一种基于社交关系的移动缓存替换算法

邢起源 王 菁 闫阿宾 韩燕波
(北方工业大学云计算研究中心 北京 100144)
(大规模流数据集成与分析技术北京市重点实验室 北京 100144)

摘 要 近年来移动互联网尤其是Android平台和iOS平台的兴起,移动应用的数量出现了爆炸式增长。这些移动应用中,用户可以发布数据或浏览其他用户发布的数据,由此产生了大量用户生成的数据。当手机使用者想要浏览这些由其他用户生成的数据时,每次都向服务器请求数据的做法是不可取的,比较合适的方法是使用缓存技术将部分数据缓存在移动端,以此来降低数据的请求频率,减小无线网络带宽压力,提升用户体验。传统的缓存技术更多关注的是缓存的访问频率、最近访问时间等因素,但是很少关注数据生成用户之间的社交关系。在存在用户关系的移动网络中,用户相关数据的请求与用户之间的社交关系紧密联系。结合用户之间的社交关系、最近最久访问时间以及缓存中每块数据的大小,提出了一种基于社交关系的移动缓存替换算法。该算法综合计算数据的最近最久访问时间、数据产生用户与使用者之间的亲密值以及缓存占用存储空间的大小,在需要时进行缓存替换。实验证明在移动社交网络中,基于社交关系的缓存替换策略可以提高缓存命中率,使用户获得更好的体验。

关键词 移动社交网络,移动应用,最近最久未使用,亲密值,缓存大小
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.009

Mobile Cache Replacement Algorithm Based on Social Network

XING Qi-yuan WANG Jing YAN A-bin HAN Yan-bo
(Research Center for Cloud Computing, North China University of Technology, Beijing 100144, China)
(Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, Beijing 100144, China)

Abstract In recent years, mobile applications grow rapidly with the development of Android and iOS platforms. Most of applications on these smart phones are based on users, and these data are always generated by users. When users want relative data, it is not very realistic to request the data from a server every time. So a suitable cache technology is required. Traditional cache technologies pay much attention to the frequency or the last access time, but do not consider more about data generators' relationship between data-generators. In mobile social network environment, data access is closely related to users' relationships, so this factor should be suitable for use in cache technologies. In this paper, we proposed a user-relationship-based cache replacement algorithm, and combined users' relationships with the classic cache algorithm LRU. Not only the access time of each data, but also the closeness value between the data requestors and the generator was taken into consideration. The experiment results show that our replacement strategy can improve cache hit ratio in mobile social environments.

Keywords Mobile social network, Mobile application, Least recently used, Closeness value, Cache size

1 引言

随着移动互联网技术的快速发展,人们已经逐渐习惯在生活中随时随地使用智能手机中的各种应用来处理日常事务。根据NetMarketshare网站的统计数据,当前智能手机的市场占有率已经超过了90%,移动应用在逐渐改变着人们的生活方式。当下的移动应用大多采用"云+端"的方式来请求数据:当移动应用需要数据时,它会通过无线网络向云端服务器发送请求,云端服务器会返回相应的数据,移动应用接收到相关数据并呈现在手机屏幕上。但是由于移动终端本身的内存和计算能力有限,并且无线网络传输并不稳定,每次都从云端服务器请求所有的数据会消耗很多网络流量。一个常用的解决方法是使用合适的缓存技术。移动缓存技术是指把经常访问的数据缓存到移动节点上,以减少移动节点对基节点的访问,从而减小服务器压力,节省移动主机的电能和通信带宽,减少客户端响应时间,进而提升用户体验。

到稿日期:2015-06-29 返修日期:2015-09-07 本文受北京市属高等学校创新团队建设与教师职业发展计划项目(IDHT20130502),北京市自然科学基金(4131001)资助。

邢起源 男,硕士生,主要研究方向为移动计算与云端集成,E-mail:ackoko11@gmail.com;**王 菁** 女,博士,副研究员,主要研究方向为云计算与服务计算,E-mail:wangjing@ict.ac.cn;**闫阿宾** 男,硕士生,主要研究方向为移动计算与云端集成,E-mail:1576702791@qq.com;**韩燕波** 男,博士,教授,博士生导师,主要研究方向为分布式系统、云计算、大规模数据集成、互联网服务、业务流程管理等,E-mail:yhan@ict.ac.cn。

