A Buffer Cache Algorithm Using the Characteristic of Mobile Applications based on Hybrid Memory Architecture

Chansoo Oh
Hanwha Techwin
6, Pangyo-ro 319
Gyeonggi-do, Korea
+82-31-290-7219
Chansoo.oh@skku.edu

Dong Hyun Kang
Sungkyunkwan University
300 Cheoncheon-dong
Suwon, Korea
+82-31-290-7219
kkangsu@skku.edu

Minho Lee
Sungkyunkwan University
300 Cheoncheon-dong
Suwon, Korea
+82-31-290-7219
minhozx@skku.edu

Young Ik Eom
Sungkyunkwan University
300 Cheoncheon-dong
Suwon, Korea
+82-31-290-7219
yieom@skku.edu

ABSTRACT

In general, current mobile devices employ a buffer cache mechanism which has been designed to mitigate the performance gap between CPU and storage. However, it cannot fit in mobile devices because it does not consider the characteristics of the mobile applications (e.g., foreground or background state). Therefore, existing buffer cache mechanisms can cause performance degradation by I/O interference between foreground applications and background applications. In addition, DRAM accelerates energy consumption by performing periodic refresh operations. Therefore, DRAM-based memory system can cause critical power problem due to its limited battery capacity in the mobile devices. In this paper, we propose a novel buffer cache algorithm for mobile devices based on hybrid memory architecture composed of DRAM and non-volatile PCM. Proposed algorithm is based on key observation that background applications rarely issue I/O requests and they may interfere with foreground applications. For evaluation, we implemented the proposed algorithm and compared the performance with two widely known algorithms. Our experimental results show that our algorithm reduces the elapsed time of the foreground applications by 70% on average and the power consumption by 67% on average while operating same workloads.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles; D.4.2 [**Operating Systems**]: Storage Management

General Terms

Algorithms, Management, Performance, Design

Keywords

Hybrid memory architecture, buffer cache algorithm, mobile device, foreground application, background application.

IMCOM '16, January 04-06, 2016, Danang, Viet Nam

DOI: http://dx.doi.org/10.1145/2857546.2857575

1. INTRODUCTION

Mobile devices need gigabytes of memory capacity to run several applications simultaneously and the memory system is used to store data for these applications. Mobile applications can be classified into two states according to its operating environments such as foreground state and background state. Foreground application is the application which is currently being used by user and the users can perceive performance of the foreground applications as the performance of the mobile device. On the other hand, user could not recognize the performance of background application and background applications are on sleep mode in background state. Also, mobile application includes several processes and these processes can be classified into several types in accordance with its working state. Despite the different characteristics of applications and processes, most of previous buffer management algorithms for mobile device do not consider the characteristics of mobile applications. For these reasons, we try to exploit the characteristics of mobile applications to improve previous buffer management algorithms.

Also, the traditional DRAM-based memory architectures for mobile device have weaknesses in power issue because DRAM (Dynamic Random Access Memory) periodically needs refresh operations that consume battery power of the device to maintain data in DRAM. In order to mitigate the weakness of DRAM, many researchers focused on the non-volatile memory, such as PCM (Phase Change Memory), to take the benefits of lower power consumption over DRAM. However, PCM has slower read/write latency than DRAM and it has limited lifecycle [Table 1]. To complement these disadvantages of PCM, some researchers proposed buffer cache algorithms that efficiently reduce the number of write requests on PCM by using hybrid main memory architecture, which consists of DRAM and PCM [1, 2, 3]. However, previous researches for hybrid memory architecture have only been focused on the general computing environment and never consider the various states of the mobile application, such as foreground and background.

Table 1. Characteristics of DRAM and PCM [4]

	DRAM	**PCM**
Read Latency	50ns	50~100ns
Write Latency	~20-50ns	~1us
Read Energy	~0.1nJ/b	0.1nJ/b
Write Energy	~0.1nJ/b	~1mJ/b
Static Power	~W/GB	<<0.1W
Endurance	∞	10^8

In this paper, we propose a novel buffer cache algorithm which saves operational energy and improves the performance of foreground applications based on hybrid memory architecture composed of DRAM and PCM. Our algorithm is motivated by key observation that background applications rarely issue I/O requests and they may interfere with a foreground application. Based on this observation, we have classified processes into critical process and non-critical process as a first step of our algorithm. We explain details of the classification for mobile application in Section 2. Then proposed algorithm allocates the pages for critical processes in DRAM and non-critical processes in PCM because non-critical processes have no direct impact on the user experience [5]. For evaluation, we implemented our algorithm and compared its performance against two representative algorithms, such as traditional LRU (Least Recently Used) and 2QLRU (Two Queue Least Recently Used) [6]. Our experimental results clearly show that our algorithm reduces the elapsed time of the foreground applications by 70% on average and the power consumption by 67% on average.

