Differentiating Cache Files for Fine-grain Management to Improve Mobile Performance and Lifetime

Yu Liang†, Jinheng Li†, Xianzhang Chen,[*] Rachata Ausavarungnirun‡, Riwei Pan†, Tei-Wei Kuo†§, Chun Jason Xue†
† *Department of Computer Science, City University of Hong Kong*
∗ *College of Computer Science, Chongqing University*
‡ *TGGS, King Mongkut's University of Technology North Bangkok*
§ *Department of Computer Science and Information Engineering, National Taiwan University*

Abstract

Most mobile applications need to download data from the network. The Android system temporarily stores these data as cache files in the local flash storage to improve their re-access performance. For example, using Facebook for two hours in one case generated 1.2GB cache files. Writing all cache files to the flash storage has a negative impact on the overall I/O performance and deteriorates the lifetime of mobile flash storage. In this paper, we analyze the access characteristics of cache files of typical mobile applications. Our observations reveal that the access patterns of caches files are different from application-level to file-level. While existing solutions treat all cache files equally, this paper differentiates cache files into three categories, burn-after-reading, transient, and long-living. A Fine-grain Cache File Management (FCFM) framework is proposed to manage different cache files differently to improve the performance and lifetime of the mobile system. Evaluations using YouTube show that FCFM can significantly improve the performance and lifetime of mobile devices.

1 Introduction

Mobile devices have become increasingly important in our daily lives. In 2019, the number of worldwide smartphone users surpasses three billions and Android smartphones account for 80% of the sales [15]. Performance and lifetime of the mobile device are two key metrics of a mobile device [6–9, 16]. Currently, mobile systems store all the temporary data downloaded by mobile applications as cache files in the local flash storage. As a result, existing mobile applications generate heavy writes to the flash storage. For example, in one case, Facebook generated more than 1GB of data and 13000 writes in two hours. Such high number of writes not only damages the lifetime of the flash storage but also leads to many I/Os to the mobile system, potentially lowering performance. Hence, a fundamental question is that can we improve the existing cache file management for mobile devices?

The benefit of storing the cache files in the flash storage is based on an implicit assumption: the cached files are going to be accessed again often and for a long time. In this paper, we examine this assumption by tracing the cache file accesses of ten widely-used mobile applications, covering social media, map-related apps, games, video apps, and browsers. Then, we analyze the collected traces from the perspective of read/write amount, the number of files, and read operations. Our observations reveal that the assumption that the cached files are going to be accessed again does not hold on lots of cache files across multiple applications. In fact, the characteristics of cache file accesses vary greatly for each application and each individual file.

Based on our observations, this paper advocates that cache files should not be treated equally in mobile devices. This paper differentiates cache files and proposes a Fine-grain Cache File Management (FCFM) framework to improve the performance and lifetime of the mobile devices. FCFM adopts a filter to differentiates cache files and stores them in the main memory and the flash storage by an in-memory file system and a flash file system, respectively. The filter categorizes the cache files into three types, burn-after-reading (BAR), Transient, and Long-living, according to the access patterns on cache files. FCFM stores BAR files and Transient files in main memory using in-memory file system and discards them when the memory is running out of space. The Long-living files are stored in the flash storage to speedup the re-accesses.

We evaluate the proposed FCFM in Android using YouTube. FCFM adopts RAMFS [12] and F2FS [10] as the hybrid storage. The experimental results show that compared to the existing cache file management, FCFM can reduce the write amount and the number of writes by 93% and 65%, respectively. The contribution of this paper is listed as follows.

- We investigate the access patterns of cache files of mobile applications and find that the existing cache file management may have negative impact on the perfor-

[*] Corresponding author. Email: xzchen@cqu.edu.cn

mance and lifetime of the mobile devices;

- We propose a Fine-grain Cache File Management (FCFM) to optimize the performance and lifetime of mobile devices utilizing the observed access patterns of cache files;

- We verify the effectiveness of FCFM over existing solutions in the Android system.

