Efficient Memory Deduplication for Mobile Smart Devices

Sung-hun Kim, Jinkyu Jeong, and Joonwon Lee
College of ICE, Sungkyunkwan Univ., Suwon, Rep. of Korea
{shkim, jinkyu}@csl.skku.edu, joonwon@skku.edu

***Abstract*--Memory deduplication, which can remedy memory scarcity in mobile systems, can hardly be used due to its high computation cost. This paper proposes a computation efficient memory deduplication scheme that avoids unnecessary computations for memory deduplication. Our prototype shows significant computation cost reduction while providing the same memory savings as in previous approaches.**

I. Introduction

Smart mobile devices, such as smartphones and tablets, provide many useful functions in the form of applications. These applications are usually cached in memory as background processes since running an application from its cached state is much faster than running it by spawning a new process as shown in Fig. 1. This application caching, however, comes at a cost of memory occupation. Many well-known mobile applications require about 20 MBs for caching them on average as depicted in Table I. Meanwhile, smart devices have their memory capacity limited due to cost, small form factor and power consumption. Accordingly, even though caching applications has the benefit of fast launching, less likely to be used applications are killed to obtain free memory in order to satisfy other applications' memory demands. These killed applications will results in long latencies for re-launching them since they should accompany spawning new processes.

Memory deduplication (or transparent page sharing) is one of approaches to increase memory density in computer systems [1][2]. In this approach, a memory deduplication procedure periodically scans a system's memory pages and merges pages storing identical contents so that it can secure additional free memory. For example, a Galaxy Nexus smartphone has 200MBs of free memory after booting. When the memory deduplication is performed, its free memory increases to 280MBs, gaining 80MBs of additional free memory. By exploiting this surplus memory, the system can cache several applications additionally so to be able to decrease long latencies from spawning new processes. A problem, however, is that existing memory deduplication approaches cannot be directly used in mobile systems because scanning memory contents is highly computation-intensive and drains a battery significantly.

This paper tackles this problem and proposes an efficient memory deduplication scheme for mobile smart devices. Mobile smart devices have two characteristics: (1) background processes do not change their memory contents and (2) a small number of virtual pages have memory deduplication chances. By exploiting both characteristics, the proposed scheme decreases the computation cost for the memory deduplication significantly. Our scheme is implemented in a Galaxy Nexus Android smartphone and evaluated with a real user workload. The proposed scheme showed 98% of computation reduction while providing the same memory savings as in the previous approach.

This work was partly supported by the IT R&D program of MKE/KEIT. [10041244, SmartTV 2.0 Software Platform]

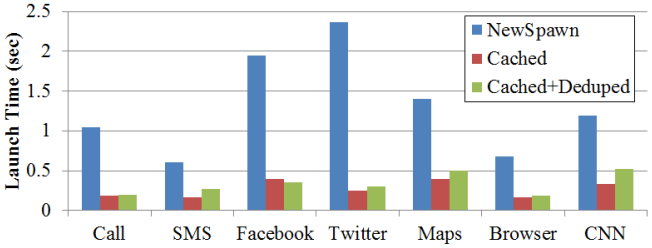

Fig. 1. Launch time of applications in three cases: spawning a new process, resuming a cached application, and resuming a cached one whose memory pages are duplicated and so copy-on-write-protected.

TABLE I
Memory Costs for Caching Applications

Application	Memory Cost	Application	Memory Cost
Call	14.3 MB	Twitter	18.6 MB
SMS	5.7 MB	Maps	32.4 MB
Facebook	24.5 MB	Browser	65.1 MB
CNN	8.0 MB	Mail	6.7 MB
GooglePlay	8.8 MB	**Average**	**20.5 MB**

II. Proposed Scheme

The advantage of memory deduplication is to be able to secure additional free memory for caching data or applications more in memory. On the other hand, the cost for memory deduplication is to inspect memory contents and to write-protect merged pages. When multiple pages are merged into one page, multiple processes share the merged page and the page is copy-on-write (CoW) protected. When each process updates the shared page, the page is CoW-breaked and the process updating the shared page has its own copy of the page. Accordingly, when a process often updates its merged pages, frequent merging and CoW breaking of pages become overheads of a system. Therefore, it is important to exclude processes and pages which cause this frequent merging and breaking.

A. Background process pruning

Most mobile devices adopt an application framework that provides one foreground application and many background applications. Since these devices have a small screen, only one application is active in foreground and the rest of the

applications are inactive in background. An important point is that, from the perspective of memory, a foreground application is actively changing its memory pages while background applications are not updating their memory contents.

By exploiting this characteristic, our memory deduplication approach only targets a process's memory pages only when the process turns out to be in background. When a process is in foreground, this process is excluded from the target of periodic page scanning. When a process goes into background, its memory pages do not change over time. Accordingly, if its memory pages are merged, the merged state persists for a long time so that saved free pages can be exploited by other applications.

