IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 19, 2018, accepted September 14, 2018, date of publication October 1, 2018, date of current version October 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2872794

A Hybrid Swapping Scheme Based On Per-Process Reclaim for Performance Improvement of Android Smartphones (August 2018)

JUNYEONG HAN[1], SUNGEUN KIM[1], SUNGYOUNG LEE[1], JAEHWAN LEE[2], AND SUNG JO KIM[2]

[1]LG Electronics, Seoul 07336, South Korea
[2]School of Software, Chung-Ang University, Seoul 06974, South Korea

Corresponding author: Sung Jo Kim (sjkim@cau.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2016R1D1A1B03931004 and in part by the Chung-Ang University Research Scholarship Grants in 2015.

ABSTRACT As a way to increase the actual main memory capacity of Android smartphones, most of them make use of zRAM swapping, but it has limitation in increasing its capacity since it utilizes main memory. Unfortunately, they cannot use secondary storage as a swap space due to the long response time and wear-out problem. In this paper, we propose a hybrid swapping scheme based on per-process reclaim that supports both secondary-storage swapping and zRAM swapping. It attempts to swap out all the pages in the working set of a process to a zRAM swap space rather than killing the process selected by a low-memory killer, and to swap out the least recently used pages into a secondary storage swap space. The main reason being is that frequently swap- in/out pages use the zRAM swap space while less frequently swap-in/out pages use the secondary storage swap space, in order to reduce the page operation cost. Our scheme resolves both the response time and wear-out problems of secondary-storage swapping and zSWAP, and overcomes the size limitation of the zRAM swap space. According to performance measurements, it also increased the extension ratio of main memory by 15 $\sim$ 17% and 6 $\sim$ 17% and reduced the page operation cost by 9 $\sim$ 22% and 18 $\sim$ 28%, respectively, compared with zRAM swapping and zSWAP.

INDEX TERMS zRAM swapping, zSWAP, hybrid swapping, extension of main memory, wear-out problem.

I. INTRODUCTION

Recently, as smartphone hardware performance has improved dramatically, the performance of smartphones has become similar to that of personal computers in terms of hardware [1], [2]. This allows smartphone users to install dozens or hundreds of high-demand applications similar to personal computers.

Operating systems like Unix/Linux and Windows rely on a swapping scheme which utilizes secondary storage as main memory to extend its actual size. The secondary-storage swapping scheme may extend the size, but may increase system response time. Secondary storage may be inappropriate for a swap space, especially in a mobile environment, due to slow system response time. Either eMMC [3] or UFS [4] based on flash memory with a wear-out problem [5] has been used as secondary storage; however, it cannot be used as a swap space in a mobile environment because its lifetime is shortened due to frequent swap operations. Since secondary-storage swapping cannot be utilized in the mobile environment, an application running in the background will be terminated unlike the desktop environment. For example, while a user is playing a mobile game on the Android smartphone, if he or she takes an incoming phone call, the game may be terminated unintentionally.

To resolve these problems, this paper proposes a hybrid swapping scheme based on per-process reclaim suitable for mobile environment. Hybrid swapping addresses both the slowdown in system response time and the flash memory wear-out problem by using both secondary-storage swapping and zRAM swapping [6] to extend main memory.

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Hybrid swapping swaps out all of the pages that belong to an application to be killed into zRAM swap space rather than killing it so that it can be restarted quickly later. Our scheme resolves both response time and wear-out problems of secondary-storage swapping and zSWAP, and overcomes the size limitation of zRAM swap space.

The rest of the paper is organized as follows: In Section 2, we compare existing schemes proposed for extending main memory by using swap space in mobile environments. In Section 3, we propose a hybrid swapping scheme that improves the extension ratio of main memory and system performance, and solves the wear-out problem. In Section 4, we compare the extension ratio of main memory and the page operation cost of hybrid swapping with those of zRAM swapping, respectively, and show that our scheme has virtually no wear-out problem. Finally, Section 5 concludes this paper and discusses future research directions.

II. RELATED WORKS

Since Android uses virtual memory and paging, it is not necessary for program code and data to be simultaneously loaded into main memory during program execution. A portion or all of unreferenced program code and data will be saved on secondary storage during program execution. The secondary-storage swapping scheme utilizes secondary storage as main memory in order to increase the effective main memory size by using swap space on secondary storage.

The zRAM swapping scheme utilizes a certain area of RAM as storage space in the same manner as the compressed RAM block device [7]. However, it is possible to store the amount of data larger than the allocated RAM size by compressing the data. According to the zRAM compression ratio of Android smartphones using zRAM, zRAM saves approximately a half of RAM space on average [6]. Various Android smartphones have adopted zRAM swapping using these zRAMs as swap space.

If swap-in and/or swap-out occur frequently when swap space is used, the overall performance of the system should degrade. To alleviate this performance degradation, swapping schemes usually utilize a swap cache located between main memory and secondary storage. Compressed cache swapping schemes such as zSWAP [8] and zRAM DM-Cache [9] provide high-capacity compressed swap caches. Since zSWAP uses zRAM as a swap cache, system performance degradation can be reduced significantly compared to secondary-storage swapping due to the increased cache hit ratio by increasing the amount of swap cache while using only a small amount of main memory. DM-Cache is a component of the Linux kernel's device mapper [10] used for mapping a block device to a high-level virtual block device. For example, DM-Cache improves read performance of a block device by mapping a block device with a slow operation speed, such as a hard disk or a flash memory, to a RAM block device. The zRAM DM-Cache swapping scheme increases the swap-in/out operation speed of swap space of the secondary storage by mapping its swap space to the RAM block device through zRAM.

TABLE 1. Comparison of swapping schemes.

Swapping Scheme	Storage Type of Swap Space	Pros	Cons
Secondary-storage swapping	Flash Memory or Hard Disk	The cost per bit of swap space is cheap [11].	The swap space has a wear-out problem & low speed [3], [4], [13].
zRAM Swapping	zRAM	The swap space has no wear-out problem & high speed [12].	The cost per bit of swap space is expensive [11].
Compressed Cache Swapping (zSWAP, zRAM DM-Cache)	Flash Memory or Hard Disk	The cost per bit of swap space is cheap [11].	As swap space becomes larger, both performance degradation & wear-out problem get worse [3], [4].

