LETTER

# Dual Management of Real-Time and Interactive Jobs in Smartphones

**Eunji LEE**[†], **Youngsun KIM**[†], ***Nonmembers***, ***and*** **Hyokyung BAHN**[†a)], ***Member***

**SUMMARY** A dual management of real-time and interactive jobs in dual-core smartphones is presented. The proposed scheme guarantees the end-to-end QoS of real-time applications, while also provides reasonable latency for interactive applications. To this end, high performance NVRAM is adopted as storage of real-time applications, and a dual purpose CPU scheduler, in which one core is exclusively used for real-time applications, is proposed. Experiments show that the proposed scheme reduces the deadline miss ratio of real-time applications by 92%.

***key words:*** *real-time job, smartphone, NVRAM, scheduling*

## 1. Introduction

With the enhanced computational abilities of smartphones and the subsequent explosion of mobile applications, multitasking is becoming a common feature of smartphones. In particular, not only real-time applications (e.g., video player), but also interactive applications (e.g., web browser) are supported together in smartphones. Unfortunately, current smartphone platforms do not support these heterogeneous applications efficiently. Android is a widely used platform for smartphones, which runs on top of Linux. Since Linux has been developed as a general-purpose OS to manage best-effort applications, it does not support deadline-guaranteed services. Specifically, there are two sources of unpredictable latency in the program execution path of smartphones. The first one is the sporadic execution of garbage collection performed in NAND flash memory and the second one is the time-quantum-based scheduler used in Android platforms.

This letter eliminates the aforementioned unpredictable latency and guarantees the end-to-end QoS of real-time applications by two architectural supports with novel mechanisms. First, we adopt high performance NVRAM such as PCM (phase-change memory) or STT-MRAM (spin torque transfer magnetic RAM) as the storage of real-time applications. Unlike flash memory, as its underlying structure does not require garbage collection but provides small and constant access latency, NVRAM ensures predictable I/O performance for real-time tasks [1], [2]. Second, we propose a dual purpose CPU scheduler, in which one core is exclusively used for a real-time job. This scheduler enables the real-time job to have CPU control even during the I/O execution, thereby resuming its CPU execution right after the I/O completion. Though CPU cycles may be wasted during the I/O execution of the real-time job, we show that the CPU utilization is not degraded largely as NVRAM provides sufficiently fast I/O. Experimental results show that the proposed scheme reduces the deadline miss ratio of real-time applications by 92% on average compared to conventional blocking I/O used in Android.

The remainder of this letter is organized as follows. Section 2 describes the details of the mechanism we propose for real-time tasks in smartphones. Section 3 presents the experimental results to assess the effectiveness of the proposed scheme, and Sect. 4 finally concludes this letter.

## 2. The Proposed Scheme

In Android, when a user application requests a file I/O, it invokes a specific framework API to access storage. Then, the API transmits the request to kernel through system calls, and the kernel subsequently conveys the I/O request to storage through device drivers and controllers. Then, the kernel blocks the calling application and performs context-switch to execute other tasks during I/O processing. When the I/O is completed, the device controller notifies it to the kernel by an interrupt, and the kernel transfers the requested data to the buffer space if any and wakes up the blocked process. After waking up, however, even a high priority process like real-time tasks cannot obtain CPU instantly because the current Android scheduler, CFS (completely fair scheduler), schedules CPU non-preemptively. That is, even a real-time task should wait in the ready-queue until the time quantum of the currently running task expires [3]. The time quantum is about 1 millisecond, which is long time considering the access time of NVRAM, which is only tens of nanoseconds [4]. Furthermore, as this latency varies depending on the remaining time slice of the currently running task, the real-time task may suffer from unpredictable delays.

NAND flash memory is another source of unpredictable latency in I/O paths. As NAND flash memory is an erase-before-write medium, it needs garbage collection, which collects invalid pages and erases them to recycle their spaces [5], [6]. As garbage collection is triggered sporadically and takes long time due to bulky erase operations, it results in the fluctuation of I/O time in smartphones. As Table 1 shows, an erase operation takes an order of magnitude longer latency compared to other operations.

This letter eliminates all sources of the aforementioned unpredictable latency by making use of dual purpose

Manuscript received September 17, 2013.

