A Solution to Speed up the Loading of Pictures under Android Environment

Yahui Cheng[1][2][3]
CAD and Network Technology Research Institute, Dalian University of Technology
No.2 Linggong Road, Ganjingzi District,116000,Dalian City, Liaoning Province
E-mail: 842571627@qq.com

Yanping Hu
CAD and Network Technology Research Institute, Dalian University of Technology
No.2 Linggong Road, Ganjingzi District,116000,Dalian City, Liaoning Province
*E-mail:*hypok@dlut.edu.cn

Fei Xu
CAD and Network Technology Research Institute, Dalian University of Technology
No.2 Linggong Road, Ganjingzi District,116000,Dalian City, Liaoning Province
E-mail: 761866521@qq.com

Yan Zhang
CAD and Network Technology Research Institute, Dalian University of Technology
No.2 Linggong Road, Ganjingzi District,116000,Dalian City, Liaoning Province
E-mail: 907402251@qq.com

Zhe Sun
CAD and Network Technology Research Institute, Dalian University of Technology
No.2 Linggong Road, Ganjingzi District,116000,Dalian City, Liaoning Province
E-mail: 523921551@qq.com

A solution based on cache and prefetching was proposed to speed up the loading of pictures and avoid the out of memory(OOM). Cache, prefetching, monitoring and picture recycling technologies are applied to the solution. If some pictures are more likely to be accessed than others according to user's behavioral characteristic, they would be prefetched and put into cache to accelerate their loading speed and reduce the response time to the user's request. The solution was designed and implemented in the album management system and can also be used in most Android applications. In the end, experiment results show that the problems of picture thumbnails browse can be effectively solved by the solution.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker
[2]Correspongding Author
[3]Fund project: major national science and technology projects(2011ZX04015-021)

1. Introduction

With the rapid growth of Android mobile phone, there are a growing number of Android applications. Browsing photo thumbnails is common to most Android applications. In Android applications, especially in the album management system, slow loading of images and out of memory often occur in the process of browsing photographs because of such reasons shown as follows: (1) the required memory of each photo is too large; (2) the network connection is frequently used to get pictures;(3) the network connection requires a lot of traffic and time. In order to solve the problems of photo browsing, cache is usually used to reduce the frequency of network connection and the image compression is used to reduce the required memory by Android applications [1~3]; however, cache only stores the accessed pictures passively and reduces their loading time while prefetching can initiatively obtain the non-accessed pictures and reduce their loading time as well [4~5]. Thus, prefetching can make up for the shortcoming of cache.

Based on the above analysis, a solution based on prefetching and cache was put forward to solve the problem of thumbnails browsing. As the accessing pictures and upcoming accession pictures have been prefetched and put into cache, the way of obtaining image is changed from remote access to cache access. This way of cache access has effectively reduced the frequency of network connection and speeded up the loading of image. Taking the album management system as an example, the design and implementation of the solution is illustrated in this paper.

2. Performance of Problems in Thumbnails Browsing

There are some problems of thumbnails browsing. Taking the album management system as an example, the problems mainly occur in three scenes: the first scene when the users click on the icon of a photo album to view photo thumbnails; the second scene that the users browse thumbnails by page in an album; the third scene that the users browse thumbnails by sliding interface quickly. Thus, a solution based on cache and prefetching was put forward to solve the problems of thumbnails browsing.

3. Solution Based on Cache and Prefetching

3.1 The Overall Solution

Fig. 1 shows the composition of the solution, namely, image cache, image prefetching and the monitoring. The theory consists of the content as follows:

(1) The accessing photos are more likely to be accessed again than others based on the temporal locality [6], so they are put into cache to reduce their loading time;

(2) according to the space locality which indicates that the photos whose address space is close to the accessing photos are more likely to be accessed[6], the solution

prefetches these photos to reduce their loading time;

(3) According to the theory of “80-20”, which means that most access focuses on a few albums, the solution prefetches some photos from the albums with high access frequency to reduce their loading time.