As shown in Table 1, pros and cons of secondary-storage swapping, zRAM swapping, and compressed cache swapping differ, depending on the types of the storage used for swap space. Comparing with secondary-storage swapping and zRAM swapping, compressed cache swapping has lower system performance and wear-out problem since the cache hit rate decreases as swap space becomes larger. In a mobile environment, system performance should not be degraded due to paging operations since the response time is very crucial. Therefore, in the mobile environment, only considered are the zRAM swapping scheme and the compressed cache swapping scheme; the former has minimal performance degradation while the extension ratio of main memory is limited, and the latter has limitation in increasing the swap space size.

Liu *et al.* [14] proposed a scheme called HybridSwap, which integrates a hard disk with a solid state disk (SSD) for virtual memory management in order to extend the main memory size; however, this scheme is inappropriate to mobile environments because Android smartphones are not equipped either with SSDs or with hard disks. Chae *et al.* [15] suggested CloudSwap as a swap mechanism for mobile devices by utilizing remote storage as a swap space. Since accessing remote storage increases power consumption dramatically, it is inappropriate to apply this scheme to commercial smartphones. Zhang *et al.* [16] proposed MemFlex as a shared memory swapper in order to enhance swapping performance in virtualized systems. Zhang *et al.* [17] also proposed MemLego, which relies on a shared memory based swapping facility called MemSwap in virtualized systems. These schemes cannot be used for Android smartphones because they are designed especially for high-performance server environments that support virtual machines. Another schemes [18], [19] utilizing fast and byte-addressable non-volatile memory (NVM) as a swap space have been proposed. Because NVM is more expensive than DRAM and requires extra mounting space on

smartphone PCBs, and there is no affordable way to embed NVM in multi-chip package for now, it is more appropriate to increase the size of DRAM than to adopt NVM. Finally, Zhu *et al*. [20] proposed SwapBench to evaluate various swapping schemes, especially in terms of two performance metrics such as application launch and switch on Android smartphones. We enhance this framework to overcome such limitation that it cannot accurately evaluate swapping performance when it repeatedly appraises 10 or more applications based on different user scenarios.

III. HYBRID SWAPPING

We propose hybrid swapping that uses both secondary-storage swapping and zRAM swapping. In our scheme, pages are swapped out from main memory to either swap space in secondary storage or zRAM swap space, depending on their reference patterns. To be specific, infrequently swap-in/out pages are swapped out to secondary storage swap space, while frequently swap-in/out pages are swapped out to zRAM swap space using per-process reclaim [21] to avoid killing processes for getting free memory. In this way, hybrid swapping addresses the performance degradation and wear-out problems that occur with secondary-storage swapping and compressed cache swapping, while extending the available main memory size.

After defining the extension ratio of main memory, the cost of page operation, and no wear-out period of the hybrid swapping scheme in Section A, we calculate the maximum extension ratio of main memory and the cost of page operation for each swapping scheme in Section B, and describe how to implement hybrid swapping in Section C.

A. RELATED DEFINITIONS

When a program is invoked, its code and data are loaded into main memory from secondary storage. Read operations are executed when loading program code and data into main memory. During program execution, one or more pages are allocated in main memory for data processing, and are managed as "anonymous pages."

If a page of an executable and data file called "file page" not in main memory is referenced when there is no free page frame, the file page is loaded into main memory after reclaiming one or more page frames that are not referenced for a certain period of time. In addition, when a new anonymous page is to be allocated when no page frames are available, it is also allocated in main memory after reclaiming one or more page frames. A memory management system will discard file pages from main memory in the absence of swap space, while anonymous pages remain "pinned" in main memory. On the other hand, if there is swap space available, it logically extends the size of main memory by storing the anonymous pages in swap space. To summarize, the page operation requires read and discard operations of file pages, and swap-in/out operations of anonymous pages.

The extension ratio of main memory is defined as the ratio of the size of the logical main memory (physical main memory + swap space) to the size of the physical main memory. Because the overhead for discarding file pages during page operations is negligible, the page operation cost can be defined as the sum of the costs necessary for reading and swapping in/out file pages.

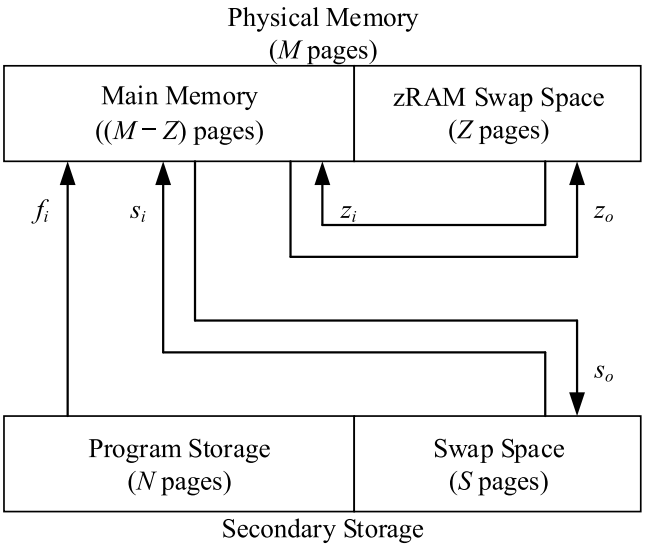

FIGURE 1. Hybrid swapping.

Fig. 1 shows how hybrid swapping works. Parameters related to hybrid swapping are defined in Table 2.

TABLE 2. Definition of parameters related to swapping system.

Symbol	Definition
N	the number of pages allocated to user programs
M	the number of page frames in the main memory
S	the number of pages in secondary storage swap space
Z	the number of compressed pages allocated to zRAM swap space ($Z < M$)
U	the number of uncompressed pages swapped out to zRAM swap space ($Z \leq U$)
C	U/Z
f_i	read operation of a file page for loading it into the main memory
s_i	swap-in operation of a page from secondary storage
s_o	swap-out operation of a page to secondary storage
z_i	swap-in operation of a page from zRAM
z_o	swap-out operation of a page to zRAM
f_{it}	the time taken to execute operation f_i
s_{it}	the time taken to execute operation s_i
s_{ot}	the time taken to execute operation s_o
z_{it}	the time taken to execute operation z_i
z_{ot}	the time taken to execute operation z_o

Prior to initiating execution of a program, its code to be executed and data are read from secondary storage in the hybrid swapping system of Fig. 1. As time elapses, the number of programs that are running simultaneously increases, the swap space will be eventually exhausted. In order for a

new program to run when both main memory and swap space being exhausted, one or more currently running programs must be killed.

