基于 MapReduce 的矩阵分解推荐算法研究

张 宇[1] 程久军[2]
(同济大学电子与信息工程学院 上海 201804)[1]
(同济大学嵌入式系统与服务计算教育部重点实验室 上海 201804)[2]

摘 要 矩阵分解是近几年提出的一种协同过滤推荐技术,但其每项预测评分的计算都要综合大量评分数据,同时在计算时还需要存储庞大的特征矩阵,用单一结点来进行推荐将会遇到计算时间和计算资源瓶颈。结合 MapReduce 分布式计算框架和矩阵分解推荐算法,设计了一种基于 MapReduce 的矩阵分解推荐算法来解决该问题,利用 Hadoop 的分布式缓存技术和 MapFile 文件结构解决了大特征矩阵在多结点间的高效共享问题并实现了多正则因子的并行处理。通过在 Netflix 数据集上的实验表明,该 MapReduce 算法及数据存储方案能带来较高的加速比,从而提高了推荐算法的计算效率。

关键词 协同过滤推荐,矩阵分解推荐,MapReduce,Hadoop
中图法分类号 TP391 **文献标识码** A

Study on Recommendation Algorithm with Matrix Factorization Method Based on MapReduce

ZHANG Yu[1] CHENG Jiu-jun[2]
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)[1]
(Key Laboratory of Embedded System and Service Computing of Ministry of Education, Tongji University, Shanghai 201804, China)[2]

Abstract Matrix factorization is a collaborative filtering recommendation technique proposed in recent years. In the process of recommendation, each prediction depends on the collaboration of the whole known rating set and the feature matrices need huge storage. So the recommendation with only one node will meet the bottleneck of time and resource. A MapReduce-based matrix factorization recommendation algorithm was proposed to solve this problem. The big feature matrices were shared by Hadoop distributed cache and MapFile techniques. The MapReduce algorithm could also handle multi-λ situation. The experiment on Netflix data set shows that the MapReduce-based algorithm has high speedup and improves the efficiency of collaborative filtering.

Keywords Collaborative filtering recommendation, Matrix factorization, MapReduce, Hadoop

1 引言

协同过滤推荐(collaborative filtering recommendation)[1]是推荐技术中运用最为成功的技术之一,基于潜在因子模型(latent factor models)[2]的推荐算法是一类更加精确高效的协同过滤推荐技术。矩阵分解是其中最常用的方法,该方法在文献[3]中首次提出,它与其他方法结合的混合推荐算法最终获得了 Netflix 推荐算法比赛大奖。此后,很多混合推荐算法[4,5]中都包含了该方法。

可是,矩阵分解推荐算法中的每项预测评分也要综合整个已有的评分集信息,因此,在用户数与项目数增长的情况下,算法的计算量也迅速膨胀。同时在矩阵分解推荐算法中,为了提高推荐准确度,往往选取较大的特征值,使得计算量更大,只靠单一结点无法满足这样的计算需求[6],因此,需要多台机器的并行处理。

MapReduce[7]是由 Google 公司针对海量数据处理提出的一种分布式编程模型。在该模型中,整个计算过程分为 Map 和 Reduce 两个阶段,每个阶段的输入输出都是形如(key,value)形式的键值对。Map 阶段的输入输出为:

$Map:\langle K1,V1\rangle \rightarrow list\langle K2,V2\rangle$

Reduce 阶段的输入输出为:

$Reduce:\langle K2,list(V2)\rangle \rightarrow list\langle K3,V3\rangle$

框架用户只需编写 Map 和 Reduce 阶段输入输出之间的映射函数,任务装载、调度和结点间的通信由框架自动完成。Hadoop[8]是 MapReduce 的开源实现。有关 MapReduce 上的一些机器学习和推荐算法的实现可参考文献[9]。

本文将设计一种基于 MapReduce 的矩阵分解推荐算法。第 2 节给出推荐系统问题的形式化描述并介绍矩阵分解推荐算法;第 3 节介绍所设计的 MapReduce 算法的数据结构与 4 个步骤;最后给出在 Netflix 数据集上的实验结果与分析。

2 推荐系统问题描述及矩阵分解推荐算法

2.1 推荐系统问题描述

在协同过滤推荐中,推荐问题可描述如下:

