FLIC: Fast, Lightweight Checkpointing for Mobile Virtualization using NVRAM

Kan Zhong, Duo Liu*, Liang Liang§, Linbo Long, Yi Lin, and Zili Shao†
Key Lab. of Dependable Service Computing in Cyber Physical Society (Chongqing Univ.), Ministry of Education, China
College of Computer Science, Chongqing University, China
§College of Communication Engineering, Chongqing University, China
†Department of Computing, The Hong Kong Polytechnic University, Hong Kong

***Abstract*—Checkpointing is a key enabler of hibernation, live migration and fault-tolerance for virtual machines (VMs) in mobile devices. However, checkpointing a VM is usually heavy-weight: the VM's entire memory needs to be dumped to storage, which induces a significant amount of (slow) I/O operations, degrading system performance and user experience. In this paper, we propose *FLIC*, a fast and lightweight checkpointing machinery for virtualized mobile devices by taking advantages of recent byte-addressable, non-volatile memory (NVRAM). Instead of saving the VM's entire memory to storage, we store its working set pages in NVRAM, avoiding accessing slow flash memory (compared to server-grade SSDs). To cope with the energy constraint of mobile systems, we further deduplicate VM snapshots, reducing the VM's image size and saving storage space. Experimental results based on an Exynos 5250 SoC show that our approach can effectively improve the performance of checkpointing in mobile virutalization and save energy.**

I. INTRODUCTION

Mobile virtualization [1], [2], [3] enables tiny mobile devices to run multiple virtual machines (VMs)—each running an isolated OS—on a single device, for benefits higher resource utilization, better security and so on [4], [5], [6], [7]. Though hardware extensions (e.g., the ARM virtualization extension) can significantly improve the performance of mobile virtualization, running multiple VMs at the same time still inevitably requires more resources, especially for memory and storage. In particular, checkpointing is one of the most heavy-weight (and yet important) features that need a significant amount of hardware resources to run. Checkpointing a VM involves periodically writing the VM's entire memory pages to persistent storage (usually flash memory), and it is common for a checkpoint file to occupy hundreds of MBs or even GBs of storage space. Combined with the sub-optimal storage performance typically found in mobile devices [8], such long, bursty writes not only degrade checkpointing performance, but also slow down normal I/O requests issued at runtime.

New byte-addressable, non-volatile memories (NVRAMs) such as phase change memory (PCM) [9], [10], [11], spin-transfer torque RAM (STT-RAM) [12], [13] and memristor [14], [15] provide opportunities to optimize checkpointing in virtualized mobile systems. These NVRAM products blur the distinction between memory and storage: they are byte-addressable like DRAM, but are also non-volatile like storage (e.g., flash memory). For example, compared to flash memory, PCM offers not only faster (near-DRAM) performance, but also larger erase cycles. Moreover, PCM exhibites much better read/write latency and endurance when compared to flash memory. Unlike DRAM, PCM does not need constant current to maintain data; it also has much higher density. Though mature NVRAM products are not yet on the market, they are expected to provide near-DRAM performance, high density, and good scalability [16].

Specifically, NVRAM's non-volatility, byte-addressablity and high performance make it an ideal candidate for optimizing checkpointing in mobile devices. Note that in this paper, we do not target at any specific NVRAM product to make our approach general enough. To reduce (expensive) writes to flash memory, we propose *FLIC*, a fast and lightweight checkpointing scheme for mobile virtualization utilizing NVRAM's non-volatility and byte-addressability. FLIC stores frequently accessed VM pages (denoted as working set pages) on NVRAM, and only writes unfrequently accessed pages to flash memory. As a result, when restoring a VM from a checkpoint, FLIC can load working set pages from NVRAM quickly to resume normal VM operation, while load the remaining cold pages quietly in background from flash memory. To further reduce write activities to flash memory and the amount of data needed to read when resuming a VM. We propose an energy-aware deduplication mechanism to eliminate redundant data in VM snapshots, which also makes the background load operation easier, faster, and more energy friendly.

We evaluate FLIC with a Samsung Exynos 5250 SoC [17] running multiple Google Android VMs and applications. Experimental results show that FLIC can accurately identify working set pages, and effectively save energy through its energy-aware deduplication scheme. The energy consumption during checkpointing and size of checkpoints are reduced by $\sim$50% and more than 20%, respectively.