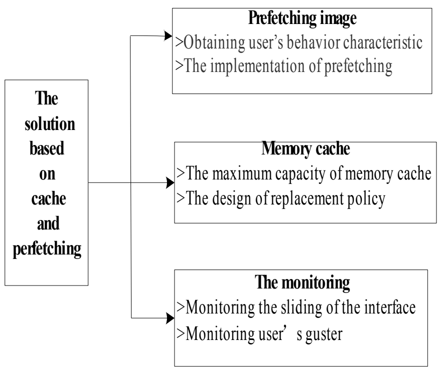
The solution is shown as follows:

(1)when the user is browsing the list of albums and is about to choose an album to view photos, the solution will prefetch some photos from albums with high visit frequency recorded in SQLite database;

(2)when the user is viewing some photos in an album by page, the solution will put these accessing photos into cache and prefetch some other photos according to the slipping direction of the interface and the space locality;

(3) When the user is browsing the thumbnails by fast sliding interface in an album, the solution will only load the photos on the stopped interface.

The solution needs to examine whether the cache is overflowed before a photo put into. If the cache is overflowed, some photos will be removed from cache according to the LRU, and then the photo will be put into cache. If not, the photo will be put into cache directly.


The solution based on cache and perfetching
Prefetching image
>Obtaining user's behavior characteristic
>The implementation of prefetching
Memory cache
>The maximum capacity of memory cache
>The design of replacement policy
The monitoring
>Monitoring the sliding of the interface
>Monitoring user' s guster

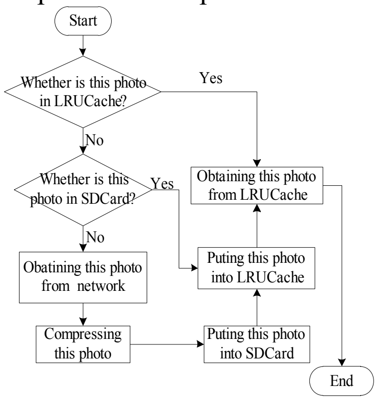

Start
Whether is this photo in LRUCache?
Yes
No
Whether is this photo in SDCard?
Yes
No
Obtaining this photo from LRUCache
Obatining this photo from network
Puting this photo into LRUCache
Compressing this photo
Puting this photo into SDCard
End

Figure 1: Composition of the Solution **Figure 2:**. Process of Obtaining Image based on Cache

3.2 Image Cache

The image cache is used to reduce network traffic and loading time of the photos. The remote accesses need to consume network traffic and time. Each photo to be accessed needs to connect to the server whenever the user accesses them, and the connection will delay the response time of the application. The image cache is used to solve this problem. If the photo to be accessed is just in the cache, the solution will obtain it from the cache directly. There are two types of cache methods supported by the Android system: the memory cache and the SD Card cache. Considering the speed of memory access and the permanently stored of objects in SD Card, the memory cache and the SD Card cache are comprehensively adopted.

3.2.1 Design of the Memory Cache

Every Android application has a strict memory limit. When the memory usage exceeds certain limit, OOM will occur; thus the memory cache must set a maximum capacity. When the memory cache is overflowed, it is necessary to remove some photos based on a certain replacement policy; therefore, the memory cache needs to solve two problems: the maximum capacity of memory cache and the replacement policy.

(1) In terms of the maximum capacity of memory cache. Because each Android device has its own memory limit for apps, the solution sets 1/8 of memory limit as the maximum capacity of memory cache.

(2) In terms of the design of replacement policy. When the memory cache capacity reaches the max size, the replacement strategy will remove some photos which are less likely to be accessed again than others according to some basis. At present, there have been three replacement strategies, namely, LRU, LFU and the size of the object. Because of the temporal locality, LRU is used to remove photos. Android system has LRUCache class which will remove some objects from it according to LRU When the capacity reaches it's setting value since the launch of Android 3.0. LRUCache is used to manage the cached photos.

3.2.2 Implementation of LRUCache

The solution uses the LRUCache class to manage the cached photos, which is as follows:

(1) An LRUCache object with specified space is initialized to manage the cached photos;

(2) The sizeof () function is rewritten to return the required memory of a picture;

(3) The put (K key, V value) is called to put a photo into LRUCache. In order to store more photos in the limited memory space, the photo is compressed before its put into LRUCache.

(4) The get (K key) function is called to obtain bitmap from LRUCache.

The process of obtaining image based on cache is shown in Fig. 2.