到稿日期:2012-05-31 返修日期:2012-09-08 本文受国家科技支撑计划项目(2012BAH15F03)资助。
张 宇(1987—),男,硕士生,主要研究方向为并行与分布式计算等,E-mail:timzhangyu@163.com;程久军 男,副教授,主要研究方向为移动计算、物联网等。

存在 N 个用户，编号 $u\in(1,2,\cdots,N)$，同时存在 M 个项目，编号 $i\in(1,2,\cdots,M)$，r_{ui} 表示用户 u 对于项目 i 的评分，这样用户对项目的评分就可构成评分矩阵 $R^{N\times M}$，其行标为用户编号 u，列标为项目编号 i。由于在系统中每个用户只对很少的项目进行过评分，因此该评分矩阵为一稀疏阵，协同过滤推荐的目标就是要通过已有的用户对于项目的评分来预测矩阵中那些空缺的评分 $\hat{r}_{ui}$。一般可由均方根误差(root mean squared error)来衡量推荐的准确度：

$$RMSE=\sqrt{\frac{1}{|\mathcal{R}|}\sum_{(u,i)\in\mathcal{R}}(\hat{r}_{ui}-r_{ui})^2} \tag{1}$$

式中，$\mathcal{R}$ 表示已知的用户对项目的评分集合，$\hat{r}_{ui}$ 是通过推荐算法预测的用户 u 对项目 i 的评分，而 r_{ui} 是已知的用户 u 对项目 i 的评分。均方根误差越小，说明推荐的准确度越高。

2.2 矩阵分解推荐算法

针对 2.1 节对于推荐系统问题的形式化定义，再设存在如下两个矩阵：用户特征矩阵 $P^{N\times K}$ 和项目特征矩阵 $Q^{K\times M}$，其中 K 为特征数，其编号 $k\in(1,\cdots,K)$。则 P 的每一行为用户 u 的特征向量，由 p_u 表示，$u\in(1,\cdots,N)$；Q 的每一列为项目 i 的特征向量，由 q_i 表示，$i\in(1,\cdots,M)$，这样评分矩阵 R 中某个评分的预测值可由式(2)表示：

$$\hat{r}_{ui}=\sum_{k=1}^{K}p_{uk}q_{ki}=p_uq_i^{\mathrm{T}} \tag{2}$$

计算预测评分值的问题便转化为了通过已知评分集 $\mathcal{R}$ 学习用户特征矩阵 P 和项目特征矩阵 Q 的问题。目前常用的方法有两种：随机梯度下降法(Stochastic gradient descent)[10]和交替最小二乘迭代法(Alternating least squares)。采用交替最小二乘迭代方法进行特征矩阵的学习在文献[6，11]中提出，本文也将采用该方法。

3 基于 MapReduce 的矩阵分解推荐算法

3.1 特征矩阵的数据结构

首先将已知评分集分为训练集 $\mathcal{R}_1$(80%)和测试集 $\mathcal{R}_2$(20%)。训练集用来学习特征矩阵，而测试集用来判断特征矩阵学习过程的截止条件并确定合适的学习参数(正则因子 λ)。数据集每一项皆为(i,u,r)三元组，i 为项目号，u 为用户号，r 为对应评分值。

利用训练集 $\mathcal{R}_1$ 初始化项目特征矩阵 Q，

$$q_{0i}=\frac{\sum_{(u,i)\in\mathcal{R}_1}r_{ui}}{n_i},1\leqslant i\leqslant M \tag{3}$$

式中，r_{ui} 为训练集中用户对项目的评分。

$$q_{ji}=\mathrm{random}(0.01,0.001),0<j<k,1\leqslant i\leqslant M \tag{4}$$

式中，q_{ji} 表示项目 i 的第 j 个特征值。

最小二乘迭代法学习特征矩阵时会利用岭回归(ridge regression)方法[11]计算每个特征向量的值，需要设定最合适的正则因子 λ。本文的 MapReduce 算法在初始化时设定多个 λ，并行学习这些正则因子对应的特征矩阵，最后选择在测试集上 RMSE 值最小的 λ 及其特征矩阵进行最后的预测评分。因此，项目特征矩阵初始化时输入多个 λ，最终，项目特征矩阵中的每个特征向量的存储结构变为：

