


ElasticZRAM: Revisiting ZRAM for Swapping on Mobile Devices

Wentong Li^{1,2}, Dingcui Yu², Yunpeng Song², Longfei Luo²,  Liang Shi^{1,2}

¹Software/Hardware Co-design Engineering Research Center, Ministry of Education

²East China Normal University, Shanghai, China

ABSTRACT

Modern mobile devices adopt two-level memory swapping consisting of ZRAM and storage devices to relieve memory pressure. In the swap subsystem, ZRAM can improve application responsiveness and reduce write traffic to storage devices while consuming physical memory and additional CPU cycles. To better utilize ZRAM and improve system performance, we propose ElasticZRAM, an elastic ZRAM to redesign the traditional memory swapping with full awareness of the characteristics of applications and NAND flash-based storage devices on mobile devices. Experimental results on Google Pixel 6 demonstrate that ElasticZRAM improves application response time by up to 24.8% with negligible overhead compared with state-of-the-arts.

1 INTRODUCTION

Modern mobile systems cache applications in memory after they have been closed to preserve the application context while switching quickly. However, since applications are becoming more and more feature-rich and powerful, mobile systems are constantly under memory pressure, especially for mobile devices equipped with limited DRAM [1]. To relieve memory pressure, mobile systems always directly kill the victim applications or perform memory swapping. Specifically, the former, named low memory killer (LMK) [2], directly kills the selected processes, which inevitably causes the loss of application context and a significant application relaunching latency. Compared with LMK, memory swapping can move inactive memory pages to the compressed memory or NAND flash-based storage device and effectively caches application state to enable fast application responsiveness [3].

There are two existing memory swapping schemes: memory compression-based swapping (i.e., ZRAM) and NAND flash-based swapping. For the former, once memory pressure happens, victim memory pages are compressed and placed into ZRAM (a compressed partition of DRAM) and decompressed when copied from ZRAM. In the past decade, ZRAM has been adopted by mobile systems as the default memory swapping mechanism [4]. However, as applications grow, ZRAM (de)compresses pages more and more frequently, which are highly compute-intensive and thus significantly consume massive CPU cycles [5]. Also, since ZRAM occupies the already limited memory space, the LMK mechanism is frequently triggered due to low memory and low available swap space [6]. For the latter, with the rapid development of NAND flash-based storage

devices (e.g., UFS and eMMC) for mobile devices [7], there has been lots of research efforts on optimizing the efficiency of NAND flash-based swapping [8–10]. However, due to sub-optimal read/write performance compared with direct memory interface accesses and the lifetime concern of flash storage, it seems challenge to only use NAND flash as a swap space.

To further improve the efficiency of memory swapping, a two-level memory swapping architecture consisting of ZRAM and NAND flash has been widely studied on mobile devices [11–13]. For instance, SEAL [11] first proposed a two-level memory swapping subsystem and placed data accessed during app launching in the compressed memory and data accessed during app execution in the local storage. ICSWAP [12] further adopted the compressed memory as a swap-out buffer to cluster neighbor pages and a swap-in cache to service swap-in requests to reduce the number of swap I/Os from local storage. SWAM [13] performed memory swapping adaptively into ZRAM or storage devices by examining the information on the victim page to further reduce LMK and guarantee a quick response for the running applications. Thanks to the two-level memory swapping, almost all well-known commercial mobile device manufacturers have launched memory expansion technologies [14–16], including Samsung, Huawei, etc. However, the management strategy of ZRAM has been ignored in previous work. On the one hand, since the state-of-the-art storage devices are still slower than the DRAM-based main memory, ZRAM can effectively improve application responsiveness and reduce write traffic to the storage device. On the other hand, compressed-based ZRAM induces additional CPU overhead and occupies the limited physical memory. Different from existing works, this paper will propose the first elastic ZRAM mechanism to better weigh the benefits and expenses of ZRAM for swapping on mobile devices.

In this paper, we first study the two-level memory swapping system. There are two observations from the study: First, the earlier the application is swapped out to the swap partition, the longer the response time will be, and vice versa. This is because the earlier swapped-out applications have often been written back to the storage device, while the latest swapped-out applications are often stored in ZRAM based on least recent used algorithm (LRU). Second, storage devices currently used in mobile devices are often equipped with hybrid flash device, which include a high-performance region and a high-density region to achieve both high performance and large capacity at the same time. However, their I/O performance fluctuates wildly as the storage utilization increases. Based on these two observations, ElasticZRAM is proposed to redesign the two-level memory swapping by an elastic ZRAM management strategy to improve application responsiveness. Specifically, it combines the access characteristics of applications and performance characteristics of storage devices to adaptively guide the swapping of applications between ZRAM and storage through different

