International Journal of Internet, Broadcasting and Communication Vol.15 No.1 79-84 (2023)
http://dx.doi.org/10.7236/IJIBC.2023.15.1.79

IJIBC 23-1-10

Memory Allocation in Mobile Multitasking Environments with Real-time Constraints

Hyokyung Bahn

Professor, Department of Computer Engineering, Ewha University, Korea
bahn@ewha.ac.kr

Abstract

Due to the rapid performance improvement of smartphones, multitasking on mobile platforms has become an essential feature. Unlike traditional desktop or server environments, mobile applications are mostly interactive jobs where response time is important, and some applications are classified as real-time jobs with deadlines. When interactive and real-time jobs run concurrently, memory allocation between multitasking applications is a challenging issue as they have different time requirements. In this paper, we study how to allocate memory space when real-time and interactive jobs are simultaneously executed in a smartphone to meet the multitasking requirements between heterogeneous jobs. Specifically, we analyze the memory size required to satisfy the constraints of real-time jobs and present a new model for allocating memory space between heterogeneous multitasking jobs. Trace-driven simulations show that the proposed model provides reasonable performance for interactive jobs while guaranteeing the requirement of real-time jobs.

Keywords: *Mobile platform, Android, real-time job, smartphone, memory allocation, multitasking*

1. Introduction

Mobile platforms such as Android basically support multitasking functions as they adopt the Linux kernel at the bottom of the software architecture [1, 2]. However, in the early days of Android, only one foreground application that occupied the screen was activated except for some specific functions such as playing music or downloading files. This is because the screen of a smartphone device is small, so it is difficult to show multiple applications at the same time. Also, there are restrictions in realizing complete multitasking features as the smartphone's memory capacity is limited and virtual memory swapping is not supported.

However, due to the rapid performance improvement of smartphones, multitasking on mobile platforms has become a necessity. The latest Android reference phone, the Pixel 6 pro, has 12GB of memory and 8 CPU cores, which is sufficient for multitasking [3]. Virtual memory swapping can also be activated by making use of fast storage media like Optane™ [4, 5]. Smartphone screens have also become large enough with the advent of foldable functions, allowing multiple applications to be displayed on one screen. This situation clearly shows that smartphones are increasingly moving towards multitasking devices.

Manuscript Received: December. 5, 2022 / Revised: December. 7, 2022 / Accepted: December. 12, 2022
Corresponding Author: bahn@ewha.ac.kr
Tel: +82-2-3277-2368, Fax: +82-2-3277-2306
Professor, Department of Computer Engineering, Ewha University, Korea

Meanwhile, unlike conventional desktop and server applications, smartphone applications are mostly interactive jobs where response time is important, and some applications are classified as real-time jobs with deadlines [6]. In general-purpose systems like desktops and servers, batch jobs that do not need prompt results coexist with interactive jobs that must respond immediately to user input. Thus, it is possible to efficiently schedule the two different types of jobs by appropriately assigning priorities. However, in the smartphone environment, most of the jobs have time constraints, so a mechanism that can manage latency well during multitasking is needed. When interactive and real-time jobs run concurrently, CPU scheduling and memory allocation are the two significant issues. In CPU scheduling, basic issues can be resolved by allocating dedicated cores for real-time jobs as the number of cores increases in modern smartphone systems [7]. However, in memory allocation, as time progresses after booting, the number of multitasking applications consistently increases, and free memory space is finally exhausted [8].

In this paper, we study how to allocate memory space when real-time and interactive jobs are simultaneously executed in a smartphone to meet the requirements of multitasking between heterogeneous jobs. Currently, in most smartphone environments such as Android, all jobs in the system compete for memory allocation without special considerations for real-time jobs. However, such a method results in deadline misses of real-time jobs seriously as the number of multitasking applications increases [9].

In order to resolve this issue, this paper analyzes and models the memory size required to satisfy the time constraints of real-time jobs, and presents a novel method for allocating memory space between multitasking jobs. To do so, this paper extracts storage access traces of different applications and validates the effectiveness of the proposed model through trace-driven simulations. Specifically, we compare our model with the real-time OS method, where the entire footprint of real-time jobs resides in memory and the general-purpose OS method, which uses global memory allocation without special cares for real-time jobs. As a result of our experiment, we show that the proposed model provides reasonable performance for interactive jobs while guaranteeing the time constraints of real-time jobs. In contrast, the real-time OS method degrades the memory hit ratio of interactive jobs significantly, and the general-purpose OS method could not guarantee the time constraints of real-time jobs.