$\langle(i,\lambda_a),(q_{0i},q_{1i},\cdots,q_{ji})\rangle$

式中，i 为项目号，$a<num$，num 为 λ 值的数量，j 为特征号。

以上项目特征向量键值对及之后相同格式，以及(u,λ)为关键字的用户特征向量键值对均以 MapFile[8] 结构存储。MapFile 格式分为索引文件与内容文件两部分，内容文件中的键值对按关键字有序存放，索引文件指示内容文件中若干键值对在文件中的偏移量。这样每次根据关键字查找对应特征向量的 $Search(i,\lambda,Q)$ 和 $Search(u,\lambda,P)$ 操作可采用二分查找，复杂度为 $O(N\log|\lambda||U|)$或是 $O(N\log|\lambda||I|)$，N 为文件数，可认为是一常量，$|\lambda|$为设定的正则因子数量，$|I|$为项目数，$|U|$为总的用户数，且查找只需 N 次磁盘 IO。

特征矩阵通过 Hadoop DistributedCache[8] 技术共享为每个计算结点的本地磁盘文件，避免了利用配置文件或是直接加载到内存共享时所造成的用户、项目、特征数较大以及多 λ 情况下的内存瓶颈。

3.2 算法 4 个阶段的描述

基于 MapReduce 的矩阵推荐算法分为以下 4 个 MapReduce 作业。

3.2.1 用户特征矩阵更新(P-Update)

在该步骤中，将项目特征矩阵 Q 作为初始已知条件，在各计算结点间共享，然后通过输入训练集 $\mathcal{R}_1$ 训练得到以 MapFile 结构存储的用户特征矩阵 P，具体算法及输入输出的形式如算法 1 所示。

算法 1 P-Update

输入：测试评分集 $\mathcal{R}_1$，项目特征矩阵 Q，λ 值列表$(\lambda_0,\lambda_1,\cdots,\lambda_n)$(按升序排列)，特征数 k

输出：用户特征矩阵 P

```
class MAPPER
  method MAP(line_key,(i,u,r))
    EMIT(u,(i,r))
class REDUCER
  method REDUCE(u,[(i0,r0),…(in,rn)])
    Qu_{n×k}
    for λ in (λ0,λ1,…,λn) do
      for x in (0,n) do
        [Qux1,…,Quxk]←Search(ix,λ,Q)
        R_{k×k}←(λ×n×I+Qu^T Qu)^{-1}
        for x in (0,k) do
          pux ← Σ_{z=0}^{n} (Σ_{y=0}^{k-1} Rxy Quyz) × rz
        EMIT((u,λ),[pu0,pu1,…,puk])
```

P-Update 过程的复杂度集中在了 Reduce 阶段利用岭回归计算用户特征向量的过程中。每次 Reduce 操作的复杂度为：$O(K^3+2K\cdot n_u+\log|\lambda||I|)$，其中，$|I|$为项目数，$n_u$ 为该用户评分过的项目，K 为特征数，$|\lambda|$为设定的正则因子数量。考虑到评分矩阵的稀疏性，$n_u<<|I|$，可认为是一常量，所以该算法在 MapReduce 环境下的复杂度为 $O\left(\frac{|\lambda||U|\cdot(K^3+2K+N\log|\lambda||I|)}{N}\right)$，$|U|$为总的用户数，$N$ 为 Reduce 结点数。

3.2.2 项目特征矩阵更新(Q-Update)

该过程的算法及复杂度与 P-Update 的算法类似，只是这次以 3.2.1 节计算出的用户特征矩阵 P 为已知条件，以训练集 $\mathcal{R}_1$ 作为输入，执行一次 Reduce 则计算项目特征矩阵 Q 的一个列向量。

3.2.3 RMSE 计算(RMSE Calculation)

利用 3.2.1 节和 3.2.2 节中得到的项目特征矩阵 Q 和用

户特征矩阵 P，可通过式(2)计算预测评分，结合测试集 $\mathcal{R}_2$，根据式(1)计算 RMSE 值，判断合适的 λ 值及迭代截至条件。具体的 MapReduce 算法如算法 2。

