单位代码：10293　密　级：

南京邮电大学

专业学位硕士论文

论文题目：基于流计算的推荐系统设计与实现

学　　号　1212043135

姓　　名　陈俊安

导　　师　郑彦

专业学位类别　工程硕士

类　　型　全　日　制

专业（领域）　软件工程

论文提交日期　二零一五年三月

南京邮电大学学位论文原创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

本人学位论文及涉及相关资料若有不实，愿意承担一切相关的法律责任。

研究生签名：陈俊安 日期：2015.3.31

南京邮电大学学位论文使用授权声明

本人授权南京邮电大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档；允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入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、汇编本学位论文。本文电子文档的内容和纸质论文的内容相一致。论文的公布（包括刊登）授权南京邮电大学研究生院办理。

涉密学位论文在解密后适用本授权书。

研究生签名：陈俊安 导师签名： 日期：2015.3.31

Design and Implementation of Recommender System Based on Stream Computing

Thesis Submitted to Nanjing University of Posts and Telecommunications for the Degree of

Master of Engineering

By

Chen Junan

Supervisor: Prof. Zheng Yan

March 2015

摘要

随着信息技术和互联网的发展，人类正处于信息爆炸的时代。推荐系统是解决信息过载问题的有效方法，通过分析用户行为给用户推荐有价值的信息。而提升系统实时响应能力以及海量数据计算性能，满足日益增长的业务需求，不仅是互联网企业主要面临的技术难题，也是科研机构的重点关注方向。

流计算技术是针对流数据的实时计算，它主要应用于产生大量流数据、同时对实时性要求高的领域。针对上述推荐系统存在的问题，本文通过将流计算技术与推荐算法相结合，设计实现了一个基于 Storm 的推荐系统。系统以分布式流计算平台 Storm 为基础，通过 Trident 计算方法、Redis 存储方案对基于用户的协同过滤算法进行实现，提升了算法的计算效率，减少了系统的反应时间；通过分布式 RPC 服务解决了集群中的消息响应。系统不仅支持对海量数据的计算与分析，而且满足针对用户的实时响应需求，同时具备良好的可扩展性。通过分布式流计算平台 Storm 提供的框架实现的实时流计算应用，有效降低了系统开发部署的复杂性。

本文通过实验对系统的处理性能和算法执行准确率进行了测试，证明了基于 Storm 的推荐系统相较于传统方案实现的协同过滤推荐系统，无论在并行计算性能、数据实时反馈还是系统可扩展性方面都更具优势，对相关行业的实际应用具备一定的参考价值。

关键词：流计算，Storm，推荐系统，协同过滤

Abstract

With the development of information technology and Internet, people are in the information explosion era. Recommender system is an effective method to solute information overload, by means of analyze user behavior to recommend valuable information. Improve system's real-time response capacity and massive data computing performance, meets the even growing business needs, these issues are not only the main technical problems faced by many internet enterprises, but also the attentional directions of many scientific research institutions.

Stream computing technology is aimed at the real-time stream data computation, it mainly applied to produce enormous numbers of stream data and the fields where have a high requirement of real-time. Aiming at the above-mentioned problems which exist in the Recommender system, the author designed a recommender system based on Storm which combined the stream computing technology and recommendation algorithms. This system is based on one distributed stream computing platform called "Storm", besides, In order to enhance the efficiency of the algorithm, reduced the reaction time of the system, the system implemented User-based Collaborative Filtering Algorithm by Trident calculation method and Redis storage scheme. Moreover, it implemented message response of the cluster by means of Distributed RPC service. The system not only supported the calculation and analysis of massive data, but also met the real-time response requirements of users, and it had good extensibility as well. It reduced the complexity of system development and deployment effectively by the application of real time storm calculation which provided by the distributed stream computing platform-Storm.

The author confirmed Recommender system based on Storm had many advantages, such as the performance of parallel computing, data real time feedback, system extensibility, over collaborative filtering recommender system based on traditional solution, hence this system have a certain reference value for practical application of the related industries.

Key words: Stream Computing, Storm, Recommender System, Collaborative Filtering

目录

第一章 绪论

1.1 研究背景与意义

当今社会随着互联网的深入发展，越来越多的信息在互联网上传播，各类公司依靠在线电商业务为目标用户提供资讯的步伐加快。然而互联网规模和普及范围的急速扩大，社会逐步从资讯缺乏的环境过渡到信息冗余的环境[1]，大量资讯的交替出现造成人们难以从中发现对自身有意义的内容，资讯的运用效率因此变低。如何从海量的商业数据中挖掘到对自己有价值的内容，已经逐步成为令企业和个人用户头疼的问题。虽然搜索引擎[2]、专业数据索引等技术本质上都是帮助用户筛选所需要的信息，但是它们并无法为每一位用户提供个性化的服务，而且反馈的信息量也比较大，仍无法有效解决信息超载的问题。