根据艾瑞咨询公司2014年的移动应用市场研究报告，移动端社交网络App覆盖用户占比已经达到60.2%，而且呈快速上升的趋势。移动社交应用在使用过程中产生了大量的数据，其中很多都是由用户发布生成的：1）比如国外的Facebook和国内的人人网中的状态。用户在使用移动应用时最为常见的操作就是查看一条条的状态，而大多数状态都是由与当前用户存在好友关系的用户发布的。再比如Twitter和国内的微博，网站内的Tweet或微博也是由用户编撰发布的。2）在移动即时通讯软件中，如腾讯公司的微信和手机QQ，用户经常查看好友的历史聊天记录。3）查看其他用户的个人信息也是一个经常使用的功能。

当用户在移动环境中进行上述操作时，如果每次都从服务器请求查看相关数据，将会消耗很多的时间和网络流量。并且由于移动终端的存储能力有限，将这些数据全部保存在移动终端的数据库中的做法是不可取的。因此，开辟一片存储空间将一些访问频率比较高的数据缓存进去是一种较好的解决方案。本文以这些用户生成数据的发布用户之间的社交关系为出发点，提出了一种基于社交关系的移动缓存替换算法。当用户使用移动终端查看数据时，该算法先从缓存中查找数据是否存在，若不存在则从云端服务器端请求数据。当接收到服务器返回的数据后再调用缓存替换算法将不合适的数据替换出缓存，将新数据插入缓存。实验证明，在存在社交关系的移动环境中，该算法相比传统缓存替换算法能取得更好的效果。

本文第1节为引言，引出并介绍问题；第2节介绍了移动缓存替换的相关工作；第3节详细描述了提出的基于社交关系的移动缓存替换算法；第4节进行了实验的设计和验证；最后总结全文。

2 相关工作

缓存技术是计算机领域中的经典技术，许多研究者在操作系统和数据库领域中提出了一些缓存替换算法。1）例如经典的缓存替换算法(Least Recently Used，LRU)，其目标是把最近使用频率最低的数据替换出存储空间。First in First out (FIFO)是按照数据存入存储空间的先后顺序，将先进入的替换出存储空间。还有一些如LRU2、2Q、ARC、Second Chance的缓存替换算法也是基于上述两个算法。2）在缓存空间方面，Size算法的目标是将最大的内容替换出Cache，LRU—MIN算法力图使被替换的文档个数最少。

在移动环境下，由于移动终端存在一些新的特点，比如客户端的位置经常变化，随着位置的变化，用户请求的数据也会有所差别。移动用户对数据的访问也常常与用户关系密切相关，频繁请求的数据往往是与当前用户关系密切的用户有关。传统的环境下不曾考虑这些因素，因此传统的缓存替换算法在移动环境下的效果并不好。一些研究人员结合用户之间的关系，提出了一些移动环境下的缓存替换算法。

文献[2]提出了一种基于社会关系的高速缓存分配策略，它通过网络用户之间的相互作用检测出有影响力的用户，再结合朋友关系来预测有可能被替换的内容。这种策略通过提前分配缓存到缓存服务器来尽可能满足无线移动用户即将发生的数据请求。但是这种策略主要运用于有较大缓存空间或存在缓存服务器的场景，现有的移动应用存储空间有限，不足以在移动终端中创建缓存服务器来存储大量数据。

文献[3]针对搜索引擎查询结果提出了一种基于用户特性的缓存与预取方法，用于提高搜索引擎系统的性能。通过分析国内某著名商业搜索引擎用户的查询贡献，得出用户对搜索引擎的贡献具有长尾分布特性，结合该特性设计了查询结果预测模型来进行预取和分区缓存。该策略主要针对搜索引擎相关领域对搜索结果进行缓存，并和用户相关联来进行缓存预取和分区管理，但是对于大多数常见的移动应用搜索数据以外的其他数据的数据请求并不合适。

文献[4]设计了一种大容量的缓存算法，提出把缓存空间分为3部分，分别用来缓存前缀部分、后缀部分和公共部分，并分别设计了准入控制策略和替换策略；分段方式采用线性分段方式；采用重视即时信息、兼顾历史信息的价值函数的方式来进行缓存替换等工作。该算法对缓存空间的要求比较高，对用户之间的关系未做较为深入的研究，因此对存在用户关系的移动应用来说也并不十分合适。

本文提出的缓存替换算法从数据的产生用户与当前用户的社交关系出发，结合传统的LRU策略，并考虑到每个缓存对象占用存储空间的大小，使每个缓存数据块都拥有价值，并将其作为缓存替换的依据。当需要缓存替换时，用新的数据块替换出价值比较小的若干数据块，以此来提高缓存的命中率，并更加合理地分配缓存空间。

3 缓存替换算法

本节首先定义了适用于存在社交关系的移动环境中的缓存对象，然后描述了缓存替换模型，紧接着提出了一种缓存对象之间的亲密值计算公式、LRU价值的计算方法以及每个缓存对象的价值计算公式，最后详细解释了缓存替换算法的执行流程。

