AN EFFECTIVE SCHEME TO MANAGE STORAGE IN SMARTPHONE DEVICE BY REMOVING DUPLICATE FILES

Ammar Asaad, Ali A.Yassin

Computer Department, University of Basrah, Education College for Pure Sciences, Iraq.

Abstract: During the last decade, mobile usage has increased dramatically. Due to the potential of mobile devices, technology is now widely used by many people. The Internet with the mobile, gives the mobile many additional features like file sharing (images, videos, audios, documents) by social media network apps, exchanging information among relatives and friends. This also means that mobiles receive numerous similar files via applications. As a result, there are many drawbacks, such as save duplicate files in the device's storage and low performance in (CPU, RAM, and battery). In this paper, we suggest and design an improved scheme to eliminate duplicate files. The duplicate files are eliminated using a Message Digest 5 (MD5) hash function, the American Standard Code for Information Interchange (ASCII), and Fast Fibonacci. These methods are used to generate a unique code that can remove the duplicate files in a fast and efficient way. As a result, we got a good result in storage and performance of the mobile device**.**

Key-words: Duplicate files; Mobile Device; Hash Code, MD5, Fast Fibonacci.

1. Introduction

Simple mobile phones can be utilised for simple computations, while smart devices are used for more complex computations [1]. Mobile devices play a key role in users' daily lives. Mobile devices are overtaking the place of laptops and PCs in all areas of life, including peoples' daily work, communications, and bank transactions work, communications, and bank transformation [2]. In view of social media networks and their high computational complexity, there are many challenges for users in terms of storage space, performance, and energy consumption [1, 2]. Smart device hardware has the ability to perform massive computing tasks, such as image processing, gaming, checking email, bank transfers, transaction authorisations, and other operations [1]. Many people cannot imagine their daily lives without a mobile phone device, or without the internet. Despite these properties, mobile devices still have drawbacks, such as being unable to handle complex tasks like face recognition and games with high-definition graphics[1, 3]. The amount of digitally stored data is growing at a high rate of approximately ten times per five years, overtaking the storage space available. With the increasing use of smartphones such as mobile phone devices and the improvement and development in this field, there is still a low storage problem with the mobile devices [4, 5]. The usage of these devices for a long time causes them to lose storage space, the performance of the CPU decreases, and energy is consumed [6]. One of the solutions to the storage problem is removing duplication files from the mobile phone devices [7].There are many fields based on deleting duplication files, such as paging, image matching, pattern recognition, and similarity of files. Sung-hun et al. [7] suggested a system implemented on a mobile device that is used like mobile apps to reduce the CPU overhead due to the memory deduplication based on the contents of each memory page. Miller et al. [8] proposed a strategy called Cross-layer I/O-based Hints to extend the memory deduplication scanners and then overcome the deduplication overhead rapidly. Furthermore, many authors focused on the main memory of mobile phones, which is one of the most important resources. Byeoksan et al. [9] developed a system known as MemScope that identifies which memory segment contains duplicate memory pages by examining the page table and the memory content. Nohhyun et al. [10] provided a framework to remove duplication files based on dataset contents.

In the field of cloud computing, Waraporn et al. [11] presented an effective data deduplication system for cloud storage to manage storage efficiency and to enhance the performance in the cloud storage environment. Their system computes hash code for each file in cloud storage by applying one-way. The hash functions are implemented in many fields, for example, file coding, authentication, and security. Benjamin et al. [12] described three techniques (summary vector, stream-informed segment layout, and locality preserved caching) to build a fingerprint for each file based on the SHA-1 hash function. They employed these techniques in the deduplication files to remove bottlenecks in the storage.

Currently, with the rapid development in cloud computing and mobile devices, there is a platform to support low latency network access and extra storage called mobile cloud computing (MCC) [13]. Moreover, MCC can provide mobile devices with extra storage based on pay-as-you-go within the cloud computing principle, but customers have to pay for this technique [14, 15]. The duplicate files remain in the storage of the cloud service provider.

Marques et al. [5] suggested a good scheme to delete duplicate files depending on their type. The method works well with known files and suffers to deal with new files that do not exist in the index file of their work [13]. Haustein et al.'s patent [16] involves selecting the deduplication method depending on the file type and computing the deduplication rate, and the deletion of duplicate files is done by the server side. Ryan et al. [13] presented a scheme to reduce energy consumption and the amount of data by detecting duplicate files. However, this scheme initially suffered from low duplicate detection performance and deduplication throughput for a few files. Although their work focused on using cloud storage to deduplicate files, the duplicate files are still in devices' storage. Several researchers focused on detection of duplicate data by using hash functions such as MD5 and SHA-256 [12, 17]. These schemes cannot face the collisions growing the length of the signature increase and the processing time increased with the size of signatures [18]. Ammar et al. [19] proposed a good scheme to delete duplicate images depending on MD5 hash function and Huffman code to generate unique code to remove duplicate image. In this paper, we present a scheme that has been

applied as mobile apps to remove the duplication files imported to mobile through social media applications such as Facebook, WhatsApp, Viber or mobile's. Our work focus to use hash function and Fibonacci code to build unique code for each image and overcomes the drawbacks of collisions grows as the size of the signature increases. However, the scheme generates a good search index file used to insert image or ignore in the repeating case. Our experimental results relied on two types of mobile devices "Samsung and Huawei. Moreover, our scheme has the safe cost of consumption energy of battery and less used for both RAM and CPU of the mobile device. Furthermore, our work can manage the storage of mobile and Cloud storage (based on some of Google's application, such as Google drive, Google photos) by removing duplicating files.

