计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20130056
51(10):2255-2269,2014

云计算环境下基于协同过滤的个性化推荐机制

朱 夏 宋爱波 东 方 罗军舟
(东南大学计算机科学与工程学院 南京 211189)
(xzhu@seu.edu.cn)

A Collaborative Filtering Recommendation Mechanism for Cloud Computing

Zhu Xia, Song Aibo, Dong Fang, and Luo Junzhou
(*School of Computer Science and Engineering, Southeast University, Nanjing* 211189)

Abstract The advent of cloud computing has witnessed a sharp augmentation in the amount of application data, the result of which gives more and more prominence to a personalized recommendation technique, which has been used by different kinds of Web applications. However, traditional collaborative filtering (CF) techniques, when applied to cloud computing which features a huge scale and distributed processing architecture, are confronted with some problems, such as low recommendation accuracy, long recommendation time and high network traffic. Consequently, the performance of recommendation mechanisms degrades drastically. To address this issue, a CF recommendation mechanism for cloud computing (RAC) is proposed in this paper. RAC first employs a candidate neighbor, the definition of which will also be given, to screen those items which exert great influence on the recommendation results. Then a 2-stage rating indexing structure based on distributed storage system will be constructed to ensure the quick and precise location of CN by RAC. On this basis, a CN-based CF recommendation algorithm CN-DCFA which searches k nearest neighbors between candidate neighbors can be provided. Meanwhile, CN-DCFA utilizes similarity to predict the recommendation results top-N of active users. Experiments have showed that our recommendation mechanism RAC, enjoying great efficiency and exactitude, is worthy of recommendation.

Key words collaborative filtering; personalized recommendation; cloud computing; candidate neighbor; distributed rating management

摘 要 随着云计算时代的到来,应用数据量剧增,个性化推荐技术日趋重要.然而由于云计算的超大规模以及分布式处理架构等特点,将传统的推荐技术直接应用到云计算环境时会面临推荐精度低、推荐时延长以及网络开销大等问题,导致推荐性能急剧下降.针对上述问题,提出一种云计算环境下基于协同过滤的个性化推荐机制RAC.该机制首先制定分布式评分管理策略,通过定义候选邻居(candidate neighbor, CN)的概念筛选对推荐结果影响较大的项目集,并构建基于分布式存储系统的2个阶段评分索引,保证推荐机制快速准确地定位候选邻居;在此基础上提出基于候选邻居的协同过滤推荐算法(candidate neighbor-based distribited collaborative filtering algorithm, CN-DCFA),在候选邻居中搜索

收稿日期:2013-01-14;**修回日期**:2013-12-23
基金项目:国家"九七三"重点基础研究发展计划基金项目(2010CB328104);国家自然科学基金项目(61202449,61272054,61370207,61320106007);国家"八六三"高技术研究发展计划基金项目(2013AA013503);国家科技支撑计划基金项目(2010BAI88B03,2011BAK21B02);高等学校博士学科点专项科研基金项目(20110092130002);江苏省产学研前瞻性联合研究项目(BY2012202);江苏省科技成果转化专项资金项目(BA2012036);江苏省网络与信息安全重点实验室资助项目(BM2003201);计算机网络和信息集成教育部重点实验室资助项目(93K-9)

目标用户已评分项目的 k 近邻,预测目标用户的推荐集 top-N. 实验结果表明,在云计算环境下 RAC 拥有良好的推荐精度和推荐效率.

关键词 协同过滤;个性化推荐;云计算;候选邻居;分布式评分管理

中图法分类号 TP393

互联网规模的迅速增长带来了信息超载问题,用户很难从海量数据中获取有价值的信息,信息使用效率急剧下降. 个性化推荐作为一种重要的信息过滤手段,通过分析用户的兴趣爱好和历史行为,主动向用户推荐其存在潜在兴趣的项目(item),有效解决互联网的信息超载问题[1]. 协同过滤(collaborative filtering, CF)推荐是迄今为止最成功的个性化推荐技术之一,其基本思想是搜索与目标用户(active user)兴趣相似的邻居用户,并将邻居感兴趣的项目推荐给目标用户[2]. 目前协同过滤推荐已经成为电子商务、社交网络和视频分享等互联网应用的核心技术[3-4].

由于云计算[5]的超大规模以及分布式处理架构等特点,部署到云计算环境下的互联网应用在数据存储位置、数据存储容量以及应用规模等方面均区别于传统模式,因此将传统的协同过滤推荐算法直接应用到云计算环境会产生以下诸多问题:

1) 推荐精度问题. 传统的协同过滤推荐算法将用户对项目的评分数据集中存储在服务器端的数据库中,推荐算法通过计算用户间的相似度获取目标用户的 k 近邻,并根据 k 近邻的访问记录产生推荐结果. 然而在云计算环境下,云服务供应商为了提高数据的并行处理效率和 I/O 性能,构建基于无共享(shared nothing, SN)架构的数据中心. 在 SN 架构下,数据中心的每个节点都拥有独立的本地存储,数据被分布式存储到各个节点[6]并行处理. 在 SN 架构下根据存储系统的数据分块策略互联网应用的项目可能会被分散存储到各个节点,用户访问项目后对其给出的评分数据也会相应分散存储在各个节点,传统的集中式协同过滤推荐算法在本地节点计算用户相似度时可能会失去部分具有推荐价值的邻居,从而导致算法的推荐精度降低.

2) 推荐时延问题. 选择云计算平台作为其业务运营环境的互联网企业通常具有较大的应用规模,业务运营过程中将产生海量的评分数据. 传统的协同过滤推荐算法在处理该类应用时,频繁遍历评分矩阵将导致推荐时延急剧提升. 与此同时,评分数据在节点间的传输时延会进一步加剧推荐算法的高时延问题,这显然无法满足应用的实时推荐需求,目前 Amazon 和京东商城等电子商务应用已经需要根据用户浏览和购买行为给出实时或近实时的推荐结果.

3) 网络开销问题. 由于用户兴趣会随着时间动态变化,因此传统的协同过滤推荐算法为了提高推荐精度,需要定期更新相似度矩阵,尤其是基于用户的协同过滤推荐算法(user-based CF)的更新频率更高. 如前文所述,在云计算环境下评分数据是分散存储的,各个存储节点与推荐节点间将频繁交换评分数据,跨节点甚至跨数据中心的数据传输将占用大量数据中心的网络带宽,增加数据中心的网络开销.

综上所述,将传统的协同过滤推荐算法直接应用到云计算环境时会面临推荐精度低、推荐时延高以及网络开销大等问题,推荐算法的性能难以得到保证. 为了解决上述问题,本文提出一种云计算环境下基于协同过滤的个性化推荐机制(RAC). 该机制首先制定评分数据在数据中心内的分布式管理策略,通过定义候选邻居的概念,筛选出对推荐结果较为重要的项目集合. 在此基础上构建 2 个阶段评分索引结构,将评分数据及其元数据分布式存储到数据中心,保证推荐机制能够准确快速地查询候选邻居. 最后设计基于候选邻居的协同过滤推荐算法(candidate neighbor-based distributed collaborative filtering algorithm, CN-DCFA),在候选邻居中搜索目标用户已评分项目的 k 近邻,预测目标用户的推荐集 top-N,该算法基于评分筛选和增量更新的思想极大降低了算法的推荐时延.

1 相关工作

目前分布式推荐机制大多遵循以下思想:通过在网络间传输评分向量,分布式计算每个用户的邻居,最终在网络上构建代表邻居关系的覆盖网络. Tveit[7]提出基于泛洪(flooding)策略的推荐算法,目标用户向网络中全体用户传输评分向量计算邻居,该算法的优点是简单易部署,但是缺点同样明显,泛洪策略会产生巨大的网络开销,覆盖网络不易实时更新,且推荐算法的时延较高. 针对上述缺点很多研究者提出了改进方案. Miller 等人[8]为每个用

户分别构建唯一的描述项目相似度的覆盖网络,通过随机获取用户评分向量的方法更新覆盖网络,该方法能够显著降低更新覆盖网络的频率和网络开销. Kim 等人[9]提出在覆盖网络更新阶段用面向邻居传输评分向量的方法替代文献[7]的全局泛洪策略,同样能够有效控制网络开销. Zhen 等人[10]提出以用户为中心逐层向周边邻居扩散更新覆盖网络的方法,直至到达预先设定的搜索深度阈值. Bakker 等人[11]提出在覆盖网络的初始化阶段即采用随机选择的策略,进一步降低初始化构建覆盖网络时所需的大规模网络开销. Ormándi 等人[12]提出将用户邻居的评分向量存储在本地以保证覆盖网络的负载均衡. Xie 等人[13-14]利用 Hash 机制将对同一资源评分相同的评分记录索引到同一节点,用户仅向对同一资源作出相同评价的用户传输评分向量,简化相似度计算过程,降低网络开销. 由分析可知,上述研究均通过在覆盖网络的构建或更新阶段减少评分向量的传输对象,以达到降低网络开销的目的,但是该策略并未预先考虑用户在推荐过程中的作用,从而误删大量具有高推荐价值的用户,推荐精度难以达到理想值.

Berkvosky 等人[15]提出按照项目所属类别划分评分数据集,并将子数据集部署在多个节点并行计算,最终在节点间聚合推荐结果,该方法在处理倾斜数据集时会面临负载均衡问题. Liu 等人[16]利用聚类(cluster)思想搜索用户邻居,算法逐一将覆盖网络中权值最大的边加入集合,当集合中出现回路时将涉及的所有节点归入同一类,并根据该轮聚类结果选定中心点,利用 K-means 算法进行二次聚类. 由于聚类算法的前提是需要预先计算边的权值,即用户相似度,因此算法不容易动态更新.