When multiple programs are running simultaneously in a hybrid swapping system, if a page to be referenced is not in main memory and not swapped out to secondary storage swap space or zRAM swap space, it will be loaded into the main memory from the program repository on secondary storage using f_i. On the other hand, if the referenced page is swapped out to secondary storage swap space, it is swapped in main memory through s_i. If the referenced page is swapped out to the zRAM swap space, it is swapped in the main memory through z_i. If no free page frame is available in the main memory at this time, swapped out is an anonymous page that is not referenced for the longest time through s_o or z_o to the secondary storage swap space or zRAM swap space, respectively.

Since the number of pages referenced increases while executing a program, the cost of page operations can be calculated using Equation (1) over the time span t from t_1 to t_n when the number of pages referenced by f_i, s_i, s_o, z_i, and z_o is denoted by $\Delta\mathrm{FI}(t_k)$, $\Delta\mathrm{SI}(t_k)$, $\Delta\mathrm{SO}(t_k)$, $\Delta\mathrm{ZI}(t_k)$, and $\Delta\mathrm{ZO}(t_k)$, respectively:

$$O(t) = \sum_{t=t_1}^{t_n} [f_{ic} \cdot \Delta\mathrm{FI}(t_k) + s_{ic} \cdot \Delta\mathrm{SI}(t_k) + s_{oc} \cdot \Delta\mathrm{SO}(t_k) + z_{ic} \cdot \Delta\mathrm{ZI}(t_k) + z_{oc} \cdot \Delta\mathrm{ZO}(t_k)], \quad \text{where } k = 1, 2, \ldots, n \tag{1}$$

If write operations are repeated more than a certain number of times at a specific address in the flash memory, NAND flash wear-out may occur. In case of flash memory-based secondary storage such as eMMC or UFS, an internal flash translation layer (FTL) is used to mitigate its wear-out by changing the address to be written in the flash memory. In order to ensure that NAND flash wear-out does not occur, we need to know the maximum number of pages that can be written to eMMC or UFS devices, which guarantees no wear-out pages, called *GP*. *GP* can be defined by Equation (2), where $\Delta\mathrm{FO}(t_k)$ is the number of pages written by Android file output operations

$$\mathrm{GP} = \sum_{t=t_1}^{t_n} [\Delta\mathrm{SO}(t_k) + \Delta\mathrm{FO}(t_k)], \quad \text{where } k = 1, 2, \ldots, n \tag{2}$$

B. MAXIMUM EXTENSION RATIO OF MAIN MEMORY AND COST OF PAGE OPERATIONS FOR SWAPPING SYSTEMS

In this section, we calculate the maximum extension ratio of main memory and page operation cost of each swapping system through formulas and compare them with each other.

f_{it}, s_{it}, s_{ot}, z_{it}, and z_{ot} depend on the speeds of I/O operations between main memory and secondary storage, and the CPU speed. According to performance metrics of LPDDR2 memory and eMMC v5.1 which have been used for main memory and secondary storage by the Qualcomm MSM8952 chipset [22], the speeds of page read and write operations are 20MB/s and 10MB/s, respectively, for the secondary storage, and are 100MB/s and 50MB/s, respectively, for the zRAM. Since the Linux's default page size is 4KB, f_{it}, s_{it}, s_{ot}, z_{it}, and z_{ot} can be calculated by using Equation (3) $\sim$ (6), respectively.

$$f_{it} = s_{it} = \frac{1s}{20MB} = \frac{1s}{5120pages} = \frac{0.1953\mathrm{ms}}{\mathrm{page}} \tag{3}$$

$$s_{ot} = \frac{1s}{10MB} = \frac{1s}{2560pages} = \frac{0.3906\mathrm{ms}}{\mathrm{page}} \tag{4}$$

$$z_{it} = \frac{1s}{100MB} = \frac{1s}{25600pages} = \frac{0.0390\mathrm{ms}}{\mathrm{page}} \tag{5}$$

$$z_{ot} = \frac{1s}{50MB} = \frac{1s}{12800pages} = \frac{0.0781\mathrm{ms}}{\mathrm{page}} \tag{6}$$

We now derive the maximum extension ratio of main memory(ME) and the cost of page operation at time t($O(t)$) for each type of swapping schemes as below:

1) NO SWAPPING

$$\mathrm{ME} = \frac{\mathrm{M}}{\mathrm{M}} = 1 \tag{7}$$

$$O(t) = \sum_{t=t_1}^{t_n} \left[0.1953\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{FI}(t_k)\right], \quad \text{where } k = 1, 2, \ldots, n \tag{8}$$

2) SECONDARY-STORAGE SWAPPING

$$\mathrm{ME} = \frac{\mathrm{M}+\mathrm{S}}{\mathrm{M}} \tag{9}$$

$$O(t) = \sum_{t=t_1}^{t_n} \left[0.1953\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{FI}(t_k) + 0.1953\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{SI}(t_k) + 0.3906\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{SO}(t_k)\right], \quad \text{where } k = 1, 2, \ldots, n \tag{10}$$

3) ZRAM SWAPPING

Assuming that the average compression ratio(C) of zRAM is 2 [6], ME and $O(t)$ can be given by Equation (11) and Equation (12), respectively:

$$\mathrm{ME} = \frac{\mathrm{M}+\mathrm{Z}\cdot(\mathrm{C}-1)}{\mathrm{M}} = \frac{\mathrm{M}+\mathrm{Z}}{\mathrm{M}} \tag{11}$$

$$O(t) = \sum_{t=t_1}^{t_n} \left[0.1953\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{FI}(t_k) + 0.0390\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{ZI}(t_k) + 0.0781\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{ZO}(t_k)\right], \quad \text{where } k = 1, 2, \ldots, n \tag{12}$$

4) ZSWAP

$$\mathrm{ME} = \frac{\mathrm{M}+\mathrm{S}}{\mathrm{M}} \tag{13}$$

$$O(t) = \sum_{t=t_1}^{t_n} \left[0.1953\left(\frac{\mathrm{ms}}{\mathrm{page}}\right) \cdot \Delta\mathrm{FI}(t_k)\right.$$

$$+0.0390\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{ZI}\,(t_k)+0.0781\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{ZO}\,(t_k)+(0.1953+0.0781)\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{SI}\,(t_k)+\,(0.0390+0.3906)\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{SO}\,(t_k)\Bigg],\quad \text{where } k=1,2,\ldots,n \tag{14}$$

If the number of swap pages in zSWAP is smaller than U, the page operation cost of zRAM swapping and zSWAP are equal because there are no s_i, s_o operations. However, if it becomes greater than U, the cache hit rate gets worse and the page operation cost becomes higher than those of zRAM swapping and secondary-storage swapping.