2. Materials and methods

We explain the algorithms and methods that are used in our work to achieve the scheme's goals.

2.1 Message Digest 5 Algorithm (MD5)

MD5 is one of the most commonly used algorithms to compress data files such as images, videos, and audios. The output of this algorithm is a fixed value (128 bit) when the algorithm is implemented once or many times on the same data. This algorithm was developed by Rivest and is considered as a simple way to compute the MD5 value. The MD5 processes 512-bit blocks and breaks them into 32-bit words. It includes 64 rounds. The MD5 steps are as follows. The first step is adding the bits to the block to equal the value of (448 mod 512), starting with 1, followed by 0's. The second step is adding the length of the message to the value (448 mod 512). The third step sets the MD5 register values (A, B, C, D), each one 32 bits (hexadecimal number). The fourth step is the heart of the algorithm; it has four pressure functions, and each of these pressure functions has logical operations [20-22]. Figuer1 illustrates the mechanism of MD5.

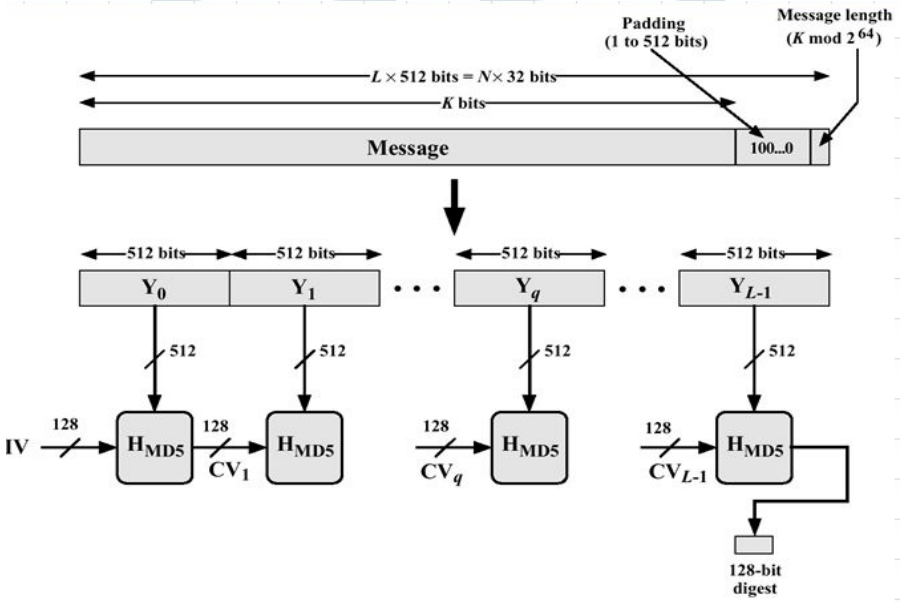

Figure 1: The mechanism and the main steps of implementing MD5 algorithm.

2.2 Fibonacci Code Generation

Data compression has been used in several fields, especially in data processing. There are various algorithms treated with several files, such as text, documents, images, and video. This method encodes an integer number to binary using Fibonacci representation. Zeckendorf's theorem is the base of this method. Fibonacci code is mostly used to compress small numbers because it is suitable for small numbers [23-25]. We will use this algorithm to generate a unique code for each file by calculating the fast Fibonacci code for each ASCII code. The main steps of this method are as follows:

- Input an integer number (***no***).
- Find the most significant Fibonacci number ***F*** (number in the Fibonacci series first chose); ***F*** > ***no***.
- Find ***f*** <= ***no*** (***f*** number in the Fibonacci series).

- If ***f*** <= ***no*** add 1 to the code string and make ***no*** = ***no*** - ***f*** otherwise add 0 to the code string. Then move ***f*** to a Fibonacci number lower than ***f***, and repeat the step (4) until (***no*** = ***0***) and append 1 to the end. The output binary numbers in Figure 2 explains the mechanism of fast Fibonacci.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	48	49	50	51	52	53	54	55	56	57
786	788	789	800	801	802	804	805	808	809	810	832	833	834	836	837	840	841	842	848	849	850	852	853	1536	1537	417	418	420	421	424	425	426	768	769	770

A ASCII is 65

Fibonacci	1	2	3	5	8	13	21	34	55	89
Fibonacci code	0	1	0	0	1	0	0	0	1	1
Binary base	1	2	4	8	16	32	64	128	256	512

= 786

B ASCII is 66

Fibonacci	1	2	3	5	8	13	21	34	55	89
Fibonacci code	0	0	1	0	1	0	0	0	1	1
Binary base	1	2	4	8	16	32	64	128	256	512

= 788

1 ASCII is 49

Fibonacci	1	2	3	5	8	13	21	34	55	89
Fibonacci code	0	1	0	0	0	1	0	1	1	-----
Binary base	1	2	4	8	16	32	64	128	256	512

= 418

5 ASCII is 53

Fibonacci	1	2	3	5	8	13	21	34	55	89
Fibonacci code	1	0	0	1	0	1	0	1	1	-----
Binary base	1	2	4	8	16	32	64	128	256	512

= 425

Figure 2: Shows each character and its code.

