基于 Hadoop 平台协同过滤推荐算法①

杨志文, 刘 波

(华南师范大学 计算机学院, 广州 510631)

摘 要: 针对协同过滤推荐算法在数据稀疏性及在大数据规模下系统可扩展性的两个问题, 在分析研究 Hadoop 分布式平台与协同过滤推荐算法后, 提出了一种基于 Hadoop 平台实现协同过滤推荐算法的优化方案. 实验证明, 在 Hadoop 平台上通过 MapReduce 结合 Hbase 数据库实现算法, 能够有效地提高协同过滤推荐算法在大数据规模下的执行效率, 从而能够进一步地搭建低成本高性能、动态扩展的分布式推荐引擎.

关键词: Hadoop; MapReduce; Hbase; 协同过滤推荐算法

Hadoop-Based Collaborative Filtering Recommendation Algorithm

YANG Zhi-Wen, LIU Bo

(School of Computer Science, South China Normal University, Guangzhou 510631, China)

Abstract: In order to solve data sparsity and scalability of the Collaborative Filtering (CF) recommendation algorithm when the volume of the dataset is very large. After deeply analyzing the Hadoop distributed computing platform and the characteristic of Collaborative Filtering recommendation algorithm, the paper propose a optimization scheme on Hadoop platform. The experimental results show that it can effectively improve the execution efficiency of Collaborative Filtering recommendation algorithm in large data size, when it is realized by MapReduce with Hbase database on the Hadoop platform.And then, it contribute to build one recommendation system which is low cost, high-performance and dynamic scalability.

Key words: Hadoop; MapReduce; Hbase; collaborative filtering recommendation algorithm

随着互联网的迅速发展, 人们如何在繁多的信息获取自己需要的内容成为了一个越来越重要的问题. 至 20 世纪 90 年代提出推荐系统概念以来, 对于推荐系统的研究得到了广泛的关注和飞速的发展. 其中, Goldberg 等人提出的协同过滤算法以其很强的普适性在各类个性化推荐系统中得到了广泛的重视和应用.

目前的大部分协同过滤推荐算法主要是通过计算某一用户对未评分项目的预测评分并以此作为主要依据向该用户进行推荐[1]. 根据算法采用最近邻对象的不同可分为基于用户(User-based)的协同过滤、基于项目(Item-based)的协同过滤和基于模型(Model-based)的协同过滤三种算法. 但协同过滤推荐算法存在着与生俱来的稀疏性问题、冷启动问题和可扩展性等缺陷, 一直无法得到有效的解决[2]. 迄今为止很多学者致力于采用各种方法对基本的协同过滤算法本身进行改进. 如文献[3]利用神经网络预测用户未评分项的评分从而降低数据的稀疏性, 文献[4]提出基于项目间相似性提高协调过滤推荐算法的可扩展性. 然而随着评分数据规模的急剧扩大, 庞大的计算量将严重影响协同过滤推荐算法的执行效率与推荐效果.

针对协同过滤推荐算法数据稀疏性和可扩展性问题, 本文将跳开对推荐算法本身的改进, 在实现机制上提出一种基于 Hadoop[5,6]分布式平台实现协同过滤推荐算法的改进方案, 即使用对稀疏数据具有良好支持的分布式数据库 Hbase 来保存用户评分矩阵作为数据源, 在此基础上实现 MapReduce 框架下协同推荐算法的分布式计算, 将计算任务分配给 Hadoop 集群内的每台机器, 从而有效地提高荐算法的执行效率.

① 收稿时间:2012-12-16;收到修改稿时间:2013-01-14

1 协同过滤推荐算法与Hadoop平台

1.1 协同过滤推荐算法

传统的基于项目(Item-based)的协同过滤推荐算法为例, 它基于这样一个假设[7]: 如果用户对一些项目的评分比较相似, 则他们对其他项目的评分也比较相似. 通过统计技术搜索目标用户的若干最近邻居, 然后根据最近邻居对项目的评分预测目标用户对项目的评分, 产生对应的推荐列表. 其具体算法的工作流程一般可以分为三步:

1) 数据表述

输入数据一般被表述为一个 m×n 的用户-项目评估矩阵 R, 如图 1 所示, m 是用户数, n 是项目数, 矩阵元素 Rij 表示第 i 个用户对第 j 个项目的评估值.

$$\begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ R_{21} & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ R_{m1} & R_{m1} & \cdots & R_{mn} \end{pmatrix}$$

