Database Management Strategy and Recovery Methods of Android

Qian Li, Xueli Hu, Hao Wu
Department of Computer Science
Information Engineering University
Zhengzhou, 450003, China
liqian0349@gmail.com

Abstract—**Database analysis and the recovery of deleted record are two of the most important parts in digital forensics. This paper focuses on the management mechanism of Android SMS database, involving the analyses of write-in rules of database and log files during the data updating process. Based on the research results, a recovery method for database operating records from WAL file is proposed. To solve the problem that the log file is emptied, this paper presents a recovery method for the deleted log file for ext4 file system, thus constructing a database operating record timeline. Finally, an experiment based on images generated in different time and conditions is carried out to validate the effectiveness of the proposed method, discussion on its limitations is conducted.**

Keywords- SQLite, Android, WAL, database recovery, ext4

I. INTRODUCTION

Android is a set of software designed for mobile devices which includes an operation system, middleware and some key applications. Android is developed by Google Inc. and launched by a number of hardware and software manufacturers, and communications operators from around the world.

Most of the Android system apps, such as SMS, Contacts, Call History, E-mail Client, and a large number of third-party apps, have adopted the SQLite3 database for storage and management of data. SQLite database is a lightweight database which is widely used on a variety of embedded platforms due to its compactness, convenience and high speed [1].

With the rapid development of Android, forensics and data recovery technology on it have become increasingly important. Especially for digital forensics and judicial investigation, the deleted information tends to have a higher evidentiary value. Research on Android database management strategy and recovery methods will be of great value and have very broad application prospects.

II. RELATED WORK

There are about three kinds of methods on SQLite database recovery.

The first method is to recover through the transaction files. Haerder [2] and Pratik [3] proposed to recover database via database transaction logs. They also described various methods to recover the database, the advantages and disadvantages of various log systems, and what to do when database goes wrong.

The second method is based on the storage mechanism of YAFFS. Wu [4] studied the characteristics of NAND Flash memory and out-of-place-write strategy of YAFFS2 file system. They sort the log files according to the timestamp and generate a database event timeline. As Wu said, for ext4 file system have been used widely in android phone and Linux system, this method will not work on newer devices.

The third method is based on the structure of the database itself. We can find some patent documents about this which belong to Meiya Pico Co., Ltd. [5][6] The first step is to study the table structure, and then judge the deleted area by sliding windows to find out appropriate data units and recover the deleted records. This method works well when deleted areas are kept complete.

III. SQLITE AND WAL

Typically, SQLite save each database to a single file which consists of fixed-size pages which can be B-tree page, B+tree page, free page or overflow page. For database in auto-vacuum mode, there is another significant page called pointer bitmap page. One SQLite database file consists of multiple Btrees. Indexes are stored in B-trees and data is stored in B+tree. Each page is a node of Btree and the first page is called root page. All page numbers of table and index are stored in system table which is named sqlite_master and places its root page on the first page.

A. SQLite file header

First 100 bytes of the database is the file header that consists of version number, page size, encoding, parameters of Btree, auto-vacuum identifier and so on. Among them, the adoption of auto-vacuum mode makes the greatest impact on database recovery. Frequently inserting, updating, and deleting will make the database generate a lot of fragments. Free pages will be recycled to reduce the size of the file when vacuuming operation is performed. This gives the database recovery great difficulties.

B. Btree page

Data Unit is the basic element to organize data within Btree page. They are placed from bottom to top of the page. For size

978-1-4799-3279-5 /14/$31.00 ©2014 IEEE

being unfixed, the offset of its first byte also needs to be recorded. These offset values are placed from top to bottom of the page. This bi-growing design makes it easier to add new record without defragmentation.

A leaf node of B+tree page from Android SMS database is shown in Figure 1. There are 0x0C units and the first one is 0x01EB bytes away from page header. Read first byte of the unit at 0x531EB, and we will get 0x4C which is the payload length of this unit. The following two bytes stands for row id of the record and then is the payload content consisting of data format and data content that won't be described in this paper.

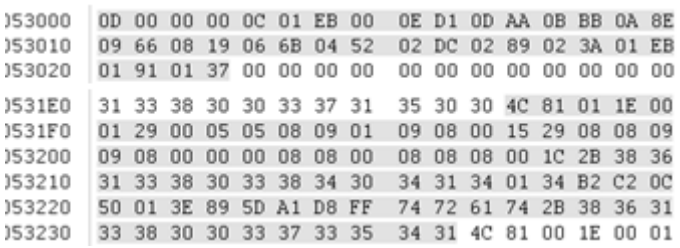

Figure 1 Btree page from Android SMS database

C. *Deletion and update in SQLite database*

Deletion and update in database tend to produce data fragments, free blocks or free pages. A typical example is shown in Figure 2. First, the unit starting from 0x5318B was deleted and produced a 0x60-byte length free space which is marked at the first four bytes of itself. And then a new record was inserted into the space so that the earlier deleted record was overwritten. At last the new inserted record was deleted too and a new 0x5A-byte length free space was produced. The former deleted record was overwritten nearly completely and can't be restored any more but the latter one only losts its first four bytes and can be restored according to a certain algorithm.