The remainder of the paper is organized as follows. Section 2 explains the related works, such as PCM and Android system. Then we present design of our algorithm in Section 3. Section 4 evaluates its performance while comparing it with previous buffer cache algorithms. We conclude our research and suggest future work in Section 5.

2. BACKGROUND

Mobile device has a limitation on extending hardware such as CPU, memory, and battery because it should be fulfilled necessary conditions such as reasonable price, portability, and small shape. However, the performance of mobile devices is increasing rapidly and therefore mobile devices need much more energy to operate. As a result, the battery efficiency becomes one of the significant issues in mobile device era. Therefore, various researches which try to reduce the power consumption to preserve battery capacity have been studied so far [7, 8, 9, 10, 11]. Also, mobile device can run several applications at the same time because it supports a multitasking environment. And these mobile applications are simultaneously running as multiple processes. These processes could be classified into various types (e.g., system process, foreground process, and background process) depending on its state. System processes are generated by Android platform and these processes are the most important processes in mobile operating system. System processes should not be interfered by other processes. And the application which is being used by user is called foreground application. The processes which are currently being used by user are foreground processes and the processes which are on standby in background state are called background processes. Only a few processes can run in foreground state and other processes should run in background state. And foreground process produce a large number of I/O requests while interacting with user, whereas background processes rarely issue I/O requests, which are generated by push services or audio file playbacks. Generally, foreground process run with high priority because users can perceive performance of the foreground process as the performance of the mobile device. On the other hand, background processes may be running with lower priority than foreground ones because they do not need fast response and do not affect directly to the user. In general mobile device environment, we can see that most of read/write requests are generated by foreground process and we confirmed this in our experimental results in Section 4.

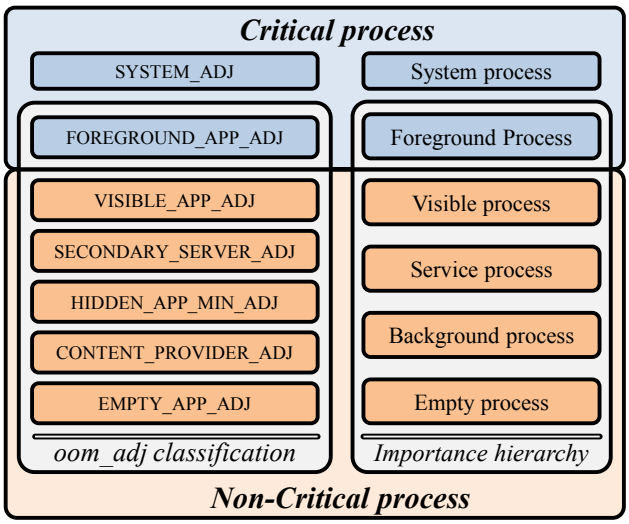

Figure 1. Various states of the processes

Android platform is the one of the most popular software platform for mobile devices [12]. It has been evolved to be suitable with mobile environment. Especially, Android adopts various functions from Linux kernel, such as wakelock [13], binder [14], and low memory killer [15], to customize the kernel for mobile device. In Android kernel, many applications, which are currently not being used by user, are maintained in the background state until being killed by LMK (Low Memory Killer) or changed to foreground state. LMK kills processes to obtain essential memory space to launch new processes. When LMK selects a process to kill among background processes, it checks the oom_adj value of the process. The oom_adj value which is represented various state of the processes has one of the value in the -17 to +15 range and it is defined in the file which name is init.rc. Foreground process has a zero value and system process has a minus value. On the other hand, background process has one of the plus values according to its state. LMK does not kill the process whose oom_adj value is below zero such as system process and foreground process. It kills a process which has the lowest oom_adj value in the list of the background application. Through this operating characteristic, oom_adj values can be an indicator which represents priority of processes. Android platform reduces these priority orders into five standard classifications called 'importance hierarchy'. The importance hierarchy consists of foreground process, visible process, service process, background process, and empty process [5]. Foreground process means the process which is using currently by user. And visible process means the process which is not being controlled currently by user but user can see it on display. These two types of processes can affect to the user directly or indirectly because they are visible to user. However, service process, background process and empty process do not affect any direct effect to the foreground applications. Figure 1 represents various state of the application in both oom_adj classifications and importance hierarchy classifications. In the rest of this paper, we treat both system process and foreground process as a 'critical process'. And other processes which have lower priority than critical processes, such as visible process, service process, background process, and empty process, are regarded as a 'non-critical process'.

