A User-Relationship-Based Cache Replacement Strategy for Mobile Social Network

Qiyuan Xing
Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data
North China University of Technology
Beijing, China
ackoko11@gmail.com

Yue Li
Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data
North China University of Technology
Beijing, China
liyue5156@gmail.com

Jing Wang
Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data
North China University of Technology
Beijing, China
wangjing@ict.ac.cn

Yanbo Han
Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data
North China University of Technology
Beijing, China
yhan@ ict.ac. cn

In recent years, mobile applications grew rapidly with the development of Android and iOS platforms. Most of applications on these smart phones generate and make use of user-generated data. When users need relative data, it is not very realistic to request from the server every time. So a suitable cache technology is required. Traditional cache technologies pay more attention on data access time, frequency or data space, but do not consider data generators' relationships with each other. In mobile social network environment, data access is closely related to users' relationships, so this factor is suitable to be used in cache technologies. In this paper, we proposed a user-relationship-based cache replacement strategy. We combined user relationship with the classic cache algorithm LRU. Not only the access times of each data blocks are considered, but also users' relationships are computed by the closeness between data requesters and generators. The experiment results show that our replacement strategy can improve the cache hit ratio in mobile social environment.

Keywords—cache replacement; social network; mobile computing;

I. INTRODUCTION

Recent advances in the mobile technology have brought the notion of "anything, anytime, anywhere" information access close to reality. More and more applications have been created and are becoming an important part of human's daily life. According to the data from website NetMarketshare, smart phones share of market is more than 90 percent. Due to the limited memory and computing ability of wireless mobile terminals, mobile applications often use the architecture of "Cloud and Terminal". When wireless mobile terminals need data, they can make requests and get required data from cloud servers. Because the mobile devices have limited bandwidth and the wireless mobile network is not always stable, data transfers between mobile terminals and cloud servers do not perform very well. For solving these problems, a common method is to use suitable mobile cache technologies. Mobile cache technology refers to store frequently accessed data into a mobile node to reduce access frequency to the server node. Therefore it can reduce pressure on the server, save energy and mobile communication bandwidth, and reduce client's response time.

Recent years, social network is becoming a basic module in most mobile applications. Mobile social networking is that where individuals with similar interests converse and connect with one another through their mobile phone and/or tablet[8]. Usually it is related to user's behavior and habits.

Wireless mobile terminals travel along with users and do an excellent job of exhibiting social properties, such as similar interests to social hotspots, direct or indirect friendships, geographic features, etc. Data in mobile social network always closely related to users' relationships, and thus traditional ways will not be as suitable as before.

This paper uses people's social relationships to propose a mobile cache replacement strategy for mobile social environment. We focus on the wireless mobile owners' relationship with others in social network, and make use of them in mobile cache replacement. In mobile social network, what people do most is to communicate with others. People often read status generated by others in a homepage in Facebook, or read their followers' tweets in Twitter. And people often send messages to other users in WeChat.Mobile applications will load these user-generated data. If requesting these data every time, it will cost a lot. So storing some data in cache is suitable. Because these data is generated by users, this paper proposes a user-relationship-based cache block model, and computes each cache block's value combined with time and users' relationships to improve the cache hit ratio. When

978-1-4673-9295-2/15 $31.00 © 2015 IEEE
DOI 10.1109/FCST.2015.54

the mobile terminal gets a new data block and the cache space in this mobile terminal is not enough to store, it will find the useless cache blocks and replace them with the new data block. Experiment shows that it can improve the cache hit ratio when compared with LRU cache replacement algorithm.

The remainder of this paper is organized as follows. Section II describes a summary of related works and their limitations. Section III gives the detailed design and implementation of the user-relationship-based cache replacement strategy. Section IV presents data preparation and simulation, and then evaluates the results. We summarize the work in Section V.

II. Related Works

Cache technology is not a new research direction in the field of computer science. Many researchers have done many researches in this direction. Such as the classic cache replacement algorithm Least Recently User (LRU), it is to discard and replace least recently used items. And First In First Out (FIFO) is to replace the first entered item in the cache space. And others like LRU2, 2Q, ARC, Second Chance, Clock are designed based on above two strategies.