3.1 缓存对象

在移动应用中，当用户查看状态或聊天记录等其他用户生成的数据时，移动应用往往是按照分页的方式从云端服务器请求一页数据，然后将其呈现在屏幕上。用户可能会通过滑动屏幕来查看，当查看到最后一条状态时继续滑动，移动应用会通过再次向云端服务器请求另一页数据，将更多的数据返回给移动终端供用户浏览。因此，本文将一页用户产生的数据作为一个缓存对象，一个缓存对象中可以包含若干条记录。在移动应用中，一条记录可以是微博或Twitter中的一条状态，可以是聊天记录中的一条消息，也可以是一个用户的个人信息。

在缓存空间中，首先需要针对不同的内容划分不同的区域，缓存替换时只针对内容所属的类型在相应的缓存空间中执行缓存替换算法。本文选取存放状态的缓存空间进行讨论。

缓存块的逻辑结构为如下表示的六元组：

$$CB=(ID_{CB},C_{CB},V_{CB},P_{CB},S_{CB}) \quad (1)$$

其中各属性的含义如下：

1）ID_{CB}，表示缓存对象 id 编号，由发布用户 id 和内容的页号拼接而成，用来唯一标识缓存内容。

2）C_{CB}，表示缓存对象中存储的内容，即若干条由其他用

户发布的数据。数据的内容可以是文本，也可以是占用存储空间较大的图片等数据。

3) V_{CB}，表示缓存对象的总价值。它取决于该缓存块的访问频率、该缓存块中内容的发布者与当前用户的亲密程度以及缓存对象占用存储空间的大小。3.3 节给出其详细计算方法。

4) P_{CB}，表示缓存对象在存储空间中的位置，本算法结合了最近最久未使用缓存替换算法（Least Recently Used，LRU），因此缓存对象的位置会对替换产生影响，在计算 LRU 价值时将会用到。

5) S_{CB}，表示缓存对象占用存储空间的大小，在缓存替换时，不同缓存对象的大小并不一定相同，其占用存储空间的大小也会影响缓存的替换。

3.2 缓存替换模型

使用移动应用时较为理想的情况是当用户需要访问相关的数据时，可以直接在缓存中读取到并将其呈现在界面上，本文的缓存替换模型便基于此。如图 1 所示，假设当前用户为 T1，试图浏览用户 T2 发布的一页数据，移动应用会首先在缓存中查找是否存在。若缓存中能够找到一个缓存对象，其内容就是要请求的数据，则将其内容呈现到屏幕上，并将缓存空间中比这个缓存对象的位置值小的对象的位置值增加 1，将这个缓存对象放到第一个位置并将位置值设置为 1；否则，移动应用通过网络向云端服务器发送请求，云端服务器计算后返回请求浏览的内容及当前用户和数据产生用户之间的亲密值。移动终端接收到云端服务器返回的数据后，一方面将返回的内容呈现在屏幕上。另一方面，调用基于社交关系的缓存替换算法，得到各个缓存块的价值。再将价值进行排序，当价值最小的若干个缓存对象占用的存储空间刚好大于新的数据块所占用的存储空间时，将这些缓存对象从缓存中移出，将新的缓存块插入缓存中第一个位置。