5) HYBRID SWAPPING

$$\text{ME}=\frac{M+S+Z}{M} \tag{15}$$

$$O\,(t)=\sum_{t=t_1}^{t_n}\Bigg[0.1953\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{FI}\,(t_k)+0.1953\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{SI}\,(t_k)+0.3906\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{SO}\,(t_k)+0.0390\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{ZI}\,(t_k)+0.0781\left(\frac{\text{ms}}{\text{page}}\right)\cdot\Delta\text{ZO}\,(t_k)\Bigg],\quad \text{where } k=1,2,\ldots,n \tag{16}$$

When both secondary-storage swapping and zRAM swapping are utilized, since $M < M+S < M+S+Z$ or $M < M+Z < M+S+Z$, the hybrid swapping scheme provides the largest main memory extension ratio. As can be seen from (12) < (8) < (10), the secondary-storage swapping increase the page operation costs, while the zRAM swapping reduces them. Therefore, if both secondary-storage swapping and zRAM swapping are utilized simultaneously, the page operation cost can be reduced more compared to utilizing only secondary-storage swapping, but becomes higher than the case where only zRAM swapping is utilized.

C. IMPLEMENTATION OF HYBRID SWAPPING

The terms $\sum_{t=t_1}^{t_n}\Delta\text{FI}(t_k)$, $\sum_{t=t_1}^{t_n}\Delta\text{SI}(t_k)$, $\sum_{t=t_1}^{t_n}\Delta\text{SO}(t_k)$, $\sum_{t=t_1}^{t_n}\Delta\text{ZI}(t_k)$, and $\sum_{t=t_1}^{t_n}\Delta\text{ZO}(t_k)$ are the number of pages referenced by each operation in (16) of which values increase over time. If frequently referenced pages are swapped out to zRAM swap space, the extension ratio of main memory increases along with lower page operation cost. On the other hand, in order not to increase the page operation cost as long as possible, least frequently references pages are swapped out to secondary storage swap space. It also makes it possible to increases the guaranteed non-wear-out time because it decreases $\sum_{t=t_1}^{t_n}\Delta\text{SO}(t_k)$. Therefore, hybrid swapping can be implemented by swapping out frequently used pages to zRAM swap space and the least frequently used pages to secondary storage swap space.

Fig. 2 shows a block diagram, which shows how hybrid swapping works. As shown in its upper left corner, there are two memory reclamation techniques: direct page reclaim and *kernel swap daemon*(*kswapd*). When there is no enough physical memory to allocate, the former technique will be useful. On the other hand, in order to ensure as much free memory as its watermark set by Linux, the latter is periodically invoked.

As shown on the left side of Fig. 2, function *shrink_zone()*[23], which is the main entry point for memory reclamation in Linux, relocates pages in the *active list* to the *inactive list*, using the LRU algorithm in function *shrink_active_list()*[23]. The *active list* is a group of the most recently referenced pages, and the *inactive list* is a group of pages which have not been recently referenced. Then, function *shrink_inactive_list()*[23] sends pages in the *inactive list* to the *shrink list* using the LRU algorithm. Finally, while function *shrink_page_list()*[23] reclaims pages in the *shrink list*, file pages are discarded while anonymous pages are swapped out to swap space.

Linux relies on the LRU algorithm to select a page to be swapped out to secondary storage, since it will not be swapped in or out frequently. This will also increase the guaranteed non-wear-out time because it reduces the number of read and write operations from/to secondary storage. Swap-in/out of pages by periodic invocation of *kswapd* may be delayed because it is less time critical than a case when a page should be swapped out due to memory reclaims directly by memory allocator. Therefore, in hybrid swapping, as shown in ① of Fig. 2, pages chosen by the LRU algorithm for swapping are to be swapped out to the secondary storage swap space by *kswapd*. However, since direct memory reclaims are performed in a very urgent situation where there is no memory to allocate immediately, delay caused by swap-out cannot be tolerated. Therefore, in the situation where direct memory reclaim is required, pages in the *shrink list* [23] are swapped out to zRAM swap space, as shown in ② of Fig. 2.

Function *shrink_slab()* [23] is invoked when free memory cannot be reclaimed anymore through function *shrink_zone()*[23]. If the reclamation fails, free memory can be obtained by killing a process that the *Linux out-of-memory(OOM) killer* [24] regards as being unimportant.

Usually in Android, one process runs in foreground while the others run in the background. Users repeatedly execute several to dozens of preferred processes alternatively. Since each process utilizes tens to hundreds of Mbytes of memory, it is necessary to allocate and deallocate several tens to hundreds of Mbytes of memory when running new process. Because of this usage characteristics, Android invokes *Android Low Memory Killer(LMK)* [25] instead of *OOM killer* to kill a process in order to reclaim free memory space.

As shown in the right part of Fig. 2, *LMK* is executed as a call back function of function *shrink_slab()* and selects the process with the largest resident set size(*RSS*) among a group of processes residing in foreground. When a user initiates a new process, the time taken to invoke it must be very short, since the response times that the user experiences matter.