算法 2 RMSE Calculation

输入：测试集 $\mathcal{R}_2$，用户特征矩阵 P，项目特征矩阵 Q，λ 值列表(λ_0，λ_1，…，λ_a)(按升序排列)，特征数 k

输出：每个 λ 对应的 RMSE

```
class Mapper
  method MAP(line_key,(i,u,r))
    p_{k×1},q_{k×1}
    for λ in (λ_0,λ_1,…,λ_a) do
      [p_1,p_2,…,p_k]=Search(u,λ,P)
      [q_1,q_2,…,q_k]=Search(i,λ,Q)
      r_c ← Σ_{x=1}^{k} p_x q_x
      ErrSqu ← (r_c − r)^2
      EMIT (λ,ErrSqu)
class Reducer
  method Reduce(λ,[ErrSqu_1,ErrSqu_2,…,ErrSqu_n])
    RMSE ← sqrt((Σ_{x=1}^{n} ErrSqu_x)/n)
    EMIT(λ,RMSE)
```

该步骤的计算复杂度集中在了 Map 阶段，其复杂度为 $O\left(\frac{|\lambda||R_2|\cdot(K+N\log|\lambda||I|+N\log|\lambda||U|}{N}\right)$，$|R_2|$ 为验证集规模。

3.2.4 计算预测评分值(Prediction)

迭代执行 3.2.1－3.2.3 节，当在 3.2.3 节计算出的 RMSE 不再减少或是经历过足够次数的迭代后，选取 RMSE 最小的正则因子 λ 及其特征矩阵，以类似于算法 2 的方法计算预测评分，但在 Search(u,λ,P)和 Search(i,λ,Q)时只选择所选定 λ 对应的向量，此时输入集则变为了待评分集，输出以(u,i)为关键字的评分结果或是以 u 为关键字的每个用户评分最高的几个项目的编号。

4 实验结果与分析

根据上述研究成果，利用 4 台配备了 Intel Core 双核、主频 3.00GHz、内存 2GB、磁盘大小 160GB 的计算机来搭建 MapReduce 集群，其中 1 台计算机作为 Namenode 和 Jobtracker，其余 3 台负责具体的计算任务。

4.1 各阶段加速性能测试

选择 λ 为 0.01～0.10、间隔为 0.01 的 10 个值，特征数 k＝400。然后从 Netflix 数据集[6](其中的评分值为 1～5)中抽取了两个数据集，分别为 3554 个项目和 3408 个用户(2.7MB)、初始项目特征矩阵 111.1MB，P-Update 后的用户特征矩阵 105.7MB；10640 个项目和 23512 个用户(27.2 MB)、初始项目特征矩阵 385.0MB，P-Update 后的用户特征矩阵 846.3MB。

最后，绘制出了两个数据集在 P-Update、RMSE Calculation 这 2 个作业中增加计算结点时的加速比[12]曲线(另两个步骤分别与以上两个步骤相似，不再重复绘制)。从图 1 可以看到在 P-Update 作业中，计算结点从 1 增加到 6，对于两个数据集，计算时间都明显缩短，加速比增长明显(虽然与线性加速比有差距)，这说明 MapReduce 算法通过增加计算结点能明显减少计算时间。但是，在 RMSE Calculation 中根据之前的复杂度分析可见，在数据集较小的情况下其本身计算复杂度小，而增加计算结点却增加了通信、任务调度等额外开销，因此在小数据集(2.7MB)下其加速比不理想。但是在数据集规模较大(27.2MB)的情况下，即使是在 RMSE Calculation 上其加速性能也较高。另外，整个算法的计算主要集中在了 P-Update 和 Q-Update 作业的迭代操作，所以整个 MapReduce 算法的总体加速性能较高。在 k＝100 且已知评分集为前述大数据集的情况下，经过 10 次迭代后在测试集上得到各 λ 对应的 RMSE、运行时间和最终的加速比曲线如图 2 所示。

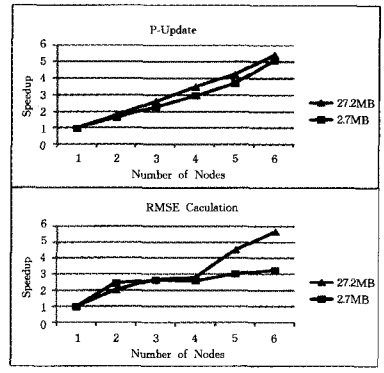

图 1 P-Update 和 RMSE Calculation 步骤的加速比曲线

Lambda	RMSE
0.01	0.9797
0.02	0.9532
0.03	0.9428
0.04	0.9308
0.05	0.9238
0.06	0.9204
0.07	0.9216
0.08	0.9229
0.09	0.9338