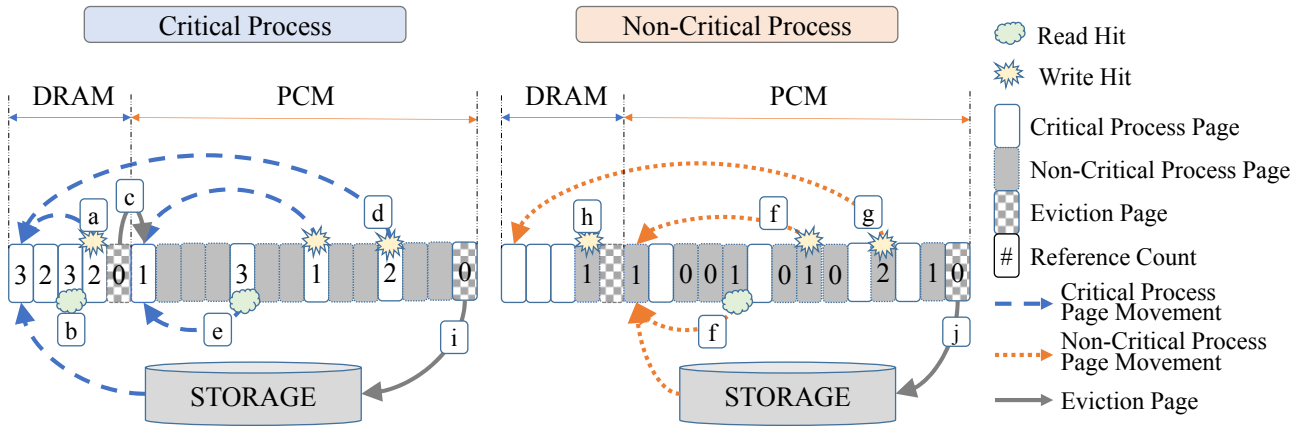

Figure 2. The example of proposed buffer caching algorithm with the process state

3. PROPOSED ALGORITHM

In this paper, we proposed a novel buffer cache algorithm based on hybrid memory architecture, which is consist of DRAM and PCM. In order to consider characteristics of mobile application, proposed algorithm classifies mobile processes into critical process and non-critical process. For classification, our algorithm exploits the oom_adj value of android platform. Android platform gives priority to process of applications by using the oom_adj value. The oom_adj value of critical processes (e.g., system and foreground process) is set to zero or negative value for giving higher priority, called critical process. In contrast, the oom_adj value of processes for background application is set to positive value to give lower priority, called background process.

In general, foreground application generates several processes and these processes request a large number of read/write operations. Therefore, proposed algorithm places the pages, which are accessed by critical processes, on DRAM to guarantee fast response time of application and to improve the lifecycle of PCM. On the other hand, the pages, which are accessed by non-critical processes, are firstly placed on PCM because non-critical processes do not need fast response time. In order to avoid performance drop, the proposed algorithm also places the pages that are used by Android system on DRAM because latency of DRAM is faster than that of PCM.

Although this architecture consists of two different types of memories, we can access like previous DRAM memory architecture because they are connected via memory bus. Only the difference is that p architecture is divided into two parts: (1) front part for the critical process and (2) rear part for the background process. DRAM which has fast response speed is placed on the front part of memory architecture and PCM is located right after the LRU page of DRAM. Consequently, hot pages, which are working on DRAM, could be remained longer than cold pages in the page cache layer. Besides, the main idea of proposed algorithm is that two types of memory space are separately managed according to the state of the processes. So, our algorithm can be easily applied to any types of buffer cache algorithm. In this paper, we extend traditional LRU algorithm, which is widely used, based on hybrid memory architecture composed of DRAM and PCM to consider temporal locality.

Figure 2 illustrates examples of page migration and eviction process of the proposed algorithm. Every page on the memory system has a reference count which is used to select a victim page. Reference count is incremented whenever the page is accessed again (the page hit) and it is decremented when the page is selected as a candidate for eviction. If the reference count of a page on the LRU (Least Recently Used) position of DRAM is zero, the page is migrated to the MRU (Most Recently Used) position of PCM. If PCM also have no free pages, PCM selects a victim page and the victim page in PCM is evicted to secondary storage.

3.1 Critical Process

When a page is referred by critical process at the first time, proposed algorithm places the page on the MRU position of DRAM. If page hit occurs on DRAM, proposed algorithm manages the page with different method according to its state: (1) dirty pages are moved to the MRU position of DRAM [Figure 2a] and (2) clean pages are maintained its original position without the movement [Figure 2b]. As a result, clean page is migrated to the MRU position of PCM earlier than dirty pages. This is because the read latency of PCM is not much slower than DRAM and write latency of that is more than 20 times slower than DRAM. To avoid performance degradation of *critical process*, proposed algorithm tries to maintain pages, which are frequently used by the *critical process*, on DRAM as long as possible. As critical process requests read and write operations, DRAM is likely to full soon because we assume that our hybrid memory architecture has small size DRAM. If a new page is requested by critical process and there is no empty slot on DRAM, the LRU page on DRAM is selected as a victim page for migration to PCM. At this time, if a reference count of the page is bigger than one, proposed algorithm decrements the reference count by one and moves the page to the MRU position of DRAM. Then, the proposed algorithm checks the reference count of the next LRU page until finding a victim page whose reference count is zero. If the reference count of the victim page is zero, the page is migrated to the MRU position of PCM and these pages are managed by LRU policy on PCM [Figure 2c]. When the pages which are migrated to PCM are accessed again for a short period, proposed algorithm checks the reference count of the page. If the reference count is more than two, proposed algorithm regards the page as a hot page and the page is migrated to the MRU position of DRAM [Figure 2d]. However, clean pages on PCM are not migrated to DRAM even if its reference count is more than two because read latency of PCM is similar to that of DRAM [Figure 2e]. As a result, write requests which cause considerably long