3.3 Prefetching Image

As the cache only reduces the loading time of the accessed photos, prefetching is used to reduce the loading time of non-accessed photos. When the user is viewing some photos, the solution will predict the next request data and prefetch them according to user's current and historical request; therefore, the loading time of prefetched photos is shortened. As the album management system is used to browse private information, prefetching based on single user behavior characteristics is used in the solution [7].

3.3.1 Obtaining User's Behavior Characteristic

As to the theory of "80-20", the prefetching is based on the visit frequency of albums in the solution. The albums with high visit frequency are more likely to be accessed again. In the album management System, the visit frequency of every album is recorded by a table named album frequency in SQLite database. The frequency field is used to record the albums' visit frequency. When a user visits an album, the value of its frequency field will be automatically added one. Considering that some albums have extremely high access frequency over a period of time but low access frequency recently, the solution shall reset the value of frequency filed to zero periodically.

3.3.2 Implementation of Prefetching

In the implementation of prefetching, it is necessary to consider the time when to begin the prefetching photos and the prefetching amount which determines how many photos will be prefetched. Too small prefetching amount will weaken the prefetching effect while excessive amount will cause cache pollution [8]. In the album management system, prefetching is mainly applied to two scenes: the first scene and the second scene.

In the first scene, the system will start a thread to prefetch photos after loading the albums' list. By using while() circulation, the thread prefetches some photos from two albums with high access frequency recorded by album_frequency. The prefetching amount is controlled by duration of the while() circulation, which will be stopped when the value of flag variable equals false. If the required memory of the prefetched photos from Album 1 is more than 30% of the memory cache capability, the while() circulation will be stopped. If the required memory of the prefetched photos from Album 2 is more than 20% of the memory cache capability, the while() circulation will be stopped too. The default value of the flag is true. The value of the flag will be changed to false after an album icon is clicked.

In the second scene, the solution will start to prefetch photos when the interface stops sliding and the photos on the interface have been loaded. The prefetching amount is controlled by the duration of the while() circulation. The while() circulation will be stopped when the value of flag_scroll variable equals true. If the prefetching photo is the first or the last image of the album, the while() circulation will be stopped. The flag_scroll is used to mark the sliding state of the interface. When the interface begins to slide, flag_scroll will be set to true and prefetching will be stopped subsequently. The flag_scroll will be set to false and prefetching will be performed when the sliding is stopped. The process of the prefetching in the scene two is shown in Fig. 3:

1) The onTouchEvent() is rewritten to monitor the user's gesture. When the value of ev.getAction() equals to MotionEvent. *ACTION_DOWN, the* value of ev.getY() will be assigned to y_tmp1 and flag_scroll will be set to true;

2) When the value of ev.getAction() equals to MotionEvent.*ACTION_MOVE*, the value of ev.getY() will be assigned to y_tmp1 and flag_scroll will be set to false;

3) Load pictures on the current interface and assume that the position of the first picture on the current interface be m and the last one be n;

4) The sliding direction of the interface depends on y_tmp2 and y_tmp1. If the difference between y_tmp2 and y_tmp1 is greater than 0, the sliding direction will be upward and the K will be set to 1 and the next step will be 5). If the difference between y_tmp2 and y_tmp1 is less

than 0, the sliding direction will be downward and the K will be set to 1 and the next step will be 7).

5) The thread shown in Fig. 2 is called to prefetch the photo whose position is n + k.

6) If n+k is less than the total number of photos in the album and flag_scroll equals false, k will be added 1 and the next step will be 5); or else the prefetching will be stopped.

7) The thread shown in Fig. 2 is called to prefetch the photo whose position is m- k.

8) If the difference between m and k is greater than 0 and flag_scroll equals false, k will be added by 1 and the next step will be 7); otherwise, the prefetching will be stopped.

3.4 Monitoring the Sliding of the Interface

In the album management system, there will be stuck phenomenon when the user browses thumbnails by sliding the interface quickly because too many items of ListView will be loaded in a short time when the interface is sliding; however, it is unnecessary to load the items which flash on the interface. Then, the solution uses monitoring in which only the items on the stopped interface are loaded, as follows:

1) The onScroll() function will start a thread to load items which are visible within the current interface when the ListView control is initialized;

2) The getview() function is rewritten to declare the controls on items;

3) The onScrollStateChanged()function starts a thread to load the photos which are visible on the interface when the sliding is stopped.

