中国科学技术大学

专业硕士学位论文

（专业学位类型）

**基于Spark网上商城推荐系统研究与实现**

作者姓名： 张宁

专业领域： 软件工程

校内导师： 戴礼荣 教授

企业导师： 程宁 博士

完成时间： 2017年9月13日

University of Science and Technology of China

A dissertation for master’s degree

（Professional degree type）



**Research and implementation of collaborative filtering recommendation system based on Spark**

Author： Ning Zhang

Speciality： Software Engineering

Supervisor：Lirong Dai

Advisor： Ning Cheng

Finished time: September 13,2017

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开 □保密（\_\_\_\_年）

作者签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 摘 要

在大数据时代的背景下，信息的挖掘过滤对于用户来说是十分重要的。大多数应用场景，由于资源信息的过于丰富以及用户对自身需求的不明确表达，用户往往很难从中找到自己需要的内容，这就是如今的信息过载的现象。目前缓解信息过载问题的方式有很多，推荐系统作为其中的一种方式在社交媒体中也变得越来越重要。系统通过计算提取用户或商品的特征信息，根据一定的算法规则，更加主动、智能的为用户推送可能感兴趣的产品信息，使得互联网信息由局部的高度紧密，转换成松散的随机可达，帮助用户找到自己需要的内容，提升互联网体验质量。

随着信息量的不断增加，以及推荐系统应用场景的多样性，推荐系统的实现遇到了各种各样的挑战。其一、是推荐信息的时效性，用户某一时刻产生的行为日志信息可能随着时间推移产生无效的推荐结果因此如何根据用户当前行为，为用户提供实时的推荐成为了关键问题之一。其二，对于新用户或商品，缺少必要的特征信息，如何进行推荐（冷启动问题）也成为了一项巨大挑战。

本文设计实现了基于Spark的实时计算的推荐系统。其主要内容如下：

1) 基于Spark分布式计算框架，根据需求，设计推荐系统基础架构。整个系统结构分为三层：一、实时流处理层，基于Spark Streaming对消息队列的数据进行实时流计算；二、离线计算层，用于处理耗时任务，为实时推荐提供数据基础；三、业务逻辑层，整合实时计算和离线计算资源。为上层结构提供推荐服务。

2) 本文分析并测试了基于Item的协同过滤推荐算法的性能，总结了影响该算法推荐性能的主要因素，并改进了基于矩阵分解的协同过滤算法的计算方式，采用异步分布式进行随机梯度矩阵分解算法，并进行相关测试实验，发现该计算方式可以很好的缩短算法的收敛时间。

3) 针对新用户而言，通过用户时间权重的大小，利用三分图的形式描述各个属性间的关系，进一步获取个性化的预测评分公式，从而解决推荐系统的冷启动问题。

**关键词：**信息过载,spark,推荐系统,协同过滤,冷启动

# ABSTRACT

In the background of big data age, the filtering of information is very important to the users. Most of the scenarios, because of the richness of resource information and the ambiguous expression of users' needs, it is very difficult for users to find the content they need, which is the phenomenon of information overload today. As one of the effective ways to alleviate the problem of information overload, recommender system plays an increasingly important role in social media. The system can extract the characteristic information of the user or the product, according to certain algorithm rules, more active and intelligent to push the product information which may be interested to the user, thus helps the user to find the content which is interesting from the massive data.

With the increasing amount of information and the diversity of recommender system application scenarios, the implementation of Recommender system meets various challenges. The first is the timeliness of the recommended information, the user's behavior log information generated at a certain moment may result in ineffective recommendations over time. Therefore, how to provide real-time recommendations to users according to the current behavior of users has become one of the key problems. Second, for new users or commodities, the lack of the necessary feature information, how to recommend (cold start problem) has become a huge challenge.

In this paper, a recommender system based on Spark is designed and implemented. Its main contents are as follows:

(1) Based on the Spark distributed computing framework, according to the requirements, design the recommended system infrastructure. The whole system structure is divided into three layers: first, real-time flow processing layer, based on spark streaming to the message queue data for real-time flow calculation; second, off-line computing layer provides the data base for processing time-consuming tasks and real-time recommendations; third, the business logic layer, the integration of real-time provide referral services for the upper structure.

(2) This paper analyzes and tests the performance of the collaborative filtering recommendation algorithm based on item, sums up the main factors that affect the recommendation performance of the algorithm, improves the calculation method of collaborative filtering algorithm based on matrix decomposition, uses the asynchronous distributed random gradient matrix decomposition algorithm, and carries on the related test experiment, It is found that this method can shorten the convergence time of the algorithm well.

(3) In view of the new user, through the user time weight of the size, using the form of three graphs to describe the relationship between the various attributes, and further obtain personalized predictive scoring formula, so as to solve the recommendation system cold start problem.

**Key Words:** Information overload, spark, recommender systems, collaborative filtering, cold start

# 目 录

[摘 要 I](#_Toc493591613)

[ABSTRACT II](#_Toc493591614)

[目 录 IV](#_Toc493591615)

[第1章 绪论 1](#_Toc493591616)

[1.1 课题背景 1](#_Toc493591617)

[1.2 研究现状 2](#_Toc493591618)

[1.3 本文的研究内容 3](#_Toc493591619)

[1.4 论文结构安排 3](#_Toc493591620)

[1.5 本章小结 4](#_Toc493591621)

[第2章 相关工作 5](#_Toc493591622)

[2.1 Spark 分布式框架 5](#_Toc493591623)

[2.1.1 Spark简介 5](#_Toc493591624)

[2.1.2 Spark Core技术 6](#_Toc493591625)

[2.1.3 Spark SQL 7](#_Toc493591626)

[2.1.4 Spark Streaming 7](#_Toc493591627)

[2.1.5 Spark MLlib 技术 8](#_Toc493591628)

[2.1.6 Spark与Hadoop区别 8](#_Toc493591629)

[2.2 HDFS 分布式存储 9](#_Toc493591630)

[2.2.1 数据块Block 9](#_Toc493591631)

[2.2.2 NameNode和DataNode 9](#_Toc493591632)

[2.2.3 NameNode管理元数据机制 10](#_Toc493591633)

[2.3 Hbase 数据库 11](#_Toc493591634)

[2.3.1 Hbase概述 11](#_Toc493591635)

[2.3.2 Hbase表的数据结构 12](#_Toc493591636)

[2.3.3 Hbase系统组成 12](#_Toc493591637)

[2.4 推荐算法 13](#_Toc493591638)

[2.4.1 推荐系统介绍 13](#_Toc493591639)

[2.4.2 协同过滤概述 14](#_Toc493591640)

[2.4.3 基于SVD的推荐 14](#_Toc493591641)

[2.4.4 基于人口统计学的推荐 15](#_Toc493591642)

[2.4.5 基于内容的推荐 16](#_Toc493591643)

[2.5 本章小结 17](#_Toc493591644)

[第3章 基于Spark 的推荐系统设计 18](#_Toc493591645)

[3.1 推荐系统离线计算 18](#_Toc493591646)

[3.1.1 业务需求分析 18](#_Toc493591647)

[3.1.2 功能需求和系统整体结构 18](#_Toc493591648)

[3.2 推荐系统实时流计算 20](#_Toc493591649)

[3.2.1 业务需求分析 20](#_Toc493591650)

[3.2.2 功能需求和系统整体结构 21](#_Toc493591651)

[3.3 其他模块详细介绍 22](#_Toc493591652)

[3.3.1 分布式日志框架 22](#_Toc493591653)

[3.3.2 推荐引擎组 23](#_Toc493591654)

[3.3.3 推荐系统整体架构 24](#_Toc493591655)

[3.4 本章小结 25](#_Toc493591656)

[第4章 关于推荐算法的相关问题研究 26](#_Toc493591657)

[4.1 基于用户的相似度计算 26](#_Toc493591658)

[4.2 基于Item的协同过滤算法 27](#_Toc493591659)

[4.2.1 项目相似度计算 27](#_Toc493591660)

[4.2.2 预测计算 28](#_Toc493591661)

[4.3 基于item的正则化和异步分布式矩阵分解 29](#_Toc493591662)

[4.3.1 算法概述及引入原因 29](#_Toc493591663)

[4.3.2 用户矩阵分解和基于Item的矩阵正则化 30](#_Toc493591664)

[4.3.3 异步分布式随机梯度优化 31](#_Toc493591665)

[4.4 关于算法冷启动问题的研究 33](#_Toc493591666)

[4.4.1 冷启动问题概述 33](#_Toc493591667)

[4.4.2 一种解决项目冷启动问题的推荐算法 34](#_Toc493591668)

[4.5 本章小结 38](#_Toc493591669)

[第5章 实验与测试 39](#_Toc493591670)

[5.1 基于Item的协同过滤算法测试 39](#_Toc493591671)

[5.1.1 实验环境 39](#_Toc493591672)

[5.1.2 数据集 39](#_Toc493591673)

[5.1.3 评价指标 39](#_Toc493591674)

[5.1.4 实验过程 40](#_Toc493591675)

[5.1.5 实验结果 40](#_Toc493591676)

[5.2 基于异步SGO算法测试 42](#_Toc493591677)

[5.3 本章小结 44](#_Toc493591678)

[第6章 总结与展望 45](#_Toc493591679)

[6.1 总结 45](#_Toc493591680)

[6.2 展望 46](#_Toc493591681)

[参考文献 47](#_Toc493591682)

[致 谢 49](#_Toc493591683)

# 第1章 绪论

## 1.1 课题背景

随着互联网用户的急剧增多，用户数据也随之剧增，用户画像也日益丰富起来，数据增多的同时也出现了严重的信息过载[1]问题，给人们的生活带来了些许不便，由于互联网上的信息过多繁杂导致用户很难自己定位到自己需要的内容，抑或是应用需要引导用户进行浏览消费，为了解决上述问题，个性化推荐系统应运而生，并且已经得到了越来越多的关注，相关的专家学者们对其研究也更加深入。因为个性化推荐需要实时推送，所以海量数据的实时性和准确性要得到保证。

个性化推荐系统[2]正好和搜索引擎相辅相成，弥补了搜索引擎的部分缺陷。 目前，推荐系统主要用于处理信息过载的情况，系统通过分析日志数据、用户历史行为、用户属性等信息创建用户兴趣偏好模型，首先通过算法根据用户的喜好对物品进行排序，之后经过后端服务向用户推送排序列表。因此推荐系统与搜索引擎相互补充，相辅相成，当用户有明确需求时则采用捜索引擎搜索相关信息，而在用户没有明确自身需求或者以新成员身份进入到一个新的环境中时推荐系统可以帮助他们获取到感兴趣的内容。推荐系统也借此在电子商务领域取得广泛应用。 然而，如何推送高质量推荐列表、快速地响应推荐请求、实时生成推荐列表、海量数据的分析都成为必须面对和解决的难题。

随着消费类网站的数量增多，互联网用户和数据也在急剧增加，随之而来的矩阵稀疏性问题也显现出来，协同算法在这个问题上无法得到解决，导致结果比较差，从而影响用户体验，造成用户的流失。 而且在协同过滤的推荐算法中，往往不能很好的展现用户的爱好迁移问题，因此会对推荐系统的性能造成一定的影响。所以，缓解数据稀疏性问题和适应用户喜好化问题已经成为了网络购物系统发展的严峻考验。

在本文的推荐系统中，为了解决如上难题以及保证系统的可扩展性，需要存储不断增长的数据集以及快速并行化处理。 常用的技术主要包括大数据的高速计算和持久性保存。推荐算法有着共享数据计算，多个线程同时并行操作，该特点属于非循环应用，上述特点适合选取SPARK作为该系统的计算组件。 SPARK是由 UC Berkeley推出的大数据处理框架，该组件的特点是弹性分布式数据集RDD，存储技术是基于分布式抽象而成的，而在后续操作中重用该工作集，这样可以极大加速处理过程。 由于大部分数据挖掘算法涉及到数据集迭代计算，因此 SPARK框架更符合推荐算法的需求。本文根据改进的过滤算法结合SPARK组件进行个性化系统推荐，提高用户的满意。

## 1.2 研究现状

推荐系统从上世纪90年代开始被美国科学家研究，最早的推荐项目是对影片进行推荐，主要是通过根据用户点击习惯和一些影片评分综合起来创建用户画像，从而分析出用户可能喜欢感兴趣的电影，然后进行推荐。早期的算法主要是基于内容的推荐，通过对推荐物品本身进行特征提取，该方式会导致如果用户没有对某类产品评价，该类产品就永远不会被推荐。随着科学技术的发展，上述问题得以解决，原始的推荐算法即将被替代。

协同过滤推荐（Collaborative Filtering，简称CF）起源于1992年。该算法的理论基础在目前的推荐算法领域也占有很重要的地位，由于该算法效果明显，所以科学家基于该算法提出了诸多优化，例如基于物质的或者基于模型的协同过滤推荐算法，基本理论都是基于最初的协同过滤算法。推荐方式有很多种，具体如下：一、基于上下文的推荐[5]，即推荐系统会根据用户的访问位置、访问时间、用户心理以及已有信息预测该用户对其他物品可能的估值。由于个体用户的心理是变化莫测的所以预估的结果有可能有偏差，二、基于用户画像的推荐，即将该用户的所有信息进行整合抽象，把用户的所有数据进行展示，建立该用户的一个用户画像，从多个方面进行映射然后进行个性推荐。三、基于协同过滤的推荐算法分为基于模型推荐算法和基于物理内存的推荐算法，基于模型的推荐算法是指利用数学模型进行评估，判断用户的需求，推送给用户满意的答案，基于内存的推荐算法包括基于用户的和基于项目的推荐算法，效果要基于具体的场景进行比对。四、基于内容的推荐、即对数据内容进行分析并得到物品的结构化描述信息，分析用户在过去时间中评分或评论的商品， 利用用户画像[7]构建预测模型，从而生成推荐策略。五、基于机器学习的推荐系统、根据数据的多样性选取合适的算法策略，自动的归纳逻辑和规则，并根据归纳总结的结果（模型）来对未来数据进行推荐。当然还有其它的一些推荐算法，例如基于内容的推荐算法、基于效果的推荐算法以及基于规则的推荐算法。但是，上述每种单一方式的推荐系统都有各自的缺点，比如基于内容的推荐有严重的矩阵稀疏问题，基于关联规则的推荐较为耗时只能做离线处理等等。

上述的各个算法在不同的工作场景中都有各自的优势和缺陷，所以不能选取单一的推荐算法进行推荐，目前业界的标准方式是选取多个推荐方法相结合的方式进行推荐，目前比较流行的搭配是基于协同过滤的推荐算法和基于项目的推荐相结合的方式，上述组合能够达到比较满意的效果。虽然按排列组合的方式计算会有很多种结合方式，但是需要真正有效的结合才有意义，需要提高推荐算法的准确度，使得结合的算法在准确率、实时性和稳定性方面都有提高。

## 1.3 本文的研究内容

本文主要研究基于Spark平台推荐系统技术应用与优化，着重研究推荐系统架构设计以及系统的推荐效率以及相关算法优化。 文中详细讨论基于Spark的实时推荐的设计与实现，以及该系统中其它基础设施的原理与框架架构，以及基于Spark的RDD算子优化算法执行效率。目前主流搭配是基于协同过滤算法和基于项目的推荐相结合的方式，所以本文基于上述搭配对算法方面进行研究，并对不足的地方进行修改和优化。由于基于矩阵分解的协同过滤推荐算法方面的不足，采用随机梯度优化方法对该算法进行进一步优化。最后对于推荐系统的冷启动[8]问题作出相关介绍，并对此提出解决方案。

(1) Spark生态系统的相关研究。分析Spark系统架构以及生态系统，了解Spark RDD的数据模型，通过函数式编程[9]的模式把分布式数据处理转换成分布式数据集。 使用以命名列方式组织的分布式数据集DataFrame代替直接对Spark RDD做直接操作，使得计算更加简单。用SparkSQL读取Hbase中的数据，并转换成DataFrame，将数据加载到内存，通过并行化计算完成相关数据分析。系统的实时性是用户体验的重要体现，对此本系统采用线上和线下结合的方式，实时计算和历史计算相辅相成，能够缩减响应时间，提高系统的实时性。

(2) 协同过滤算法的相关研究。通过对基于项目的协同过滤算法的研究发现了该算法的问题所在，并且通过实例计算验证推论，之后加入基于矩阵分解的协同过滤分布随机剃度算法，算法的主要过程为将用户评级矩阵分布到不同机器上并且通过网络异步的传播随机梯度优化的更新。 一旦机器收到一个更新的权重，它会使用自己的更新对参数权重进行平均，这样可以保证每台机器能够全面了解整个网络数据情况， 实验证明，该方案具有更好的收敛性。

(3) 推荐系统冷启动问题。由于推荐系统冷启动现象是推荐系统主要难点[10]，所以本文针对该问题提出了相应的推荐算法，在此我们引入了用户时间权重的概念，通过对用户评价的时间间隔判断用户活跃度并根据用户偏爱新项目的程度等因素来创建相应的个性化预测评分公式。将项目本身的各种基本属性、项目标签等和对项目的标注信息的影响都考虑在内，挖掘用户-项目-标签信息之间潜在的关系和用户-项目属性信息之间潜在的关系。最后通过得到的预测评分值使得新项目的冷启动问题得以改善。

## 1.4 论文结构安排

本文余下部分组织如下：

第二章介绍了论文的相关工作：主要是介绍了论文中使用到的关键技术，如Spark 的相关技术栈、HDFS分布式文件系统以及Hbase数据库的原理和应用。并对推荐算法以及协同过滤相关算法做了大致的介绍。

第三章主要介绍了基于Spark推荐系统的系统设计，包括离线计算和实时计算部分，包括它们各自使用到的技术以及在整个推荐系统中的角色以及作用。整个系统以数据收集、数据存储、数据计算、数据合并、数据存储为主线进行描述。

第四章主要介绍了推荐算法的相关问题以及推荐系统的冷启动问题，包括基于Item的推荐算法的优化，分析影响该算法推荐性能的主要因素。引入了矩阵分解的协同过滤的异步分布随机梯度算法，更好的优化了集群场景下推荐算法的收敛时间。此外还对于推荐系统冷启动问题进行分析研究，通过用户时间权重以及项目本身的属性和标注信息，挖掘其潜在的关系，从而改善协同过滤算法的冷启动问题。

第五章对改进的算法使用标准数据集进行测试，并对相关的实验结果进行分析。

第六章总结与展望，对本文的相关工作做出总结并对下一步工作进行展望。

## 1.5 本章小结

本章主要阐述了推荐系统的研究背景、意义和当前推荐算法的研究现状，阐述了推荐系统在实际应用中需要攻克的主要问题，最后对论文的组织结构进行了介绍，使得读者对论文的结构可以一目了然。

# 第2章 相关工作

## 2.1 Spark 分布式框架

### 2.1.1 Spark简介

作为2009年加州大学伯克利分校AMPLAB研究项目的一部分，Spark与Apache Hadoop仍在积极发展。Spark 2010年正式开源并成为Apache基金项目。 2015年有三个版本，从1.3到1.5，分别包含DataFrames API，SparkR和Project Tungsten等功能。版本1.6在2016年初发布，包括新的Dataset API和数据科学功能的扩展。Spark 2.0于2016年7月发布，这是一个主要版本，有很多新功能和增强功能。



