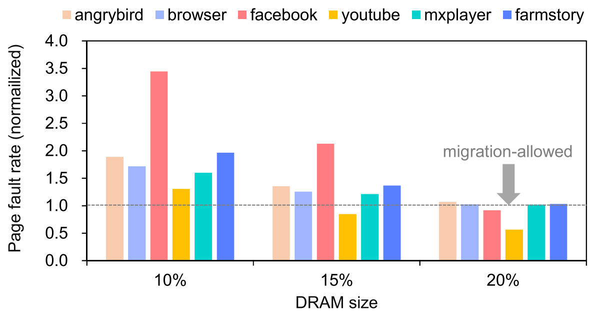


FIGURE 15. Relative Page fault ratio of Caste in comparison with the migration-allowed scheme.

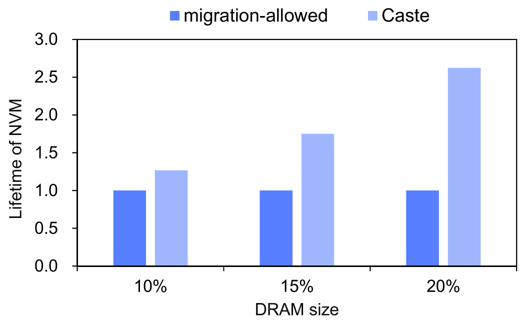


FIGURE 16. Lifetime of NVM.

TABLE 2. Access latency and power consumptions for DRAM and NVM.

	DRAM	NVM
Read latency	50 (ns)	50 (ns)
Write latency	50 (ns)	350 (ns)
Read energy	0.1 (nJ/bit)	0.2 (nJ/bit)
Write energy	0.1 (nJ/bit)	1.0 (nJ/bit)
Static power	1 (W/GB)	0.1 (W/GB)

of NVM assuming that all write requests are equally distributed to NVM. Equal distribution seems to be an unrealistic assumption, but previous studies also made such assumptions as wear-leveling techniques for evenly distributing write traffic to NVM have been devised at the architecture level [29]. Thus, we consider this to be a valid assumption. As shown in Fig. 16, Caste extends the lifetime of NVM by a large margin compared to the migration-allowed scheme. Specifically, the NVM lifetime is extended by 27%, 75%, and 162%, respectively, for the DRAM size of 10%, 15%, and 20%.

From now on, we investigate the overhead of our hybrid memory architecture by comparing the performance of Caste with the conventional memory architecture that uses only DRAM as main memory. Specifically, we compare the average memory access time of the hybrid memory architecture with Caste and the DRAM only architecture. For the DRAM only architecture, the size of DRAM is set to the sum of DRAM and NVM sizes in the hybrid memory architecture of Caste. We also compare Caste with the migration-allowed scheme under the same hybrid memory architecture. Fig. 17 shows the average memory access time

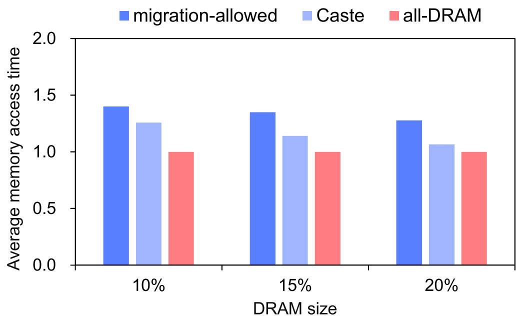


FIGURE 17. Comparison of average memory access time.

of Caste and migration-allowed under the hybrid memory architecture in comparison with that of the all-DRAM architecture. Note that the results are normalized to that of the all-DRAM architecture. As shown in the figure, the performance degradation of the hybrid memory architecture is widely varied from 6.5% to 40% according to the DRAM size and the management scheme. Specifically, Caste performs consistently better than the migration-allowed scheme. When the DRAM size is 20%, the performance degradation of the hybrid memory architecture with Caste is just 6.5% compared to the all-DRAM architecture. On the contrary, the migration-allowed scheme degrades the memory access time by 27.8% even though the DRAM size is as large as 20%. This indicates that the hybrid memory architecture may degrade smartphone memory performances, but it is feasible by judicious page allocation like Caste.