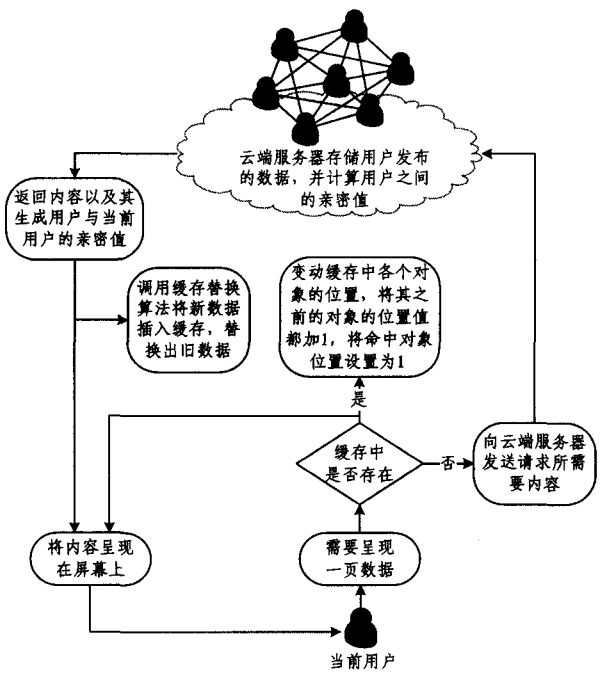

图 1 缓存替换模型原理图

3.3 缓存价值公式

根据社交网络中的六度分割理论等概念，我们可以知道存在于社交网络中的任何两个陌生人之间，只要通过有限次数的传递，都能够产生一定的联系。也就是说任意两个用户之间通过其朋友、朋友的朋友等若干次的联系，都能够产生一个连接关系。本文提出两个用户之间亲密值的概念来表示他们之间在社交网络中的亲密度。亲密值的计算方法如下：如果两个用户之间存在直接相连关系，如好友关系或关注被关注关系，称之为直接亲密值，可以由两个用户之间的联系频率或其他因素直接得出；如果两个用户之间不直接相连，则称之为间接亲密值，根据两个用户在社交网络中的间接连接路径综合计算。

图 2 是一个简单的社交图谱，图中有 A、B、C、D、E、F、G 共 7 个结点，表示社交网络中的 7 个用户。图中连接两个结点的线段表示两个用户之间存在直接连接关系，线段上的数值表示两个用户之间的直接亲密值。本文用 $d(u_i,u_j)$ 来表示直接亲密值，如图 2 中的 $d(u_A,u_B)=0.7$。直接亲密值的大小介于 0 到 1 之间。直接亲密值的设置与用户之间的共同好友数量或者用户之间的联系频率、次数等因素相关。

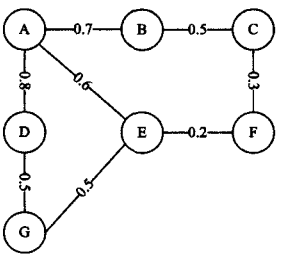

图 2 社交图谱

两个不直接相连的用户之间的亲密值由间接亲密值表示。在计算间接亲密值之前，首先定义某一条路径上的首尾两个用户之间的关系如下：

$$l(u_0,u_1,\cdots,u_n)=\prod_{i=0}^{n-1}d(u_i,u_{i+1}) \tag{2}$$

其中，$l(u_0,u_1,\cdots,u_n)$ 表示用户 u_0 与用户 u_n 之间存在一条 $u_0,u_1,\cdots,u_n$ 的路径，用户 u_0 与用户 u_n 之间在 $u_0,u_1,\cdots,u_n$ 这条路径上的关联度为路径上的各个线段之积。比如图 2 中的 $A-D-G$，在这条路径上，用户 A 与用户 D 之间的联系为 $d(A,D)\times d(D,G)=0.8*0.5=0.4$。由于两个结点之间的直接亲密度在 0 到 1 之间，可以知道当一条路径上的结点个数越多时，首尾两个用户之间的联系便会越弱，联系值便会越小。再由六度分割理论的介绍可知，任何两个用户之间的平均联系不超过 6 个人。由此，本文只计算结点个数小于 4 的路径上首尾结点的亲密值。

根据上述两种情况，提出社交网络中任意两个结点之间的亲密值公式如下：

$$\begin{cases} VC(u_i,u_j)=d(u_i,u_j)，若\ u_i\ 与\ u_j\ 直接连接 \\ VC(u_i,u_j)=\sum\left(\dfrac{l(u_i,u_k,\cdots,u_j)}{\sum l(u_i,u_k,\cdots,u_j)}\times d(u_i,u_k)\right)， \\ \quad 若\ u_i\ 与\ u_j\ 不直接连接，但可通过 \\ \quad 若干条路径间接连接 \\ VC(u_i,u_j)=0，\qquad u_i\ 与\ u_j\ 之间无连接关系 \end{cases} \tag{3}$$

其中，$VC(u_i,u_j)$ 表示任意两个用户之间的亲密值。如果两个用户之间有直接连接关系，则其亲密值就是他们之间的直接亲密值。如果两个用户之间不直接相连，但可通过路径间接

相连，那么其亲密值就是各条互不重合路径的上联系值乘以各个路径的比重，再求和即可。如果两个用户之间无连接关系，那么他们的亲密值为0；图2中，$VC(A,G)=d(A,D,G)+d(A,E,G)=0.8*0.5*(0.8/(0.8+0.6))+0.6*0.5(0.6/(0.8+0.6))=0.31$。