2. Modeling and Memory Allocation for Real-time Jobs

In order to meet the deadlines of real-time jobs, it is important to ensure that all required data to be accessed are loaded into memory before they are actually used. This implies that it is necessary to allocate enough memory space to accommodate the current working set of real-time jobs in memory [10]. For example, when a multimedia player is executed, if the requested data needs to be read from the storage frequently due to lack of memory, it will be difficult to play the video seamlessly. In contrast, if necessary data can be read into memory in advance, the problem can be resolved.

In traditional embedded systems dedicated to some fixed real-time jobs, sufficient memory space to accommodate the entire data necessary for the jobs is equipped so that storage access does not occur during the executions [11]. However, this is difficult to apply in smartphone environments where various kinds of jobs that are not pre-determined are executed at any time. As the memory capacity increases, the storage access of the workload decreases, leading to performance improvements. When the memory capacity reaches a certain level, the improvement decreases, and at some point, storage access can no longer be reduced even if more memory space is provided. It is known that this can be modeled as an exponential function with two control parameters [12]. The control parameters of this modeling function are varied according to the popularity bias of the workload's data. In this paper, we precisely determine the model parameters depending on the

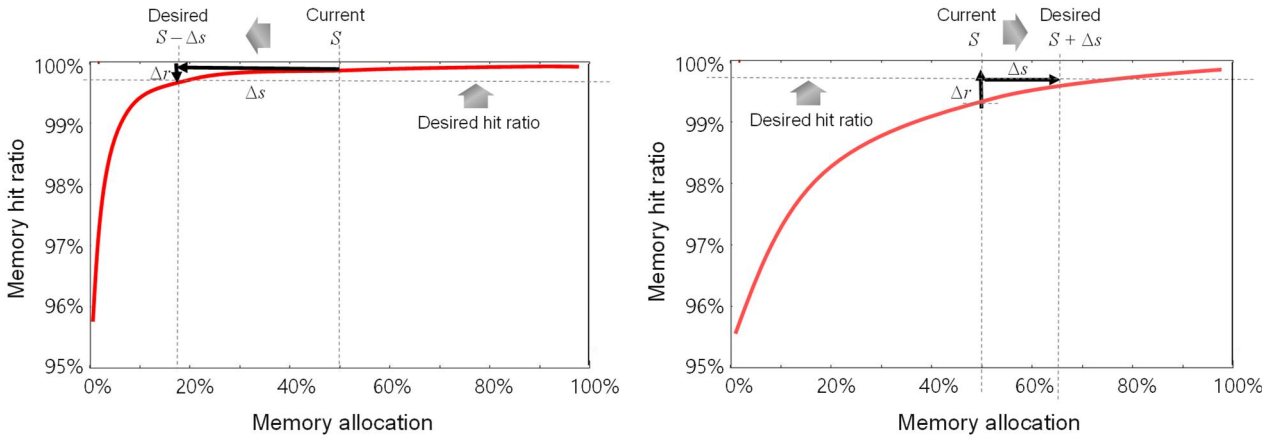


Figure 1. Memory allocation method of the proposed model; (a) when the memory hit ratio is higher than the desired hit ratio, our model decreases the allocated memory size; (b) when the memory hit ratio is lower than the desired hit ratio, our model increases the allocated memory size.

characteristics of real-time jobs to be executed, so that the memory hit ratio for the given memory capacity and workloads can be predicted well.