图 1 用户-项目矩阵

2) 最近邻搜索[8](Nearest neighbor search, NNS)

得出目标项目的最近邻集合, 即计算项目 I 与其它项目的相似度, 根据相似度的大小进行排序从而确定目标项目的最近邻, 比较常用的方法有余弦相似度或者皮尔逊相关系数等, 以余弦相似性为例.

$$sim(i_x,i_y)=\frac{\sum_{i\in U_{xy}} R_{ix}\times R_{iy}}{\sqrt{\sum_{i\in U_x} R_{iy}^{\ 2}}\ \sqrt{\sum_{i\in U_y} R_{iy}^{\ 2}}} \quad (1)$$

在评分矩阵中, 每个项目的所有评分可以视为这个矩阵的一个列向量, 计算两个项目的相似度就可以通过计算两个项目对应的两个列向量之间的余弦值, 用这个余弦值来表示这两个项目的相似度. $sim(i_x,i_y)$表示项目 i_x 和项目 i_y 之间的相似度, U_{xy} 表示评价过 i_x 与 i_y 的用户集合, U_x 和 U_y 分别表示评价过项目 i_x 和项目 i_y 的用户集合, R_{ix} 和 R_{iy} 分别表示 u_i 对项目 i_x 和 i_y 的评分.

3) 得出目标用户的推荐结果

根据目标用户的最近邻用户的所有评分项, 预测其对项目 i_y 的评分 P_{ay}, 其中 N_y 表示目标项目的最近邻, r_{ai} 表示目标用户对项目 i_i 的评分.

$$P_{a,y}=\frac{\sum_{i\in N_y} sim_{y,i}\times r_{ai}}{\sum_{i\in N_y}|sim_{y,i}|} \quad (2)$$

得出用户对所有项目的预测评分后, 根据 P_{ay} 的大小顺序排列选择最佳的推荐结果给目标用户.

1.2 Hadoop 平台[9]

Hadoop 是当前热门的云计算解决方案之一, 是 Apache 组织的一个开源的分布式计算平台, 以 Hadoop 分布式文件系统(HDFS)和 MapReduce 为核心为用户提供了系统底层细节透明的分布式基础架构. 从而用户可以利用 Hadoop 轻松地组织计算机资源, 搭建自己的分布式计算平台, 并且可以充分利用集群的计算和存储能力, 完成海量数据的处理. 现在 Hadoop 已经发展成为包含了多个子项目的集合, 它们提供互补性服务或在核心层上提供了更高层的服务.

MapReduce 是 Google 提出的一个软件编程框架, 用于大规模数据集的并行运算. 如图 2 所示: 一个 MapReduce 作业执行时会被拆分为 Map 和 Reduce 两个阶段, Map 阶段对输入数据进行分布式处理映射成一组新的键值对<key, value>, 经过一定处理后交给 Reduce 阶段, Reduce 对相同 key 下的所有 value 进行处理后再输出键值对作为最终的结果.

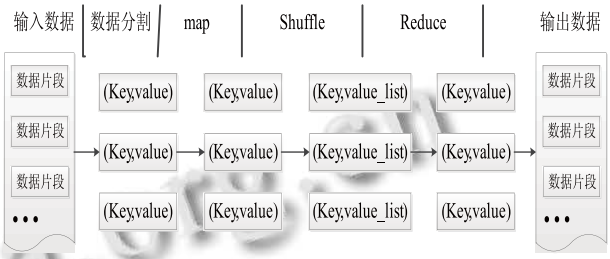

图 2 MapReduce 作业流程图

2 推荐算法的实现

如上所述, 对于基于项目的协同过滤推荐算法的主要计算集中在项目相似度计算和预测评分计算两个阶段. 因此, 下面分别给出这两个阶段的算法过程描述和 MapReduce 框架下实现方法.

2.1 相关 Web 页面获取

算法 1 项目间的相似度计算

输入: 项目集合 I、用户集合 U、评分矩阵 $R_{U\times I}$

输出: 项目间的相似度矩阵 $SIM_{I\times I}$

第 1 步: 设 i=1;

第 2 步: 若 i>=|I|, 结束;

第 3 步: i++, 若 i>|I|, 则跳转至第 2 步;

第 4 步: 从评分矩阵里面找到表示评价过 i_x 与 i_y 的用户集合 U_{xy}, 分别表示评价过项目 i_x 和项目 i_y 的用户集合 U_x 和 U_y, 分别表示 ui 对项目 i_x 和 i_y 的评分 R_{ix} 和 R_{iy};