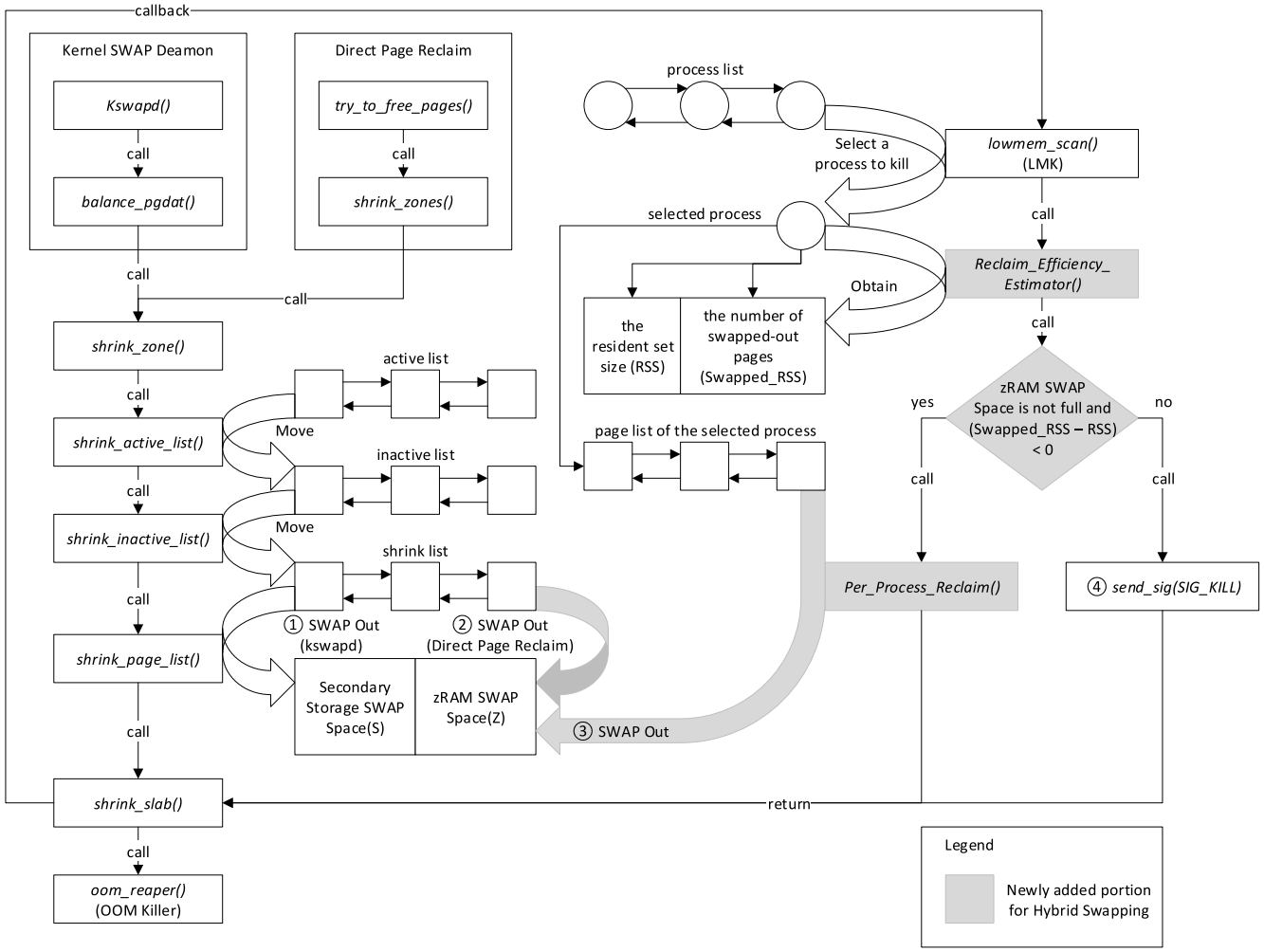

FIGURE 2. Block diagram for hybrid swapping.

Since the process selected by *LMK* is a process that has been running in foreground, it is more likely to be executed in foreground again later by the user, so it may be better to swap out and swap in later rather than killing it immediately.

Two conditions must be satisfied in order to free up main memory by swapping out all anonymous pages of a process instead of killing the process itself. One condition is that it should not take long to swap out the pages because available memory is insufficient for the normal operation of the process when *LMK* is triggered. The other is that it should be possible to ensure tens or hundreds of Mbytes of free memory by swapping out the process. In order to satisfy these conditions, hybrid swapping swaps out all pages belonging to the process selected by *LMK* to zRAM swap space as shown in ő of Fig. 2 so that a large amount of free memory can be ensured as quickly as possible.

A program's working set [26] $W(t, T)$ at time t is the set of distinct pages that have been recently referenced in the last T virtual time units. All the pages in$W(t, T)$ are loaded together into main memory, and also swapped out together. Because Android usually runs one process in foreground, processes in background selected by *LMK* will not run for a while. Therefore, the cost of page operations can be reduced because pages in $W(t, T)$ belonging to the processes in background will not be swapped in for a while once they swapped out.

As the number of processes to be executed increases, the total size of the pages used by the processes will exceed the size of (the main memory + secondary storage swap space + zRAM swap space), in which case some processes must be killed. In hybrid swapping, pages belonging to processes to be killed by *LMK* are swapped out to zRAM swap space. The zRAM swap space may become full due to these swap-outs. In order to resolve this problem, *LMK* can swap out pages of a process to the secondary storage swap space or kill the process itself. If it is killed, the number of file pages to be read from the secondary storage may increase because it may re-invoked later by user. However, since the cost of read operations for these pages is less than the cost of swap-in/out to the secondary storage swap space, hybrid swapping

kills the process rather than swapping out even if there is free swap space on the secondary storage after zRAM swap space becomes full.

As shown in the legend of Fig. 2, in order to implement hybrid swapping, function *Per_Process_Reclaim()* [21] is added to general memory reclaim procedure of Linux, and function *Reclaim_Efficiency_Estimator()* is also added to *LMK* to determine whether to kill a process or swap it out to zRAM swap space.

Function *Per_Process_Reclaim()* accepts two parameters such as structure *task_struct* and the number of pages to be reclaimed using function *reclaim_task_file_anon()*. This function retrieves file pages and anonymous pages of the process specified by the first argument, *task_struct*, reclaims as many file and anonymous pages as the number specified by the second parameter, and returns reclaimed page number. Note that file pages are discarded while anonymous pages are swapped out to zRAM swap space.

Function *Reclaim_Efficiency_Estimator()* determines whether it is more appropriate to swap out a process selected by *LMK* to zRAM swap space or to kill it. This function obtains its *RSS* and the number of pages that have been swapped out to the swap space (*Swapped_RSS*). As *Swapped_RSS* gets larger, the number of pages to be reclaimed will be dwindled because *RSS* gets smaller. If $(Swapped\_RSS - RSS) < 0$, since greater than 50% pages of the original resident set of the process can be still reclaimed from main memory, the process selected by *LMK* is swapped out as shown in ő in Fig. 2; otherwise, since there is little chance that more pages can be reclaimed, it is killed as shown in ŕ in Fig. 2.