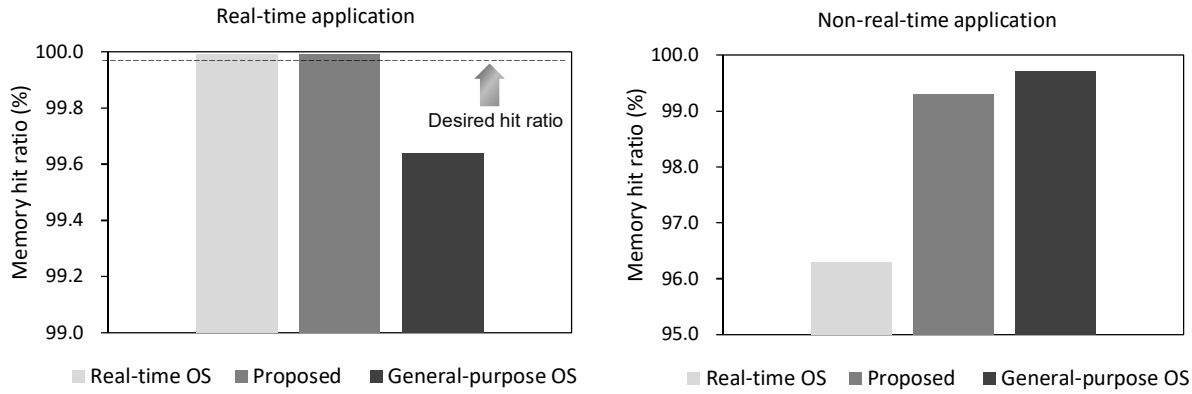
In order to fully meet the time requirement of real-time jobs, storage accesses should be completely eliminated during their executions. This requires the memory capacity to load the entire footprint of real-time jobs at the same time, which wastes excessive memory capacity in multitasking environments. To cope with this situation, this paper sets a desired memory hit ratio for real-time jobs to be executed and a memory size that can satisfy this ratio is allocated. In reality, for soft real-time jobs like video playback executed on smartphones, a small percentage of deadline misses can be allowed, and thus our approach is a realistic solution in practical systems.

Fig. 1 shows the estimated memory performance of real-time jobs as the allocated memory size is varied in the proposed model. The red curve shows the memory hit ratio according to the allocated memory size relative to the workload's footprint. As shown in Figs. 1(a) and 1(b), the slope of the curve appears differently according to the access bias of real-time job's data. Suppose that the memory capacity allocated to the real-time job is 50% and the memory hit ratio at that point is higher than the desired hit ratio as shown in Fig. 1(a) (or lower than the desired hit ratio as shown in Fig. 1(b)). Then, the proposed model satisfies the demand of real-time jobs through decreasing (or increasing) the memory allocation by Δs as shown in Fig.1 (a) (or Fig. 1(b)).

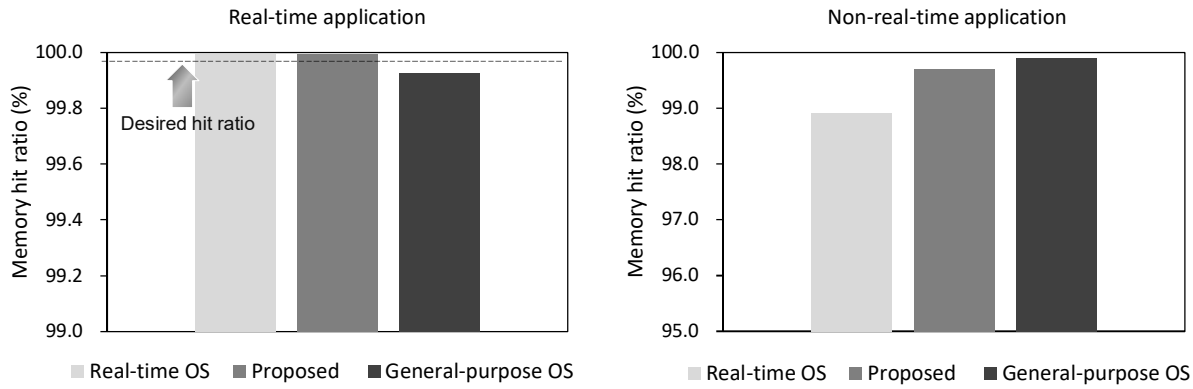
Meanwhile, compared to traditional real-time systems that keep all footprints in memory for the entire duration of the real-time job's execution, our approach allocates the minimum amount of memory space to satisfy the performance requirement of real-time jobs through periodic monitoring, thereby improving the efficiency of memory allocation in multitasking environments.

