Improving Flash Memory Performance and Reliability for Smartphones With I/O Deduplication

Bo Mao, *Member, IEEE*, Jindong Zhou, Suzhen Wu, *Member, IEEE,* Hong Jiang, *Fellow, IEEE*, Xiao Chen, and Weijian Yang

***Abstract*—Flash-based storage subsystem is the key component that affects the system performance, reliability, and cost efficiency of Android-based smartphones. In this paper, we first introduce a trace collection tool specifically designed to capture the I/O requests with important content features in Android-based smartphones, which are critically important but rarely available in content-aware designs and optimizations, such as JProbe and Netlink. Based on the analysis of the traces collected from 15 popular mobile applications, we find that 20%–40% of the I/O requests on the I/O critical path of the storage stack are redundant and this data redundancy is minimally shared among different applications. Based on this key observation, we propose a content-aware optimization, called APP-Dedupe, that applies data deduplication on the I/O critical path to improve both performance and efficiency by reducing write amplification and improving GC efficiency of the flash storage on Android smartphones. The evaluation results show that APP-Dedupe reduces the GC overhead by an average of 41.5%, reduces the response times by up to 15.4% and reduces the amount of write data by an average of 45.2%.**

***Index Terms*—Flash-based storage, I/O deduplication, smartphones, trace collection.**

I. Introduction

STORAGE is one of the key factors affecting the overall system performance and reliability of the Android-based smartphones [6], [18], [19], [28]. The mobile applications, often referred to as "APP," in Android-based smartphones generate I/O requests that have different characteristics than those generated by nonmobile applications. The storage subsystem of smartphones usually relies on flash-based embedded multimedia controller (eMMC) memory, with either a disk file system (such as Ext4) or a flash file system (such as F2FS [23]).

Generally speaking, the storage stack of current smartphones faces three challenges. First, the performance tends to degrade after repeated usages, particularly writes, due to the physical characteristics of the flash memory, also one of the reasons why smartphones slow down over time [5], [14], [40]. Second, one of these physical characteristics of flash memory is its limited life cycles [49], i.e., the number of times each cell can be programmed/written before it fails, and causes the flash storage to get sluggish after repeated usages, which affects the storage reliability of smartphones. Third, the cost of upgrading the flash capacity from one level to the next level, e.g., from 16 to 32 GB, amounts to nearly 100 USD for most smartphones (for example, Apple iPhones). Therefore, these challenges, pointing to the measures of performance, reliability, and cost, suggest that it is important to 1) understand the mobile applications and how they interact with the flash-based eMMC and 2) optimize to reduce write traffic to the flash-based eMMC in smartphones.

Data deduplication and its applications in flash-based storage systems have been well studied in [36] and [45]. Some studies have shown that by leveraging the deduplication technology to reducing the write traffic, the system performance and reliability of the storage stack of conventional, nonmobile systems can be significantly improved [4], [13]. However, the unique characteristics of mobile devices and mobile applications make straightforwardly applying deduplication in Android-based smartphones both less effective and more challenging [21]. For example, the mobile devices have much smaller memory capacity than nonmobile systems, which implies that mobile applications must be made memory efficient to attain acceptable performance. On the other hand, the user-facing nature of smartphones implies, and confirmed by our experimental observation, that only one application is usually running in the foreground while all the other opened applications are hung up in the background in the Android-based smartphones.

To make data deduplication effective and efficient in smartphones, we need to understand and gain insight into the data redundancy and unique characteristics of mobile applications by collecting and analyzing content-aware application traces. Unfortunately, due to the dataset privacy leakage risk, content-aware trace collections are rarely done in storage systems [12],

Manuscript received November 11, 2017; revised January 14, 2018 and March 3, 2018; accepted April 5, 2018. Date of publication May 8, 2018; date of current version May 20, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61772439, Grant U1705261, and Grant 61472336, and in part by the U.S. NSF under Grant CCF-1704504 and Grant CCF-1629625. This paper was recommended by Associate Editor Z. Shao. *(Corresponding author: Bo Mao.)*

B. Mao and J. Zhou are with the Software School of Xiamen University, Xiamen 361005, China (e-mail: maobo@xmu.edu.cn).

S. Wu, X. Chen, and W. Yang are with Computer Science Department, Xiamen University, Xiamen 361005, China (e-mail: suzhen@xmu.edu.cn).

H. Jiang is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: hong.jiang@uta.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2834395

0278-0070 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

let alone publicly available traces with content features, with the exception of the FIU department traces available in the SNIA trace repository [22]. To address this problem, we design a low-overhead content-aware trace collection tool, which can be used both offline and online. Using this tool, we collected traces of 15 popular mobile applications. Our workload analysis of 15 popular mobile applications reveals that an average data redundancy of 33.1% exists in mobile applications but this redundancy is minimally shared among these applications.

Therefore, we propose APP-Dedupe for Android-based smartphones to address the aforementioned challenges. Instead of treating data chunks from all application streams equally, APP-Dedupe organizes the hash (fingerprint) index in an application-aware way, effectively grouping the hash index of the same application (short for APP) together and dividing the whole hash index into different groups based on the application types. When an application is running in the foreground, APP-Dedupe loads the corresponding hash index of the application into the memory, thus improving the efficiency of the memory for the hash index. Moreover, it groups the data chunks of the same application together on the flash to alleviate the read amplification problem (i.e., fragmentation caused by deduplication) and exploits the spatial locality to improve the read performance. The extensive trace-driven experiments conducted on our lightweight prototype implementation of APP-Dedupe show that APP-Dedupe reduces the GC overhead by an average of 41.5%, reduces the response times by up to 15.4% and reduces the amount of write data by an average of 45.2%.

The main contributions of this paper are threefold.

1) We design a low-overhead content-aware trace collection tool that captures the I/O requests in the storage stack in Android-based smartphones.
2) We collect the traces with content features from 15 popular applications and perform in-depth I/O analysis. We find that 20% to 40% of the I/O requests on the I/O critical path of the storage stack are redundant and this data redundancy is minimally shared among different applications. To the best of our knowledge, currently no such study exists for the Android-based smartphones.
3) We propose APP-Dedupe to improve the storage efficiency of Android-based smartphones. The trace collection tool, the 15 traces and the image of the prototype system are available for academic purposes.

The rest of this paper is organized as follows. Section II presents the trace collection tool and workload characteristics in Android-based smartphones. The design and implementation of APP-Dedupe is presented in Section III. Section IV describes the performance results through the extensive evaluations on the APP-Dedupe prototype. The conclusion is given in Section V.

II. Trace Collection and Analysis

In this section, we first present the design of a content-aware trace collection tool in Android-based smartphones. Then we analyze the workload characteristics of the traces collected by this tool to motivate our App-Dedupe study.

A. Content-Aware Trace Collection

While trace collections on enterprise storage systems have been well studied, there is limited effort on trace collection in mobile systems. In particular, currently only MOST [16] and BIOtracer [48] are designed for I/O trace collections in Android-based smartphones and they only capture the I/O requests behaviors (e.g., size, read/write patterns, etc.) without including any content values or features. Yet, it is the content features of the traces that enable one to analyze the data redundancy characteristics of I/O accesses in Android-based smartphones. For this reason, we design a content-aware trace collection tool, called *MobileCT*. MobileCT collects the traces that contain the basic I/O request information, the process names and the content features. First, it captures the basic information of an I/O request via the bio structure, including time, R/W, offset, and size, and copies the data of the request for the subsequent hash computing of the content. Second, it explores the Linux kernel tracepoint infrastructure to track requests in-flight through the block I/O stack, similar to the blktrace infrastructure which can capture the process ID (PID) and process name information [38]. Based on the process information, the specific application is recorded by MobileCT. Third, it splits the data into 4KB chunks and calculates the MD5 fingerprint for each chunk. Finally, the basic information and hash values of the I/O requests are recorded in the trace file and transferred to the user level.