IV. PERFORMANCE EVALUATION

It is necessary to increase the extension ratio of main memory to ensure that multiple applications can run simultaneously and to provide the fast response times. In order to measure the extension ratio and the cost of page operations of the hybrid swapping scheme proposed in this paper, we installed as many high-demand applications as possible to use up all the swap space on a smartphone due to frequent swapping in and out of code and data to the swap space.

For performance evaluation, we utilized three models of LG Android smartphones like LG Stylo 4, LG G7 and LG V35. The LG Stylo 4 [27] was equipped with of the Qualcomm Snapdragon SDM450 chipset, 2GB LPDDR3, and eMMC v5.1. We have installed the six most popular mobile games on Google Play in Korea at the time of performance evaluation such as "Clash Royale," "Lineage 2 Revolution," "Asphalt 8 Airbone," "Raven," "Anypang3 for Kakao," and " FIFA Online 4M ," and 10 frequently-used web browsers, utilities and multimedia applications for Android, as shown in Table 3.

Upon triggering one application, we returned to the home screen by hitting home key button to trigger another application. We repeatedly triggered each of 16 applications and returned from it. We call this batch of 16 triggering of applications as a "turn." In order to measure performance, we performed 16 turns, and the order of triggering 16 applications was changed randomly for each turn. For performance comparison of hybrid swapping, zRAM swapping, and zSWAP, we measured the memory usage and the entry time of each application in the identical test environment.

TABLE 3. The list of test applications.

Application Name	Category
Clash Royale	Game
Lineage 2 Revolution	
Asphalt 8 Airbone	
Raven	
Anypang3 for Kakao	
FIFA Online 4M	
YouTube	Multimedia
Camera	
Gallery	
Music	
Play Store	Utility
Google Map	
Phone	
Settings	
Chrome	Web Browser
Naver	

The LG G7 [28] and the LG V35 [29] were equipped with 4GB DRAM and 6GB DRAM, respectively. We measured the extended size of main memory for the two models using 60 applications instead of 16 applications to exhaust available system memory quickly. The 60 applications consists of 9 Google Mobile Services (GMS) applications, 5 LG default applications like Q memo+, file manager, and 30 most downloaded Google Play applications like Facebook, Instagram, Netflix, etc in 2018, including 16 applications in Table 3.

A. EXTENSION RATIO OF MAIN MEMORY

Table 4 shows the average extended sizes of main memory and the extension ratios for three swapping schemes. In case of LG Stylo 4, the extension ratio of hybrid swapping was improved by 17% over that of zRAM swapping and zSWAP. It was also improved by 15% and 6% over that of zRAM swapping and zSWAP, respectively, in case of LG G7, and 15%, and 12%, respectively, in case of LG V35.

B. COST OF PAGE OPERATION

In this section, we compared the average times to trigger 16 applications for hybrid swapping, zRAM swapping and zSWAP. When an application is invoked, its entry time includes the cost of page operations needed to acquire main memory to bring virtual pages into main memory. Since three swapping schemes were evaluated under the identical execution environment, the average entry time of each application was mainly affected by the cost of page operations. Thus, it is enough to compare their costs of page operations for comparison of average entry times of applications.

TABLE 4. Comparison of average main memory usage.

Smartphone Model	zRAM Swapping		zSWAP		Hybrid Swapping	
	Extended Size (MB)	Extension Ratio	Extended Size (MB)	Extension Ratio	Extended Size (MB)	Extension Ratio
LG Stylo 4 with 2GB of DRAM	2253	1.10	2261	1.10	2655	1.29
LG G7 with 4GB of DRAM	4728	1.15	5090	1.24	5413	1.32
LG V35 with 6GB of DRAM	6755	1.10	6988	1.13	7824	1.27

TABLE 5. Average entry times of applications in LG stylo 4.

Application Name	zRAM Swapping (ms)	zSWAP (ms)	Hybrid Swapping (ms)
Clash Royale	2120	1990	1700
Lineage 2 Revolution	3120	3090	2960
Asphalt 8 Airbone	1370	1460	1460
Raven	2190	2840	1480
Anypang3 for Kakao	2140	1810	1200
FIFA Online 4M	2400	3010	1920
YouTube	1880	1840	1640
Camera	1140	1010	990
Gallery	1530	1670	1200
Music	1520	1860	980
Play Store	1120	1200	820
Google Map	2350	2230	1530
Phone	1510	1950	1470
Settings	1810	1600	950
Chrome	840	830	790
Naver	760	1770	680
Average	**1737**	**1885**	**1360**

Table 5 shows the average entry times of three swapping schemes in LG Stylo 4. Hybrid swapping reduced them by 22% and 28% over zRAM swapping and zSWAP, respectively. When hybrid swapping was utilized, the entry times of applications were reduced except "Asphalt 8 Airbone" in the game category. For this application, function *Reclaim_Efficiency_Estimator( )* estimated that the cost to kill it was cheaper than the cost of its page swap-in/outs. Since zSWAP has the highest page operation cost, the average entry time of applications should be the longest.

The numbers of swap-in/out pages are 219,892 and 265,444 for zSWAP, respectively, and are 56,492 and 81,002 for hybrid swapping, respectively, in LG Stylo 4. This demonstrates that hybrid swapping can differentiate frequently and infrequently referenced pages more accurately than zSWAP.

Table 6 shows the average entry times of applications of three swapping schemes in LG G7 with 4GB of RAM. Hybrid swapping reduced them by 9% and 18% over zRAM swapping and zSWAP, respectively. In case of hybrid swapping, the entry times of applications were reduced except "Google Map," "Naver," "Camera," and "Gallery" due to the same reason as mentioned in LG Stylo 4.

The numbers of swap-in/out pages are 1,396,442 and 1,886,562 for zSWAP, respectively, and are 220,442 and 256,562 for hybrid swapping, respectively, in LG G7.

TABLE 6. Comparison of average entry times of applications in LG G7.