本文所提出的缓存替换算法是基于最近最久未使用替换算法(LRU)。当我们要计算缓存中每个对象的价值时，首先要得到每个缓存对象的LRU价值。缓存中每个缓存对象的LRU价值的计算公式如下：

$$V_{LRU}(P_{CB})=\frac{(M+1-P_{CB})}{\sum_{i=1}^{M} i} \tag{4}$$

其中，M表示缓存空间中的对象个数。P_{CB}在3.1节中提过，表示该缓存对象在缓存中的位置。$V_{LRU}(P_{CB})$表示位置为P_{CB}的缓存对象的LRU价值。当使用本文提出的缓存替换算法时，每次执行结束后最新的请求内容所在的缓存对象都会在缓存中的第一个位置。通过上述公式可以看出，缓存中的每个缓存对象都有一个LRU价值，LRU价值的大小介于0到1之间。缓存对象的位置值越小，表示在与当前时间间隔越短的时间内，该缓存对象被访问过，该缓存对象就相应获得一个越大的LRU价值。

缓存对象的价值计算公式如下：

$$V_{CB}=\frac{w_1*VC(u_T,u_K)+w_2*V_{LRU}(P_{CB})}{S_{CB}},w_1+w_2=1.0 \tag{5}$$

其中，$VC(u_T,u_K)$为当前用户与缓存对象中的内容的发布用户之间的亲密值，在云端服务器中计算得到后返回给移动终端，每个缓存块都有一个亲密值。$V_{LRU}(P_{CB})$是前文提到过的LRU价值，由缓存对象的位置值决定。V_{CB}就是这个缓存对象的价值，或者称为总价值，它的计算是由缓存对象的亲密值、LRU价值以及缓存对象占用空间的大小共同决定。w_1和w_2就是亲密值和LRU价值分别占的权重大小。w_1越大，表示更多根据缓存对象的亲密值决定缓存替换，w_2类似。S_{CB}为缓存对象占用存储空间的大小。缓存对象的价值由缓存内容的价值、最近访问时间以及占用的存储空间共同决定。

3.4 基于社交关系的移动缓存替换算法

基于社交关系的移动缓存替换算法的具体流程如图3所示。图中对于每个缓存对象，只标示出了其位置值P，占用存储空间大小S以及其价值V，其他信息未标出。当客户端接收到一个对象要写入缓存空间，且缓存剩余空间小于数据对象大小时，首先在缓存中查找是否存在，若缓存中存在，即命中，则将缓存中位置值小于命中对象的位置值都加1，将命中的缓存对象的位置值P变为1，移动到第一个位置。若缓存中未命中，则首先计算缓存中各个对象的价值，然后按照价值从小到大排序。再从小到大选择要移出存储空间的缓存对象，当价值最小的一个或若干个缓存对象的S值(即占用存储空间大小)之和恰好大于或等于新来的内容时，将这一个或多个缓存对象移出存储空间。然后将缓存中的各个对象再按照位置大小排序，并重新将位置值置为2到$n+1$(n为剩下的缓存对象个数)。然后将新的内容插入到位置1，并将其内容返回给用户，显示到屏幕上。

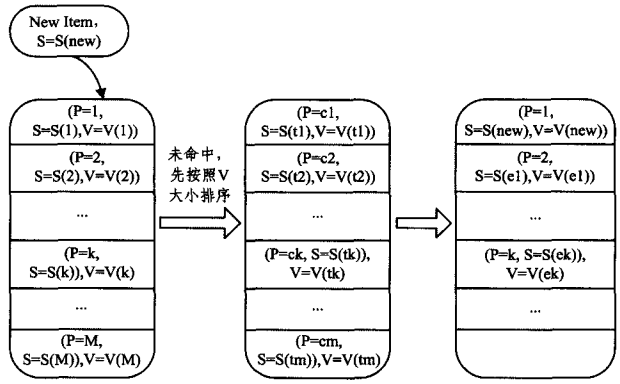

图3 缓存替换示意图

假设M表示缓存中已有对象的个数；S表示每个数据块占用的存储空间大小；缓存中已有对象的大小分别为$CB[1]$，$CB[2]$，…，$CB[M]$；请求返回的数据块为NB，其唯一标识为$NB.ID$，由用户id和页号组成来快速表示内容，返回的内容的请求用户与发布用户之间的亲密值大小是$NB.VC$，占用空间大小为$NB.S$。算法描述如下所示。

输入：请求返回的数据块NB
输出：执行缓存替换算法后的缓存空间
开始：

```
1. If(缓存中还有剩余空间，且剩余空间大小>NB. S)//缓存未满，插入即可
2. {
3.     将缓存中的各个对象后移一个位置，位置值都加1
4.     将新的内容及其与当前用户的亲密值NB. VC等信息封装成缓存对象，并插入到第1个位置
5. }
6. Else//缓存中已满
7. {
8.     将缓存中的对象按照价值从小到大排序
9.     CurrentSize=0. //设置当前大小
10.    For(i=1; i<=M; i++)//按照价值大小遍历缓存对象，当价值之和恰好大于新的内容时结束
11.    {
12.        CurrentSize=CurrentSize + CB[i]. S;
13.        If(CurrentSize >= NB. S))
14. {
15.        将1到i的缓存对象移出存储空间
16.            Break;
17.        }
18.    }
19.    将缓存中的对象按照位置值排序
20.    将缓存中的对象重新编号位置值为2到n //n为剩下的对象个数
21.    将新的内容及其与当前用户的亲密值NB. VC等信息封装成缓存对象，并插入到第1个位置
22. }
```

