基于用户共现矩阵乘子的分布式协同过滤推荐

何 明 吴小飞 常盟盟 任万鹏
(北京工业大学计算机学院 北京 100124)

摘 要 随着大数据时代的到来,应用数据量剧增,个性化推荐技术日趋重要。传统的推荐技术直接应用于大数据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,提出用户共现矩阵乘子推荐策略,将用户相似度矩阵与项目评分矩阵相乘得到用户对项目的预测评分矩阵,从而生成对每个用户的候选推荐项目集;在此基础上,根据分布式处理架构的特点对传统协同过滤算法进行并行化扩展,设计了基于用户的分布式协同过滤算法;最后通过重定义序列组合的 MapReduce 模式将多个子任务串联起来,自动地完成顺序化的执行。实验结果表明,该算法在分布式计算环境下具有良好的推荐精度和推荐效率。

关键词 协同过滤,推荐系统,分布式计算,大数据
中图法分类号 TP391 **文献标识码** A

Distributed Collaborative Filtering Recommendation Based on User Co-occurrence Matrix Multiplier

HE Ming WU Xiao-fei CHANG Meng-meng REN Wan-peng
(College of Computer Science, Beijing University of Technology, Beijing 100124, China)

Abstract With the era of big data's coming, the amount of application data increases sharply, the result of which gives more and more prominence to a personalized recommendation technique. However, traditional recommendation techniques applied to big data are confronted with some problems, such as low recommendation accuracy, long recommendation time and high network traffic. Therefore, the performance of recommendation degrades drastically. To address this issue, a user co-occurrence matrix recommendation strategy was proposed in this paper. The user for the project's predication rating matrix is got by multiplying user similarity matrix and item rating matrix. Candidate items set is generated for each user using user similarity matrix multiply by item similarity matrix. On this basis, traditional collaborative filtering algorithms were parallel expand according to the feature of distributed processing architecture, and a distributed collaborative filtering algorithm was designed. Finally, multi-sub tasks are in series utilizing combination of redefined MapReduce schema to execute automatically. Experimental results show that our approach achieve better prediction accuracy and efficiency in distribute computing environment.

Keywords Collaborative filtering, Recommendation systems, Distributed computing, Big data

1 引言

互联网技术的飞速发展和云计算、物联网、社交网络等新兴服务不断涌现带来了信息超载问题,用户很难从海量数据中及时准确地获取有价值的信息,信息使用效率急剧下降。推荐系统(recommender systems)[1]作为一种有效的信息过滤手段,是当前解决信息过载问题及实现个性化信息服务的有效方法之一。协同过滤(collaborative filtering, CF)[2]推荐是迄今为止使用最为成功的个性化推荐技术之一,其基本思想是发现与目标用户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。目前协同过滤已成为电子商务、社交网络和视频分享等互联网应用的核心技术[3,4]。

当前基于协同过滤的推荐研究主要侧重于串行计算模式。然而随着互联网规模的迅速增长,数据正以前所未有的速度在不断地增长和累积,大数据时代已经到来[5]。传统的协同过滤算法很难满足大数据环境下的推荐系统的计算需求,将其直接应用到大数据计算环境中存在一些局限性:1)需要处理的数据规模更大,从而导致算法的推荐精度降低,传统的集中式协同过滤推荐算法需要进行算法的改进和扩展才能较好地满足大数据环境下推荐系统的需求;2)传统的协同过滤推荐算法在处理海量评分数据规模时,频繁遍历评分矩阵将导致推荐时延急剧提升,加剧推荐算法的高时延问题,难以满足应用的实时推荐需求;3)在基于用户的协同过滤推荐中,相似度矩阵的频繁更新会增加数据中心的网络开销和计算资源。因此,大数据环境下,推荐系统有更高的数据处理和时效性需求,为此,大量研究者关注于传统推荐算法的扩展,希望

本文受国家自然科学基金项目(60803086),国家科技支撑计划子课题(2013BAH21B02-01),北京市自然科学基金项目(4153058,4113076)资助。
何 明(1975-),男,博士,副教授,主要研究方向为推荐系统、数据挖掘、机器学习,E-mail:heming@bjut.edu.cn;吴小飞(1991-),男,硕士生,主要研究方向为大数据与云计算、数据挖掘;常盟盟(1987-),男,硕士生,主要研究方向为机器学习;任万鹏(1990-),男,硕士生,主要研究方向为信息检索。

通过扩展使推荐系统能较好地完成大数据环境下的数据处理和推荐生成任务，分布式推荐算法成为推荐算法研究中一个新的研究方向。

Hadoop[6]是目前最流行的大数据分布式计算平台之一，已经发展成为包括分布式文件系统（HDFS）、数据库（HBase）、数据处理（MapReduce）等功能模块在内的完成生态系统（Ecosystem），用来实现大规模数据集的并行计算。对Hadoop进行改进并将其应用在大数据环境下的推荐系统已经成为新的研究热点之一[7-9]。BartoszKupisz等人[10]利用Hadoop和Spark平台对基于项目的协同过滤算法进行了实验分析和比较，验证了分布式协同过滤算法的有效性；FAN Lu等人[11]分析并改进了基于项目的分布式协同过滤算法，但该算法在处理项目数量比用户数量大的数据集时，计算效率低。上述研究提出的推荐算法都是基于项目的分布式协同过滤，未考虑用户在推荐过程中的作用，算法适用范围有局限性，并且推荐模型过于复杂，可扩展性低。

综上所述，将传统的协同过滤推荐算法直接应用到大数据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，推荐算法的性能和质量难以得到保证。为解决上述问题，本文将协同过滤算法与Hadoop的分布式计算特性相结合，提出了基于用户的分布式协同过滤推荐算法，主要贡献如下：

（1）提出了用户共现矩阵乘子推荐策略对传统的协同过滤推荐算法进行了并行化扩展，将传统协同过滤算法扩展为适应Hadoop平台的分布式协同过滤算法，以提高推荐算法的计算效率；

（2）基于用户矩阵乘子推荐策略改进传统的集中式协同过滤推荐算法，设计基于用户的分布式协同过滤推荐算法，以进一步提高推荐质量；

（3）设计了序列组合的MapReduce模式，将多个MapReduce子任务串联起来，自动地完成顺序化的执行。

本文第2节对基于用户的协同过滤推荐的基本概念、算法步骤以及MapReduce计算模型进行了介绍；第3节对分布式可行性进行分析，提出了基于用户的分布式协同过滤推荐算法，并通过MapReduce组合模式进行了实现；第4节描述了推荐算法的实验过程，并对实验结果进行了评估和分析；最后总结全文并指出未来的工作。