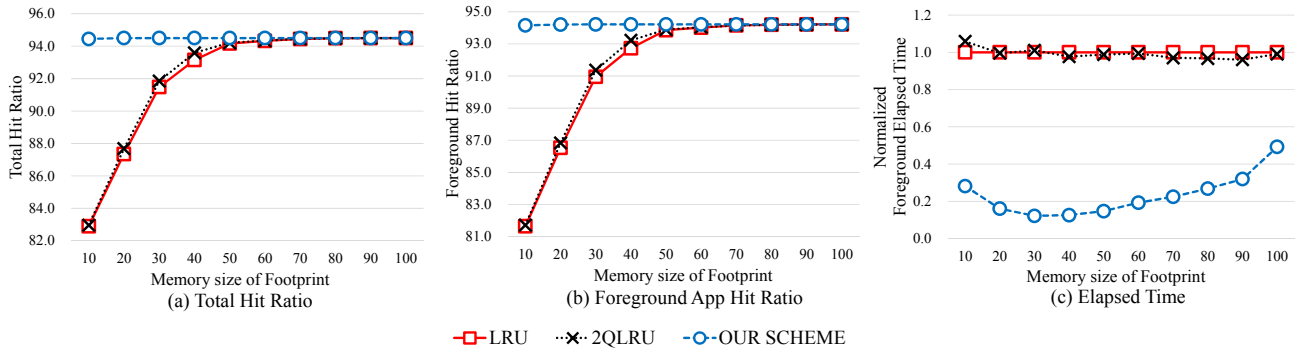

Figure 3. Experimental results for the critical process intensive workload.

latency on PCM could run on DRAM without any interference by clean pages. As such, proposed algorithm gathers hot and dirty pages on DRAM to improve the performance of the critical process.

3.2 Non-Critical Process

New pages for non-critical processes are placed on the MRU position of PCM. In proposed algorithm, pages for non-critical processes are not migrated to DRAM but they are moved to the MRU position of PCM [Figure 2f]. Therefore, the performance of the *critical process* can be ensured by maintaining pages accessed critical process on DRAM. However, if the dirty page whose reference count is bigger than two is referred again, it is regarded as a hot page and migrated to the MRU position of DRAM [Figure 2g]. Since repeated write requests on PCM could decrease the performance of the entire system even though non-critical processes do not directly affect critical process. However, migrated pages from PCM to DRAM maintain are not moved to the MRU position of DRAM although the page is accessed again (page hit) [Figure 2h]. If a reference count of a page on the LRU position of DRAM is zero, this page is migrated to the MRU position of PCM because these pages could cause performance degradation of the critical process by occupying page slots of DRAM. Although non-critical processes could be slowed down by giving low priority and working on PCM, user can not perceive the performance degradation of the mobile device because they do not directly affect the critical process and are invisible to the user. Based on this fact, proposed algorithm improves the performance of the critical process by exploiting the characteristics of mobile device environment that users can perceive performance of the critical process as the performance of the mobile device.

If there is no free page to allocate a new page for critical process, our algorithm selects a victim page, which is located on the LRU position of PCM and it is evicted to the secondary storage [Figure 2j] then the LRU page on DRAM is migrated to the PCM [Figure 2i]. After two eviction processes, a new page is written on the MRU position of DRAM. In case of a new page for the non-critical process, page migration from DRAM to PCM is not required because the new page is directly placed on the MRU position of PCM. Therefore, the page on LRU position of PCM is only evicted to the secondary storage. If the reference count of the page on the LRU position of PCM is bigger than one, proposed algorithm decrements its reference count by one. And then proposed algorithm migrates the page to the MRU position of PCM and checks the reference count of the next victim page until finding a victim page that its reference count is zero.