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0601-1/24/06...\$15.00

<https://doi.org/10.1145/3649329.3655943>

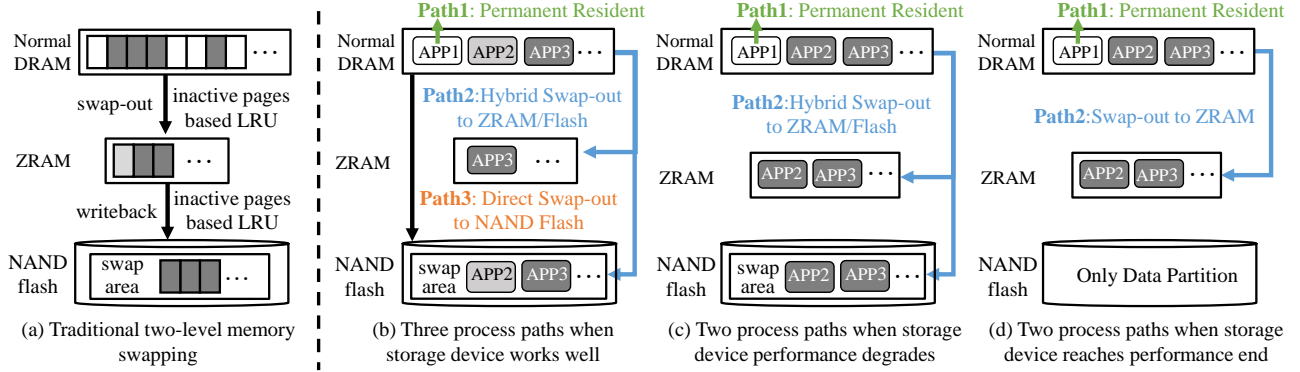


Figure 1: Comparison among traditional swapping system with ElasticZRAM.

process paths. To select a suitable process path, we profile application and storage usage to conduct the process path selection and elastic ZRAM resizing. Our evaluation results demonstrate that ElasticZRAM improves application responsiveness with negligible system overhead compared to the state-of-the-art. In summary, the major contributions of this paper are as follows:

- We clarify that the management strategy of ZRAM in two-level memory swapping is both application-agnostic and storage device-unfriendly.
- We introduce an elastic ZRAM scheme for swapping on mobile devices, ElasticZRAM, which exploits ZRAM rationally through different process paths to ensure system performance and storage awareness.
- ElasticZRAM is deployed on Google Pixel 6, and experimental results show that ElasticZRAM effectively improves application responsiveness compared to the state-of-the-art.

2 BACKGROUND AND PROBLEM STATEMENT

2.1 Memory Swapping on Mobile Devices

In mobile systems, simply using ZRAM for memory swapping faces high (de)compression overhead and frequent application killing (due to limited swap space). Likewise, using NAND flash-based swapping faces progressively deteriorating I/O performance and limited lifetime [12, 17]. Based on these, a two-level memory swapping architecture has been proposed and widely studied. It employs the swap area on a compressed portion of DRAM and the underlying storage devices. Fig. 1(a) shows the two-level swapping system, which works as follows: first, when the system's free memory reaches the low memory watermark, the kernel thread *kswapd* is awakened to perform memory reclamation. Inactive memory pages based on LRU are moved to the first level of compressed memory (i.e., ZRAM). When ZRAM is almost full, the victim pages are written back to the underlying storage device. For storage devices, the hybrid architecture is widely used in UFS to simultaneously achieve large capacity, high performance, and high endurance [18]. Specifically, it is constructed with a small-size front-end high-performance storage, such as SLC flash, and a large-size back-end high-density storage, such as QLC flash. Different from the existing design, this paper proposes to redesign the swapping system based on the characteristics of applications and storage, as shown in Fig. 1(b)-(d).

2.2 Problem Statement

2.2.1 Undesired Application Performance Variation. To quantitatively show the influence of two-level memory swapping on application responsiveness, the application response time (i.e., YouTube

and Twitter) under two typical scenarios is collected. The scenarios include: the application has just been swapped out to the swap partition, named *new arrival*; it has been in the swap partition for a long time (interval for several applications, ZRAM is repeatedly filled), named *long staying*. Also, to be more realistic, the two representative applications are run from tens of seconds to tens of minutes to obtain five different memory footprints, determining the amount of data swapped. During the experiments, ZRAM is set to 1 GB and the storage-based swap partition is 3 GB like SWAM[13].