There are two major challenges facing content-aware trace collection in Android-based smartphones, namely, how to anonymize the contents to protect the privacy of the users [36] and how to minimize the interference between the trace capturing operations and the user I/O requests to reduce collection overhead [48]. For the first privacy challenge, similar to the FIU traces [22], we also use the MD5 fingerprints to represent the content feature for the deduplication research without leakage of the location or personal information [46]. For the overhead challenge, since the processing resources in Android-based smartphones are limited and hash computing for chunk fingerprints can be resource demanding, the aforementioned interference, if not avoided or minimized, can adversely affect the application performance and/or the accuracy of the traces. To address this challenge, MobileCT uses a circular buffer to temporarily store the write data and delay the subsequent MD5 computing of the data chunks to avoid the interference and contention on the CPU resources. Fig. 1 shows the trace collection workflow in MobileCT. JProbe is a servlet for inspecting the bio→end_io() function and Netlink_sock is used for the data transfer between the user space and the kernel space.

B. Workload Characteristics

The traces presented in this paper are collected from 15 applications on the Google Nexus 5 smartphone (running Android 5.0.1 with Linux Kernel 3.4). We compare the chunk fingerprints of the 15 data sets using the chunk-level deduplication with 4 KB chunk size. The trace characteristics are summarized in Table I, which shows that the data redundancy of the mobile applications is between 20% to 40%, with an average of 33.1%. Of particular interests are the findings

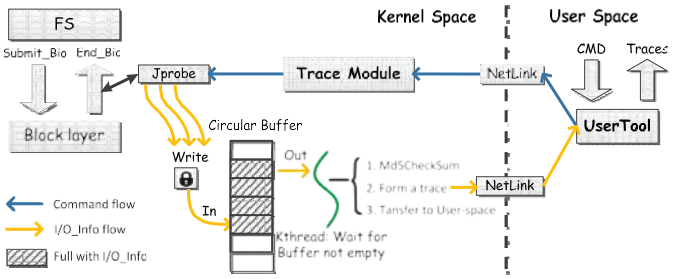


Fig. 1. Trace collection workflow in MobileCT.

TABLE I
KEY CHARACTERISTICS OF THE 15 MOBILE APPLICATIONS

APPs	*Redundancy*	*IOPS*	*Size (MB)*	*Write Ratio*
Qiu	40.5%	1.6	55.0	87.9%
58City	32.2%	2.6	153.3	69.4%
Baidu Tieba	39.5%	4.3	213.5	88.6%
Game2048	31.7%	1.7	63.2	89.2%
Meitu	31.3%	5.6	131.6	41.9%
Moji Weather	32.4%	2.6	82.1	63.4%
Opera Browser	33.1%	3.0	72.7	57.5%
Fruit Cool	36.6%	6.0	159.9	47.1%
Sohu News	32.2%	2.0	138.2	84.2%
Tencent	34.1%	1.5	82.6	95.4%
Pea Pod	35.4%	2.1	237.0	67.8%
Wechat	30.0%	4.6	345.5	90.4%
Weibo	29.0%	1.4	97.0	88.5%
Xiami Music	27.3%	7.4	122.9	39.4%
Youdao Dict	31.9%	9.2	136.5	35.2%

that the IOPS is less than 10 for all the 15 applications and the write requests dominate in the mobile applications. These findings of workload characteristics are consistent with the previous studies of mobile applications [16], [48].

The most interesting finding, as shown in Table II, is that the amount of data redundancy shared between any two different mobile applications, or the percentage of shared redundancy, is minimal. The percentage of shared redundancy, the percentage entry in the table, is the percentage ratio of the number of common redundant data chunks between the two applications (row and column) to the total number of data chunks of the row application. From Table II, we can see that the redundant data chunks shared by any two different applications are less than 5% for most cases, which implies that the amount of redundant data shared by different applications is negligible. The reason is that different applications usually have different data contents and data formats. The results are consistent with the previous studies on nonmobile applications [9], [11], [43] and indicate that there is very little overlap between hash indexes of any two different applications. The significance of this finding is that it makes it possible to effectively group the hash index of the same application together and partition the whole hash index into multiple small segments according to the application types. This in turn helps optimize hash index locality for optimal cashing efficiency.

C. Motivation

The flash storage affects the system performance for the mobile applications in Android-based smartphones [7], [14], [18], [26]. Recent studies have shown that the overlap between the file system journaling (such as EXT4) and the database journaling (such as SQLite) activities, also referred to as journal of journaling [20], [24], [35], is the root cause of the inefficiency for the flash storage in smartphones. Existing solutions to this problem to either reduce the journaling overhead in the SQLite database, such as reducing the SQLite journaling I/Os through multiversion B-tree [20], minimize the synchronization overhead, such as WALDIO [24], or optimize the file systems, such as MobiFS [33] and F2FS [23]. However, none of them exploits the content redundancy characteristics in the mobile storage subsystems, which has the potential to improve not only application performance but also system reliability and efficiency (space and energy).

In this paper, we revisit the flash performance issue in smartphones from a fresh perspective based on the content feature analysis of the mobile applications. Recent studies in the literature indicate that reducing the amount of I/Os on the critical path can be much more effective and efficient than optimizing I/Os in storage systems [27]. I/O deduplication on the I/O critical path can reduce the I/O traffic at the cost of computing and memory overhead. Motivated by the importance of the mobile storage space, performance and reliability, combined with the observations from the workload studies, we propose APP-Dedupe to improve the storage efficiency of Android-based smartphones.

III. DESIGN AND IMPLEMENTATION OF APP-DEDUPE

In this section, we first outline the main design objectives of the APP-Dedupe system. Then we present the architecture overview and design details of APP-Dedupe.

A. Design Objectives of APP-Dedupe

The design of APP-Dedupe aims to achieve the following three objectives.

1) *Improving the Applications' Performance:* By applying I/O deduplication on the critical I/O path, APP-Dedupe is designed to detect and remove a significant amount of redundant write data, thus effectively filtering out redundant write requests and improving I/O performance of smartphones.
2) *Improving the Space Efficiency:* By using I/O deduplication to remove a significant amount of redundant write data, the overall space efficiency is improved for flash storage within smartphones.
3) *Improving the Flash Reliability:* The write traffic and the amount of stored data on the flash subsystem are significantly reduced by APP-Dedupe. This leads to the number of block erase cycles to be significantly reduced, which improves the flash reliability accordingly.