2 Backgrounds

Data access performance is a key metric for mobile devices that directly affects user experience, such as displaying news feeds and watching videos. These data generally come from two sources: the network and the local storage. Different from servers (excluding cache servers [13]), most mobile applications download fresh data such as news and videos from the network. Because the bandwidths of networks are different based on different networks, e.g., the Verizon 4G LTE wireless broadband provides download speeds between 5 and 12 Mbps (0.652 and 1.5 MB/s) [17], most applications typically store the downloaded temporary data as cache files in the mobile device to minimize data transfers across the network and improve performance.

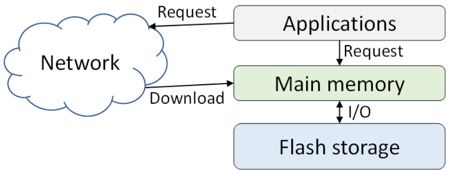


Figure 1: Framework of cache file read in Android system.

Existing Android systems first store the cache files in the main memory, and then write them back to the flash storage, as shown in Figure 1. When an application accesses a page of a cache file that is located in the main memory, the throughput can reach 25.6 GB/s [1]. If the requested page is not in the main memory, the read request is delivered to the flash storage. The data access bandwidths are hundreds of times lower than accessing the files in the main memory, such as 39.3 MB/s and 272.2 MB/s for random read and sequential read, respectively [1].

Another important metric for mobile devices is lifetime. With the increasing memory and hardware performance, the replacement cycle length of mobile devices has also increased [16]. The NAND flash storage widely adopted in mobile devices suffers from limited write/erase cycles. It is crucial to improve the lifetime of smartphones by reducing the write amount to the flash storage.

3 Cache File Analysis

Existing file systems apply the same strategy to manage the cache files with an implicit assumption that all the cache files are equal. However, this paper shows that the accesses to cache files vary greatly from application-level, file-level, and pattern-level.

Ten commonly-used apps, including social media, map, game, video, and browser, are used to characterize cache files on Android. The workloads of the apps are listed in Table 1. Similar workloads are also used in work [11].

Table 1: Workloads of applications.

Type	Applications	Workloads
Social media	Facebook	View news: (a) drag the screen to load news; (b) load the news for displaying; (c) repeat (a) and (b).
	Twitter	
Map	Map	Search address: (a) type key words; (b) load the news for displaying; (c) drag the screen, zoom in and zoom out the map; (d) repeat (a) (b) and (c).
	Earth	Scan satellite maps: (a) drag the screen, zoom in and zoom out the map; (b) repeat (a).
Game	CandyCrush	Load and play CandyCrush.
	Zombie	Load and play Plants vs. Zombies.
Video	Youtube	Play series: (a) type key words; (b) load the news for displaying; (c) autoplay long series.
	TikTok	Play vedios: (a) drag the screen to load news; (b) play short vedio; (c) repeat (a) and (b).
Browser	Chrome	Search news: (a) type key words; (b) load the news for displaying; (c) repeat (a) and (b).
	FireFox	

All the experiments are conducted on a Huawei P9 smartphone equipped with an ARM Cortex-A72 CPU, 32GB internal flash memory and 3GB RAM. The smartphone runs Android 7.0 with Linux kernel version 4.1.18. We instrument the Android kernel source code and use the Android Debug Bridge (**adb**) tool [4] to obtain read, write, lifetime, and file size information of cache files. From the conducted experiments, we uncover several unexpected observations.

We track the read and write information of cache files in the flash storage using the function *submit_bio()* in block/blkcore.c. Each application has its default cache folder in the path */data/packageName/cache*, which is private to the application itself. Some applications store cache files in */media/0/Android/data/packageName/cache*, which is accessible to other applications. We collect read and write statistics on both paths.

3.1 Cache Access Patterns of Applications

We summarize the read amount and the write amount of different applications in Table 2. “Read” represents the read amount on all the cache files in the flash storage when using

[1] This data is verified with the tool DevCheck [5].

each application for two hours. “W/R” represents the ratio of write amount over read amount.

Table 2: Read and write amount on cache files in flash storage.