In summary, we make the following major contributions:

- We discover that NVRAM is an ideal candidate for optimizing checkpointing in mobile virtualization, by storing frequently used (the "working set") memory pages in NVRAM, instead of flash memory.
- We devise a simple yet effective VM working set identification scheme, so that frequently accessed working set pages can be stored in NVRAM to reduce I/O traffic during checkpointing and recovery.
- We propose an energy-aware deduplication technique to reduce the size of VM snapshots, further reducing the write activities to flash memory.

The rest of this paper is organized as follows. Section II

*Corresponding author: Duo Liu, College of Computer Science, Chongqing University, Chongqing, China. E-mail: liuduo@cqu.edu.cn.

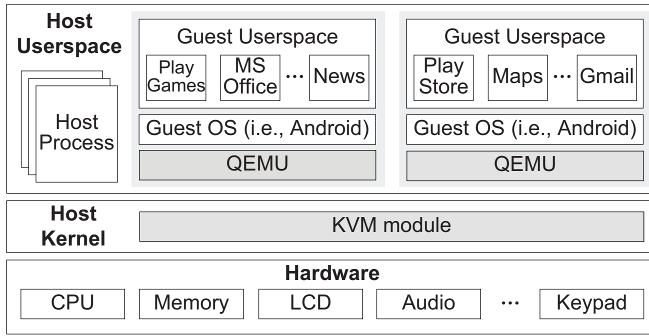
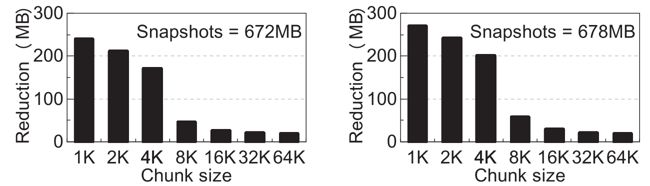


Fig. 1: Overview of a virtualized mobile system based on KVM/ARM [2]. QEMU simulates the VM's hardware system, and KVM provides the core virtualization services provided by the processor's virtualization extension.



(a) VM runs multimedia apps. (b) VM runs office apps.

Fig. 2: Redundant data in VM snapshots. Data deduplication shows a high potential in reducing write activities to flash memory, thus saving storage space and accelerating recovery of checkpoints.

gives background and our motivation. In Section III, we describe the details of FLIC, including working set identification and checkpoint deduplication. We evaluate FLIC in Section IV and conclude in Section VI.

II. Background and Motivation

In this section, we first introduce mobile virtualization. We then give the background on VM checkpointing and deduplication. Finally, we discuss our motivation.

A. Mobile Virtualization

Mobile virtualization enables multiple isolated OSes to run on a single device. It gives higher resource utilization, availability, better security and energy efficiency. A hypervisor is responsible for communicating with the underlying hardware, which is multiplexed to different guest VMs, so that multiple guests can share the same hardware, though none has exclusive access. As a result, multiple VMs could be created to run in full isolation without knowing the existence of each other. One could run a different or same mobile OS in each VM. Fig. 1 shows the architecture of a virtualized mobile system based on ARM/KVM [2]. KVM provides access to the hardware virtualization extensions provided by the processor (e.g., ARM Cortex A-15). QEMU runs in user space to emulate peripheral devices such as storage. As shown in the figure, two VMs share the same physical devices, and each OS has various apps running inside as if they were running in a native system.

B. VM Checkpointing and Deduplication

VM checkpointing refers to the process of saving the state of a VM to persistent storage, such as flash memory, so that it can be restored in its exact state later. A checkpoint of a running VM consists of the context of the virtual CPUs (VCPU) and peripheral devices connected to the VM (e.g., network adapters and storage), and a copy of all the memory pages allocated to the VM. Checkpointing a VM and then pausing it can free up resources and reduce power consumption. It also allows the user to generate a snapshot for the VM and migrate it to other devices or use that snapshot for fault-tolerance purposes.

Data deduplicaiton aims at eliminating redundant data and only stores one unique instance. Redundant instances of data are replaced with a pointer to the unique copy. Deduplication improves storage utilization and reduces the amount of data that have to be transferred between two locations. In deduplication, data are firstly split into non-overlapping chunks, and then chunks are checked for redundances by hashing and comparing them with existing values. In this paper, we adopt deduplication in checkpointing to eliminate redundant data.

C. Motivation

Mobile devices are getting increasingly powerful and demanding more and more memory to run various feature-rich applications [18]. Therefore, it is common for a VM to occupy hundreds of MBs or even GBs of memory to operate normally. Checkpointing such a VM will result in a large snapshot stored on flash memory, and the process will generate significant amount of I/O operations, which in turns slows down the I/O operations for running VMs due to the poor performance of flash storage in mobile devices [8].