B. Architecture Overview

Fig. 2 shows a system architecture overview of our proposed APP-Dedupe in the context of the storage subsystem in the Android-based smartphones. APP-Dedupe sits below the file system and the SQLite database and thus can be easily incorporated into any existing platforms to accelerate their storage subsystem performance. It must be noted that SQLite is

TABLE II
PERCENTAGE OF SHARING OF DATA REDUNDANCY BETWEEN ANY TWO DIFFERENT APPLICATIONS. NOTE: DUE TO THE SPACE LIMIT, ONLY THE FIRST WORD OF AN APPLICATION'S NAME IS PRESENTED

APPs	*Qiu*	*58City*	*Baidu*	*Game2048*	*Meitu*	*Moji*	*Opera*	*Fruit*	*Sohu*	*Tencent*	*Pea*	*Wechat*	*Weibo*	*Xiami*	*Youdao*
Qiu	**40.5%**	0.2%	1.0%	0.1%	0.3%	0.0%	0.1%	0.4%	0.2%	0.0%	0.1%	0.5%	1.5%	0.2%	0.3%
58City	0.3%	**32.2%**	3.3%	1.3%	3.6%	3.2%	4.3%	3.9%	4.9%	1.2%	3.9%	2.4%	1.6%	4.2%	3.5%
Baidu	0.4%	6.5%	**39.5%**	0.2%	2.4%	5.5%	7.1%	4.4%	3.9%	0.0%	3.5%	3.2%	3.6%	3.6%	6.4%
Game2048	0.3%	0.7%	1.0%	**31.7%**	1.0%	2.0%	2.9%	1.5%	0.7%	2.0%	1.4%	0.3%	1.5%	1.0%	2.4%
Meitu	0.3%	3.1%	2.2%	1.3%	**31.3%**	2.2%	5.9%	6.4%	1.5%	1.1%	2.3%	1.3%	1.6%	4.4%	3.1%
Moji	0.1%	2.7%	4.7%	2.8%	2.3%	**32.4%**	6.5%	5.5%	3.5%	0.8%	3.5%	1.4%	1.5%	3.8%	3.8%
Opera	0.2%	1.5%	2.7%	3.4%	3.7%	5.7%	**33.1%**	4.8%	3.0%	1.7%	1.9%	2.1%	1.1%	1.4%	4.2%
Fruit	0.3%	1.1%	2.5%	3.2%	3.8%	6.4%	4.7%	**36.6%**	2.6%	0.5%	2.2%	2.6%	2.0%	1.7%	2.7%
Sohu	0.4%	5.9%	3.5%	1.3%	1.6%	5.3%	4.5%	3.8%	**32.2%**	1.1%	2.5%	3.3%	4.6%	2.5%	4.1%
Tencent	0.0%	0.8%	0.0%	2.7%	1.8%	0.8%	2.1%	0.6%	0.8%	**34.1%**	0.5%	0.2%	0.6%	2.2%	2.8%
Pea	0.2%	6.1%	2.9%	3.9%	4.5%	7.3%	4.5%	5.0%	3.6%	1.2%	**35.4%**	2.1%	1.1%	6.2%	3.9%
Wechat	1.3%	3.5%	4.8%	0.1%	0.7%	4.2%	3.5%	5.5%	4.3%	0.2%	1.8%	**30.0%**	5.0%	1.9%	3.0%
Weibo	0.4%	0.4%	1.5%	0.0%	0.6%	0.4%	0.2%	0.9%	1.4%	0.2%	0.1%	2.1%	**29.0%**	0.4%	0.5%
Xiami	0.2%	5.0%	2.6%	3.3%	5.6%	6.5%	3.6%	3.0%	2.9%	9.3%	6.1%	2.1%	1.3%	**27.3%**	3.6%
Youdao	0.2%	1.7%	2.7%	5.8%	3.0%	4.6%	5.4%	3.7%	3.2%	2.1%	5.2%	2.1%	1.2%	1.4%	**31.9%**

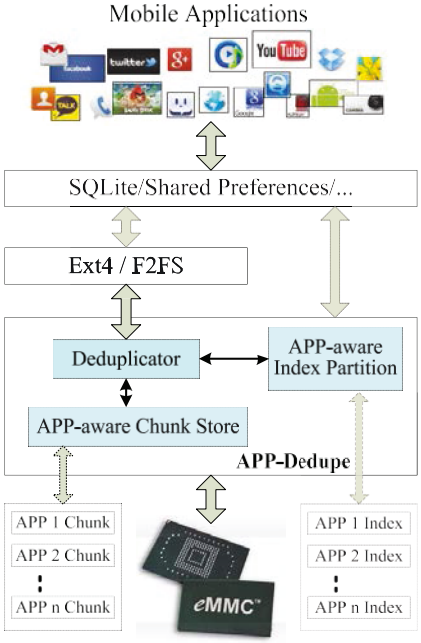


Fig. 2. System architecture of APP-Dedupe.

a popular data storage option in the Android platform to store nontrivial amounts of structured data. However, Android also provides several other data storage options, which are explained in detail on the Android Developer site, to store persistent data, such as shared preferences, internal/external storage, and network connections [8]. APP-Dedupe is independent of the upper file system, which makes APP-Dedupe amenable to be deployed in a variety of environments, including the newly proposed MobiFS [33] and F2FS [23]. Moreover, APP-Dedupe explores the Linux kernel tracepoint infrastructure to track the PID and process name information in-flight through the block I/O stack [38]. The PID and process name information is used to infer the specific application of the request.

APP-Dedupe has three main functional components: 1) Deduplicator; 2) APP-aware index partition; and 3) APP-aware chunk store. The Deduplicator module is responsible for splitting the incoming write data into data chunks, calculating the hash value of each data chunk and identifying whether a data chunk is a duplicate. The APP-aware index partition (short for AIP) module divides the whole hash index into small subsets based on the application types. When an application is in the active state, the corresponding hash index subset is loaded from the back-end eMMC storage into the memory. AIP swaps out the cached hash index to the back-end eMMC storage when the corresponding application is hung out in the background. The APP chunk store (short for ACS) module groups the data chunks of the same application together to alleviate the data fragmentation problem [17] by fully leveraging the spatial locality of the user accesses.

C. APP-Aware Index Partition

Fig. 3 illustrates the write workflow in APP-Dedupe. There are two key data structures used to deduplicate and redirect the I/O requests, and identify the popular hash index entries, namely, *Map_table* and *App_index_table*, as shown in Fig. 3. While Map_table keeps all the information of the deduplicated write requests whose write data are already stored on the back-end eMMC storage, App_index_table maintains the fingerprints of the data chunks according to the specific application. The mapping between the items in the Map_table and the items in the APP_index_table is *many-to-1*. It means that a logical block address (LBA) can only be linked to a unique and distinctive physical data block, i.e., physical block address (PBA), but multiple LBAs can be linked to the same PBA.

In order to reduce the memory and processing overhead of storing and querying the large hash index, AIP only swaps the corresponding index subset in the memory when the application is in the active state. App_index_table is organized in an LRU form and maintains the frequency of write requests to each data chunk (PBA) by using the *Count* variable (initialized to "1"), as shown in Fig. 3. When a write request hits App_index_table, the count value of the corresponding index entry in App_index_table is increased by 1, which captures the temporal locality and frequency of write requests to this PBA. The Count variable is also used to prevent the referenced data blocks from being modified or deleted.

When a write request arrives, APP-Dedupe splits the write data into multiple fixed-size data chunks and calculates their fingerprints. If a fingerprint hits App_index_table, meaning that the corresponding data chunk is a duplicate, the

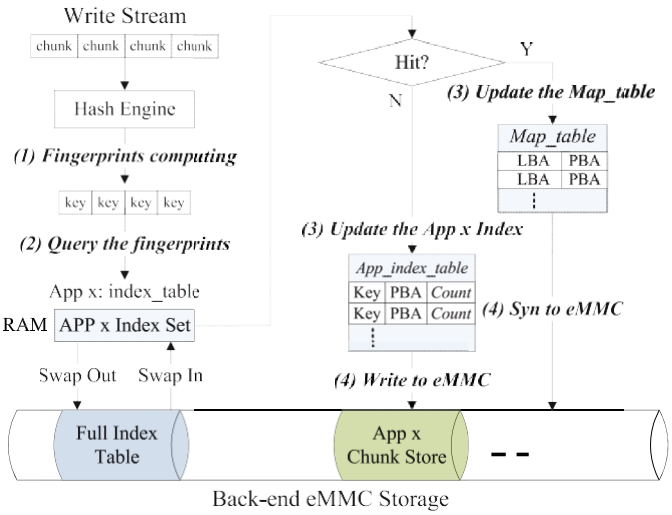


Fig. 3. Write workflow in APP-Dedupe.