Type	Applications	Read (MB)	Write (MB)	W/R
Social media	Facebook	8.9	1203.6	134.5
	Twitter	21.5	525.5	24.5
Map	Map	6.7	141.5	21.2
	Earth	210.4	931.5	4.4
Game	CandyCrush	8.6	3.0	0.3
	Zombies	30.8	92.7	3.0
Video	YouTube	1.3	800.3	638.2
	TikTok	8.0	1040.4	129.7
Browser	Chrome	14.8	276.8	18.7
	Firefox	10.8	289.6	26.7

As Table 2 shows, the read amount and the write amount of different applications are quite different. For example, social media applications write hundreds to thousands of cache file data, whereas the game applications only access less than 150MB cache file data in total. At the same time, the applications also show different Write/Read ratios. For example, the W/R ratio of YouTube is 34.1× that of Chrome. In summary, the following observations are drawn:

Observation 1. The cache file accesses of different types of applications vary greatly in terms of the total data amount and the Write/Read (W/R) ratio.

The discrepancy between different types of applications is determined by the intrinsic logic of the applications. For example, most data of a game is fixed data rather than temporary data. On the contrary, most data of social media and video applications are fresh information obtained from the network, which are subsequently treated as cache files to accelerate the possible future accesses.

Observation 1 indicates that if we find out the access pattern of cache files following the intrinsic logic of applications, mobile devices can improve system performance and lifetime by managing the cache files judiciously.

Observation 2. On average, the write amount of cache files is 100X more than the read amount.

Observation 2 reveals that for most applications, most of their cache files are rarely re-used. In this case, there exists huge potential to improve system performance and lifetime by discarding rarely accessed cache files. To construct better management for the cache files, we further exploit the characteristics of cache files at file level.

3.2 Cache-File Characteristics

We analyze the characteristics of cache files for each application from three perspectives: the amount of file accesses, the lifetime of cache files, and the file size. The CDF of read and write amount ratio of the cache files for each application is shown in Figure 2a and Figure 2b, respectively. We order the cache files according to their read/write amount.

Observation 3: Most of the reads of the applications are concentrated on a few cache files.

According to observation 3, it is beneficial to reduce write amount on the flash storage by discarding most of the cache files that are rarely accessed again. However, there may be rarely-accessed large files which are accessed again. Suppose the system simply discards such files, it may degrade the system performance since the mobile device has to search and download these files from the network to satisfy the data read requests of the application. Thus, we further exploit the connection between read amount, lifetime of files, and file sizes of cache files.

The lifetime of a file is calculated by subtracting its creation time from the time it is deleted. The collected data of each application is shown in Figure 2. The histogram represents the number of files with the same lifetime. The numbers above the histograms are the read amount of these cache files. The line chart represents the total size of the files with the same lifetime. To investigate the lifetime of cache files, applications are used daily for several days before collecting data. Since we only run experiments for two hours, all the long-lifetime (more than 2 hours) cache files were generated in the last few days while other files were generated during experiments.