结点数	总时间(s)
1	72050
2	40977
3	26204
4	23254
5	20510
6	17446

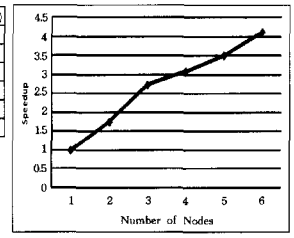

图 2 k＝100 经 10 次迭代后的 RMSE、总时间和加速比曲线

可以看到，经过 10 轮迭代后同时得到的 RMSE 中，λ＝0.06 对应的 RMSE 最小，因此，选择 λ＝0.06 及其对应的特征矩阵来进行最后的预则评分。在增加计算结点的情况下，整个过程仍保持着较高的加速比。

4.2 特征矩阵共享带来的磁盘 IO 开销分析

引入 MapFile 共享特征矩阵解决特征矩阵过大(特别是在多 λ 情况下)而无法全部加载入内存的问题，但是每次的 Search(i,λ,Q)和 Search(u,λ,P)将带来额外的磁盘 IO 开销。为了测定这项额外的磁盘 IO 的代价，在 4.1 节 k＝100 的大数据集两结点的 P-Update 过程的 Search(i,λ,Q)查找操作中增加了一次对某一随机特征向量的查找，从而增加了一次磁盘 IO 来观察所增加的时间 ΔT，可认为额外磁盘 IO 时间开销 $T_{io}=2T_{io}-T_{io}\approx\Delta T$。

经过 10 次求平均值的实验，发现 $T_{io}\approx\Delta T$＝1mins，43sec，占总时间开销 T(35mins，53sec)的 4%，比值较小，因此，通过 MapFile 实现各计算结点间的特征矩阵的共享在用户、项目及特征较大的情况下并不会明显降低算法的性能。

结束语 本文提出了一种基于 MapReduce 的矩阵分解推荐算法，解决了大量用户和项目组成的特征矩阵在各计算结点间的高效共享并可实现矩阵分解推荐算法中多正则因子的并行处理。实验结果表明，基于 MapReduce 的矩阵分解推荐算法能带来较高的加速比，为用户与项目急速增加情况下

(下转第 36 页)

的 DV-Hop 算法，本文算法在两种条件下定位误差平均减小了 19.2%和 11.76%。

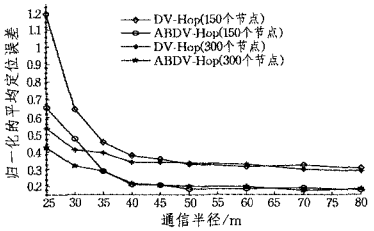

图 4　两种算法在不同通信半径下的定位误差

4.4　不同节点数条件下定位结果的比较

设初始锚节点个数分别为 20 和节点总数的 10%，通信半径为 40m，在仿真区域内随机布置从 50 到 400 变化的节点总个数，比较本文算法与 DV-Hop 算法归一化的平均定位误差，结果如图 5 所示。

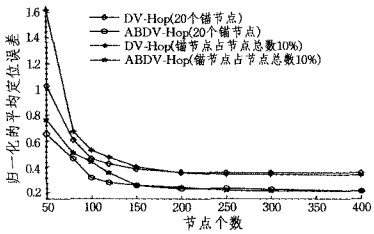

图 5　两种不同算法在节点数不同时的定位误差

从图 5 可知，随着节点个数的增加，两种算法的平均定位误差均逐渐减小并趋于稳定。当锚节点个数固定为 20 时，相比于节点总数 10%的锚节点，在节点数小于 200 时，定位误差较小，但随着节点数的增多，两种算法在占节点总数 10%的条件下，定位误差逐渐小于在锚节点固定为 20 个的情况。两种不同条件下，本文算法相比于 DV-Hop 算法定位误差分别平均减小了 16.18%和 20.7%。

结束语　本文通过对传统 DV-Hop 算法定位过程的分析，针对其定位阶段由于环境、测距误差等因素导致采用最小二乘法计算未知节点坐标时存在较大误差的问题，在分析人工蜂群算法原理的基础上，将其替代最小二乘法用于定位阶段，提出了基于人工蜂群算法的 DV-Hop 算法。该算法控制参数少、搜索精度高，且运行稳定，通过确定适应度函数，可以较快地寻求最优解。仿真实验表明，相比于传统 DV-Hop 算法，ABDV-Hop 算法在不同网络平均连通度、邻锚节点数量、通信半径及节点个数的情况下明显提高了定位精度。