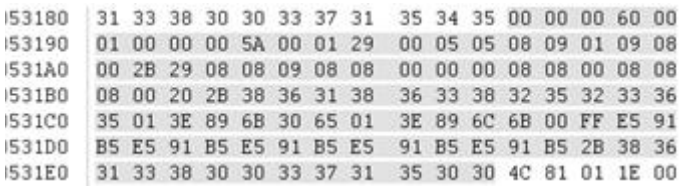

Figure 2 Data fragments from Android SMS database

D. *WAL*[7]

WAL (write-ahead log) is a kind of log for SQLite introduced from version 1.7. WAL consist of a file header and some frames which can be ranged from 0 to a lot. Each frame records one page of data from database. When checkpoint (is)triggered, data stored in WAL would be transferred into database. WAL can be reused with new frames overwriting old frames after checkpoint. WAL always grows from the beginning to the end of the sequence. The checksum and counter appended to each frame indicate that whether the frame is effective or not.

WAL has a 32-byte length header which includes eight 32-bit big-endian unsigned integers as shown in Figure 3. Among them, database page size is the same with the corresponding database. Salt-1 is a random number that is increased by 1 after checkpoint. Salt-2 is a random number too but will be replaced by another random number after checkpoint.

Frames follow after WAL file header. Each frame consists of 24-byte frame header and frame data as long as database page size. Frame header consists of six 32-bit big-endian unsigned integers as shown in Figure 4. Page Number indicates which page of the database is recorded here. Salt-1/2, being the same with or different from the corresponding value in file header, determines whether data in this frame is effective or not.

Magic Number	File Format Version	Database Page Size	Checkpoint Sequence
Salt-1	Salt-1	Checksum-1	Checksum-2

Figure 3 WAL file header

Page Number	Size after commit	Salt-1	Salt-1
Checksum-1	Checksum-2	Frame Data	

Figure 4 WAL frame header

IV. ANDROID DATABASE MANAGEMENT STRATEGY AND RECOVERY METHODS

A. *Android SMS database management strategy*

Experiment and test will be conducted on a Samsung Galaxy s3 phone. First, we clear its SMS database. Then do as Table 1 described and record the results.

Table I Operations

No.	Operations	DB Size	WAL size
1	Receive	4k,empty	Increases, adding new records
2	Send	4k,empty	Increases, adding new records
3	Delete	4k,empty	Increases, adding new records
…	Random	4k,empty	Increases, adding new records
n	Receive	228k,write first 1000	4024k, overwrite old records
…	Random	228k,unchanged	4024k, overwrite old records

At the beginning, size of database and WAL are all 0. With messages being received, sent or deleted, size of database keeps 4k and that of WAL increased with new records being added. When size of WAL reached 4024k (exactly 4120032 bytes), it stops increasing. At the same time, size of database increased to 228k with the first 1000 page data transferred into database. We can judge it is the checkpoint. After that, database kept unchanged till next checkpoint and size of WAL kept 4024k with new records overwriting old ones.

Figure 5 WAL file header

WAL file header for the target database is shown in Figure 5. The size of database page turns out to be 4096 bytes. We can calculate the total frame numbers at checkpoint by Equation 1.The result turns out to be 1000. So we can conclude that

when WAL has 1000 new frames the database reaches checkpoint and data begins to be transferred.

Equation 1: frameNums=(fileSize-fileHeader)/(pageSize+frameHeader)

B. *Recovery methods based on WAL*

Comparing the WAL before checkpoint with the one after checkpoint, we find that some of the frames are covered but others are not.