部分学者提出了针对特定网络环境的分布式推荐机制. Bickson 等人[17]设计了面向社交网络应用的分布式推荐机制,通过构造效益函数(cost function)在用户的原始评分和邻居产生的推荐评分之间寻找均衡点,利用高斯-马尔科夫随机场(Gauss-Markov random field, GMRF)模型预测整个网络达到均衡状态后各个用户对项目的评分. Höhfeld 等人[18]提出了面向多用户虚拟环境(multi-user virtual environments, MMVE)的协同过滤推荐机制,针对 MMVE 中参加同一虚拟会议的用户通常具有共同兴趣的特点,推荐机制设置用户在参加虚拟会议时交换评分向量并更新记录用户邻居和待推荐项目的列表. Wang 等人[19-20]利用先验概率模型描述用户与项目之间的相关程度,推荐算法基于增量更新的思想,假设某时刻先验概率已知,当推荐系统内任意项目被访问后,并且另一项目出现在访问该项目的用户的已访问列表中,推荐算法随即增加 2 个项目相关的先验概率. 上述研究提出的推荐机制仅能应用于特定领域,算法适用范围较窄,并且推荐模型过于复杂,可扩展性不高.

2　云计算环境的协同过滤推荐机制

针对传统推荐算法在云计算环境中面临的问题,本文提出云计算环境下基于协同过滤的个性化推荐机制 RAC,面向部署在数据中心的各类互联网应用提供高效准确的推荐服务. 本节首先介绍云计算环境下的协同过滤推荐模型,描述推荐机制 RAC 各模块间的交互关系,然后本文给出所使用相关术语的定义.

2.1　模型描述

云计算环境下的协同过滤推荐模型按功能可划分为 4 层,具体如图 1 所示:

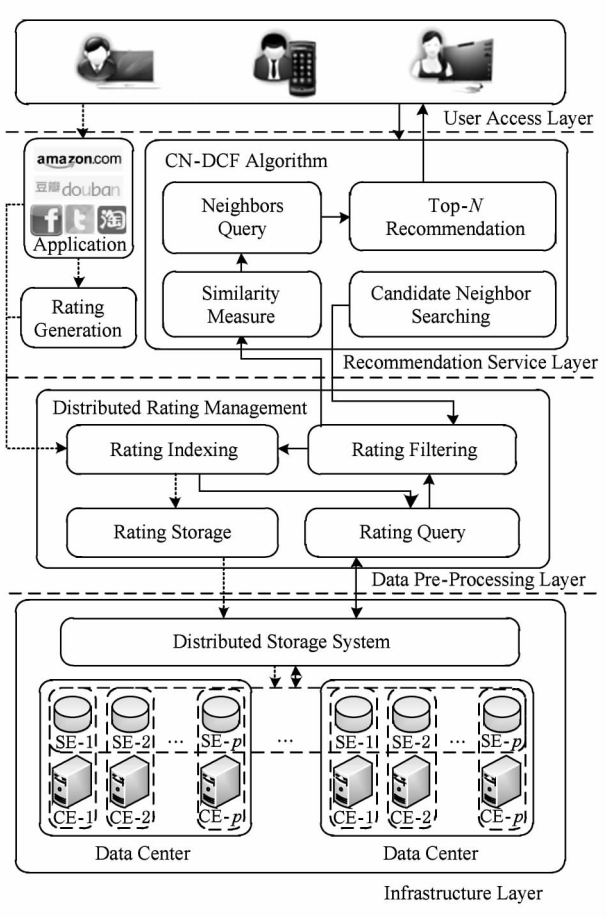


Fig. 1　CF-based recommendation model for cloud computing.

图 1　云计算环境下的协同过滤推荐模型

1) 基础设施层(infrastr-ucture layer)由基于SN架构的数据中心组成,数据中心的各个节点都拥有独立的计算单元(computing element, CE)和存储单元(storage element, SE),其中SE利用分布式存储系统Dynamo统一管理,并向CE提供一致的数据存储和查询接口;数据中心的所有CE均部署本文提出的推荐算法和数据索引,负责计算存储到本地SE的评分数据,并根据存储和查询接口向其他CE发送结果;

2)数据预处理层(data pre-processing layer)负责管理用户的评分数据,与传统的简单利用集中式数据库管理评分的方法不同,本文提出分布式评分管理机制,通过设计评分过滤模块,筛选对推荐过程最有价值的评分数据,同时基于分布式存储系统设计评分索引结构,提供评分数据在云计算环境下的分布式存储和查询功能;

3)推荐服务层(recommendation service layer)负责响应用户触发的推荐请求,本文针对部署在云计算环境下的互联网应用的特点,提出一种基于候选邻居的协同过滤推荐算法CN-DCFA,在尽量减少使用历史数据的前提下,通过计算项目间的相似度预测目标用户对项目的评分并产生推荐集;

4)用户接入层(user access layer)负责面向终端用户提供推荐机制的泛在接入.

2.2 基本定义

下面给出本文所使用术语的相关定义.

定义1. 推荐模型定义为四元组形式:

$$S=\langle U,V,N,\boldsymbol{R}_{m\times n}\rangle,$$

其中:

$$U=\{u_1,u_2,\cdots,u_m\},$$
$$V=\{v_1,v_2,\cdots,v_n\},$$
$$N=\{n_1,n_2,\cdots,n_p\},$$
$$\boldsymbol{R}_{m\times n}=\{r_{i,j}\mid 1\leqslant i\leqslant m,1\leqslant j\leqslant n\},$$

U和V分别表示参与推荐的用户和项目集合,N表示数据中心的存储节点集合,$\boldsymbol{R}_{m\times n}$表示用户-项目评分矩阵.

定义2. 评分记录定义为三元组形式:

$$rec_{ij}=\langle u_i,v_j,r_{i,j}\rangle,$$

其中,$u_i\in U,v_j\in V,r_{i,j}\in\boldsymbol{R}_{m\times n}$,$r_{i,j}$表示用户$u_i$访问项目$v_j$后给出的评分.

定义3. 用户u_i的评分向量定义为

$$\boldsymbol{Y}_i=(r_{i,1},r_{i,2},\cdots,r_{i,n}),$$

其中,$\forall v_j\notin V_i,r_{i,j}=\varnothing$,$V_i$表示用户$u_i$的已评分项目集合.

定义4. 项目v_j的评分向量定义为

$$\boldsymbol{W}_j=(r_{1,j},r_{2,j},\cdots,r_{m,j}),$$

其中,$\forall u_i\notin U_j,r_{i,j}=\varnothing$,$U_j$表示项目$v_j$的已评分用户集合.

3 分布式评分管理

分布式评分管理是推荐机制RAC数据预处理层的核心部分,其通过集成评分过滤、评分索引、评分存储和评分查询等模块将分散在各个存储节点的数据处理成推荐算法能够直接使用的评分.分布式评分管理无缝衔接了基础设施层和推荐服务层.本节首先定义候选邻居的概念,过滤对推荐结果影响较大的评分.然后构建两阶段评分索引以提供快速准确的候选邻居定位服务.最后基于上述技术设计评分存储和查询机制.

3.1 评分过滤

传统推荐算法通常采用基于用户的协同过滤推荐思想,计算用户间的相似度获取目标用户的k近邻.由于推荐算法在计算用户相似度时需要将分布在数据中心各个节点的用户评分传输至推荐节点,因此单次推荐过程可能需要传输绝大多数评分数据.本文考虑到在云计算环境下不同用户对同一项目的评分通常存储在同一节点,如果选择计算项目间的相似度,那么在推荐过程中仅需传输少量评分即可完成推荐.与此同时,在电子商务等应用领域,相比较用户而言,项目间的相似性更不易发生变化,推荐算法无需频繁计算项目相似度.基于此,本文在协同过滤推荐过程中选择衡量项目间的相似性,通过计算项目间的相似度寻找目标用户已评分项目的k近邻,最终对k近邻加权产生推荐结果[21].

推荐算法在计算项目相似度时,如果2个项目分别存储于不同节点,那么推荐算法需要将其中一方的评分向量传输至另一方.单次推荐过程至多需要传输$n(n-1)m/2$条评分记录,对于用户和项目数量较大的应用而言,所需传输的评分数据量依然过大.然而我们观察发现,如果2个项目获得所有用户的评价均相差甚远,那么在推荐算法中这2个项目被判定为邻居的可能性很低.例如在电子商务应用中如果2件商品间具有相似性,那么必然存在一个用户群体对2件商品的评价是接近的.换言之,在用户样本空间足够大的前提下,项目v_t成为项目v_j的邻居的必要条件是:至少存在一个用户u_i,其对v_j和v_t的评分$r_{i,j}$和$r_{i,t}$在数值上邻近.本文将满足上述必要条件的项目集合称为v_j的候选邻居,下面给出候选邻居的定义.

定义 5. 对于项目 v_j,如果存在项目 v_t 满足以下条件:$\forall u_i \in U_j, 1) u_i \in U_t; 2) |r_{i,j} - r_{i,t}| < \varepsilon$,其中 U_j 和 U_t 分别表示项目 v_j 和 v_t 的已评分用户集合,ε 表示评分邻近阈值且根据不同评分数据集动态调整,那么将满足上述条件的项目集合定义为项目 v_j 的候选邻居(candidate neighbor),记作:

$$CN(v_j) = \{v_t | (u_i, v_t, r_{i,t}), u_i \in U_j, |r_{i,j} - r_{i,t}| < \varepsilon\}.$$

候选邻居概念的提出缩小了参与推荐的项目范围,通过调整 ε 的取值能够过滤大多数对本次推荐过程无效的评分.推荐算法不再需要遍历整个项目空间,仅需计算项目与其候选邻居间的相似度,从候选邻居中筛选出 k 近邻,降低评分传输次数和存储节点间的网络负载.