4. EVALUATION

4.1 Experimental Preparation

In this section, we present the performance evaluation and compare the performance of our buffer cache algorithm with the most popular buffer cache algorithms, such as LRU and 2QLRU. To verify our buffer cache algorithm, we performed trace-driven simulations which are useful for emulating hybrid memory architecture. We also modified Android kernel source code, and then extracted trace data from Google Nexus 7 with 1 GB of DRAM which is using Android lollipop version 5.0.2 based Linux kernel version 3.1.10. We collected trace data while running several applications for 24 hours, including Chrome, Hangouts, Gmail, Facebook, audio streaming service and system applications. The read/write request number of each workload is indicated in Table 2. Unfortunately, trace data for critical process intensive workload was collected for 60 minutes. Although amount of trace data for critical process intensive workload is less than other workloads, it is enough to show the characteristic about critical process intensive workload. In order to get results about various operational environments, we assume four types of operational environment; 1) critical process intensive workload, 2) non-critical process intensive workload, and 3) balanced workload and stand-by workload. In critical process intensive workload and non-critical process intensive workload, the most of I/O requests are generated by critical process and non-critical process. Also there are similar amount of I/O requests in balanced I/O requests workload. For stand-by state workload, mobile device was turned on but any operation was not done.

Table 2. Operation characteristics for each workload

	Critical I/O	Non-critical I/O
Critical process intensive workload	1,027,230	131,915
Non-critical process intensive workload	11,236,137	13,470,195
Balanced workload	9,136,464	7,129,521
Stand-by state workload	1,350,282	523,500

4.2 Experimental Results

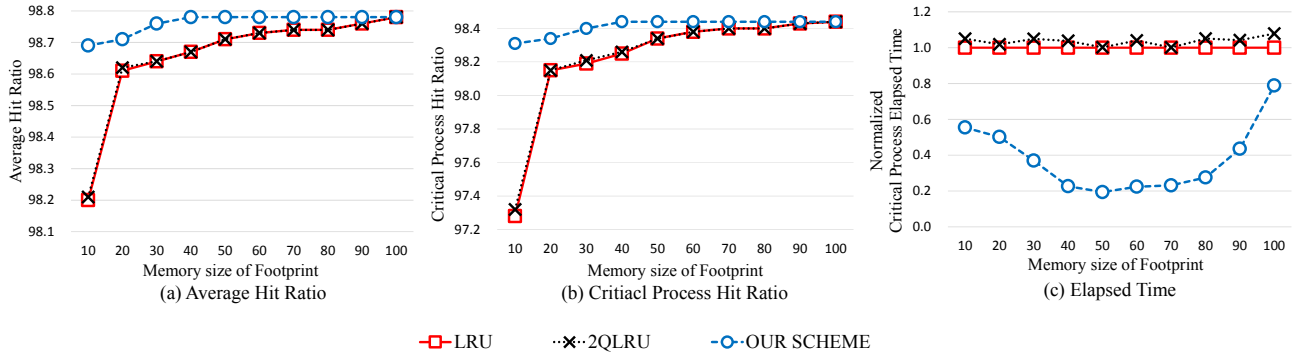

Figure 4. Experimental results for the non-critical process intensive workload.

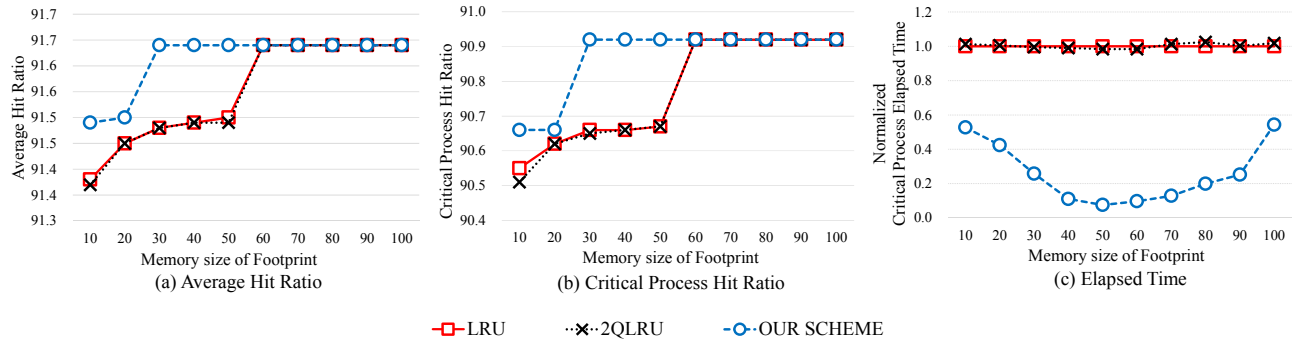

Figure 5. Experimental results for the balanced operation workload.

In this section, we explain the three experimental results for each workload. At first, average hit ratio indicates hit ratio for I/O requests for both critical process and non-critical process. However other two results are focused on the result for critical process because we assumed that users can perceive the performance of critical process as the performance of the device. Second, critical process hit ratio shows hit ratio about only critical process. Lastly, elapsed time indicates the time to operate I/O requests for critical process.