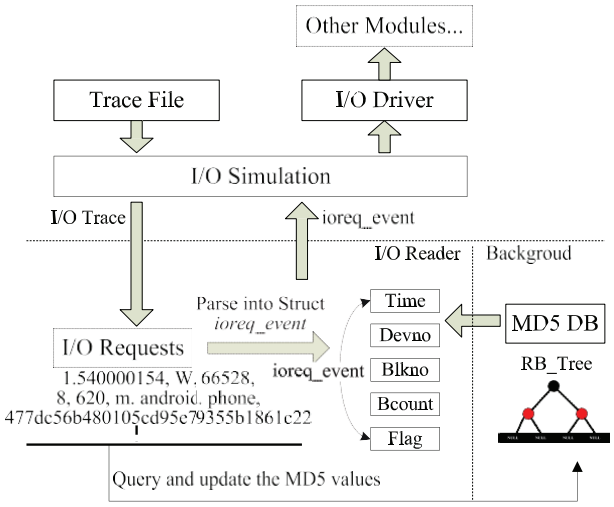


Fig. 4. Implementation of APP-Dedupe within DiskSim simulator.

corresponding Count value in App_index_table is incremented. APP-Dedupe only updates Map_table for the duplicate data chunks, and synchronizes Map_table to the back-end eMMC storage periodically. Otherwise, a new hash index entry is inserted into App_index_table and the data chunk is directly written to the back-end eMMC storage.

D. APP-Aware Chunk Store

Our experimental observation indicates that the amount of data redundancy shared by different applications is negligible and most data redundancy exists among the data chunks of the same application. To alleviate read performance degradation problem caused by the data fragmentation associated with data deduplication, the ACS module stores the data chunks of the same application in the same container. Moreover, by exploiting the semantic information of the file access correlation, the ACS module effectively groups the data chunks of these files together, thus allowing the subsequent read requests to fetch them in a single I/O request. In this way, the data fragmentation problem, which can degrade read performance (read amplification), is alleviated by concentrating the read accesses to a single container, thus improving the restore/read performance.

When a read request arrives at the block layer, having missed the upper-level cache, APP-Dedupe first checks whether the read request hits Map_table. If the read request misses Map_table, the read request is directly submitted to the block device layer. Otherwise, the address of the request will be replaced by one or more addresses according to Map_table, depending on whether the read request fully hits Map_table or not. If fully hit (i.e., all LBAs of the constituent data chunks of the request are found), the read request will be replaced by the new address in Map_table. Otherwise, partially hit (i.e., not all LBAs of the constituent data chunks of the request are found), the read request will be split into multiple new read requests based on the locations of the constituent data chunks of the original read request. Then, the newly generated read request(s) is (are) submitted to the block layer. After the completion of these read requests, the read data is reconstructed and returned to the upper layer.

E. APP-Dedupe Within DiskSim

An important design objective of APP-Dedupe is to improve the flash reliability within smartphones. However, in real smartphones, it is hard to get the internal GC statics, such as block erase count which is directly related to the flash reliability. In order to get the GC-statics within the flash memory, we embedded the APP-Dedupe within the SSD-based DiskSim simulator [2], [3] and replaying the content-based mobile traces. Fig. 4 shows the detailed implementation of APP-Dedupe within DiskSim simulator.

There are two main changes to the DiskSim simulator: 1) trace API and 2) MD5-based RB-Tree. First, to emulate the deduplication-based flash storage, we fill the DiskSim with the content-enhanced traces which are collected from real smartphones. Moreover, the APP flag is also extended to the request I/Os to identify different mobile applications. Since the main purpose of simulation is to evaluate the block erase count results within flash memory, the MD5 fingerprint computing latency is added based on a 4 KB data size in smartphones. Second, we extend the DiskSim simulator with a MD5-based RB-Tree to query the fingerprint and determine whether the write request is redundant or not. The redundant write data is not written to the flash memory, thus the total block erase count is reduced accordantly.

IV. Performance Evaluations

In this section, we first describe the experimental setup and methodology. Then we evaluate the performance of APP-Dedupe through both benchmark-driven and trace-driven evaluations.

A. Experimental Setup and Methodology

We implement a prototype of APP-Dedupe on the Google Nexus 5 smartphone, with Qualcomm MSM8974 Quadcore 2.3 GHz, 2 GB DRAM, 16 GB eMMC storage, and running

TABLE III
DEFAULT FLASH MODEL PARAMETERS

Parameter	Value
Total Capacity	16GB
Reserved Free Blocks	15%
Cleaning Policy	Greedy
Flash Chip Elements	32
Planes Per Package	2
Blocks Per Plane	1024
Pages Per Block	64
Page Size	4KB

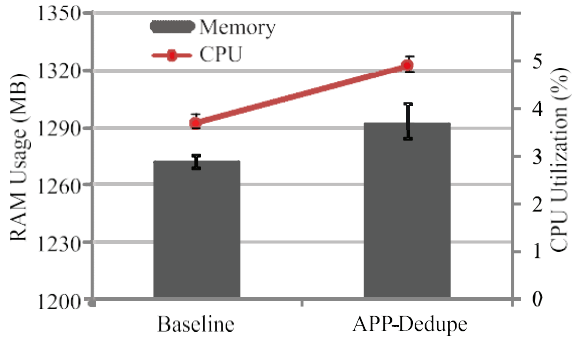


Fig. 5. Memory usage and CPU utilization under the benchmark evaluation driven by the Monkey tool.

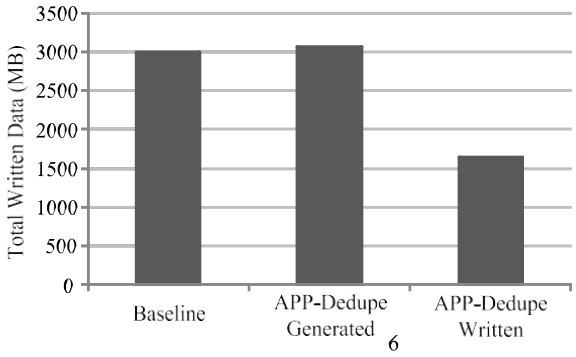


Fig. 6. Total amount of data written to the back-end eMMC storage driven by the Monkey tool.

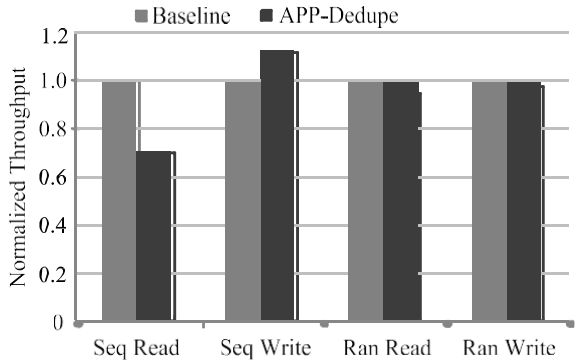


Fig. 7. System throughput normalized to the baseline system under different access patterns driven by the A1 SD Bench.