**图2-1** Spark提供的主要计算框架

如图2-1，在Spark核心框架基础上，其主要提供了四个范畴的计算框架，包括Spark Streaming, Graphx, MLbase, SparkSQL等。其中Spark Streaming是Spark核心API的一个扩展，它对实时流式数据的处理具有可扩展性、高吞吐量、可容错性等特点。Spark Streaming 可以从各种数据源中获取数据并进行复杂计算后推送到文件系统、数据库等存储介质保存。Spark Streaming接收实时的输入数据流，然后将这些数据切分为批数据供Spark引擎处理，Spark引擎将数据生成最终的结果数据。Graphx主要进行并行图计算，Spark Graphx性能良好，又具有非常丰富的API，并可以基于海量数据进行复杂的图计算从而得到了较为广泛的应用。 Mlbase[11]作为Spark生态系统中的一部分，由AMPLab实验室的Mlbase团队开发，是Apache Spark中的核心机器学习库，专门提供机器学习相关的丰富的API，从而让机器学习的门槛更低，使那些不了解机器学习的开发人员也能通过MLbase来处理自己的数据。Spark SQL的引入专门用于处理结构化数据比如（DB、JSON）。Spark SQL不仅是SQL，它对各种数据源做了抽象，能够对各种数据源进行ETL操作，并提供丰富的对复杂数据读写的API，使用非常方便。Spark SQL对多种数据源都提供了很好的支持，其中DataFrame API作为一种具备高伸缩性的数据操作方式，相比RDD API 效率更高[12]。此外Spark SQL还提供了Catalyst Optimizer查询优化器，可以自动优化程序，使得程序更加高效。

Spark引入的弹性分布式数据集RDD[13]，是一种高度受限的共享内存模型，是基于内存的。这使得其计算效率和速度相比Hadoop 的 MapReduce（基于磁盘，每次执行时都要从磁盘读取数据，计算完毕后都要把数据存放到磁盘上）提升很多。

Apache Spark是围绕速度、易用性和复杂分析构建等方面的大数据处理框架，适用于多种不同的情形的分布式应用场景，包括批处理、流处理、交互式查询等等，并大大减轻了原先需要对各种平台分别管理的负担。统一的RDD抽象和操作，基于统一的技术堆栈，使得Spark称为大数据通用计算平台。在大数据时代逐渐普及的背景下得到了越来越广泛的应用。

### 2.1.2 Spark Core技术

Spark Core是Spark整个生态系统的核心，它主要提供了一套基于RDD的API以及分布式的计算引擎。Spark采用了Master-Slave的分布式计算模型，Master作为主节点，负责任务调度，管理Worker节点，并通过RPC（远程过程调用）通知Worker 启动Executor。Worker则是计算节点，负责管理本节点资源，并向Executor结点发送执行命令，并维持与计算节点的心跳。此外Spark还引入了弹性分布式数据集RDD(Resilient Distributed Datasets)，它是一个容错的、并行的数据结构，可以让用户显式地将数据存储到磁盘和内存中，并能控制数据的分区。同时，RDD还提供了一组丰富的操作来操作这些数据。下面具体介绍spark 执行流程：

如图2-2所示，当RDD算子遇到Action方法时，DAG图（有向无环图）即确定了边界，DAG图形成，随后将DAG提交给DAGScheduler。DAGScheduler 会将DAG切分成多个stage，切分依据（宽依赖—即RDD数据需要通过网络传递）。多个stage封装到TaskSet后提交给TaskScheduler，随后TaskScheduler把任务提交给worker执行。其中DAGScheduler 和TaskScheduler都在Driver端，main函数创建SparkContext时会使得driver和Master建立连接，Master会根据任务所需资源在集群中找到符合条件的worker。随后Master对worker进行RPC通信，通知worker启动Executor，Executor会和Driver 建立连接，随后的工作worker和Master不再有关系。Driver会直接向Executor提交Task，并启动Executor执行任务。



**图2-2** Spark任务执行流程

### 2.1.3 Spark SQL

基于Spark Core，SparkSQL构建了一套基于SQL的引擎。该引擎由两层构成，最底层是查询优化器Catalyst Optimizer, 可以优化程序，使得SQL的运行更加高效。比如两个表的join操作，有多种方式：其一、轮询操作，具有O(n^2)的时间复杂度；其二、先排序再遍历，该时间复杂度为O(nlogn)；其三则是基于Hash的一种实现，其时间复杂度是O(m+n)。Catalyst Optimizer则会根据其数据分布、数据的特点等应用场景生成合适的执行计划，从而达到SQL高效运行的目的。

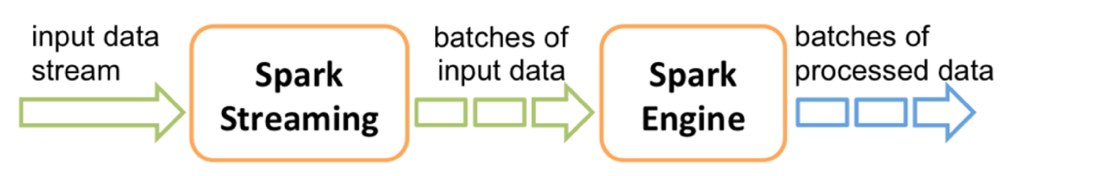


**图2-3** Spark SQL整体架构

在Catalyst Optimizer之上，Spark SQL提供了两种编写应用程序的方法，一种是经典的SQL方式，另一种是DataFrame/DataSet API。由于Spark SQL相比于Spark Core的显著优势，使得大部分程序的开发不再基于RDD API （如MLlib）而转向基于DataFrame/DataSet API进行开发。

### 2.1.4 Spark Streaming

Spark Streaming类似于Apache Storm，主要应用于流式数据计算，具有高吞吐量和容错能力强等特点。Spark Streaming支持的数据输入源很多，例如：Kafka、Flume、Twitter、ZeroMQ和简单的TCP套接字等等。而结果也能保存在很多地方，如HDFS、数据库等。另外Spark Streaming也能和MLlib（机器学习库）以及Graphx完美融合。



**图2-4** Spark Streaming 工作流程图

Spark Streaming 主要功能是根据时间片的长短，将流数据累积为RDD, 然后再将每个RDD进行批量处理，从而达到大规模处理流数据的作用。由于这样的一种模式，使它的吞吐量比现在主流框架Storm还大，同时，它还提供了丰富的API 用于流数据的计算。具体的相关应用会在实时计算章节进行展开。

### 2.1.5 Spark MLlib 技术

机器学习和统计学技术是从复杂的海量数据中挖掘有用的信息。对于一般开发者来说，即使了解其中的相关算法，但是要想实现，也是有一定困难的。因此MLlib的愿景就是解决开发者的这些问题，MLlib 构建在apache spark之上，是一个专门针对大量数据处理的通用的、快速的引擎，并实现了一些常见的机器学习算法和实用应用程序，包括分类、回归、聚类、协同过滤、降维以及相关的底层优化。这样只要具备一定的理论知识，便可以进行机器学习及统计学方面的开发了。

### 2.1.6 Spark与Hadoop区别

Hadoop和Apache Spark都是大数据框架，但它们并不真正具有相同的用途。 Hadoop本质上是一个分布式数据基础架构：它在大量服务器群集中的多个节点之间分发海量数据集，这意味着用户不需要购买和维护昂贵的定制硬件。它还可以索引并跟踪数据，从而大大提高了数据处理和分析效率。另一方面，Spark是一种数据处理工具，可以对这些分布式数据集合进行操作，但它本身不具备分布式存储能力，因此需要在Hadoop的分布式文件系统或其他文件系统的基础上工作。

Hadoop不仅包括称为Hadoop分布式文件系统的存储组件，还包含一个名为MapReduce的处理组件，因此我们可以不使用Spark来进行计算。 相反，我们也可以使用Spark而不使用Hadoop。Spark是一个多阶段的具有RAM能力的计算框架，具有机器学习，交互式查询和图形分析的实现库。但Spark并没有自己的文件管理系统，所以它需要与一个HDFS（而不是另一个基于云的数据平台）集成。Apache Spark非常适合批处理和流处理，这意味着它是一个混合处理框架。 Spark通过内存计算和处理优化加快了批处理速度，因此它的速度通常比MapReduce快得多，这使得它成为处理流式传输工作、交互式查询和基于机器学习的不错选择。 Spark也可以与Hadoop及其子模块一起使用。由此可见，实时数据处理能力使Spark成为大数据分析的首选。

Hadoop和Spark都是开源项目，但是，Spark使用大量的RAM来运行内存中的所有内容，RAM比硬盘更昂贵。Hadoop是磁盘绑定的，因此节省了购买昂贵的RAM的成本，但需要更多的系统来分配多个系统上的磁盘I / O。

就成本而言，如果要处理大量的大数据，Hadoop将会更便宜，因为硬盘空间比内存空间的价格要低得多。

## 2.2 HDFS 分布式存储

HDFS是现阶段较为流行的分布式文件系统，它是Apache Hadoop中的核心项目之一，用于解决大数据集中存储于物理机导致容量和传输速度成为瓶颈所带来的局限性问题。下面我们主要介绍HDFS框架结构以及原理。

### 2.2.1 数据块Block

HDFS文件系统为了提高磁盘的读写效率，引入了数据块(Block)概念，作为文件系统的存储数据的最小单元，其默认大小为64MB，也可以根据项目具体需求进行设置，一般设置为128MB，设置较大的目的是为了减少寻址开销，数据的备份也是以块为单位。

### 2.2.2 NameNode和DataNode

HDFS采用Master/Slave的体系结构，其中NameNode节点作为集群的主服务器，用于管理整个文件系统的元数据(Metadata)，进行文件系统的相关命名空间操作，如打开、关闭和重命名文件夹和目录等，并确定数据块到DataNode的映射。若干个DataNode节点作为HDFS集群的一般节点，负责管理用户的文件数据块、数据存储，并定期向NameNode汇报存储的文件信息。DataNode还可以根据NameNode的指示执行块创建、删除和复制，并维持NameNode之间的心跳。在系统内部，文件被分割成一个或多个Block块（文件系统可读写数据的最小单元）中，并且这些数据块被存储在一组DataNode服务器中。



**图2-5** HDFS架构图

### 2.2.3 NameNode管理元数据机制

NameNode中仅仅存储文件目录树的相关信息，具体则存储在fsimage（内存镜像文件）中，当NameNode启动时，会先读取该文件，并将目录树加载到内存中。而edits文件存储的是数据操作的日志信息，如图2-6所示。



**图2-6** 节点目录树

当有Metadata进行更新操作时，首先在NameNode中的内存中进行，并将此次操作记录在edits中。该记录并不代表Metadata本身，但可以根据该记录恢复Metadata信息，如果NameNode宕机，仍然可以根据操作日志恢复历史数据。如果随着NameNode运行时间的增加，edits文件越来越大，下次服务器启动会消费较长的时间来合并edits生成新的fsimage镜像文件。因此引入了secondNameNode, secondNameNode会定期通过rpc请求NameNode询问是否触发checkpoint，NameNode会根据自定义规则（运行时间/edits数据大小）返回给secondNameNode是否满足checkpoint条件。如果满足条件，secondNameNode会通知NameNode滚动刷新正在写入的edits\_inprogress文件，并下载该文件和之前历史的edits文件以及历史的内存镜像文件fsimage，并加载到内存，通过操作日志更新fsimage，生成最新的fsimage.chkpoint，并上传到NameNode，覆盖历史的fsimage文件。具体流程图如图2-7所示：



**图2-7** 元数据管理机制

其中checkpoint条件触发在配置文件中进行设置：

dfs.namenode.checkpoint.check.period=60 #检查触发条件是否满足的频率，60秒

dfs.namenode.checkpoint.dir=file://${hadoop.tmp.dir}/dfs/namesecondary

#以上两个参数做checkpoint操作时，secondary namenode的本地工作目录

dfs.namenode.checkpoint.edits.dir=${dfs.namenode.checkpoint.dir}

dfs.namenode.checkpoint.max-retries=3 #最大重试次数

dfs.namenode.checkpoint.period=3600 #两次checkpoint之间的时间间隔3600秒

dfs.namenode.checkpoint.txns=1000000 #两次checkpoint之间最大的操作记录

## 2.3 Hbase 数据库

### 2.3.1 Hbase概述

Hbase[14]是一个分布式的、面向列的开源的key-value数据库，其数据是按照key的字典序顺序存储。Hbase和传统的数据库的主要区别是它是一个面向列的数据库，因此对于schema限制很少，可以自由的添加字段，为半结构化数据的存储提供条件。Hbase基于Hadoop的HDFS文件系统，数据存储在hdfs上，备份机制健全。支持线性扩展，用Hbase技术可在廉价的PC server上搭建大规模结构化集群，其容错能力和处理能力随集群数量增长而变强，可以处理PB级以上的数据，并且是一个强一致性的K-V系统。

Hbase也是主从结构，主节点HMaster、从节点Hregionserver。HMaster只管理数据偏移和表的信息，不存储实际数据。Hbase作为缓存层，当数据到达一定数量后再由内存写入HDFS中，从而减少HDFS元数据个数。

### 2.3.2 Hbase表的数据结构

(1) 行：同一个Key对应的所有数据，其数量没有上限。

(2) 列簇：相似的列通常会被划分到一个列簇，列族的数量有限，通常是在 建表时定义好，后期无法动态增加。

(3) 列：一个列簇中可以有多个列，列名在数据写入中确定，列的数量没有 明确的限制。

(4) Cell及时间戳：Cell用户存放每个列的内容，其内容不能修改只能追加， 因此每个Cell都可以有任意多个版本。



**图2-8** Hbase数据库表示例图

Hbase的表类似于传统的DBMS表，数据按row-key的值从小到大排列，数据被组织到若干个Region中，一张表会被切分成若干个Region，Region是Hbase作为负载均衡调度的最小单位，Hbase之所以可以提供分布式的服务，是因为它可以把一个从宏观上看是一个整体的表，切分成若干个Region，分别由不同的机器管理。

Hbase作为Hadoop的子项目，对Hadoop的Map-reduce有着良好的支持，可以通过在Hadoop上的Map-reduce任务来并发的对Hbase表进行并发Scan（扫库）操作。

### 2.3.3 Hbase系统组成

如图2-9为Hbase系统结构图，其中 Master节点，可以由一台或多台服务组成，用于实现Region Server集群的负载均衡，当某一台的Region Server数据量过多时，Master节点会发起负载均衡请求，使该Region Server上的数据迁移到较为空闲的Region Server上。此外Master还用于管理与MetaData相关的事务控制等操作。



**图2-9** Hbase系统结构图

Hbase中每个表被分成多个Region，每个Region会被分配到多个RegionServer上，RegionServer可以看作是该架构中的Slave节点，主要用于读写数据的操作，完成Master的相关指令。RegionServer由Zookeeper集群统一管理，Zookeeper集群可以提供分布式环境的多机协调功能[15]，当某一个RegionServer宕机时，Zookeeper的通知机制会使得Master节点感知到RegionServer的宕机事件，并将该RegionServer上的数据分配到其他服务器中。Zookeeper还提供分布式锁的服务，在为多用户提供服务的系统中，多个用户可能在同一时刻对关键资源进行访问，通常我们会在程序中添加锁来控制，而对于分布式的系统则是通过Zookeeper来完成。

## 2.4 推荐算法

### 2.4.1 推荐系统介绍

推荐系统的出现大多数用于为实时互动中的用户提供相关信息、产品的个性化推送服务。在这些系统中，特别是基于k-最邻近法[16]（k-nearest）协同过滤的系统已经在网络上取得了广泛的成功。近年来，可用信息量和网站访问量的巨大增长形成了对推荐系统的一些关键挑战。最关键的技术挑战是产生高质量的建议，并为数百万用户和项目在每秒钟内提供准确有效的建议，在面对较少的数据的情况下实现高覆盖率。在传统的协同过滤系统中，工作量随着系统参与者的数量而增加，因此需要一个即使在非常大规模的问题的情况下，也可以快速产生高质量建议的新的推荐系统技术。为了解决这些问题，我们探讨了并分析了多种推荐算法，并介绍了它们各自的优缺点、原理和应用场景。

### 2.4.2 协同过滤概述

协同过滤算法的目标是根据用户之前的兴趣喜好和他相似用户的兴趣爱好来推荐新项目和预测用户对该项目的评分。在典型的CF场景中，有一个m个用户U={u1,u2,…..um}列表和n个项目I={i1,i2,….in}的列表。 每个用户ui都有一个项目列表Iui用于来表达用户对项目的评价。 用户可以通过分析定时日志、挖掘Web超链接等来将用户的意见明确地进行评分操作，通常这些分数在一定数值范围内，或者可以隐含地从记录中导出。注，和Iui可能是一个空集。协同过滤算法的任务先使用统计技术寻找与目标用户有相同喜好的邻居，然后根据目标用户的邻居的喜好产生向目标用户的推荐。

预测是一个数值，Pa, j，表示对于活跃用户Ua的项目ij ∉Iua的预测可能性。该预测值与Ua提供的意见值在相同的比例内。

推荐是N个项目的列表，Iτ∈I，是活跃的用户最喜欢的项目。请注意，推荐列表必须是活动用户尚未购买的项目，即。CF算法的这个接口也被称为N类（Top-N）推荐。

协同过滤过程的示意图如图2-10。CF算法将整个m x n个用户项目数据表示为评级矩阵A。A中的每个条目ai, j表示第i个用户在第j个项目上的偏好分数（评分）。每个单独的评分都在数字范围内，数值为0，表示用户尚未对该项目作出评价。主流的协同过滤算法，可以分为两大类：基于内存（基于用户）和基于模型（基于项目的）两种算法[17]。在本节中，我们提供了基于项目的CF的推荐系统算法的详细分析。



**图2-10**协同过滤过程

### 2.4.3 基于SVD的推荐

在实际项目中，评分矩阵往往十分庞大，假设该系统用户有10000个，而物品数量有10000个，那么该评分矩阵的规模就是10000\*10000=1亿。对于拥有巨大用户数量和物品数量的大型电商来说，这个矩阵的规模便更是难以衡量，然而更重要的一点是，这个矩阵很有可能是非常稀疏的。例如Netflix数据集中非零元素个数只占所有元素个数的1%左右，这种稀疏性给存储和计算带来了巨大的挑战。

奇异值分解(SVD)[18][19]是一种分解技术。以矩阵A为例，现假设它有m行n列，那么对矩阵A进行奇异分解，过程如下如下：

(公式2.1)

上式中的∑是一个m行n列的矩阵，除对角线元素以外，其余元素的值都为0，那么对角线上的非零元素就成为了奇异值。令s=(), 集合s中的每个元素均为一个奇异值，并且由大到小降序排列，然而集合s中的元素一般减小的速度会很快，如果我们去掉前r个奇异值对其进行降维时，就会得到：

(公式2.2)

这里的r远远小于行m和列n，这样基于 , , 三个降维后的矩阵的计算会比直接基于A的运算快很多。

基于SVD的推荐方法实际上就是先将待处理的评分矩阵进行SVD分解，去掉合适个数的r个奇异值进行降维近似，得到，,三个矩阵，然后基于这三个低阶矩阵运算进行推荐。

基于SVD的推荐方法本质上还仍然是利用用户的行为数据，不同的是在推荐前后首先离线的对评分矩阵进行分解，其次进行降维，这样大大减小了存储的数据量，同时减小了在线计算量。但它也有以下两个主要缺点：

1) 降维时r维不好确定。如果r设置的太大，那么降维的效果就会不好，如果r设置的太小，又会导致损失了太多信息。所以在实际使用的过程中一般会依据奇异值的衰减程度来确定r的值。

2) SVD本身就有非常大的计算量，当数据量达到一定程度时，消耗的时间会无法接受。由于现在Spark已经实现了对SVD的并行化计算，即当r选取较小的值时，Spark的计算速度已经可以达到比较快了，在可以接受的范围内。

### 2.4.4 基于人口统计学的推荐

基于人口统计学[20]的推荐方法则是根据人的静态描述信息来进行推荐的方法。它是根据简单的人口统计信息来发现并得到用户的相似度，然后利用相似度寻找与该用户相近的人，并给该用户推荐这些与他相近的人所喜欢的物品。

具体的来说，这种方法首先会收集并利用用户基本的统计信息，比如年龄，性别，籍贯，住址等，其次采用特定的方法计算相异度，计算用户之间的差异。然而实际计算用户属性类别却并不是相同的，比如性别属于二元属性，年龄属于序列属性，职业属于名称属性等等。现假定数据集中一共含有p个混合的属性，那么对于对象i和j的距离，我们将会采用下面的公式计算得出：

(公式2.3)