In mobile environment, because of the change frequent of wireless mobile terminals, traditional cache strategy may not be used very suitable. Some researcher did many works to find more suitable cache strategies.

(1) Access Frequency-based Replacement Strategy

Leong and Si proposed the cache replacement strategies to suitable better in mobile environment. A cached object is replaced when it has the highest mean duration among all objects. But it reacts slowly to data change.

(2) Gain-based Cache Replacement Strategy. These cache replacement strategies are through compute the function values to determine the policy cost on system performance, and then select the items to be replaced. Such cache replacement strategies have SAIU, Min-SAUD, SXO, etc.[11]

But when considered mobile holder's relationship in his social network, these strategies above may not useful as before.

(3) Semantic-based Cache Replacement Strategy. These strategies want to establish associated relations by analyzing cached content in morphology and syntax. And then find out caches which have weaker relationship with others, and replace them. In mobile environment, computing morphology and syntax cost more that cannot be easy used.

Recently years, some researchers have taken account of user behavior in cache technology.

Paper [2] proposed a novel social relation based cache distribution policy. It goal is to detect the influential users through the interaction within web users. And then to predict what data will be replaced in the future. This policy use users' relationship to help to distribute data into cache server. It wants to store data in cache to meet user's future request. However, this policy need more cache space to store these data. Many mobile applications cannot meet this request.

Paper [5] think query results caching and prefetching are crucial to the efficiency of Web search engines. So they presented a novel approach tailored for query results caching and prefetching based on the user characteristics. And design a query results prediction model for prefetching and to partition the cache exploiting the characteristics of the users. Experimental results show that this approach is good, but the shortcoming is it also need more cache space and do not take user's relationship into consideration.

In this paper, we put forward a user-relationship-based cache clock model. After that, we compute each cache block's value combined with user's relationship and access frequency. And then we put forward the user-relationship-based cache replacement strategy.

III. User-Relationship-Based Cache Replacement Strategy

In this part, we define the *Cache Block Model* for mobile social network environment, and put forward *Closeness Value* and *LRU Value*. After that, we design the *Cache Replacement Model* and define *Cache Block Priority Value*. Finally, we propose a user-relationship-based cache replacement strategy and describe it in detail.

A. Cache Block Model

In mobile environment, when we see the user-generated data like status in Twitter, mobile application usually shows us a page of data one time. When people scroll the screen but the data in mobile terminal is not enough to show, the mobile terminal will request more data from cloud server. Usually these data is another page of data items.

So in this paper, we select a page of user-generated data as cache granularity. It can be several items of Tweets in Twitter. Also can be several items of user's chat history in WeChat.

The cache block model in cache is defined as follows.

$$CB = (Gid, Position, CV, Page, Size, Content) \quad (1)$$

Gid represents for the cache block's content is generated by a user, and his id in cloud server's database is Gid. $Position$ represents for the cache block's position in cache. CV represents for the *Closeness Value* between mobile terminal owner and the user who generated these page of data. We will describe it in detail later. $Page$ is the page number request by current user. And $Size$ tell us the space of the data. $Content$ represents for the page of data request by current user.

B. Closeness Value

According to the basic theory of social network, we know that a person can relate to anyone through friends by transferring a certain number. That is to say each two users can produce a weak relationship through some certain links. We always use closeness rate to represent this weak relationship. And in social network, two members' closeness rate can be represented by the *Closeness Value*. Let us talk about how to compute *Closeness Value* between two users. Firstly, we should get the *Closeness Value* between each two direct linked nodes in the social network graph. Usually it should be computed by interaction or common hobbies or other factors. Then we can compute the *Closeness Value* by transferring through some not direct linked nodes.

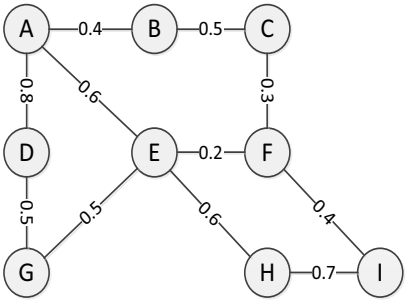

Fig. 1. A simple social network.