2 传统的协同过滤推荐算法

2.1 用户-项目评分矩阵

传统协同过滤推荐算法基于用户-项目评分矩阵$R(m,n)$寻找目标用户的最近邻集合。$R(m,n)$是一个$m\times n$阶的用户-项目评分矩阵，如表1所列，其中m行表示m个用户，n列表示n个项目。$R_{i,j}$表示用户i对项目j的评分值。

表1 用户-项目评分矩阵$R(m,n)$

	Item_1	Item_2	…	Item_n
User_1	$R_{1,1}$	$R_{1,2}$	…	$R_{1,n}$
User_2	$R_{2,1}$	$R_{2,2}$	…	$R_{2,n}$
⋮	⋮	⋮	⋮	⋮
User_m	$R_{m,1}$	$R_{m,2}$	…	$R_{m,n}$

2.2 计算最近k近邻

计算与目标用户相似的最近k邻居是基于用户协同过滤推荐算法中最重要的一步。对于一个用户u而言，最近邻居的形成是为了找到一个队列：$UsersN=\{N_1,N_2,\cdots,N_i\}$。如果$sim(u,N_i)$表示用户$u$与用户$N_i$的相似度，那么其中$sim(u,N_1)$的值最大，$sim(u,N_2)$的值其次，这样依次递减，最后得到队列的前$k$个用户即为用户$u$的$k$近邻用户。

2.3 相似度计算

对于目标用户u最近k邻居是通过计算相似性去比较的。在大多数协同过滤算法中，无论是余弦相似性度量还是皮尔逊相关系数，其共同特征都是去计算两个用户评分过的项目集合的交集形成的一维向量的线性相关性或者距离，如图1所示。

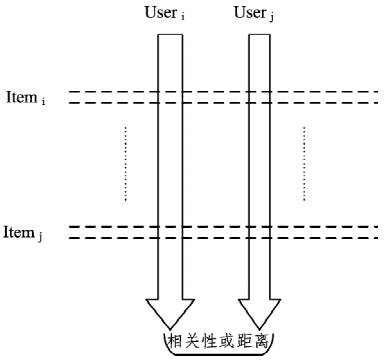


图1 用户间的线性相关性或距离

用户相似性度量方法主要有余弦相似性（cosine similarity）、Pearson相关系数（Pearson correlation）、约束Pearson相关系数（constrained Pearson correlation）等，具体计算方法如下：

（1）余弦（Cosine）相似性：余弦相似性也叫向量相似性。每个用户的所有评分记录被看成n维空间中的一个向量，两个用户之间的相似度被看成这两个用户的评分向量之间夹角的余弦。假设两个用户x、y的评分向量分别为$\vec{x}$和$\vec{y}$，集合I_x、I_y分别表示x和y评分过的项目集合，则它们之间的余弦相似性为：

$$Sim(x,y)=\cos(\vec{x},\vec{y})=\frac{\sum\limits_{i\in I_{x,y}} r_{x,i}r_{y,i}}{\sqrt{\sum\limits_{i\in I_x} r_{x,i}^2}\sqrt{\sum\limits_{i\in I_y} r_{y,i}^2}} \tag{1}$$

（2）相关（Correlation）相似性：在相关相似性的计算中，根据两个用户共同评分过的所有项目的评分来计算Pearson相关系数，并以此作为两个用户之间的相似度。假设两个用户U_i、U_j共同评分过的项目集合为I_{ij}；R_{ik}和R_{jk}分别表示用户U_i和U_j对项目k的评分；A_i和A_j分别表示用户U_i和U_j对所有项目评分的平均分。则用户U_i、U_j之间的相关相似性为：

$$Sim(U_i,U_j)=\frac{\sum\limits_{k\in I_{ij}}(R_{ik}-A_i)(R_{jk}-A_j)}{\sqrt{\sum\limits_{k\in I_{ij}}(R_{ik}-A_i)^2\sum\limits_{k\in I_{ij}}(R_{jk}-A_j)^2}} \tag{2}$$

2.4 推荐生成

当目标用户的最近k邻居集形成后，就可以为目标用户生成推荐项目。用户u对于项目t的预测评分如下：

$$P_{ut}=A_u+\frac{\sum\limits_{i=1}^{k}(R_{it}-A_i)Sim(u,i)}{\sum\limits_{i=1}^{k}Sim(u,i)} \tag{3}$$

其中，A_u和A_i分别表示用户u和用户i的平均评分，R_{it}表示最近邻居用户i对项目t的评分，$sim(u,i)$表示用户u和用户i之间的相似性。式（3）的实质是在用户的最近邻居集中查找用户，并将目标用户与查找到的用户间的相似度作为权值，然后将邻居用户对该项目的评分与此邻居用户的所有评分的

差值进行加权平均，通过预测出目标用户对未评分项目的评分，进而选择预测评分最高的 *Top-N* 项推荐给目标用户。

3 分布式协同过滤推荐算法

3.1 MapReduce

MapReduce[12] 是一个使用大量商用或者廉价计算机并在庞大的数据集合上执行高度并行化和分布式算法的框架。MapReduce 的引入是为了解决大规模数据的计算问题，并且可以部署在同一个分布式集群上运行。MapReduce 采用了对数据“分而治之”的方法来完成并行化的大数据处理。图 2 展示了这种基于数据划分和“分而治之”策略的基本并行化计算模型。

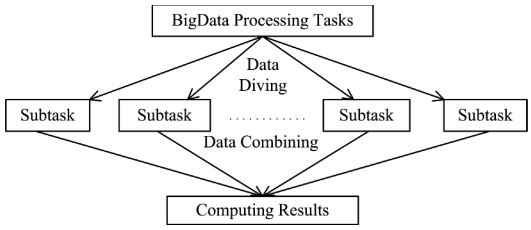


图 2 “分而治之”策略的基本并行化计算模型

Hadoop 集群中，用于执行 MapReduce 任务的机器有两个角色：JobTracker 和 TaskTracker。JobTracker 是用于管理和调度工作的，TaskTracker 是用于执行工作的。一个 Hadoop 集群中只有一台 JobTracker。每个 MapReduce 任务都被初始化为一个 Job。每个 Job 又可以分为两个阶段：Map 阶段和 Reduce 阶段，这两个阶段分别用两个函数来表示，即 *Map* 函数和 *Reduce* 函数。*Map* 函数接受一个〈*key*，*value*〉形式的输入，然后产生同样为〈*key*，*value*〉形式的中间输出，*Hadoop* 会负责将所有具有相同中间 *key* 值的 *value* 集合到一起传递给 *Reduce* 函数，*Reduce* 函数接收一个如〈*key*，(*list of values*)〉形式的输入，然后对这个 *value* 集合进行处理并输出结果，*Reduce* 的输出也是〈*key*，*value*〉形式的。经过 Map 和 Reduce 的抽象后，MapReduce 将演化为如图 3 所示的并行计算模型。