Fortunately, we find that deduplication is an effective way to solve the excessive I/O problem. As shown in Fig. 2, duplicate data are common in VM snapshots. Depending on chunk sizes, as much as nearly 40% of the snapshot is redundant. This motivates us to reduce the write activities to flash memory through deduplication. Considering the energy constraint in mobile systems, in this work, an energy-aware deduplication mechanism is proposed to eliminate the redundant data in VM snapshots.

III. Making Checkpointing Fast and Lightweight with NVRAM

In this section, we first give an overview of FLIC. We then discuss the key techniques of FLIC: (1) working set identification and (2) energy-ware deduplication.

A. Overview

Fig. 3 shows an overview of FLIC. To identify the working set pages and eliminate redundant data in VM snapshot, two additional components: working set scanner and data deduplicator are added to the virtual machine monitor (VMM).

Working set scanner is designed to inspect all the pages that have been allocated to the VM, and identify working set pages based on access patterns. The basic idea is to collect the most recent used pages upon a VM checkpointing request comes. To determine the working set pages, all VM memory pages are checked for being accessed or not in a time interval. Any page has been accessed in the interval belongs to the working set pages.

Data deduplicator is a dedicated background thread and aims to remove duplicate data in the saved VM snapshot.

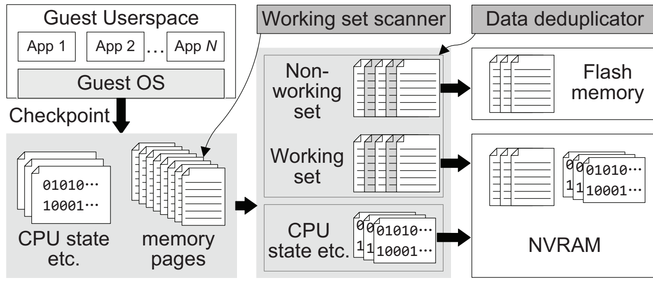


Fig. 3: Overview of FLIC. The working set scanner splits the target VM's memory pages into working and non-working set pages. The data deduplicator eliminates redundant data in the VM snapshots.

Since deduplication is usually energy-consuming and CPU-consuming, instead of using *full hashing*, *partial hashing* is adopted to reduce the hashing energy consumption. For saving energy, partial hashing computes a chunk's hash value using only a fraction of its data (i.e., sample data chunk at fixed positions with fixed length). As a comparison, full hashing computes a chunk's hash value using its entire data. For simplicity, we assume that the possibility of full hashing collision is negligible, and we do not discuss it in this work.

B. Identifying working set pages

Working set refers to the collection of the most recently used pages by a process or OS [19]. In VM checkpointing, we use working set to predict the pages that the VM touches during recovery. Therefore, we designed a working set scanner to collect working set pages based on the page types and page access pattern.

In VM's memory content, pages can be simply classified into page table pages and non-page table pages. VMM maintains page tables for the host memory management unit (MMU) to translate guest virtual address to host physical address during the VM execution. Therefore, all physical page frames of page tables are regarded as part of the working set pages.

For non-page table pages, we use the flags in the page table entries (PTEs) to monitor page access patterns. If a page is accessed, we set the "accessed" bit in the corresponding PTE. To decide whether a page belongs to the working set, we use a *two-phase scanning* method: When the VMM receives a checkpointing request, a timer is set and the working set scanner will begin to scan page tables, and clear the access bit in each PTE. Upon timeout, the working set scanner scans all the PTEs in the page table again and check the access bit. Any page whose access bit is set now belongs to the working set. The accuracy of the working set depends on the timer and it is difficult to set the timeout interval: a too long and too short interval will lead to an overestimated and underestimated working set, respectively. As the non-working set pages are stored in the flash memory and loaded in background during VM recovery, the working set should contain as many pages which the VM will touch after recovery as possible. Therefore, the timer's time should depend on the recovery time of the *non-working set*, which is determined by the size of non-working set and flash memory's read performance.

Let $N_{working_set}$ denote the number of working set pages, $N_{non_working_set}$ be the number of the non-working set pages,

Algorithm III.1 Energy-aware data deduplication

Input: *snapshots*: checkpointed VM snapshots; TH: battery threshold.
Output: *vector*: redundant vector, stores deduplication information.