Figure 1 is a small social network graph which has nine members. We use $\lambda(u_i,u_j)$ to represent the *Closeness Value* between two direct linked members. Just like $\lambda(u_A,u_B)$ = 0.4. This value should be computed in advance. It can be computed by users' behavior, users' similarity or some others factors.

The value between two members in a path should be computed by the product of each two direct linked members' *Closeness Value* in this path. We use $d(u_0,u_1...u_n)$ to represents the path from u_0 to u_n in the path.

$$d(u_0,u_1...u_n)=\prod_{i=0}^{n-1}\lambda(u_i,u_{i+1}) \qquad (2)$$

Just like the path $A-D-G$ in the picture, the value should be $\lambda(A,D)\times\lambda(D,G)$ = 0.8 * 0.5 = 0.4. We know that if the number in a path large enough, the value will be very small and can be ignored. And if the numbers in a path increase, the value will decrease rapidly. *Six Degrees of Separation* [9] also tell us a person can know another in no more than six people. In here, we rule that the number in a path cannot more than 3 to make our experiment easier.

The *Closeness Value* is computed as follows.

$$\begin{cases}\mathrm{CV}(u_i,u_j)=\sum d(u_i,...,u_j), & (if\ \ u_i\ and\ u_j\ can\ be\ linked);\\ \mathrm{CV}(u_i,u_j)=+\infty & else;\end{cases} \qquad (3)$$

It shows that the *Closeness Value* between two members is the sum of each path's value which started by u_i and ended by u_j. $CV(A,G)$ = $d(\mathrm{A},D,G)$ + $d(\mathrm{A},E,G)$ = 0.8 * 0.5 + 0.6 * 0.5 = 0.7.

C. LRU Value

Our cache replacement strategy is expended by LRU(Least Recently Used) algorithm. When we compute every block's value in cache, we compute the block's *LRU Value* firstly. We let the value of the k's item in cache is:

$$LRU(k)=\frac{(M+1-k)}{\sum_{i=1}^{M} i} \qquad (4)$$

M is the size in this cache. k is the position of current item in cache. $LRU(k)$ represents a cache block's *LRU Value* whose position is k. Each time we use user-relationship-based cache replacement strategy, the cache block be accessed will be moved to the first position. If a cache block be accessed recently, it will have a smaller position number. Otherwise, if a cache block does not be accessed for a long time, it will be move behind and will have a larger position number. Our algorithm is based on the last time of access, and each item in cache has a *LRU Value*. In (4) we can see that the sum of every item's *LRU Value* in cache is equal to 1. The smaller the position is, the bigger *LRU value* an item will has. If a data block is requested and can be found in cache, the cache block will be moved to the first position, and it will have the biggest *LRU Value*.

D. Cache Replacement Model

The *Cache Replacement Model* in this paper is described like follows.

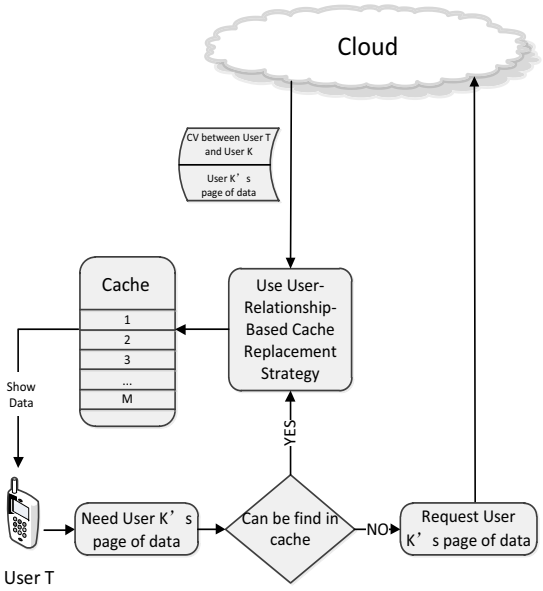

Fig. 2. Cache replacement model.