4.2.1 Critical process intensive workload

First experiment is about general usage of the mobile device that critical process generates most of read/write requests. Figure 3 shows experimental results for critical process intensive workload. We did web surfing with Chrome web browser as a foreground application and make several applications to background state. And we send an instant message per 30 seconds to the mobile device using Hangouts to generate push services because general background application rarely issues read/write requests. In the critical process intensive workload, the number of the read/write requests for critical process is eight times bigger than the number of read/write requests for non-critical process [Table 2]. In the simulation result, proposed algorithm shows higher hit ratio by 12% points than LRU and 2QLRU algorithm for both hit ratio about entire workload and critical process [Figure 3a, Figure 3b]. Especially, our algorithm performs better in the range of small size memory. In other words, our algorithm is suitable for the process which issues a lot of read/write requests at critical state. Also, elapsed time for critical process has been decreased by 60% point compared to LRU and 2QLRU algorithms [Figure 3c]. Because the number of eviction operation to the secondary storage was reduced due to giving preference to the critical process. Through first experiment, we can see that proposed algorithm improves the overall performance of critical process by preventing page eviction to the secondary storage. As a result, proposed algorithm makes better sensory performance because users can perceive performance of the critical processes as the performance of the mobile device as we mentioned in Section 3.

4.2.2 Non-critical process intensive workload

We tested second experiment to check performance impact of the critical process in the environment of non-critical process intensive workload and the results are indicated in Figure 4. Although several applications have been executed in background state, these background applications did not generate lots of read/write requests. Therefore, we generated artificially push service by sending an instant message per five seconds to the mobile device. At this time, we executed the Chrome as a foreground application and did nothing with it because we need to get non-critical process intensive trace. In second trace, the number of read/write requests by non-critical process is 1.2 times bigger than the number of critical process. Although we did nothing with critical process, many system processes caused by non-critical process have generated lots of read/write requests. According to this workload, three algorithms were showed similar but the best performance in both hit ratio [Figure 4a, Figure 4b] because proposed algorithm manages hot pages in DRAM. And elapsed time about non-critical process intensive workload has the smallest performance improvement compared to results for other workloads [Figure 4c]. However, proposed algorithm shows better performance than LRU and 2QLRU in entire range. Because DRAM is protected from many pages written by non-critical process, therefore pages for critical process are not evicted

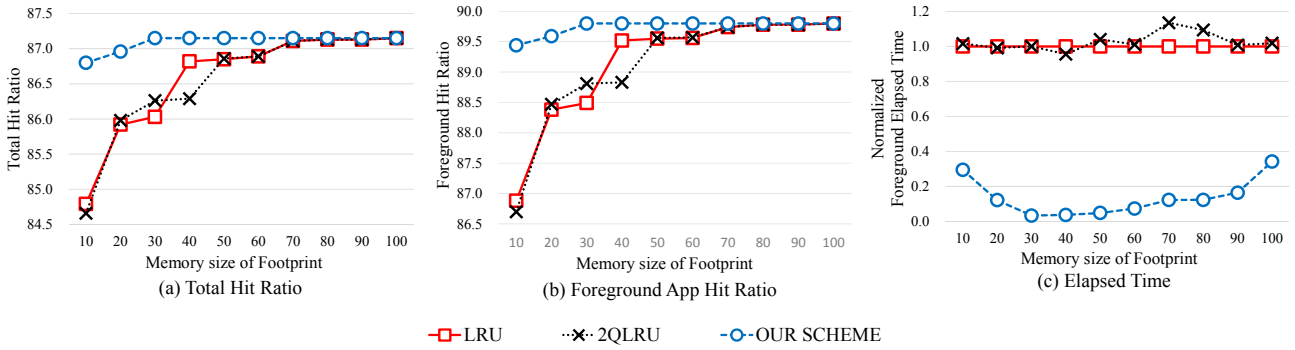

Figure 6. Experimental results for the stand-by state workload.

frequently to the secondary storage and they are remained on DRAM for a long time. Thus these pages have a second chance to survive before being evicted to the secondary storage in contrast with LRU and 2QLRU algorithm. On the other hand, these algorithms need considerably long time to run the same workload because they do not give higher priority to critical process. Consequently, many pages for critical process are evicted frequently from memory to the secondary storage because non-critical process writes new pages on DRAM. Through experimental results for non-critical process intensive workload, we can see that proposed algorithm is also suitable for even non-critical process intensive environment.

4.2.3 Balanced workload

In this section, we tested the third workload which has similar number of the read/write requests between critical process and non-critical process. We used Chrome web browser as a foreground application like the method we tested in previous section and we did not any operation with the foreground application. However, we generated an instant message using Hangouts per 30 seconds operation in order to make balanced workload because background applications do not produce read/write requests. In the experimental, proposed algorithm showed better performance in both graphs about average hit ratio and hit ratio for critical process [Figure 5a, Figure 5b]. First of all, hit ratio about proposed algorithm is saturated over the range of 30% while hit ratio about other algorithms is saturated over 60%. Because we did not operate the foreground application, small size memory is enough to complete read/write requests for foreground application. Especially, proposed algorithm used even smaller memory than other algorithms and our algorithm had managed properly these small pages. Furthermore, proposed algorithm has higher hit ratio in entire range. Same as previous experiments, the elapsed time has been reduced more than 70% on average compared with LRU and 2QLRU algorithms [Figure 5c]. It means that evicted page for critical process was considerably reduced in the experimental result of the proposed algorithm. Through the experiment about balanced workload, we can find out that proposed algorithm, which gives high priority to critical process, is suitable even if non-critical processes generate lots of read/write requests.