2.3 Hashing Search Method

Hashing search is a method used to generate a unique code for each item and save this item in the hash table to reduce search time. As an example, we assume that we have items and then want to compute a unique code for each item to reduce the search time. In this technique, the unique codes are converted into small numbers by using equations [26], so we used this method to reduce the time for searching for duplicate files. The two main steps of the hashing technique are as follows:

- Each item is transformed into an integer number by the hash equation.
- Each item is stored in the hash table based on the integer number generated previously [26]. Figure 3 shows the how this method works.

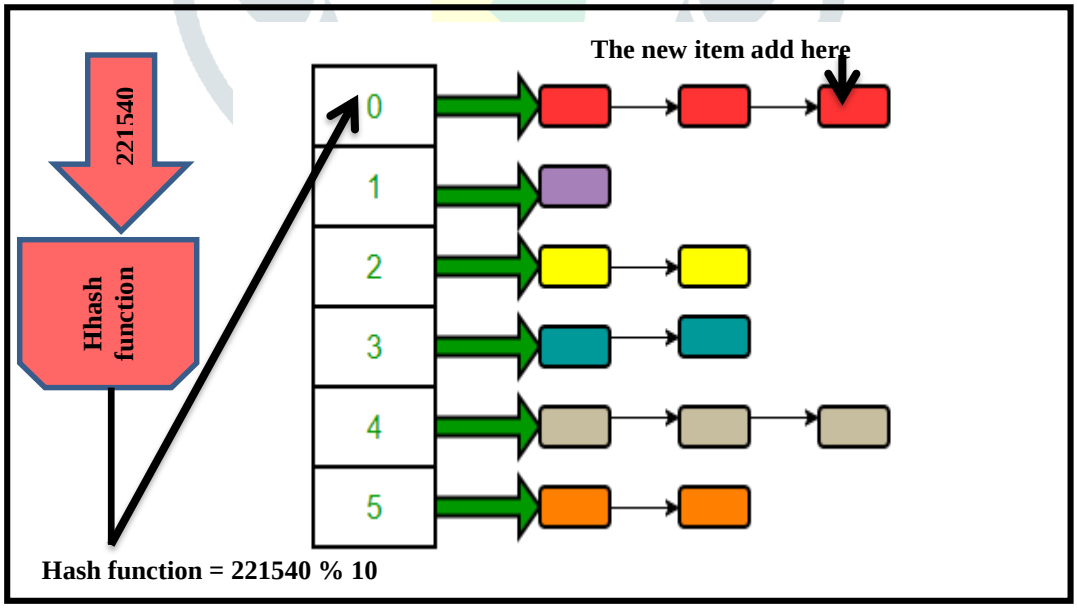

Figure 3: The mechanism of hash search method

3. Removing Duplicate Files Based on MFC Scheme

The MFC scheme proposed includes three phases: **building an index file, removing duplicate files, and daily phase.** The functions of each phase are as follows: the 'building an index file' phase works to create *IF* (See Table 1) for all files. The second phase is used to remove all duplicate files from the device. The final phase functions daily by continuously operating phases. All these phases play a crucial role in removing duplicate files received daily from social media apps. We propose a good method by using the MD5 hash function to improve performance. The main objective of our scheme is to delete duplicated files, such as

photos, and movies received from different social media apps. Additionally, we utilise the MD5 hash function and fast Fibonacci coding in order to build a unique code that is used to remove duplicate files.

3.1 Building Index File phase

This phase creates an index file to prepare everything for the next phases. The following steps explain the working of this phase:

- Generate two sets represented of all mobile's files $S = \{fil_1, fil_2, \dots fil_n\}$, moreover, their paths $P = \{p_1, p_2, \dots p_n\}$.
- Build a unique code for each file ($file_i \in S$) based on MD5 and Fast Fibonacci coding mentioned in the section 2.1 and 2.2. Figure demonstrates the process of generating a unique code.

$$H_i = H(fil_i) \dots (\mathbf{1})$$

$$\boldsymbol{Uni\_code}_{\mathrm{i}} = \sum_{\mathrm{j}=1}^{\mathrm{n}} \mathrm{febo}(\mathrm{ASCII}(H_i(j))) \dots (\mathbf{2})$$

The index file (IF) consists of the file path, unique code, size of file, feature vector for image file, and the received file date for each file. Algorithm 1 and figure 5 refers to the mechanism of creating IF.

Table 1: Notification of symbols

Symbol	Description
MHC	Coding based on MD5 and Huffman code.
FIL_i	The file that saves inside the mobile device.
S	Set of filesFIL_i.
p_i , P	p_i are meaning the $FILi's$ path while the P the set of $FIL's$ path.
IF	Index file has p_i of FIL_i, and it is details.
H	MD5 hash function, Hash code for FIL_i.
ASCII code	ASCII code function.
Uni_code_i	The unique code for FIL_i .
$Create\_First\_Record()$	Function to create first record in the index file.
$Search\_IF()$	Function to check FIL_i in the IF.
$Create\_New\_Record$ ()	Function for creating a new record in IF (Uni_code_i, p_i,Size(p_i)).
DEL_from_Device ()	Function to detect duplicate FIL_i.
$Get\ Newfiles()$	Function for get all new files.
$Get\_file()$	Function to get file.
MFC	Coding based on MD5, Fibonacci code.
$febo$ ()	Function to compute the Fibonacci code for ASCII code
fil_date	The date of creation file.
$Update\_IF()$	Function for update.