3.2　两阶段评分索引

由于项目的候选邻居可能分布式存储在多个节点,推荐算法需要将项目的评分向量传输至其候选邻居所在节点,因此如何快速准确地定位候选邻居的存储位置将成为影响推荐效率的关键.本文基于分布式存储系统 Dynamo,提出 2 个阶段评分索引,分别提取评分记录 rec_{ij} 的元数据和主键,将评分数据及其存储信息分布式存储到数据中心,并提供候选邻居的定位搜索服务.

我们首先将候选邻居的定义转化成存储系统能够识别的查询请求格式.假设目标用户 u_a 对项目 v_j 的评分是 $r_{a,j}$,如果推荐算法需要获取项目 v_j 的候选邻居,那么提交至存储系统的查询请求如下:

$$REQ = (rec_{it} | u_i = u_a, r_{a,j} - \varepsilon < r_{i,t} < r_{a,j} + \varepsilon). \quad (1)$$

通过分析式(1),REQ 是针对用户维度和评分维度的二维查询(u 和 r),同时 REQ 在评分维度上是区间查询.考虑到在 Dynamo 中评分记录只能选择某一维度作为数据在存储系统中的主键,并且 Dynamo 的一致性 Hash 思想破坏了主键顺序,数值邻近的主键 Hash 后所得标识符往往相差甚远,因此 Dynamo 无法提供对诸如 REQ 的二维查询和区间查询的支持.基于此,本文将评分索引过程分为 2 个阶段,分别索引评分记录的主键和存储位置信息,将查询请求 REQ 索引成 Dynamo 能够识别的查询主键.

在评分索引的第 1 阶段,本文选择项目维度 v_j 作为评分记录 $rec_{ij} = (u_i, v_j, r_{i,j})$ 的主键,记作:

$$key(rec_{ij}) = v_j. \quad (2)$$

选择项目维度作为评分记录的主键可以保证与同一项目相关的评分记录均存储在同一节点,进一步降低计算项目间相似度时所需的评分传输开销.

在评分索引的第 2 阶段,本文提出一种新的评分索引结构,提取评分记录的存储位置信息的主键,根据主键将位置信息存储到数据中心.为了能够快速准确地定位候选邻居在数据中心内的存储位置,本文通过构建基于用户维度 u_i 和评分维度 $r_{i,j}$ 的二维评分索引结构 *R-Indexing*,提取项目维度 v_j 的主键,并将 v_j 作为评分记录 rec_{ij} 的位置信息分布式存储到数据中心的各个节点,记作:

$$key(v_j) = R\text{-}Indexing(u_i, r_{i,j}), \quad (3)$$

其中 *R-Indexing* 是针对用户维度和评分维度的二维评分索引结构,存储系统通过构建 *R-Indexing* 支持搜索候选邻居的查询请求 REQ,下面给出 *R-Indexing* 的构建过程.

本文首先通过定义逐行扫描-空间填充曲线(progressive space filling curve, P-SFC)将二维向量 $(u_i, r_{i,j})$ 索引成一维向量.空间填充曲线(SFC)[22]是一种简单有效的降维思想,其通过对多维数据近似排序构建一维索引,目前在高维数据库等领域获得了广泛应用.本文针对 REQ 的查询模式特点和评分记录的数据结构,设计适合推荐机制 RAC 的逐行扫描曲线 P-SFC,保证在评分查询过程中没有冗余结果.

P-SFC 的构建过程如下:如图 2 所示,将二维空间的横坐标定义为评分维度,纵坐标定义为用户维度,坐标轴的最大值分别取 $[0, 2^{\alpha}-1]$ 和 $[0, 2^{\beta}-1]$,其中 α 和 β 分别根据评分区间和用户个数取值.例如假设评分的取值范围是 0～5,那么取 $\alpha = 3$ 就可以将所有评分包括在二维空间中.二维空间中数据均被转化成二进制形式 $(r_{i,j}^{(2)}, u_i^{(2)})$.由于 REQ 仅针对评分维度是区间查询,用户维度依然是精确查询,因此在对二维数据排序时优先将纵坐标相同的数据排在一起.根据这一原则将用户维度 u_i 置于评分维

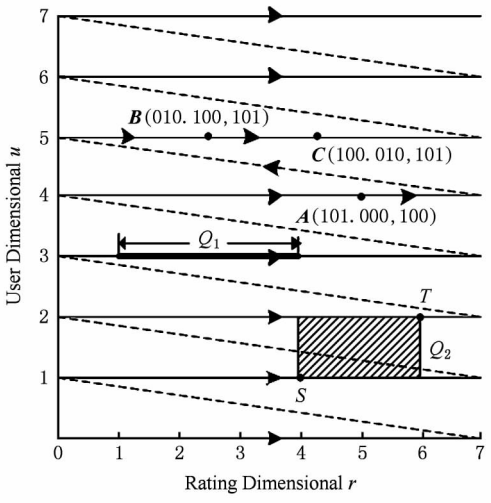


Fig. 2　Progressive space filling curve.

图 2　逐行扫描-空间填充曲线

度 $r_{i,j}$ 之前，将所得数据 $u_i^{(2)}r_{i,j}^{(2)}$ 作为二维向量 $(r_{i,j}^{(2)}, u_i^{(2)})$ 的一维索引 id，记作：

$$id(r_{i,j}^{(2)}, u_i^{(2)}) = u_i^{(2)}r_{i,j}^{(2)}. \quad (4)$$

举例说明：假设推荐机制面向 8 个用户，且用户对项目的评分区间是 0～5，评分的小数位数取 1，那么取 $\alpha=3$，$\beta=3$ 即可满足推荐机制需求. 假设推荐机制中存在 3 条评分记录：$rec_{42}=(u_4, v_2, 5)$，$rec_{53}=(u_5, v_3, 2.5)$，$rec_{54}=(u_5, v_4, 4.3)$. 根据式(3)，我们提取评分记录的评分维度和用户维度得到 3 个二维向量 $\boldsymbol{A}(5.0,4)$，$\boldsymbol{B}(2.5,5)$和 $\boldsymbol{C}(4.3,5)$，在图 2 所示的二维空间中，向量间显然无法比较大小关系. 根据式(4)，我们将 3 个二维向量转换成一维索引形式：

$$id(\boldsymbol{A}) = id(101.000,100) = 100101000;$$
$$id(\boldsymbol{B}) = id(010.100,101) = 101010100;$$
$$id(\boldsymbol{C}) = id(100.010,101) = 101100010.$$

那么根据一维索引的大小关系为 $100101000 < 101010100 < 101100010$，对 $\boldsymbol{A}$，$\boldsymbol{B}$ 和 $\boldsymbol{C}$ 进行排序：$\boldsymbol{A}<\boldsymbol{B}<\boldsymbol{C}$. 如果对二维空间内所有向量均采用上述方法进行排序，那么我们将顺序连接而成的曲线称为二维空间的逐行扫描曲线 P-SFC. P-SFC 在处理部分查询请求时可能存在冗余结果，例如图 2 中的 Q_2，在根据 P-SFC 转换成一维查询后将返回曲线上从点 S 到点 T 的所有值，在实际查询结果的基础上增加了无效结果. 但是由于本文所需的搜索候选邻居的查询请求 REQ 关于纵坐标是精确查询，关于横坐标是区间查询，如 Q_1 表示查询用户 3，评分属于 1～4 之间的所有评分，Q_1 与 P-SFC 曲线完全重合，因此利用 P-SFC 处理 REQ 时所得结果全部是有效结果.

鉴于 Dynamo 的一致性 Hash 可能会破坏一维索引值域区间内的数值顺序，以一维索引 id 为主键的存储系统无法支持 REQ 在评分维度上的区间查询. 然而我们通过观察发现，数值临近的二进制数具有共同前缀的概率往往较高，因此我们考虑将区间 $[\alpha,\beta]$ 划分成 $n+1$ 个范围较小的子区间 $[\delta_0,\delta_1][\delta_1,\delta_2]\cdots[\delta_n,\delta_{n+1}]$（其中 $\alpha=\delta_0$，$\beta=\delta_{n+1}$），属于区间 $[\delta_i,\delta_{i+1}]$ 的一维索引选择区间上下界的共同前缀 $p_i=Prefix(\delta_i,\delta_{i+1})$ 作为主键. 数据查询过程根据查询区间与子区间的交集将区间查询转化成关于 p_i 的精确查询. 遵循上述思路，我们给出前缀二叉树[23]的定义.

定义 6. 前缀二叉树(prefix binary tree, PBT)定义为一棵深度动态变化的二叉树，包括 3 种节点：根节点 $root$、有效节点 $value$ 和分支节点 $branch$. 将 PBT 中度为 0 的节点(叶节点)命名为有效节点，除根节点和有效节点外的其他节点命名为分支节点. 根节点和分支节点有且仅有 2 个子节点. PBT 的每个节点都用二进制数标记，记作 $label$，其中根节点 $label$ 为空，左、右子节点 $label$ 分别等于其父节点 $label$ 增加 0 和 1，记作：

$$label_{root} = \text{null};$$
$$label_{left\text{-}child} = label_{parent}0;$$
$$label_{right\text{-}child} = label_{parent}1.$$

图 3 是一棵深度为 5 的 PBT，其中有效节点集合 $S=\{0000,0001,001,010,0110,0111,100,101,1110,1111\}$.