(公式2.3) 中代表用户的属性f的指示符，则是含有用户的属性f的记录i和j之间的相异度。

如图2-11所示，左边一列分别是用户甲、乙和丙的信息，用户的属性都是名称类和二值类的属性，那么根据公式（2.3）计算后可以得到用户丙与用户甲的距离为0，和用户乙的距离为1，由此可得用户丙和用户甲之间更加相似，所以我们就可以将用户甲喜欢的东西推荐给用户丙。这里假设的邻居数只有一个，然而在实际应用中，可以通过对邻居数目做相应的调整，从而进行覆盖率更高的推荐。



**图2-11**基于人口统计学的推荐

### 2.4.5 基于内容的推荐

基于内容的推荐[21]是利用物品的内容信息，首先算出物品之间的相似度，根据用户以前喜欢的物品信息，向用户推荐与这些物品信息记录相似的类似物品。具体计算相似度的方式有很多，这里可以我们采用Jaccard公式：

(公式2.4)

或者采用简化的余弦相似度公式：

(公式2.5)

公式（2.4）和公式（2.5）中的代表两个电影u与v之间的相似度，其中代表电影u的内容，代表电影v的内容。

如图2-12所示，右侧一列为电影的类型信息，A为“爱情，浪漫”类型的电影，B为“恐怖，惊悚”类型的电影，C为“爱情，浪漫”类型的电影。利用（公式2.5）计算可以得知，电影A与电影B之间的相似度为0，电影A与电影C之间的相似度为1，由于用户甲喜欢电影A，并且电影A与电影C的相似度较高，所以系统会将电影C推荐给甲。



**图 2-12**基于内容的推荐

## 2.5 本章小结

本章主要介绍了Spark的生态系统以及相关框架，并阐述了其优越性。介绍了大数据技术的一些框架和其工作原理，并且简单介绍了几种常见的推荐系统和一些常用的推荐算法。

# 第3章 商城及推荐系统需求分析

## 3.1 系统业务需求

本系统以网上商城为主体，通过对用户行为日志的分析，结合协同过滤推荐算法来为用户推送可能感兴趣的商品列表，从而帮助用户在海量商品中快速定位到适合自己的产品，同时也为更好的帮助商家扩展业务渠道，增加业务流水。

整个商城推荐系统的业务流程如下：

通过用户分析用户的行为日志信息、浏览记录、停留时间、点击数等来预测用户偏好。以用户数据库信息为基础，来进行多维度商品推荐。基于用户注册信息（性别、年龄、地区等）通过user-based 协同过滤算法给出基于用户的推荐结果。通过用户浏览、收藏、以及购物车中的商品同时结合商品标签，通过item-based协同过滤算法给出基于商品的推荐结果，随着用户量以及商品总数的不断增加，推荐算法的准确性和时效性是考验整个系统好坏的主要标准之一。

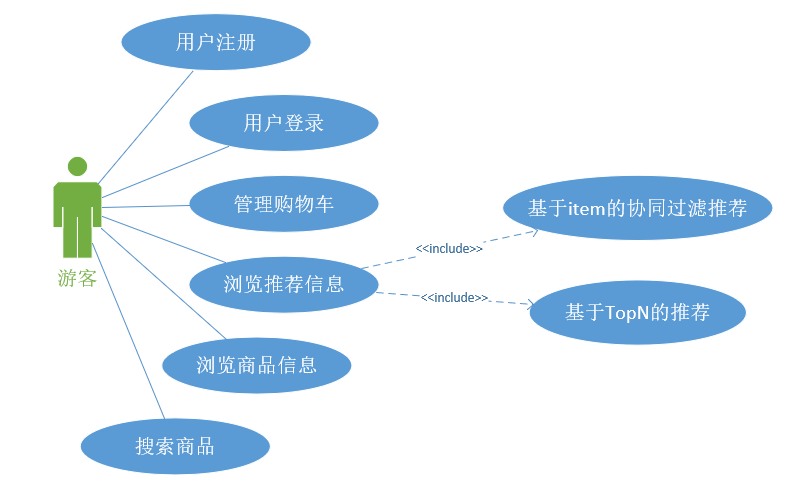
## 3.2  系统功能性需求

### 3.2.1 商城系统总体功能需求

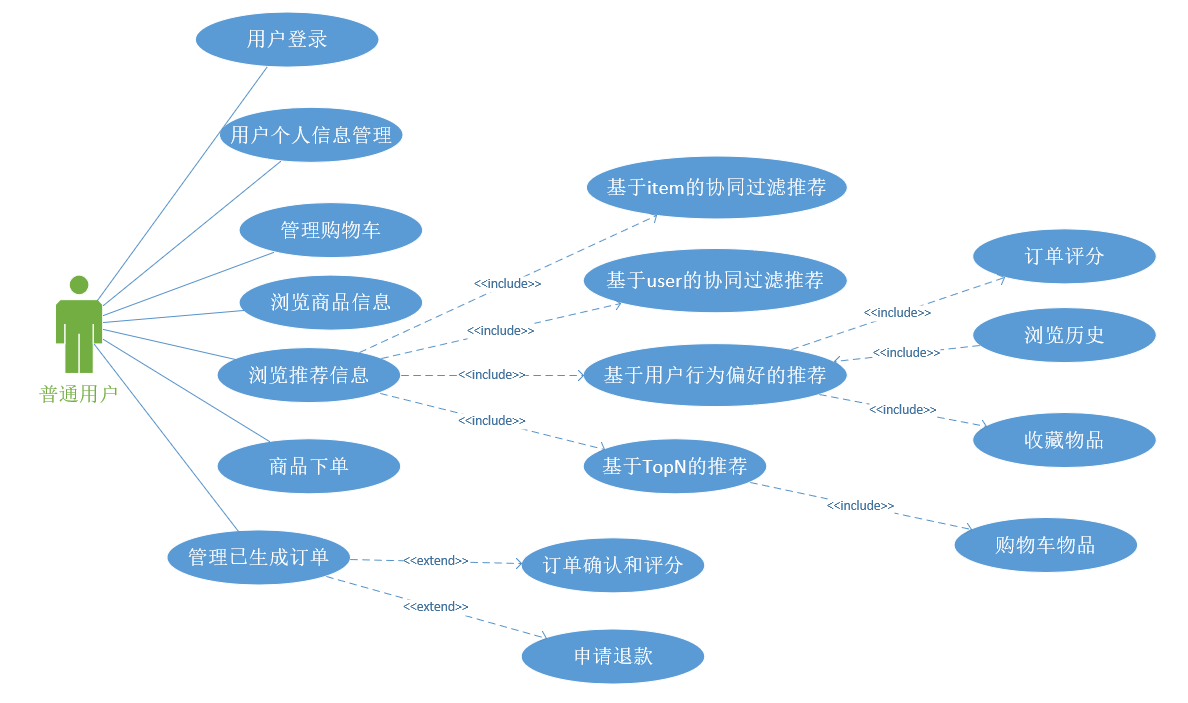
用户通过注册接口，填写个人信息并登录， 浏览页面调用商品详情接口，查看物品详情，对喜欢的商品进行收藏、加购物车、购买等操作，并在确认收货后，对商品进行评价和打分。用户状态包括游客，普通用户，会员用户。推荐算法会根据不同的用户状态使用不同的推荐算法生成推荐结果。

不同角色的需求用例图如下图所示：

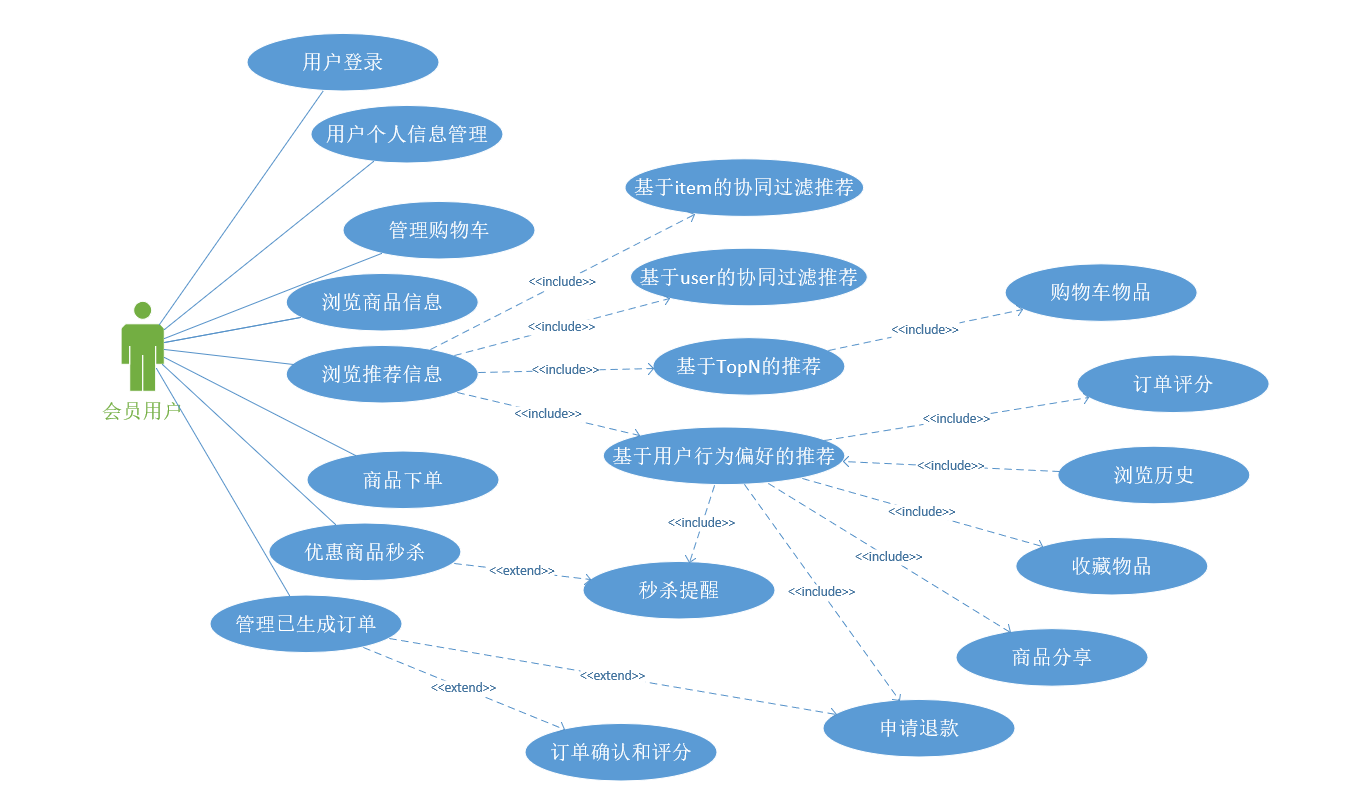
用户状态为游客时的用例图：



用户状态为普通用户时的用例图：



用户状态为会员时的用例图：



### 3.2.2 单点登录模块需求分析

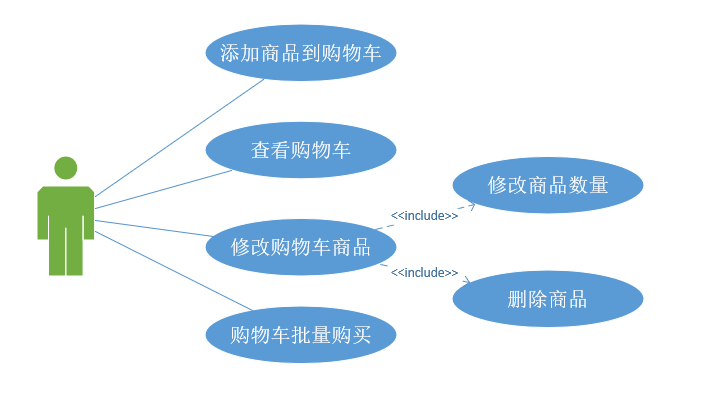
对于单一服务器应用系统而言，用户登录成功后，会在服务器端生成session保存用户相关信息，然后将用户唯一的session\_id返回，并存入客户端对应的cookie中。用户下次访问其他页面会携带此cookie，服务器会根据cookie中的session\_id来验证用户是否登录。

但是对于多服务器应用而言，每个应用模块可能分布在不同的服务器节点上，而登录产生的session只存储在登录的服务器中，其他模块如果需要获取登录状态就必须将session共享，因此基于该需求设计并实现了类似session共享的单点登录模块，实现只需要用户登录一次就可以在多服务器的系统中保持登录状态。

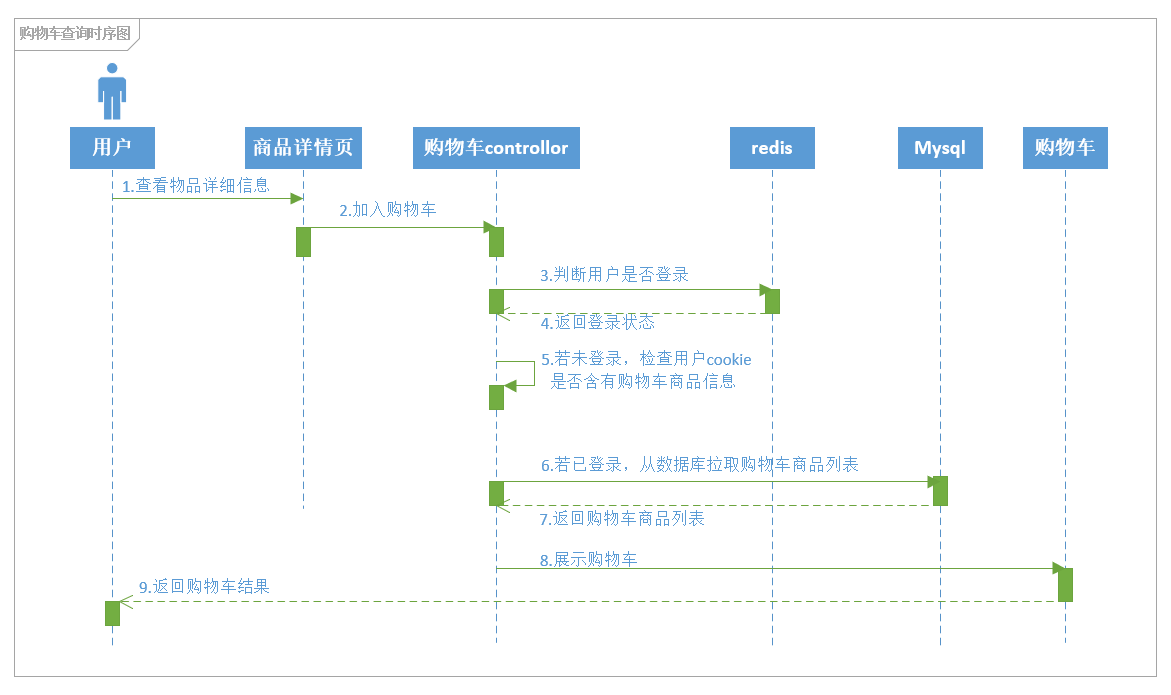
### 3.2.3 购物车模块需求分析

购物车主要用于保存用户待下单商品，主要包括未登录状态保存商品到购物车、已登录状态保存商品到购物车、查询购物车商品列表、修改或删除购物车商品、以及购物车商品批量购买等。

购物车添加查询时序图如图？-？所示：



购物车模块用例图

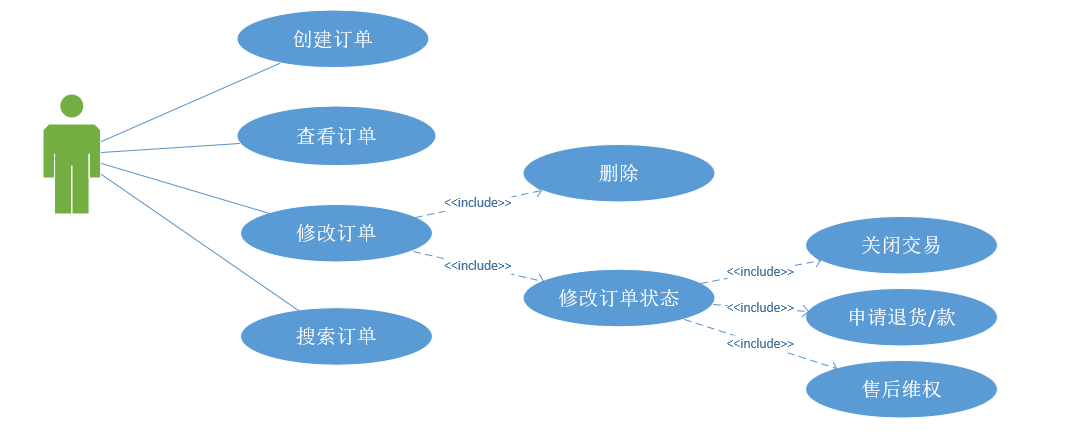


购物车模块时序图

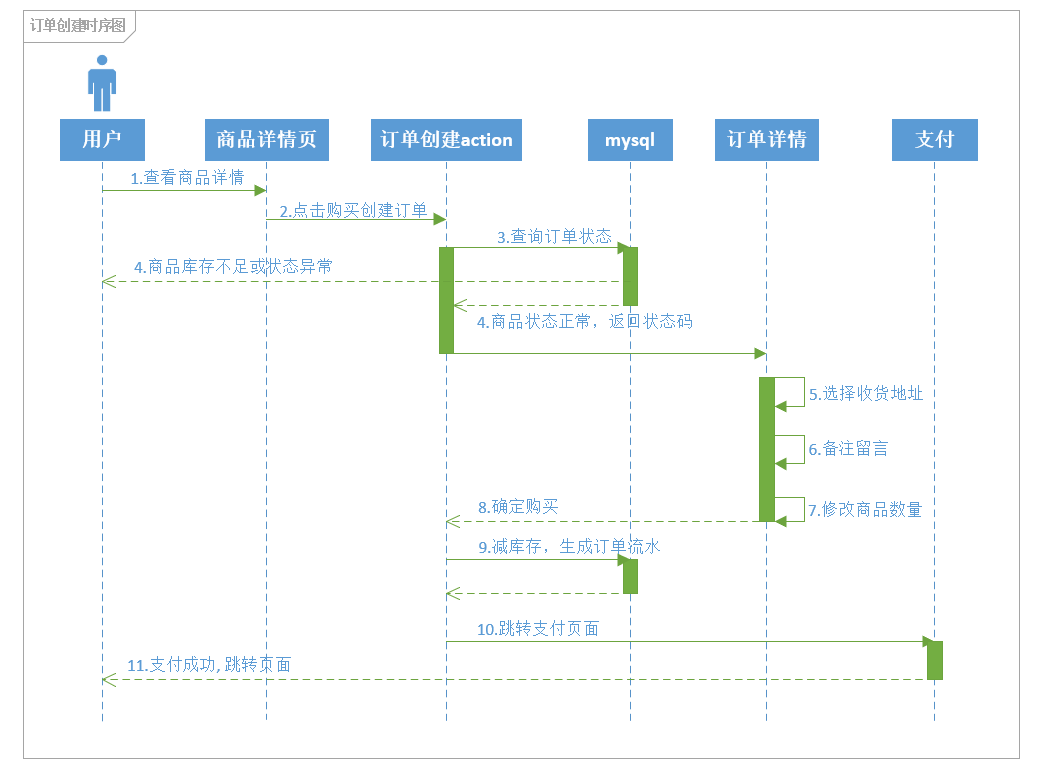
图？-？是购物车模块用例图，其中包括用户对购物车的相关操作，其中添加商品到购物车，若用户未登录，会将购物车信息保存在浏览器的cookie中，若用户已登录，那么就将数据保存在数据库中。查看购物车详情，会罗列该用户的所有购物车商品。修改购物车中的内容则包括删除购物车中的商品以及修改购物车中商品的数量等。最后我们可以批量的选取购物车中的指定商品进行购买操作。

### 3.2.3 订单管理模块需求分析

订单系统



订单管理模块用例图



订单创建时序图

### 3.2.4 用户日志收集模块需求分析

由于整个商城系统是由多个模块组成，每个子模块可能分别处于不同的服务器上，

### 3.2.5 推荐系统计算框架需求分析

通过分析用户行为日志和用户基础数据为用户实时的推荐可能感兴趣的商品列表，在此为了使得推荐系统的效果在时效性和准确性达到性能平衡，我们引入和离线计算和实时计算相结合的方式，下面我们对两种计算方式的业务需求进行介绍。