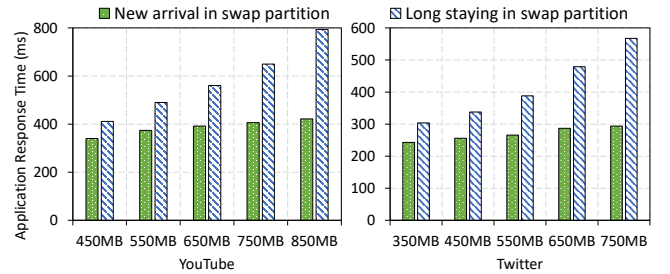


Figure 2: Influence of two-level memory swapping on application response time under different swap-out durability.

Fig. 2 shows the results. There are several findings: First, with applications swapped with different duration, their corresponding response time vary considerably. For example, if the memory footprint is 850 MB, the response time of YouTube long staying in the swap partition increases by 88% compared to the scenario when it is new arrival, while 93% for Twitter. This vast gap stems from the current two-level memory swapping paging strategy. When the system faces memory pressure, the victim page is first compressed and placed in ZRAM based on the traditional LRU algorithm. When ZRAM is almost full, the page is decompressed and written to the storage device. For applications long staying, most pages have been written back to the storage device, and vice versa. Second, applications with smaller memory footprints are less susceptible to the durability of swap-out. For example, the difference is 21% for YouTube and 25% for Twitter when the memory footprint is 450 MB and 350 MB, respectively.

2.2.2 Swap Writes and Performance Degradation of Storage Device. Compared with ZRAM, two-level memory swapping allows more data to be written to the storage device. The increased data written will affect the device lifetime and the performance of storage devices. To assess the impact of these two aspects, we first collect the write volume of storage devices under various swapping systems. Then, we analyze the read/write bandwidth of storage devices with different storage utilization.

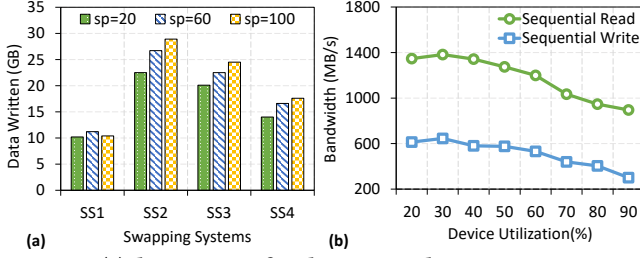


Figure 3: (a) data written for the storage device among various swapping systems with different swappinesses (sp, controls the tendency of the kernel to move memory pages to storage device). (SS1: independent ZRAM (1 GB); SS2: Flash-swap (3 GB); SS3: ZRAM (512 MB)/Flash-swap; SS4: ZRAM (1 GB)/Flash-swap); (b) I/O bandwidth of storage device under various device utilizations.

Fig. 3(a) shows that compared with ZRAM, flash-swap significantly increases write pressure by 2.3x on average. For two-level memory swapping, when ZRAM is gradually increased, the amount of writes to the storage device gradually decreases. For example, when the ZRAM size is 1 GB, the amount of data written to the storage device is reduced by 51%. This is because the existence of ZRAM will naturally prevent some data from being written to the storage device. Moreover, even under different swappiness (the larger swappiness is, the more aggressively memory swapping is used), this has little effect on this trend. Another problem with enabling a storage device as a swap partition is that it will experience performance degradation as its utilization increases. As shown in Fig. 3(b), as storage utilization increases, the read/write bandwidth of storage devices declines to varying degrees. When the storage utilization exceeds 90%, compared with 20%, the sequential read performance drops by 43%, and the sequential write performance drops by 52%. This implies that the two-level swapping system must consider the performance degradation of the storage devices. However, it is completely ignored in the design of swapping systems.

3 DESIGN

3.1 ElasticZRAM Framework

Overview. Motivated by undesired application responsiveness variation and possible performance degradation of storage devices, this paper firstly redesigns the traditional two-level memory swapping, as shown in Fig. 1(b)-(d). Unlike previous design, we divide memory swapping into three periods based on storage device performance and exploit different process paths in each period based on the access characteristics of applications to guarantee user experience. First, as shown in Fig. 1(b), when the storage device works well, three process paths can be selected. Specifically, since the application usage frequency determines the hotness of memory swapping, the frequently used applications should be permanently cached in memory, corresponding to *Path1*, and applications with low frequency are swapped out in priority. Once the victim application is selected, we need to determine which swap partition it will be swapped out. Thanks to the different pages required by the application launch and execution [11], the application can be swapped out to the two-level memory swapping, corresponding to *Path2*. This selection is based on the characteristics of applications. If the footprint of application is large under the same usage frequency, it will be swapped following *Path2*, where part of its footprint is

swapped to ZRAM, and the other part is swapped to storage. Otherwise, for the small-size applications, they can be directly swapped to the storage to avoid the occupation of ZRAM, corresponding to *Path3*. Second, flash-based swapping should not be used aggressively when storage performance degrades, as shown in Fig. 1(c). In this case, *Path3* is disabled, and *Path2* should be optimized to maximize the proportion of ZRAM usage. Finally, when storage devices reach the end of their performance, flash-based swapping should be completely disabled, as shown in Fig. 1(d). In this case, *Path3* is disabled, and *Path2* is updated to enable ZRAM only.