Android 5.0.1 with Linux Kernel 3.4. The system and application software is configured with the default settings without data deduplication as the baseline system. In order to evaluate the internal GC activities within eMMC, we also incorporate the deduplication functionality into the SSD-based DiskSim simulator [2]. The reserved free space is set to be 15% and the greedy cleaning policy is used in the simulation experiments. The values of the flash specific parameters used in the SSD-based DiskSim simulator are shown in Table III.

We use both the benchmark workload, i.e., the Monkey tool, and the trace replay workload to evaluate the effectiveness of APP-dedupe. The Monkey tool is a program that runs on the smartphones to generate for mobile applications pseudo-random streams of user events such as clicks, touches, and gestures, as well as a number of system-level events [30]. In our evaluation it runs to generate pseudo-random streams of user- and system-level events for the 15 applications listed in Table I. Moreover, A1 SD Bench is used to test the I/O read and write throughput [1]. The trace replay evaluations are driven by the 15 mobile traces that are shown in Table I.

B. Benchmark-Driven Evaluations

During the benchmark evaluations driven by the Monkey tool, we use the *iostat* command to monitor the CPU utilization and I/O statistics and the *dumpsys* command to monitor the memory usage. Fig. 5 shows the memory usage and CPU utilization. It indicates that, compared with the baseline system, APP-Dedupe incurs very little memory overhead, by no more than 2.5% and with an average of 1.6%. The reason is that APP-Dedupe only loads a subset of the hash index into the memory when the corresponding APP is running in the foreground. This memory overhead is expected to further diminish given the trend of increasing memory capacity in smartphones from one generation to the next. Similarly, the processing overhead, measured in CPU utilization, is also minimal, by no more than 2% on average. The reason is that the CPU resource in smartphones is usually idle during data transmission [32]. For example, a recent study has revealed that Android smartphones spend a significant portion of their CPU active time (up to 58%) waiting for storage I/Os to complete [32]. The very low memory and processing overheads measured here, combined with the fact that user interactions with smartphones are much less intensive than those with the server and enterprise environment, as evidenced in Table I and previous studies [16], [48], make the I/O deduplication a feasible solution in Android-based smartphones.

In the benchmark evaluation, we also compare the total amounts of written data for different schemes, as shown in Fig. 6. Note that "APP-Dedupe generated" means the total amount of data generated in the APP-Dedupe system while "APP-Dedupe Written" means the total amount of data written to the back-end eMMC storage in the APP-Dedupe system. First, APP-Dedupe generates a little more data than the baseline system because the deduplication operations incur some extra metadata overhead, including the MD5 fingerprints and the mapping information. However, the size of the increased data is minimal at less than 60 MB. Second, APP-Dedupe reduces the amount of data written to the back-end eMMC storage by an average of 45.2%. Since APP-Dedupe works on the I/O path, both the redundant data to different locations and the redundant write data to the same location are eliminated. The significant reduction in write data leads directly to

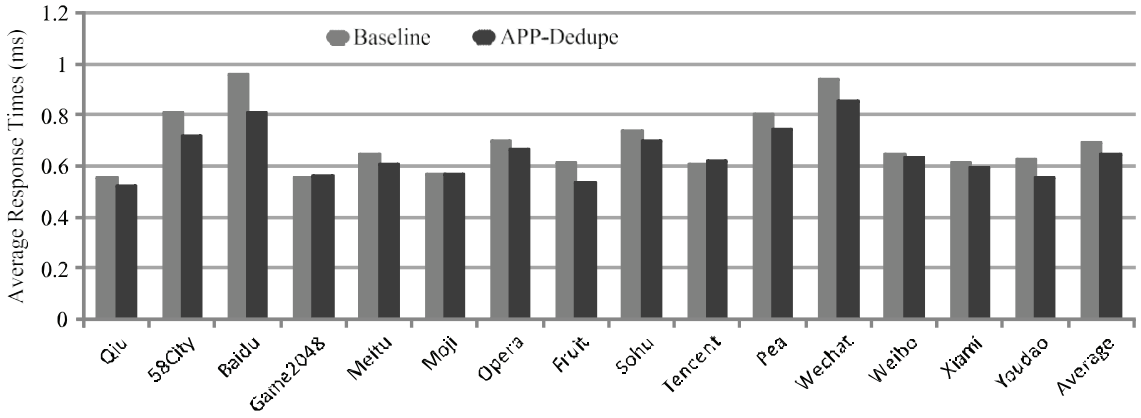


Fig. 8. Average response times in the evaluation driven by the 15 mobile traces.

a notably improved system throughput, storage efficiency and reduced cost. Reducing the write traffic to flash also directly improves the endurance of flash devices. Previous studies, such as CAFTL [4] and CA-SSD [13], have demonstrated that the flash reliability can be improved by reducing the redundant data on the write path. Recent studies on flash failure characteristics in Google and Facebook data centers have found that the total amount of data written directly affects the flash reliability [29], [34]. Thus, by reducing the write traffic, APP-Dedupe is shown to be able to improve the reliability of eMMC storage within smartphones, as detailed by the trace-driven study in Section IV-C.

Fig. 7 shows the system throughput normalized to the baseline system under different access patterns driven by the A1 SD Bench. APP-Dedupe improves the sequential write throughput by 11.5% but degrades the sequential read throughput by 30.0%. The reason for the degraded read performance stems from the fact that deduplication causes the data to be scattered across the flash memory, i.e., the known data fragmentation problem [27]. Though APP-Dedupe uses APP-aware chunk store to group the related chunks together, accesses to the deduplicated sequential read data are no longer sequential, unlike the system without deduplication. Moreover, since the bandwidth of eMMC in smartphones is much lower than that of the enterprise SSDs, the performance gap between random read and sequential read for eMMC is significant.

The increased write throughput comes from the reduced write data. In contrast, both the baseline system and the APP-Dedupe system perform similarly in random accesses. The reason is that eMMC's random access performance is much lower than its sequential access performance, which overshadows both the overhead and performance improvement of deduplication for the random accesses.

C. Trace-Driven Evaluations

Fig. 8 shows the average response times in the evaluation driven by the 15 mobile traces, indicating that APP-Dedupe reduces the average response times of the baseline system by up to 15.4% with an average of 6.2%. The reasons are threefold. First, APP-Dedupe removes a large portion of the redundant write requests. For these redundant write requests, only the metadata is updated without any disk I/O accesses. Thus, significantly reducing the write request delays. Second, by reducing the write traffic to the flash-based eMMC, the read/write interaction is alleviated. Since the write latency is much larger than read latency for flash memory, reducing the write traffic directly reduces the read latency. Third, by reducing the total write data to the flash-based eMMC, the internal block erase activities are also reduced, as that shown in the end of this section. The process time of an erase operation is an order of magnitude more than that of a read or write operation. The performance of the incoming user I/O requests during the GC period will be significantly degraded by the GC process due to the sever contention between these requests [44]. Reducing GC activities directly improves the system performance. Thus, both the read and write performances are improved.