离线计算功能需求:

推荐系统在数据基础构建阶段有大量的耗时操作比如模型训练，中间数据计算（线性分类器参数），协同过滤中item的相似矩阵的计算，用户画像构建，关联规则的发现等等，这些计算通常比较复杂并且有大量的数据库查询操作，计算周期长，是算法性能的主要瓶颈，因此需要通过离线计算获得，为实时推荐提供数据基础，最后通过实时数据校正计算结果，从而达到近乎即时的推荐。

实时计算功能需求:

对于推荐系统而言，随着应用场景的多样性，以及许多业务场景的特殊性，对推荐系统实时性的要求日益提高，如果推荐系统不能根据用户的行为变化和商品信息的变化做出及时的结果更新，推荐结果的有效性可能会大打折扣，用户可能会因此无法获取有效的符合自身需求的推荐结果。传统的推荐方式采用定时离线计算来更新推荐结果，该方式计算周期往往以小时、天为单位。推荐给用户的商品或信息往往是滞后的、过时的，是基于用户前几个小时或者前一天的行为得到的结果，往往无法满足用户当前的需求，因此这就迫使实际的推荐系统实时性有本质的提高。

整个推荐系统推送的商品主要显示区域包括主页标题滚动的商品，主页右侧猜你喜欢的模块列表，以及当用户成功添加商品到购物车时底部推荐的相似商品等。

### 3.3.6 基于user-based的协同过滤推荐模块需求分析

基于用户的协同过滤推荐算法其主要思想是找到与该用户相似的用户群，然后将这些相似用户喜好的商品进行过滤筛选，去除掉用户已知和评价低的商品，然后计算用户对物品集的兴趣度，按排名由高到低给出推荐。

因此我们可以选择合适的应用场景来使用此类算法生成推荐。当新用户首次注册时，我们根据用户填写的注册信息如年龄、地区、以及兴趣爱好、以及购物偏好等通过相似度计算找到与该用户相似度较高的用户群，然后找到该用户群购买收藏度较高的商品推荐给用户。

### 3.3.7 基于item-based 的协同过滤推荐模块需求分析

基于商品的推荐相比于基于用户的推荐而言应用较为广泛，此类推荐的主要思想是通过所有用户历史喜好的物品以及最近搜索关注度较高的物品来计算物品之间的相似度，找到相似物品后预测用户评分，然后把与用户的喜好相似度较高的商品推荐给用户。

假设商品A和商品B具有较高的相似度，那么用户甲对商品A进行收藏、加购物车、或者购买操作时，我们可以将于商品A相似的商品B推荐给该用户，从而帮助用户更好的挖掘自己可能喜欢的商品，提高购买效率。

由于商品间的相似度较为稳定，我们可以通过离线的方式预先将商品间的相似度计算并入库，后期直接通过查表的方式给出推荐结果。

## 3.2 系统非功能性需求

(1)响应时间: 整个系统的服务接口的要有足够快响应速度，包括页面的加载以及其他基本操作。推荐系统的推荐结果也要在有效的时间范围内生成，这对于增强用户体验是非常重要的。

(2)数据一致性: 在集群环境下，对个服务节点的数据的一致性是整个系统最基本的要求。无论是强一致性还是最终一致性，最终我们都要系统在不同的应用场景中，数据达到一致。

(3)系统高可用: 当某个节点的服务器宕机，集群会动态感知并获取该服务器信息，并发送告警信息，并将该节点从服务节点列表中移除，又其他节点代替该节点继续处理请求。

(4)推荐系统冷启动问题: 推荐系统的冷启动问题主要包括新项目冷启动和新用户冷启动两个方面。而对于新用户的冷启动问题一般可以通过利用用户在其他地方已经沉淀的数据和用户兴趣偏好进行冷启动或者制造选项生成粗粒度的推荐等方式进行。这里着重解决新项目的冷启动问题，在此我们引入了用户时间权重的概念，通过对用户评价的时间间隔判断用户活跃度并根据用户对新项目偏爱程度等因素创建构建个性化的预测评分公式。由于新项目不存在任何一个用户对它的评价信息，故无法使用协同过滤算法来实现推荐，因此，本章节提出解决该类冷启动问题推荐算法，该算法不仅仅使用用户对物品的评分信息对来进行推荐，而是要综合考虑项目本身的诸多属性信息、标签对项目的标注信息等较为全面的考虑影响推荐系统的诸多因素。实验证明该方式，在解决推荐系统在新项目的冷启动的问题上是较为有效的。

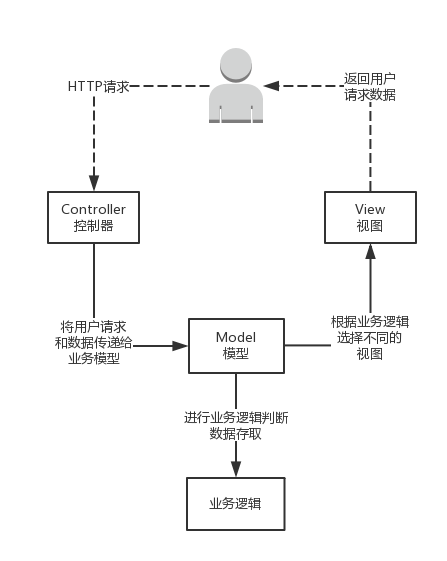
## 3.3本章小结

# 第4章 商城及推荐系统概要设计

## 4.1 系统总体结构设计

整个商城系统是在Springmvc、Spring、Mybatis三大框架基础上搭建而成，采用MVC的设计模式，其中Model模型层所管理的对象直接与数据库表相关联，负责数据的存取；View层为视图层，主要用于为系统提供图形化页面和数据显示；Controller控制器层负责具体的业务模块流程的控制，为外部请求返回相对应的视图。

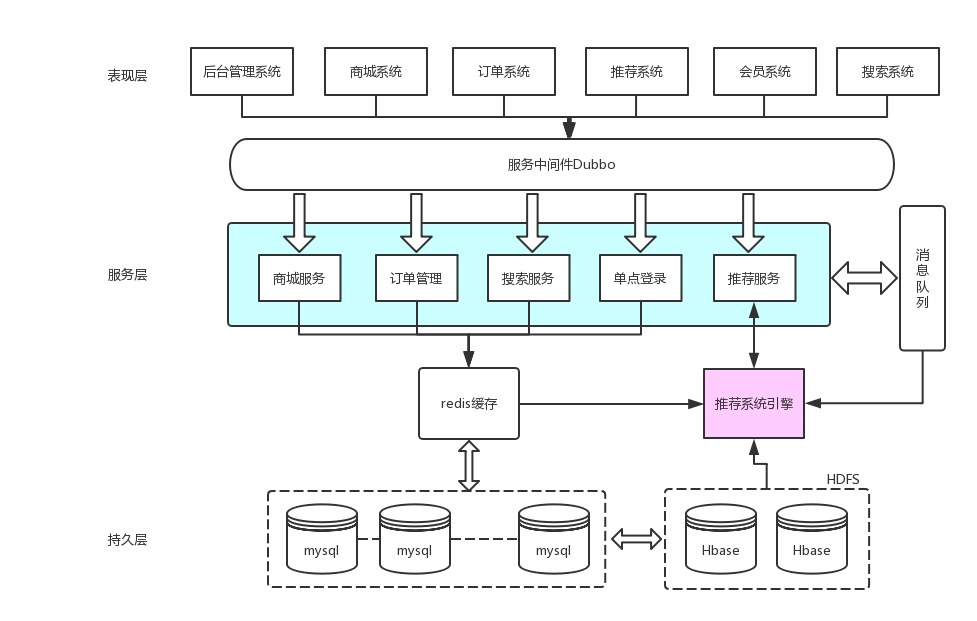
整个系统架构图如图？-？所示。



系统架构图

## 4.2 系统功能模块设计

该商城系统的技术架构主要分为三层，包括表现层、服务层、以及持久层，具体架构图如下图所示：



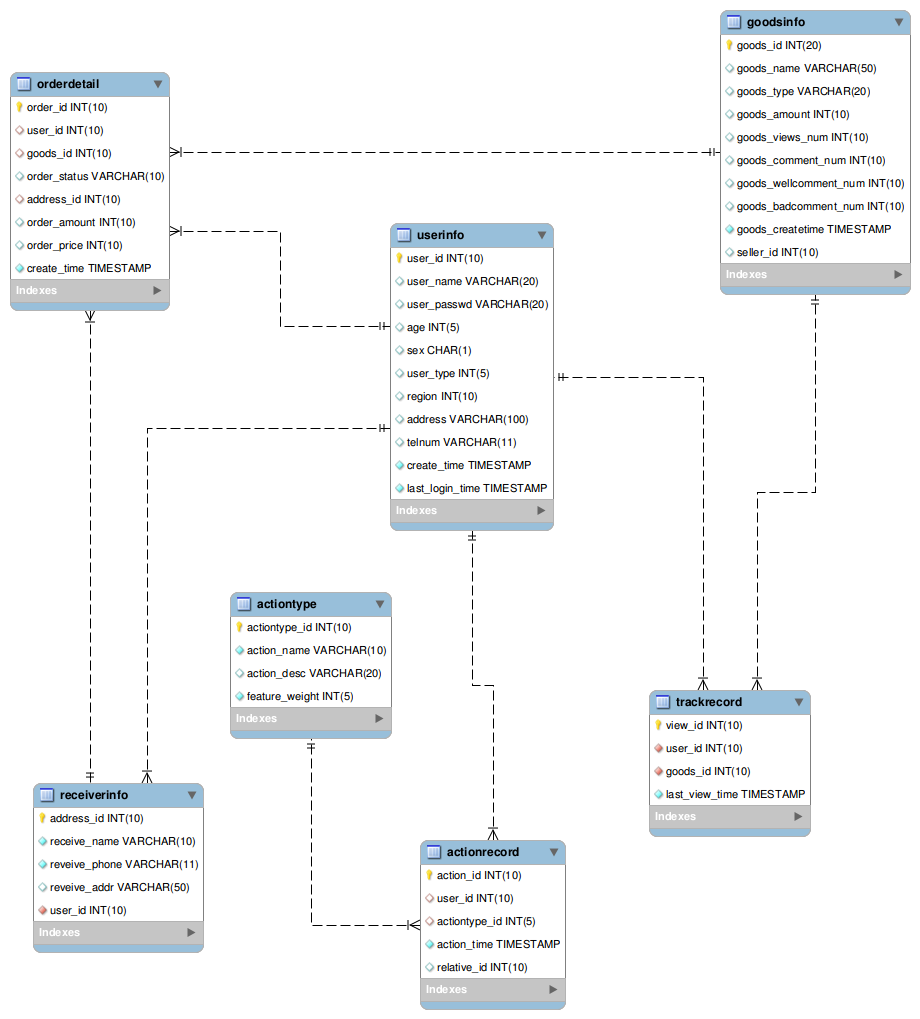
其中，表现层主要包括基础的商城功能模块，包括用户注册、登录、以及前端的商品页面显示；用户订单系统，主要用于用户管理商品订单，以及确认收货，申请退款，订单评价等相关操作；会员系统为会员用户提供相关服务和优惠活动；搜索系统基于独立的搜索应用服务器solr，支持中文分词，具有较高的查询性能；后台管理系统主要用于管理员进行商品发布，订单处理，用户管理等；最后的推荐系统，也是本文的侧重点，该系统通过对用户行为日志分析，结合用户历史标签信息，和当前实时的热门商品帮助用户挖掘可能当前可能感兴趣的商品，并推送给用户，从而实现用户与商家共赢。

服务层主要用于实现表现层相关的业务逻辑，根据不同的用户类型开放不同的用户权限，对于游客而言只能浏览和搜索商品，并可以添加物品到购物车中，推荐系统会根据短暂的用户的浏览记录以及购物车的商品，生成用户可能感兴趣的商品列表。对于已经登录的普通用户，除了包含游客用户的基本操作功能之外，还可以管理用户个人信息以及商品的订单，此时推荐系统可以掌握更多的用户数据和历史兴趣偏好，推荐的结果比游客状态更为精准。而对于会员用户而言，具备可以参与折扣商品的秒杀以及享受普通商品的会员价格。整个推荐系统推送的商品在服务层主要显示区域包括主页标题滚动的商品，主页右侧猜你喜欢的模块列表，以及当用户成功添加商品到购物车时底部推荐的相似商品等。

持久层主要有mysql集群以及分布式存储构成，为服务层提供数据基础。其中mysql集群用来存储主要的商品以及用户数据，而Hbase主要用于存储历史数据和大量的用户日志信息等。

## 4.3 数据库设计

  对于商城的基本数据存储我们使用了mysql数据库，并对于热点数据进行分库分表以增强数据库访问性能，所有的数据库表在满足三范式的前提下，各自独立，为了方便分离，放弃使用外键，由于整个系统数据库比较复杂，表数量较多，在此我们选取局部重要且与相互关系紧凑的表并导出E-R图如下图所示。



下面我们就对具体的表以及表中字段进行展开介绍。

（1）用户基本信息表，该表中记录了用户基本信息如用户名，用户登陆密码，以及用户的唯一表示user\_id，还有与推荐相关的用户年龄性别以及省市区域等。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| user\_id | int(10) | Y | 用户ID(PK) |
| user\_name | varchar(20) | Y | 用户名 |
| user\_passwd | varchar(20) | Y | 用户密码 |
| age | int(5) | N | 用户年龄 |
| sex | char | N | 用户性别 |
| user\_type | int(5) | N | 用户类型 |
| region | int(10) | N | 用户省市区域 |
| address | varchar(100) | N | 用户详细地址 |
| telnum | varchar(11) | Y | 用户手机号 |
| create\_time | timestamp | Y | 用户创建时间 |
| last\_login\_time | timestamp | N | 最后登陆时间 |

（2）商品信息表，其中goods\_id为自增主键，字段主要包含商品的基本信息，以及用户浏览和评价的相关的相关信息。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| goods\_id | int(20) | Y | 商品ID(PK) |
| goods\_name | varchar(50) | Y | 商品名称 |
| goods\_type | varchar(20) | Y | 商品类型 |
| goods\_amount | int(10) | Y | 商品总数 |
| goods\_views\_num | int(10) | N | 商品浏览总数 |
| goods\_comment\_num | int(10) | N | 商品评价总数 |
| goods\_wellcomment\_num | int(10) | N | 好评总数 |
| goods\_badcomment\_num | int(10) | N | 差评总数 |
| goods\_createtime | timestamp | Y | 商品创建时间 |
| seller\_id | int(10) | Y | 商家id |

（3）商品浏览记录表，由自增主键view\_id作为记录的唯一表示，由user\_id和goods\_id作为外键来唯一的确定用户所浏览的商品。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| view\_id | int(10) | Y | 浏览记录id(PK) |
| user\_id | int(10) | Y | 用户ID |
| goods\_id | int(10) | Y | 商品ID |
| last\_view\_time | timestamp | Y | 最后浏览时间 |

（4）订单记录，该表记录的用户的订单详情，包括商品对应的goods\_id，以及订单的数量和总计以及当前订单的状态等。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| order\_id | int(10) | Y | 订单ID (PK) |
| user\_id | int(10) | Y | 订单所属用户ID |
| goods\_id | int(10) | Y | 商品ID |
| order\_status | varchar(10) | Y | 订单状态 |
| address\_id | int(10) | Y | 收货地址ID |
| order\_amount | int(10) | Y | 订单商品数量 |
| order\_price | int(10) | Y | 订单总价 |
| create\_time | timestamp | Y | 订单创建时间 |

（5）收货地址表，该表保存的用户的所有收货地址，与用户表构成一对多的关系。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| address\_id | int(10) | Y | 地址ID(PK) |
| receive\_name | varchar(10) | Y | 收件人姓名 |
| reveive\_phone | varchar(11) | Y | 收件人手机号 |
| reveive\_addr | varchar(50) | Y | 收件地址 |
| user\_id | int(10) | Y | 地址所属用户的id |

（6）用户行为记录表，该表记录了用户的操作行为，以actiontype\_id作为外键与行为类型表做关联，来记录用户的购买、收藏、评论、搜索等操作，是推荐系统生成用户喜欢的重要依据。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| action\_id | int(10) | Y | 记录id(PK) |
| user\_id | int(10) | Y | 用户id |
| actiontype\_id | int(5) | Y | 操作类型 |
| action\_time | timestamp | Y | 操作时间 |
| relative\_id | int(10) | N | 行为关联id |

（7）行为类型表，该表内容由管理员手动添加，记录了所有需要后续持久化的行为类型，以及对应行为描述，对于不同的行为，设定不同的推荐权重来设置该行为对后期推荐的影响。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 字段类型 | 非空 | 备注 |
| actiontype\_id | int(10) | Y | 行为类型id(PK) |
| action\_name | varchar(10) | Y | 行为名称 |
| action\_desc | varchar(20) | N | 行为描述 |
| feature\_weight | int(5) | Y | 行为权重 |

## 4.4 推荐系统设计

推荐系统主要用于整合用户数据，进行相关计算，最后为应用层提供推荐服务接口，供应用层调用。该系统整体架构主要包含批处理层、实时处理层，数据存储层、应用层。其中批处理层用户进行复杂的耗时的数据处理，实时处理层主要是从消息队列中选择部分数据通过流计算快速获取一个临时解，从而实现准实时的推荐效果。数据存储层用于读取批处理层和实时处理层通过一定的合并策略（择优选择，累加等）将数据合并，支持随机读，可以在较短的时间内返回查询结果。应用层主要用于推荐结果的展示等一系列的业务逻辑处理。

该架构设计整合了大数据组件，具有高容错、低延时和可扩展性等特性。详细架构图详见图？-？。



**图** 推荐系统模块架构图

推荐系统通过实时收集用户相关操作（如浏览、搜索、收藏、评论、购买、加购物车）结合相关的推荐算法，生成推荐结果。

## 4.5 本章小结

本章介绍了系统的整体架构以及各个功能模块间的关系，并给出了相关模块的数据库表结构设计，最后对推荐系统的整体架构以及功能进行简要介绍。本章让我们从宏观上对整个系统有了大致了解，为后续详细设计奠定基础。

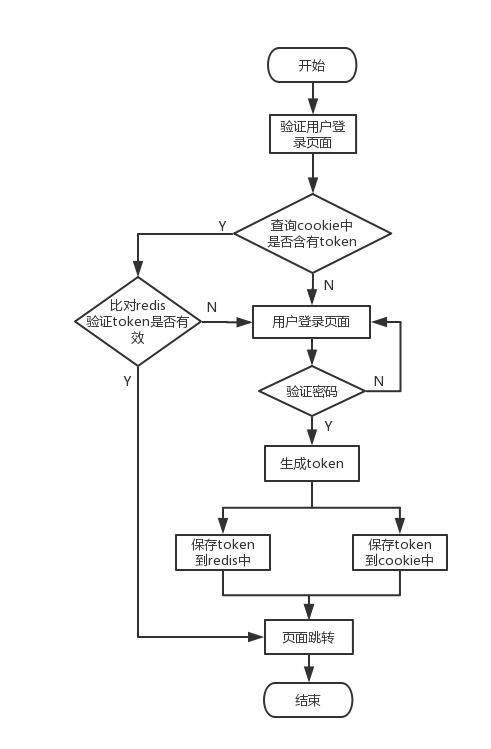
# 第五章 商城及推荐系统详细设计

## 5.1 商城基础功能模块设计与实现

### 5.1.1 单点登录模块设计与实现

由于整个系统由多个服务器节点组成，每个节点独立的运行系统的某个模块，因此设计了单点登录系统来实现多服务器共享session，考虑的session的查询需要较高的查询效率并且其本身具有时效性的要求，因此我们借助基于内存的数据库redis来实现session的功能。