结束

4 实验设计与验证

为了验证上文提出的基于社交关系的移动缓存替换算法的效果，本节进行了模拟实验。

4.1 数据准备

我们使用数据堂网站提供的Twitter数据流，其中有

2400000 条数据(已经经过隐私处理),数据集中的一条数据由两个字符串构成,两个字符串代表两个用户的 id,前面的字符串为被关注者 id,后面的字符串为关注者 id,一条数据表示一个关注与被关注的关系。首先,使用 Java 程序读取所有数据,计算数据集中每个 id 有多少关注者,即被多少其他用户关注。可知一共有 19148 个用户,大概 10000 个以上的用户只有 10 个关注者,关注者太少不利于得到实验效果。最后选取了 2000 个用户,他们的关注者数量在 100 到 400 之间。

由文献[7]可知,移动用户对好友数据的请求符合 Zipf 定律,即用户查看其他人发布的内容时,大概 80%的浏览内容是由 20%的用户发布的。由于用户之间存在社交关系,我们可知这 20%的用户与当前用户关系最密切,即为亲密值最大的那些用户。

在亲密值设定时,我们参考文献[7]中类似的方法,根据两个用户共同的关注者和被关注者数量来设定直接亲密值。具体规则如下:若当前用户 u_i 和用户 u_j 互相关注,并且共同关注 10 个以上相同用户,或被 10 个以上相同的用户关注,则两者的直接亲密值 $d(u_i,u_j)=0.8$;若当前用户 u_i 和用户 u_j 互相关注,并且共同关注 5 到 10 个相同用户,或被 5 到 10 个相同的用户关注,则两者的直接亲密值 $d(u_i,u_j)=0.6$;若当前用户 u_i 和用户 u_j 互相关注,并且共同关注小于 5 个相同用户,或被少于 5 个相同的用户关注,则两者的直接亲密值 $d(u_i,u_j)=0.4$;若当前用户 u_i 关注用户 u_j,但是用户 u_j 未关注当前用户 u_i,且两个用户共同关注 10 个以上其他用户或被 10 个以上其他用户共同关注,则两者的直接亲密值 $d(u_i,u_j)=0.4$;若当前用户 u_i 关注用户 u_j,但是用户 u_j 未关注当前用户 u_i,且两个用户共同关注小于 10 个其他用户或被少于 10 个其他用户共同关注,则两者的直接亲密值 $d(u_i,u_j)=0.2$;其他情况的直接亲密值 $d(u_i,u_j)=0$。

在实验中,我们用一段 Matlab 代码按照 Zipf 分布生成了 10000 个随机数,则可知 10000 个数中有 80%的数字是 1 到 10000 这 10000 个数字中的 20%个数字。其次在每一个随机数后面随机加上一个 10 到 100 内的数,表示该内容所占用的存储空间的大小。然后将生成的 10000 条数据与选取的 2000 个 Twitter 数据集中的用户 id 相关联,即将出现频率越高的数字和与当前用户社交关系越近的用户相关联。因为出现频率越高,表示越关心这个好友,其就是与当前用户亲密值最高的用户。

4.2 实验设计及结果分析

实验所用的云端服务器是 VMware 软件中的虚拟机,配置为 2.4 GHz,16GB RAM,1 TB RAID5 Disk,操作系统采用 Centos6.2,网络链接采用 1Gbps 以太网光纤和交换机。在虚拟机中服务器使用的数据库为 MySQL 5.1,服务器上安装 Eclipse软件,使用 Java 语言编写程序来计算用户之间的亲密值,并且发送准备好的 10000 条模拟请求。移动终端使用魅族品牌的手机魅蓝 Note,配置为:MT6752 1.7GHz CPU,2GB RAM,32GB ROM。网络使用中国联通 3G 网络 WCDMA。手机端通过编写 App 进行测试。