4. Testing

The purpose of this test is to verify whether the solution can solve the problems or not. The test obtaining the loading time of pictures both from the optimized album management system and the original. The original system uses cache and compression to load pictures and the optimized one uses the solution to load pictures.

4.1 Testing Environment

Considering the unstable network, only the browsing of local albums was tested with the device of a Huawei C8816 telephone, which features the memory is 4G, the memory limit of 64M, and the CPU frequency of 1.2GHz. There are two albums in the phone, Album 1 and Album 2 containing 300 and 100 photos respectively. The required memory for each photo is between 0.6M and 1.8M.

4.2 Obtaining the Testing Date

The loading time of the first scene was obtained as follows:

1) The system time output by Log.i() in the click event of items of the

Listview control was used as the initial time;

2) Log.i() was called after loading picture for ImageView. The time output by the last ImageView on current interface was used as the final time.

The loading time of the second scene was obtained as follows:

1) When the sliding state was SCROLL_STATE_IDLE, the system time output by calling Log.i() in onScrollStateChanged() was used as the initial time;

2) The method of obtaining the final time was the same way as the first scene.

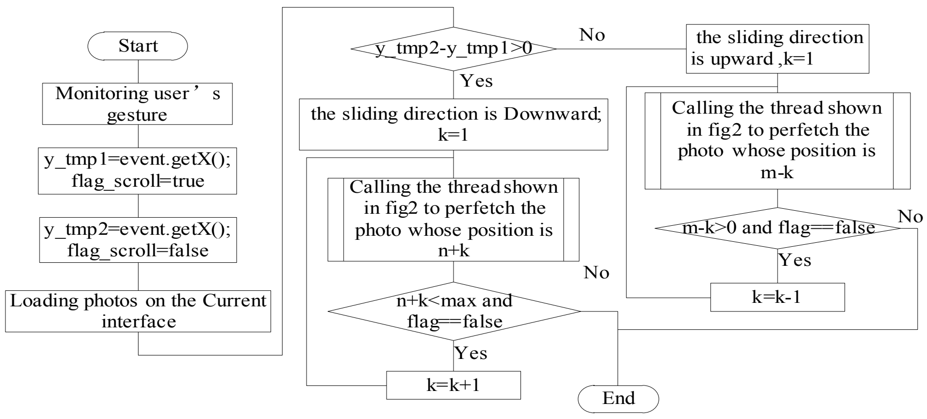

Start
Monitoring user' s gesture
y_tmp1=event.getX(); flag_scroll=true
y_tmp2=event.getX(); flag_scroll=false
Loading photos on the Current interface
y_tmp2-y_tmp1>0
No
Yes
the sliding direction is Downward; k=1
Calling the thread shown in fig2 to perfetch the photo whose position is n+k
n+k<max and flag==false
No
Yes
k=k+1
End
the sliding direction is upward ,k=1
Calling the thread shown in fig2 to perfetch the photo whose position is m-k
m-k>0 and flag==false
No
Yes
k=k-1

Figure 3: Process of Prefetching in the Second Scene

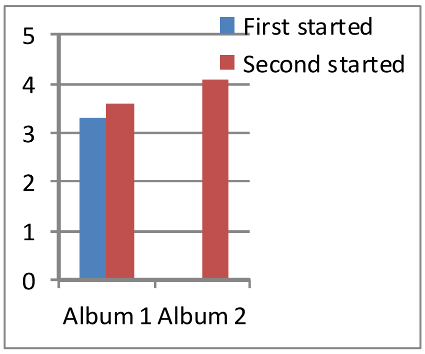
4.3 Testing Steps and Results Analysis

1) Firstly, the loading time of Album 1 was obtained by clicking it's icon in the first started system; secondly, the loading time of Album 1 and Album 2 was obtained in the restarted system. The data is shown in Fig. 4 and Fig. 5.

2) The loading time of the scene two was obtained when browsing thumbnails by page. The data is shown in Fig. 6.

3) The date of the scene three was obtained when browsing thumbnails quickly. The data is shown in Fig. 7.