第 5 步: 根据公式(1)计算项目 i_x 与 i_y 的相似度 $sim(i_x,i_y)$;

第 6 步: i++, 跳转至第 3 步;

算法 2　预测评分计算

输入: 目标用户 ua、目标项目 iy、选取的最近邻数量、评分矩阵 $R_{U\times I}$、项目间相似度矩阵 $SIM_{I\times I}$

输出: 用户 u_a 对项目 i_y 的预测评分

第 1 步: 从项目相似度矩阵中选择 K 个与项目 i_y 相似度最高的项目组成最近邻集合 N_y;

第 2 步: 根据公式(2)计算用户 u_a 对项目 i_y 的预测评分 P_{ay};

2.2 MapReduce

以 Grouplens 网站[10]下载的数据视频数据 MovieLens 1M 为例, 包含的 3 个文件(movies.dat、ratings.dat 和 users.dat)分别存储了电影信息、评分信息以及用户信息. Rating.dat 的数据格式如下所示:

UserID::MovieID::Rating::Timestamp

- UserID 表示用户 ID, 从 1 至 6040.
- MovieID 表示电影 ID, 从 1 至 3952.
- Rating 表示用户对电影的评级, 从 1 到 5 共 5 个等级.
- Timestamp 表示用户给电影评分的时间戳.

Hadoop 是用 JAVA 语言实现, 然而它的基本数据类型不是标准的 JAVA 对象, 而是对它们的一个封装, 序列化. 根据第二节对协同过滤推荐算法的分析, org.apache.hadoop.io 包中提供了一套可用于序列优化的 Writable 基本类型并不能满足需求. 所以首先需要通过实现 Writable 接口和 WritableComparable 接口定制灵活合适的 Writable 类型.

然后根据算法实现的三个步骤如图 3 所示对应地分成三个 MapReduce 任务进行联合计算. 这里需要将 Reduce 默认的 TextOutputFormat 纯文本格式输出改成 SequenceFileOutputFormat 二进制输出格式. 以便直接通过它里面存放的键值对的数据格式读取数据, 而不需要像纯文本文档输入时对每行数据进行格式分析, 减少了大量中间数据的存储, 适合多个 MapReduce 任务组成一个作业链, 提高执行效率.

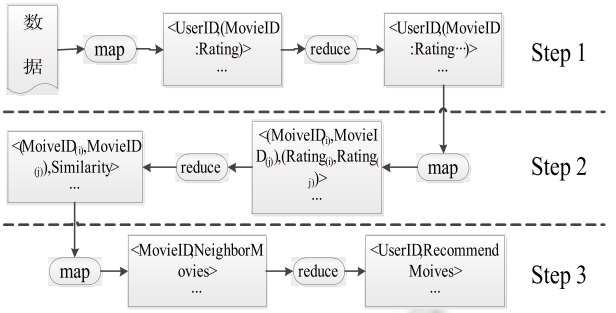

图 3　算法流程图

Step1: 得到用户-项目评分矩阵. 针对 MapReduce 并行程序设计原则可知, 输入数据的切分步骤与数据不相关, 可以并行化处理, map 阶段接收输入的<key, value>(key 是当前输入的行号, value 是对应行的内容), 然后对此行内容进行切分工作, shuffle 排序聚集分发都是按照 key 值进行的, map 的输出设计有 UserID 作为 key, MovieID 和 Rating 作为 value 输出. Reduce 阶段接收到用户对每个 MovieID 的评分后合成用户-项目评分矩阵.

Step2: 根据算法 1 计算出所有项目 MovieID 间的相似度. map 阶段接收到用户项目评分矩阵后, 对每个用户下的项目评分进行提取, 以项目对(MoiveID(i), MovieID(j))作为 key, 项目对应的(Rating(i), Rating(j))作为 value 输出给 reduce 阶段进行项目间相似度的计算, 并将计算结果保存输出.

Step3: 根据所有项目 MovieID 间的相似度, 得出每个项目 MovieID 相似度最高的 N 个项目定义最近邻居. 而后, 根据算法 2 计算出每个用户 UserID 对未评分的项目 MovieID 的预测评分, 通过对评分的排序, 得出推荐项目结果返回给用户.