Application Name	zRAM Swapping (ms)	zSWAP (ms)	Hybrid Swapping (ms)
Clash Royale	1700	1820	1320
Lineage 2 Revolution	1310	1300	1190
Asphalt 8 Airbone	1540	1600	1270
Raven	1380	1450	1240
Anypang3 for Kakao	1050	1180	910
FIFA Online 4M	1200	1290	1040
YouTube	1330	1760	1240
Camera	540	610	600
Gallery	970	720	870
Music	770	820	710
Play Store	770	820	700
Google Map	1240	1280	1350
Phone	470	890	370
Settings	670	920	620
Chrome	510	520	490
Naver	450	600	490
Average	**993**	**1098**	**900**

Those of LG G7 are much larger than LG Stylo 4 since it runs 60 test applications as mentioned before.

TABLE 7. Comparison of average entry times of applications in LG V35.

Application Name	zRAM Swapping (ms)	zSWAP (ms)	Hybrid Swapping (ms)
Clash Royale	1080	1100	1070
Lineage 2 Revolution	1260	1390	1010
Asphalt 8 Airbone	1540	1440	1210
Raven	1420	1490	1150
Anypang3 for Kakao	690	710	690
FIFA Online 4M	1040	1060	980
YouTube	1210	1500	1070
Camera	340	350	290
Gallery	840	1070	850
Music	710	910	660
Play Store	320	320	240
Google Map	890	860	820
Phone	450	470	340
Settings	1290	1370	1120
Chrome	560	620	470
Naver	600	720	520
Average	**890**	**961**	**780**

Table 7 shows the average entry times of three swapping schemes in LG V35 with 6GB of RAM. Hybrid swap-

ping reduced them by 13% and 19% over zRAM swapping and zSWAP, respectively. Note that the average entry time of hybrid swapping in G7 with 4GB DRAM is almost identical with that of zRAM swapping in V35 with 6GB DRAM.

The numbers of swap-in/out pages are 1,047,354 and 1,489,475 for zSWAP, respectively, and are 117,166 and 128,773 for hybrid swapping, respectively in LG V35. Those of LG V35 are smaller than LG G7 since the main memory size of LG V35 is larger than that of LG G7.

C. A MEASURE OF GUARANTEED NON-WEAR-OUT TIME

According to a report [30] by SK Hynix, a manufacturer of eMMC memory used by the LG Stylo 4 Android smartphone, *GP*(guaranteed non wear-out pages) of eMMC is 9Tbyte/4Kbyte pages since it guarantees writing up to 9Tbytes. We measured the total numbers of pages swapped out into the secondary storage swap space by repeatedly executing 60 applications to obtain $\sum_{t=t_1}^{t_n} [\Delta SO(t_k) + \Delta FO(t_k)]$, (k = 1, 2, . . . , n) per unit time t for hybrid swapping and zSWAP. The total number of pages swapped out into eMMC swap space and written by Android file output operations was 4.5Gbytes/4Kbytes pages and 29Gbytes/4Kbytes pages, respectively, for hybrid swapping and zSWAP, where $t = 48$ hours.

We calculated guaranteed non wear-out time (*GT*) of hybrid swapping and zSWAP using Equation (2), which is the time taken to write all *GP* pages into NAND flash. *GTs* of the former and the latter are roughly 4,096 days and 635 days, respectively. Consequently, we can expect that there will be virtually no wear-out in case of hybrid swapping since wear-out problem may not occur during almost 11 years.

V. CONCLUSION

The requirements for memory of an Android smartphone in capacity and speed have been continuously increasing recently since users tend to install applications with a higher demand for main memory. To lower the requirement, smartphone developers have recently made many efforts to extend its main memory with zRAM swapping. However, since zRAM swapping utilizes main memory, there is a limit to extend its size. On the other hand, an Android smartphone cannot adopt secondary-storage swapping using secondary storage as swap space due to long response time and wear-out problem. To overcome these limitations, we proposed hybrid swapping that supports both secondary-storage swapping and zRAM swapping.

Hybrid swapping improved the extension ratio of main memory by using zRAM swapping, which has been widely used in mobile environment, and by using secondary-storage swapping, which have not been used due to performance degradation and wear-out problem. It also reduced the cost of page operations by swapping out pages to different swap space based on their reference patterns and by swapping out all the pages in the working set of a process into zRAM swap space using per-process reclaim rather than killing that process.

According to our performance evaluation, hybrid swapping increased the extension ratio of main memory by 15 $\sim$ 17% and 6 $\sim$ 17% and reduced the page operation cost by 9 $\sim$ 22% and 18 $\sim$ 28%, respectively, compared to zRAM swapping and zSWAP for 16 real-world applications. It also showed that the wear-out problem of flash memory will not occur for almost 11 years by drastically reducing the number of swap-in/outs from/to the secondary storage swap space.

Hybrid swapping determined whether to swap out pages of a process to the secondary storage swap space or zRAM swap space, or to kill the process, depending on the probability of referencing the pages. As a future work, we will investigate a technique to improve the performance of hybrid swapping by predicting the reference probabilities of pages more accurately. We will also investigate how to enhance the performance of our scheme by utilizing NVM if it becomes cost-effective and can be embedded in multi-chip package.