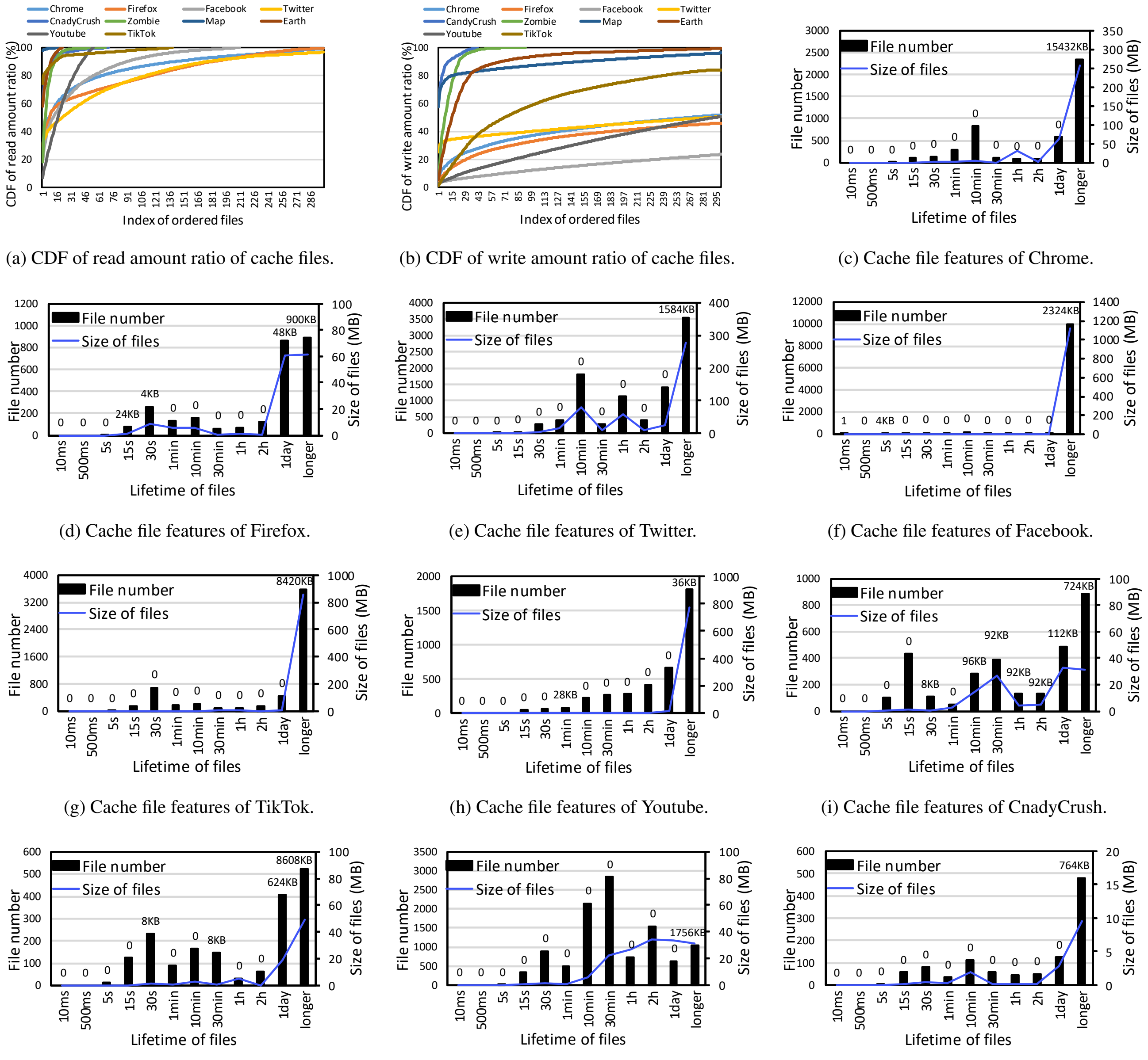
As shown in Figure 2 (b)-(k), except for CandyCrush and Google Map, applications store more than 50% of their cache files at least one day. Meanwhile, the blue lines indicate that these long-lifetime files also account for more than 50% of the total size of the cache files in these applications. Furthermore, except for YouTube and CandyCrush, more than 90% of the read amounts of applications focus on the cache files stored at least one day. In summary, we have the following observation:

Observation 4: For most applications, most of the reads are conducted on the long-lifetime cache files. Moreover, the total size of these files is more than 50% of the total size of all the application’s cache files.

According to observation 4, suppose the system discards the short-lifetime cache files and stores the long-lifetime cache files in the flash storage, it is possible to halve the writes on flash storage with small performance cost. If the long-lifetime cache files stay in memory when it is read, its access performance can be further improved. Since choosing files to put in memory needs to consider their read patterns, we exploit the read patterns of cache files.

3.3 Read Patterns of Cache Files

To show the read pattern of read-intensive files, we order the cache files according to their read amount. Figure 3 shows the read patterns of the ten most-read cache files of each application, where the selected cache files in each application are displayed by ten colors. Since many read operations upon a single file could happen in a short period of time, we merge the number of reads of a file in every 80 seconds. Thus, larger circles represent more reads in this period. Each frame repre-



(a) CDF of read amount ratio of cache files.

(b) CDF of write amount ratio of cache files.

(c) Cache file features of Chrome.

(d) Cache file features of Firefox.

(e) Cache file features of Twitter.

(f) Cache file features of Facebook.

(g) Cache file features of TikTok.

(h) Cache file features of Youtube.

(i) Cache file features of CnadyCrush.

(j) Cache file features of Plant vs Zombie.

(k) Cache file features of Google Map.

(l) Cache file features of Google Earth.