Algorithm 1 Creating Index File
Input: *n* where *n* is number of file path P saved in the device
Output: Index File (*IF*)
Compute H_1 based on Equation Eq (1)
Compute Uni_code_1 based on Equation Eq (2)
$Create\_First\_Record$ (p_1, Uni_code_1, Size(p_1), fil_date)
For $i = 2$ to n Do
Compute H_i as in Equation (Eq 1)
Compute Uni_code_i as in Equation (Eq 2)
IF ($Search\_IF$ (Uni_code_i , IF) ==True)
DEL_from_device (p_i)
$Update\_IF()$;
Else
$Create\_New\_Record$ (Uni_{code_i} , p_i, $Size(p_i)$)
End IF
End for
End Algorithm 1 Creating Index File

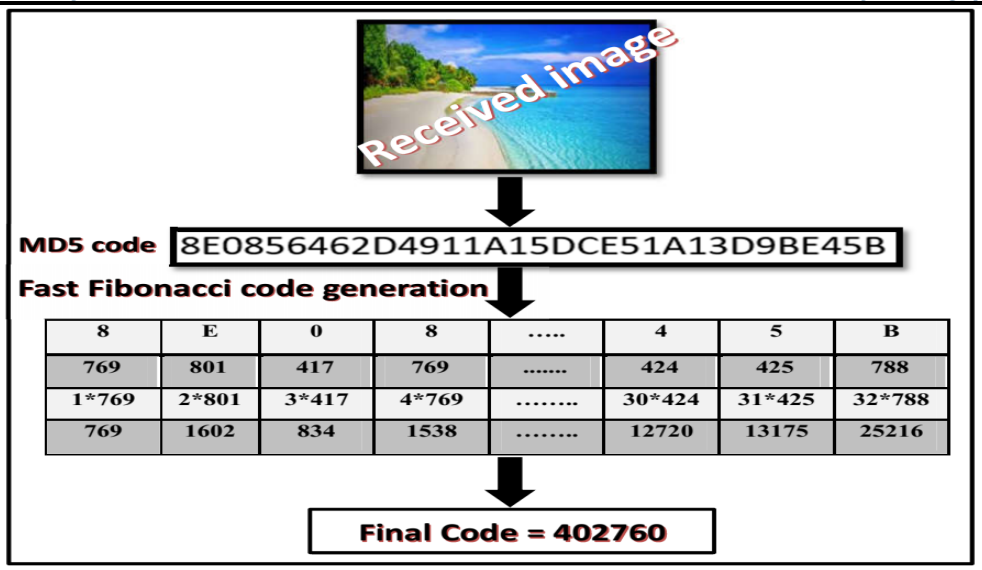

Figure 4: The process of generating a unique code based on MD5 and Fibonacci.

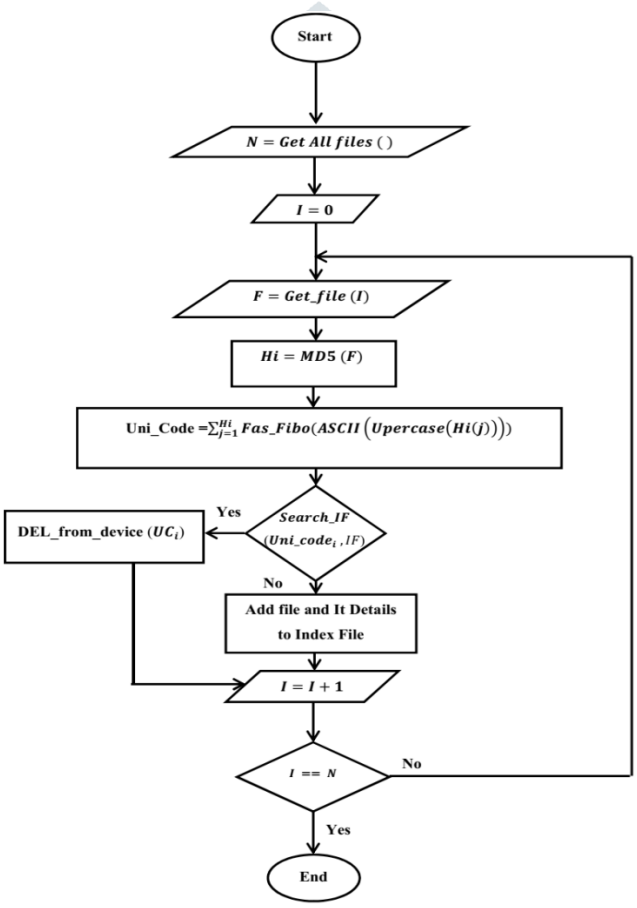

Figure 5: The flowchart for MFC scheme.

3.2 Remove Duplicate Files Phase

This phase works to remove all duplicate files based on their codes generated in the first phase. Additionally, this phase is applied automatically to remove all duplicated files received daily via social media networks. Algorithm 2 demonstrates the working of this phase.

Algorithm 2 Remove duplicate files
Input: n, P where n is the number of the new files;
Output: reduce storage space and update *IF*
For $i = 1$ to n Do
Compute H_i as in Equation Eq (1)
Compute Uni_code_i as in Equation Eq (2)
IF ($Search\_IF\ (Uni\_code_i\ , IF)$ ==True)
$DEL\_from\_device\ (p_i)$
$Update\_IF\ ()$;
Else
$Create\_New\_Record\ (Uni_{code_i}, p_i, Size(p_i))$
End IF

End For
End Algorithm 2 Remove duplicate files.