Architecture. The above design is not simple since there are two challenges to overcome. First, a set of criteria is needed for the selection of various process paths. Second, the size of ZRAM needs to be determined in the case *Path2* with the two-level memory swapping. To solve these issues, we first profile application and storage usage. And then the process path selection and elastic ZRAM resizing is conducted. Fig. 4 shows the architecture of ElasticZRAM. It consists of three components, including application and storage usage profiling, process path selection and elastic ZRAM resizing. Specifically, the profiling module first collects the application and storage usage statistics including the application memory footprint, usage frequency and device utilization. Then, applications are selected to perform swap-out by different paths based on the collected information. Finally, an elastic ZRAM resizing scheme is designed to appropriately increase ZRAM and reduce NAND flash-based swap to ensure system performance and storage awareness.

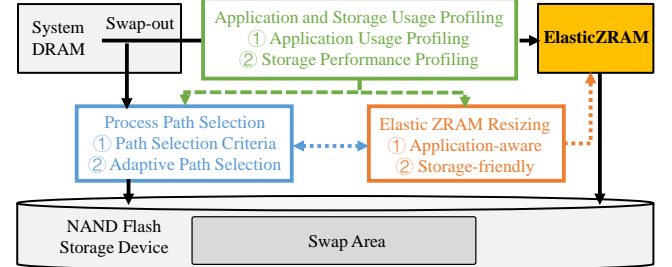


Figure 4: The architecture of ElasticZRAM.

3.2 Application and Storage Usage Profiling

To support process path selection and ZRAM resizing, application and storage device usage information is profiled. Fig. 5 shows the collected information related to memory swapping.

Application Usage Profiling. First, we record the number of times an application is opened, called usage rate, to record the frequency of application usage, which decides the hotness of memory swapping. Whenever the application starts, we record its usage rate plus one. Also, the usage rates of applications are sorted from high to low. And then, once the application is closed, the memory footprint is recorded and updated each time, which decide the amount of data swapped. Finally, to evaluate the performance of memory swapping, we record the response time of each application launch. For example, the memory footprint of Facebook's last run is 650 MB and it is opened 8 times in one day.

Storage Performance Profiling. The performance degradation of storage devices is mainly reflected in I/O performance and device lifetime. First, the I/O performance of hybrid flash will decrease significantly as the size of SLC partition is reduced. Based on the size of the SLC partition, storage device performance can be divided

into three stages, including the new stage, the middle stage, and the end stage [18]. The new stage indicates that the device performance has little changes, and the middle stage indicates that the storage device performance changes significantly as the SLC shrinks. The end stage indicates severe deterioration in equipment performance. For device lifetime, the greater the writing volume is, the lower the device lifetime will be. To identify the impact of storage device utilization on performance variations, device utilization and data written are monitored periodically.

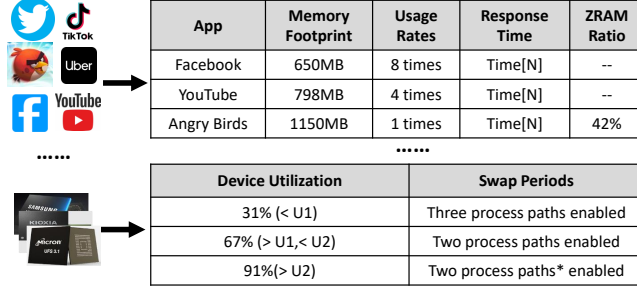


Figure 5: Application and storage usage profiling. $Ratio_{zram}$ will only be assigned when using hybrid ZRAM and NAND flash; * indicates the $Path2$: swap-out directly to ZRAM.

3.3 Process Path Selection

Path Selection Criteria. In our design, the selection of process paths includes two factors: storage device performance and application responsiveness. First, to deal with the performance degradation of storage devices, memory swapping is divided into three periods. Their selection depends on device utilization and storage write volume. When the device utilization is lower than $U1$, it means the performance is stable; when it is higher than $U1$ and lower than $U2$, it means the performance is degraded; and when it is higher than $U2$, it means the performance is almost unavailable [18]. At the same time, when the writing amount approaches the end, it will directly enter the last period. In our implementation, $U1$ is 40%, and $U2$ is 85%; this value varies on different devices [18]. Second, each period corresponds to different process paths to guarantee application responsiveness. Their selection depends on the application usage frequency and memory footprint.