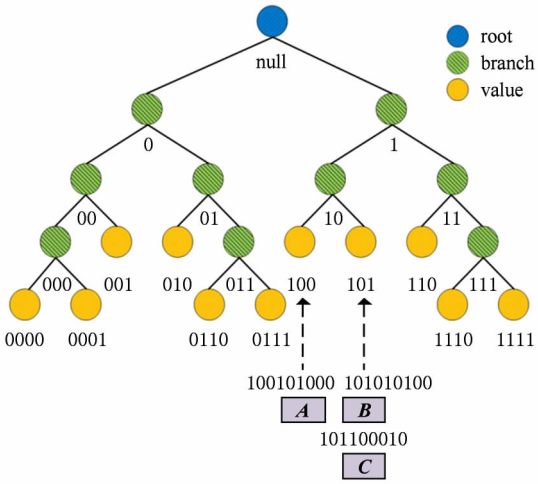


Fig. 3 PBT with $depth=5$.

图 3 深度为 5 的前缀二叉树

本文利用 P-SFC 和 PBT 构建评分记录的评分索引 $R\text{-}Indexing$，其过程如下：首先利用 P-SFC 将评分维度和用户维度索引成一维向量 id，在此基础上通过为数据中心的所有存储节点共同定义一棵 PBT，将 id 与 PBT 的有效节点集合 S 作前缀匹配操作，根据 PBT 性质 1 和性质 2，有且仅有一个有效节点 $label$ 是 id 的前缀，那么将该 $label$ 定义为用户维度和评分维度索引后的值，记作：

$$R\text{-}Indexing(u_i, r_{i,j}) = Prefix(u_i r_{i,j}, label_{value}). \quad (5)$$

3.3 评分存储

如 3.2 节所述，推荐机制在为评分记录构建评分索引后，将评分记录及其存储位置信息存储到数据中心，以支持候选邻居的查询请求 REQ，具体过程见规则 1.

规则 1. 评分记录 $rec_{ij}=(u_i, v_j, r_{i,j})$ 在数据中心中的存储过程如下：

1) 选择项目维度 v_j 作为评分记录 rec_{ij} 的主键,并将 rec_{ij} 提交至 Dynamo,存储系统根据一致性 Hash 机制将 rec_{ij} 存储到相应节点.

2) 选择项目维度 v_j 作为评分记录的存储位置,并且选择用户维度和评分维度的二维评分索引 $R\text{-}Indexing(u_i,r_{i,j})$ 作为位置存储信息 v_j 的主键,将 v_j 提交至 Dynamo,存储系统根据一致性 Hash 机制将 v_j 存储到相应节点.

仍以 3.2 节所述的评分记录为例,$\boldsymbol{A}$ 的一维索引 $id(\boldsymbol{A})=100101000$,在图 3 所示的 PBT 中,有效节点 100 是 100101 的前缀,因此推荐机制选择 100 作为二维向量 $\boldsymbol{A}(5.0,4)$ 的评分索引值,记作:$R\text{-}Indexing(u_4,5)=100$. 评分记录在数据中心中的存储过程如下:对于评分记录 $rec_{42}=(u_4,v_2,5)$ 而言,$key(rec_{42})=v_2=010$,$key(v_2)=R\text{-}Indexing(u_4,5)=100$,评分记录 rec_{42} 和其存储位置信息 v_2 分别以 010 和 100 为主键加入存储系统 Dynamo. 同理可计算 rec_{53} 和 rec_{54} 的索引,$R\text{-}Indexing(u_5,2.5)=101$,$R\text{-}Indexing(u_5,4.3)=101$. 可得 $key(rec_{53})=v_3=011$,$key(v_3)=R\text{-}Indexing(u_5,2.5)=101$;$key(rec_{54})=v_4=100$,$key(v_4)=R\text{-}Indexing(u_5,4.3)=101$.

3.4　评分查询

评分查询的功能是在数据中心的节点间搜索候选邻居. 考虑到在协同过滤推荐过程中只有目标用户的已评分项目才会对推荐结果产生影响,因此本文只搜索已评分项目的 k 近邻,将目标用户的任一已评分项目的评分向量传输给属于该项目候选邻居与目标用户未评分项目的交集的全体项目,记作:

$$\boldsymbol{W}_j(\forall j,v_j\in V_a)\rightarrow v_t\in(CN(v_j)\bigcap(V-V_a)). \tag{6}$$

由于在大多数应用中目标用户通常只访问或评价过少量项目,未评分项目集合 $V\text{-}V_a$ 近似等于集合 V,集合 $CN(v_j)\bigcap(V\text{-}V_a)$ 近似等于 $CN(v_j)$,因此我们将式(6)简化成:

$$\boldsymbol{W}_j(\forall j,v_j\in V_a)\rightarrow v_t\in(CN(v_j)). \tag{7}$$

我们首先计算目标用户所有已评分项目的候选邻居. 由式(1)可知搜索候选邻居的查询请求 REQ 是针对项目维度和评分维度的二维区间查询,本文在将 REQ 提交至 Dynamo 前先对 REQ 进行预处理,在将 REQ 转换成能够被 Dynamo 所识别的一维精确查询后,再将 REQ 提交到数据中心查询评分记录,具体过程见规则 2.

规则 2. $\forall v_j\in V_a$,搜索 v_j 的候选邻居 $CN(v_j)$ 的过程如下:

1) 针对 v_j 的评分向量 $\boldsymbol{W}_j=(r_{1,j},r_{2,j},\cdots,r_{m,j})$,其中 $\forall u_i\in U_j,r_{i,j}\neq\varnothing$,将相应 $u_i,r_{i,j}-\varepsilon$ 和 $r_{i,j}+\varepsilon$ 转换成 n 位二进制编码 $u_i^{(2)},(r_{i,j}-\varepsilon)^{(2)}$ 和 $(r_{i,j}+\varepsilon)^{(2)}$. 根据 P-SFC 的编码规则,将 REQ 转换成查询下界和上届分别为 $u_i^{(2)}(r_{i,j}-\varepsilon)^{(2)}$ 和 $u_i^{(2)}(r_{i,j}+\varepsilon)^{(2)}$ 的一维区间查询.

2) 计算 $u_i^{(2)}(r_{i,j}-\varepsilon)^{(2)}$ 和 $u_i^{(2)}(r_{i,j}+\varepsilon)^{(2)}$ 的共同前缀 $prefix^{(2)}$,并判定 $prefix^{(2)}$ 与 PBT 节点间的关系:如果 $prefix^{(2)}$ 是 PBT 的有效节点或以有效节点为前缀,则将该有效节点作为 REQ 的查询键值 key_1;如果 $prefix^{(2)}$ 是 PBT 的中间节点,则将该中间节点的 q 个子女有效节点作为 REQ 的查询键值 $key_s(0<s\leqslant q)$.

3) 将 q 个查询键值 key_s 分别提交至 Dynamo,存储系统返回的查询结果 $res(key_s)(0<s\leqslant q)$ 是 REQ 拟查询的评分记录的存储位置.

4) 将 $res(key_s)$ 作为查询键值提交至 Dynamo 进行二次查询,存储系统返回的查询结果 $res(res(key_s))(0<s\leqslant q)$ 是满足 REQ 的所有评分记录.

5) 将在查询所得的评分记录中出现的全体项目加入候选邻居集合 $CN(v_j)$,记作:

$$CN(v_j)=\{v_t\mid(u_i,v_t,r_{i,t})\in res(res(key_s)),0<s\leqslant q\}. \tag{8}$$

如图 4 所示,假设数据中心拥有 5 个存储节点:N_0,N_2,N_5,N_8 和 N_{11},仍然以 rec_{42},rec_{53} 和 rec_{54} 为例,上述评分记录注册加入数据中心的过程如下:由 3.3 节可知,$key(rec_{42})=v_2=010$,$key(v_2)=100$;

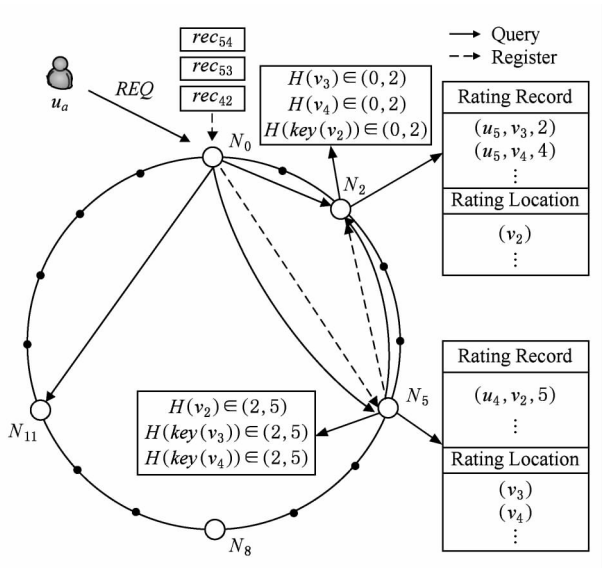


Fig. 4　Storage and query process of rating records.

图 4　评分记录的存储和查询过程

$key(rec_{53})=v_3=011$, $key(v_3)=101$; $key(rec_{54})=v_4=100$, $key(v_4)=101$. 假设存储系统采用的 Hash 函数有: $H(010)\in(2,5)$, $H(100)\in(0,2)$, $H(011)\in(0,2)$, $H(101)\in(2,5)$, 那么 rec_{42} 以 010 为主键存储到 N_5, 同时 rec_{42} 的存储位置 v_2 以 100 为主键存储到 N_2; rec_{53} 以 011 为主键存储到 N_2, 同时 rec_{53} 的存储位置 v_3 以 101 为主键存储到 N_5; rec_{54} 以 100 为主键存储到 N_2, 同时 rec_{54} 的存储位置 v_4 以 101 为主键存储到 N_5.