参 考 文 献

[1] 尚志军，曾鹏，于海斌. 无线传感器网络节点定位问题[J]. 计算机科学，2004，31(10)：35-38

[2] 张佳，吴延海，石峰，等. 基于 DV-Hop 的无线传感器网络定位算法[J]. 计算机应用，2010，30(2)：323-326

[3] Nicolescu D, Nath B. Positioning in ad hoc networks[J]. Journal of Telecommunication System, 2003, 22: 667-280

[4] Lu Qing-ling, Bai Meng-liang, Zhang Wei, et al. A kind of improved DV-Hop Algorithm[C]// Proc. of the 2nd International Conference on Intelligent Control and Information Processing. Harbin, China: IEEE Press, 2011: 867-869

[5] Li Wen-wen, Zhou Wu-neng. Genetic algorithm-base localization algorithm for wireless sensor network[C]// Proc. of 2011 Seventh International Conference on Natural Computation. Shanghai, China, 2011: 2096-2099

[6] 姜钧，程良伦. 采用虚拟锚节点的高精度 VAD-Hop 定位算法[J]. 传感技术学报，2011，24(07)：1048-1051

[7] 王新生，赵衍静，李海涛. 基于 DV-Hop 定位算法的改进研究[J]. 计算机科学，2011，38(2)：76-78

[8] Karaboga D, Basturk B. A powerfull and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm[J]. Journal of Global Optimization, 2007, 39(3): 459-471

[9] Basu B, MAhanti G K. A competitive study of modified particle swarm optimization, differential evolution and artificial bees colony optimization in synthesis of circular array[C]// Proc. of Power, Control and Embedded System(ICPCES). 2010: 1-5

[10] 欧阳丹彤，何金胜，白洪涛. 一种约束粒子群优化的无线传感器网络节点定位算法[J]. 计算机科学，2011，38(7)：46-50

[11] 林金朝，陈晓冰，刘海波. 基于平均跳距修正的无线传感器网络节点迭代定位算法[J]. 通信学报，2009，30(10)：107-113

(上接第 21 页)

的协同过滤推荐算法的性能瓶颈问题提供了一种可行的解决方案。

参 考 文 献

[1] 许海玲，吴潇，李晓东. 互联网推荐技术比较研究[J]. 软件学报，2009，20(2)：350-362

[2] Koren Y, Bell R, Volinsky C. Matrix Factorization Techniques for Recommender Systems[J]. Computer, 2009, 42(8): 30-37

[3] Bell R M, Koren Y. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights[C]// Proc of the 7th IEEE International Conference on Data Mining. Omaha NE, USA: IEEE, 2007: 43-52

[4] Takacs G, Pilaszy I, Nemeth B, et al. Matrix Factorization and Neighbor Based Algorithms the Netflix Prize Problem[C]// Proceedings of the 2008 ACM conference on Recommender systems. Lausanne, Switzerland: ACM, 2008: 267-274

[5] 杨阳，向阳，熊磊. 基于矩阵分解与用户近邻模型的协同过滤推荐算法[J]. 计算机应用，2012，32(2)：395-398

[6] Zhou Y, Wilkinson D, Schreiber R, et al. Large-Scale Parallel Collaborative Filtering for the Netflix Prize[C]// Proc of the 4th international conference on Algorithmic Aspects in Information and Management. 2008

[7] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communication of the ACM 50th anniversary issue, 2008, 51(1): 107-113

[8] Hadoop. Open-source software for reliable, scalable, distributed computing[EB/OL]. http://hadoop.apache.org/, 2011

[9] Mahout. Scalable machine learning and data mining[EB/OL]. http://mahout.apache.org, 2011

[10] Takacs G, Pliaszy I, Nemeth B, et al. Investigation of Various Matrix Factorization Methods for Large Recommender Systems[C]// Proc of the IEEE International Conference on Data Mining Workshops. IEEE, 2008: 553-562

[11] Pilaszy I, Zibriczky D, Tikk D. Fast ALS-based Matrix Factorization for Explicit and Implicit Feedback Datasets[C]// Proceedings of the fourth ACM conference on Recommender systems. New York: ACM, 2010: 71-78

[12] 陈国良. 并行计算——结构、算法、编程(修订版)[M]. 北京：高等教育出版社，2003