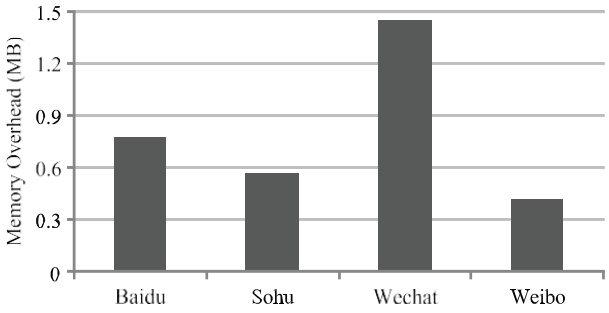


Fig. 9. Memory usages by the index table driven by the four traces.

On the other hand, it is interesting to notice that APP-Dedupe increases the average response times of the Game2048 and Tencent applications by 1.1% and 2.3%. The reason is that, while these two applications have relatively very small amount of write data and low IOPS to begin with, making the amounts of reduced write requests and GC activities very small and limiting the benefits of deduplication. These limited benefits from deduplication are more than offset by the hash computing overhead incurred by deduplication. The results on the two applications also imply that applying data deduplication within smartphones should be carefully designed to minimize the deduplication-induced memory and computing overhead.

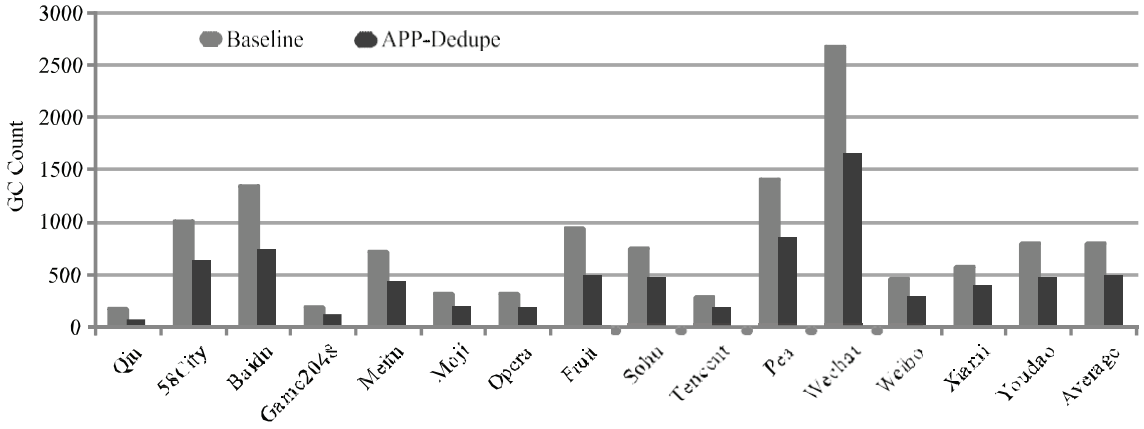


Fig. 10. Total block erase counts within the eMMC device in the evaluation driven by the 15 mobile traces.

Fig. 9 shows the memory usages by the hash index table driven by the four traces, Baidu, Sohu, Wechat, and Weibo. We can see that the increased memory consumption by each application is minimal. While memory usage increases with the increase of the stored data within each application, APP-Dedupe only loads in memory the hash index of the running application, thus making the increased memory overhead still acceptable. Moreover, even with the multiwindow multitasking mode enabled within the Android N-based smartphones [31], the memory usage by APP-Dedupe can still be affordable because only a small number of applications can run on the screen concurrently. Based on these results, the hash index of a specific application is small. Thus, it is possible and feasible to store the hash indices of a small number of applications in memory concurrently.

The block erase count within flash memory is directly related to the system reliability. Fig. 10 shows the total block erase counts within the eMMC device in the DiskSim evaluation driven by the 15 mobile traces, indicating that APP-Dedupe reduces the block erase counts by up to 53.8% with an average of 40.2%. The reason is that the GC frequency in flash memory is highly correlated to the amount of data written to it, i.e., the more the data written, the higher the GC frequency. The large fraction of write data reduced by APP-Dedupe leads directly to reduced GC activities [44], [47] within flash memory. Moreover, by reducing the block erase counts within flash-based eMMC, APP-Dedupe also improves the reliability and enhances the lifespan of the smartphone storage system [4].

The percentage of the reserved free blocks directly affects the GC activities within the flash memory. To evaluate the sensitivity of the GC efficiency to the percentage of the reserved free blocks, we conduct the experiments by changing this percentage under the Wechat and Weibo traces. Fig. 11 shows that APP-Dedupe has consistently reduces lower erase block count than the baseline system. The reason is that APP-Dedupe reduces the total amount of data actually written to the flash memory, which in turn reduces the number of invalid data blocks. As a result, the block erase count is reduced. However, with a larger percentage of free block count, the improvement brought by APP-Dedupe is seen to be reduced because the baseline system also has a much larger percentage of the reserved free blocks. On the other hand, we also see that with an increasing percentage of the reserved free blocks, the block erase count is also reduced accordantly. This is because more reserved free blocks lead to more blocks available to assist the GC activities, resulting in a reduced number of blocks that must be erased.

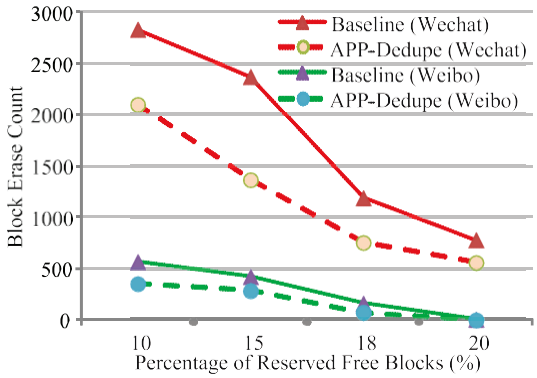


Fig. 11. Block erase counts with respect to the different percentages of the reserved free blocks driven by the Wechat and Weibo traces.

D. Extended Evaluations

To evaluate the effectiveness of APP-Dedupe in different Android smartphones, we also implement APP-Dedupe on the Huawei P9 smartphone with HUAWEI Kirin 955 Qcta core 2.5 GHZ, 4 GB DRAM, 64 GB eMMC storage, and running Android 6.0 with Linux Kernel 3.10.4. Fig. 12 shows the average response times of APP-Dedupe and Baseline driven by the mobile traces, indicating that APP-Dedupe reduces the average response time of the baseline system by up to 38.1% with an average of 29.2%, which is much better than that on the Google Nexus 5 smartphone. The obvious reason for this is that the Huawei P9 smartphone has significantly reduced overhead on computing fingerprints due to its much higher processing power of HUAWEI Kirin 955 than that of Qualcomm MSM8974. It also implies the good feasibility and applicability of APP-Dedupe on modern and future smartphones that are equipped with much more powerful processors.

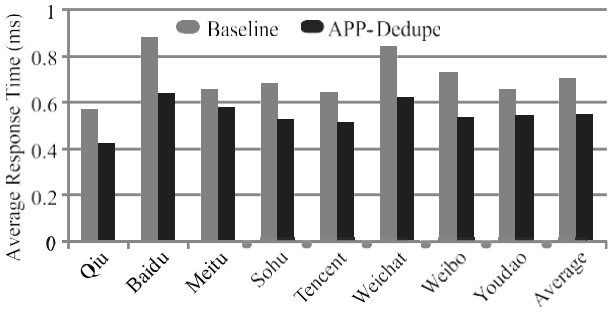


Fig. 12. Average response times on Huawei P9 smartphone driven by mobile traces.