3. Performance Evaluations

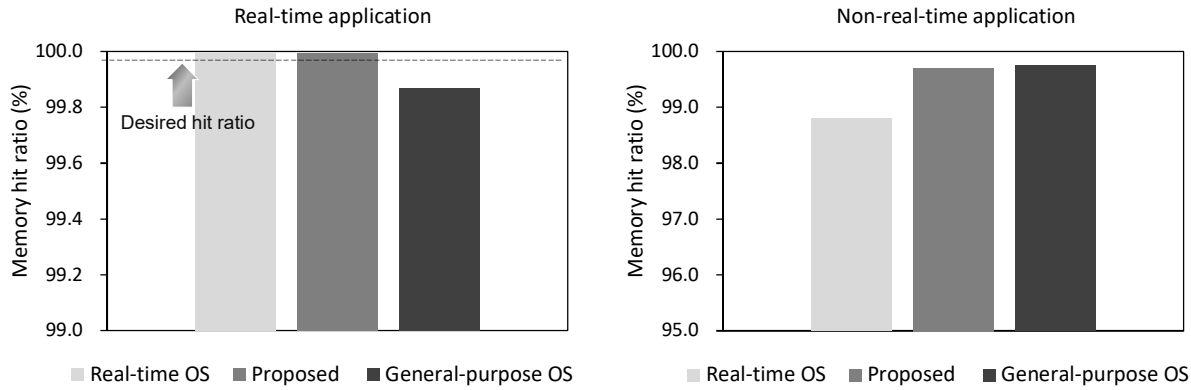
In this section, we validate the performance of the proposed model through simulations with file I/O traces extracted while Android applications are executed. Six popular applications were used for trace extractions: video player, real-time messenger, video game, map application, email client, and social network. Although



(a) Hit ratio of the three methods when video game (left) and email client (right) run together.



(b) Hit ratio of the three methods when video player (left) and social network (right) run together.



(c) Hit ratio of the three methods when real-time messenger (left) and map app (right) run together.

Figure 2. Comparison of real-time OS, general-purpose OS, and the proposed model when multitasking of real-time app (left) and non-real-time app (right).

response time is important in all these applications, we classify them into two categories based on the existence of deadline requirements: video player, video game, and real-time messenger applications are classified as real-time jobs as they have deadlines, whereas map application, email client, and social network applications are classified as non-real-time jobs as they do not have deadlines. Considering the characteristics of the current smartphone system where only one foreground application is activated at a time, we conduct multitasking

experiments with two concurrent applications, one real-time application and one non-real-time application competing for memory space.

We developed a functional simulator that has the ability to evaluate the effectiveness of memory allocation methods with the exact memory and storage situations in a mobile device. We assume the Android reference device Pixel 6 pro to mimic multitasking situations, and decide the memory size for each situation based on the footprint of applications we simulate. Specifically, the simulator replays the collected traces and when the requested content is not in the memory, we simulate storage I/O activities. For NAND flash memory, we use the parameters of Samsung UFS 3.1, of which the read and write performances are 100,000 IOPS and 70,000 IOPS for 4 KB, respectively. The size of a page is set to 4 KB as is typically used in Android and Linux. For comparison with the proposed memory allocation model, we further experiment traditional real-time OS in which the entire footprint of real-time jobs resides in memory and the general-purpose OS that competes for memory without considering the characteristics of real-time tasks.

Fig. 2 shows the memory hit ratio for real-time and non-real-time jobs according to the memory allocation strategies. Specifically, Figs. 2(a), 2(b), and 2(c) show the concurrent executions of video game and email client, video player and social network, and real-time messenger and map application, respectively. As shown in the figure, the proposed model guarantees the desired memory hit ratio for real-time jobs in all cases. In the traditional real-time OS method, the performance of the real-time job exhibits the best, but it reduces the memory allocation for non-real-time jobs excessively in order to accommodate the entire footprint of the real-time job in memory. As a result, the memory hit ratio of non-real-time jobs is degraded significantly compared to the proposed model.

On the other hand, the general-purpose OS method could not guarantee the desired memory hit ratio of real-time jobs. This is because all kinds of applications compete for equal priority under general-purpose OS. As shown in the figure, there was not a large difference in the memory hit ratio between real-time and non-real-time jobs in the general-purpose OS method, and as a result, the memory hit ratio for non-real-time jobs is significantly improved compared to the other two methods. When comparing our approach with general-purpose OS, although the performance of non-real-time jobs is slightly better in general-purpose OS, our approach exhibits significantly better results for real-time jobs. This is important as the performance of real-time jobs should be evaluated based on the satisfaction of their requirements such as the desired hit ratio.