实验中,设置存储空间的大小为 1000,表示能够存储 1000 条 Tweet。由 4.1 节可知每次请求占用的存储空间为 10 到 100,即每次查看的一页数据中有 10 到 100 条 Tweet,存储空间设置为 1000 比较符合现实场景。首先,实验设置用户亲密值的权重 w_1 为 0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0 时,则对应的 LRU 价值权重 w_2 为 1.0,0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1,0。实验先从 1000 个用户中选取 100 个用户,其中每个用户作为当前用户,他们的关注用户数大概在 200 左右。实验使用生成的与用户关联的 10000 条随机数作为模拟请求,存储空间为 1000,调用基于社交关系的移动缓存替换算法,在亲密值和 LRU 价值的权重不同的情况下,得到的实验结果如图 4 所示。

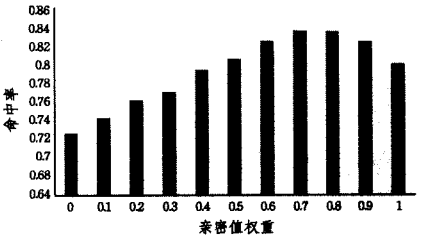

图 4 亲密值权重 w_1 取不同值时执行缓存替换算法后的命中率

图 4 的横坐标表示亲密值权重 w_1,其取值为 0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0 共 11 个值;纵坐标表示 w_1 取不同的值时缓存命中率的大小,由命中次数除以总数据量得出。由图表我们可知当把亲密值权重设置为 0.7 时,即 LRU 价值权重设置为 0.3 时,缓存命中率最高。可见在移动环境下,用户之间的社交关系对用户请求的影响更大,用户之间的亲密度影响着缓存的替换。当 w_1 大于 0.7 时,缓存的命中率又会出现小幅度下降,可知缓存内容的最近访问时间也依旧会对缓存的替换产生一定的影响。

基于上面实验得出的结果,下面选取 $w_1=0.7$,$w_2=0.3$ 来继续实验。

在这里,实验分别选取准备的 1000 个用户中的 100,200,300,400,500,600,700,800,900,1000 个来再次进行实验。同时,在同等条件下使用最近最久未使用缓存替换算法来进行对比实验,结果如图 5 所示。

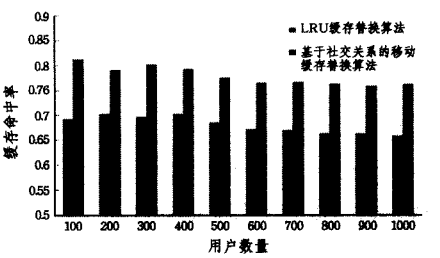

图 5 用户数量不同时 LRU 与基于社交关系的缓存替换算法的命中率对比

图 5 的横坐标表示用户数量,取 100,200,300,400,500,600,700,800,900,1000 共 10 组数据;纵坐标表示不同用户数量时的缓存命中率大小,由命中次数除以总数据量得出。在上图中,当数据量的规模较小时,实验结果会上下浮动,但是随着数据量的增大,命中率会趋于平稳。由实验结果可知,在存在社交关系的移动环境下,本文提出的基于社交关系的移动缓存替换算法的命中率明显高于传统的 LRU 缓存替换算法的命中率。

结束语 本文先介绍了在存在社交关系的移动环境下所遇到的一些问题来引出缓存技术,接着分析了已有的缓存替换算法及其优缺点,然后提出了一种基于社交关系的移动缓存替换算法并对其进行了详细的描述。通过模拟真实的使用场景进行实验验证,同经典缓存替换算法相比,本算法可以在

不降低系统性能和用户体验的基础上提高缓存命中率，合理使用缓存空间。下一步，我们将更深入研究移动社交网络中用户的行为对缓存的影响，以优化现有的缓存替换算法。

参 考 文 献

[1] Jain D K, Sharma S. Growth Rate of Cached Data Items at clients in Mobile Ad Hoc Networks[C]// 2014 IEEE Global Conference on Wireless Computing and Networking(GCWCN). 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] Ma Hong-yuan, Wang Bin. Query Results Caching and Prefetching in Web Search Engines Based on User Characteristics [J]. Journal of Chinese Information Processing, 2012, 26(6): 19-26 (in Chinese)
马宏远，王斌. 基于用户特性的搜索引擎查询结果缓存与预取[J]. 中文信息学报，2012，26(6)：19-26

[4] Yan Xia. Caching Based on Actual User Behavior [D]. Beijing: University of Science and Technology of China, 2011 (in Chinese)
夏琰. 基于实际用户行为分析的缓存研究[D]. 北京：中国科学技术大学，2011

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

[6] Kumar V R, Swati M. Cache Replacement Algorithms for Coordinated Cooperative Social Wireless Networks[J]. International Journal of Computer Science and Mobile Computing, 2014, 3(10): 718-725