Let us now move on to the energy consumption. The memory energy consumption $Energy_{\mathrm{M}}$ is the sum of dynamic energy E_{dynamic} and static energy E_{static}, that is

$$Energy_{\mathrm{M}} = E_{\mathrm{dynamic}} + E_{\mathrm{static}} \tag{1}$$

The dynamic energy E_{dynamic} is the energy consumed during read or write operations [30], which can be calculated as

$$\begin{aligned} E_{\mathrm{dynamic}} = & \sum_{p \in \mathrm{DRAM}} \{r_p * E\_read_{\mathrm{DRAM}} + w_p * E\_write_{\mathrm{DRAM}}\} \\ & + \sum_{p \in \mathrm{NVM}} \{r_p * E\_read_{\mathrm{NVM}} + w_p * E\_write_{\mathrm{NVM}}\} \end{aligned} \tag{2}$$

where r_p and w_p are the number of read and write operations on page p, respectively, E_read_{DRAM} and E_write_{DRAM} are the read and write energies in DRAM, respectively, and E_read_{NVM} and E_write_{NVM} are the read and write energies in NVM, respectively.

The static energy E_{static} is the energy consumed constantly irrespective of any operations in memory [30], which can be calculated as

$$\begin{aligned} E_{\mathrm{static}} = & \{Unit\_power_{\mathrm{DRAM}} * size_{\mathrm{DRAM}} \\ & + Unit\_power_{\mathrm{NVM}} * size_{\mathrm{NVM}}\} * \tau \end{aligned} \tag{3}$$

where $Unit\_power_{\mathrm{DRAM}}$ and $Unit\_power_{\mathrm{NVM}}$ are the static power of DRAM and the static power of NVM, respectively, per capacity, $size_{\text{DRAM}}$ and $size_{\text{NVM}}$ are the size of DRAM and NVM, and τ is the total execution time.

Fig. 18 shows the energy consumption of Caste and migration-allowed normalized to that of the conventional all-DRAM architecture. As shown in the figure, the energy-saving effect of the hybrid memory architecture is consistent regardless of the DRAM size although small DRAM saves more energy. When the DRAM size becomes large, static energy required for DRAM refresh operations accounts for a large portion of energy consumption compared to the dynamic energy required for actual read/write operations. When the DRAM size is 20%, Caste reduces the energy consumption of the all-DRAM architecture by 68%. The migration-allowed scheme and Caste exhibit similar results but Caste consumes 2.2% less energy than the migration-allowed scheme. The energy consumption trend of these two schemes is similar as they adopt the same hybrid memory architecture, and thus static energy consumptions are identical. However, Caste reduces the energy consumption of the migration-allowed scheme by eliminating NVM write traffic that is responsible for a large portion of dynamic energy consumption.

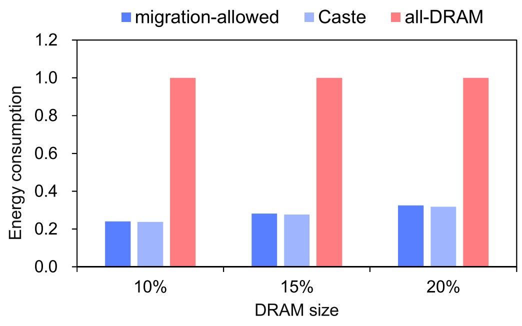


FIGURE 18. **Comparison of energy consumptions.**

V. RELATED WORKS

A. NON-VOLATILE MEMORY TECHNOLOGIES

Non-volatile memory (NVM) has been considered as a new memory medium to reduce the energy consumption of DRAM memory. Specifically, NVM allows the byte-addressability similar to DRAM, but it spends less energy than DRAM because NVM does not need to perform refresh of cells due to its non-volatile characteristics [31]. However, write operations on NVM are vulnerable in terms of the access latency and/or endurance cycles, and thus studies on NVM usually build hybrid memory architectures consisting of DRAM and NVM to solve the write vulnerability of NVM with a small amount of DRAM. For example, writing data in PCM is several times slower than reading, and the endurance cycle (i.e., maximum number of writes allowed for a cell) in PCM is in the range of 10^6 to 10^8, which is shorter than that of DRAM. Thus, studies on PCM as main memory adopt additional DRAM in order to reduce the number of write operations on PCM [11], [17].

Recently, there are attempts to focus on a pure-NVM structure for main memory. This is possible as extremely promising NVM technologies such as STT-MRAM (spin-transfer torque magnetic random access memory) and SOT-MRAM (spin orbit torque magnetic random access memory) relieve the write vulnerabilities of NVM, and there are lots of work discussing their superiority over DRAM and/or SRAM. Accordingly, studies are being conducted to configure the main memory with only NVM or even use NVM as a replacement of SRAM-based on-chip cache memory [32], [33].