假设目标用户 u_a 的已评分项目集合 V_a 中包括项目 v_1, 那么根据规则 2, 推荐机制需要搜索 v_1 的候选邻居. 假设存在评分记录 $(u_5, v_1, 4.1)$, 同时取 $\varepsilon=0.3$, 那么 rec_{54} 是满足候选邻居定义的评分记录. 在推荐机制中实际查询过程如下: 针对区间查询 [3.8, 4.4] 和 u_5, 将其转换成一维区间查询 [101011110, 101100011], 取共同前缀为 101, 在如图 3 所示的 PBT 中, 101 是有效节点, 根据前文假设的 Hash 函数, $H(101)\in(2,5)$, 在节点 N_5 上必定能够查询到 rec_{54} 的存储信息 v_4, 由 $H(100)\in(0,2)$, 在 v_2 上可以查询到 rec_{54}, 推荐机制将判定 v_4 是 v_1 的候选邻居.

4 基于候选邻居的协同过滤推荐算法

鉴于项目间的相似性不易发生变化, 本文在推荐项目时基于增量更新的思想, 通过构建一个预先计算好的兴趣度模型维护项目相似度, 生成推荐结果时直接使用模型拟合的数据. 建模的初始化阶段通过在各存储节点间传输评分计算项目间的相似度, 建模完成后, 模型利用每一次推荐过程更新相似度数据. CN-DCFA 在保证推荐精度的前提下无需重复计算相似度, 显著降低了推荐时延和评分传输开销. 在获取目标用户已评分项目的候选邻居的基础上, 我们对项目间的邻居关系进行建模. CN-DCFA 通过记录与每个项目相似度最近的 k 个邻居, 构建代表项目兴趣的覆盖网络. 为了避免增加额外的计算和传输开销, 我们充分利用推荐算法的每一次推荐过程初始化或更新兴趣度模型, 具体过程见算法 1.

算法 1. 基于候选邻居的协同过滤推荐算法 CN-DCFA.

输入: 任意资源 v_j 的评分向量 $\boldsymbol{N}_j$;

输出: 推荐集 top-N.

1) 在构建兴趣度模型前的初始状态下, 推荐算法为所有项目 v_j 都维护一个 n 维邻居向量 $\boldsymbol{N}_j=(0,0,\cdots,0)$, 并设置 $\boldsymbol{N}_j$ 的时间戳为 T_j.

2) 推荐算法的任意一次推荐过程均可能对模型产生影响. 假设推荐算法启动针对用户 u_a 的一次推荐过程, 那么 $\forall v_j\in I_a$, 首先判定本轮推荐过程所处的建模阶段:

① 如果 $\boldsymbol{N}_j$ 中非零的维数小于 k, 那么判定本轮推荐过程属于模型初始化阶段, 转至步骤 3).

② 如果 $\boldsymbol{N}_j$ 中非零的维数等于 k, 且 $T-T_j\leqslant\delta$, 其中 T 是当前时间, δ 是预先设定的模型更新时间间隔, 那么判定模型属于更新阶段且 $\boldsymbol{N}_j$ 的时间戳有效, 本轮推荐过程不需要更新模型的相似度, 直接使用 $\boldsymbol{N}_j$ 计算推荐结果, 转至步骤 6).

③ 如果 $\boldsymbol{N}_j$ 中非零的维数等于 k, 且 $T-T_j>\delta$, 那么判定本轮推荐过程属于模型更新阶段且 $\boldsymbol{N}_j$ 的时间戳已失效, 推荐过程将重新计算相似度并更新 $\boldsymbol{N}_j$, 转至步骤 3).

3) 根据规则 2 的计算结果, 将 v_j 的评分向量 $\boldsymbol{W}_j$ 顺序传输给 v_j 的候选邻居, 计算 v_j 与候选邻居的相似度:

$$sim(v_j,v_t)=\frac{\sum\limits_{u_i\in U_{jt}}(r_{i,j}-\bar{r}_j)(r_{i,t}-\bar{r}_t)}{\sqrt{\sum\limits_{u_i\in U_{jt}}(r_{i,j}-\bar{r}_j)^2}\sqrt{\sum\limits_{u_i\in U_{jt}}(r_{i,t}-\bar{r}_t)^2}},\tag{9}$$

其中, $v_t\in CN(v_j)$, U_{jt} 表示同时评价过项目 v_j 和 v_t 的用户集合, $\bar{r}_j$ 和 $\bar{r}_t$ 分别表示 v_j 和 v_t 上评分的平均值.

4) 如果 $\boldsymbol{N}_j$ 中非零的维数小于 k, 将 $\boldsymbol{N}_j$ 的时间戳 T_j 设置为当前时间, 并将 $\boldsymbol{N}_j$ 的第 t 维赋值为 $sim(v_j,v_t)$. 将 $\boldsymbol{N}_j$ 转发至下一个候选邻居, 转至步骤 3). 重复上述过程, 直至转发过程结束, 转至步骤 6).

5) 如果 $\boldsymbol{N}_j$ 中非零的维数大于或等于 k, 通过比较 $sim(v_j,v_t)$ 与 $\boldsymbol{N}_j$ 各维中最小值 $\min(a_{j1}, a_{j2},\cdots,a_{jn})$: 如果 $sim(v_j,v_t)\leqslant\min(a_{j1},a_{j2},\cdots,a_{jn})$, 则将 $\boldsymbol{N}_j$ 转发至下一候选邻居, 转至步骤 3); 如果 $sim(v_j,v_t)>\min(a_{j1},a_{j2},\cdots,a_{jn})$, 则将 $\boldsymbol{N}_j$ 的第 t 维的值设置为 $sim(v_j,v_t)$, 原先拥有最小值的维度设置为 0, 同时修改 v_j 的时间戳 T_j 为当前时间, 并将 $\boldsymbol{N}_j$ 转发至下一个候选邻居所属节点, 转至步骤 3). 重复上述过程, 直至转发过程结束, 转至步骤 6).

6) 预测目标用户 u_a 对未评分项目的评分 $Pr_{a,j}$:

$$Pr_{a,j}=\bar{r}_a+\frac{\boldsymbol{N}_j\cdot(\boldsymbol{Y}_a-\boldsymbol{Y}_{\text{avg}})}{\sum_{v_t\in kNN(v_j)}sim(v_j,v_t)}=\bar{r}_a+\frac{\sum_{v_t\in kNN(v_j)}sim(v_j,v_t)(r_{a,t}-\bar{r}_t)}{\sum_{v_t\in kNN(v_j)}sim(v_j,v_t)},\quad(10)$$

其中,$\bar{r}_a$ 是 u_a 评分的平均值,$\bar{r}_t$ 是针对 v_t 评分的平均值,$kNN(v_j)$是由 $\boldsymbol{N}_j$ 的非零维对应的项目集合,$\boldsymbol{Y}_{\text{avg}}$是记录各个项目评分的平均值的 n 维向量.

7) 在未评分项目中选择 $Pr_{a,j}$最大的 N 个项目组成推荐集 top-N.

5　仿真实验与分析

5.1　实验设置

本文选择 overlay 仿真工具 PlanetSim[24] 模拟基于 SN 架构的数据中心环境,并在 PlanetSim 上添加 Beehive 路由算法,以此模拟评分数据在 Dynamo 中的传输过程. 在此基础上本文选择 Apache Mahout[25] 作为基础推荐引擎,该开源项目已经集成了协同过滤、聚类和分类等多种推荐算法,本文在 Mahout 的基础上开发部署 RAC,并选择 Booking-Crossing Dataset[26] 作为实验的测试数据集.

5.2　评估标准

本文选择推荐精度、推荐时延和网络开销作为评估推荐算法质量的度量标准. 其中在推荐精度方面本文选择平均绝对误差(mean absolute error),记作:

$$MAE=\frac{1}{\|A\|}\sum_{a\in A}\frac{\sum_{j\in B}|r_{a,j}-p_{a,j}|}{\|B\|},\quad(11)$$

其中,$r_{a,j}$是用户 u_a 对项目 v_j 的实际评分,$p_{a,j}$是预测评分,A 和 B 分别表示目标用户集合和测试集.

在推荐时延方面将推荐分为 2 部分:

$$T=T_{\text{trs}}+T_{\text{rec}},\quad(12)$$

其中,T_{trs}和 T_{rec}分别表示评分转发时延和协同过滤推荐时延,本文用查询请求在存储节点间的转发跳数(hops)作为衡量转发时延的标准.

在网络开销方面,由于在分布式环境下参与协同过滤推荐的项目或用户数量会直接影响数据中心的网络开销,因此本文选择参与推荐的项目数量(number of item fetched)或用户数量(number of user fetched)作为衡量推荐算法的网络开销的标准,分别记作:NIF 和 NUF,NIF 和 NUF 的取值越小表明推荐算法所造成的网络开销越低.

5.3　参数分析

本节通过调节 RAC 的候选邻居数量、兴趣度模型更新间隔以及模型规模等参数优化 RAC 的 MAE 和 NIF. 实验中训练集的项目数量分别设置为 500,2500 和 5 000,并从测试集中分别选择 500 个用户作为目标用户.

实验 1. 候选邻居数量变化时的 MAE.

选择 All-but-one 作为目标用户的预测评分选取模式,比较 RAC 在候选邻居的邻近阈值 ε 变化时 MAE 的变化情况. 如图 5 所示,随着 ε 逐渐增大,MAE 逐渐降低,本文分析这是由于当 ε 取值较小时,RAC 会损失部分能够对推荐结果产生影响的邻居项目. 而当 $\varepsilon>0.7$ 后,MAE 的变化趋于稳定,当 $\varepsilon>1.1$ 后,MAE 几乎不再变化,该情况表明候选邻居数量的改变并不会一直影响推荐精度,当 ε 达到某一阈值时推荐精度将不再变化. 本文分析认为,当 ε 取值属于区间 $A=[0.7,1.4]$时,RAC 的推荐精度能够达到稳定状态.