单点登录模块流程图如下图所示：

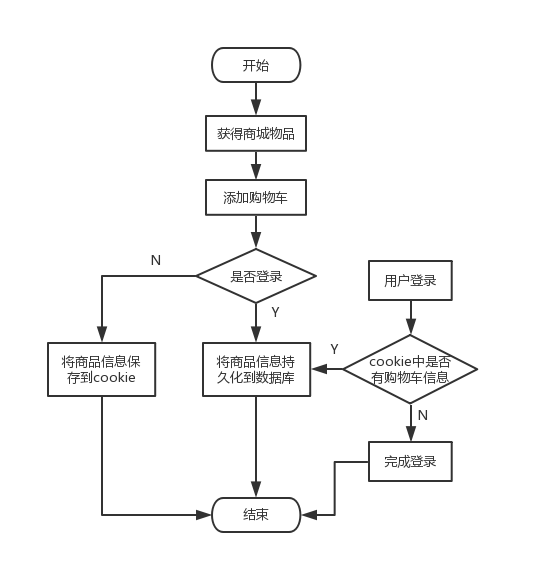


单点登录模块是替代了单服务器情况下的session的功能，当用户进入登录界面输入用户名和密码进行登录时，如果密码正确，登录的服务接口会根据用户名和当前时间戳通过MD5加密算法生成token，并将该token作为key，将用户对象的转换json字符串作为value存储到redis缓存中，并将token加入到用户浏览器的cookie中。

整个redis缓存作为认证服务器，可以被多个系统访问，当用户访问需要用户认证的服务接口，如购买、收藏、查看订单时，接口会访问认证服务器查看当前用户是否登录，如果没有登录，会自动跳转到登录界面。

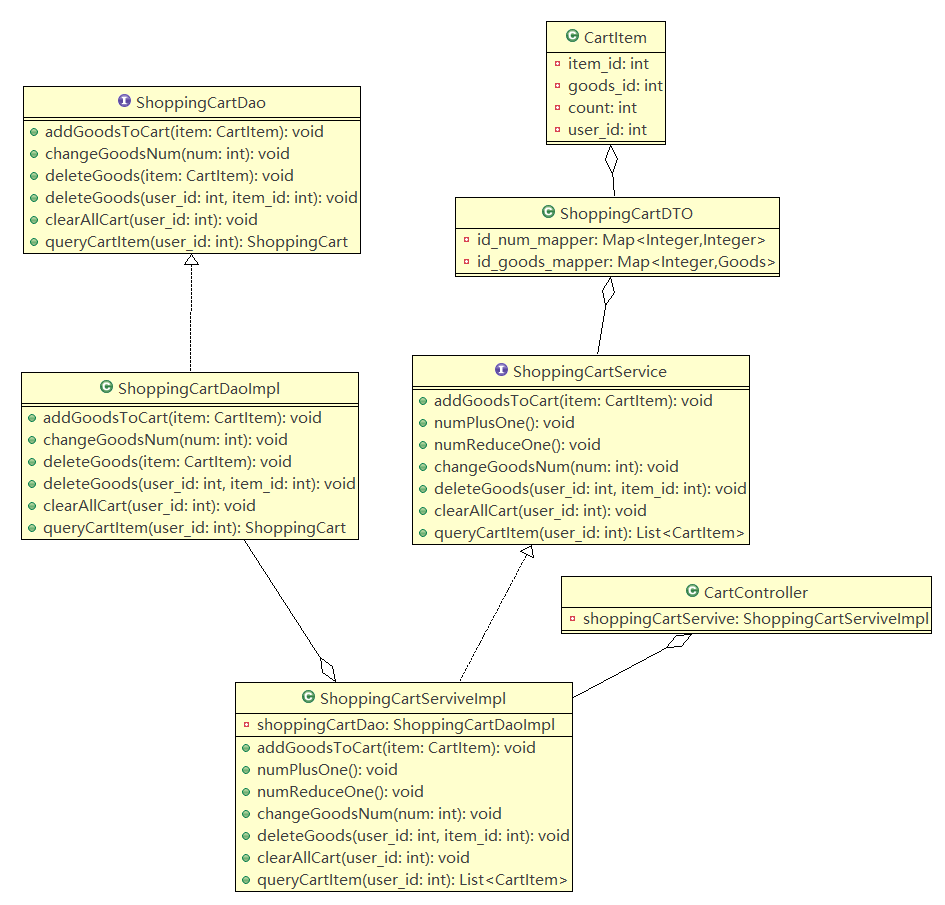
### 5.1.2 购物车模块设计与实现

购物车模块主要包括用户添加商品到购物车、查看购物车信息，修改购物车中商品数量，批量结算购物车中商品等。图？-？为添加商品到购物车的流程图。



当用户没有登录时，我们将用户的购物车内容以json的格式存储在cookie中，当用户登录后，我们先检查cookie是否购物车的key对应的value是否为空，如果不为空，我们会将购物车中的商品信息持久化到用户数据库中。

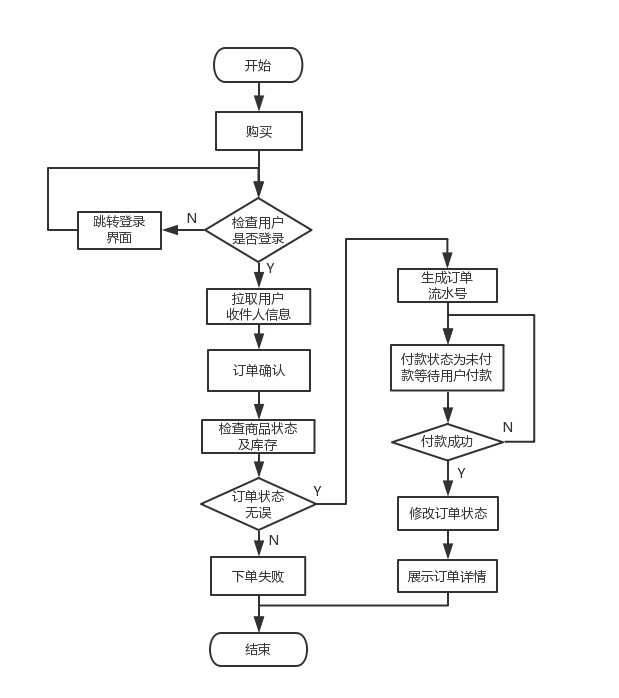
购物车模块相关类图如图？-？所示。

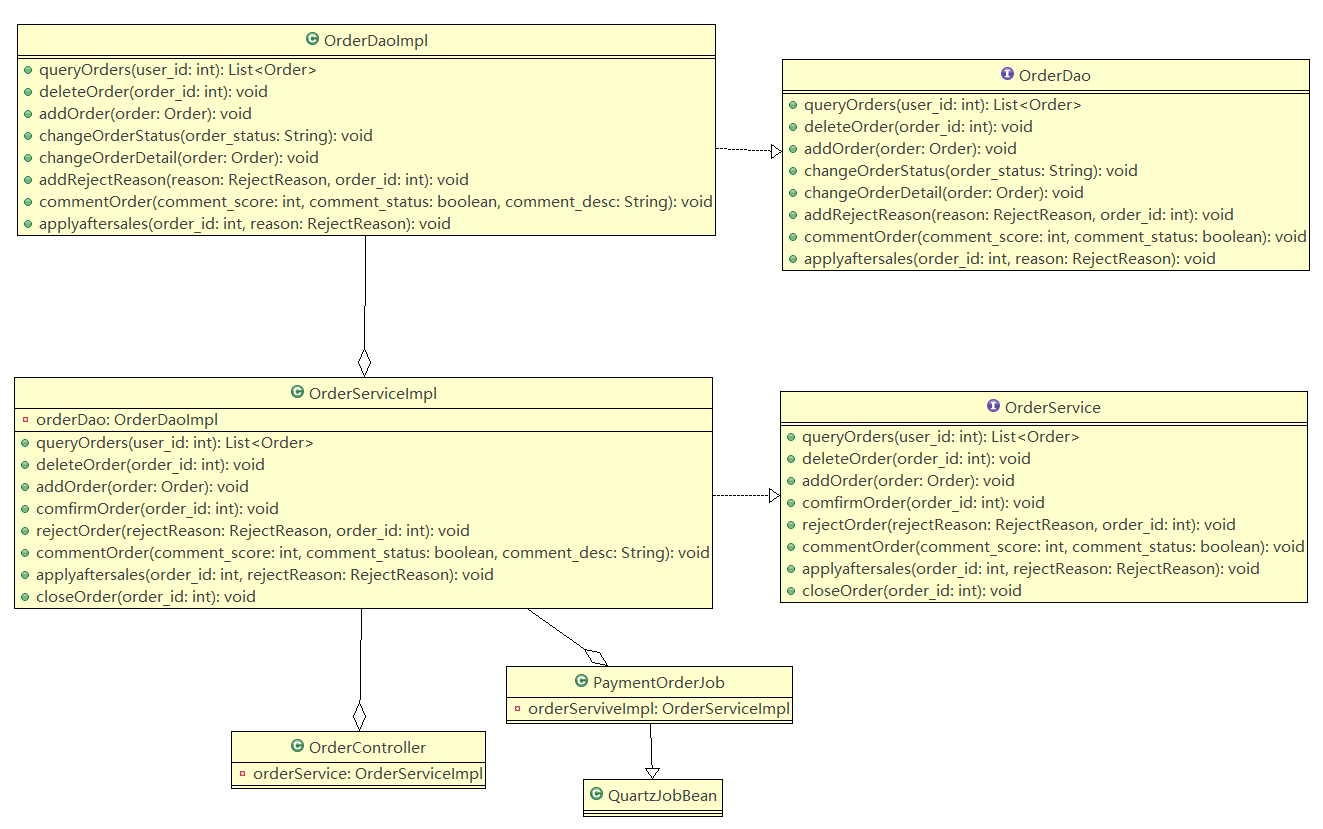


### 5.1.3 订单管理模块的设计与实现

用户订单管理模块主要为用户提供订单管理功能，主要包括商品的购买(订单生成)、订单取消、订单查询、未支付订单自动关闭、以及订单售后的相关服务。

下面主要针对订单生成功能进行详细介绍，具体流程如图？-？所示。



其中订单流水号由用户ID与时间戳组合而成。商品购买、订单确认、订单取消、订单详情等订单管理模块的相关类图如图？-？所示。

## 5.2 推荐系统相关模块设计与实现

### 5.2.1 分布式日志收集设计与实现

基于Spark平台的分布式日志框架如图3-4所示，该框架用于采集访问各个业务系统的日志信息，通过可靠的消息传送中间件作为数据计算与数据源采集之间的纽带，系统构建了基于Kafka集群的消息分发中间件，实现了日志数据的统一下发。由于日志数据中包含着各个业务系统的日志以及用户点击流的日志，在进入离线或实时推荐节点前，需要进行统一的数据清洗。本系统采用Spark Streaming技术按时间分片，对固定时间间隔内收到的数据进行统一的批处理，从而达到实时处理的效果，并具有很高的吞吐量。

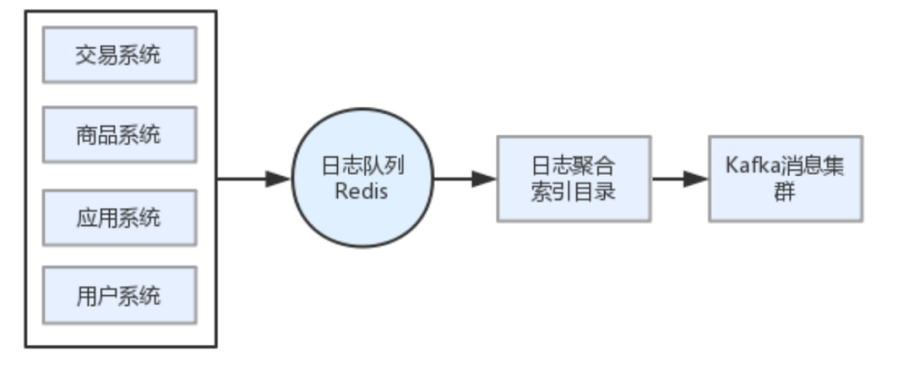


**图3-4** 基于Spark分布式日志框架

数据收集主要会经历如下阶段：日志数据采集，日志数据聚合和日志数据过滤。

1) 分布式日志实时采集

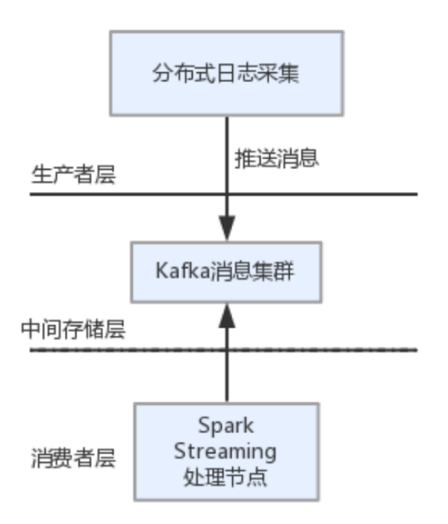
实时推荐系统需要大量的隐式的用户行为作为基础数据,而且每种用户的行为的源日志信息分布在不同的业务系统中，需要构建分布式日志汇总系统数据资源，将日志进行收集。以备后续流程使用。分布式日志采集如图3-5所示



**图3-5** 分布式日志采集模块

2) 基于Kafka集群的数据传输

通过构建分布式日志实时采集模块，完成用户行为日志的采集。但是在进入日志过滤阶段之前，由于日志流并发产生且数据量大，为了保证数据的实时性以及尽量减少数据丢失，使用分布式消息订阅和发布系统-Kafka,图3-6为基于Kafka集群的数据分发架构。



**图3-6** 基于Kafka集群的数据传输

### 5.2.2 离线计算设计与实现

### 5.2.3 实时计算模块设计与实现

对于推荐系统而言，随着应用场景的多样性，以及许多业务场景的特殊性，对推荐系统实时性的要求日益提高，如果推荐系统不能根据用户的行为变化和商品信息的变化做出及时的结果更新，推荐结果的有效性可能会大打折扣，用户可能会因此无法获取有效的符合自身需求的推荐结果。

传统的推荐方式采用定时离线计算来更新推荐结果，该方式计算周期往往以小时、天为单位。推荐给用户的商品或信息往往是滞后的、过时的，是基于用户前几个小时或者前一天的行为得到的结果，往往无法满足用户当前的需求，因此这就迫使实际的推荐系统实时性有本质的提高。

离线数据的存储过程是当新数据产生时，我们通过flume/sqoop（用于关系型数据库）将其收集到hdfs中，每半小时存放到hdfs的一个目录中，然后进行批处理（可以使用mapReduce或者spark）处理完后倒入impala, impala是一个sql引擎，能够通过sql语句查询存储在Hadoop的HDFS和HBase中的PB级大数据。

批处理层特点，因为写到了hdfs系统中，因此数据不可修改，只可以追加，可通过spark或map-reduce进行任意计算，支持水平扩展（通过扩展机器），缺点是高延迟，根据计算量或者数据量的不同，运行时间可能是几分钟到几个小时。具体设计架构见图3-1。



**图3-1** 离线计算系统架构

该模块主要承担两个主要职责，一、存储Master DataSet, 即一个持续增长的数据集。数据来源于系统的实时数据采集模块。二、对Master DataSet 进行预计算。离线计算的模式与批处理为主，我们使用Hadoop的Map-Reduce方式进行：

|  |
| --- |
| **批处理计算伪代码：** |
| **function runOffLineCompute():**  **while(true)**  **recomputeBatchViews()** |

利用离线计算模块进行数据预处理的主要作用是将实时计算的数据量尽可能的减小，从而有效地利用资源，改善实时计算的性能。但这里有一个前提，就是我们需要预先知道实时计算需要的数据，如此才能在离线计算中安排执行计划，定期对数据进行批量处理。此外，还要求这些预运算的统计数据是支持合并。

一、功能需求

(1) 数据收集

数据采集我们使用的是flume框架，该框架主要用于采集非关系型数据-如日志信息等，flume为收集、移动大量的日志数据提供了一种可靠的可用的分布式框架，它具有基于流式数据简单灵活的架构，并具有故障转移和恢复机制，为系统提供一定的可靠性和容错能力。flume需要在产生日志的agent结点上部署,用于监听节点产生的数据，agent节点可以将收集的数据传输给collector, 然后collector对数据流进行汇总（该设计的主要目的是为了减少hdfs中小文件的个数，减轻hdfs的并发压力），并将其写入hdfs中。



**图3-2** flume数据采集流程图

(2) 用户评分模块设计

该模块用户存储用户历史评分行为，为实时计算提供数据查询。本小节主要涉及到用户ID,商品ID,商品类别ID,用户-商品评分,时间戳五个字段，其中用户ID和商品ID分别代表用户和商品的唯一标识，用户-商品评分代表用户对商品的喜爱程度，范围是1-5的整数，喜爱程度与数值大小成正比。详细数据表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户ID | 商品ID | 商品类别 | 用户-商品评分 | 时间戳 |
| 1 | 2312 | 13 | 3 | 1502204779 |
| 2 | 4212 | 22 | 5 | 1502207499 |
| 3 | 3211 | 9 | 1 | 1502208499 |
| 4 | 4212 | 6 | 3 | 1503207900 |

当用户对商品进行打分时，我们会在相应的log文件中记录这条评分数据，包括表1的5个字段，其中两个字段间用双冒号连接，得到的数据格式为：

3::3211::9::1:: 1502208499。随后我们会将数据批量写入到HDFS中。供离线的批量计算使用。

(3) Spark离线计算数据准备

推荐系统离线计算由后台守护线程定期读取HDFS行为日志文件大小，若该文件大小超过指定阈值，便会触发离线计算部分。我们会读取HDFS的日志数据并将其转化成便于计算的DataFrame的数据结构。

|  |
| --- |
| spark分布式离线批处理部分源码 |
| case class RatingDate(userId:Int, itemId:Int, catId:Int, rating:Float,timestamp:Long)  def parseData(str:String): RatingDate = {  val fields = str.split(“::”)  RatingDate(fields(0).toInt, fields(1).toInt, fields(2).toInt, fields(3).toFloat,  fields(0).toLong)  }  //不定期根据URL读取hdfs数据  val conf = new SparkConf.setAppName(“test”)  val sc = new SparkContext(conf)  //将hdfs数据转换成DataFrame  val ratings = sc.testFile(“hdfs:”+url).map(parseData).toDF()  //进行相关算法计算并返回评级矩阵  //… |

## 3.2 推荐系统实时流计算

### 3.2.1 业务需求分析

对于推荐系统而言，随着应用场景的多样性，以及许多业务场景的特殊性，对推荐系统实时性的要求日益提高，如果推荐系统不能根据用户的行为变化和商品信息的变化做出及时的结果更新，推荐结果的有效性可能会大打折扣，用户可能会因此无法获取有效的符合自身需求的推荐结果。

传统的推荐方式采用定时离线计算来更新推荐结果，该方式计算周期往往以小时、天为单位。推荐给用户的商品或信息往往是滞后的、过时的，是基于用户前几个小时或者前一天的行为得到的结果，往往无法满足用户当前的需求，因此这就迫使实际的推荐系统实时性有本质的提高。

### 3.2.2 功能需求和系统整体结构

一、功能需求

(1) 数据收集

要完成用户实时推荐的计算任务，第一步，要对数据进行采集和清洗操作。对于实时推荐系统的计算需要两部分数据集，一部分是用户之前的离线的数据，第二部分是用户最新的实时数据。其中实时数据主要包括用户ID,用户最新的行为日志（如：用户对商品和评价操作，即用户-商品信息，包括用户ID，商品ID，商品分类，评分等级，时间戳字段等）。

(2) 数据计算

实时流计算是指对源源不断的数据流进行系统的连续计算，并实时的返回结果。为了确保计算的精确性和有效性，就要求系统提供实时的用户行为信息，然后通过相关计算，将结果写入Hbase中，为后续推荐操作提供推送和查询。