[7] Liu Qian. Friend Recommendation Based on Social Network and Location Information [D]. Hangzhou: Zhejiang University, 2013 (in Chinese)
刘乾. 基于社交网络和地理位置信息的好友推荐方法研究[D]. 杭州：浙江大学，2013

[8] Branzei 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

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

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

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

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

[13] Leong H V, Si A. On Adaptive Caching in Mobile Databases [C]// Proceedings of the 1997 ACM Symposium on Applied Computing. 1997: 302-309

(上接第 31 页)

参 考 文 献

[1] Molloy E, Yang H, Browne J. Feature-based Modeling in Design for Assembly [J]. International Journal of Computer Integrated Manufacturing, 1993, 6(12): 119-125

[2] Wang Jun-feng, Li Shi-qi, Liu Ji-hong, et al. Computer Aided Assembly Planning: a Survey [J]. Journal of Engineering Graphics, 2005, 26(2): 1-6(in Chinese)
王俊峰，李世其，刘继红，等. 计算机辅助装配规划研究综述[J]. 工程图学学报，2005，26(2)：1-6

[3] Uma R N, Wein J, Williamson D P. On the Relationship between Combinatorial and LP-Based Lower Bounds for NP-hard Scheduling Problems [J]. Theoretical Computer Science, 2006, 361(2): 241-256

[4] Fazio D T, Whitney D E. Simplified Generation of All Mechanical Assembly Sequences [J]. IEEE Journal Robotics and Automation, 1987, 3(6): 640-658

[5] de Mello L S H, Sanderson A C. A Correct and Complete Algorithm for Mechanical Assembly Sequences [J]. IEEE Transaction on Robotics and Automation, 1991, 7(2): 228-240

[6] Bryant R E. Symbolic Boolean Manipulation with Ordered Binary Decision Diagram [J]. ACM Computing Surverys, 1992, 24(3): 293-318

[7] Xu Zhou-bo, Gu Tian-long, Zhao Ling-zhong. A Novel Symbolic ADD Algorithm for Maximum Flow in Networks [J]. Journal on Communications, 2005, 26(2): 1-8(in Chinese)
徐周波，古天龙，赵岭忠. 网络最大流问题的一种新的符号 ADD 求解算法 [J]. 通信学报，2005，26(2)：1-8

[8] Luiz S, Homen D M, Sanderson A C. AND/OR Graph Representation of Assembly Plans [J]. IEEE Transactions on Robotics and Automation, 1990, 6(2): 188-198

[9] Gu Tian-long, Liu Hua-dong. Symbolic OBDD-based technique for generating assembly sequences [J]. Computer Integrated Manufacturing Systems, 2008, 14(2): 321-328(in Chinese)
古天龙，刘华东. 基于符号有序二叉决策图的装配序列生成技术[J]. 计算机集成制造系统，2008，14(2)：321-328

[10] Minato S. Zero-suppressed BDDs and Their Applications [J]. International Journal on Software Tools for Technology Transfer, 2001, 3(2): 156-170

[11] Li Feng-ying, Gu Tian-long, Chang Liang, et al. Timed Petri and ZBDD Based Approach for Assembly Sequence Planning [J]. Computer Science, 2012, 39(2): 170-174(in Chinese)
李凤英，古天龙，常亮，等. 一种基于赋时 Petri 网和 ZBDD 的装配序列规划方法 [J]. 计算机科学，2012，39(2)：170-174

[12] Gu Tian-long, Xu Zhou-bo, Yang Zhi-fei. Symbolic OBDD Representations for Mechanical Assembly Sequences [J]. Computer-Aided Design, 2008, 40(4): 411-421

[13] Hu Min, Wang Yan-wei, Nie Bin, et al. Assembly Sequence Planning Based on Contact-Relation Analysis [J]. Journal of Computer-Aided Design & Computer Graphics, 2014, 26(8): 1374-1384(in Chinese)
胡敏，王彦伟，聂斌，等. 基于接触关系分析的装配序列规划[J]. 计算机辅助设计与图形学学报，2014，26(8)：1374-1384

[14] Homem D E, Mello L S, Sanderson A C. Representation of Mechanical Assembly Sequence [J]. IEEE Transactions on Robotics and Automation, 1991, 7(2): 211-227

[15] Gottipolu R B, Ghost K. A Simplified and Efficient Representation for Evaluation and Selection of Assembly Sequences [J]. Computer in Industry, 2003, 50(2): 251-264

[16] Gottipolu R B, Ghosh K. An Integrated Approach to the Generation of Assembly Sequences [J]. International Journal of Computer Application in Technology, 1995, 8(3): 125-138