1: $ck_size \leftarrow$ 4096 bytes. `//4KB by default`
2: $vector \leftarrow \emptyset$, $h_table \leftarrow \emptyset$, $ck_array \leftarrow \emptyset$.
3: $battery_level \leftarrow$ check for battery level.
4: **if** $batter_level < TH$ **then**
5: $\quad ck_size \leftarrow$ 8092 bytes.
6: **end if**
7: $ck_array \leftarrow$ split $snapshots$ with ck_size.
8: **for** $i \leftarrow 1$ **to length**(ck_num) **do**
9: $\quad part_val \leftarrow$ **partial-hash**($ck_array[i]$).
10: $\quad$**if** $\exists\, k$, $h_table[k].part_val = part_val$ **then**
11: $\quad\quad full_val \leftarrow$ **full-hash**($ck_array[i]$).
12: $\quad\quad$**if** $h_table[k].full_val = null$ **then**
13: $\quad\quad\quad h_table[k].full_val \leftarrow$ **full-hash**($ck_array[k]$).
14: $\quad\quad$**end if**
15: $\quad\quad$**if** $h_table[k].full_val = full_val$ **then**
16: $\quad\quad\quad vector[i] \leftarrow k$.
17: $\quad\quad$**else**
18: $\quad\quad\quad$insert (i, $part_val$, $full_val$) into h_table.
19: $\quad\quad\quad vector[i] \leftarrow i$.
20: $\quad\quad$**end if**
21: $\quad$**else**
22: $\quad\quad$insert (i, $part_val$, $null$) into h_table.
23: $\quad\quad vector[i] \leftarrow i$.
24: $\quad$**end if**
25: **end for**
26: **return** $vector$

N_{total} as the total number of checkpointed VM pages, and T_{timer} denotes the timer's timeout interval. We can then obtain the following equation:

$$\begin{cases} N_{working_set} + N_{non_working_set} = N_{total} \\ N_{working_set} = \alpha \times T_{timer} \\ T_{timer} = \beta \times \dfrac{N_{non_working_set}}{r} \end{cases} \quad (1)$$

In equation (1), r denotes the read performance of flash memory (r pages/second). α and β are coefficients. α represents the average number of pages the VM touches per second, which is regard as the VM memory access pattern. β is a tunable value that gives the ratio between the timeout interval and the non-working set's recovery time. We tune the value of β to strike a balance between working set size and accuracy. After knowing the values of α and β, we can figure out T_{timer} according to equation (1).

Based on the locality principle, when the scanner times out, we checkpoint the running VM immediately, including the CPU and devices states, as well as the entire memory. To represent the working set, working set bitmaps are used to keep track of the working set pages. After eliminating redundant data, pages are saved to NVRAM and flash memory according to the bitmaps.

C. Energy-aware deduplication

We implement data deduplicator considering the constraint resources in mobile systems. Since deduplication is usually heavyweight, which not only degrades performance but also consumes significant amount of energy. To solve this problem, we design an energy-aware deduplication algorithm to remove duplicate data in the saved VM snapshot.

To reduce energy consumption, we first use partial hashing instead of full hashing when comparing chunk values. In partial hashing, a part of the chunk is used to compute the hash value. If two chunks have the same partial hash value,

TABLE I: Google Android experimental workloads

VM name	Applications
Browser-VM	Firefox, Chrome, Opera, UC Browser, Next Browser
Social-VM	Facebook, Pinterest, QQ, Sina Weibo, Instagram
Multimedia-VM	Google Play Music, MX Player, TTpod Player, Youtube, KMPlayer
Shopping-VM	Amazon, Ebay, Fancy, Google Play, TaoBao
News-VM	BBC News, Flipboard, NetEase News, TED, Zaker
Mixed-VM	Firefox, Facebook, Youtube, Amazon, BBC News

then the full hash values of the two chunks are computed for comparison. By comparing partial hash values, we reduce the number of full hash computing, thus saving system energy. To further reduce energy consumption, before deduplication starts, we first check the system's remaining energy. If the remaining energy is enough (i.e., battery level is higher than a pre-defined threshold), we use the page size (i.e., 4KB) as the chunks size for deduplication; otherwise, a larger chunk size (e.g., 8KB) will be used. To reconstruct memory pages from deduplicated data, we use a *redundant vector* as the metadata to re-reference duplicate chunks.

Algorithm III.1 gives the detailed description of our energy-aware deduplication method. The remaining battery level is first checked to determine chunk size (lines 5 - 10). We then split the VM snapshot into chunks with the determined chunk size (line 11). For each chunk, we first compute its partial hash value. If the hash table already contains a chunk with the same partial hash value (lines 13–14), then the full hash values of the two chunks are computed and compared to further confirm they are duplicate chunks (lines 15–21). If the hash table does not contain any chunk with the same partial hash value or their full hash values do not match, we insert a new entry into the hash table (lines 23–27). Note that the redundant vector is also updated during this process, if $vector[i] = k$ and $i \neq k$, which means the ith chunk and the kth chunk are the same, they are duplicate chunks. After deduplication, FLIC saves working and non-working set pages in NVRAM and flash memory according to the working set bitmaps, respectively.