To begin with, user T wants to see the data generated by user K from a mobile application. The application needs the page of data and show them to user T. It will check if these data exist in cache firstly. If these data can be find in cache, using cache replacement strategy to change data's position in cache and then showing them on the mobile screen. Otherwise it should post a URL to the cloud server to request for these data. The server in cloud will compute the *Closeness Value* between user T and user K, and return the data block and *Closeness Value* to wireless mobile terminal. When the terminal gets the *Closeness Value* and the data block, it will check if the cache is full. If the cache has space to store, it will store the data and show the content on screen. But if the cache does not have enough space to store these data, it will use the

user-relationship-based cache replacement strategy to find the useless cache block, and then insert the new arrival data block to first position.

E. Cache Block Priority Value

In here, we let every cache block has a value to be used for user-relationship-based cache replacement strategy. When we use the cache replacement strategy, we will firstly compute each item's value in cache to represent the useful of a cache block. We define this value as *Cache Block Priority Value* in this paper.

Cache Priority Block Value is defined as follows:

$$V_k = \sigma 1 * CV(u_T, u_K) + \sigma 2 * LRU(k), \quad \sigma 1 + \sigma 2 = 1 \qquad (5)$$

$CV(u_T, u_K)$ is represents the *Closeness Value* between u_T and u_K, and shows that this data is request by u_T and generated by u_K. $\sigma 1$ is the weight of *Closeness Value* between u_T and u_K. $LRU(k)$ is the *LRU Value* of a cache block which position in cache is k. $\sigma 2$ is the weight this cache block's *LRU Value*. The sum of $\sigma 1$ and $\sigma 2$ is equal to 1. V_k represents the *Cache Block Priority Value* of one cache block. We can see that an item's *Cache Block Priority Value* is computed combined with its *Closeness Value* and *LRU Value.* We can make our strategy more effective by changing the value of $\sigma 1$ and $\sigma 2$.

F. URB Cache Replacement Strategy

We define our user-relationship-based cache replacement strategy as *URB Cache Replacement Strategy*. It works as follows.

INPUT: ***Gid*: the request data's generator id.**
***page*: the page number of data.**

OUTPUT: ***Content*: generated by user whose id is *Gid*, and page number is *page***

BEGIN
If(The content can be find in cache)
{
Move the item in cache to the first position.
Move other items before that cache block to next position, that is set position = position + 1
}
Else
{
Request for data from cloud server
Get the request data and the data generated user's *Closeness Value* with mobile terminal owner
If(Cache have enough space to store these data)
{
Move each cache block to next position, that is set position = position+1
Insert the new data block to the first position.
}
Else
{
Calculate the each item's *Cache Block Priority Value* in cache.
If(new data block's size < cache block which has minimum *Cache Block Priority Value*)
{
Remove the cache block which has minimum *Cache Block Priority Value*.
}
Else
{
Remove the cache blocks which have minimum *Cache Block Priority Value*, and their sum of size just exceed the size of new data block.
}
Insert new cache block to the first position.
Change other item's position.
}
}
Return first item's content in cache
END

IV. PERFORMANCE EVALUATION

To test the efficiency and effectiveness of the *URB Cache Replacement Strategy*, we constructed a simulation experiment.

A. Data Preparation

Our dataset have 2400000 items from Twitter[1]. One item in the dataset has two numbers and it represents for a relationship. One number is the followed user's id and another number is the following user's id. Firstly, we use Java program language to compute each one followed user have how many following users. We totally have 19148 users in the dataset. Statistics suggest that about 10000 users only have 10 less following people, so it is to less to test our algorithm. They are not suitable for us to do experiments. About 1000 users have more than 400 following users. They have too many following users that they may not use Twitter as normal people. Then we select 1000 users whose following people number is between 100 and 400. Their following numbers is suitable as a normal Twitter user and is good for us to do experiment.

According to references, we can know that wireless mobile terminal's requesting for data obeys Zipf's Law. It indicates that when user's accessing data of other user's Tweets, about 80 percent of these request data are created by 20 percent users. In mobile social networks, we can know that people often contact with their closely friends, so these 20 percent users are those who have the most closely relationship with current user. That is who have larger *Closeness Value* with current user.