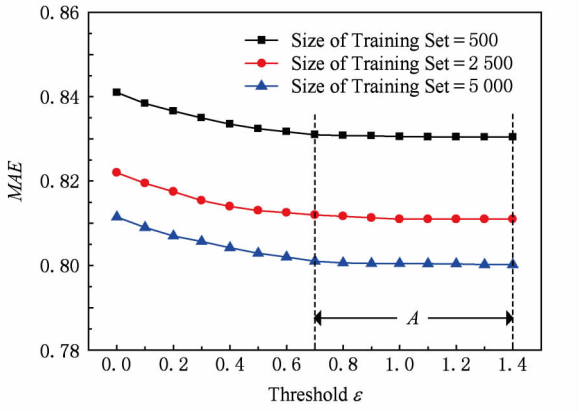


Fig. 5　Comparison of MAE with different threshold values ε (All-but-one).

图 5　参数 ε 变化时的平均绝对误差比较(All-but-one)

实验 2. 候选邻居数量变化时的 NIF.

选择 All-but-one 作为目标用户的预测评分选取模式,比较 RAC 在候选邻居的邻近阈值 ε 变化时 NIF 的变化情况. 如图 6 所示,随着 ε 的逐渐增大,参与推荐的项目数量缓慢增加,当 $\varepsilon>0.8$ 后,项目数量的增长速度明显增快,而当 $\varepsilon>1.8$ 后,NIF 已经趋近于项目总数. 本文分析认为,当 ε 属于区间 $B=[0,0.8]$时,RAC 能够有效过滤无效评分,参与协同过滤推荐的项目数较少,系统可扩展性较好.

综合实验 1 和实验 2,随着实验逐渐扩大候选邻居的范围,推荐精度会相应提高,但是参与推荐的项目数量也会相应增加. 本文通过分析认为,针对数

据集 Booking-Crossing，当 $\varepsilon=0.7$ 时，RAC 能够同时取得较优的推荐精度和可扩展性.

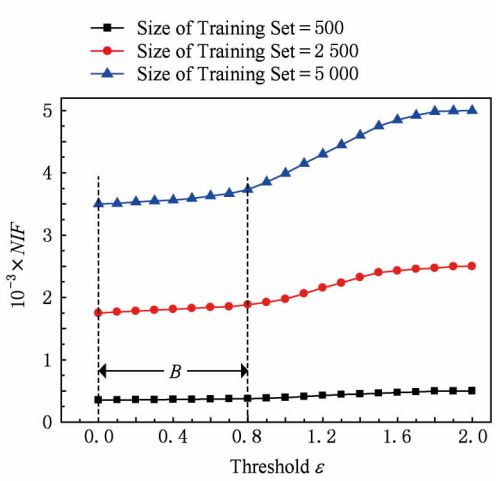


Fig. 6 Comparison of *NIF* with different threshold values ε (All-but-one).

图 6 参数 ε 变化时的参与项目数比较(All-but-one)

实验 3. 模型更新间隔变化时的 *MAE*.

对 Booking-Crossing 数据集 4 周内的数据按照时间排序，假设推荐过程到达服从泊松分布，并利用推荐过程的产生次数表示模型更新间隔 δ，比较 δ 变化时 RAC 的 *MAE* 变化情况. 如图 7 所示，当 $\delta<500$ 时，*MAE* 变化缓慢，表明在该阶段项目间的相似度较为稳定. 而当 δ 从 500 开始，*MAE* 曲线缓慢升高，本文分析出现该现象的原因是项目存在新的访问者，根据式(9)计算所得的项目相似度存在动态变化的可能性. 综合推荐精度结果本文选取模型更新间隔 $\delta=500$. 在该阶段 RAC 具有较好的推荐精度，同时能将模型更新开销降至较低范围.

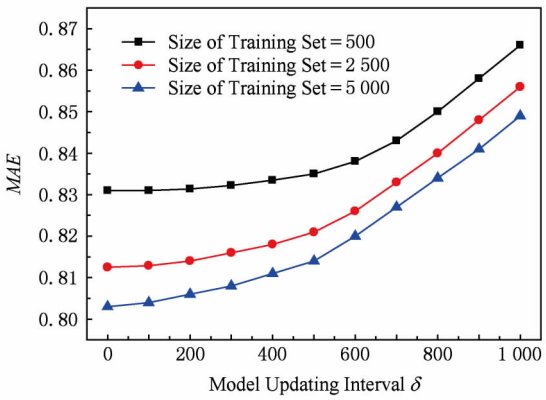


Fig. 7 Comparison of *MAE* with different threshold values δ (All-but-one).

图 7 参数 δ 变化时的平均绝对误差比较(All-but-one)

实验 4. 模型规模变化时的 *MAE*.

本文分别选择 All-but-one 和 Given-5 作为目标用户的预测评分选取模式，利用项目的邻居数量 k 表示兴趣度模型的规模，对 k 分别取值 10，20，30，40 和 50，比较当邻居数量 k 变化时 RAC 的 *MAE* 变化情况. 如图 8 所示，在 2 种模式下，随着 k 的取值变化 *MAE* 的变化并不明显，本文取 $k=20$.

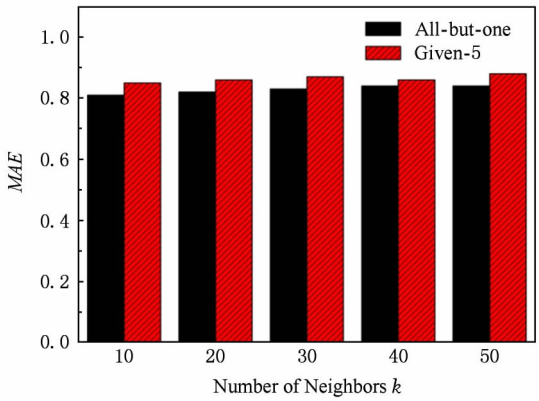


Fig. 8 Comparison of *MAE* with different threshold *k*.

图 8 参数 k 变化时的平均绝对误差比较

5.4 与其他算法的性能比较分析

本节将 RAC($\varepsilon=0.7$，$\delta=500$，$k=20$)与 5 种推荐算法进行比较，这 5 种算法是：PCC (Pearson correlation coefficient)[3]，DCF-BCA[9]，DRKS[10]，DCFLA[14]，DCF-FSS[20]，测试比较各算法在推荐精度、推荐时延和网络开销方面的性能.

实验 5. 不同推荐算法间的推荐精度比较.

如图 9 所示，在 All-but-one 模式下，RAC 的推荐精度明显优于其他 5 种算法，其中 PCC 的推荐精度最低. DCFLA 由于评分筛选机制过于严格，导致其失去多数具有高推荐价值的邻居，而本文引入的候选邻居概念能够有效解决该问题. DRKS 是上述机制中推荐精度与 RAC 最接近的，但是随着训练集的数量逐渐增大，DRKS 与 RAC 的差距亦

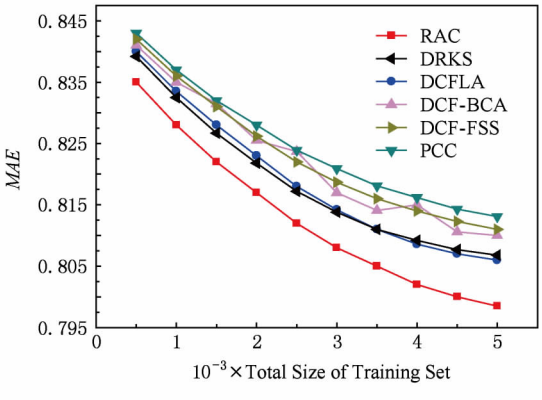


Fig. 9 Comparison of *MAE* (All-but-one).

图 9 推荐精度的比较(All-but-one)

逐渐增大,本文分析出现该状况的原因是因为DRKS采用的不公平扩散机制随着节点数的增多计算邻居的误差会相应增加.如图10所示,在Given 5模式下,RAC仍然最优,但是差距缩小,这表明目标用户的已评分项目数量会显著影响RAC的推荐精度.

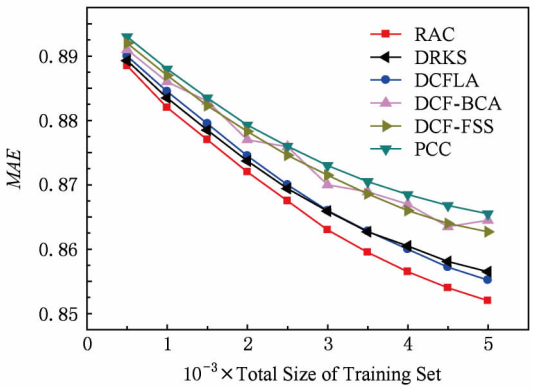


Fig. 10　Comparison of *MAE* (Given 5).

图10　推荐精度的比较(Given 5)

实验6.　不同推荐算法间的网络开销比较.

由于Booking-Crossing数据集的用户和项目数量大致相等,因此实验能够横向比较RAC的*NIF*和其他算法的*NUF*.考虑到RAC的项目兴趣度模型不需要在每次推荐过程中都重新计算相似度,因此设置实验以500次推荐过程为周期统计6种推荐算法参与的项目(用户)数量.如图11、图12所示,RAC和DCFLA的网络开销明显优于其他算法,其原因是由于RAC和DCFLA均对项目(用户)进行了初步筛选,有效降低了网络开销.RAC的*NIF*略高于DCFLA的*NUF*,但是RAC计算相似度的频率远低于DCFLA,因此RAC的网络开销仍优于其他算法.