IV. Evaluation

We evaluate FLIC using a Samsung Exynos 5250 SoC that supports hardware-assisted virtualization. Fig. 4 shows the top view of our experimental platform, the Arndale development board. The board is equipped with an ARM dual-core processor, which is based on the ARMv7 architecture and was the first ARM processor with hardware virtualization support. The processor is clocked at 1.7GHz, and contains 32KB instruction cache and 32KB data cache. The board features 2GB DRAM and 4GB internal flash memory. We use DRAM to emulate NVRAM. We also add a 64GB SD card as secondary storage to install the host OS and store checkpoints.

We use Ubuntu 12.04 with Linux kernel 3.8 as the host OS, and Google Android as guest OS. The kernel is patched with KVM-on-ARM to utilize the virtualization extensions provided by the processor. We use real Android applications to evaluate the effectiveness of FLIC. TABLE I lists the six categories of applications in our experiments. Currently, we only run one VM instance at once and each VM is allocated with 512MB of main memory.

A. Working set identification

To evaluate the accuracy of FLIC's working set identification, we run all the VMs listed in TABLE I. The major metric is the *hit rate* of working set pages during VM recovery, which can be defined as follows:

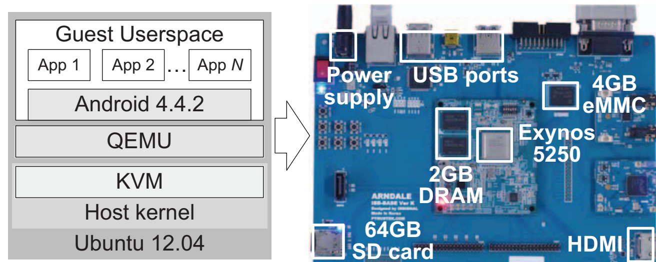


Fig. 4: Software configuration in our experiments (left) and the development board (right), which is equipped with a 1.7GHz ARM Cortex-A15 dual-core processor with virtualization extensions and 2GB DRAM.

TABLE II: Working set identification configurations.

VM name	α	β	T_{timer} (s)
Browser-VM	1000	2, 3, 4	10.66, 16.75, 21.42
Social-VM	1200	2, 3, 4	11.46, 16.33, 20.74
Multimedia-VM	1500	2, 3, 4	11.16, 17.41, 22.52
Shopping-VM	1500	2, 3, 4	11.16, 17.41, 22.52
News-VM	1000	2, 3, 4	10.66, 16.75, 21.42
Mixed-VM	1300	2, 3, 4	11.3, 16.13, 20.42

Let S_{wr} denotes the set of the working set pages, S_{rt} denotes the set of pages the VM touches during recovery, and N_{rt} denotes the number of pages in S_{rt}. For each page i in S_{rt}, we define $h(i)$ as:

$$h(i) = \begin{cases} 1 & \text{if} \quad i \in S_{wr}; \\ 0 & \text{if} \quad i \notin S_{wr}. \end{cases} \tag{2}$$

Therefore, the working set hit rate can be expressed as:

$$hit_rate = \frac{\sum_{i=1}^{N_{rt}} h(i)}{N_{rt}} \tag{3}$$

In our experiments, α is determined based on VM type. β defines the ratios between the timeout interval and the non-working set recovery time. To obtain a higher hit rate, the timeout interval must be longer than the non-working set recovery time. Therefore, to show the impact of β on hit rate, we try different values of β: 2, 3, and 4; the timeout interval is set to 2 times, 3 times, and 4 times longer than the non-working set recovery time, respectively. In the experiments, we notice that for each VM, there is a small amount of available memory left after launching applications (usually less than 3MB). Therefore, in equation (1), we set the N_{total} to the VM's total number of memory pages. Based on the read performance of the flash memory product we use, which is up to 80MB/s, we set r to 20480 pages/second. Then equation (1) can be solved to obtain the timeout interval T_{timer}, the estimated working set size $N_{working_set}$ and non-working set size $N_{non_working_set}$. TABLE II lists the experimental configurations for each VM. We use these configurations to run experiments and collect hit rate results.

TABLE III compares the size of estimated working set based on equation (1) and the real working set size collected offline. Since a larger β leads to a long timeout interval, more pages can be collected before the timeout. Therefore, as shown in the table, the size of the working set keeps growing as β increases.