3.3 Daily Phase

This phase is considered the most critical phase because it prevents the accumulation of duplicate files in the mobile device's storage. This phase operates all previous phases daily. Figure 6 and Figure 7 show the added and removed images, respectively. The figure above explains the steps of adding/removing a received file. After computing a unique code for an image, we used search hash then checked whether it was in the index file.

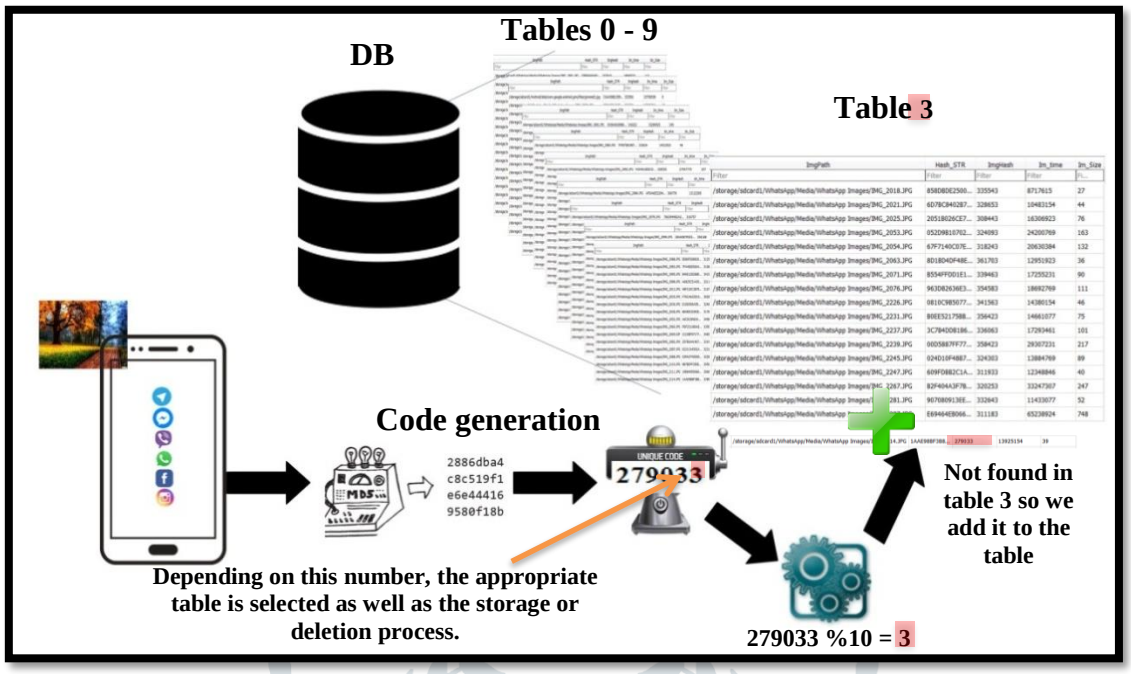

Figure 6: The main processes represented by the image, generating the code, checking IF and making the decision, to add a new file to IF.

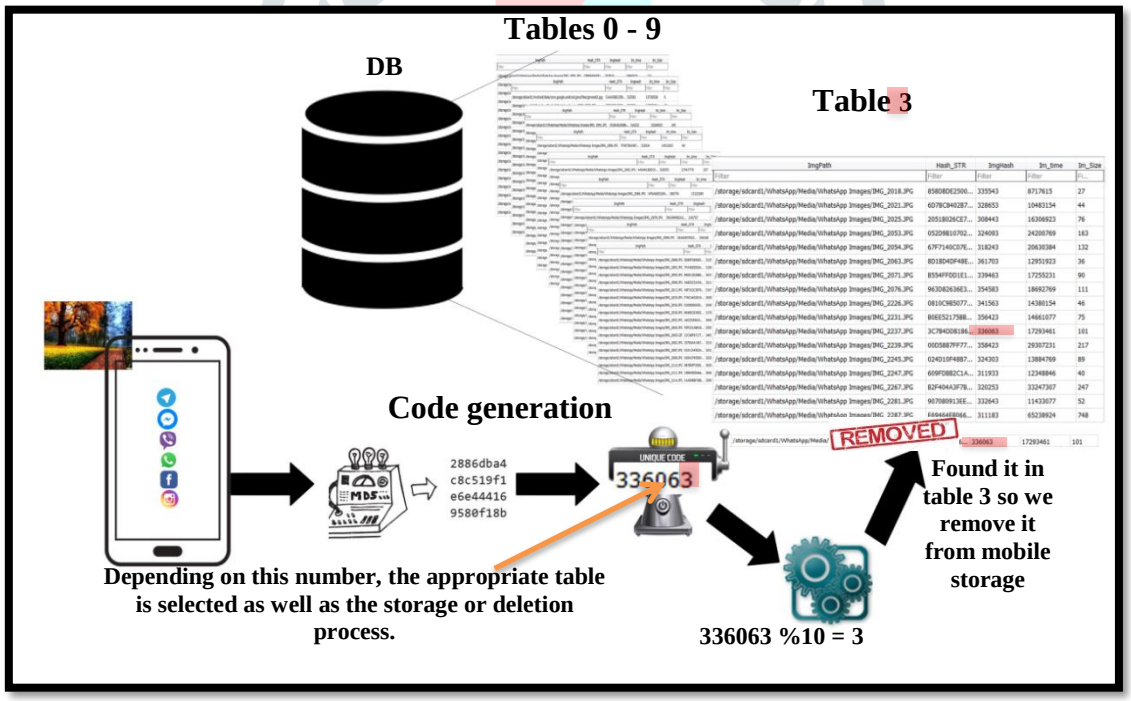

Figure 7: The main processes represented by the image from social media apps, generating the code, checking IF and making the decision to delete the image from mobile's storage.