Adaptive Path Selection. Based on the criteria, we perform an adaptive path selection. Algorithm 1 shows how paths are selected. $AppUsage_{info}$ and $StorageUsage_{info}$ represents the collected application and storage information respectively. First, the device utilization and the application with the lowest usage frequency are obtained by accessing $AppUsage_{info}$ and $AppUsage_{info}$ (Lines 2-3). If the device utilization is less than $U1$, there are three process paths. If there is only one application of the lowest usage frequency, that application will be directly swapped out by $Path2$ (Lines 5-8). If there is more than one, the memory footprint of applications at the same frequency is compared and swapped out bigger application through $Path2$ (Lines 9-10). Secondly, even though ZRAM only holds a portion of each application's pages, it will hit the space limit as the number of cached applications grows. Whenever this happens, the usage frequency of applications that has been cached in the ZRAM is compared again and the application with the smallest frequency is swapped out through $Path3$ (Lines 13-16). Finally, if the device utilization exceeds $U1$, there are two process paths,

Algorithm 1 Adaptive Path Selection

```

1: Procedure Adaptive_Path_Selection( $AppUsage_{info}$ ,  $StorageUsage_{info}$ )
2: Device_Utilization = GetDeviceInfo( $StorageUsage_{info}$ )
3:  $App_i$  = MinUsageRate( $AppUsage_{info}$ )
4: if Device_Utilization <  $U1$  then
5:   if ! ZRAM_IS_FULL() then
6:     /* App_Count: the number of applications that run in the background */
7:     if App_Count == 1 then
8:       Victim App =  $App_i$ , ProcessPath = Path2
9:     else
10:      Victim App = MaxMemoryfp( $App_i$ ), ProcessPath = Path2
11:    end if
12:  else
13:    if App_Count == 1 then
14:      Victim App = MinUsageRate( $App_i$ ), ProcessPath = Path3
15:    else
16:      Victim App = MaxMemoryfp( $App_i$ ), ProcessPath = Path3
17:    end if
18:  end if
19: else
20:   /* ProcessPath = Path2: Device_Utilization >  $U1$  and Device_Utilization <
21:     $U2$ ; ProcessPath = Path2* (swap-out to ZRAM): Device_Utilization >  $U2$  */
22:   if App_Count == 1 then
23:     Victim App =  $App_i$ , ProcessPath = Path2/Path2*
24:   else
25:     Victim App = MaxMemoryfp( $App_i$ ), ProcessPath = Path2/Path2*
26:   end if
27: RealTimeUpdate( $AppUsage_{info}$ ) /* Once an application is started or killed */
28: PeriodicUpdate( $StorageUsage_{info}$ ) /* Once a day */

```

and $Path2$ is different in different periods (Line 20). Consistently, applications use $Path2$ based on the usage frequency and memory footprint (Lines 21-25). Note that for background applications, $Path1$ is the result of the selection of $Path2$ and $Path3$, and we do not specify the applications which should be permanently resident.

3.4 Elastic ZRAM Resizing

When the performance of the storage device is basically stable, $Path2$ based on hybrid ZRAM and NAND flash faces the issue of how much proportion of the application's pages should be placed in ZRAM. In the rest of the paper, this ratio is called $Ratio_{zram}$ (less than 100%), which has a critical impact on application responsiveness. To tackle this challenge, a ZRAM resizing model is provided. This model is responsible for calculating the $Ratio_{zram}$ for $Path2$. It estimates the ratio as a function of historical application response time and the magnitude of $Ratio_{zram}$ changes (δ). Assuming that the average response time of the application in the past N times is $\sum_{i=1}^N T_i/N$, and the latest time is T_N . The proportion of pages placed in ZRAM by the application is:

$$Ratio_{zram} = \begin{cases} \text{Max}(0, Ratio_{zram} - \delta) & (T_N < \sum_{i=1}^N T_i/N) \\ \text{Min}(100\%, Ratio_{zram} + \delta) & (T_N > \sum_{i=1}^N T_i/N) \end{cases}$$

This model is calibrated dynamically. The value of $\sum_{i=1}^N T_i/N$ is calculated based on app_i 's response time in history. δ is a configurable rangeability from 0 to 1 to tolerate the bias of collected application response time. The motivation of δ is to provide a safe margin. If it is set to 20%, ZRAM stores 20% more application pages at a time, and pages related to application launching are prioritized [11]. With this model, application responsiveness is guaranteed and ZRAM can be minimized. In our design, N depends on the usage frequency of an application. If it is 0, $Ratio_{zram}$ remains unchanged. Based on the information analyzed above, we reconsider the process paths that can be used when storage device performance degrades. When the storage device reaches the middle stage,

we disable the original *Path3*. At the same time, when using *Path2*, I/O performance degradation is directly reflected in application response time, and the above ZRAM resizing model can deal with this performance degradation.