It is shown in Fig. 5 that the loading time of Album 1 in the restarted system was less than that of the firstly started system after the system is optimized because the table album_frequency used to record the user's behavior characteristic was initialed and the visit frequency of Album 1 were added one after Album 1 had been loaded in the first started system. Then, some photos of album 1 had been perfeched before Album 1 was accessed in the restarted system based on the table album_frequency to reduce their loading time. It is shown in Fig. 6 that the loading time of the photos on the next page in the optimized system was less than that of the original system because in the optimized system, the photos on the next page had been perfeched before user browsed the next page according to sliding direction of the interface. It is shown in Fig. 7 that the items which flashed on the interface were not loaded in the optimized system; therefore, the stuck phenomenon was solved. It is shown in Fig. 8 that every photo was compressed and the maximum capacity of memory cache was 8M; therefore, the memory usage could not exceed the memory limit and the OOM could be avoided. All the above analysis showed that the solution could solve the problems of picture thumbnails browsing effectively.



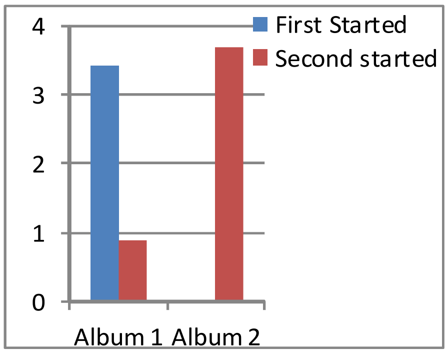


Figure 4: the Frist Scene in the Original System **Figure 5:** the First Scene in the Optimized System

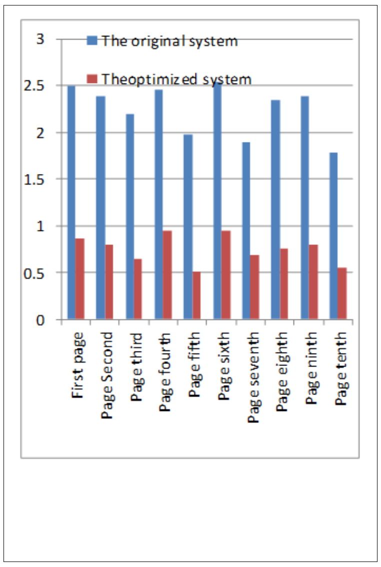
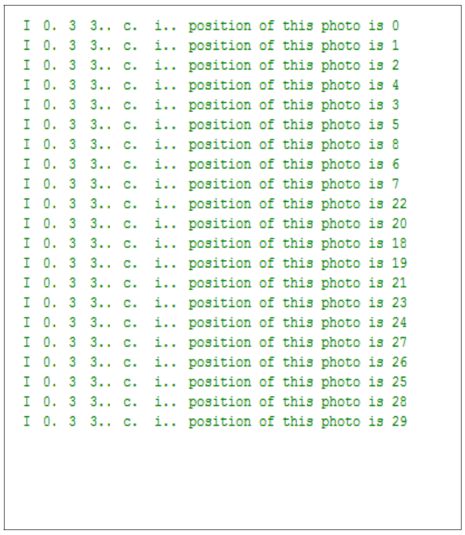
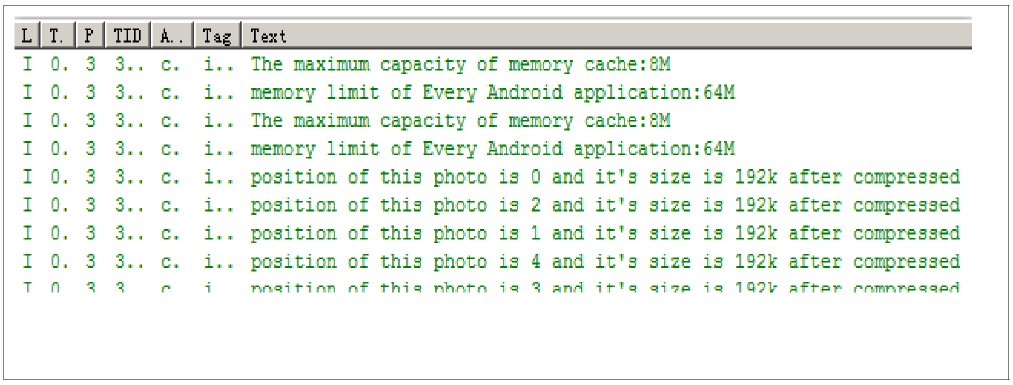