3　数据源的优化

Hadoop 平台支持文本数据以及 RDBMS 数据库(如 MySQL)作为输入数据源. 但文本输入时需要对数据进行格式分析和序列化处理, 占用了大量的计算时间. 而传统的 RDBMS 数据库在数据量过大时会出现读写分离策略. 考虑到协同过滤推荐算法计算数据评分矩阵的稀疏性(一般用户评价信息所涉及的项目只能占总数的 1%～2%左右), 本文引用 Hadoop 平台项目组下对稀疏性数据具有良好支持的 HBase 分布式数据库作为数据源.

HBase[11]是一个分布式、面向列的开源数据库, 在 Hadoop 之上提供类似 Bigtable 的能力. 不同于一般关系数据库的数据模型, 用户将数据存储在一个表里, 一个数据行拥有一个可选择的键和任意数量的列. 主要用于需要随机访问、实时读取的大数据. 相比文本数据及 RDBMS 数据库, HBase 数据库有以下优势.

HBase 是一个基于列模式的映射数据库, HTable 为 null 的 Column 不会被存储, 这样既节省了空间又提高了读性能, 很适宜存储松散型数据.

HBase 架构在 Hadoop 上, 不仅具有很好的可收缩性, 当数据越来越大时, 只要扩展 Hadoop 集群, HBase 会自动水平切分扩展, 跟 Hadoop 的无缝集合保障了其数据库可靠性和海量数据分析的高性能.

HBase 能够结合使用 MapReduce 编程框架并行处理大规模数据, 使得数据存储与并行计算完美地结合在一起[12].

如表 1, 使用 HBase 数据库需要对原先 MapReduce 程序进行修改. 扩展(继承)MapReduce 里面的相关类(如 Mapper 和 Reducer, InputFormat 和 OutputFormat)来方便 MapReduce 直接读写 HTable 中的的数据.

表 1 Hbase 扩展包

Hbase MapReduce	Hadoop MapReduce
org.apache.hadoop.hbase.mapreduce.TableMapper	org.apache.hadoop.mapreduce.Mapper
org.apache.hadoop.hbase.mapreduce.TableReducer	org.apache.hadoop.mapreduce.Reducer
org.apache.hadoop.hbase.mapreduce.TableInputFormat	org.apache.hadoop.mapreduce.InputFormat
org.apache.hadoop.hbase.mapreduce.TableOutputFormat	org.apache.hadoop.mapreduce.OutputFormat

根据数据源 MovieLens 1M 的数据特性, 设计基于 HBase 数据库的数据模型如表 2. 将数据写入 HBase 后作为底层“存储结构”, MapReduce 在上进行调用处理数据, 可以有效提高程序的执行效率.

4 实验结果与分析

4.1 实验环境

Hadoop 集群: 4 台普通 PC 机搭建 Hadoop 组成集群, 命名为 hd0～hd3, 其中 hd0 作为 NameNode 同时作为 JobTracker, 其它 hd1～hd3 作为 DataNode 和 TaskTracker.

软件环境: Ubuntu10.10; JDK1.6; Hadoop0.20.203.0

表 2 数据模型

Row Key	Column Family		Column Family		Column Family	
	info	value	movie	value	timestamp	value
<User ID>	Info:Gender Info:Age Info:Occupation Info:Zip-code	the gender the age the occupation the zip code	Movie:<MovieID> …	<Rating>	Movie:<MovieID> …	<Timestamp>

4.2 实验数据

本文使用的是 Grouplens 网站上提供的视频评分数据包:

ml-1M: 数据包括了 6040 个用户对 3900 部电影.

ml-10M: 数据包括了 71567 个用户对 10681 部电影.

试验过程中, 根据节点数量分别启动 1～4 台 Task Tracker 节点, 构成不同规模的分布式集群环境. 对不同级别的数据量进行分组实验, 每次测试重复进行三次取平均值. 获取不同的数据量在不同规模 Hadoop 集群下所需要的计算时间 T. 由于计算时间受到集群各节点自身硬件水平及网络状况等影响, 并不具有稳定客观的参考价值, 而本文实验是为了验证通过对 Hadoop 集群的扩展能有效地提高超大数据规模下协同过滤推荐算法的执行效率, 所以我们采用参数 f=T1/Tn 作比较. T1 是 TaskTracker 节点为 1 时算法的执行时间, Tn 是 TaskTracker 节点为 n 时算法的执行时间. 通过比较 f 参数可以得出 Hadoop 节点数量对算法执行效率的影响状况.