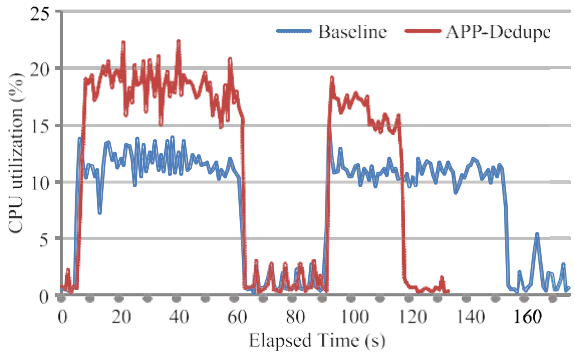


Fig. 13. CPU utilizations on Huawei P9 smartphone by writing a 4 GB file twice.

The CPU utilization is also an important factor of system performance and power consumption in smartphones. To evaluate the I/O intensity usage scenario, we write a 4 GB file twice with different file names and record the CPU utilizations. Fig. 13 compares the two schemes in terms of CPU utilization. First, we see that APP-Dedupe increases the CPU utilization by about 7% during I/O intensive periods when fingerprints are also computed. However, it performs similarly to the baseline system during system idle periods. Second, APP-Dedupe can significantly reduce the write time and the CPU utilization when the write data is redundant, as indicated in the time period when the file is being written for the second time. The results on the Huawei P9 smartphone not only clearly demonstrate APP-Dedupe's effectiveness but also shows that its performance advantage is more pronounced on more powerful smartphones.

V. Related Work

Data deduplication as a space-efficient technique has received a great deal of attention from both industry and academia. It has been demonstrated to be effective in cloud backup and archiving applications to reduce the backup window, improve the storage-space efficiency and network bandwidth utilization [45]. Recent studies have shown that moderate to high data redundancy also exists in primary storage systems and leverage the data deduplication technique to improve the I/O performance [22], [27], [37]. However, these studies are based on HDD-based storage which is different from the flash-based storage devices in terms of performance and reliability.

Due to the unique characteristics of flash memory, applying data deduplication within flash-based storage systems has also been well studied in [4], [13], and [45]. CAFTL [4] and CA-SSD [13] employ data deduplication to eliminate redundant write data to improve the endurance and performance of flash-based SSDs. Delta-FTL [42] uses delta compression to eliminate both duplicate and similar data to enlarge the logical space of flash memory within FTL layer. Nitro [25] implements data deduplication and data compression for SSD-based cache to enlarge the cache capacity, thus improving the cache efficiency. Moreover, data deduplication has become a commodity feature in flash-based storage products for many leading companies, such as Nimble Storage [41], Pure Storage [10], and Tintri [39], for the purpose of enhancing the system performance, reliability, and space efficiency.

However, the unique characteristics of mobile devices and mobile applications make straightforwardly applying deduplication in Android-based smartphones both less effective and more challenging. To the best of our knowledge, APP-Dedup is the first study that investigate the data deduplication on flash memory within smartphones. The smartphones have much smaller memory capacity and different application usage characteristics. Thus, we have designed a low-overhead content-aware trace collection tool and collected 15 mobile traces from popular mobile applications. The workload analysis results show that data redundancy exists in Android-based smartphones and presents some unique characteristics. Based on these workload characteristics, APP-Dedupe effectively groups the hash index of the same application together and dividing the whole hash index into different groups based on the application types.

VI. Conclusion

Flash-based storage subsystem in smartphones plays an important role in the application performance and system reliability. In this paper, we first investigate the data redundancy characteristics within Android-based smartphones by design a content-aware trace collection tool and collecting 15 mobile traces. From the trace analysis, we find that 20% to 40% of the I/O requests on the I/O critical path of the storage stack are redundant and this data redundancy is minimally shared among different applications. Based on this key observation, this paper proposes APP-Dedupe to detect and eliminate the I/O redundancy. The extensive benchmark-driven and trace-driven experiments conducted on our lightweight prototype implementation of APP-Dedupe show that APP-Dedupe reduces the GC overhead by an average of 41.5%, reduces the response times by up to 15.4% and saves the storage capacity by an average of 45.2%.

It is worth noting that our application of deduplication to the smartphone storage is still preliminary and an on-going research topic. As such, some research issues remain to be addressed as our future work. First, the issue of power consumption is critically important in smartphones [15], [32]. Deduplication is effective in reducing the I/O traffic and GC activities and thus has a potential to improve the storage

energy efficiency. On the other hand, the hash computing consumes extra processing power. As a result, another important objective of APP-Dedupe is to improve the energy efficiency, in addition to the performance improvement and capacity saving. As a direction of future work, we will investigate the power consumption issue associated with deduplication in smartphones. Second, from our experimental results, we observe that different applications have different data characteristics. Depending on such characteristics, deduplication may not always improve the performance. Thus, deduplication for the mobile applications should be dynamically enabled or disabled. We will investigate how to dynamically apply deduplication on the smartphone storage at runtime to improve the flexibility of APP-Dedupe.

ACKNOWLEDGMENT

The authors would like to thank the ASTL members for their continues support and discussion.

REFERENCES

[1] (2017). *A1 SD Bench—SD Card Benchmarking App*. [Online]. Available: http://a1dev.com/sd-bench/

[2] N. Agrawal *et al.*, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2008, pp. 57–70.

[3] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger. (May 2008). *The DiskSim Simulation Environment Version 4.0 Reference Manual*. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf

[4] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2011, p. 6.

[5] R. Chen *et al.*, "Image-content-aware I/O optimization for mobile virtualization," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 1, pp. 1–24, 2016.

[6] R. Chen *et al.*, "vFlash: Virtualized flash for optimizing the I/O performance in mobile devices," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 7, pp. 1203–1214, Jul. 2017.

[7] J. Courville and F. Chen, "Understanding storage I/O behaviors of mobile applications," in *Proc. IEEE 32nd Symp. Mass Storage Syst. Technol. (MSST)*, Santa Clara, CA, USA, May 2016, pp. 1–11.

[8] (2017). *Data Storage Options for Android Developers*. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html

[9] A. El-Shimi *et al.*, "Primary data deduplication—Large scale study and system design," in *Proc. 2012 USENIX Annu. Tech. Conf. (ATC)*, Jun. 2012, pp. 285–296.

[10] (2016). *FlashReduce Data Reduction in Pure Storage*, [Online]. Available: http://www.purestorage.com/flash-array/flashreduce.html

[11] Y. Fu, H. Jiang, N. Xiao, L. Tian, and F. Liu, "AA-Dedupe: An application-aware source deduplication approach for cloud backup services the personal computing environment," in *Proc. IEEE Int. Conf. Cluster Comput. (Cluster)*, Austin, TX, USA. Sep. 2011, pp. 112–120.

[12] R. Gracia-Tinedo *et al.*, "SDGen: Mimicking datasets for content generation in storage benchmarks," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2015, pp. 317–330.

[13] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2011, p. 7.

[14] S. Hahn *et al.*, "Improving file system performance of mobile storage systems using a decoupled defragmenter," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*, Jun. 2017, pp. 759–771.

[15] J. Huang, A. Badam, R. Chandra, and E. B. Nightingale, "WearDrive: Fast and energy-efficient storage for wearables," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2015, pp. 613–625.

[16] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/O stack optimization for smartphones," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2013, pp. 309–320.

[17] C. Ji *et al.*, "An empirical study of file-system fragmentation in mobile storage systems," in *Proc. 8th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, Jun. 2016, pp. 1–5.

[18] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2012, p. 17.

[19] S.-H. Kim, J. Jeong, and J. Lee, "Selective memory deduplication for cost efficiency in mobile smart devices," *IEEE Trans. Consum. Electron.*, vol. 60, no. 2, pp. 276–284, May 2014.