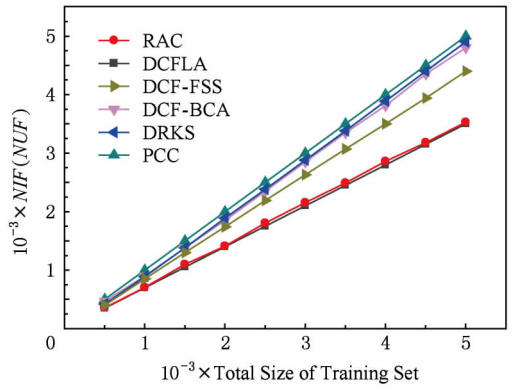


Fig. 11　Comparison of *NIF* (*NUF*) (All-but-one).

图11　网络开销的比较(All-but-one)

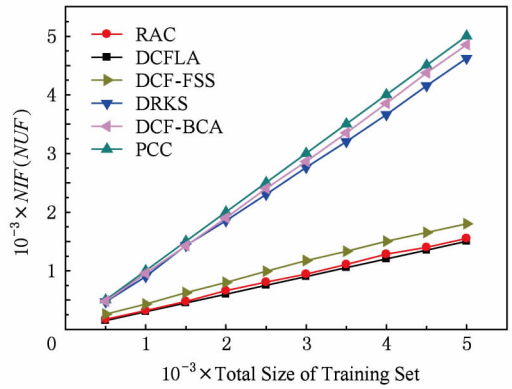


Fig. 12　Comparison of *NIF* (*NUF*) (Given 5).

图12　网络开销的比较(Given 5)

实验7.　不同推荐算法间的推荐时延比较.

由于DCF-FSS的推荐时延完全取决于用户访问项目的频率,只能在实际应用中进行测量,因此本次实验不考虑DCF-FSS.实验首先比较各算法的评分转发时延T_{trs}.如图13所示,RAC的性能最优,转发跳数随着存储节点数的增加没有显著变化,这是由于Dynamo的Beehive算法的Finger Table通过维护全局邻居信息提高查询效率,而DCF-BCA由于基于泛洪算法,性能最差.

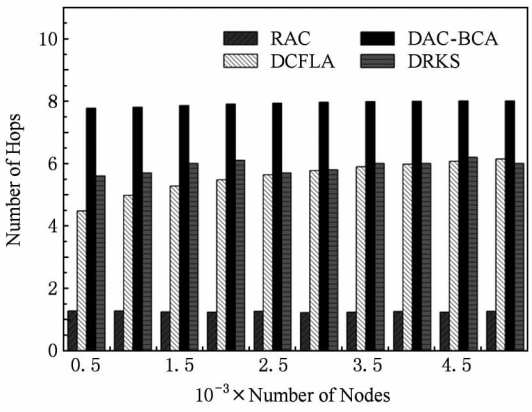


Fig. 13　Comparison of number of hops.

图13　推荐算法的转发跳数比较

如图14所示,RAC计算top-N步骤所需时延T_1远低于其他算法,甚至加上计算k近邻所需时延T_2仍然低于其他算法,而在真实系统中RAC并不需要频繁重复计算k近邻.本文分析出现该情况的原因是评分数据集中普遍存在热点项目,即用户访问相同项目的概率远高于两个项目获得相同用户访问的概率.当数据较为稀疏时,RAC的候选邻居策略能够过滤大量未获得相同用户评价的2个项目相似度的计算过程.

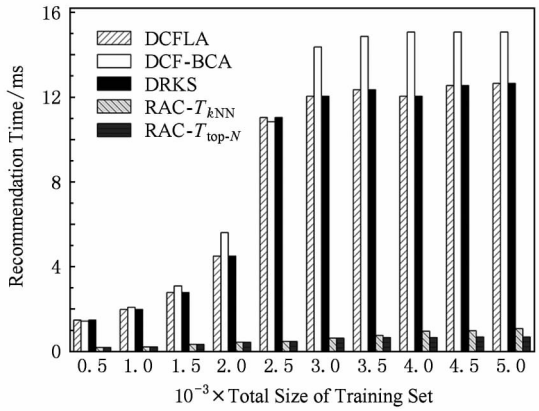


Fig. 14 Comparison of CF recommendation time.

图 14 推荐算法的协同过滤推荐时延比较

6 云计算环境实验分析

本节在真实的云计算环境下测试推荐算法的性能. 如图 15 所示,实验环境的节点来源分为 3 部分: 1)东南大学高性能计算中心(申请了 50 个节点); 2)小型集群(32 个节点);3)公有云服务(向 Amazon EC2 申请 10 个虚拟机节点). 限于实验环境的资源规模,实验将上述节点近似看成分布在不同地理位置的 3 个数据中心. 实验环境的网络连接情况如下: 数据中心 1 以及数据中心 2 内部的节点之间使用千兆以太网交换机连接,理论传输速度为 128 MBps;数据中心 1 与数据中心 2 之间通过教育网(CERNET)连接,实际测试传输速度峰值约为 4 MBps,数据中心 1 和数据中心 2 的节点通过 CERNET 连入 Internet 与数据中心 3 相连,实际测试传输速度峰值接近 1 MBps.

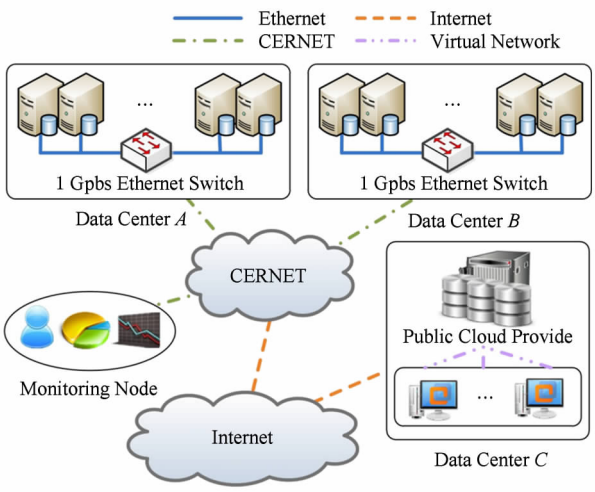


Fig. 15 Topology of experiment environments.

图 15 实验环境拓扑图

数据中心 1 的部分节点预先部署了 Dynamo 的开源实现 Cassandra[27],Cassandra 是用于处理结构化数据的分布式数据库系统,数据集仍采用 Booking-Crossing,并将全部数据加入训练集. 实验选择在第 5 节仿真实验中与 RAC 性能最接近的 DRKS 和 DCFLA 作为算法参照,其中 DRKS 搜索深度设置为 5.

根据目前数据中心的组成方式,设置 2 种实验环境:单数据中心(single data center)环境和跨数据中心(crossing data centers)环境,其中单数据中心环境由数据中心 1 的 50 个节点组成,跨数据中心环境分别来自数据中心 1 的 20 个节点、数据中心 2 的 20 个节点以及数据中心 3 的 10 个节点. 分别在 2 种实验环境下测试算法性能.

实验 8. 2 种数据中心环境下各推荐算法的推荐时延比较.

通过增加推荐请求数,测试各算法在面临真实的并发用户请求时推荐时延的变化. 图 16 中的实线表示 3 种算法在单数据中心环境下的推荐时延情况. DCFLA 和 DRKS 的推荐时延随着推荐请求数量的增加近似于线性增长,这是因为随着发起推荐的用户增加,推荐算法计算 k 近邻需要遍历大多数用户,推荐时延有所增加. 而 RAC 的推荐时延始终维持在 1 s 左右,并且在实验后半段增长速度明显降低,主要原因是由于 RAC 经过初始化阶段后已经拥有了相对固定的相似度矩阵,算法不需要在每一次推荐过程中都重新计算相似度.

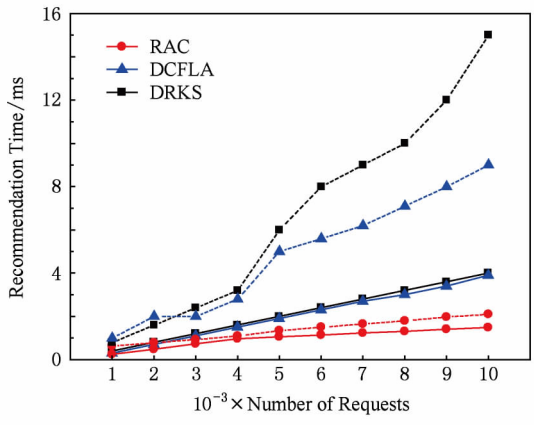


Fig. 16 Comparison of recommendation time.

图 16 推荐时延比较

图 16 所示的虚线部分代表 3 种算法在跨数据中心环境下的推荐时延情况. 其中 DRKS 的曲线上升趋势最为明显,本文分析出现该情况的原因是

DRKS 的深度优先机制需要遍历多数节点,在网络带宽有限的跨数据中心环境下推荐时延必然会急剧增加. DCFLA 和 RAC 的推荐时延在初期增长幅度均不大,但当推荐请求数高于 4 000 后,DCFLA 的推荐时延显著提升,并和 RAC 的差距逐渐增大.

实验 9. 2 种数据中心环境下各推荐算法的网络开销比较.

仿真实验使用 $NUF(NIF)$衡量网络开销,但是在真实的网络环境下,由于带宽分配的不均衡,$NUF(NIF)$不再适用于真实网络环境,因此本节选择网络的带宽利用率(bandwidth utilization)来衡量算法的网络开销. 如图 17 所示的实线部分,RAC 和 DCFLA 变化相对平缓,DRKS 在处理约 10 000 个并发请求时占用了约 50%的网络带宽. 由于数据中心 1)仅采用了带宽为 1 Gbps 的千兆以太网,如果选择带宽更高的 Infiniband 或光纤连接数据中心的节点,推荐算法所占用的网络带宽将更低.