Figure 2: Characterizing cache files in four dimensions: read, write, lifetime, and file size.

sents an application. Based on Figure 3, we have the following observation:

Observation 5: From the temporal perspective, the cache files basically show two types of read patterns: the files with concentrated reads in a short time period and the files are contiguously read across the application execution time.

For the cache files have highly concentrated reads, such as the file 0 in Zombie, suppose the system keeps the data of such a cache file in the memory, the system performance can be improved by eliminating many I/O operations. For the cache files that have scattered reads across the app execution time, such as file 0 in Twitter and Google Map, it is not necessary to store all of these files in the memory since there are gaps of hundreds of seconds between the accesses.

In summary, we show that not all cache files are equal at application level, file level, and pattern level. However, the existing Android system is unaware of the uneven character of cache files and stores all the cache files in the local storage, which not only degrades the system performance but also damages the lifetime of the mobile devices. We believe that it is necessary to store cache files in both memory and flash storage to improve the system performance and lifetime. Fur-

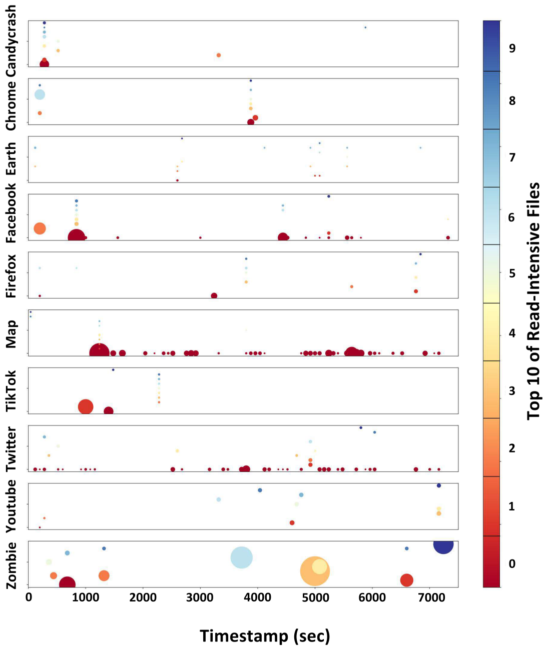


Figure 3: Read pattern of cache files for each application.

thermore, we should revisit the management of cache files to fully exploit the per-application and per-file access patterns.

4 Fine-grain Cache File Management

Based on the observations, we present a Fine-grain Cache File Management (FCFM) framework to improve the performance and lifetime of mobile devices by exploiting the characteristics of cache files. Figure 4 shows the proposed FCFM.

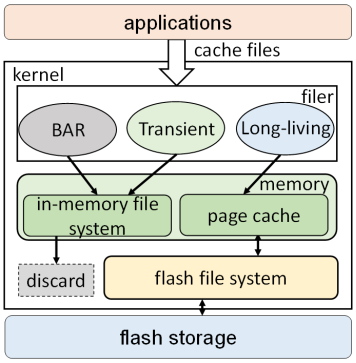


Figure 4: Framework of FCFM.

Different from existing solutions, FCFM maintains cache files in main memory and flash storage hybrid architecture. Aside from the flash file system for the flash storage, FCFM employs an in-memory file system, such as RAMFS [12] and TMPFS [14] to manage the cache files in main memory. The cache files belonging to the flash file system will be written back to the flash storage, whereas the cache files in the in-memory file system will be discarded. Thus, it is important to determine the placement of cache files in these two file systems.

Then, we propose a filter to determine the placement of cache files by exploiting their characteristics. The proposed filter divides cache files into three categories:

Burn-After-Reading (BAR): The cache files that have a large write amount but a tiny read amount. When an application closes a BAR cache file, the system will directly discard the file, i.e., delete it from the in-memory file system. If the application tries to access the file again, the application should download it from the network. Discarding the BAR files is expected to have inconspicuous damage to the system performance since the read amounts of BAR files are small.

Transient: The cache files that have a large write amount and a large read amount as well as a short active period. FCFM also stores the transient cache files in the main memory by an in-memory file system. In the case of memory running out of space, the system will first discard the BAR files, then delete the transient files using an LRU algorithm. Discarding the transient cache files using LRU is expected to have negligible damage to the system performance since they are rarely accessed after the active period.