REFERENCES

[1] PassMark Software. Surry Hills, NSW, Australia. (2018). *CPU Benchmarks*. [Online]. Available: https://www.cpubenchmark.net/
[2] PassMark Software. Surry Hills, NSW, Australia. (2018). *Android Benchmarks*. [Online]. Available: https://www.androidbenchmark.net/
[3] *Embedded Multi-Media Card (eoMMC), Electrical Standard (5.1)*, Standard JESD84-B51, Feb. 2015.
[4] *Universal Flash Storage (UFS) Unified Memory Extension, Version 1.1*, Standard JESD220-1A, Mar. 2016.
[5] Y. Pan, G. Dong, and T. Zhang, "Exploiting memory device wear-out dynamics to improve NAND flash memory system performance," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST)*, San Jose, CA, USA, Feb. 2011, p. 18.
[6] N. Gupta. zram: Compressed RAM Based Block Devices. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://www.kernel.org/doc/Documentation/blockdev/zram.txt
[7] M. Johnson, "An introduction to block device drivers," *Linux J.*, Jan. 1995. [Online]. Available: https://www.linuxjournal.com/article/2890
[8] S. Jennings. zSwap. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://www.kernel.org/doc/Documentation/vm/zswap.txt
[9] J. Thornber, H. Mauelshagen, and M. Snizer. dm-cache. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://www.kernel.org/doc/Documentation/device-mapper/cache.txt
[10] A. Kergon, N. Brown, M. Broz, and L. Torvalds. Device-Mapper. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://www.kernel.org/doc/Documentation/device-mapper/
[11] TrendForce Corp. Taipei, Taiwan. (2018). *Price Trend*. [Online]. Available: https://www.trendforce.com/price
[12] PassMark Software. Surry Hills, NSW, Australia. (2018). *RAM Benchmarks*. [Online]. Available: https://www.memorybenchmark.net/
[13] PassMark Software. Surry Hills, NSW, Australia. (2018). *Hard Drive Benchmarks*. [Online]. Available: https://www.harddrivebenchmark.net/
[14] K. Liu, X. Zhang, K. Davis, and S. Jiang, "Synergistic coupling of SSD and hard disk for QoS-aware virtual memory," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 24–33.
[15] D. Chae *et al.*, "CloudSwap: A cloud-assisted swap mechanism for mobile devices," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, 2016, pp. 462–472.
[16] Q. Zhang, L. Liu, G. Su, and A. Iyengar, "MemFlex: A shared memory swapper for high performance VM execution," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1645–1652, Sep. 2017.
[17] Q. Zhang, L. Liu, C. Pu, W. Cao, and S. Sahin, "Efficient shared memory orchestration towards demand driven memory slicing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Jul. 2018, pp. 1212–1223.
[18] K. Zhong, D. Liu, L. Long, J. Ren, Y. Li, and E. H.-M. Sha, "Building NVRAM-aware swapping through code migration in mobile devices," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3089–3099, Nov. 2017.

[19] D. Liu, K. Zhong, X. Zhu, Y. Li, L. Long, and Z. Shao, "Non-volatile memory based page swapping for building high-performance mobile devices," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1918–1931, Nov. 2017.
[20] X. Zhu, D. Liu, L. Liang, K. Zhong, M. Qiu, and E. H.-M. Sha, "Swap-Bench: The easy way to demystify swapping in mobile systems," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun.*, Aug. 2015, pp. 497–502.
[21] M. Kim. (Apr. 25, 2013). Per Process Reclaim. LWM.net. [Online]. Available: https://lwn.net/Articles/548431/
[22] Qualcomm, San Diego, CA, USA. (2015). *Qualcomm Snapdragon 617 Processor*. [Online]. Available: https://www.qualcomm.com/documents/snapdragon-617-processor-product-brief
[23] L. Torvalds. (1994). VMSCAN. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/tree/mm/vmscan.c
[24] R. van Riel and D. Rientjes. (2010). Out of Memory Killer. Linux Foundation, San Francisco, CA, USA. [Online]. Available: https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/mm/oom_kill.c
[25] S. Mehat, A. Hjønnevåg, D. Rientjes, G. Kroah-Hartman, and D. Walker. (2008). Lowmemorykiller.c. Google, Inc., Mountain View, CA, USA. [Online]. Available: https://android.googlesource.com/kernel/common.git/+/android-4.4-o-release/drivers/staging/android/lowmemorykiller.c
[26] P. J. Denning, "The working set model for program behavior," *Commun. ACM*, vol. 11, no. 5, pp. 323–333, May 1968.
[27] LG Electronics. Seoul, South Korea. (2018). *LG Stylo 4*. [Online]. Available: http://www.lg.com/us/cell-phones/lg-Q710MS-MetroPCS-stylo-4
[28] LG Electronics. Seoul, South Korea. (2018). *LG G7 ThinQ*. [Online]. Available: http://www.lg.com/us/cell-phones/g7-thinq
[29] LG Electronics. Seoul, South Korea. (2018). *LG V35 ThinQ*. [Online]. Available: http://www.lg.com/us/cell-phones/lg-V350AWM-v35-thinq-att
[30] *Endurance 14nm_32GB_eMMC51_12nm_2GB_LG_Stylo4_170731_r2*, SK Hynix Inc., Icheon, South Korea, 2017.

JUNYEONG HAN received the B.S. and M.S. degrees in computer engineering from the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, in 2006 and 2008, respectively.

He is currently with LG Electronics, Seoul, as a Principal Research Engineer. He has been working to improve the performance of the Linux Kernel for Android smartphone. His research interests include Linux systems and machine learning for mobile systems.

SUNGEUN KIM received the B.S. and M.S. degrees in electronic engineering from the School of Electronic Engineering, Korea University, Seoul, South Korea, in 1997 and 2000, respectively.

He was with Samsung Electronics, Seoul, as a Linux Kernel Developer, from 2000 to 2005. Since 2008, he has been developing the Linux-based operating system and frameworks for consumer electronic products at LG Electronics, Seoul.

SUNGYOUNG LEE received the B.S. and M.S. degrees in computer science from Korea University, Seoul, South Korea, in 1994 and 1996, respectively.

He has been with LG Electronics, Seoul, as a Research Fellow, since 1996. He holds three domestic patents and one U.S. patents. He is currently developing mobile system software, embedded systems, and embedded Linux systems. His research interests include Linux memory management and machine learning for mobile memory management optimization depending on user's usage pattern at the mobile.

JAEHWAN LEE received the B.S. degree in computer engineering from the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, in 2015, where he is currently pursuing the Ph.D. degree in software engineering.

His research interests include mobile operating systems, embedded systems, and Linux systems.

SUNG JO KIM received the B.S. degree in applied mathematics from Seoul National University, Seoul, South Korea, in 1975, the M.S. degree in computer science from KAIST, Seoul, in 1977, the Ph.D. degree in computer engineering from The University of Texas at Austin, Austin, TX, USA, in 1987.

Since 1987, he has been a Professor with the School of Software, Chung-Ang University, Seoul. He served as the 24th President of the Korean Institute of Information Scientists and Engineers from 2009 to 2010. He is the author of five books and over 210 articles, and he holds over 21 patents. His research interests include cyber-physical systems, context-aware smart home, and power management of mobile operating systems such as Android.

Dr. Kim was a recipient of the 13th Haedong Award in 2017, which has been given to a person who had contributed to significant innovations in engineering education in South Korea by the National Academy of Engineering of Korea since 2005.