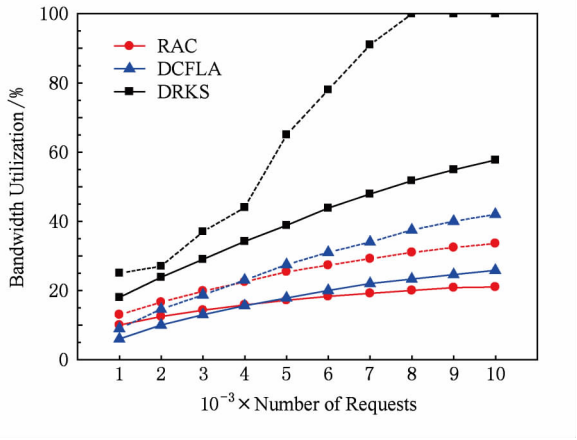


Fig. 17　Comparison of bandwidth utilization.

图 17　带宽利用率比较

如图 17 所示的虚线部分,DRKS 的带宽利用率上升趋势显著,当并发推荐请求数达到 8 000 后网络几乎全部被占用,推荐算法到达瓶颈状态. RAC 和 DCFLA 的网络开销也明显上升,RAC 在初期所占带宽高于 DCFLA,但是随着推荐请求的增加,RAC 的增长趋势逐渐平缓,但是 DCFLA 仍然显著上升.

实验 10. 2 种数据中心环境下各算法的推荐精度比较.

如图 18 所示的实线部分,在单数据中心环境下,3 种推荐算法的推荐精度变化并不明显,这表明在带宽足够的情况下推荐精度能够得到保证. 但是如虚线部分所示,在跨数据中心环境下,3 种推荐算法的推荐精度均有所下降,其中 RAC 的变化幅度是最小的,本文分析这主要是由于 RAC 所采用的一致性 Hash 机制能够保证查询邻居的准确性,而 DRKS 的推荐精度随着推荐请求的增加剧烈震荡,类似泛洪的查询机制会导致查询请求丢失,进而影响推荐精度.

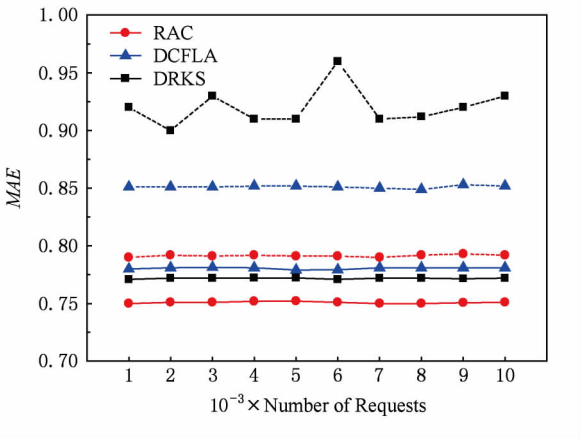


Fig. 18　Comparison of *MAE*.

图 18　推荐精度比较

7　结　　论

本文提出一种云计算环境下基于协同过滤的个性化推荐机制 RAC,解决传统推荐算法应用到云计算环境下面临的评分缺失,推荐时延长以及网络负载重等问题. 本文首先提出候选邻居的概念,筛选对本次推荐过程影响较大的项目集合;其次基于分布式存储系统 Dynamo 构建两阶段评分索引,将评分数据及其元数据分布式存储到数据中心,保证推荐算法能够准确快速的定位候选邻居;最后提出基于候选邻居的协同过滤推荐算法 CN-DCFA. 实验结果表明,在数据中心环境下 RAC 拥有良好的推荐精度和推荐效率.

参　考　文　献

[1] Tunkelang D. Recommendations as a conversation with the user [C] //Proc of the 5th ACM Conf on Recommender Systems. New York: ACM, 2011: 11-12

[2] Breese J, Heckerman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering [C] //Proc of the 14th Conf on Uncertainty in Artificial Intelligence. San Francisco: Morgan Kaufmann, 1998: 43-52

[3] Adomavicius G, Tuzhilin A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions [J]. IEEE Trans on Knowledge and Data Engineering, 2005, 17(6): 734-749

[4] Manzato M, Goularte R. A multimedia recommender system based on enriched user profiles [C] //Proc of the 27th Annual ACM Symp on Applied Computing. New York: ACM, 2012: 975-980

[5] Mell P, Grance T. The NIST definition of cloud computing, SP800-145 [R]. Gaithersburg: National Institute of Standards and Technology, 2011

[6] Ghemawat S, Gobioff H, Leung S. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[7] Tveit A. Peer-to-peer based recommendations for mobile commerce [C] //Proc of the 1st Int Workshop on Mobile Commerce. New York: ACM, 2001: 26-29

[8] Miller B, Konstan J, Riedl J. PocketLens: Toward a personal recommender system [J]. ACM Trans on Information Systems, 2004, 22(3): 437-476

[9] Kim B M, Li Qing, Howe A E, et al. Collaborative web agent based on friend network [J]. Applied Artificial Intelligence, 2008, 22(4): 331-351

[10] Zhen Lu, Jiang Zuhua, Song Haitao. Distributed recommender for peer-to-peer knowledge sharing [J]. Information Sciences, 2010, 180(18): 3546-3561

[11] Bakker A, Ogston E, Steen M. Collaborative filtering using random neighbours in peer-to-peer networks [C] //Proc of the 1st ACM Int Workshop on Complex Networks Meet Information & Knowledge Management. New York: ACM, 2009: 67-75

[12] Ormándi R, Hegedüs I, Jelasity M. Overlay management for fully distributed user-based collaborative filtering [C] //Proc of the 16th Int Euro-Par Conf on Parallel Processing. Berlin: Springer, 2010: 446-457

[13] Xie Bo, Han Peng, Shen Ruiming. PipeCF: A scalable DHT-based collaborative filtering recommendation system [C] //Proc of the 13th Int Conf on World Wide Web. New York: ACM, 2004: 224-225

[14] Xie Bo, Han Peng, Yang Fan, et al. DCFLA: A distributed collaborative-filtering neighbor-locating algorithm [J]. Information Sciences, 2007, 177(6): 1349-1363

[15] Berkovsky S, Kuflik T, Ricci F. Distributed collaborative filtering with domain specialization [C] //Proc of the 2007 ACM Conf on Recommender Systems. New York: ACM, 2007: 33-40

[16] Liu Zhaobin, Qu Wenyu, Li Haitao, et al. A hybrid collaborative filtering recommendation mechanism for P2P networks [J]. Future Generation Computer Systems, 2010, 26(8): 1409-1417

[17] Bickson D, Malkhi D, Zhou Lidong. Peer-to-peer rating [C] //Proc of the 7th IEEE Int Conf on Peer-to-Peer Computing. Los Alamitos, CA: IEEE Computer Society, 2007: 211-218

[18] Höhfeld A, Gratz P, Beck A, et al. Self-organizing collaborative filtering in global-scale massive multi-user virtual environments [C] //Proc of the 2009 ACM Symp on Applied Computing. New York: ACM, 2009: 1719-1723

[19] Wang J, Reinders M, Lagendijk R, et al. Self-organizing distributed collaborative filtering [C] //Proc of the 28th Annual Int ACM SIGIR Conf on Research and Development in Information Retrieval. New York: ACM, 2005: 659-660

[20] Wang J, Pouwelse J, Lagendijk R, et al. Distributed collaborative filtering for peer-to-peer file sharing systems [C] //Proc of the 2006 ACM Symp on Applied Computing. New York: ACM, 2006: 1026-1030

[21] Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms [C] //Proc of the 10th Int Conf on World Wide Web. New York: ACM, 2001: 285-295

[22] Asano T, Ranjan D, Roos T, et al. Space-filling curves and their use in the design of geometric data structures [J]. Theoretical Computer Science, 1997, 181(1): 3-15

[23] Zhu Xia, Luo Junzhou, Song Aibo, et al. A multi-dimensional indexing for complex query in cloud computing [J]. Journal of Computer Research and Development, 2013, 50(8): 1592-1603 (in Chinese)
(朱夏, 罗军舟, 宋爱波, 等. 云计算环境下支持复杂查询的多维数据索引机制[J]. 计算机研究与发展, 2013, 50(8): 1592-1603)

[24] Pujol-Ahullo J, Garcia-Lopez P. PlanetSim: An extensible simulation tool for peer-to-peer networks and services [C] //Proc of the 9th Int Conf on Peer-to-Peer Computing. Los Alamitos, CA: IEEE Computer Society, 2009: 85-86

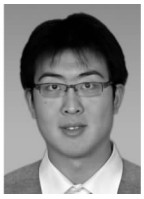
[25] Apache. Mahout [EB/OL]. [2012-12-20] http://mahout.apache.org/

[26] Institut for Informatic Freiburg. Booking-Crossing Data Sets [EB/OL]. [2012-12-20] http://www.informatik.uni-freiburg.de/~cziegler/BX/

[27] Lakshman A, Malik P. Cassandra: A decentralized structured storage system [J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40

Zhu Xia, born in 1983. PhD. His research interests include grid computing, cloud computing and P2P.

Song Aibo, born in 1970. PhD and associate professor. Member of China Computer Federation. His research interests include grid computing, cloud computing, mass data processing and Petri nets.



Dong Fang, born in 1982. PhD and lecturer. Member of China Computer Federation. His research interests include cloud computing, grid computing and mass data processing.

Luo Junzhou, born in 1960. PhD, professor and PhD supervisor. Senior member of China Computer Federation. His research interests include next generation network architecture, protocol engineering, network security, cloud computing and wireless local area network.