3.5 Implementation and Discussion

ElasticZRAM is implemented on Google Pixel 6. In the current implementation, most functions of ElasticZRAM include three parts. First, application usage data, including memory footprint, usage frequency, and response time, are analyzed and recorded. Specifically, the information is maintained in a struct *app_info*. Second, the swap-out path of the swapping subsystem is expanded to three types. The original LRU-based memory swapping policy is modified to be based on application-specific memory swapping. Finally, storage device utilization and write volume are monitored periodically.

NAND flash-based storage devices have been iteratively upgraded, while the natural flaws of ZRAM are inevitable. Therefore, flash-swap will be more and more actively used, and the system will become less and less dependent on ZRAM. By dynamically adjusting the size of ZRAM, ElasticZRAM ensures application responsiveness and gives full play to storage device performance. Moreover, it reduces the system's demand for ZRAM, which perfectly fits the development of two-level memory swapping in future mobile devices. Also, to confirm the overhead, we analyzed the extra behaviors of ElasticZRAM by comparing it with the existing system in detail. Then, we found that ElasticZRAM shows negligible overhead, which comes from monitoring and recording the system status, including application and storage device usage.

4 EVALUATION

4.1 Experimental Platforms and Workloads

We adopt Google Pixel 6 with LineageOS 20.0 [19] (based on Android 13 and kernel 5.10) as our platform. It has an Octa-core ARM CPU, 8 GB DRAM, and 128 GB UFS 3.1 storage space. We selected widely used mobile applications, as mentioned in Table 1, and conducted experiments by following the two steps: (1) We installed the pre-selected 36 applications (10 switching applications and 26 background running applications) on the device to begin each experiment. (2) We used *adb* [20] command to collect evaluation results while performing automated tests with UI Automator [21] that emulates UI touches of users. We performed the same automated tests in ten rounds and calculated the average. The order of application switching changed randomly in each round.

Table 1: Applications and automated user interaction.

Category	Foreground Applications	Auto user inputs
Browser	Chrome	Browse/Read posts
Social Network	Facebook, Twitter	Browse/Read posts
Multimedia	YouTube, Tiktok	Watch videos
Business Utility	Amazon, CNN, Uber, Spotify	Browse and search Listen music
Game	Angry Birds	Play a stage

*Background applications: Browser (Firefox, Opera), Social Network (WhatsApp, Instagram, Skype, WeChat, LinkedIn), Multimedia (Spotify, MXPlayer, Netflix, Capcut), Online shopping (Taobao, eBay, AliPay, BOA, Paypal), Business Utility (Booking, Gmail, New York Times, BBC News, OfficeMobile, GoogleDrive), and Game (Hill Climb Racing, Boom Beach, ClashRoyale, Call of Duty).

The following five schemes are evaluated to show the advantages of ElasticZRAM: (1) **ZRAM** with 1 GB memory for swap space and *lz4* algorithm for (de)compression represents the baseline adopted by mobile systems by default, which inevitably kills processes when

the compressed memory is almost used up. (2) **Flash-swap** with a 3 GB swap partition reserved on the underlying storage device represents the traditional flash-based swapping, which stores the swapped-out pages in the local storage. (3) **ZRAM/Flash-swap** represents the original two-level memory swapping, which stores the swapped-out pages in the local storage (1 GB/3 GB). (4) **SEAL** [11] represents the state-of-the-art work, which places data accessed during app launching in the compressed memory and data accessed during app execution in the local storage (1 GB/3 GB). (5) **ElasticZRAM** is the proposed scheme in this paper (1 GB/3 GB).

4.2 Experiment Results

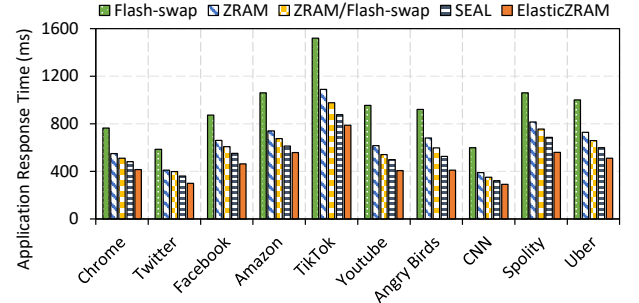


Figure 6: Application performance among different schemes.

4.2.1 Benefit on Application Performance. To show the contribution of ElasticZRAM on application performance, we repeated the launch of the pre-installed switching applications in the foreground and calculated the average application response time. Fig. 6 shows the results. First, ElasticZRAM has a noteworthy response time in all applications; it is the most effective method in that it reduces application response time by 34.5%, 32.1%, and 24.8%, compared to ZRAM, ZRAM/Flash-swap and SEAL, respectively. This is because popular applications are seldom chosen for memory reclamation with the practical interaction scenario. Therefore, the applications can instantly react to users almost at all times. As a result, ElasticZRAM surpasses flash-swap by two times in some applications, including YouTube, TikTok, and Angry Birds.