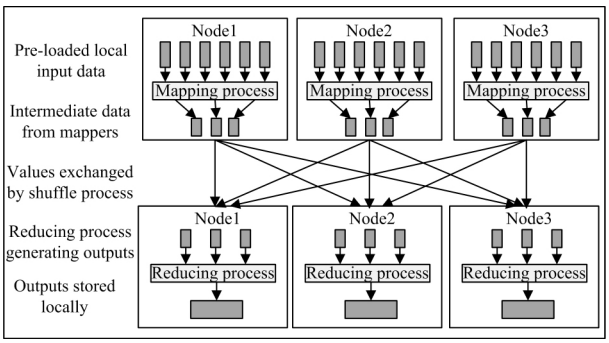


图 3 MapReduce 并行编程模型

图 3 并行编程模型的基本处理过程如下：

(1)各个 *Map* 结点对所划分的数据进行并行处理，从不同的输入数据产生相应的中间结果输出；

(2)各个 *Reduce* 结点也各自进行并行计算，各自负责处理不同的中间结果数据集合；

(3)进行 Reduce 处理之前，必须等到所有的 *Map* 节点处理完，因此在进入 Reduce 前需要有一个同步障(Barrier)，这个阶段也负责对 *Map* 的中间结果数据进行收集整理(Aggregation&Shuffle)处理，以便 *Reduce* 节点可以完全基于本节点上的数据计算最终结果；

(4)汇总所有 Reduce 的输出结果即可获得最终结果。

3.2 分布式可行性分析

传统的协同过滤算法若要在分布式下实现，需要将传统的协同过滤推荐算法进行并行性化分析和扩展，从而使其能够适应分布式环境。不同的分布式平台需要考虑的并行化机制也会不同。如果设计一种算法既能够保证跟传统的推荐算法一致又能够适应 MapReduce 的计算模式，那么这种算法必然可以从单机环境迁移到 Hadoop 分布式平台下实现。

MapReduce 的计算过程为：先从 HDFS 分布式文件中读取大文件数据并切分成多份作为每个 Mapper 的输入，输入形式为许多健值对(k1，v1)；每个 Mapper 处理一份数据，通过 Map 函数执行同样的运算并产生许多键值对(k2，v2)；Reducer 把多个 Mapper 的结果组合成一个，最终将数据输出键值对(k3，v3)并写回到 HDFS 中，如图 4 所示。所以，通过 MapReduce 计算模式来解决问题，则必须套用这个模式或由多个基于该模式的计算串在一起来完成。形成这个计算模式后就可以借助 Hadoop 和 HDFS 有效地进行分布。

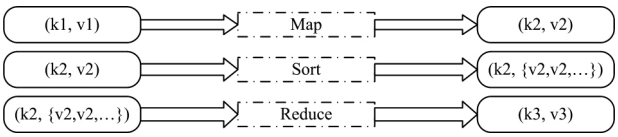


图 4 MapReduce 执行过程

MapReduce 适用于问题相对简单但数据量很大的情形，且它必须在多台机器上并行地处理。一个算法是否能够应用 MapReduce，必须满足以下条件：

(1)需要运行的计算必须是可组合的。这意味着应该能够在数据子集上运行计算，然后合并各局部结果。

(2)数据集规模要足够大(或者计算时间足够长)，使得为各个独立计算进行数据拆分和结果合并产生的基础设施开销不会伤害到整体性能。

(3)计算主要取决于正在处理的数据集。可以使用 HBase、分布式缓存或其他技术来添加额外的小数据集。

传统基于用户的协同过滤算法是无法满足条件(1)的。MapReduce 初始计算会将输入的数据集划分为独立的块(InputSplit)，只有输入文件的数据之间不满足数据相关性，即可以将它拆分成可以独立运行的子计算，这样数据在分块划分进行子计算并输出结果时才能保证最终计算的正确性。

尽管传统的协同过滤算法不能直接通过 MapReduce 进行分布式实现，但对于相同的底层问题来说，通常存在可以利用 MapReduce 解决的替代方案。虽然在一定程度上可能会损失精确度，但是在时间性能上将得到提升。

3.3 算法设计与实现

将传统单机环境下的协同过滤推荐算法直接应用到分布式环境下并不适用。算法内部循环中的相似性计算需要随机访问整个用户-项目矩阵的行和列，在算法只能计算被分割的部分数据且不能够完全通过 Hadoop DistributedCache 技术实现数据共享的分布式环境下，想要直接进行并行计算是很困难的。因此，本文以 MapReduce 框架作为依据，设计并实现了适用于分布式环境下的两种协同过滤方法。

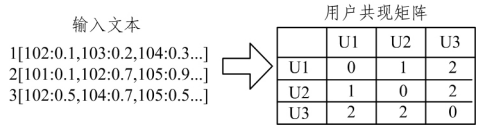
3.3.1 用户共现矩阵乘子推荐策略

“单词共现算法”[13] 是自然语言处理中应用非常广泛的一种算法，通过在海量语言文本库中发现在固定窗口内单词

a 和单词 b 共同出现的频率，从而构建单词共现矩阵。本文受此算法的启发，提出了用户共现矩阵乘子推荐策略，其核心思想是将用户间的相似度矩阵与项目的评分矩阵相乘得到用户对项目的预测评分矩阵，从而得到对每个用户的候选推荐项目集。其工作原理是首先计算用户间的相似度矩阵，为了便于使用 MapReduce 编程模型，将用户间的相似度矩阵简化为用户间的共现矩阵；然后，将共现矩阵与用户未评分过的所有项目向量相乘得到用户对项目的预测评分矩阵；最后，根据得到的预测评分值对用户做出推荐。其具体过程的步骤如下。

Step1　构建用户共现矩阵

用户共现矩阵不是用于计算每个用户对之间的相似性，而是计算在某些项目的用户评分列表中每个用户对共同出现的次数并以此来填充构建的。它描述了用户之间的关联，两个用户同时出现得越多，他们越有可能相关或相似。用户共现矩阵的作用类似于基于用户的非分布式算法中的用户相似性度量。因此，它是一个对称方阵，行和列的数目等于数据模型中的用户数，对角线的元素没有意义就用 0 代替。每行（以及每列）表达在一个特定用户和其他所有用户之间的相似性，构建过程如图 5 所示。



	U1	U2	U3
U1	0	1	2
U2	1	0	2
U3	2	2	0

图 5　用户共现矩阵构建过程

Step2　计算项目向量

将项目所对应的所有用户的评分看作是一个向量。在该向量中，对于用户表示评价过的项目必然会有个偏好数值；而对于用户没有评分过的项目，用 0 表示。项目向量就是项目-用户矩阵中的某一行向量，包含所有用户对某特定项目给出的评分。