[†]The authors are with the Department of Computer Engineering, Ewha University, Seoul, 120–750 Korea.

a) E-mail: bahn@ewha.ac.kr

DOI: 10.1587/transinf.E97.D.323

**Table 1** Performance characteristics of memory and storage media.

| Type | HDD | PCM | Flash |
|---|---|---|---|
| Read time | 10ms | 50-100ns | 60us |
| Write time | 10ms | 400-500ns | 800us |
| Ease time | N/A | N/A | 1.5ms |

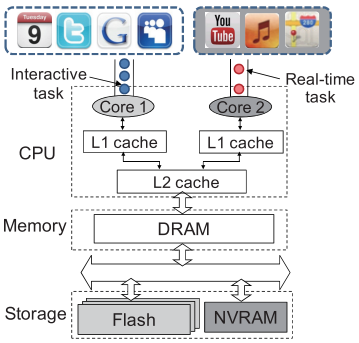


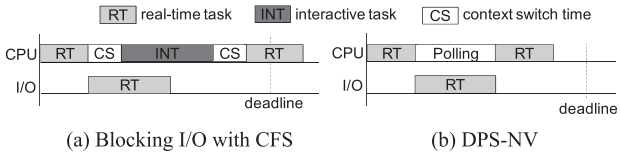

**Fig. 1** The proposed system architecture.

scheduling with NVRAM (DPS-NV). Figure 1 shows the architecture of DPS-NV. Both NVRAM and flash memory are adopted as storage. NVRAM is dedicated to real-time applications and their data, whereas all the other programs and data are located in flash partitions. This architecture not only prevents unexpected I/O delay of real-time tasks, but also minimizes the latency caused by I/O bus contention. The CPU of our architecture consists of two cores. We use one core for a real-time task, and the other for interactive tasks. However, if no real-time task is active, both cores can be allocated to interactive tasks like conventional systems. The L1 cache is private to each core and the L2 cache is shared among two cores.

In our architecture, DRAM is used as main memory media like the conventional system architectures. Since NVRAM is byte-accessible and sufficiently fast compared to DRAM, there is a prospect that NVRAM might unify the functionalities of main memory and storage, eventually leading to a unified memory architecture [7], [8]. In this case, a real-time task can be executed in its place without moving to DRAM, and thus more accurate prediction of I/O time can be possible. However, many components of the system including operating system kernel should be redesigned to support this functionality. Thus, we use NVRAM only for storage media.

This architecture also resolves the scheduling latency incurred by the current blocking I/O mechanism. The fundamental principle behind the blocking I/O is to improve CPU utilization by executing other tasks while performing I/O because storage I/O takes millions of cycles in conventional systems. However, it cannot provide predictable I/O execution for real-time tasks. DPS-NV relieves this problem by using non-blocking I/O for real-time applications. It allows the running task to control CPU during its I/O, and thus avoids long scheduling latency.

Though a certain CPU time is wasted with non-blocking I/O, it is not that much as the access time of NVRAM is an order of magnitude shorter than flash memory. Figure 2 depicts the comparison of the conventional blocking I/O and DPS-NV. As shown in the figure, DPS-NV meets the deadline of the real-time task, but blocking I/O fails to do so due to the context-switch overhead and the scheduling latency.




(a) Blocking I/O with CFS (b) DPS-NV

**Fig. 2** Program execution in blocking I/O with CFS and DPS-NV.

In DPS-NV, CPU is under-utilized during the I/O time of real-time tasks, but the wasted time can be offset by the saved context-switch overhead due to the high performance characteristics of NVRAM. Furthermore, as the core allocated to the real-time task can be used by other tasks when it is not active, degradation of the CPU utilization is limited.

## 3. Experimental Results

To assess the effectiveness of DPS-NV, we performed trace-driven simulations. The target architecture is Odroid A4 open mobile development platform running with Android 4.0.4 ICS. The traces were obtained with a system profiler implemented on Linux 2.6.32 by playing the StarWars movie with smplayer. Smplayer plays a series of movie frames periodically (e.g., 24 frames/sec) and in each period, it reads, decodes, and displays frames from the movie file, and then sleeps until the next period starts. The developed simulator measures the completion time of the movie player for each period. In this experiment, the context-switch time is set to 15 us [9]. We use PCM as the NVRAM storage since PCM is expected to be used as secondary storage like flash memory or hard disks in the next few years [2].

Figure 3 (a) shows the deadline miss ratio of the real-time task as a function of the cache size. As shown in the figure, DPS-NV reduces the deadline miss ratio of blocking I/O by 92% on average. Specifically, DPS-NV guarantees zero percent of deadline miss ratio when the cache size is 7 K pages, while blocking I/O incurs 48% of deadline miss ratio on that point. Figure 3 (b) shows the deadline miss ratio according to the variations of the waiting time in the ready-queue. CPU waiting time of 100% implies the situation that a running process uses CPU for its full time quantum when the real-time task arrives at the ready-queue. We set the time quantum to 1 millisecond following the conventional setting [4]. The deadline miss ratio of blocking I/O exponentially increases as the waiting time becomes longer, whereas DPS-NV provides consistently good performance. Note that the deadline miss ratio rises up to 84% even when the waiting time is only 20% in the blocking I/O.