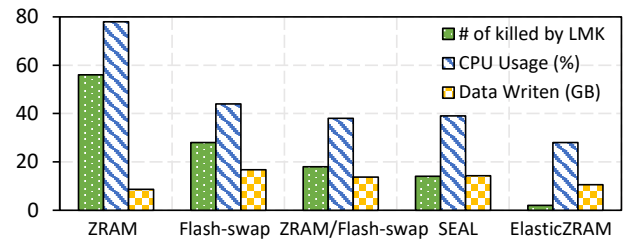


Figure 7: The number of killed applications, CPU usage and data written among different schemes.

To further estimate the integrated benefit of ElasticZRAM, the number of killed applications, CPU usage and the write traffic are collected, as shown in Fig. 7. First, although using ZRAM alone can significantly reduce the data written, the number of killed applications and CPU usage increase significantly. Using flash-swap alone, although more applications can be cached, the data written to the storage device will increase significantly. Second, the two-level memory swapping scheme balances these three indicators better. In particular, ElasticZRAM reduces the number of processes to be killed to 2 compared to traditional ZRAM/Flash-swap, while reducing CPU usage by 45%. Even compared with SEAL, the number of

killed processes and CPU overhead are significantly reduced, and the amount of data written is reduced by 24%. This is because ElasticZRAM guarantees application performance through application-aware ZRAM resizing, which can effectively reduce application kills and avoid excessive CPU usage caused by heavy ZRAM usage.

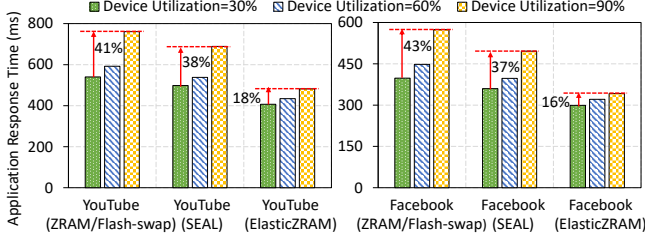


Figure 8: Application responsiveness under different device utilization.

4.2.2 Deal with Storage Performance Degradation. To understand the benefit of ElasticZRAM in dealing with storage performance degradation, we measured the response time of two representative applications under various device utilization. Fig. 8 shows the results with three device utilization. First, the performance fluctuations of traditional ZRAM/Flash-swap and SEAL are very obvious. For example, for YouTube under two-level memory swapping, the response time difference is 41% when the device utilization is 30% and 90%, and it also reaches 38% in the case of SEAL. This is because as the device utilization increases, the read/write bandwidth of the storage device decreases. The above two schemes cannot adapt to the performance degradation of the storage device, causing poor application responsiveness. Second, compared with previous work, ElasticZRAM guarantees almost stable application responsiveness. This result comes from the fact that ElasticZRAM periodically monitors device utilization and performs adaptive process path selection to avoid massive deteriorating I/O access. Based on different process paths, ZRAM is appropriately enlarged, and flash-swap is used less and less. Therefore, the impact of storage device on upper-layer memory swapping will become controllable, effectively ensuring application responsiveness and improving user experience.

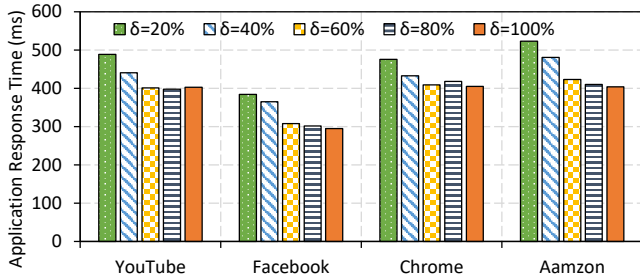


Figure 9: Application responsiveness by varying δ (the magnitude of the change in $Ratio_{zram}$).

4.3 Sensitive Study

To further understand ElasticZRAM, an additional sensitive study about the rangeability δ is performed. δ aims to control the magnitude of each change in $Ratio_{zram}$, which directly affects the percentage of pages placed in ZRAM. The initial value of ZRAM tends to be around 30%-50% (pages every app launches related) [11] and the experimental scenario is that the proportion of ZRAM needs to be increased. Fig. 9 shows the effect of changing δ from 20% to

100% on the application response time. It can be found that the time decreases as δ increases, but when δ increases to a certain percentage, it does not decrease any more. This is because while an increase in δ certainly increases the percentage of pages in ZRAM, a large portion of the application's pages are already in ZRAM. The latency of pages read from the storage device can be hidden, so it does not affect the application responsiveness. It suggests that δ is not as big as it should be but only as big as the application needs, which saves ZRAM space and satisfies the user experience.