Step3　生成推荐结果

通过将项目的行向量与用户共现矩阵相乘，可以得到一个新的行向量。比如将项目 103 推荐给哪个用户，只需将 103 项目作为行向量，用它乘以用户共现矩阵即可，计算过程如图 6 所示。

103	0.2	0	0

X

	U1	U2	U3
U1	0	1	2
U2	1	0	2
U3	2	2	0

=

R	0	0.2	0.4

图 6　项目 1.3 乘以用户共现矩阵生成推荐结果 R

在该向量中，项目推荐的目标用户（对此项目没有评分过的用户）对应的计算值最大。通过迭代，多个项目的行向量相乘得到的结果在某个用户下会对应多个推荐的项目。因此，此方法不仅能将某个项目推荐给合适的用户，也能为某个用户推荐合适的项目。

从图 6 可以看出，项目 103 是 U1 已评分的项目，评分值为 0.2。U2 和 U3 并没有对项目 103 进行评分，所以应该将其推荐给用户 U2 和 U3。在候选推荐的用户集中，选择用户 U3 为推荐目标用户是考虑 U3 对应的 R 值是三者之中最大的。在计算过程中，R 中第 3 列的值为项目-用户矩阵中 103 所在的行向量与用户共现矩阵中第 3 列的列向量 U3 的点积，矩阵运算式子为 $0.2\times2+0\times2+0\times0=0.4$。项目 103 作为用户 U1 喜爱的项目，喜爱程度为 0.2，同时又是 U2 和 U3 没有表达过偏好的项目。而 U1 和 U3 的相似度为 2，和 U2 的相似度为 1，说明 U1 和 U3 更相似。总之，如果一个用户非常偏好某个项目，并且这个项目是其他用户没有接触过的，但其他用户中有兴趣和这个用户非常相似的，那么这个用户喜爱的项目在很大程度上会受到兴趣相投的用户的偏好。可以参照最优二叉树中带权路径长度的定义来加以说明，把项目 103 向量的值作为权，用户 U3 向量作为路径，那么当各个权值很大且对应的路径分量值很大时，整个乘积的和必然会很大。这就是 R 中较大值会反映偏好程度偏高的原因。需要注意的是，这里的 R 值并不代表一个估计偏好值，其作用是用来决定推荐顺序，在实际评价时可以利用一些额外的信息将其归一化为估计偏好值。

通过以上分析可知，基于用户共现矩阵乘子推荐策略所生成的推荐结果是合理的。同时，用户共现矩阵的计算每次只需遍历一个向量，而且生成推荐向量时仅需每次加载矩阵的一行或一列。由此表明，该策略适合大规模的分布式计算框架，可以使用 MapReduce 范式实现。

3.3.2　用户共现矩阵乘子推荐的 MapReduce 实现

通过 MapReduce 实现用户共现矩阵乘子策略，需要将前几个设计步骤向 MapReduce 模式转换，具体实现步骤如下：

（1）生成项目向量

输入文件采用 $userID$：$itemID1$，$preference$；$itemID2$，$preference$…的形式。通过 $Mapper$ 和 $Reducer$ 将用户-项目矩阵转化成项目-用户矩阵，得到所有项目行向量。$MapReduce$ 作业描述如下：

Map：$\langle UserID,(List\ of\ rated\ items)\rangle\rightarrow\langle ItemID\ in\ list,(UserID,rating\ of\ item)\rangle$。

Reduce：$\langle ItemID\ in\ list,(UserID,rating\ of\ item)\rangle\rightarrow\langle ItemID,(List\ of\ users\ and\ ratings)\rangle$。

（2）生成共现矩阵

输入到 $Mapper$ 的是上一步得到的项目-用户矩阵的行，$Mapper$ 对其处理得到所有相邻用户对，$Reducer$ 接受所有用户对进行共现次数统计。$MapReduce$ 作业描述如下：

Map：$\langle ItemID,(List\ of\ users\ and\ ratings)\rangle\rightarrow\langle(UserIDi,UserIDj),concurrenceTimes\rangle$。

Reduce：$\langle(UserIDi,UserIDj),concurrenceTimes\rangle\rightarrow\langle(UserIDi,UserIDj),sumConcurrenceTimes\rangle$。

（3）生成推荐

通过 MapReduce 将得到的项目向量和用户共现矩阵相乘，从而得到一个推荐向量 R。

矩阵乘法的定义为：$P_{ik}=(\boldsymbol{M}\times\boldsymbol{N})\ ik=\sum_j m_{ij}n_{jk}$，其中 m_{ij} 记作矩阵 $\boldsymbol{M}$ 的第 i 行第 j 列的元素，n_{jk} 记作矩阵 $\boldsymbol{N}$ 的第 j 行第 k 列的元素。由公式可以看出，决定最后 P_{ik} 位置的是 (i,k)，因此可以将其作为 Reducer 的输入 key 值。为了求解 $m_{ij}n_{jk}$，需要分别知道 m_{ij} 和 n_{jk}。对于 m_{ij}，其所需要的属性有矩阵名称 $\boldsymbol{M}$，所在行数 i，所在列数 j 和其本身的数值大小 m_{ij}；同样对于 n_{jk} 也是如此。这些属性值由 Mapper 处理得到，基本处理思路如下：

Map 函数：对于矩阵 $\boldsymbol{M}$ 中的每个元素 m_{ij}，产生一系列的 Key-value 对 $\langle(i,k),(\boldsymbol{M},j,m_{ij})\rangle$，其中 $k=1,2,\cdots,|\boldsymbol{N}|$；对于矩阵 $\boldsymbol{N}$ 中的每个元素 n_{jk}，产生一系列的 Key-value 对 $\langle(i,k),(\boldsymbol{N},j,n_{ijk})\rangle$，其中 $i=1,2,\cdots,|N|$。

Reduce 函数：对于每个键 (i,k) 相关联的值 $(\boldsymbol{M},j,m_{ij})$ 及 $(\boldsymbol{N},j,n_{jk})$，根据相同的 j 值将 m_{ij} 和 n_{jk} 分别存入不同数组中，然后将两者的第 j 个元素抽取出来分别相乘，最后相加，即可

得到 P_{ik} 的值。

上述计算过程中会对每一列做一次向量的点积，需要遍历整个用户共现矩阵。如果在输入文本规模很大的情况下，共现矩阵数据可能无法本地化。我们注意到用户共现矩阵是对称的，即行与列相同，则矩阵相乘可转化为一个对共现矩阵中行的点积。因此，可以对上述矩阵乘法计算进行适当优化：

```
if vector〈R〉 is not null do
        vector〈R〉=null
for each row i in User-Item Matrix do
    vector〈R′〉=element〈item_ki〉×vector
        〈user_i〉
  vector〈R〉=vector〈R′〉+vector〈R〉
end for
```