二、性能需求

(1) 低要求，高效率，可扩展

对于该系统使用的大数据模型采用分布式框架，支持良好的横向扩展性，对于并发的提高只需要横向扩展机器即可。

(2) 高容错性，高可靠性

对于分布式数据集的容错方式一般来说有两种，数据检查点和记录数据的更新。Spark采用记录更新的方式。但是如果更新粒度太细太多，则记录更新成本也会相应的增加。因此spark提供的弹性分布式数据集RDD具有数据流模型的特点：自动容错，位置感知调度和良好的可伸缩性。与分布式共享内存系统需要付出高昂代价的检查点以及回滚机制不同，RDD通过Lineage来创建丢失的分区：RDD记录了单个块上执行的单个操作，每个RDD都记录了他变换的所有操作，以便恢复丢失的分区，这种容错机制有着较低的资源开销和一定的性能提升。

三、系统设计

实时流计算作为加速层主要用于实时处理，缺点是不能进行复杂计算。

实时计算层的特点主要是对数据进行流式处理，持续性计算，存储和分析某个窗口内的数据，该方式通常是基于实时性和准确性的一种折中，因此准确性有时并不高（临时解），后期最终的结果，会通过批处理获取，批处理会覆盖掉流式处理的结果。具体架构图详见图3-3：



**图3-3** 实时计算系统架构图

其中对于新产生的数据会收集用于离线计算部分，因为实时计算不允许进行耗时的复杂计算。因此其中一小部分数据会选择性的提交给kafka消息总线，用于实时计算。对于数据收集部分会通过节点上部署的flume上进行，而且flume与kafka也有很好的兼容性，而对于flume的数据收集部分在之前离线计算章节已经有详细的介绍。

## 3.3 其他模块详细介绍

### 3.3.2 推荐引擎组

本系统在线计算部分与具体的推荐目标相关，需要在离线计算的基础上，以最快的速度给出推荐结果。在线计算部分计算量不是很大，数据量也不多，引入spark后，先将输入载入内存，来提高推荐速度。通过部分离线数据和实时的用户行为数据进行在线计算，得到的的结果就是推荐引擎的输出。在经过一系列处理后，就可以将结果存入Hbase并提供对外查询接口。

****

**图3-7**推荐系统整体架构

### 3.3.3 推荐系统整体架构

## 3.4 本章小结

本章节主要对推荐系统架构设计进行了详细的分析，以离线计算和实时计算为主线，介绍了业务需求、功能性需求等，并描述了相关模块的主要业务功能。最后对于重要的核心模块进行了详细介绍。

# 第4章 关于推荐算法的相关问题研究

电子商务在过去十年中的日益普及，以及互联网信息量的迅速增长推动了推荐系统的发展，通过推荐计算帮助用户找出最符合用户需求的产品成为推荐系统的核心目标和重要挑战。推荐系统系统将数据分析技术用于帮助用户在网站中找到他们需要的信息，产生用户对项目的预期得分和给用户的N类推荐项目列表。近年来提出了许多推荐系统算法的实现，其中基于协同过滤[22]的算法被发现是非常有效的，协同过滤算法的基本思想是根据其相似用户或产品通过相似度计算提出对项目的建议和预测。下面章节我们将重点介绍早期基于用户的协同过滤推荐算法和基于Item的协同过滤推荐算法的研究，通过矩阵分解和Item正则化对后者在算法上进行改进，并使用异步分布式的计算方式优化算法收敛时间。并对于系统冷启动问题提出相应的解决方案。

## 4.1 基于用户的相似度计算

基于内存的算法是利用整个用户-项目数据库来生成预测的。使用统计技术来找到具有与目标用户相似的历史的一组用户（称为邻居，他们给项目打分具有一定的相似性，或者倾向于购买相似的项目集合）[23]。 一旦形成了用户邻域，就可以使用不同的算法来预测邻居的偏好并产生对活跃用户的预测。这些技术，也称为最近邻或基于用户的协同过滤，在以往的实践中很受欢迎并被广泛使用。

基于用户的协同过滤系统在过去的研究已经取得了成功，但它们的广泛应用逐渐暴露出很多缺陷和不足[24]，如：

(1) 稀疏性。实际上，许多商业推荐系统被用于评估大型项目集（例如，淘宝（taobao.com）推荐的商品和网易云音乐推荐的歌曲或专辑。在这些系统中，活跃的用户可能甚至只购买了不到0.01％的项目。因此，基于最近邻算法的推荐系统也许不能为特定用户做出任何有价值的项目建议。因此，推荐的准确性可能很差。

(2) 可扩展性。最近邻算法需要随着用户数量和项目数量的增长而增长。 拥有数百万用户和项目，典型的基于Web的推荐系统使用现有算法时将遭受严重的可扩展性问题。

对于大型稀疏性数据库，最近邻算法的弱点使得我们探索其他推荐系统算法。我们的第一种方法试图通过将半智能过滤代理纳入系统来弥合稀疏性问题。这些代理使用语义特征评估每个项目。通过提供一个密集的评级集合，有助于减轻覆盖面并提高质量。然而，过滤代理的解决方案没有完全解决稀疏评级矩阵与用户之间关系差的根本问题。在本文中，我们研究了另一种技术，其主要思想是分析用户-项目矩阵以识别不同项目之间的关系，然后使用这些关系来计算给定的某对用户的预测分数。主要思想是用户有兴趣购买与用户之前购买项目相似的项目，并且倾向于避免/不予以推荐类似于用户之前不喜欢的项目。当请求推荐时，这些技术不需要识别类似用户的邻域。因此，他们倾向于给予更快的推荐结果。

## 4.2 基于Item的协同过滤算法

由于基于用户的推荐在实际项目中有诸多缺陷[25]，在本节中，我们研究了一类基于项目的推荐算法。与4.1节中讨论的基于用户的协同过滤算法不同，基于项目的方法根据目标用户已评价的项目集合，计算它们与目标项目i的相似度{si1，si2，….，sik}，然后选择k个最相似的项目{i1，i2，…..，ik}。一旦经过计算得出相似项目的列表，则通过对这些相似项目采用目标用户的评级的加权平均来获得预测结果。我们在这里详细描述这两个方面，即相似度计算和预测生成。

### 4.2.1 项目相似度计算

基于项目的协同过滤算法的一个关键步骤是计算项目之间的相似度，然后选择最相似的项目。两个项目i和j之间的相似度计算的基本思想是首先隔离对这两个项目进行过评分的用户，然后应用相似度计算技术来确定相似度si,j。图4-1说明了这个过程，这里矩阵行表示用户，列表示项目。



**图4-1 共评项目的隔离与相似度计算**

计算项目之间的相似性有许多不同的方法。在这里我们提出三种这样的方法。这些是基于余弦的相似性计算，基于相关的相似性和调整余弦相似度。

(1) cosine相似度计算

在这种情况下，两个项目被认为是m维用户空间中的两个向量。它们之间的相似性是通过计算这两个向量之间的角度的余弦来获得的[26]。形式上，在图4-1中给出的个等级矩阵中，项目i和j之间的相似度表示为sim(i,j)，sim(i,j)由以下公式给出：

(公式4-1)

(2) 皮尔逊相似度

通过计算Pearson-r相关系数corrij来测量两个项目i和j之间的相似度。 为了使相关性的计算更加准确，我们必须首先隔离被同一用户评分的item（即用户同时对i和j进行的评分）。具体定义如下：

(公式4-2)

其中U表示用户u,i共同评价的物品，表示u对i的评分。R̅i表示所有评价物品的平均分。

(3) 调整余弦相似度

基于用户的协同过滤算法和基于项目的协同过滤算法之间的相似度计算之间的一个根本区别在于，在基于用户的CF的情况下，沿着矩阵的行计算相似度，而基于项目的CF的情况下，是沿着矩阵列计算相似度。在基于项目的案例中使用基本余弦测量的计算相似度有一个重要缺点，未考虑到用户评分尺度问题。调整的余弦相似性通过从每个共同对中减去相应的用户平均值来尽可能抵消这个不足。一般来说，使用该item i和j之间的相似性由下式给出:

(公式4-3)

其中，R̅u是第u项用户评级的平均值。

### 4.2.2 预测计算

协同过滤算法中最重要的一步是根据预测生成推荐的接口。一旦我们基于相似性度量来找到最相似项目的集合，下一步就是研究用户评级，并使用一种技术来获得预测。这里我们考虑两种这样的技术。

(1) 加权求和

顾名思义，该方法通过计算用户对类似于i的项目给出的评分的总和来计算用户u对项目i的预测。每个评级是由项目i和j之间的对应相似度si,j加权求和获得。使用图4-2所示的概念，我们可以表示预测Pu,i的公式为

(公式4-4)

基本上，这种方法尝试了解活跃用户如何评价相似项目。通过相似项的加权和，以确保预测在预计范围内。



**图4-2** Item-based 协同过滤算法预测产生过程

(2) 回归

该方法类似于加权求和的方式，使用基于回归模型的评级来计算相似度[27]，而不是直接通过相似项目的评分的方式计算。实际上，使用余弦或相关性度量计算出的相似度可能会产生误差，因为两个评估矢量可能是相差甚远的（欧几里得），但仍可能具有非常高的相似度。在这种情况下，使用所谓的“相似项目”的原始评级可能导致较差的预测。因此基本思想是使用与加权求和技术相同的公式，而不是使用相似项目N的“原始”评分Ru, N的值，该模型基于线性回归模型使用其近似值。如果我们用Ri和RN分别表示目标项目i和相似项目N的各自 的特征向量，则线性回归模型可以表示为

(公式4-5)

其中，回归模型的参数α和β的值是通过两个评估向量来确定。是回归模型的误差。

## 4.3 基于item的正则化和异步分布式矩阵分解

### 4.3.1 算法概述及引入原因

该方法的主要思想是将用户评分矩阵分发在不同的机器上，每个机器只能访问一部分信息，通过网络异步的传播并通过随机梯度优化的方式进行更新。每当机器接收到一个参数向量时，它就会将当前参数的特征向量与所接收特征向量进行平均，并从该点继续进行迭代。此外，我们还使用了基于正则化的相似度计算方式，使得用户和项目的特征向量逼近于相似用户和项目的特征向量。我们分析了正则化对MovieLens（100K，1M，10M）和NetFlix数据集的影响，实验结果表明它对于计算结果的均方根误差（RMSE）和平均绝对误差（MAE）有明显的改善，并且与同步分布式方法相比，异步分布式方法在收敛时间方面显着提高。

上一章节我们提到基于item的协同过滤推荐算法，其推荐结果是较为精准和有效的，其基本思想是，如果两个用户对一组项目具有相似的喜好程度，则它们可能在新项目上也具有相似的喜好程度。基于这个想法开发的的协同过滤技术有：(1) 邻域方法[28]，使用相似用户或项目的过去的评分生成对用户未知物品的预测。(2) 矩阵分解技术[29]，其目标就是把用户-项目评分矩阵分解成用户因子矩阵和项目因子矩阵乘的形式。

在本章节中，我们将分解后的矩阵分发到集群的各个服务器中，并提出了一种矩阵分解异构框架，使用SGO（随机梯度算法）的方式进行推荐。在同步计算场景中，由于网络中的机器计算速度各不相同，因为每台机器的每个阶段都必须等待最慢的机器计算完毕后才能进行下一步的计算，因此同步更新一般非常慢。在我们的异步框架中，每个机器并行执行SGO来计算评分矩阵的不同部分，并将其计算的权重发送到网络中的其他机器。一旦机器收到一个更新的权重，它会使用自己的更新对参数权重进行平均，并继续进行下一阶段的SGO运算。这样可以保证每台机器都能够全面了解整个数据的计算情况。

我们仍然使用Movielens和Netflix数据集进行测试，实验表明与同等的同步分布式方法相比，实验结果表明异步分布式更新方案具有更好的的收敛性。

### 4.3.2 用户矩阵分解和基于Item的矩阵正则化

在本章节中我们将介绍矩阵分解中的基本定义和目标。常见的矩阵分解主要包括基本矩阵分解(basic MF),正则化矩阵分解,以及基于概率的矩阵分解。这里我们主要介绍正则化矩阵分解，该方式可以解决base MF的过拟合问题，是较为常用的矩阵分解方式。我们用集合U和I分别表示用户和项目的数据集，其中ui和ij分别是第i个用户和第j个项目。此外，我们用rij表示用户ui的项目ij的评分，相应的评分矩阵R为rij的矩阵。用户（相应的项目矩阵为），pi（和qj）是对应于第i个用户（第j项项目）的特征向量，它们分别构成用户矩阵P和项目矩阵Q。

随机梯度优化（SGO）的矩阵分解[30]，其前提是用低维度因子矩阵PQT的乘积来代替高维矩阵R.这些因子矩阵试图对k个隐含因子的项目进行用户偏好建模。对于一对用户和项目(ui，ij)，如果存在该用户对该项目的评分rij，则在原有的损失函数的基础上加入正则化因子，整体作为损失函数：

(公式4-6)

其中λ≥0是正则化参数。 因此，整个矩阵分解问题转化为：

(公式4-7)

注意，误差仅依赖于矩阵P和Q; 然而，项目ij由用户ui来评价，因此最佳因子qj取决于pi和pi'。随机梯度优化（SGO）方法可表示为：在每次迭代计算中)随机选择一对用户和项目(uit,ijt)，并确保用户对该项目做出过评价)采用梯度下降的方式求解U和I。

为了加强相似用户具有相同喜好的假设，我们强调每个用户的特征向量应接近于其类似用户的平均特征向量。为了计算最相似的用户（或项目），我们考虑了对两个用户ui和uj的Pearson相关系数做出修改，公式如下：

(公式4-8)

这里，Ic是由两个用户共同评分的项目，和分别表示ui和uj的平均评分。因此，我们可以找到N个最相似的用户ui，用Ni表示（对应相似项目为ij）。我们对用户评分的目标函数做适当修改，对于对应评分rij存在的一对用户和项目(ui,ij)，引入相似性正则化项后的目标函数为：

(公式4-9)

其中≥0和≥0分别是与相似用户和相似item相关联的正则化参数，即正则项的权重，具体取值需要根据数据集调参获得。执行与传统SGO相同的更新，但将l替换为l1，我们得到算法l。

|  |
| --- |
| 算法l：基于正则化的User-Item相似度计算 |
| 入参：R,,, 初始化：随机给出P、Q |
| Procedure Modified SGO  While not converged do  Choose randomly(ui ,ij) ∈R  Ni = GetSimilarUsers(i,N)  Ni = GetSimilarItems(j,N)  Update pi and qj by a gradient step on l1() (见公式4.9) |

因此(公式4-8)转化为基于正则化的相似度计算：

(公式4-10)

### 4.3.3 异步分布式随机梯度优化

即使通过随机梯度优化在推荐系统数据集上能够达到较高的推荐精度，但它仍然存在与之相关的一些计算挑战。在单个机器上顺序执行SGO算法需要较长的收敛时间。 因此，需要以分布式方式对大型数据集进行随机梯度优化。对于并行化SGO而言，直接实现的一个缺点是因子矩阵的更新可能不是独立的。 例如，对于位于相同行（或列）的训练点，它们需要同时更新P和Q矩阵对应行（或列）。因此，对于同步更新因子矩阵而言计算节点之间的有效通信是非常必要的。

一种常见的方法是将评分矩阵划分为几个块，并将分割后的矩阵分发到多个服务器上进行SGO计算。在每个机器会根据对应的评级来更新因子矩阵。虽然每个机器上的评级矩阵都各不相同，但是因子矩阵之间的更新是同步的。因此，在每个阶段，每个机器中计算得到的因子矩阵是同步的。我们将该方法称为“同步SGO”，由于所有机器会每个计算阶段完成之后同步它们的因子矩阵，因此是较为低效的，主要是因为同步SGO的每个服务器的各个阶段的计算时间是随其机器的配置不同而不同。因此，在每个计算阶段之后，这种机器之间的同步更新操作会变得滞后，因为更快的机器必须等待较慢的机器计算完成才能进行同步操作。

根据上述的不足之处在此提出了一种基于参数矩阵的异步传播的分布式框架。每个机器被分配了相似用户评分矩阵的一部分，我们可以通过大规模图形聚类技术离线找到对应的分割矩阵。 每台机器仅通过其本身的评分矩阵来更新因子矩阵，与同步SGO方法相反，只要其中一台机器完成一个时期的计算，就会广播给集群中所有的机器更新其相应的因子矩阵。因此，每当一个新的时期开始，它都会收集其他机器中最新的因子矩阵，并且在完成一个时期计算之后，将其更新的因子矩阵广播到所有其他机器，并立即开始下一个时期，而不需要考虑其他机器的计算状态。 这样，所有的机器都是相互独立运行的。 因此，更快的机器将更快地执行，而较慢的机器将在时间上落后，但在完成每个时期之后，它们将从更快的机器接收最新的因子矩阵。



**图4-3**异步分布式SGO框图

基于MovieLens和NetFlix数据集，我们对不同的方法计算其MAE和RMSE。详见下表：

表4-1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | | SGO | | 基于正则化的SGO | | | |
| MAE | RMSE | Similar Users and Similar Item | | Similar Users | |
| MAE | RMSE | MAE | RMSE |
| ML-100LK | ua | 0.7380 | 0.9469 | 0.7388 | 0.9328 | 0.7396 | 0.9355 |
| ub | 0.7592 | 0.9580 | 0.7555 | 0.9557 | 0.7539 | 0.9580 |
| ML-1M | ra | 0.7314 | 0.9699 | 0.7207 | 0.9514 | 0.7166 | 0.9476 |
| rb | 0.6882 | 0.8859 | 0.6894 | 0.8780 | 0.6510 | 0.8398 |
| ML-10M | rb | 0.6491 | 0.8409 | 0.6479 | 0.8382 | 0.6510 | 0.8386 |
| NF-Subset | NA | 0.6500 | 0.8277 | 0.6459 | 0.8257 | 0.6485 | 0.8265 |

在我们的设置中，我们将等级矩阵R划分到m行新的矩阵中，其中m是可用的机器（或进程）的数量。P矩阵也以相同的方式被阻塞。每一个计算节点都有一个R矩阵，对应的P矩阵和基于R矩阵的本地Q矩阵。在每个计算时期，我们基于R矩阵和各个误差函数来更新P和本地Q矩阵的块。整个过程如图4-4所示，整个网络由三个计算节点组成，R1，R2和R3表示每台机器上的评分矩阵。ep代表了每个时期的结束和不同机器上新时期的开始。如图所示，在一个时期结束后，每个机器将其更新的Q矩阵广播到给集群中的其他机器，而在新时期开始时，每个机器将通过接受到的其他机器更新的Q矩阵来更新自身的Q矩阵。通过这种方式，计算较快的机器产生的更新矩阵可以帮助较慢的机器更快地收敛，从而改善整个分布式网络的收敛时间。

## 4.4 关于算法冷启动问题的研究

### 4.4.1 冷启动问题概述

### 4.4.2 一种解决项目冷启动问题的推荐算法

冷启动问题[31]有很多解决方法，本文采用的方法主要是将时间信息加入到推荐算法中。我们都知道，现实世界中，用户群体一般分为两类：一类用户即所谓的活跃用户，这类用户的特点是喜欢追逐并评价新鲜事物；另一类用户即所谓的消极用户，这类用户更多的是追随角色。在这种情况下，积极用户对项目的评价时间往往在项目发布之后很短的一段时间内，而消极用户的反应则要相对滞后。由于这两类用户的普遍存在，推荐系统应用的时候会考虑把新项目优先推荐给积极用户，以此来带动评价。我们同时把时间信息[33]加入到系统中后，冷启动问题也就得到了解决。

由于时间信息对推荐算法有重要的影响，因此我们对用户的时间权重作了如下定义:

时间权重=

其中sum表示用户评价过的项目总数；timeui 表示用户评价的第i个项目的时间，datai 表示项目i发布的时间。则用户u的时间权重Wu 计算公式如下：

(公式4-11)

由此可知，timeui - datai越小，说明项目评价时间与项目发布时间的距离越近，用户越积极。反之亦然。

与实际项目为例，用户评价时间表如下：

表4-2 用户评价时间表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Item | U1 | | U2 | | U3 | |
| Time | Date | Time | Date | Time | Date |
| i1 | 2016 | 2015 | 2015 | 2014 | 2017 | 2017 |
| i2 | 2016 | 2014 | 2016 | 2013 | 2017 | 2014 |
| i3 | 2015 | 2012 | 2017 | 2014 | 2017 | 2016 |