In pure-NVM memory architectures, studies focus on the utilization of persistent data structures as main memory becomes non-volatile. That is, while traditional systems regard memory data as volatile and permanent data is only maintained at the storage layer, studies on pure-NVM memory systems are now being conducted to utilize the persistency of data at the memory layer. In such environments, the memory system is specially called Persistent Memory (PM) rather than NVM as it is differentiated from hybrid memory architectures that do not utilize the persistency of NVM.

As main memory becomes persistent, a cache line is expected to be the unit of data transfer between volatile and non-volatile devices. Thus, the failure-atomicity of write operations could be guaranteed in the granularity of cache lines. To do so, studies have been conducted to use in-memory data structures for NVM. For example, Cho *et al.* present Failure-atomic Byte-addressable R-tree (FBR-tree) that guarantees the crash consistency by making use of NVM [33]. They manage the order of memory writes and cacheline flush instructions, thereby eliminating the inconsistency of the FBR-tree. Lee *et al.* present the journaling transaction function at the main memory layer by utilizing NVM [34]. They also extend the persistent memory characteristics of NVM by allowing the original data location from flash storage to NVM memory [25].

Some recent studies focus on the on-chip cache memory architecture that replaces SRAM by NVM. Talebi *et al.* present an on-chip cache that consists of STT-MRAM instead of SRAM [32]. They suggest a replacement policy for STT-MRAM cache memory to improve the robustness of STT-MRAM against write failures.

B. HYBRID MEMORY TECHNOLOGIES

Mogul *et al.* suggest a hybrid memory architecture that locates read-intensive pages to NVM and write-intensive pages to DRAM [10]. Dhiman *et al.* also make use of hybrid memory consisting of NVM and DRAM, and try to balance the number of write operations on NVM as there are write endurance problems in NVM [11]. Lee *et al.* suggest a new page eviction algorithm for memory systems composed of NVM and DRAM [17]. Their algorithm classifies memory pages as hot pages and cold pages based on their write characteristics, and locates hot pages on DRAM and cold pages on NVM.

Narayan *et al.* suggest an object-level placement policy for DRAM and NVM hybrid memory [35]. Specifically, their policy aims at saving the power consumptions and improving the performances simultaneously by placing memory objects to appropriate memory media. Kannan *et al.* propose a memory management scheme for virtualized environments [36]. They present a hybrid memory management scheme that determines the appropriate memory media for placing a page of a guest machine, and does not allow the migration of pages. Instead, they present another scheme that allows the migration of memory pages and sharing pages between virtual machines for performance improvement.

Lin *et al.* make use of dynamic programming and greedy approximation for solving the memory mapping between heterogeneous memory systems [37]. Zhang *et al.* propose a task allocation scheme for heterogeneous memory [38]. Specifically, their scheme locates tasks one by one to NVM and investigates the schedulability of the tasks. This is performed repeatedly until the allocation of all tasks is finished.

Sun *et al.* present a hybrid memory management technique for AIoT (Artificial intelligence Internet of Things) systems [28]. Specifically, they reduce the energy consumption and optimize I/O performance of AIoT systems by migrating write-intensive data from NVM to DRAM. Also, they present an in-memory file system to reduce the number of data movements between memory and storage.

Liu *et al.* present a memory management framework called Memos [27], which manages DRAM and NVM hybrid memory over the hierarchy of cache, channels, and main memory. By monitoring the memory access patterns through TLB and main memory levels, they optimize the data placement in the memory hierarchy, improving memory performance and increasing the lifetime of NVM.

Wang *et al.* propose an SOT-MRAM based PIM (processing-in-memory) accelerator for neural network training [39]. Specifically, they present a floating point precision cell that features the balance between computation flexibility and memory density, improving the energy and area efficiency as well as performances.

Jin *et al.* present an on-chip cache management scheme that considers the asymmetrical penalty of memory access to DRAM and NVM [40]. They argue that hit ratio is not an effective metric for hybrid memory systems, and propose MALRU (Miss-penalty Aware LRU) to preserve high-latency NVM blocks preferentially in the last level cache.