实际上，上述方法也可以视为另一种形式的矩阵相乘，只要项目向量中的元素 i 为 0，可跳过此循环，只需对项目向量的非零元素执行循环即可。当项目向量稀疏时，可以大大减少计算量。

3.3.3 基于用户的分布式协同过滤算法

用户共现矩阵乘子推荐策略的分布式实现比较简单，其主要原因有两点：1）用户相似度计算采用共现次数表示，而没有使用评分值；2）通过简单的矩阵乘运算计算推荐值，而没有使用预测评分公式。在用户共现矩阵乘子推荐策略的基础上，本文进一步对计算用户相似度和预测评分两方面进行改进和扩展，提出了一种基于用户的分布式协同过滤推荐算法，主要步骤如下。

Step1 数据预处理

基于用户和基于项目的协同过滤算法对于给定的数据文件都需要将其进行预处理，转化成用户-项目矩阵或者项目-用户矩阵的形式。给出 Hadoop 平台上根据用户-项目评分记录文件生成评分矩阵的过程，该过程采用 2 个 MapReduce 作业完成，分别得到用户-项目矩阵和项目-用户矩阵，它们互为倒置关系且可以相互转换，如图 7 和图 8 所示。

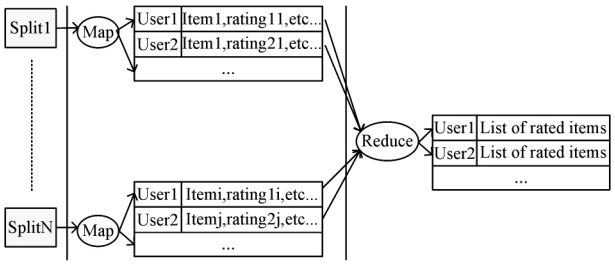


图 7 用户-项目矩阵构建过程

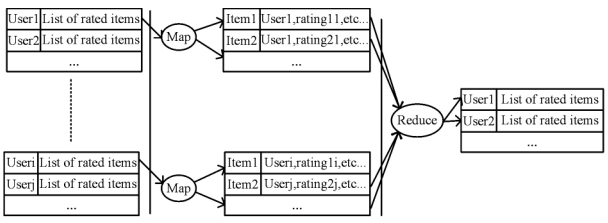


图 8 项目-用户矩阵构建过程

Step2 用户相似度矩阵计算

本文采用 Pearson 相关系数度量方法来计算用户间的相似度。根据公式可知，若要计算用户 U_i 和 U_j 的相似度，则需要得到两个用户共同评分的项目集合，即用户向量对 $[\overrightarrow{U_i}, \overrightarrow{U_j}]$。MapReduce 作业如表 2 所列。

表 2 计算两个用户共同评分的项目集合的作业

处理阶段	描述
mapper	将作业的输入〈项目-用户评分矩阵〉的每一行向量〈item i, List of users and ratings〉拆分成多个〈{userA，userB}，{ratingA$_i$，ratingB$_i$}〉，其中 userA 和 userB 属于用户集中两两互不相同的用户，ratingA$_i$ 是用户 A 对项目 i 的评分。
shuffle	MapReduce 的 shuffle 将基于节点的键{userA，userB}来分割排序所有 key-value 记录，并将具有相同键的两条记录发送到相同的 reducer。
reducer	合并具有相同 key 的 key-value 记录。合并后的形式为〈{userA，userB}，{all items with rating of userA and userB}〉。其中，value 是一个集合，表示的是用户 A 和用户 B 共同评分的项目列表向量。
结果	〈{userA，userB}，sim(A，B)〉。其中 sim(A，B)是 reducer 得到的两两用户向量对根据 Pearson 相关系数计算得到的，最后的结果输出到 HDFS 本地上。

在上一个 MapReduce 作业的基础上，进一步将离散的用户相似度值聚合成矩阵向量，并筛选出所有用户的最相似 k 位近邻用户，从而构建用户相似度矩阵。MapReduce 作业如表 3 所列。

表 3 计算用户-用户相似度矩阵的作业

处理阶段	描述
mapper	将上一阶段输出的〈userA，userB〉key 值打散，获得〈userA，{userB，sim(A，B)}〉和〈userB，{userA，sim(A，B)}〉。
shuffle	将基于节点的键 user$_i$ 来分割排序所有 key-value 记录，并将具有相同键的两条记录发送到相同的 reducer。
reducer	在该作业中，reducer 的角色是合并具有相同 key 的 key-value 记录。聚合后的形式为〈user$_i$，{Top-k similarities of user$_i$}〉，其中 value 是一个集合，表示的是用户 i 的最相似 k 近邻用户集。
结果	一个包含所有用户的用户相似度矩阵，最后的结果输出到 HDFS 本地上。

Step3 推荐生成

在用户共现矩阵乘子推荐策略中，预测评分是用户相似度矩阵与项目向量矩阵相乘得到的，而本算法则采用传统的评分预测公式进行计算。由式(3)可知，计算用户 u 对项目 t 的预测评分，必须知道用户 u 的最近 k 邻居以及每一个最近邻对所有项目的评分向量。在推荐生成前，已经得到了用户-项目评分矩阵 $R_{m\times n}$、项目-用户评分矩阵 $R^{\mathrm{T}}_{m\times n}$ 和用户相似度矩阵。在推荐生成过程中，同样分为两个 MapReduce 作业。作业 1 使用了两个 Mapper，分别处理用户-项目评分矩阵 $R_{m\times n}$ 和用户-用户相似度矩阵的输入。然后通过矩阵转置后的合并得到了每位用户的项目集合和 k 近邻用户集合。Mapper 作业如表 4 所列。

表 4 计算用户评分项目和 k 近邻用户的作业

处理阶段	描述
mapper	在该作业中，mapper 担任了两个不同的功能：一方面，处理用户相似度矩阵的输入，将每行〈user$_i$，{Top-kSimlilarities of user$_i$}〉进行键-值互换得到〈user$_k$，{user$_i$，sim(A，B)}〉，其中 user$_k$ 属于 user$_i$ 的 Top-k 集合；另一方面，直接读入用户-项目评分矩阵 $R_{m\times n}$ 的每行数据〈userk，{List of items and ratings}〉。
shuffle	将基于节点的键 userk 来分割排序所有 key-value 记录，并将具有相同键的两条记录发送到相同的 reducer。
reducer	合并具有相同 key 的 key-value 记录。聚合后的形式为〈user$_k$，{user vector and Top-k Similarity vector of user$_k$}〉，其中 user vector 代表 user$_k$ 评分过的所有项目集，Top-k Similarity vector 代表 user$_k$ 的 k 近邻用户集。
结果	该作业的执行结果得到了每位用户评过分的所有项目以及转置后的相似用户集合，最后的结果输出到 HDFS 本地上。