4. The Experimental Results

In this part, the experiment results are outlined to explain the usefulness of our work. Several devices' files are utilised in this section to provide an in-depth analysis of the efficiency of our proposed scheme. Our work will implement the removal of the repeated files based on file matching.

Matches found between new files and files stored in the index file were based on standard matching. So, all redundant files were removed from storage. Throughout the paper, the proposed mobile application used Intel Core i5-2450M CPU @ 2.50GHz 250GHz, 8 GB RAM, a Windows 7 Enterprise operating system and Android Studio 3.3.1. Moreover, the specifications of the mobile app applied to the two mobile phones are as follows:

- HUAWEI CUN-U29, RAM 956.52 MB, internal storage 4.15 GB with an external storage 8 GB, battery 2200mAh, and Android version 5.1(Lollipop_MR1).
- Galaxy A5 (2017) SM-A520F, Android version 8.0(Oreo), battery 3000mAh, processor arm64-v8a 8 core, and total RAM 2815 MB.

4.1 Time of Index File Creation

For the purpose of experimentation and verification, experiments were conducted using 957 and 2816 original files in Huawei and Galaxy A5 devices respectively. Tables (2 and 3) show the performance of creation index files for two devices.

Table 2: The creation time for coding based on MD5 and Fibonacci code scheme in Huawei device.

Group	Samsung Galaxy A5 2017		
	file number	Time to generate file code	Size of files
0	284 files	7.425 s	290482 kb
1	289 files	10.612 s	415534 kb
2	255 files	9.651 s	373361 kb
3	295 files	8.301 s	325203 kb
4	294 files	8.002 s	308358 kb
5	284 files	9.601 s	383633 kb
6	285 files	9.15 s	360646 kb
7	258 files	4.88 s	182609 kb
8	282 files	7.831 s	297154 kb
9	290 files	24.422s	986751 kb
Total	2816 files	99.875 s	3923731 kb

Table 3: The creation time for coding based on MD5 and Fibonacci code scheme Samsung Galaxy A5 (2017) SM-A520F device.

Group	HUAWEI CUN-U29		
	file number	Time to generate file code	Size of files
0	93 files	7.539 s	129284 kb
1	100 files	9.114 s	161625 kb
2	82 files	5.414 s	89093 kb
3	89 files	5.632 s	90076 kb
4	97 files	8.281 s	140703 kb
5	99 files	5.246 s	83335 kb
6	108 files	5.419 s	87238 kb
7	79 files	4.06 s	62301 kb
8	99 files	7.232 s	118083 kb
9	111 files	8.04s	131194 kb
Total	957 files	65.977	1092932 kb

4.2 Daily Phase

The proposed scheme was implemented through mobile apps (Figure 8) over several days. This was done in order to apply the primary operations, such as detecting and removing redundant files alongside adding new files. **Error! Reference source not found.**4 describes the central operations including the addition of files to the index file over five days. Figures (9, 10, 11, and 12) explain the results of our scheme applied in search, storage and performance.

Table 4: The main operations for daily phase.

Day	Received	Add	Remove
1	29	23	6
2	20	19	1
3	29	17	12
4	31	20	11
5	25	9	16

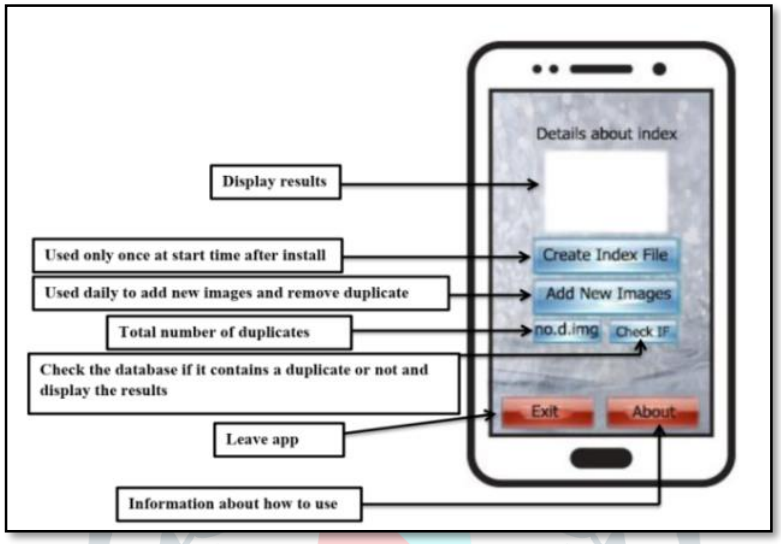

Figure 8: Removing Duplicate files Scheme.