4.2.4 Stand-by state workload

Generally, most of applications are not terminated but they are remained in background state when user switches current foreground application to another application. These applications are remained in background state until being killed by LMK or being terminated by users and they do not generate many read/write requests. So we made fourth workload and did experimental work to analyze how many background applications produce read/write requests in sleep state. We executed several applications in background state with no foreground application. And we did not generate any operation with background application. Although we collected the trace during 24 hours, the total number of read/write requests is evidently small compared with previous workloads. Also System processes generated more than 2.5 times read/write operations than non-critical processes even if there is no critical process. This confirms what we mentioned in Section 2 that the background applications do not generate many read/write requests in sleep mode. The evaluation results of fourth workload is similar to the results about critical process intensive workload, therefore proposed algorithm showed the highest performance in both hit ratio and elapsed time among three algorithms [Figure 6]. Especially, the elapsed time in standby state workload about proposed algorithm is reduced more than 85% than other algorithms.

Through our experimental results about four workloads, we can see that the proposed algorithm improves the performance of the critical process in all kind of operational environments. It means that users can feel the performance improvement when they are using any critical process. Also proposed algorithm shows better performance if critical process generates lots of read/write requests. In the real operational environment, most of read/write requests in mobile device are generated by foreground application, therefore proposed algorithm may be the most suitable buffer management algorithm for the mobile device.

4.3 Energy consumption

Proposed memory architecture minimizes refresh operations by using PCM which is non-volatile memory and it significantly reduces static power. Although PCM needs more energy per single operation than DRAM, proposed algorithm reduces energy consumption by collecting hot and dirty page on DRAM and minimizing write requests on PCM.

$$\mathbf{W_D + R_D + W_P + R_P + PM_D + PM_P + NP_D + NP_P}$$

W_D, R_D, W_P, R_P rite/read energy on DRAM and PCM
PM_D: page migration energy from DRAM to PCM
PM_P: page migration energy from PCM to DRAM
NP_D: page missing (new page) energy to DRAM
NP_P: page missing (new page) energy to PCM

Equation 1. Calculation method for energy consumption

In this section, we calculate operational energy consumption on each workload based on Table 1 and Equation 1. Equation 1 indicates simple mathematical analysis of energy consumption and we explain the meaning of each character below the equation. Especially, PM_D includes one read operation on DRAM and one write operation on PCM and PM_P includes one read operation on PCM and one write operation on DRAM. Also NP_D includes one write operation on DRAM and NP_P includes one write operation on PCM. We did not consider energy decline in static power consumption because we have assumed that experiment is performed on the same hybrid memory architecture.

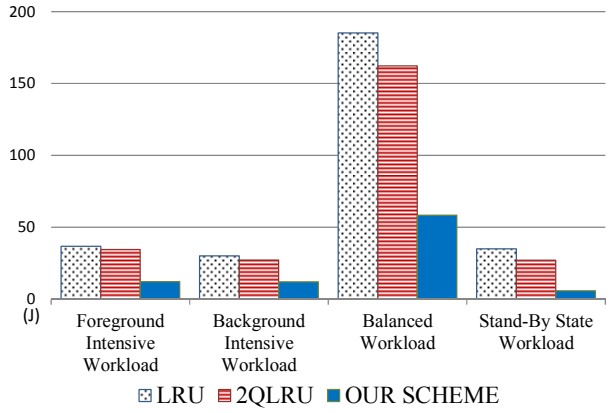

Figure 7. Consumed energy for workloads

Figure 7 indicates energy consumption value by working each workload in the memory system. X-axis represents four workloads and y-axis indicates consuming energy at each workloads. In experiment, we set that memory size is 50% of the footprint and PCM is 90% of total memory size. Proposed algorithm saves more energy when the workload has lots of read/write requests which are mostly generated by foreground application. In mobile device environment, most of read/write requests are generated by foreground application. And the number of read/write requests will gradually be growing when user continuously uses mobile device. In this point of view, we can see that proposed algorithm can save significant amount of limited battery capacity in mobile device.