[20] W.-H. Kim, B. Nam, D. Park, and Y. Won, "Resolving journaling of journal anomaly in android I/O: Multi-version B-tree with lazy split," in *Proc. 12th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2014, pp. 273–285.

[21] Y. Kim, M. Imani, S. Patil, and T. S. Rosing, "CAUSE: Critical application usage-aware memory system using non-volatile memory for mobile devices," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, Nov. 2015, pp. 690–696.

[22] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," in *Proc. 8th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2010, pp. 211–224.

[23] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2015, pp. 273–286.

[24] W. Lee *et al.*, "WALDIO: Eliminating the filesystem journaling in resolvingthe journaling of journal anomaly," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2015, pp. 235–247.

[25] C. Li *et al.*, "Nitro: A Capacity-optimized SSD cache for primary storage," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf. (USENIX)*, Jun. 2014, pp. 501–512.

[26] H. Luo, H. Jiang, Z. Yan, and Y. Yang, "Fast transaction logging for smartphones," in *Proc. IEEE 32nd Symp. Mass Storage Syst. Technol. (MSST)*, May 2016, pp. 1–5.

[27] B. Mao, H. Jiang, S. Wu, and L. Tian, "POD: Performance oriented I/O deduplication for primary storage systems in the cloud," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2014, pp. 767–776.

[28] B. Mao, S. Wu, H. Jiang, X. Chen, and W. Yang, "Content-aware trace collection and I/O deduplication for smartphones," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol. (MSST)*, May 2017, pp. 1–8.

[29] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, Jun. 2015, pp. 177–190.

[30] (2017). *Monkey Tool*. [Online]. Available: https://developer.android.com/studio/test/monkey.html

[31] *Multi-Window Support–Android Developers*. [Online]. Available: https://developer.android.com/guide/topics/ui/multi-window.html

[32] D. T. Nguyen *et al.*, "Reducing smartphone application delay through read/write isolation," in *Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, May 2015, pp. 287–300.

[33] J. Ren, M.-J. M. Liang, Y. Wu, and T. Moscibroda, "Memory-centric data storage for mobile systems," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, Jun. 2015, pp. 599–611.

[34] B. Schroeder, R. Lagisetty, and A. Merchant, "Flash reliability in production: The expected and the unexpected," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2016, pp. 67–80.

[35] K. Shen, S. Park, and M. Zhu, "Journaling of journal is (almost) free," in *Proc. 12th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2014, pp. 287–293.

[36] P. Shilane, R. Chitloor, and U. Jonnala, "99 deduplication problems," in *Proc. 8th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, Jun. 2016, pp. 1–5.

[37] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2012, p. 24.

[38] (2017). *The Linux Kernel Tracepoint API*. [Online]. Available: https://www.kernel.org/doc/html/latest/core-api/tracepoint.html

[39] (2016). *Tintri VMstore*. [Online]. Available: http://info.tintri.com/vmstore-whitepaper

[40] (2015). *Why do Samsung Phones Slow Down After Time of Usage?*. [Online]. Available: https://www.quora.com/why-do-samsung-phones-slow-down-after-some-time-of-usage

[41] (2016). *With Nimble, Less Is More*. [Online]. Available: https://www.nimblestorage.com/its-all-about-data-reduction/

[42] G. Wu and X. He, "Delta-FTL: Improving SSD lifetime via exploiting content locality," in *Proc. 7th Eur. Conf. Comput. Syst. (EuroSys)*, Apr. 2012, pp. 253–266.
[43] S. Wu, X. Chen, and B. Mao, "Exploiting the data redundancy locality to improve the performance of deduplication-based storage systems," in *Proc. 22nd IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 527–534.
[44] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. 30th Int. Conf. Supercomput. (ICS)*, Jun. 2016, p. 28.
[45] W. Xia *et al.*, "A comprehensive study of the past, present, and future of data Deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.
[46] H. Xu, Y. Zhou, C. Gao, Y. Kang, and M. R. Lyu, "SpyAware: Investigating the privacy leakage signatures in APP execution traces," in *Proc. 26th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 348–358.
[47] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2017, pp. 15–28.
[48] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/O characteristics of smartphone applications and their implications for eMMC design," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Atlanta, GA, USA, Oct. 2015, pp. 12–21.
[49] K. Zhou, S. Hu, P. Huang, and Y. Zhao, "LX-SSD: Enhancing the lifespan of NAND flash-based memory via recycling invalid pages," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol. (MSST)*, May 2017, pp. 1–13.

Bo Mao (M'10) received the B.E. degree in computer science and technology from Northeastern University, Shenyang, China, in 2005 and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2010.

He is an Assistant Professor with the Software School of Xiamen University, Xiamen, China. He has over 40 publications in international journals and conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, ACM-TOS, USENIX FAST, USENIX LISA, ICS, ICCD, MSST, and IPDPS. His current research interests include storage system, cloud computing, and big data.

Dr. Mao is a member of ACM and USENIX.

Jindong Zhou received the B.E. degree in software engineering from Xiamen University, Xiamen, China, where he is currently pursuing the master's degree in software engineering.

His current research interests include flash storage systems and data deduplication.

Suzhen Wu (M'10) received the B.E. degree in computer science and technology and the Ph.D. degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2005 and 2010, respectively.

She has been an Associate Professor with Computer Science Department, Xiamen University, Xiamen, China, since 2014. Her current research interests include computer architecture and storage system. She has over 40 publications in journal and international conferences, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, ACM-TOS, USENIX FAST, USENIX LISA, ICS, ICCD, MSST, and IPDPS.

Dr. Wu is a member of ACM.

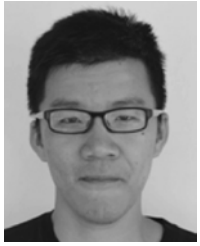
Hong Jiang (F'14) received the B.Sc. degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the M.A.Sc. degree in computer engineering from the University of Toronto, Toronto, ON, Canada, in 1987, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 1991.

He served as a Program Director with National Science Foundation from 2013 to 2015. He has been with the University of Nebraska–Lincoln, Lincoln, Nebraska, since 1991, where he was a Willa Cather Professor of computer science and engineering. He is currently the Chair and the Wendell H. Nedderman Endowed Professor with Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX, USA. He has graduated 16 Ph.D. students who upon their graduations either landed academic tenure-track positions in Ph.D.-granting U.S. institutions or were employed by major U.S. IT corporations. His research has been supported by NSF, DOD, the State of Texas and the State of Nebraska, and industry. His current research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He has over 300 publications in major journals and international conferences in the above areas, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, Proceedings of IEEE, ACM-TACO, ACM-TOS, JPDC, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, SOCC, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, and ICPP.

Dr. Jiang recently served as an Associate Editor for the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is a member of ACM.

Xiao Chen received the master's degree in computer science and technology from Xiamen University, Xiamen, China, in 2017.

Since 2017, he has been a Software Engineer with Wangsu Science & Technology, Xiamen. He has published two papers in MSST and ICPADS. His current research interests include flash storage systems and data deduplication.



Weijian Yang received the B.E. and master's degrees in computer science and technology from Xiamen University, Xiamen, China, in 2013 and 2017, respectively.

Since 2017, he has been a Software Engineer with Huawei Technology, Shenzhen, China. He has published two papers in MSST and ICA3PP. His current research interests include flash storage systems and data deduplication.