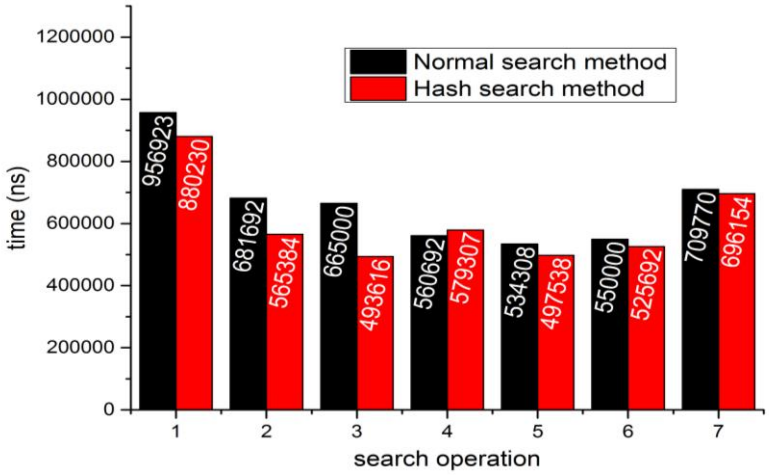

Figure 9: Compares between the proposed search and the ordinary search for duplicate files.

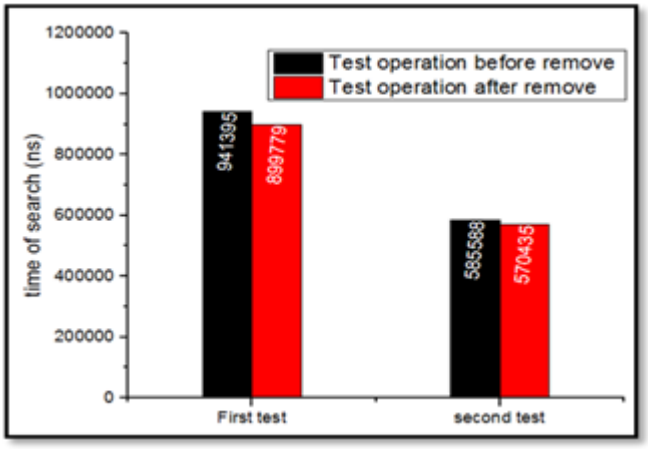

Figure 10: Shows the performance result by apply specific operations for two types of devices.

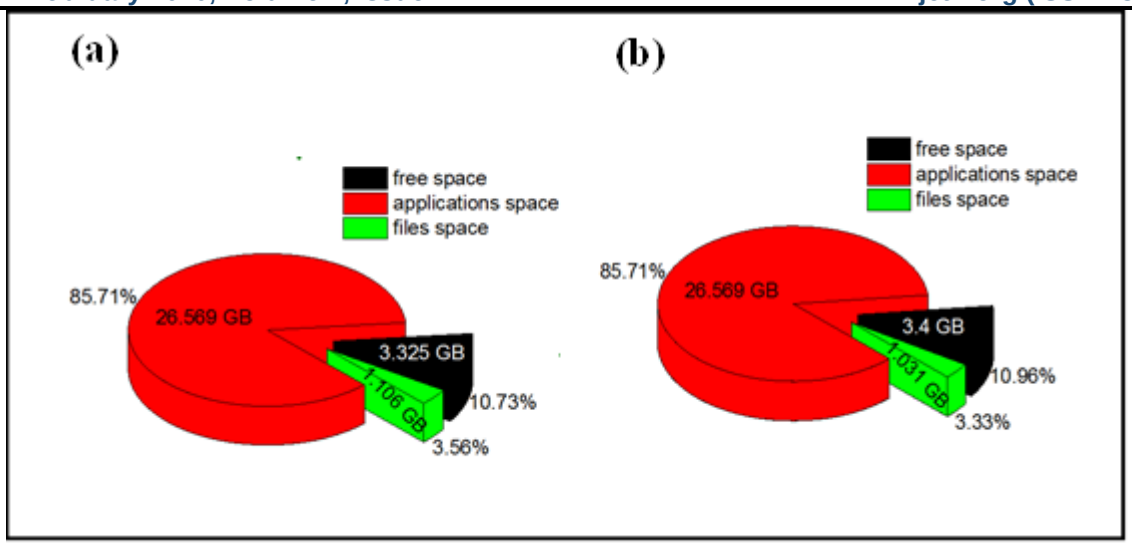

Figure 11: storage management results after implemented our scheme on HUAWEI (a) removing duplicate files

, (b) removing similarity images.