VI. CONCLUSION

In this article, we presented an efficient memory management scheme, called Caste, for DRAM and NVM hybrid memory architectures in a smartphone. Unlike previous studies, Caste analyzes write reference characteristics of Android applications precisely and determines the priorities of pages for DRAM placement. Also, by preventing online relocation of pages between DRAM and NVM, Caste completely eliminates unnecessary NVM writes that account for 32-42% of the total write traffic. This article also quantified the number of hot pages that should be allocated to DRAM under the given system situations based on the page fault ratio model we devised. Experiment results with real smartphone workloads showed that Caste reduces the NVM write traffic by 42% on average and up to 87%.

REFERENCES

[1] S. Bae, H. Song, C. Min, J. Kim, and Y. I. Eom, "EIMOS: Enhancing interactivity in mobile operating systems," in *Proc. Int. Conf. Comput. Sci. Appl.*, in Lecture Notes in Computer Science, vol. 7335, 2012, pp. 238–247.

[2] I. Bisio, F. Lavagetto, M. Marchese, and A. Sciarrone, "GPS/HPS- and Wi-Fi fingerprint-based location recognition for check-in applications over smartphones in cloud-based LBSs," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 858–869, Jun. 2013.

[3] F. Huang, X. Li, S. Zhang, J. Zhang, J. Chen, and Z. Zhai, "Overlapping community detection for multimedia social networks," *IEEE Trans. Multimedia*, vol. 19, no. 8, pp. 1881–1893, Aug. 2017.

[4] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, and C.-H. Hsu, "On the quality of service of cloud gaming systems," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 480–495, Feb. 2014.

[5] N. Islam and R. Want, "Smartphones: Past, present, and future," *IEEE Pervas. Comput.*, vol. 13, no. 4, pp. 89–92, Oct. 2014.

[6] *Google Pixel 4a*. Accessed: Apr. 1, 2021. [Online]. Available: https://store.google.com/?srp=/product/pixel_4a_specs

[7] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. USENIX Annu. Tech. Conf.*, 2010, p. 21.

[8] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. ACM ASPLOS*, 2011, pp. 213–224.

[9] S. Eilert, M. Leinwander, and G. Crisenza, "Phase change memory: A new memory technology to enable new memory usage models," in *Proc. 1st IEEE Int. Memory Workshop (IMW)*, May 2009, pp. 1–2.

[10] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Proc. HotOS*, 2009, pp. 4–14.

[11] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. 46th ACM/IEEE Design Automat. Conf. (DAC)*, Jul. 2009, pp. 559–664.

[12] International Technology Roadmap for Semiconductors, Semiconductor Industry Association, Washington, DC, USA. (2007). *Emerging Research Devices*. [Online]. Available: https://www.semiconductors.org/resources/2007-international-technology-roadmap-for-semiconductors-itrs/

[13] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 14–23.

[14] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 24–33.

[15] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 2–13.

[16] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proc. 37th Int. Symp. Comput. Archit. (ISCA)*, 2010, pp. 153–162.

[17] S. Lee, H. Bahn, and S. H. Noh, "CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2187–2200, Sep. 2014.

[18] L. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proc. ACM ICS*, Tucson, AZ, USA, May 2011, pp. 85–95.

[19] H. Lee, H. Bahn, and K. G. Shin, "Page replacement for write references in NAND flash based virtual memory systems," *J. Comput. Sci. Eng.*, vol. 8, no. 3, pp. 157–172, Sep. 2014.

[20] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 2, pp. 44–66, 2003.

[21] J. Choi, S. Cho, S. Noh, S. Lyul, and Y. Cho, "Analytical prediction of buffer hit ratios," *Electron. Lett.*, vol. 36, no. 1, pp. 10–11, 2000.

[22] S. Bock, B. Childers, R. Melhem, and D. Mosse, "Concurrent page migration for mobile systems with OS-managed hybrid memory," in *Proc. 11th ACM Conf. Comput. Frontiers*, 2014, pp. 1–10.

[23] E. Lee and H. Bahn, ''Caching strategies for high-performance storage media,'' *ACM Trans. Storage*, vol. 10, no. 3, pp. 1–22, Jul. 2014.

[24] E. Lee, H. Bahn, S. Yoo, and S. H. Noh, ''Empirical study of NVM storage: An operating system's perspective and implications,'' in *Proc. IEEE 22nd Int. Symp. Modelling, Anal. Simulation Comput. Telecommun. Syst.*, Sep. 2014, pp. 405–410.

[25] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh, ''Reducing write amplification of flash storage through cooperative data management with NVM,'' *ACM Trans. Storage*, vol. 13, no. 2, pp. 1–13, Jun. 2017.