Figure 4 shows the actual completion time of the real-time task in each cycle as time progresses. DPS-NV satis-

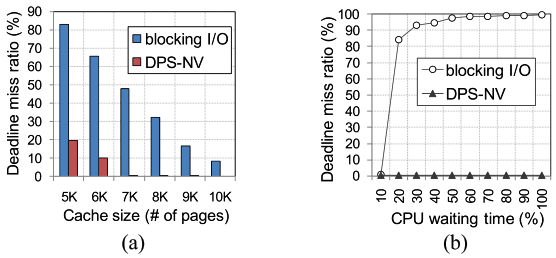



**Fig. 3** Performance comparison of DPS-NV and blocking I/O; (a) deadline miss ratio as the cache size is varied; (b) deadline miss ratio as the CPU waiting time is varied.

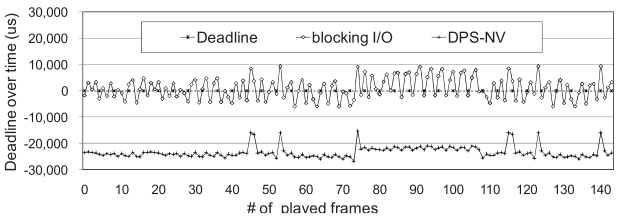



**Fig. 4** Deadline over time.

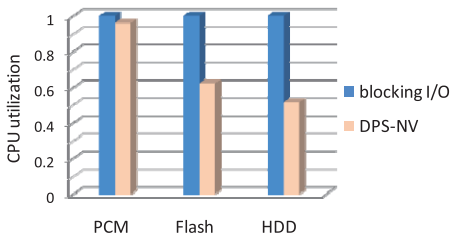



**Fig. 5** CPU utilization for different storage.

fied the deadlines in all periods and completed the real-time task 58% earlier than its deadline on average. However, blocking I/O incurred 78 times of deadline misses while playing 150 frames and delayed the completion time by 22% on average.

As the real-time task does not release CPU during its I/O, the performance of other tasks may be degraded. To investigate this effect, the CPU utilization of DPS-NV is measured. For the comparison purpose, we also measure the CPU utilization of DPS-NV coupled with flash memory and hard disks whose performance characteristics are described in Table 1. As shown in Fig. 5, the CPU utilization of DPS-NV is reduced by only 4% in PCM compared to blocking I/O, but 38% and 49% in flash memory and hard disks, respectively. When using slow storage media, I/O processing consumes considerable CPU cycles, and thus occupying CPU during I/O processing might result in serious performance degradation. However, with high performance storage like PCM, the wasted CPU cycles of DPS-NV are not large and most of them can be offset by the saved context-switch time.

## 4. Conclusion

DPS-NV is an end-to-end QoS support mechanism for real-time applications in smartphones. DPS-NV eliminates two sources of unpredictable latency in the program execution paths of smartphones: non-uniform access time of flash storage and the scheduling latency of the completely fair scheduler. Specifically, DPS-NV adopts NVRAM as the storage of real-time applications, and then performing non-blocking I/O. Experimental results showed that DSP-NV reduces the deadline miss ratio of real-time applications by 92% on average.

## Acknowledgments

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2011-0028825).

### References

[1] S. Lee, H. Bahn, and S.H. Noh, "Characterizing memory write references for efficient management of hybrid PCM and DRAM memory," Proc. IEEE Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS), Singapore, pp.168–175, 2011.

[2] E. Lee, S. Yoo, J. Jang, and H. Bahn, "WIPS: A write-in-place snapshot file system for storage-class memory," Electron. Lett., vol.48, no.17, pp.1053–1054, 2012.

[3] S. Wang, Y. Chen, W. Jiang, P. Li, T. Dai, and Y. Cui, "Fairness and interactivity of three CPU schedulers in Linux," Proc. IEEE Conf. Embedded and Real-Time Computing Systems and Applications, pp.172–177, 2009.

[4] B.B. Brandenburg, J.M. Calandrino, and J.H. Anderson, "On the scalability of real-time scheduling algorithms on multicore platforms: A case study," Proc. IEEE Real Time Systems Symposium (RTSS), 2008.

[5] J. Kim, J.M. Kim, S.H. Hoh, S.L. Min, and Y. Cho, "A space efficient flash translation layer for compact flash systems," IEEE Trans. Consum. Electron., vol.48, no.2, pp.366–375, 2002.

[6] L. Chang, T. Kuo, and S. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," ACM Trans. Embedded Computing Systems, vol.3, no.4, pp.837–863, 2004.

[7] J. Condit, E.B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," Proc. ACM Symp. Operating Systems Principles (SOSP), 2009.

[8] X. Wu and A.L.N. Reddy, "SCMFS: A file system for storage class memory," Proc. IEEE Conf. Supercomputing (SC), 2011.

[9] H.S. Choi and H.C. Yun, "Context switching and IPC performance comparison between uClinux and linux on the ARM9 based processor," SAMSUNG Tech. Conf., 2005.