TABLE III: Comparison between estimated working set size and real working set size collected offline.

VM name	$\beta = 2$			$\beta = 3$			$\beta = 4$		
	Est. size	Real size	± (%)	Est. size	Real size	± (%)	Est. size	Real size	± (%)
Browser-VM	45.55 MB	44.54 MB	2.28	65.41 MB	61.27 MB	6.76	83.66 MB	89.66 MB	6.70
Social-VM	53.70 MB	50.29 MB	6.79	76.54 MB	72.65 MB	5.35	97.21 MB	82.62 MB	17.66
Multimedia-VM	65.41 MB	65.91 MB	0.75	92.23 MB	89.76 MB	2.75	116.01 MB	114.64 MB	1.19
Shopping-VM	65.41 MB	63.50 MB	3.00	92.23 MB	87.23 MB	5.73	116.01 MB	106.41 MB	9.02
News-VM	45.55 MB	43.11 MB	5.66	65.41 MB	64.80 MB	0.95	83.66 MB	80.91 MB	3.40
Mixed-VM	57.68 MB	57.88 MB	0.34	81.90 MB	77.50 MB	5.67	103.68 MB	92.36 MB	12.25

As α represents the memory access pattern, different different α values lead to different working set size. For VMs with the same α, their working set size are close to each other. For example, the real working set sizes of multimedia-VM and social-VM are respectively 65.91MB and 63.50MB when $\beta = 2$, which are very close to each other. As shown in the table, the average size difference between the estimated and real working set sizes is no more than 10% on average. For multimedia-VM, the average difference is $\leq 2\%$, showing that equation (1) can set the timeout interval accurately.

Fig. 5 shows the working set hit rate based on equation (3). As the working set pages are first recovered to resume the running of a VM, a high working set hit rate is quite important to improve recovery performance. As shown in the figure, a higher β can achieve higher hit rate. When $\beta = 4$, the hit rate is higher than 95% on average. The main reason behind this is that a larger β leads to a long timeout interval, and finally results in a larger working set. Therefore, based on locality, pages the VM accessed during the recovery will have a higher chance to be part of the working set. However, a larger working set can occupy more NVRAM space. According to TABLE III and Fig. 5, we find that setting β to 3 can strike a balance between working set size and hit rate, in which the size of the working set is moderate while we still maintain a high hit rate.

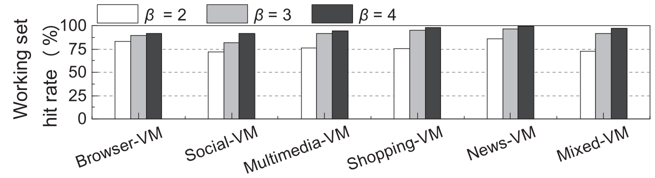
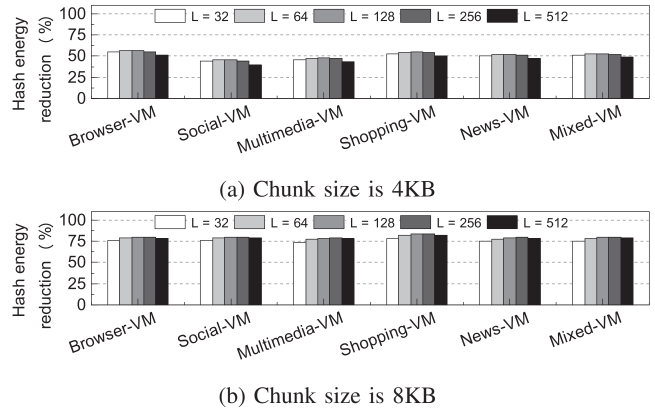


Fig. 5: Working set hit rate under different $\boldsymbol{\beta}$ values.



(a) Chunk size is 4KB

(b) Chunk size is 8KB

Fig. 6: Percentage of reduced energy using energy-aware hashing of FLIC with different sample lengths and chunk sizes, compared to full hashing.