5 CONCLUSION

In this paper, we redesign the swapping subsystem on mobile devices combines the access characteristics of applications and performance characteristics of storage devices to adaptively guide the swapping of applications between ZRAM and storage through different process paths. Experimental results demonstrate that ElasticZRAM effectively improves application responsiveness.

ACKNOWLEDGE

This work is supported by the NSFC (62072177, 62141213), CCF-Huawei Populus Grove Fund (CCF-HuaweiSY202204), and Shanghai Science and Technology Project (22QA1403300). The corresponding author is Liang Shi (shi.liang.hk@gmail.com).

REFERENCES

- [1] S.-H. Kim, J. Jeong, and J.-S. Kim, "Application-aware swapping for mobile systems," *TECS*, 2017.
- [2] "Low memory killer daemon: Android developers docs," 2023, <https://source.android.com/docs/core/perf/lmkd>.
- [3] C. Li, L. Shi, and C. J. Xue, "Mobileswap: Cross-device memory swapping for mobile devices," in *DAC*, 2021.
- [4] A. Developers, "Low memory management," 2023, <https://developer.android.com/topic/performance/memory-management>.
- [5] S. Son, S. Y. Lee, Y. Jin, J. Bae, J. Jeong, T. J. Ham, J. W. Lee, and H. Yoon, "ASAP: fast mobile application switch via adaptive prepagging," in *USENIX ATC*, 2021.
- [6] "Lmk: kill reasons," 2023, <https://android.googlesource.com/platform/system/memory/lmkd/+refs/heads/master/lmkd.cpp>.
- [7] M. Arbaz, "Ufs 4.0 vs ufs 3.1 vs ufs 3.0 | speed comparison," 2023, <https://www.thephonetalks.com/ufs-4-0-vs-ufs-3-1-vs-ufs-3-0comparison/>.
- [8] X. Zhu, D. Liu, K. Zhong, J. Ren, and T. Li, "Smartswap: High-performance and user experience friendly swapping in mobile systems," in *DAC*, 2017.
- [9] N. Lebeck, A. Krishnamurthy, H. M. Levy, and I. Zhang, "End the senseless killing: Improving memory management for mobile operating systems," in *USENIX ATC*, 2020.
- [10] W. Li, L. Shi, H. Li, C. Li, and E. H.-M. Sha, "Iosr: Improving i/o efficiency for memory swapping on mobile devices via scheduling and reshaping," *TECS*, 2023.
- [11] C. Li, L. Shi, Y. Liang, and C. J. Xue, "Seal: User experience-aware two-level swap for mobile devices," *TCAD*, 2020.
- [12] Y.-X. Wang, C.-H. Tsai, and L.-P. Chang, "Killing processes or killing flash? escaping from the dilemma using lightweight, compression-aware swap for mobile devices," *TECS*, 2021.
- [13] G. Lim, D. Kang, M. Ham, and Y. I. Eom, "Swam: Revisiting swap and oom for improving application responsiveness on mobile devices," in *MobiCom*, 2023.
- [14] SamMobile, "Ram plus: Samsung's extra ram feature is arriving on more phones, both mid-range and flagship," 2023, <https://www.sammobile.com/news/samsung-galaxy-a52s-5g-virtual-plus-feature/>.
- [15] "Huawei's memory expansion technology: 8gb ram works as 10gb and 12gb ram as 14gb: Huawei community," 2023, https://consumer.huawei.com/aen/community/details/Huawei-s-Memory-Expansion-Technology-8GB-RAM-works-as-10GB-and-12GB-RAM-as-14GB/topicId_121377/.
- [16] "All Xiaomi that will have the RAM expansion function," 2023, <https://www.xiaomist.com/2021/07/these-are-all-xiaomi-that-can-already.html>.
- [17] T. Zhang, A. Zuck, D. E. Porter, and D. Tsafir, "Apps can quickly destroy your mobile's flash: why they don't, and how to keep it that way," in *MobiSys*, 2019.
- [18] B. Gu, L. Luo, Y. Lv, C. Li, and L. Shi, "Dynamic file cache optimization for hybrid ssds with high-density and low-cost flash memory," in *ICCD*, 2021.
- [19] "Build lineageos for google pixel 6," 2023, <https://wiki.lineageos.org/devices/oriole/build>.
- [20] A. Developers, "Android debug bridge(adb)," 2023, <https://developer.android.com/studio/command-line/adb>.
- [21] Xiaocong, "uiautomator," 2023, <https://github.com/xiaocong/uiatutorator>.