作业 2 在第一个作业的输出后继续进行相似度矩阵转置

处理，得到预测评分计算所需的条件，并最终实现了用户的 top-N 推荐。MapReduce 作业如表 5 所列。

表 5　用户 top-N 推荐的作业

处理阶段	描述
mapper	在该作业中，mapper 使用第一个作业流的结果〈$user_k$，{user vector and Top-k Similarity vector of $user_k$}〉作为输入，通过值-键互换得到〈$user_i$，{$user_k$，sim(k，i)，user vector of $user_k$}〉，其中 $user_i$ 属于 $user_k$ 的 k 近邻集合。
shuffle	基于节点的键 $user_i$ 来分割排序所有 key-value 记录，并将具有相同键的两条记录发送到相同的 reducer。
reducer	合并具有相同 key 的 key-value 记录。聚合后的形式为〈$user_i$，{ Neighbors and their preferences}〉，其中 Neighbors 代表 $user_i$ 的 Top-k 近邻集，preferences 代表 $user_i$ 的所有近邻的评分向量集；最后，根据预测公式计算出用户对所有未评分项目的预测评分，并按照降序排序。
结果	该作业的执行结果是为某用户推荐的 top-N 项目，最后的结果输出到 HDFS 本地上。

3.3.4　MapReduce 序列组合模式

通过对以上两种算法进行分析，发现在 MapReduce 模式下重新定义基于用户的协同过滤算法时，很难用一趟 MapReduce 处理过程来完成，需要将其分解成一个作业序列流。其中一个 MapReduce 作业的输出会成为下一个 MapReduce 作业的输入。本文设计了序列组合的 MapReduce 模式，通过将多个 MapReduce 子任务串联起来自动地完成顺序化的执行，配置核心代码如图 9 所示。

```
//数据预处理配置执行代码
Configuration preDataConf = new Configuration();
Job preDataJob = new Job(preDataConf, "PreDataJob");
preDataJob.setJarByClass(PreDataJob);
    ……
FileInputFormat.addInputPath(preDataJob, inpath1);
FileOutputFormat.setOutputPath(preDataJob, outpath1);
preDataJob.waitForCompletion(true);

//相似度矩阵计算配置执行代码
Configuration buildSimMatrixConf = new Configuration();
Job buildSimMatrixJob = new Job(buildSimMatrixConf "BuildSimMatrixJob");
buildSimMatrixJob.setJarByClass(BuildSimMatrixJob);
    ……
FileInputFormat.addInputPath(buildSimMatrixJob, outpath1);
FileOutputFormat.setOutputPath(buildSimMatrixJob, outpath2);
buildSimMatrixJob.waitForCompletion(true);

//推荐计算配置执行代码
Configuration recommendConf = new Configuration();
Job recommendJob = new Job(recommendConf, "RecommendJob");
recommendJob.setJarByClass(RecommendJob);
    ……
FileInputFormat.addInputPath(recommendJob, outpath2);
FileOutputFormat.setOutputPath(recommendJob, outpath3);
recommendJob.waitForCompletion(true);
```

图 9　序列组合的 MapReduce 作业配置

4　实验

4.1　实验环境的搭建

4.1.1　集群环境

实验环境是由 5 个节点组成的 Hadoop 集群，包括 1 个 Master，4 个 Slave，节点之间通过局域网连接，IP 地址分布如表 6 所列。

表 6　Hadoop 集群节点的 IP 地址

主机名称	IP 地址
Master. root	192. 168. 140. 2
Slave1. root	192. 168. 140. 3
Slave2. root	192. 168. 140. 4
Slave3. root	192. 168. 140. 5
Slave4. root	192. 168. 140. 6

每个计算节点配置为：CPU(Intel Core 4 核)、主频(2. 60 GHz)、内存(2G)、硬盘容量(SATA 1T)。节点操作系统配置：5 个节点均安装的是 CentOS7. 0 版本的 linux 操作系统，并且有一个相同的用户 root。每台机器上都安装配置有 JDK6. 0、Hadoop1. 0. 1、HBase0. 94、ZooKeeper3. 3. 6、Mahout0. 6 等软件，构成了一个可以运行 MapReduce 程序的 Hadoop 分布式环境。

Hadoop 集群架构如图 10 所示。在集群中，Master Node 主要充当 NameNode 和 JobTracker 两个角色，负责总管分布式数据和分解任务的执行；4 个 Salve Node 充当 DataNode 和 TaskTracker 两个角色，负责分布式数据存储以及 MapReduce 任务的执行。

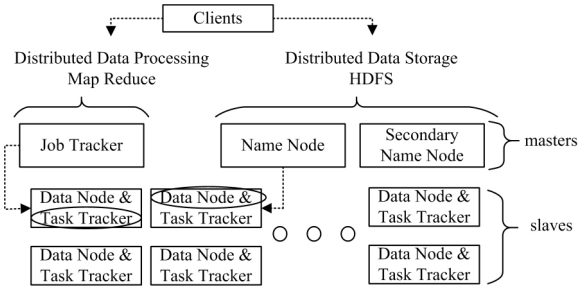


图 10　Hadoop Cluster Architecture

4.1.2　开发环境

Hadoop 虽然提供了与 HDFS 进行交互的 shell 方式，但在开发复杂的 MapReduce 程序时，使用 IDE 可以很方便地编写和调试 MapReduce 代码。本文实验采用了 Eclipse，它作为开源 IDE 有着非常强大的功能，可以通过提供插件的方式来满足很多需求。如为了编译构建 java 程序可以安装 Maven 插件，同时对于支持 MapReduce，Eclipse 也提供了可以安装的插件。安装后的配置如图 11 所示。

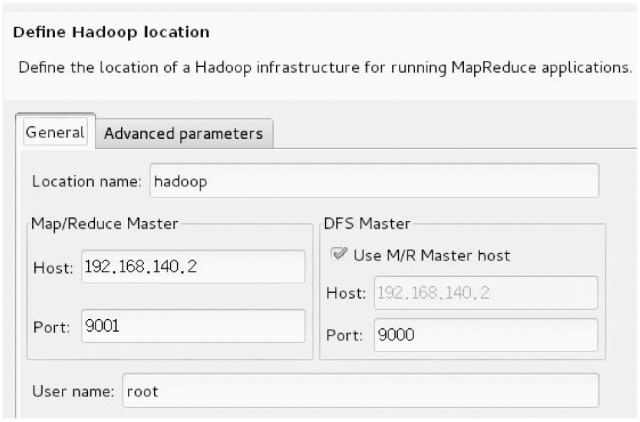


图 11　Eclipse 安装 MapReduce 插件后的配置

4.2　实验评估与分析

4.2.1　数据集

推荐系统常用的数据集有 MovieLens、EachMovie、BookCrossing 和 Netflix 等。MovieLens 数据集由 GroupLens 网站(http://grouplens. org/datasets/movielens/)提供，本文选择 MovieLens(https://movielens. org)作为实验数据集。按照数据集的大小分为：