B. Energy-aware data deduplication

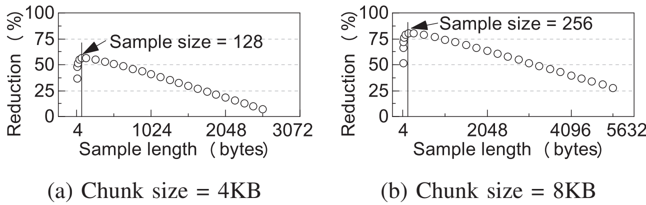
Energy consumption is one of the major main concerns for deduplication in mobile devices. The hash function is the key component and also is the most energy-intensive part of deduplication. As the energy consumed by the hash function is proportional to the input data size, we compute the energy reductions of our deduplication scheme according to the total size of the hashed data. Fig. 6 shows the energy reductions when we compare our hashing method to full hashing. Note that "L" in the figure denotes the length of the samples in byte for partial hashing. As shown in the figure, our approach can reduce the energy consumption of the hashing by around 50% with 4KB chunks. Since the size of the data for computing the fingerprint is reduced when size of the data chunk grows, our approach can reduce more energy. When chunk size is set to 8KB, we can reduce more than 75% energy consumption. Therefore, when the system's energy is not sufficient, we suggest to use a larger chunk size to save energy.

We also observe that sample size in partial hashing impacts energy savings. For 4KB chunks, as shown in Fig. 6(a), using a sample length of 64 or 128 bytes can achieve the maximum energy saving. For 8KB chunks, as shown in Fig. 6(b), the maximum energy saving comes with a 128 or 256 bytes samples. To show the trends more clearly, we use the browser-VM as an example and plot the energy reduction trends in Fig. 7 when using different sample sizes. As shown in the figure, when the sample sizes are respectively 128 and 256 bytes, using 4KB and 8KB chunks can achieve the maximum energy reduction.

TABLE IV lists the deduplication ratio and duplicate size for each VM with different chunk sizes. As shown in the table, with 4KB chunks, the VM snapshots size can be reduced by around 20.9%–28.3%. Which means a significant amount of write activities reduction to flash memory. However, with 8KB chunks, deduplication becomes less effective, reducing snapshot size by around 4.9%–7.6%. This is mainly because 4KB is the size of a physical page in our hardware and can provide proper size to split the VM snapshots. We therefore recommend using 4KB chunks to eliminate redundancy. Although 8KB chunks can only eliminate a small amount of duplicate pages, it consumes much less energy, which can be used when the system is running out of energy.

V. Related Work

Various studies have been conducted to enable virtualization in mobile devices. For example, KVM has been ported to the ARM architecture to run unmodified OSes [2], [20]. To optimize checkpointing/recovery of VM in virtualized mobile systems, Park et al. [21] proposed a fast and space-efficient VM checkpointing technique. Zhang et al. [22] proposed to accelerate the recovery process using working set estimation.



(a) Chunk size = 4KB (b) Chunk size = 8KB

Fig. 7: The trends of hash energy reduction with different sample lengths.

TABLE IV: Deduplication ratio and duplicate size for each VM with different chunks sizes.

VM name	Chunk size = 4KB		Chunk size = 8KB	
	Deduplication ratio (%)	Duplicate size (MB)	Deduplication ratio (%)	Duplicate size (MB)
Browser-VM	20.90	107	7.62	39
Social-VM	28.32	145	7.03	36
Multimedia-VM	25.39	130	7.03	36
Shopping-VM	21.29	109	4.88	25
News-VM	22.27	114	6.64	34
Mixed-VM	21.48	110	6.45	33

Other existing optimization methods such as unnecessary memory exclusion aim to reduce the checkpointed memory size. In this paper, we optimize the checkpointing of VM by storing working set pages in NVRAM, instead of dumping the VM's entire memory pages to flash memory. Moreover, we propose an energy-ware data deduplication scheme is to reduce checkpoint size.

Other related work lies in area of working set estimation. The working set is defined as the collection of the most recently used pages [19] for the OS or some process. The majority of related work aimed at identifying working set memory pages [23], [24], [25]. Similar to the working set estimator proposed in [21], working set identification scheme of FLIC tries to predict the collection of pages which the VM accessed during a specific period of time. However, different from [21], FLIC adopts two-phase scanning, and the time elapsed between the two scans is configurable. To achieve accurate estimations, both access pattern and non-working set recovery time need to be taken into consideration to determine the timeout interval.

VI. Conclusion

In this paper, we have proposed FLIC, a fast and lightweight checkpointing scheme for mobile virtualization system using NVRAM based main memory. Instead of writing the whole VM memory to flash memory, FLIC saves the working in fast, byte-addressable NVRAM. Experimental results show that FLIC can achieve a very high accuracy of more than 97%. To further reduce write activities to flash memory, we have proposed an energy-aware data deduplication scheme for FLIC to eliminate redundant data in VM snapshots. Instead of hashing full data chunks, we use partial hashing to reduce the energy consumption of hashing. In partial hashing, we sample the data chunk at fixed positions with fixed length to generate partial hash values. Experimental results show that our approach is able to reduce the energy consumption of hashing by around 50% and 80% for 4KB and 8KB chunks with 128 byte and 256 byte samples, respectively.