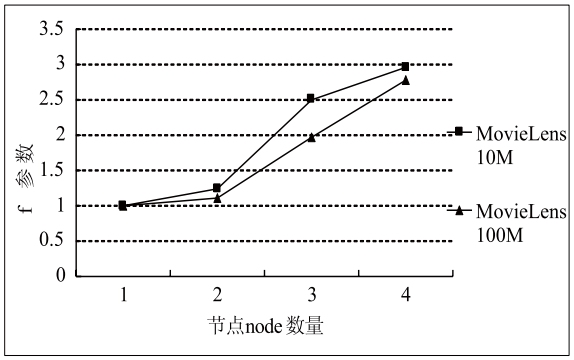

图 4

从实验结果图 4, 可以看出, 对于同一数据集, 增加 TaskTracker 节点数量, 可以有效地提高推荐算法的效率. 而且相比较下计算数据集较大, Hadoop 集群的大小影响越明显. 因为对Hadoop集群进行扩展只需要将新的 TaskTracker 节点地址添加到 Hadoop 的 slaves 列表中, 所以在 Hadoop 平台实现协同过滤推荐算法, 能够有效地克服算法本身数据稀疏性和可扩展性问题. 通过实验, 发现 MapReduce 框架下实现协同过滤推荐算法存在着一个缺点, 即在每次的计算中, 多需要对数据源进行初始化, 这个过程耗费一定的计算时间, 从而影响了推荐算法的实时响应性.

5 总结

本文提出了一种基于 Hadoop 平台对协同过滤推荐算法实现的研究, 试图通过云计算平台实现推荐算法的分布式计算, 从而克服算法本身数据稀疏性及可扩展性问题. 与基于单机实现协同过滤推荐算法的方法相比, 应用 Hadoop 云计算平台, 不仅可以使用更适合存储稀疏性数据的分布式数据库 HBase 代替传统关系数据库 RDBMS, 还能结合 MapReduce 编程框架实现算法的并行计算. 实验数据表明, 在采用多台 PC 机组成的 Hadoop 集群通过增加计算节点能够有效减少计算时间. 随着计算数据规模的增加, 可以廉价动态地扩展集群的计算能力.

参考文献

1 范波,程久军.用户间相似度协同过滤推荐算法.计算机科学, 2012(1).

2 Sarwar MB. Sparsity, scalability and distribution in recommender systems [Ph.D dissertation]. University of Minisota, 2001.

3 张锋,常会友.使用 BP 神经网络缓解协同过滤推荐算法的稀疏性问题.计算机研究与发展,2006:(4).

4 Sarwar B, Karypis G, Konstan Jet. Item-Based Collaborative Filtering Recommendation Algorithms C. Proc. of the 10th International World Wide Web Conference. New York 2001: 285–295.

5 The Apache Hadoop project.http://Hadoop.apache.org/.

6 Dean J, Ghemawat S. MapReduce: Simplified data processing on large cluseters. Communications of the ACM, 2005,51(1): 107–113.

7 Brees J, Hecherman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering. Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98). 1998:43–52.

8 季昀.基于协同过滤推荐算法电影网站的构建[学位论文]. 哈尔滨工业大学,2009.

9 White T,曾大聘,周傲英.Hadoop 权威指南.北京:清华大学出版社,2010.9–10.

10 Miller BN, Albert I, Lam S K, et al. MovieLens unplugged: experiences with an occasiionally connected recommender system. Proc. of the Conference on Human Factors in Computing Systems. 1995:210–217.

11 The Apache HBase project.http://hbase.apache.org/.

12 陆嘉恒.Hadoop 实战.北京:机械工业出版社.245–246.

(上接第 116 页)

6 Zhang ZY. Flexible Camera Calibration By Viewing a Plane From Unknown Orientations. Proc. of IEEE International Conference on Computer Vision, 1999,11(1):666–673.

7 Zhang ZY. MEMBER S.A Flexible New Technique for Camera Calibration. IEEE Trans. on Pattern Analysis And Machine Intelligence, 2000,22(11):1330–1334.

8 马颂德,张正友.计算机视觉.北京:科学出版社,1998.

9 Steve V. Binocular vision-based augmented reality system with an increased registration depth using dynamic correction of feature positions. Virtual Reality, Proc. IEEE, 2003: 271–272.

10 于舒春,朱延河,闫继红,等.基于新型双目机构的立体视觉系统标定.机器人,2007,29(4):353–362.

11 Heikkila J, Silven O. A Four-step Camera Calibration Procedure with Implicit Image Correction. Proc. of the IEEE Computer Society Conference on ComputerVision and Pattern Recognition. Los Alamitos, CA, USA: IEEE Computer Society, 1997:1106–1112.