4. Conclusions

This paper analyzed the requirements of multitasking between different kinds of applications in smartphones and presented a memory allocation model for real-time and non-real-time jobs. Smartphone platforms such as Android support multitasking but applications should compete for memory space without special consideration for real-time requirements, meeting the demands of real-time jobs difficult. In contrast, traditional real-time systems should equip enough memory space to accommodate the entire data needed, so that storage access does not occur during the execution of real-time jobs. We observed that this is difficult to utilize in smartphone platforms in which the number of applications executed concurrently cannot be known in advance. To cope with this situation, this paper presented a memory performance model that accurately estimates the memory hit ratio for the given memory size, and presented a memory allocation strategy for multitasking workloads consisting of real-time and non-real-time jobs based on this model. Through trace-driven simulations, we showed that the proposed model provides reasonable performance for non-real-time applications while ensuring the desired level of memory hit ratio for real-time applications.

Acknowledgement

This work was supported in part by the NRF (National Research Foundation of Korea) grant (No. 2019R1A2C1009275) and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant (No.RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)) funded by the Korean government (MSIT).

References

[1] Android, https://www.android.com.
[2] F. Khomh, H. Yuan, and Y. Zou, "Adapting Linux for mobile platforms: An empirical study of Android," 28th IEEE Int'l Conf. Software Maintenance (ICSM), pp. 629-632, 2012.
DOI: https://doi.org/10.1109/ICSM.2012.6405339
[3] Google Pixel 6 Pro, https://store.google.com/gb/product/pixel_6_pro
[4] J. Kim and H. Bahn, "Analysis of Smartphone I/O Characteristics — Toward Efficient Swap in a Smartphone," IEEE Access, vol. 7, pp. 129930-129941, 2019.
DOI: https://doi.org/10.1109/ACCESS.2019.2937852
[5] Intel Optane™ Technology, https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html
[6] J. Kim and H. Bahn, "Maintaining Application Context of Smartphones by Selectively Supporting Swap and Kill," IEEE Access, vol. 8, pp. 85140-85153, 2020.
DOI: https://doi.org/10.1109/ACCESS.2020.2992072
[7] S. Yoo, Y. Jo, and H. Bahn, "Integrated Scheduling of Real-Time and Interactive Tasks for Configurable Industrial Systems," IEEE Trans. on Industrial Informatics, vol. 18, no. 1, pp. 631-641, 2022.
DOI: https://doi.org/10.1109/TII.2021.3067714
[8] Y. Park and H. Bahn, "Challenges in memory subsystem design for future smartphone systems," IEEE Inte'l Conf. Big Data and Smart Computing (BigComp), pp. 255-260, 2017.
DOI: https://doi.org/10.1109/BIGCOMP.2017.7881707
[9] S. Yoon, H. Park, K. Cho, and H. Bahn, "Supporting Swap in Real-Time Task Scheduling for Unified Power-Saving in CPU and Memory," IEEE Access, vol. 10, pp. 3559-3570, 2022.
DOI: https://doi.org/10.1109/ACCESS.2021.3140166
[10] S. Nam, K. Cho, and H. Bahn, "Tight Evaluation of Real-Time Task Schedulability for Processor's DVS and Nonvolatile Memory Allocation," Micromachines, vol. 10, no. 6, 2017.
DOI: https://doi.org/10.3390/mi10060371
[11] X. Cheng, Y. Guan, and Y. Zhang, "Design and Implementation of Dynamic Memory Allocation Algorithm in Embedded Real-Time System," Int'l Conf. Pioneering Computer Scientists, Engineers and Educators, pp. 539-547, 2018.
DOI: https://doi.org/10.1007/978-981-13-2203-7_43
[12] S. Lee and H. Bahn, "Characterization of Android Memory References and Implication to Hybrid Memory Management," IEEE Access, vol. 9, pp. 60997-61009, 2021.
DOI: https://doi.org/10.1109/ACCESS.2021.3074179