As is shown in Figure 6, the frame offsets at 0x4080 records the first page of the database before checkpoint. In Figure 7, we find it is replaced by the 0x39th page after checkpoint and the value of Salt-1 is increased by 1 to match the one in WAL file header. For data in the frame is overwritten completely, it can't be restored any more.

```
4080  00 00 00 01 00 00 00 00  44 03 BE 42 20 93 92 DA
4090  01 C5 C8 C8 B7 E7 6F 1B  53 51 4C 69 74 65 20 66
40A0  6F 72 6D 61 74 20 33 00  10 00 02 02 00 40 20 20
```

Figure 6 Frame from WAL before checkpoint

```
04080  00 00 00 39 00 00 00 39  44 03 BE 43 19 F5 64 DD
04090  6F 88 3D 87 AB 8A 2C 15  0D 00 00 00 05 0C 28 00
040A0  0F 3F 0E 60 0D AB 0C F6  0C 28 00 00 00 00 00 00
```

Figure 7 Frame from WAL after checkpoint

As is shown in Figure 8, both header and data of the frame offsets at 0x42650 remain unchanged. Data recorded in it is still the 0x0Dth page. Value of Salt-1 is less than the one in file header and the same with the one before checkpoint. It is concluded that this kind of frames can be stored.

```
42650  00 00 00 0D 00 00 00 00  44 03 BE 42 20 93 92 DA
42660  9A 00 67 19 2A B2 B9 53  0D 00 00 00 01 0F A6 00
43600  00 00 00 00 00 00 00 00  00 00 00 00 00 00 58 01
43610  1E 00 09 2D 00 05 05 08  09 01 09 08 00 2B 29 08
43620  08 08 08 08 00 00 00 08  08 00 08 08 08 00 31 32
43630  35 32 30 31 35 30 39 33  32 30 35 34 33 32 01 44
43640  AA B2 B3 6A 01 44 AA B2  A3 F0 FF E7 AC AC E4 B8
43650  80 E6 9D A1 E7 9F AD E4  BF A1 2B 38 36 31 33 38
43660  30 30 33 37 32 35 34 39  00 00 00 14 00 00 00 00
```

Figure 8 An uncovered frame from WAL after checkpoint

In conclusion, when database is updated, if checkpoint is not reached, records within a period of time can be restored. When checkpoint executed, the result is uncertain. If the value of Salt-1 in a frame is the same with that in WAL file header, the frame has be overwritten and data before checkpoint can no longer be restored. If they are different and the former is less than the latter, the frame hasn't been overwritten and data can be used to reconstruct history versions of the database. What we should do is to order the frames by time and insert them into a database timeline.

V. DATABASE RECOVERY FROM EXT4 FILE SYSTEM

Users may delete important messages stored in their phones to protect personal privacy. Some anti-forensics software may clear log files to prevent deleted information being restored. When WAL file is deleted, the associated app will create a new one during its next initialization. For deleted files still remain in storage, we can export the storage and find the historical WAL files to restore database records.

A. *Collection of Android data image*

App data of Android is typically stored in the directory "/data/data/app package name". In order to recover the deleted files, we need to collect the image of data partition through physical or logical method. We will adopt logical method in this pager for convenience.

Samsung Galaxy s3 uses MMC (Multimedia Card) as its storage. The MMC block located on "/dev/block/mmcblk0p12" is mounted as its data partition. Before reading device file, the program need elevate to root privileges. At the same time, to avoid influence on storage while data processing, we need to transfer the image to PC. We can use Adb (Android Debug Bridge) tool which can be found in Android SDK for this. Note that Adb will replace "0A0D" by "0A" to accommodate the display of the output shell, so the output image should be processed conversely. The whole process is shown as follows.

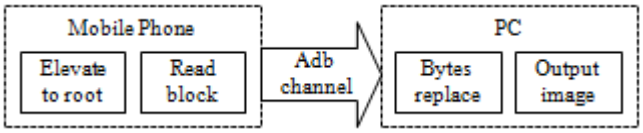

Figure 9 Collection of Android data image

B. *Finding deleted WALs from ext4 file system*

Ext4 is one of the most popular file systems on Linux. Ext4 consists of superblock, file system description, block bitmap, inode bitmap, inode table, data block and other elements. Superblock describes information of the file system including the amount of inodes and blocks, size of the storage and the format information. Inode represents attributes of file or directory and the block numbers belonging to it. Content of file or directory is stored in block or blocks if it is too long to be stored in one block. Inode bitmap and block bitmap are used to identify whether inodes and blocks are used or not.

Ext4 doesn't really modify the content of data blocks when file is deleted, but only delete or modify the relevant metadata and change the identifiers in corresponding inode bitmap and block bitmap to unused state to make the file can't be indexed correctly. When a directory is deleted, the space will be merged into the last one, but the inode number of the directory won't be emptied. So we can find names of deleted files by searching the matched inode in directory entries.

So we can restore deleted WAL files from the exported image and insert the records into database of event timeline ordered by time. Pseudo code for the method can be expressed as follows.