Figure 6: Date of the Second Scene



```
I  0.  3  3..  c.  i..  position of this photo is 0
I  0.  3  3..  c.  i..  position of this photo is 1
I  0.  3  3..  c.  i..  position of this photo is 2
I  0.  3  3..  c.  i..  position of this photo is 4
I  0.  3  3..  c.  i..  position of this photo is 3
I  0.  3  3..  c.  i..  position of this photo is 5
I  0.  3  3..  c.  i..  position of this photo is 8
I  0.  3  3..  c.  i..  position of this photo is 6
I  0.  3  3..  c.  i..  position of this photo is 7
I  0.  3  3..  c.  i..  position of this photo is 22
I  0.  3  3..  c.  i..  position of this photo is 20
I  0.  3  3..  c.  i..  position of this photo is 18
I  0.  3  3..  c.  i..  position of this photo is 19
I  0.  3  3..  c.  i..  position of this photo is 21
I  0.  3  3..  c.  i..  position of this photo is 23
I  0.  3  3..  c.  i..  position of this photo is 24
I  0.  3  3..  c.  i..  position of this photo is 27
I  0.  3  3..  c.  i..  position of this photo is 26
I  0.  3  3..  c.  i..  position of this photo is 25
I  0.  3  3..  c.  i..  position of this photo is 28
I  0.  3  3..  c.  i..  position of this photo is 29
```

Figure 7: Date of the Third Scene



```
L  T.  P  TID  A..  Tag  Text
I  0.  3  3..  c.   i..  The maximum capacity of memory cache:8M
I  0.  3  3..  c.   i..  memory limit of Every Android application:64M
I  0.  3  3..  c.   i..  The maximum capacity of memory cache:8M
I  0.  3  3..  c.   i..  memory limit of Every Android application:64M
I  0.  3  3..  c.   i..  position of this photo is 0 and it's size is 192k after compressed
I  0.  3  3..  c.   i..  position of this photo is 2 and it's size is 192k after compressed
I  0.  3  3..  c.   i..  position of this photo is 1 and it's size is 192k after compressed
I  0.  3  3..  c.   i..  position of this photo is 4 and it's size is 192k after compressed
I  0.  3  3..  c.   i..  position of this photo is 3 and it's size is 192k after compressed
```

Figure 8: Required Memory of Photos and Lrucache.

5. Conclusion

A solution based on perfecting and cache has been proposed to solve the problem of picture thumbnails browsing. This paper has illustrated the design and implementation of the solution. The main conclusions of this study can be summed up as follows: 1) Prefetching can reduce the loading time of non-accessed photos and combine with cache to shorten the system response time efficiently; 2) monitoring can be used to reduce the items to be loaded when the ListView is fast sliding, then the computational burden of the CPU can be reduced and the stuck phenomenon can be avoid. In order to further reduce the loading time of the photos, the next step is to promote the accuracy of prefetching.

References

[1] Ji Xie. *The Design and Implementation of The Mobile Client of Alibaba*. com Based on Android Platform[D]. Harbin:Harbin Institute of Technology,2012(In Chinese).

[2] Yang Liu. *Design and Implementation of Photo Viewer Based on Open GL ES Under Android Platform*[D]. Harbin:Harbin Institute of Technology,2012(In Chinese).

[3] Guojian Tan, *Key Technology Research of Mobile Internet Applications Based on Android*[D].Guangzhou: South China University of Technology,2014(In Chinese).

[4] Zhijie Ban, Zhimin Gu, Yu Jin, *A Survey of Web Prefetching*[J]. Journal of Computer Research and Development,46(2):202-210(2009).

[5] Wei Niu,Yanyan Zhang,*The Research on Web Prefetching Technology*[J].Microcomputer Applications, 29(7):90-94(2008).

[6] Libo Li,*Research and Improvement of Cache Replacement Algorithm in Energy Monitor and Control Platform*[D].Guangzhou:South China University of Technology,2011(In Chinese).

[7] XuLiang Wang, *Research and Application of Massive Data Caching Algorithms and Design Patterns*[D]. Zhejiang:Zhejiang University,2013.

[8] Yongyun Zhang, *Sample Analysis of Cache Prefetching Technology*[J]. Modern computer,13(2):38-40(2011).