• MovieLens 100K Dataset：包含了 943 位用户对 1682 部电影的 100000 条评分(1-5)数据，稀疏等级为 0. 937；

• MovieLens 1M Dataset：包含了 6040 位用户对大约 3900 部电影的 1000209 条评分(1-5)数据，稀疏等级为0. 954；

• MovieLens 10M Dataset：包含了 71567 位用户对

10681 部电影的 10000054 条评分(1-5)数据,稀疏等级为 0.987。

在实验过程中,选用的数据集太小或太大都不适宜,应按照不同的实验设计方案选用相对合适的数据集。并且为了评价一个推荐系统的推荐准确度,必须把用户评分数据集划分为训练数据集(Training Set)和测试数据集(Test Set)两部分。其中,训练集用来计算用户间的相似性以及预测用户对未评分项目的评分;测试集用来测试算法利用训练集预测评分的准确性。

4.2.2 评估标准

本文选择推荐精度和推荐效率作为评估分布式推荐算法质量的度量标准。其中在推荐精度方面选择平均绝对误差(Mean Absolute Error,MAE),它通过计算所有的预测用户评分值和实际用户评分值之间的偏差来衡量预测结果的准确度。MAE 的值越小,表明算法的推荐精度越高。对于一个目标用户 u,假设其在测试集中的评分数目为 T_u,相应的评分分别为 $\{R_1,R_2,\cdots,R_{T_u}\}$;对于测试集中的这 T_u 个评分,系统会根据训练集的评分信息计算 T_u 个预测评分,设对应的预测评分为 $\{P_1,P_2,\cdots,P_{T_u}\}$,则目标用户 u 的 MAE 为:

$$MAE_u=\frac{\sum_{i=1}^{T_u}|P_i-R_i|}{T_u} \tag{4}$$

在推荐效率方面,由于协同过滤算法是在分布式环境下并行实现的,因此本文不仅要衡量算法的时间开销,还要对分布式协同过滤算法的并行效率进行分析。算法的时间开销包括数据预处理 T_{pre}、相似度计算 T_{sim} 和生成推荐 T_{rec} 整个推荐过程的运行时间:

$$T=T_{pre}+T_{sim}+T_{rec} \tag{5}$$

衡量并行效率的一个典型指标是加速比,计算公式如式(6)所示。其中 T_1 为集群中单个 DataNode 的计算时间,T_p 为包含 p 个 DataNode 的集群并行计算的总时间。

$$Speedup=\frac{T_1}{T_p} \tag{6}$$

4.2.3 参数分析

本节通过调节候选邻居数量 k、训练集与测试集的比例 TP 两个参数对用户共现矩阵乘子推荐算法(UCMB-DCF)和基于用户的分布式协同过滤算法(UB-DCF)的推荐精确性进行优化。两个实验选用 MovieLens 100K Dataset 作为数据集。

实验 1 候选邻居数量变化时的 MAE

如图 12 所示,近邻数量 k 从 5 变化到 50,MAE 也在随之变化。可以看出,两种算法的 MAE 变化基本一致,并且在 k 确定时,UB-DCF 比 UCMB-DCF 的精确度高;当 $k=5$ 时,MAE 值都很大,随着 k 慢慢增加到 30 左右后,MAE 值基本不再继续减小。由此说明,近邻数量太小会影响推荐质量,但过多的近邻数量并不会明显地提高推荐算法的推荐质量,相反还会增加计算量。因此,本文选择候选邻居数量参数 $k=35$。

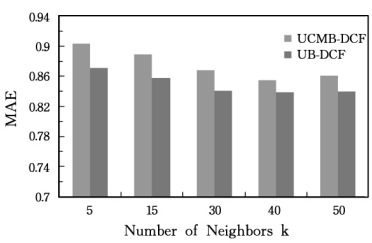


图 12 参数 k 变化时的 MAE 比较

实验 2 训练集占比变化的 MAE

将数据集随机分成 10 组,即每组的数据都占原始数据集的 10%。如果选择其中的若干组为训练集,那么其他组的数据就构成了测试集,对于两个算法,都做 5 次实验,分别测试训练集占比为 50%,60%,70%,80%,90%时的 MAE 值。如图 13 所示,随着训练集占比 TP 的不断增加,两个算法的 MAE 值都在逐渐下降,说明推荐的数据规模越大,推荐结果越精确,推荐效果越好。同时,当 TP 超过 70%时,MAE 值变化相对平缓,当超过 90%时,UCMB-DCF 的 MAE 值略微上升。因此,本文选择训练集占比参数 $TP=85\%$。

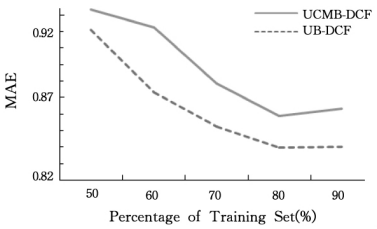


图 13 参数 TP 变化时的 MAE 比较

4.2.4 与其他算法的性能比较分析

实验 3 不同推荐算法间的运行时间比较

本节将本文中的两个算法与传统单机下基于用户的协同过滤算法(UB-CF)[14]和分布式下基于项目的协同过滤算法(IB-DCF)[11]进行比较,测试比较各算法在时间效率和推荐精度方面的性能。实验选用 MovieLens 100K Dataset,并根据用户数量将这个数据集分为 5 组实验数据,如表 7 所列。

表 7 实验数据分组情况

组号	用户数量	项目数量	评分数量
1	50	1080	4851
2	200	1420	17748
3	400	1546	40318
4	800	1671	77537
5	943	1682	100000

图 14 可知,用户数量为 50 时,UB-CF 运行时间比其他分布式算法都要少,原因是在用户数量很少时,数据量能够加载到单机内存中进行计算,而分布式算法在使用 MapReduce 计算时会带来额外的时间开销;随着用户数量增多,分布式算法的运行时间相对传统算法开始逐渐缩小,说明在数据量很大的情况下,分布式算法要比传统算法更有优势;UB-DCF 和 UCMB-DCF 相比较下,UCMB-DCF 的运行时间明显比 UB-DCF 要少,这也说明了改进后的 UB-DCF 在计算用户相似度和推荐计算的过程中消耗了不少时间;另外,本文提出的两种算法明显比 IB-DCF 的运行时间少,说明本文算法在处理用户数量比项目数量少的数据集时,计算效率上更有优势。