In addition, a background process's pages should be scanned only once because its memory contents do not change over time until it goes into foreground. Therefore, in our scheme, once a background process's pages are scanned, the pages are not scanned in successive rounds.

B. *Virtual page pruning*

Many smart device platforms, such as Android, adopts fork()-dlopen() execution model. In this execution model, application processes have a similar address space layout to each other because they are forked from the same parent process (*zygote* in Android). In this environment, identical pages are found in a small set of virtual pages over the entire address space.

By using this characteristic, our scheme manages a profile to filter out virtual pages which do not generate memory savings. When a process enters background for the first time, the profile records virtual pages being merged. When the process goes into background next time, the memory deduplication procedure only scans virtual pages within the profile so that it does not waste CPU for unnecessary virtual page scanning.

In addition, our scheme manages one profile for all applications in a system. Since each application has mostly the same address space layout, most virtual pages are overlapped in each process's profile.

III. Evaluation

The proposed scheme is implemented in the Android Linux kernel 3.0.8 that runs on Galaxy Nexus. We modified kernel samepage merging (KSM) in the Linux kernel. We ran a real user workload which consists of many well-known Android applications as in [4]. The memory deduplication procedure (*ksmd*) periodically scans 100 pages and sleeps for 20 milliseconds by default.

Fig. 2 shows the computation cost for scanning one page in each scheme. *Hash-based* denotes a modified version of KSM. It uses hashtable to find identical pages [1] while vanilla KSM exploits red-black tree [2]. Our two pruning methods are implemented based on the hash-based implementation.

As shown in the figure, both pruning methods decrease computation cost by 94% and 98%, respectively, as compared to the vanilla KSM implementation. In comparison with hash-based one, both pruning methods still reduce computation cost by 41% and 81%.

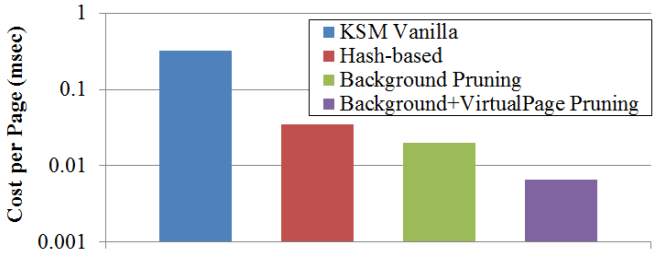

Fig. 2. Computation time for scanning one page in each scheme (time spent by *ksmd* / # of pages scanned).

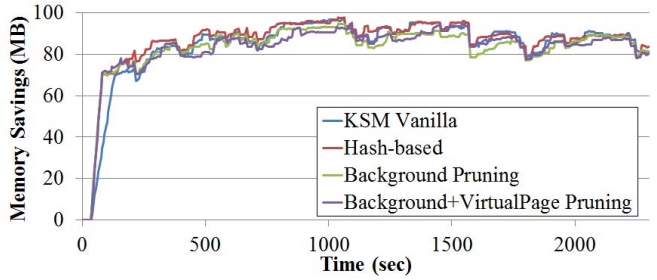

Fig. 3. Memory savings while a real-user workload is running.

Fig. 3 shows the memory savings over time while the workload is running. As depicted in the figure, even if our schemes are applied, the amount of memory savings is mostly the same as that in vanilla KSM or hash-based KSM. This result means that our scheme minimizes computation costs for finding identical pages while securing the same amount of free memory.

IV. Conclusion and Future Work

We suggested and evaluated an efficient memory deduplication scheme for mobile smart devices. Our pruning scheme with hash based indexing is effective for decreasing the computation cost of memory deduplication in mobile systems.

In spite of this computation cost reduction, periodic scanning of identical pages is meaningless when free memory is sufficient. But, when free memory is low, it could be better to perform memory deduplication rather than reclaiming cache pages or cached applications. We are conducting research on finding proper time to perform memory deduplication.

Reference

[1] C. A. Waldspurger, "Memory resource management in VMware ESX server," *in Proc. 5th Symp. Operating System Design and Implementation*, Boston, MA, USA, 2002, pp. 182-194

[2] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," *in Proc. Ottawa Linux Symp.*, Montreal, Quebec, Canada, 2009, pp. 19-28

[3] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," *in Proc. 10th Int. Conf. Mobile Systems, Applications and Services*, Low Wood Bay, Lake District, UK, 2012, pp. 113-126

[4] J. Jeong, H. Kim, J. Hwang, J. Lee, and S. Maeng, "DaaC: device-reserved memory as an eviction-based file cache," *in Proc. 21th Int. Conf. Compilers Architecture and Synthesis for Embedded Systems*, Tampere, Finland, 2012, pp. 191-200