[26] S. Lee and H. Bahn, ''Characterizing memory references for smartphone applications and its implications,'' *J. Semicond. Technol. Sci.*, vol. 15, no. 2, pp. 223–231, Apr. 2015.

[27] L. Liu, S. Yang, L. Peng, and X. Li, ''Hierarchical hybrid memory management in OS for tiered memory systems,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2223–2236, Oct. 2019.

[28] H. Sun, L. Chen, X. Hao, C. Liu, and M. Ni, ''An energy-efficient and fast scheme for hybrid storage class memory in an AIoT terminal system,'' *Electronics*, vol. 9, no. 6, p. 1013, Jun. 2020.

[29] L. Yavits, L. Orosa, S. Mahar, J. D. Ferreira, M. Erez, R. Ginosar, and O. Mutlu, ''WoLFRaM: Enhancing wear-leveling and fault tolerance in resistive memories using programmable address decoders,'' in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 187–196.

[30] S. Yoo, Y. Jo, and H. Bahn, ''Integrated scheduling of real-time and interactive tasks for configurable industrial systems,'' *IEEE Trans. Ind. Informat.*, early access, Mar. 22, 2021, doi: 10.1109/TII.2021.3067714.

[31] E. Lee, H. Kang, H. Bahn, and K. G. Shin, ''Eliminating periodic flush overhead of file I/O with non-volatile buffer cache,'' *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1145–1157, Apr. 2016.

[32] M. Talebi, A. Salahvarzi, A. M. H. Monazzah, K. Skadron, and M. Fazeli, ''ROCKY: A robust hybrid on-chip memory kit for the processors with STT-MRAM cache technology,'' *IEEE Trans. Comput.*, early access, Nov. 26, 2020, doi: 10.1109/TC.2020.3040152.

[33] S. Cho, W. Kim, S. Oh, C. Kim, K. Koh, and B. Nam, ''Failure-atomic byte-addressable R-tree for persistent memory,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 601–614, Mar. 2021.

[34] E. Lee, H. Bahn, and S. Noh, ''Unioning of the buffer cache and journaling layers with non-volatile memory,'' in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2013, pp. 73–80.

[35] A. Narayan, T. Zhang, S. Aga, S. Narayanasamy, and A. Coskun, ''MOCA: Memory object classification and allocation in heterogeneous memory systems,'' in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 326–335.

[36] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, ''HeteroOS—OS design for heterogeneous memory management in datacenter,'' in *Proc. ISCA*, 2017, pp. 521–534.

[37] Y. Lin, N. Guan, and Q. Deng, ''Allocation and scheduling of real-time tasks with volatile/non-volatile hybrid memory systems,'' in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, Aug. 2015, pp. 1–6.

[38] Z. Zhang, P. Liu, L. Ju, and Z. Jia, ''Energy efficient real-time task scheduling for embedded systems with hybrid main memory,'' in *Proc. RTCSA*, 2014, pp. 1–10.

[39] H. Wang, Y. Zhao, C. Li, Y. Wang, and Y. Lin, ''A new MRAM-based process in-memory accelerator for efficient neural network training with floating point precision,'' in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[40] H. Jin, D. Chen, H. Liu, X. Liao, R. Guo, and Y. Zhang, ''Miss penalty aware cache replacement for hybrid memory systems,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4669–4682, Dec. 2020.

[41] J. Kim and H. Bahn, ''Maintaining application context of smartphones by selectively supporting swap and kill,'' *IEEE Access*, vol. 8, pp. 85140–85153, 2020.

SOYOON LEE received the B.S., M.S., and Ph.D. degrees in computer science from Ewha University, Seoul, South Korea, in 2004, 2006, and 2011, respectively. She is currently a Research Professor of computer science and engineering with Ewha University. Her research interests include emerging memory and storage systems, operating systems, and embedded systems.

HYOKYUNG BAHN (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, in 1997, 1999, and 2002, respectively.

He is currently a Full Professor of computer science and engineering with Ewha University, Seoul, South Korea. He has published more than 100 papers in leading conferences and journals including USENIX FAST, IEEE Transactions on Computers, IEEE Transactions on Knowledge and Data Engineering, and *ACM Transactions on Storage*. His research interests include operating systems, caching algorithms, storage systems, embedded systems, system optimizations, and real-time systems. He received the Best Paper Awards from the USENIX Conference on File and Storage Technologies, in 2013.