Long-living: The rest of the cache files, especially those have large read amount and long active period. Storing the long-living files in the flash storage can avoid the high cost for fetching them from the network again.

According to observation 2 and observation 3 in Section 3, when the system employs the filter to place the cache files, it is expected to significantly improve the lifetime of the mobile device since only the long-living cache files are written back to the flash storage. Meanwhile, the system performance will be improved since the BAR files and Transient files are all directly accessed in the main memory.

5 Evaluation and Analysis

In this section, we conduct a set of experiments on YouTube to verify the effectiveness and feasibility of FCFM. In the experiments, we use RAMFS [12] and F2FS [10] as the in-memory file system and the flash file system, respectively.

We collect and analyze the data access information of YouTube on the cache files. Video cache files account for 93% of the total cache files of YouTube. Hence, we choose these cache files as target files of the experiments. We find that most of the video cache files, about 98%, are never read again once they are written back to the flash storage. According to the definition of three categories, the filter regards the video cache files as BAR files. We compare the FCFM with two baselines: 1) Fully-in-memory: which stores all the target cache files in RAMFS [2, 3]; 2) Fully-in-flash: the existing management of Android that stores all the target cache files in the flash storage. To implement the case of fully-in-memory, we mount */cache/exo* path to RAMFS. The evaluation results are shown in Table 3.

Table 3: Comparing FCFM with two baselines for video cache files of YouTube.

Categories	Write amount	Number of writes	cache/exo read
Fully-in-memory	71MB	5757	60KB from memory
Fully-in-flash	345MB	5000	60KB from storage
FCFM	24MB	1736	60KB from network

The results show that the total write amount of cache files into flash storage in FCFM is 66% and 93% less than that of the fully-in-memory and fully-in-storage mechanism, respectively. The total write amount in fully-in-memory mechanism is 79% smaller than that in fully-in-storage mechanism because the targeted files are not written back into flash storage but are stored in memory. However, taking up memory space makes the other applications share a smaller memory space, and thus the number of writes of the flash storage in fully-in-memory mechanism is increased by 15% due to additional evict operations. The number of writes of the flash storage in FCFM is 65% and 70% less than that of the fully-in-memory and fully-in-storage mechanism, respectively because its target cache files are discarded. In a word, FCFM can largely reduce the number and amount of writes in flash storage.

The writes of cache files into flash storage could compete with other user I/Os and thus degrade the I/O performance of mobile devices. Moreover, when the written-back data is deleted, Garbage Collection (GC) will be triggered to reclaim the free space and thus the lifetime of mobile devices will be reduced. Thus, reduce the number of writes and write amount of cache files is very important to mobile devices.

Notably, there are 2% of video cache data will be reused and need to be downloaded from network again because they are discarded. Moreover, an incorrect classification could cause additional overhead (including latency, energy, and money) to use additional network bandwidth to bring uncached data back. Thus, the penalty of FCFM depends on the accuracy of classification. If the classification is accurate, the penalty is very small (re-downloading 2% cache data).

6 Conclusion

To improve the re-access performance of downloaded data, the current Android system temporarily stores cache files for applications. This paper investigated the access patterns of the cache files by tracing and analyzing the file accesses of representative mobile applications. Based on the analysis, we observe that the mobile system should treat cache files differently as each cache files could exhibit very different access patterns. Thus, we proposed a Fine-grain Cache File Management (FCFM) framework to properly place the cache files in the in-memory file system or the flash file system according to their access patterns. The evaluation results showed that FCFM can significantly improve the performance and lifetime of mobile devices.

7 Discussion Topics

The proposed Fine-grain Cache File Management (FCFM) framework has three main challenges that need to be discussed.

Topic 1: How to systemically categorize cache files. In FCFM, all cache files should be categorized into three classes according to their characteristics (read, write, file size, and lifetime). When a cache file is downloaded, the system does not know its exact characteristics. One direction is to analyze each type of cache file based on its extension. For example, we find that .exo files produced by YouTube usually have a long lifetime (longer than one day) and will not be read again. This type of files can be categorized as BAR class. While the .db files produced by Map usually have a long lifetime (longer than one day) and will be read a lot. Moreover, the read operations are scattered over time. This type of files should be stored in the storage.