Acknowledgments

The work described in this paper is partially supported by the grants from the National Natural Science Foundation of China (Project 61272103, 61309004, and 61373049), National 863 Program 2015AA015304, Research Fund for the Doctoral Program of Higher Education of China (20130191120030), Chongqing High-Tech Research Program cstc2013jcyjA40025, Fundamental Research Funds for the Central Universities (CDJZR14185501, 0214005207005), Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 152138/14E and GRF 15222315/15E), and the Hong Kong Polytechnic University (4-ZZD7, G-YK24, G-YM10 and G-YN36).

References

[1] K. Barr, P. Bungale, S. Deasy, V. Gyuris, P. Hung, C. Newell, H. Tuch, and B. Zoppis, "The VMware mobile virtualization platform: is that a hypervisor in your pocket?" *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 124–135, 2010.

[2] C. Dall and J. Nieh, "KVM/ARM: the design and implementation of the linux ARM hypervisor," in *ASPLOS*, 2014, pp. 333–348.

[3] C. Dall, J. Andrus, A. Van't Hof, O. Laadan, and J. Nieh, "The design, implementation, and evaluation of cells: A virtual smartphone architecture," *ACM Trans. on TOCS*, vol. 30, no. 3, pp. 9:1–9:31, 2012.

[4] L. Long, D. Liu, X. Zhu, K. Zhong, Z. Shao, and E.-M. Sha, "Balloonfish: Utilizing morphable resistive memory in mobile virtualization," in *ASP-DAC*, 2015, pp. 322–327.

[5] W. Zhao and Z. Wang, "Dynamic memory balancing for virtual machines," in *VEE*, 2009, pp. 21–30.

[6] Y.-C. Lee and C.-W. Hsueh, "An optimized page translation for mobile virtualization," in *DAC*, 2013, pp. 85:1–85:6.

[7] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Computing Surveys*, vol. 45, no. 2, pp. 17:1–17:39, 2013.

[8] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Trans. on Storage*, vol. 8, no. 4, pp. 14:1–14:25, 2012.

[9] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[10] H. Khouzani, C. Yang, and J. Hu, "Improving performance and lifetime of dram-pcm hybrid main memory through a proactive page allocation strategy," in *ASP-DAC*, 2015, pp. 508–513.

[11] D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E.-M. Sha, "Application-specific wear leveling for extending lifetime of phase change memory in embedded systems," *IEEE TCAD*, vol. 33, pp. 1450–1462, 2014.

[12] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: spin-RAM," in *IEDM*, 2005, pp. 459–462.

[13] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *CODES+ISSS*, 2011, pp. 325–334.

[14] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, 2008.

[15] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training itself: Mixed-signal training acceleration for memristor-based neural network," in *ASP-DAC*, 2014, pp. 361–366.

[16] S. Li, A. Li, Y. Zhe, Y. Liu, P. Li, G. Sun, Y. Wang, H. Yang, and Y. Xie, "Leveraging emerging nonvolatile memory in high-level synthesis with loop transformations," in *ISLPED*, 2015, pp. 61–66.

[17] "Arndale board Exynos 5250," http://www.arndaleboard.org/wiki/index.php/Main_Page, 2014.

[18] K. Zhong, T. Wang, X. Zhu, L. Long, D. Liu, W. Liu, Z. Shao, and E. H.-M. Sha, "Building high-performance smartphones via non-volatile memory: The swap approach," in *EMSOFT*, 2014, pp. 30:1–30:10.

[19] P. J. Denning, "The working set model for program behavior," *Communications of the ACM*, vol. 11, no. 5, pp. 323–333, 1968.

[20] C. Dall and J. Nieh, "KVM for ARM," in *Linux Symposium*, 2010.

[21] E. Park, B. Egger, and J. Lee, "Fast and space-efficient virtual machine checkpointing," in *VEE*, 2011, pp. 75–86.

[22] I. Zhang, A. Garthwaite, Y. Baskakov, and K. C. Barr, "Fast restore of checkpointed memory using working set estimation," in *VEE*, 2011, pp. 87–98.

[23] S. Bansal and D. S. Modha, "CAR: Clock with Adaptive Replacement," in *FAST*, 2004, pp. 187–200.

[24] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance," in *SIGMETRICS*, 2002, pp. 31–42.

[25] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement," in *USENIX ATC*, 2005, pp. 35–35.