以表一为例，采用公式1计算得到每个用户的时间权重wui即：

根据如上计算可知，用户积极程度由大到小依次是Wu1>Wu3>Wu2，即用户Wu1更偏爱于新发布的产品。

此外我们还根据项目标签信息预测评分值[34]。

1) 基于项目评分信息预测评分值

这里，我们假定有一个用户u，分析用户-项目评分矩阵可以发现用户与项目之间存在评价与未评价，以及评价次数的关系。对于评价与未评价的关系，我们设置一个变量ruj，当ruj的值为1时表示用户评价过该项目，0则表示未做评价。对于评价次数，本文设定以ICk来表示用户评价的个数，UCj表示某一项目被共同评价的次数。在这个关系中，用户评价项目的个数和项目被评价次数可以从用户-项目评分矩阵中得到。根据用户对项目的评价个数、评分信息和某一项目被用户评价的次数，我们可以定义基于项目评分信息的预测评分值，从而观察和研究他们间的关系。具体公式如下:

(公式4-12)

2) 基于项目属性信息预测评分值

如公式所示表示基于用户u对项目j的预测评分值。如果用户u 对项目 j 做了评价ruj =1, 反之为0。如果项目j具有属性l，则hlj=1，反之为0。

(公式4-13)

3) 基于项目标签信息预测评分值

基于项目标签信息的预测评分计算公式如下：

(公式4-14)

公式(3-4) 中， 表示基于标签的用户u对项目j的预测评分值。如果标签g标注的项目j，则bgj=1分之为0。

下面我们基于具体的数据矩阵和公式来围绕研究对象u1计算预测评分值。

表4-3 用户项目评分矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| rij | i1 | i2 | i3 | i4 |
| u1 | 2 | 0 | 4 | 0 |
| u2 | 0 | 3 | 1 | 0 |
| u3 | 5 | 0 | 2 | 0 |

表4-5 标签-项目标注矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bij | i1 | i2 | i3 | i4 |
| t1 | 2 | 3 | 0 | 0 |
| t2 | 0 | 3 | 1 | 0 |
| t3 | 1 | 0 | 0 | 0 |
| t4 | 0 | 2 | 1 | 0 |
| t5 | 0 | 0 | 2 | 0 |

表4-4 属性-项目矩阵 表4-5 标签-项目标注矩阵

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hij | i1 | i2 | i3 | i4 |  |  | bij | i1 | i2 | i3 | i4 |
| a1 | 1 | 0 | 1 | 0 |  |  | t1 | 2 | 3 | 0 | 0 |
| a2 | 1 | 1 | 1 | 0 |  |  | t2 | 0 | 3 | 1 | 0 |
| a3 | 1 | 0 | 1 | 1 |  |  | t3 | 1 | 0 | 0 | 0 |
| a4 | 0 | 1 | 0 | 1 |  |  | t4 | 0 | 2 | 1 | 0 |
| a5 | 0 | 1 | 0 | 1 |  |  | t5 | 0 | 0 | 2 | 0 |

其中{i1,i2,…in}为n个项目的集合，{u1,u2…um}表示m个用户，rij表示用户ui对项目j的评分。{a1,a2…,an}表示p个属性的集合，hij表示项目j是否具有an属性。{t1,t2…,tq}表示q个标签的集合,bij表示项目j被标注的次数。为了更好的理解公式与表之间的关系，使用两个三分图的形式来表示。



**图4-4** 用户-项目-标签信息 **图4-5** 用户-项目-属性信息

因此我们可以根据上述理论基础，分别根据用户-项目-标签信息和用户-项目属性信息来挖掘他们之间的潜在关系。从而通过预测评分值来解决新项目的冷启动问题。

根据之前的分析发现，结合用户、标签、项目属性提出的个性化推荐算法从理论上可以提高推荐系统的准确率。在此，为了更好的观察用户评分信息、项目属性信息，项目标签信息以及用户评分时效性信息在推荐算法中所产生的影响，我们定义权重, 其中 得到预测评分公式如下：

(公式4-15)

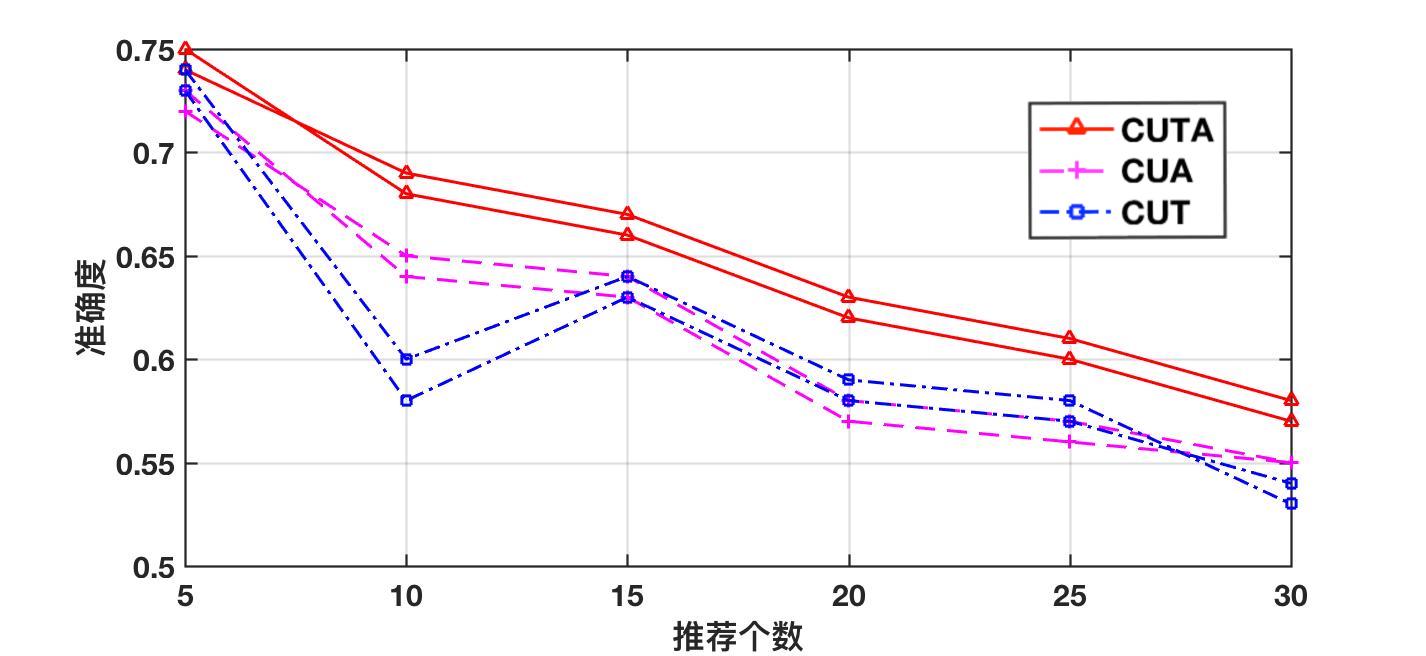
由公式4-15可知，该个性化推荐充分利用了用户、标签以及项目属性的相关信息。当, 会综合考虑用户、标签与项目属性信息进行预测评分计算，即用户、标签与项目属性的个性化推荐算法(consider user, tag, and item-attribute personalized recommendation algorithm, 简称CUTA)。同理，当时为用户项目属性推荐算法（简称CUA），当时，为用户标签信息推荐算法（简称CUT）。

为了解决推荐系统冷启动问题，我们在上述基础上加入了用户时间权重信息，得到CUTATime算法，该算法会在获取基于项目评分(CUTA)、基于标签评分(CUT)、基于项目属性信息(CUA)得到的预测评分值的基础上通过分析用户评价项目的时间信息，获取用户时间权重，并结合之前得到的预测评分实现推荐。

实验一：不考虑新项目的不同推荐算法的准确度对比

为了进一步验证我们提出的算法的性能，我们对上述的三类推荐算法进行了相关的实验对比，我们使用MovieLens数据集，该数据集主要包括用户对电影的评分信息，电影的属性信息以及电影的标签信息等。我们从该数据集中选取至少被30个用户同时评论的电影，以及至少评论过10部电影的用户.经过筛选我们得到2035个用户以及3200部电影并将其50%划分为训练集，另外50%为测试集。

通过调整不同的的取值，并针对近邻个数为5，10，15，20，25，30的情况下进行了相关实验，实验结果如下图所示：



**图4-6** 不同推荐算法的准确度对比

通过观察我们可以发现，CUTA算法的准确度明显高于CUA和CUT算法，而当邻域个数大于15时，调整的取值对CUA和CUT推荐的准确率并无明显影响。因此我们可以发现，在不考虑新项目的前提下，CUTA的准确率明显优于CUA和CUT算法。

实验二：解决冷启动问题的相关实验对比

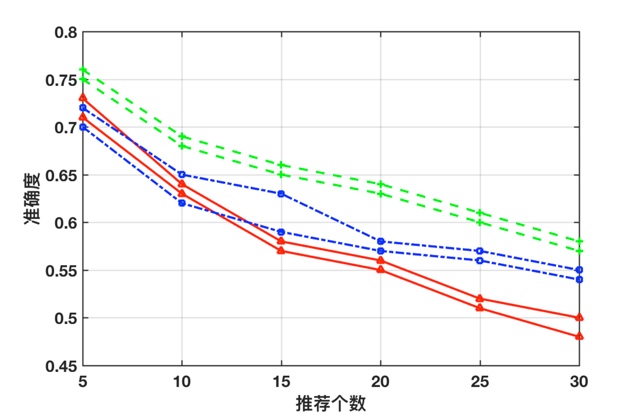
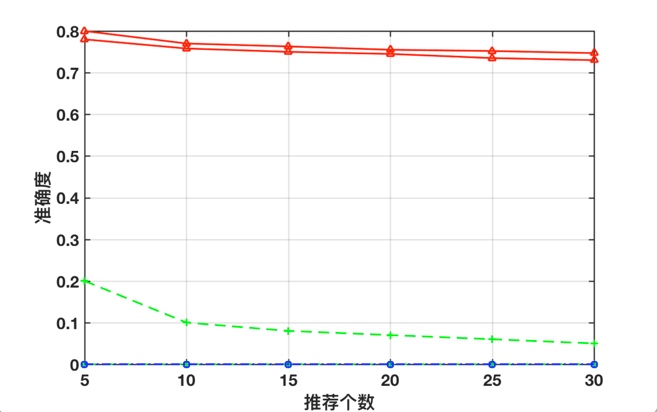
对于冷启动的问题，即通常用户并未曾对项目做出过任何评价，所以在数据集中并不会存在用户对项目的任何评分信息。因此当面临冷启动问题，我们主要会通过项目的基本属性信息和用户的时间权重两项综合来为用户推荐项目。

这里我们借鉴了一个新的概念：新颖度[35]，这里所说的新颖度是指在给目标用户推荐的列表中，新项目的个数占推荐项目的总数的度，即一个比值：

novelty = Recommended\_SUM /SUM (公式4-16)

其中novelty表示新颖度，Recommended\_SUM表示推荐列表中包含的新项目的个数，SUM表示项目总数。

首先我们将实验一中的测试集分成两部分，其中一部分是随机抽取150个项目当作新的项目，同时将训练集中这些对应项目的评分信息和标签信息全都置为0，其余未被抽取的部分为另一部分。的取值和实验一取值相同，并在不同邻域的情况下进行相关实验。

**图4-7**解决冷启动问题算法的准确度对比 **图4-8**解决冷启动问题算法的新颖度对比

从实验结果(图4-7，图4-8)可以看出，本文提出的CUTA算法的准确率略高于CUT算法，当近邻值取5左右时，CUTATime的准确度略高于CUT，但是当近邻值大于5时，CUT算法的准确率依然可观。而对于新项目的推荐情况如图4-9 ，可知CUT算法所推荐的项目的新颖度为0，不能解决冷启动问题，综合考虑，CUTATime算法的新颖度最高，因此在处理新项目冷启动问题时其效果最佳。

## 4.5 本章小结

本章主要介绍了基于Item的协同过滤算法，以及性能优势。引入了基于矩阵分解的协同过滤异步分布随机梯度算法，介绍了相关的基于相似的正则化矩阵矩阵分解，以及异步分布式随机梯度优化，以及其意义，同时描述了关于推荐系统的冷启动问题，并提出相关算法，并通过实验验证该算法对于冷启动问题的改善程度。

# 第6章 实验与测试

## 6.1 基于Item的协同过滤算法测试

### 6.1.1 实验环境

实验采用Linux（Centos6.4）操作系统，2.9 GHz Intel Core i5，需要安装的软件有：JDK1.7、Hadoop2.2.0、Spark1.2.0、Scala2.10.4等。编译工具为IntelliJ IDEA，采用Scala语言编写算法和相关代码。

### 6.1.2 数据集

本文使用的是MovieLens数据集。 MovieLens是一个基于网络的研究推荐系统，在1997年秋季首次亮相。每周成千上万的用户访问MovieLens评分和提供电影建议。该网站如今已有超过83000名用户对4800多种不同的电影发表意见。本文随机选择了足够多的用户以及他们对电影的100000个评分。我们将数据库分为两个部分，一部分作为训练集另一部分作为测试集。为此，引入了一个变量来确定用作训练和测试集的数据百分比；我们称这个变量为x。当x = 0.8时表示80％的数据用作训练集，20％的数据用作测试集。将数据集转换为具有943行（即943个用户）和1682列（即由至少一个用户评级的1682个电影）的用户项目矩阵A。对于我们的实验，我们还考虑了另一个因素，数据集的稀疏程度。对于数据矩阵R这被定义为 。因此，电影数据集的稀疏性程度是 = 0.9369。在整个论文中，我们称这个数据集为ML。

### 6.1.3 评价指标

推荐系统研究使用了几种类型的评估推荐系统质量的措施。它们主要分为两类：

(1) 统计精度指标：通过将预测推荐分数与测试数据集中实际用户对项目评级进行比较来评估系统的准确性。实际评分和预测之间的平均绝对误差（Mean Absolute Error，MAE）是广泛使用的指标。MAE是衡量推荐结果偏离其真实值的度量。对于每个评级 - 预测对 ˂pi，qi˃，这个度量对于它们之间的绝对误差，即平等。本文通过对N个相应的等级-预测对的绝对误差求和，然后计算平均值来计算MAE，公式如下：

(公式5-1)

MAE越低，推荐引擎预测用户评级越准确。此外均方根误差（Root Mean Squared Error ，RMSE）也可用作统计精度度量。

(2) 决策支持精度指标：无论项目是被推荐（好）还是不被推荐（坏）, 这些指标假设预测过程都是二进制的。通过观察，如果用户仅选择考虑4或更高的预测，那么项目所有预测评分低于4的项目均变得无关紧要。最常用的决策支持精度指标是反转率、加权误差和ROC敏感度。

因为MAE指标是最常用和最简单的直接解释，因此我们使用它作为评估指标，以此来评价试验的好坏。

### 6.1.4 实验过程

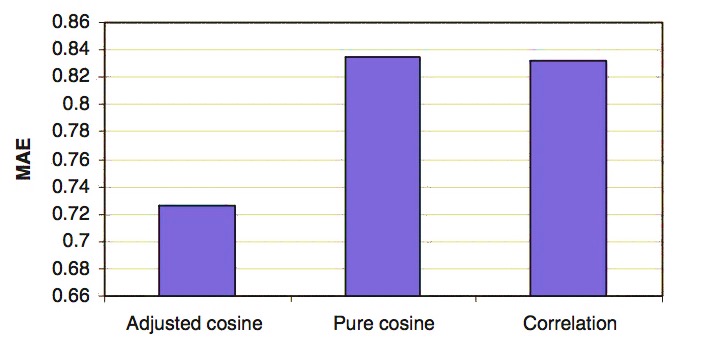
首先我们将实验数据集划分为训练和测试部分。在开始对不同算法进行全面实验评估之前，我们确定了不同参数对不同算法的灵敏度，并从灵敏度图中确定了这些参数的最优值，并将其用于其它实验。为了确定参数灵敏度，我们仅与训练数据一起工作，并进一步细分为训练和测试部分，并对其进行实验。通过每次随机选择不同的训练和测试集合并获取MAE的平均值。我们的实验共进行了10次交叉验证。

### 6.1.5 实验结果

在本节中，通过使用基于项目的协同过滤算法生成推荐的实验结果数据，主要分为两部分：质量结果和性能结果。在评估推荐质量时，我们首先在运行主要实验之前确定了一些参数的敏感度。这些参数包括邻域大小、不同相似度的影响。 为了确定各种参数的灵敏度，我们仅关注训练数据集，并进一步将其分为训练和测试部分。

(1) 相似度算法的影响

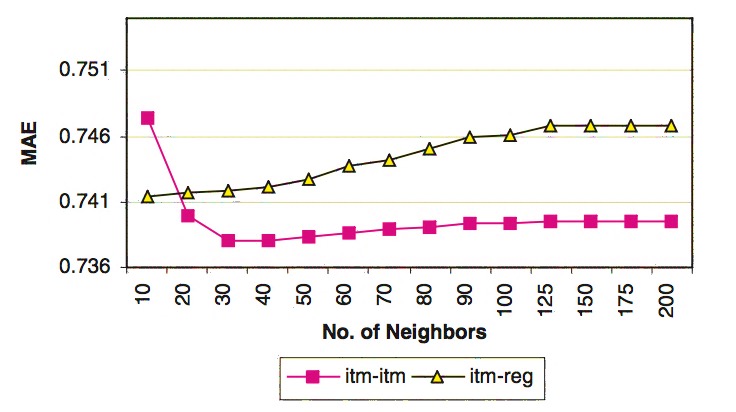
我们实现了三个不同的相似度算法，如4.2.1节所述的余弦相关性算法，调整余弦和相关性，并用指定的数据集进行测试。对于每种相似度的计算方式，我们都通过算法计算其邻域和加权和来生成预测。我们对我们的训练数据进行了推荐预测实验，并使用测试集来验证并计算平均绝对误差（MAE）来评判算法质量。图5-1为实验结果。从结果可以看出，设置调整余弦相似度计算的用户平均绝对误差值具有明显的优势。因此，我们认为使用调整余弦的方式计算相似度最为准确。



**图5-1**相似度计算对项目协同过滤算法的影响

(2) 邻域大小对实验结果的影响

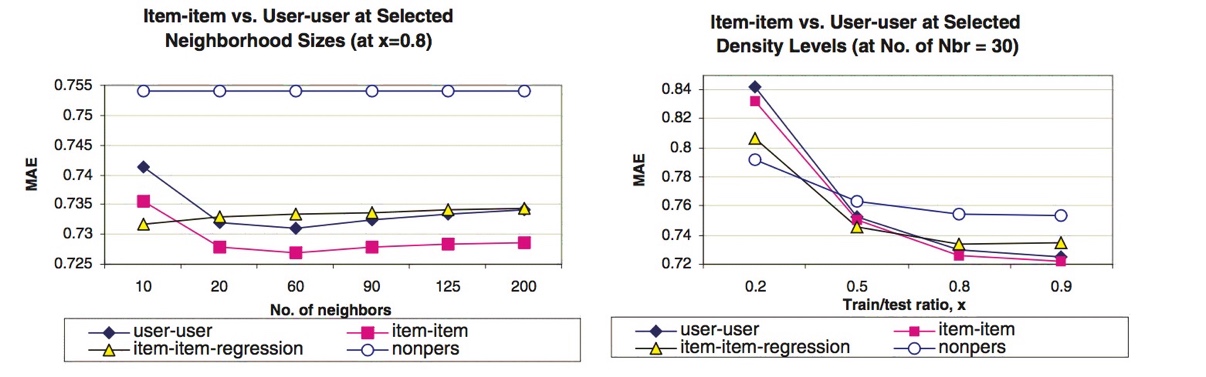
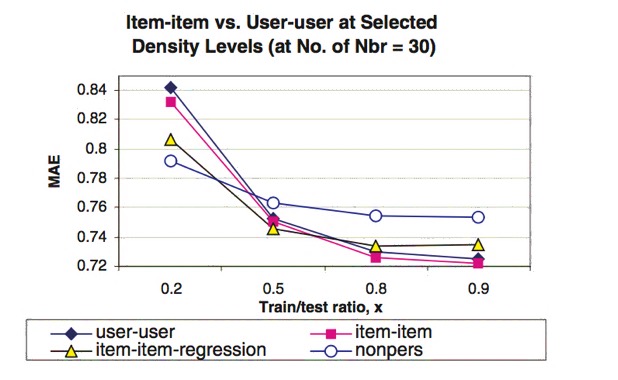
邻域的大小对预测的质量有很大影响。 为了确定这个参数对推荐预测结果的影响，我们做了进一步的实验，我们改变了邻域数并计算MAE的值。实验结果如图5-2所示。我们可以观察到邻域的大小确实影响预测的质量。但是这两种方法显示出了不同的曲线变化。基于Item-Item的算法的MAE随着邻域数的增加而增加，之后增加率逐渐减小，曲线趋于变大。另一方面，基于回归的算法表明，邻域数较少时（邻域数量少于30）随着邻域数量的增加，预测质量有非常明显的改善，之后随着邻域数量增加，推荐质量逐渐下降。考虑到这两个趋势，我们选择30作为邻域大小的最佳选择。