Here, we referred to reference [6]. In our experiment, we use a similar way to set *Closeness Value* between direct linked users in the Twitter dataset. We define the direct linked users'

[1] http://www.datatang.com/datares/go.aspx?dataid=616605

Closeness Value with current user which is $\lambda(u_i, u_j)$ in Section III as follows. If current user u_i and another user u_j followed each other, and they followed 10 more common users, or they be followed by 10 more common users. It shows that they are very close friends, so we set their *Closeness Value* $\lambda(u_i, u_j)$=9.0. Else if u_i and u_j followed each other, and they followed 5 more common users, or they be followed by 5 more common users, we set $\lambda(u_i, u_j)$=0.8. Else if u_i and u_j followed each other, and they followed 5 less common users, or they be followed by 5 less common users, the *Closeness Value* between these two direct linked user $\lambda(u_i, u_j)$ =0.6. Else if u_i followed u_j but u_j did not follow u_i. And they two users followed 10 more common users, or they be followed by 10 more common users, we set the *Closeness Value* $\lambda(u_i, u_j)$=0.4. Otherwise Their Close Rate $\lambda(u_i, u_j)$=0.

In our experiment, we use a piece of code of Matlab to generate 100000 random numbers from 1 to 100000, and these 100000 random numbers obey Zipf's Law. That is to say 80 percent of these 100000 numbers is 20 percent between 1 and 100000. We use these data to simulate user's request data from the server in cloud. Then we let these 100000 numbers relate to the user ids in Twitter data. If a number occurs more frequently in the 100000 random numbers, it means that current user request data generated by this user more. We should let this number related to a user those who has a higher *Closeness Value* (CV).

B. Simulation Settings

The server in cloud is a virtual machine which has dual-core Core 2.4 GHz, 16GB RAM, 1 TB RAID5 Disk. And the operating system is CentOS 6.2, and has access to 1GB Ethernet. In server, we use Java language. The database is MySQL 5.1. And we use Java program to compute the *Closeness Value* and send it and Tweets of Twitter's user (related to 100000 random numbers) to wireless mobile terminals.

The wireless mobile terminal, we use a Meizu Mobile Device. It has MT6752 1.7GHz CPU, 2GB RAM, 32GB ROM. And use WCDMA network. And we created a test App to test the cache hit ratio. We wrote a function in this App to request for Tweets created by other users (100000 random numbers) one by one to simulate user's behavior in daily life.

We set $\sigma 1$ be 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0. And $\sigma 2$ is 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.

In this experiment, we set cache size be 100. If we use a higher cache size, it will reduce the number of request for data from cloud. So we use this suitable number as cache size.

Firstly, we select a user who has 200 following users from the 100 users we prepared. The results of LRU and user-relationship-based cache replacement strategy are shown below.

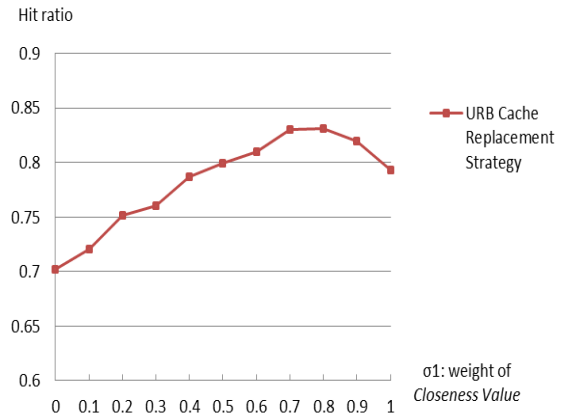

Fig. 3. Simulation results of cache hit ratio with the change of user's *Closeness Value* weight.

Figure 3 shows that when we set $\sigma 1$ be 0.7, the user-relationship-based cache replacement strategy have best performance. The reason why is as follows. User's relationship is closely related to data's access. If $\sigma 1$ no more than 0.7, the user's relationship is not paid enough attention. But if $\sigma 1$ more than 0.7, the last access data affect less.