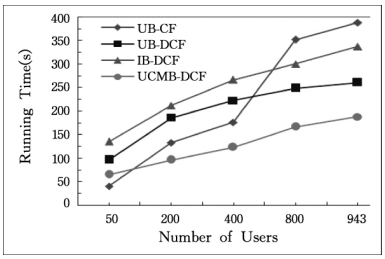


图 14 运行时间的比较

实验 4 不同推荐算法间的 MAE 比较

为了比较分布式算法和传统算法的推荐精确度,继续对

5 组实验数据进行测试，实验结果如图 15 所示。从图 15 中可以看出，在用户数量很少的情况下，UCMB-DCF 和 IB-DCF 的 *MAE* 值仍然比较大，这是因为数据比较稀疏时，计算得到的推荐值 R 波动性比较大，当用户数量增长时，R 值趋于稳定，最后在用户数量增大的情况下，*UCMB-DCF* 的精确度比 *IB-DCF* 略高；在所有实验组中，UB-DCF 的 MAE 值与 UB-CF 基本一致，反映了改进之后的 UB-DCF 与 UB-CF 在计算上的相似之处，并且 UB-DCF 和 UB-CF 的精确度均比 UCMB-DCF 和 IB-DCF 要高，这也说明了它们在计算相似度和推荐计算阶段的不同。

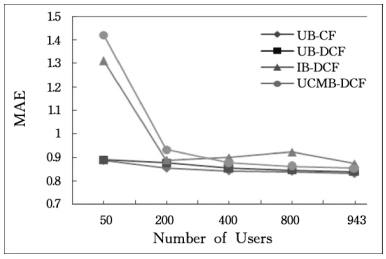


图 15　推荐精度 *MAE* 的比较

4.2.5　算法加速比分析

实验 5　UCMB-DCF 和 UB-DCF 的加速比分析

本节将对本文中的两个算法进行加速比分析。实验选用 MovieLens 100K Dataset 和 MovieLens 1M Dataset 两个数据集。如图 16、图 17 所示，随着节点数的增加，两个算法的加速比也在近似于线性增长，只是不同的算法在不同数据集下增长幅度不同。一方面，UCMB-DCF 对于 100K 的数据集，当其节点数增加到 3 个时，加速比增长明显，但随着节点数继续增加，加速比却增长缓慢；而对于 1M 的数据集，加速比却仍然继续增长，这是因为 100K 的数据量相对于 1M 而言比较少，在节点数增加的情况下，可并行计算的空间不足。另一方面，UB-DCF 对于 100K 的数据集，由于算法计算复杂度比 UCMB-DCF 高，因此仍然有并行计算的空间；最后，当数据规模增大时，加速比会更大。综上所述，本文中的两个算法能有效提高大数据规模下的推荐效率且具有良好的可扩展性。

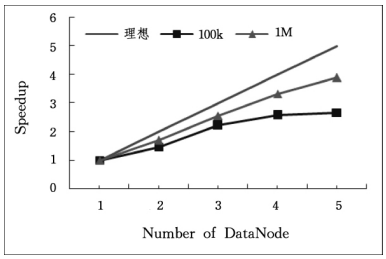


图 16　UCMB-DCF 的加速比曲线

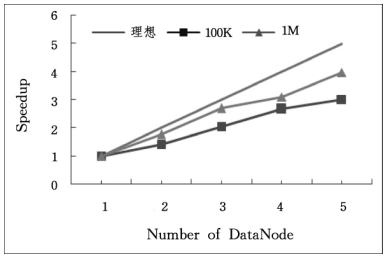


图 17　UB-DCF 的加速比曲线

结束语　本文提出一种分布式协同过滤推荐算法，解决传统推荐算法应用到大数据环境下面临的推荐精度低、推荐时延长以及网络负载重等问题。本文首先提出基于用户共现矩阵乘子推荐策略，将用户间的相似度矩阵与项目的评分矩阵相乘得到用户对项目的预测评分矩阵，从而得到对每个用户的候选推荐项目集。该策略适合大规模的分布式计算框架，可以使用 MapReduce 范式实现。在此基础上，本文进一步对计算用户相似度和预测评分两方面进行改进和扩展，提出了一种基于用户的分布式协同过滤推荐算法。最后本文设计了序列组合的 MapReduce 模式，将多个 MapReduce 子任务串联起来，自动地完成顺序化的执行。实验结果表明，在大数据环境下，该方法具有良好的推荐精度和推荐效率。下一步的研究工作将重点研究 Hadoop 平台与 Spark 平台下推荐算法的性能对比，通过 RDD 计算模式来对分布式协同过滤算法的性能进行进一步优化。

参 考 文 献

[1] Resnick P, Varian H R. Recommender Systems [J]. Communications of the ACM, 1997, 40(3): 56-58

[2] Goldberg D, Nicholas D, Oki B M, et al. Using Collaborative Filtering to Weave an Information Tapestry [J]. Communications of the ACM, 1992, 35(12): 61-70

[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 Transactions on Knowledge and Data Engineering, 2005, 17(6): 734-749

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

[5] 孟晓峰，慈祥. 大数据管理：概念、技术与挑战[J]. 计算机研究与发展，2013，50(1)：146-169

[6] Hadoop [EB/OL]. [2016-05-18]. http://hadoop.apache.org/index.html

[7] Chiky R, Ghisloti R, Kazi-Aoul Z. Development of a distributed recommender system using the Hadoop framework [C]// Proc of EGC. 2012: 495-500

[8] Jiang Jing, Lu Jie, Zhang Guang-quan, et al. Scaling-up item-based collaborative filtering recommendation algorithm based on Hadoop [C]// Proc of SERVICES 2011. Piscataway, NJ: IEEE, 2011: 490-497

[9] De Pesemier T, Vanheke K, Dooms S, et al. Content-based recommendation algorithms on the Hadoop MapReduce framework [C]// Proc of WEBIST 2011. New York: ACM, 2011: 237-240

[10] Kupisz B, Unold O. Collaborative filtering recommendation algorithm based on Hadoop and Spark [C]// IEEE International Conference on Industrial Technology. IEEE, 2015: 1510-1514

[11] Fan L, Li H, Li C. The improvement and implementation of distributed item-based collaborative filtering algorithm on Hadoop [C]// Proc of 34th Chinese Control Conference. IEEE, 2015: 9078-9083

[12] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[C]// Porc of OSDI 2004. Berkeley, CA: USENIX Association, 2004: 137-150

[13] 冯璐，冷伏海. 共词分析方法理论进展[J]. 中国图书馆学报，2006：32(2)：88-92

[14] Breeese J S, Hecherman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering [C]// Proc of 14th conference on Uncertainty in artificial intelligence. San Francisco: Morgan Kaufmann, 1998: 43-52