Read basic parameters from Superblock to get the size of inode and block and offset of the inode table. Then traverse the inode table to locate the existing and deleted log files belonging to target database, get attributes of them, extract file contents from data blocks and then insert the records into the database.

Table II Method to restore database records from data image

```
input: image of data partition named data.img
output: event timeline named EventTimeline.db
[block_size,inode_size,block_nums,inode_nums]=read_SBlock(img);
[offset_of_inode,offset_of_block]=read_img(img);
while(!end_of_inode_table==(inode=read_next_inode(img)))
 {
    if（read_filename(inode)=="targetdatabase-wal"）
    {
        create_time=read_time(inode);
        while(!end_of_block==(block=read_next_block(img,inode)))
        data[n]+=block;
        insert data[n] into EventTimeline.db ordered by create_time;
    }
}
```

VI. EXPERIMENT

An experiment will be conducted to test the method proposed above in this part. We will take Samsung Galaxy S3 phone as a sample and experimental procedure is described as follows.

1 Collect the image of data partition from the sample phone and name it img1. (The sample phone was initialized on April 2013 and this image was taken on December 28, 2013)

2 Use the sample phone normally. After a period of time, take a new image of data partition and name it img2. (Date: March 10, 2014)

3 Send some messages to the phone and delete some of them. Repeat till the checkpoint. Take a third image of data partition and name it img3. (Date: March 10, 2014)

4 Extract SMS databases and WAL files from the three images, note the database size, WAL file size and record page numbers down. Restore deleted WAL files from images and insert records into timeline.

Table III Records restored from three images

	size (byte)	*db size(byte)*	*WAL size (byte)*	*records in WAL*	*del WALs*	*total records*
1	123815	913408	4120032	1000	12	≈4100
2	85408	4096	634512	154	17	≈7254
3		233472	4120032	1000	17	≈8100

As is shown in Table 3, a complete WAL file usually carries 1000 records and an incomplete one carries less. We get 12 deleted WAL files from image1 and 17 from image2 or image3. Comparing group 1 with group 3, the 13 files in group 1 contain about 4100 records and the 18 files in group 3 contain about 8100 records. Experiencing shorter time, WAL files in group 3 are more complete than those in group 1. We can conclude that take image in time will avoid deleted files being overwritten more probably. The numbers of records an incomplete WAL file carries depends on the format completeness of its frames. At the same time, taking into account the complex structure of the database, one message being inserted or deleted will cause several data tables to be updated and so will the WAL. 1000 records may contain only about 100 times update operations or less.

In summary, the method proposed above is effective and an image being taken in time will make it work better. On the other hand, effect of this method is mostly limited to the recovery result of deleted files from ext4 file system.

VII. SUMMARY AND FUTURE WORK

We researched the management strategy of Android SMS database and found that data would be transferred from WAL to database after 1000 new records is added which should be the checkpoint of the database system. Comparison between frames from WAL files before and after checkpoint shows how old records are replaced and overwritten by new records. We proposed a method of restoring database records from deleted WAL files in ext4 file system and designed an experiment to verify the effectiveness of the method and discussed the limitations of it.

There are some deficiencies in this paper. Firstly, recovery algorithm of ext4 file system is not very comprehensive and efficient. Secondly, the data source in experiment is too simple and lacks objectivity. Collection of persuasive and exhaustive data will take more time and money. Thirdly, due to the lack of other SQLite database recovery tools, it is hard to be compared with other methods. Future work will focus on the deficiencies above.

REFERENCES

[1] Anonymous, "Features Of SQLite". Features page. [Online] . Available: SQLite homepage, http://www.sqlite.org/features.html. [Accessed: December 31, 2013]

[2] T. Haerder, A. Reuter, "Principles of transaction- oriented database recovery " ACM Computing Surveys (CSUR) 15.4 (1983), pp. 287-317.

[3] P. Patodi, "Database Recovery Mechanism For Android Devices", Diss. Indian Institute of Technology, Bombay, 2012.

[4] B. Wu, M. Xu, H. Zhang, "A recovery approach for SQLite history recorders from YAFFS2." Information and Communication Technology. Springer Berlin Heidelberg, 2013. 295-299.

[5] Meiya Pico Co., Ltd., "Method for restructuring deleted records of sqlite", China Patent 102,298,634, December 28, 2011.

[6] Meiya Pico Co., Ltd., "Method and system for mining deleted records of free space of sqlite", China Patent 102,591,979, July 18, 2012.

[7] Anonymous, " The SQLite Database File Format ". File Format introduction page. [Online] . Available: SQLite homepage, http://www.sqlite.org/fileformat2.html. [Accessed: December 31, 2013].