5. CONSLUSION

In this paper, we propose a novel buffer cache algorithm for devices to mitigate the weaknesses of previous buffer cache algorithms. Proposed algorithm exploits the characteristics of mobile application according to its different state to manage the pages with different method. At this time, we classified the processes into two types depending on its criticalness to system performance. Also, proposed algorithm can reduce static power consumption which is caused by performing refresh operation in DRAM because it is based on hybrid memory architecture composed of DRAM and non-volatile PCM. In addition, proposed algorithm reduces operating energy by minimizing write operation on PCM. We confirmed the performance and energy conservation of proposed algorithm through the simulation using various types of trace which are extracted from real devices. As a result, proposed algorithm shows high hit ratio by maintaining the pages, which belong to critical processes, as long as possible on DRAM. Our experimental results clearly show that our algorithm reduces the elapsed time of critical processes by 70% on average and the power consumption by 67% on average compared with other algorithms while operating same workloads.

6. ACKNOWLEDGMENTS

This work was supported by ICT R&D program of MSIP/IITP. [R0126-15-1065, (SW StarLab) Development of UX Platform Software for Supporting Concurrent Multi-users on Large Displays]. Young Ik Eom is the corresponding author of this paper.

7. REFERENCES

[1] Qureshi, M. K., Srinivasan, V., and Rives, J. A. 2009. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceeding of the International Symposium on Computer Architecture* (Austin, USA, June 20 - 24, 2009). ISCA '09 ACM, New York, NY, 24-33. DOI= http://dx.doi.org/10.1145/1555754.1555760

[2] LEE, S., Bahn, H., and Noh, S. H. 2013. CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures. *IEEE Transactions on Computers.* 63, 9 (Apr. 2013), 2187-2200. DOI= http://dx.doi.org/10.1109/TC.2013.98

[3] Dhiman, D., Ayoub, R., and Rosing, T. 2009. PDRAM: A Hybrid PRAM and DRAM Main Memory System. In *Proceeding of the Design Automation Conference* (San Francisco, USA, July 26 - 31, 2009). DAC '09. ACM, New York, NY, 664-669. DOI= http://dx.doi.org/10.1145/1629911.1630086

[4] Eilert, S., Leinwander, M., and Crisenza, G. 2009. Phase Change Memory: A New Memory Technology to Enable New Memory Usage Models. In *Proceeding of the International Memory Workshop* (Monterey, CA, May 10 - 14, 2009). IMW '09. IEEE. 1-2. DOI= http://dx.doi.org/10.1109/IMW.2009.5090604

[5] Android Open Source Project [Online]. Available: https://developer.android.com/guide/components/processes-and-threads.html

[6] Johnson, T and ShaSha, D. 1994. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceeding of the 20th International Conference on Very Large Data Bases* (Santiago, Chile, September 12 - 15, 1994). VLDB '94. Morgan Kaufmann Publishers Inc. San Francisco, CA, 439-450.

[7] Carroll, A. and Heiser, G. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the USENIX Annual Technical Conference* (Boston, USA, June 23 - 25, 2010). USENIX '10. USENIXATC, Berkeley, CA, 271-284.

[8] Datta, S. K., Bonnet, C., and Nikaein, N. 2012. Android Power Management: Current and Future Trends. In *Proceeding of the Enabling Technologies for Smartphone and Internet of Things* (Seoul, Korea, June 18 - 21, 2012). ETSIoT SECON '12. IEEE, 48-53. DOI= http://dx.doi.org/10.1109/ETSIoT.2012.6311253

[9] Lim, G., Min, C., Kang, D. H., and Eom, Y. I. 2013. User-Aware Power Management for Mobile Devices. In

Proceeding of the Global Conference on Consumer Electronics (Tokyo, Japan, October 1 - 4, 2013). GCCE '13. IEEE, 151-152. DOI= http://dx.doi.org/10.1109/GCCE.2013.6664780

[10] HAN, S. J., Kang, D. H., and Eom, Y. I. 2015. Low Power Killer: Extending the Battery Lifespan by Reducing I/O on Mobile Devices. In *Proceeding of the IEEE International Conference on Consumer Electronics* (Las Vegas, USA, January 9 - 12, 2015). ICCE '15. IEEE, 579-580. DOI= http://dx.doi.org/10.1109/ICCE.2015.7066534

[11] Chu, S. L., Chen, S. R., and Weng, S. F. 2012. Design a Low-Power Scheduling Mechanism for a Multicore Android System. In *Proceeding of the Parallel Architectures, Algorithms and Programming* (Taipei, Taiwan, December 17 - 20, 2012). PAAP '12. IEEE. 25-30. DOI= http://dx.doi.org/10.1109/PAAP.2012.12

[12] Gandhewar, N. and Sheikh, R. 2011. Google Android: An Emerging Software Platform for Mobile Devices. *Journal of Computer Science and Engineering.* 1, 6 (Nov. 2014), 12-17.

[13] Android Open Source Project [Online]. Available: https://source.android.com/devices/tech/power/index.html

[14] Android Open Source Project [Online]. Available: https://source.android.com/devices/#Binder IPC

[15] Android Open Source Project [Online]. Available: https://source.android.com/devices/tech/ram/low-ram.html