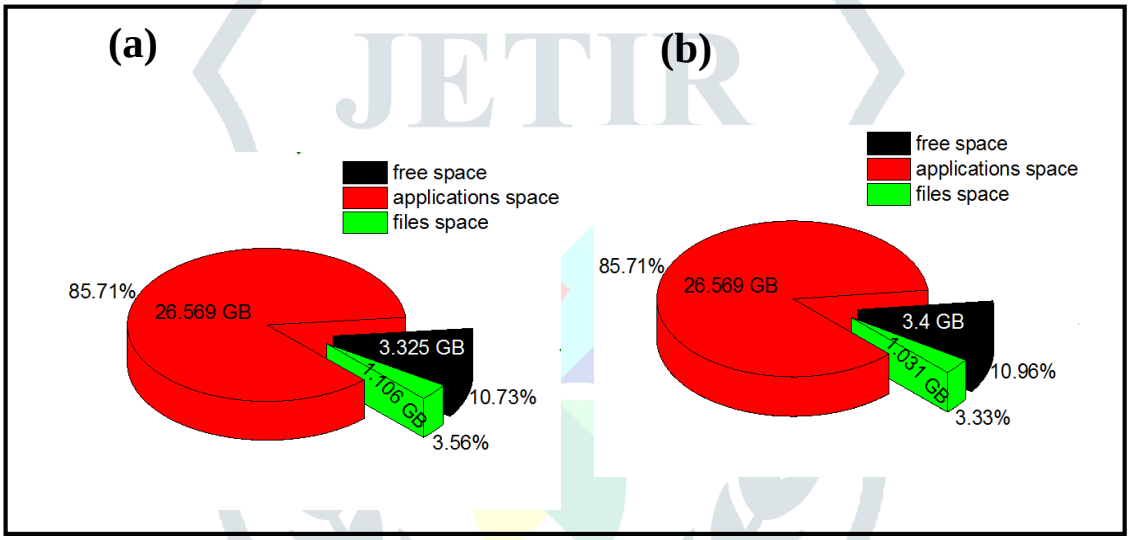

Figure 12: storage management results after implemented our scheme on Galaxy A5 (a) removing duplicate files, (b) removing similarity images.

5. Conclusions

One of the essential things in mobile devices is to keep the mobile works best and without any effect on the user. We proposed a new scheme for storage management by removing duplicate files and near-duplicate images using file coding. We have used a one-way hash function (MD5), and Fast Fibonacci code to generate unique-code. Our proposed scheme was able to preserve mobile storage and energy and enhance the processing of mobile devices. The proposed mobile applications were created and installed on two android mobile phone devices. Our scheme executed on two devices (Galaxy A5 (2017) SM-A520F and HUAWEI CUN-U29) using real-world data and get a better result in removing these duplicate files in fast and efficient. For future studies, we will use more than one factor, such as the enhancement performance of processor, keeping the battery capacity, and increasing size of memory space to make the mobile device work better. Another idea for faster processing might be to use cloud computing to overcome the limitations in the mobile device storage.

6. References

[1] R. A. Aldmour, "Mobile cloud computing for reducing power consumption and minimising latency," Anglia Ruskin University, 2018.

[2] H. Atre, K. Razdan, and R. K. Sagar, "A review of mobile cloud computing," in *2016 6th International Conference-Cloud System and Big Data Engineering (Confluence)*, 2016, pp. 199-202.

[3] N. Lee, C. Kim, W. Choi, M. Pyeon, and Y. Kim, "Development of indoor localization system using a mobile data acquisition platform and BoW image matching," *KSCE Journal of Civil Engineering*, vol. 21, pp. 418-430, 2017.

[4] J. F. Gantz, "The Diverse and Exploding Digital Universe-An Updated Forecast of Worldwide Information Growth Through 2011," *An IDC White Paper Sponsored by EMC*, 2008 2008.

[5] L. Marques and C. J. Costa, "Secure deduplication on mobile devices," in *Proceedings of the 2011 workshop on open source and design of communication*, 2011, pp. 19-26.

[6] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy cost models of smartphones for task offloading to the cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, pp. 384-398, 2015.

[7] S.-h. Kim, J. Jeong, and J. Lee, "Selective memory deduplication for cost efficiency in mobile smart devices," *IEEE Transactions on Consumer Electronics*, vol. 60, pp. 276-284, 2014.

[8] K. Miller, F. Franz, M. Rittinghaus, M. Hillenbrand, and F. Bellosa, "{XLH}: More Effective Memory Deduplication Scanners Through Cross-layer Hints," in *Presented as part of the 2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*, 2013, pp. 279-290.

[9] B. Lee, S. M. Kim, E. Park, and D. Han, "MemScope: analyzing memory duplication on android systems," in *Proceedings of the 6th Asia-Pacific Workshop on Systems*, 2015, p. 19.

[10] N. Park and D. J. Lilja, "Characterizing datasets for data deduplication in backup applications," in *IEEE International Symposium on Workload Characterization (IISWC'10)*, 2010, pp. 1-10.

[11] W. Leesakul, P. Townend, and J. Xu, "Dynamic data deduplication in cloud storage," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, 2014, pp. 320-325.

[12] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," in *Fast*, 2008, pp. 1-14.

[13] R. N. Widodo, H. Lim, and M. Atiquzzaman, "SDM: Smart deduplication for mobile cloud storage," *Future Generation Computer Systems*, vol. 70, pp. 64-73, 2017.

[14] T. Liu, F. Chen, Y. Ma, and Y. Xie, "An energy-efficient task scheduling for mobile devices based on cloud assistant," *Future Generation Computer Systems*, vol. 61, pp. 1-12, 2016.

[15] E. Ahmed, A. Gani, M. Sookhak, S. H. Ab Hamid, and F. Xia, "Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges," *Journal of Network and Computer Applications*, vol. 52, pp. 52-68, 2015.

[16] N. Haustein, C. A. Klein, U. Troppens, and D. J. Winarski, "Method of and system for adaptive selection of a deduplication chunking technique," ed: Google Patents, 2009.

[17] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009, p. 8.

[18] Y. Won, R. Kim, J. Ban, J. Hur, S. Oh, and J. Lee, "Prun: eliminating information redundancy for large scale data backup system," in *2008 International Conference on Computational Sciences and Its Applications*, 2008, pp. 139-144.

[19] A. Asaad and A. A. Y. Alamri, "A New Scheme for Removing Duplicate Files from Smart Mobile Devices," *Cihan University-Erbil Scientific Journal*, vol. 3, pp. 5-13, 2019.