**图5-2** 邻域大小对MAE的影响

(3) 算法对比

一旦我们获得了参数的最优解，我们将基于项目的方法与基于用户算法进行比较。从图5-3中可以看出，在x = 0.5时，基于用户的推荐算法的MAE为0.755，而基于Item的协同过滤算法的MAE为0.749。类似地，邻域数为60时，基于Item和基于User的MAE分别为0.726和0.732。然而，基于回归的算法显示出有意思的结果：当x值和邻域数较小的情况下，执行其他两种协同过滤算法，随着数据集的密度x增加，或者随着我们添加更多的邻域，它们所给出的推荐质量越来越差，甚至与user-based的算法的推荐效果相类似。

**图 5-3** 基于Item和基于User的协同过滤算法对比

因此我们可以从这些结果中得出两个结论。首先，基于Item的算法在所有稀疏级别上提供比基于用户的算法更好的质量。第二，基于回归的算法使用非常稀疏的数据集执行得更好，但随着我们添加更多数据，推荐质量下降。我们认为这是因为回归模型收到高密度的数据时会出现过拟合的现象。

(4) 实验结论

推荐系统用于从其用户行为数据中提取潜在的有价值的数据。这些数据可帮助用户在海量数据中找到他们需要的内容。推荐系统通过让用户找到他们喜欢的项目来满足用户的需求，同时帮助企业增加相应的销售额。推荐系统正在迅速成为电子商务的重要工具。推荐系统受到现有公司数据库中庞大的用户数据量的压力，尤其是受到网络上用户数据量的增加的压力。本章节中提出的协同过滤推荐算法虽然在不考虑负载消耗的前提下能够提供较高质量的推荐，但对于目前实时性要求更加苛刻的推荐场景下，需要新的可以显著提高推荐系统的可扩展性的技术。

## 5.2 基于异步SGO算法测试

我们进行了一系列的实验，以评估相似用户和商品的正则化项对矩阵分解的影响，以及所提出的异步框架如何帮助加速SGO算法的收敛性。

我们对MovieLens数据集（ML-100K，ML-1M和ML-10M）和NetFlix集合的一个子集（参见表54-1）进行了实验。对于ML-100K和ML-1M，我们使用了两组数据集（ra和rb或ua和ub），而对于ML-10M和Netflix，我们使用了单个数据集。

我们使用PySpark 1.5.1实现了分布式框架。我们部署了五台具有不同计算负载的服务器。其中一台机器为其他机器分配评级矩阵。当验证集上两次验证之间的RMSE差值低于我们在试验中设置的预定义阈值ε，将停止所有机器上的运算。本文提出的异步方法收敛时是当最慢的机器运算结束后，而对于同步的SGO来说则是在所有机器同时停止计算后完成。

表5-1：实验数据集特征

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DataSet | |U| | | I | | K | training size | test size | sparsity |
| ML-100K | 939 | 1679 | 20 | 90570 | 9430 | 93.7% |
| ML-1M | 6039 | 3952 | 40 | 939809 | 60439 | 95.8% |
| ML-10M | 71567 | 10681 | 100 | 9301274 | 688380 | 98.7% |
| NF-Subset | 28978 | 1821 | 40 | 3235492 | 101078 | 93.7% |

其中, | U | 和| I | 分别表示用户和项目的数量。

以下，数据集ML-10M的潜在因素数量K固定为100，而对于其他较小的数据集，实验选择最佳结果/速度为40,40和20。学习率和正则化参数λ 固定为0.005和0.05。

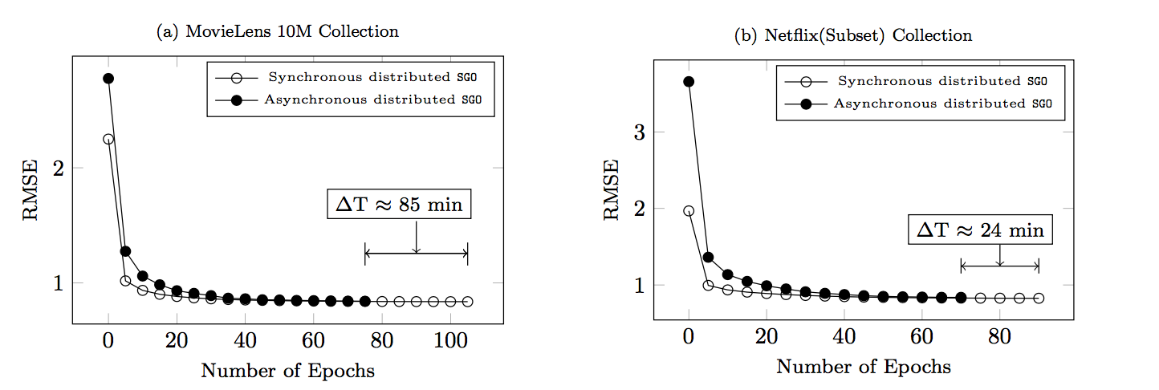
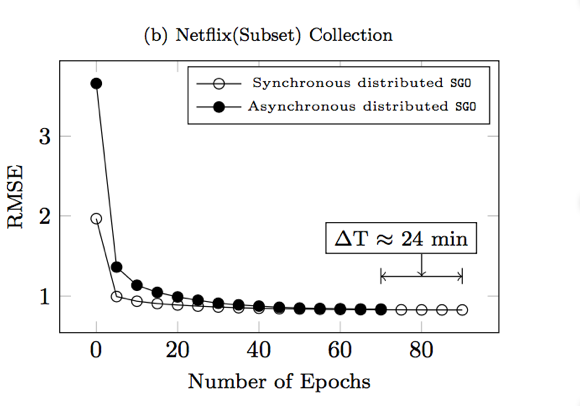
对于之前提出的基于相似性的正则化参数λu，λi，和相似的用户/物品的数量N，其中 λu, λi 从集合{10−1, 5.10−2 , 10−2 , 5.10−3 , 10−3 , 5.10−4 , 10−4} 中选择，N从集合{10, 20, 30, 40, 50}中选择，恰当的选择λu, λi以及N的值会使我们通过验证集得到较好的RMSE的值。

在我们的实验中，我们使用平均绝对误差(MAE)和均方根误差(RMSE)作为性能测量。此外，我们比较了同步和异步计算中每个算法的收敛时间。

(1) 基于相似的正则化效果

首先，我们比较了基于传统SGO方法和我们提出的基于相似性正则化修正的SGO方法的结果。其区别仅仅是最小化的目标函数。

我们进行了基于分布式的同步SGO和异步SGO方式对不同的数据集(MovieLens/Netflix) 进行基于相似用户和基于项目的正则化矩阵分解，得到的实验结果见图5-4。

**图5-4 不同数据集同步和异步SGO收敛时间对比**

我们测试了两种情况：(1) 只有用户具有相似性规则(即项目的正则化参数 λi = 0)。(2) 当用户和项目都使用相同的正规化参数(λu =λi)。图5-4显示了我们实验的完整结果。

实验结果表明用户和物品的特征向量位于他们最相似的用户的中心。Pearson相似性度量中找到的物品是有效的，因为最终的RMSE和MAE与算法1总是比传统的SGO更好。我们还通过实验验证了相似用户（不是item）的正则化带来的影响(λu> 0，λi= 0)。如表5-1所示，对于某一数据集而言，用户的正则化(λu> 0，λi= 0)比传统SGO的效果更好更稳定，甚至比用户和项目的正规化效果更好。

(2) 收敛时间的评估

现在，我们通过与同步分布式SGO方法进行比较来评估我们之前提出的异步分布式SGO的效果。为了展示其可扩展性，我们对用户-项目相似性正则化问题的收敛速度进行评估。如图5-1所示，对于一定的epochs, 不同的数据集Movielens和Netflix，得出了对于我们提出的算法和同步的SGO方法的RMSE曲线对比图。

就异步方式而言，我们通过实验展示了最慢机器的收敛时间。正如预期的那样，由于最慢的机器会从较快的机器上获取更新数据从而使得整体以较少的阶段趋于收敛。而对于传统的同步计算方式，运行速度较快的机器则不得不等待每个阶段运行较慢的机器的计算结果。

## 5.3 本章小结

本章主要基于MovieLens和Netflix数据集对基于Item的协同过滤算法和异步分布式框架SGO进行测试。对于基于Item的协同过滤算法，经过试验我们找到了影响推荐质量的因素以及相关规律。而异步分布式框架SGO的实验结果展示了对于不同的数据集MovieLens和Netflix使用不同方法时收敛速度的变化。

# 第6章 总结与展望

## 6.1 总结

目前，大数据背景下，互联网海量信息中蕴含了很多有价值的信息，如何挖掘这些信息为人所用变得越来越重要，同时也是大数据平台首要需要解决的问题，其中推荐系统的出现，其本质正是通过挖掘用户的行为信息，更好的了解用户的喜好，从而帮助用户了解自己，并帮助用户从海量数据中找到自己需要的内容。如今推荐系统已经广泛的应用于众多大型互联网公司中，并成为公司战略推广的主要技术手段。通过深度挖掘用户行为信息和偏好信息，为用户提供有效的高质量的推荐结果，可以大大提高互联网公司运营服务能力，进一步提高公司营业额，因此优秀的高质量的推荐系统具有广泛的发展和服务场景。本文以设计实现了基于分布式计算框架spark的推荐系统，介绍了大数据的相关技术以及推荐系统的相关概念和面临的问题。并基于所研究的算法在数据集MovieLens和Netflix下进行了相关指标的测评。

首先我们介绍了推荐系统所解决的问题，即大数据背景下的信息过载问题。然后我们详细介绍了Spark生态系统的相关框架以及所使用到的大数据处理的相关模块。使用Spark来代替MapReduce，使用Scala编程语言，从而实现更高的编程效率以及算法的执行效率。

在设计和实现部分，介绍了系统的整体架构，包括系统各层次的作用以及相关的功能，详细介绍了分布式日志框架的实现等。接着我们介绍了论文的主要研究内容，即主流推荐算法的相关原理及性能。并针对推荐系统最常见的冷启动问题进行深入分析，并提出时间权重的概念，改善了推荐系统的冷启动问题。具体工作如下：

(1) 基于Spark分布式框架设计推荐系统的整体架构，通过spark实现算法的并行计算，针对系统实时性问题，设计能够实现实时计算的流处理架构，将离线计算与实时计算相结合，根据用户在线实时行为和历史行为数据进行实时计算，得出推荐结果。

(2) 协同过滤算法的相关研究。重点研究了基于Item的协同过滤算法，测评了算法的推荐准确性，以及影响该算法准确率的因素。并引入了基于矩阵分解的协同过滤的异步分布随机梯度算法缩短算法收敛时间。

(3) 分析研究推荐系统冷启动问题。引入了用户时间权重的概念，根据用户-项目-标签信息和用户-项目属性信息来挖掘他们之间的关系。从而通过预测评分值来解决新项目的冷启动问题。

## 6.2 展望

本文基于Spark分布式框架通过异步计算等方式加速算法收敛，并对于相关算法进行了一定的优化。不过该系统仍存在很多不足和需要改进的方面。

(1) 整个系统如何在数据快速增长的情况下，有效的调配计算资源，进行有效的推荐。

(2) 系统设计本身没有考虑到安全因素，例如商家恶意操作影响推荐结果以谋取利益的情况。

(3) 针对算法提出相应的优化方案，考虑扩展更多的应用场景，并考虑在应用环境下设计相应的特征向量。

(4) 本文的所有实验都是在离线的环境下完成的，并且实验的数据集均为MovieLens和Netflix提供的数据，希望在有条件的前提下，可以将其部署在线上，并测试其真实的结果。

# 参考文献

1. 朱扬勇，孙婧.推荐系统研究进展[J].计算机科学与探索，2015，9( 5) : 52-61.
2. 赵良辉，熊作贞.电子商务推荐系统综述及发展研究[J].电子商务，2013( 12) : 58-60.
3. 杨凤萍. 基于神经网络集成和用户偏好模型的协同过滤推荐算法研究[D]. 华中师范大学, 2016.
4. 吴颜, 沈洁, 顾天竺,等. 协同过滤推荐系统中数据稀疏问题的解决[J]. 计算机应用研究, 2007, 24(06):94-97.
5. 吴奕, 乐嘉锦. 基于上下文的分布式协同过滤推荐技术[J]. 计算机工程, 2010, 36(12):90-93.
6. R. Burke, M. O'Mahony, and N. Hurley. Robust collaborative recommendation. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, Recommender Systems Handbook, chapter 25, pages 805-835.
7. 刘海, 卢慧, 阮金花,等. 基于"用户画像"挖掘的精准营销细分模型研究[J]. 丝绸, 2015, 52(12):37-42.
8. Li G, Li. A new algorithm of cold-start in a collaboration filtering system. Journal of Shandong University (engineering science),2012,2(24):12-17 (in Chinese with English abstract).
9. 梁彦.基于分布式平台Spark和YARN的数据挖掘算法的并行化研究[D].中山大学,2014.
10. 孙冬婷, 何涛, 张福海. 推荐系统中的冷启动问题研究综述[J]. 计算机与现代化, 2012(05):59-63.
11. Kraska T, Talwalkar A, Duchi J, et al. MLbase: A Distributed Machine-learning System[J]. Cidr, 2013.
12. 冯琳.集群计算引擎Spark中内存优化研究与实现[D].清华大学，2013.
13. Huang C Q, Yang S Q, Tang J C, et al. RDDShare: Reusing Results of Spark RDD[C]// IEEE International Conference on Data Science in Cyberspace. IEEE, 2016:370-375.
14. 李崇欣. 分布式数据库Hbase: the definitive guide[M]. Sebastopol: O’Reilly Media, Inc. 2015.
15. 刘冬鸣. 分布式组件Zookeeper事务一致性探析[J]. 信息通信, 2017(06).
16. Maillo J, Ramírez S, Triguero I, et al. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors Classifier for Big Data[J]. Knowledge-Based Systems, 2017.
17. 郭艳红. 推荐系统的协同过滤其法与应用研究[D]. 大连:大连理工大学.2014.
18. 陈清浩. 基于SVD的协同过滤推荐算法研究[D].西南交通大学.2015.
19. Kim J K, Cho Y H. Using Web Usage Mining and SVD to Improve E-commerce Recommendation Quality[J]. 2003, 2891:86-97.
20. 王媛媛, 李翔. 基于人口统计学的改进聚类模型协同过滤算法[J]. 计算机科学, 2017, 44(03):63-69.
21. Yoshida T, Kitayama D. A feature terms extraction method based on polarity analysis of customer reviews for content-based recommendation[C]// The, International Conference. 2016:7-11.
22. Cao J, Wu Z Wang Y, et al. Hybrid collaborative filtering algorithm for bidirectional Web service recommendation[J]. Knowledge and Information Systems,2013,36(3):607-627.
23. YIN H Z，CUI B，SUN Y Z，et al.Lcars: a spatial item recommender system[J].Journal of the ACM Transactions on Information Systems，2014，32 ( 3) : 1-11，37.
24. 张亮，赵娜.改进的协同过滤推荐算法[J].计算机系统应用，2016，25( 7) : 45-52.
25. Polatidis N, Georgiadis C K. A multi-level collaborative filtering method that improves recommendations[M]. Pergamon Press, Inc. 2016.
26. Huang CG, Yin J, Wang J, Liu YB, Wang JH. Uncertain neighbors collaborative filtering recommendation algorithm. Chinese Journal of Computers,2010,8(33):1370-1377(in Chinese with English abstract).
27. Park S T, Chu W.Pairwise preference regression for cold-start recommendation[M].
28. 程飞，贾彩燕.一种基于用户相似性的协同过滤推荐算法[J].计算机工程与科学，2013，35( 5) : 161-165.
29. He X, Zhang H, Kan M Y, et al. Fast Matrix Factorization for Online Recommendation with Implicit Feedback[J]. 2017:549-558.
30. W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In Advances in Knowledge Discovery and Data Mining, pages 442–455. Springer, 2015.
31. Jesús Bobadilla,Fernando Ortega,Antonio Hernando,Jesús Bernal.  [A collaborative filtering approach to mitigate the new user cold start problem](http://www.cnki.net/kcms/detail/detail.aspx?filename=SJES13011501719153&dbcode=SSJD&v=Mjc0MTdvOUVZK29HRFhrNm9HY1U2VDU4UUh1VXJtUXpDOGZpVEw3d1plWnVFU25zVTdqSUlWd1JhQnM9TmlmT2ZiSzdIdEROcQ==)[J]. Knowledge-Based Systems . 2011.
32. 丁少衡，姬东鸿，王路路.基于用户属性和评分的协同过滤推荐算法[J].计算机工程与设计，2015(2):487-491.
33. 李改. 一种解决协同过滤系统冷启动问题的新算法[J]. 山东大学学报(工学版),2012.
34. Zheng N, Li QD. A recommender system based on tag and time information for social tagging systems. Expert Systems with Applications, 2011,38(4):457-485.
35. 刘建国，周涛.个性化推荐系统的研究进展[J].自然科学进展,2009,19(1);1-15.

# 致 谢

首先，感谢我的母校中国科学技术大学给了我读研深造的机会，并且给了我良好的教育。在读研期间，学校的学习氛围以及严谨的治学态度给我留下深刻的印象，同时学校尽可能的为学生提供学习资料与良好的环境，增长了学生的见识，我以我的母校而自豪。

其次，要感谢我的班主任朱洪军老师，感谢他在读研期间对我的生活和学习上的关心与帮助，朱洪军老师在教学上的认真和对学生负责的态度给我留下深刻印象，同时也是我学习的榜样。最重要的是要感谢戴礼荣老师作为我的校内导师，感谢戴礼荣老师细心指导我的开题报告及论文的撰写，戴礼荣老师无论多忙多晚都会耐心为我解决疑惑，同时给予技术上的支持以及论文修改的建议。若没有戴礼荣老师的细心指导，论文的完成非常的困难。在这里非常感谢戴礼荣老师的指导，戴礼荣老师对学术要求严谨，同时又平易近人，给我留下深刻印象，衷心祝愿她永远健康快乐，工作顺利。

再次，要感谢平安科技(深圳)有限公司给了我实习的机会，使我了解了企业工作和学校学习的区别，让我提前进入工作氛围。同时感谢企业导师程宁，感谢他在工作中给予我的指导，为我解决了很多工作中的疑惑，同时为我的论文选题以及工作学习提供很多宝贵的建议，感谢她对我的工作和生活的关心。也要感谢同事们在工作上给予的技术指导，使我的技术得到很大提升。

最后，感谢我的家人和朋友，是他们一直在背后默默的支持和鼓励我，在遇到挫折时他们总会为我加油。感谢父母对我的栽培和付出，希望他们永远健康快乐。

2017年8月