Topic 2: How much RAM should be used for the in-memory file system. There is a trade-off between the RAM size of the in-memory file system and the performance of the whole system. The in-memory file system with more RAM can store more cache files, which improves the cache file access performance. On the other hand, the in-memory file system competes for memory with the applications and the OS. A larger in-memory file system may lower the overall performance of the mobile device. Moreover, the amounts of cache files that need to be stored in memory vary across applications. For some applications, there is a maximum size for cache files. For example, YouTube configures 250MB maximum size for the exo folder. The system may set a maximum size for the in-memory file system of these applications. While for other applications, such as Facebook, the cache file could be increased without a preset limitation. Thus, some files will be discarded when the in-memory file system is full.

Topic 3: Cache file eviction scheme. An LRU based eviction scheme is used in current Android systems. However, a page-based evict scheme is not suitable for cache file eviction because when a page of a file is evicted and discarded, the whole file is invalid. A file-based eviction scheme will be more suitable for the file discard from in-memory file system. The file-base evict scheme needs to consider the read pattern, file size, and lifetime of the files.

Acknowledgment

This paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (11204718) and National Natural Science Foundation of China under Grant No. 61802038.

References

[1] ENGINEER, A. Huawei p9 read performance. https://www.mobigyaan.com/huawei-p9-review-is-it-the-ultimate-phone-camera, 2017.

[2] ENGINEER, A. How to mount a folder directly into the ram? https://wp-rocket.me/blog/mount-folder-ram/, 2019.

[3] ENGINEER, A. How to improve your computer performance and ssd life span with a ram disk. https://www.softperfect.com/articles/how-to-boost-computer-performance-with-ramdisk/, 2020.

[4] ENGINEERS. Android debug bridge (adb) tool. https://androidmtk.com/download-minimal-adb-and-fastboot-tool, 2019.

[5] FLAR2. Devcheck hardware and system info. https://play.google.com/store/apps/details?id=flar2.devcheckhl=zh, 2020.

[6] HAHN, S. S., LEE, S., YEE, I., RYU, D., AND KIM, J. Fasttrack: Foreground app-aware i/o management for improving user experience of android smartphones. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)* (2018), pp. 15–28.

[7] JEONG, D., LEE, Y., AND KIM, J.-S. Boosting quasi-asynchronous i/o for better responsiveness in mobile devices. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (2015), pp. 191–202.

[8] JEONG, S., LEE, K., LEE, S., SON, S., AND WON, Y. I/o stack optimization for smartphones. In *Proceedings of USENIX Annual Technical Conference (ATC)* (2013), pp. 309–320.

[9] KIM, S.-H., JEONG, J., AND KIM, J.-S. Application-aware swapping for mobile systems. *ACM Trans. Embed. Comput. Syst. 16*, 5s (Sept. 2017), 182:1–182:19.

[10] LEE, C., SIM, D., HWANG, J., AND CHO, S. F2fs: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (2015), pp. 273–286.

[11] LIANG, Y., PAN, R., YAJUAN, D., FU, C., SHI, L., KUO, T.-W., AND XUE, C. Read-ahead efficiency on mobile devices: Observation, characterization, and optimization. *IEEE Transactions on Computers PP* (04 2020), 1–1.

[12] MCKUSICK, M. K., KARELS, M. J., AND BOSTIC, K. A pageable memory based filesystem. In *USENIX Summer* (1990).

[13] SHEN, Z., CHEN, F., JIA, Y., AND SHAO, Z. Didacache: A deep integration of device and application for flash based key-value caching. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (Santa Clara, CA, Feb. 2017), USENIX Association, pp. 391–405.

[14] SNYDER, P. tmpfs: A virtual memory file system. In *In Proceedings of the Autumn 1990 European UNIX Users' Group Conference* (1990), pp. 241–248.

[15] STATISTA. Number of smartphone users worldwide from 2016 to 2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, 2020.

[16] STATISTA. Replacement cycle length of smartphones worldwide. https://www.statista.com/statistics/786876/replacement-cycle-length-of-smartphones-worldwide/, 2020.

[17] VERIZON. 4g lte speeds vs. your home network. https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/.