So we set $\sigma 1$ =0.7, and $\sigma 2$ = 0.3. Then we use the all 1000 users to do experiments. We let the average cache hit ratio be the result. The results of LRU and user-relationship-based cache replacement strategy are shown below.

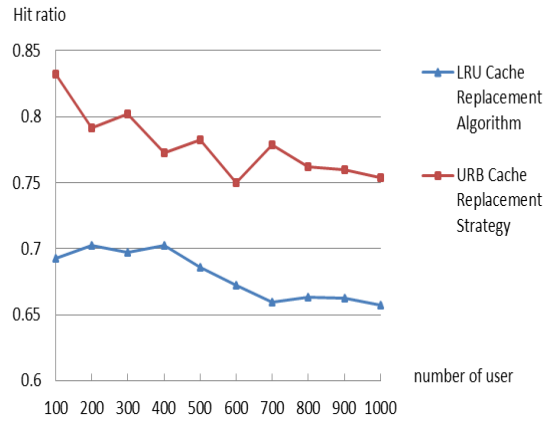

Fig. 4. Simulation results between user-relationship-based cache strategy and LRU algorithm.

Figure 4 shows that with the increase of user's number, the cache hit ratio is decline in whole. User-relationship-based cache replacement strategy performs better than LRU cache replacement algorithm.

V. CONCLUSION

In this paper, we have explored the cache replacement in mobile environment and presented a new user-relationship-based cache replacement strategy URB for mobile social network. The strategy takes into account the access frequency and users' relationships for cache replacement. Then we used Twitter data to do simulation experiments and compared the cache hit ratio between URB and LRU. Experimental results show that URB cache replacement strategy can significantly improve the cache hit ratio when compared with LRU cache replacement algorithm.

ACKNOWLEDGMENTS

This research has been partially supported by the Natural Science Foundation Project of Beijing Municipal under Grant No. 4131001, Scientific Research Common Program of Beijing Municipal Commission of Education under Grant KM201310009003 and Project of Construction of Innovative Teams and Teacher Career Development for Universities and Colleges under Beijing Municipality under Grant No.IDHT20130502.

REFERENCES

[1] Jain D K, Sharma S. Growth Rate of Cached Data Items at clients in Mobile Ad Hoc Networks[C]//Wireless Computing and Networking (GCWCN), 2014 IEEE Global Conference on. IEEE, 2014: 157-159.

[2] Yang L, Qin Y, Zhou X, et al. Social Relation Based Cache Distribution Policy in Wireless Mobile Networks[J]. Journal of Networks, 2014, 9(9): 2279-2288.

[3] Wang Y, Wu J, Xiao M. Hierarchical cooperative caching in mobile opportunistic social networks[C]//Proc. of IEEE GLOBECOM. 2014.

[4] Kumar M V R, Swati M M. Cache Replacement Algorithms for Coordinated Cooperative Social Wireless Networks[J]. 2014.

[5] MA Hongyuan, WANG Bin. A Query Result Caching and Prefetching Approach Based on Query Characteristics[J]. Journal of Chinese Information Processing, 2011, 25(5).

[6] Liu Q. Friend Re commendation g Based on Social Network and Location Information [D]. Master's thesis, Zhejiang University, 2013.

[7] Brânzei S, Larson K. Social distance games[C]//The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3. International Foundation for Autonomous Agents and Multiagent Systems, 2011: 1281-1282.

[8] http://en.wikipedia.org/wiki/Mobile_social_network

[9] http://en.wikipedia.org/wiki/Six_degrees_of_separation

[10] Madhukar A, Özyer T, Alhajj R. Dynamic cache invalidation scheme for wireless mobile environments[J]. Wireless Networks, 2009, 15(6): 727-740.

[11] Rathore R, Prinja R. An Overview of Mobile Database Caching[J]. CiteSeerX, doi, 2007, 10(1.100): 9481.

[12] Hong V. Leong, Antonio Si "On Adaptive Caching in Mobile Databases" April 1997, Proceedings of the 1997 ACM symposium on Applied computing