推荐系统作为有效的信息过滤技术，是目前专注于解决信息过载难题的一种方式。通过在用户与项目之间进行关联，推荐系统可以在充满海量数据的状态下找到用户可能喜欢的内容，并将其主动推荐给这些用户。对于用户没有明确表示出需求的场景，通过对无效信息进行过滤，把握人们也许会偏好的内容，并给他发送相应的结果，推荐系统就是一种合适的方案。推荐系统并不是新鲜的概念，从上世纪九十年代就激发了人们在该领域进行科研以及商业实践，但是推荐系统真正地被大规模的应用，特别是在众多互联网企业进行实际的应用，是从最近几年才开始的。

随着互联网领域各类企业应用的不断发展，信息爆炸带来的数据量的激增造成系统推荐任务的复杂度不断增加，对推荐系统的各方面需求也在不断提升。其中面向用户的实时交互能力和面对海量数据的计算能力要求也日益加大，单纯通过提高服务器性能来处理海量数据的成本高昂而且不现实。而流计算技术无论在对数据流运算的实时性以及面对巨大数据量的并行计算能力上都具备一定的优势。由此可见，通过分布式流计算技术实现推荐系统，对企业应用的性能提升具有实际意义。

1.2 国内外研究现状

1.2.1 流计算研究现状

实时数据流计算在科研领域已有多年的研究[3]，近年来由于网络数据的不断膨胀和用户

需求的持续增大，互联网公司也加入到广泛研究和应用流计算技术的队伍中。

在国外，广为应用的商业搜索引擎 Google，通过嵌入依据流量的有偿访问模式的广告达到在网页最醒目区域呈现最有关的内容的效果；Facebook 作为主流 SNS 站点，其对实时流计算的应用首要来源于对站内日志的分析，向用户精确的推荐好友以及实时响应好友圈的新闻，对应用体验带来了可观的优化，增加用户黏性；Twitter 通过实时数据流计算系统 Storm 对微博话题的趋势进行预测分析，目前已升级成为 Apache 顶级项目；Yahoo 为了提高网站内广告访问数据的处理效率而设计了 S4（Simple Scalable Streaming System）[4]；而 Spark[5]是由加利福尼亚大学 Berkeley 分校的 AMP 实验室开发的开源通用并行分布式计算框架，它可在内存中运算、多覆盖批处理、即时查找，流计算等多类方案，在解决海量数据实时处理和处理精确度上具备优势。

在国内市场，主要基于国外主流开源流计算框架进行研究应用，阿里巴巴淘宝自主研发了 iStream[6]流式计算引擎，iStream 基于 YARN 设计，在设计理念上考虑了与其他计算模型的共存，在达到实时计算效果的同时，实现了计算平台的全局最优化。在阿里巴巴的大数据处理集群内，两种计算框架得到了有效地配合：iStream 主要负责流计算工作，为其相关业务提供实时变化的数据；而 MapReduce 负责对数据进行批处理，为业务提供全量数据。百度的大数据团队自 2011 年起对 Spark 开展研究工作，并在三年后将其加入百度分布式处理系统中。为推动流计算的发展，百度率先向研发人员推出支持 Spark 的大数据处理产品[7]。

由此可见，流计算技术的发展处于起步阶段，市场及用户需求也在逐步扩大，无论在科研领域还是企业应用都具备成熟的发展前景。

1.2.2 推荐系统研究现状

学术界有关于推荐算法的报告最早出现在上世纪九十年代，即利用网络上数百万人的意见帮助我们发现更有用和有趣的内容，帕罗奥多研究中心实现的 Tapestry 系统[8]优先引入了协同过滤的思想概念，而 GroupLens 系统[9]也是为了过滤文本文档而开发。随后推荐系统在多个领域长期受到研究人员的重点关注，并最终发展成为了独立的学科。如今，几乎所有电商类型的网站都在其业务应用内运用着推荐算法。

著名电子商务平台亚马逊（Amazon）致力于推荐系统研究十余年，其提出的 Item-to-Item 协同过滤[10]实现了在海量数据集下高质量的实时推荐功能；当前世界上最大的在线影片租赁提供商 Netflix 十分关注个性化推荐技术，并且通过在线上开展 Netflix Prize 竞赛[11]，有效提高了推荐系统在业界和学术界的影响力；美国最大的视频网站 YouTube 对于个性化推荐领域

也进行了深入研究[12]；而 Facebook 与推特主要通过个性化推荐技术给用户推荐好友，并根据个人社交状态对用户开展定制化的物品推荐；而社会化阅读工具 Google Reader、FlipBoard 等则根据用户历史行为对不同读者的阅读偏好来找出相似度，从而给读者推荐与其兴趣类似的新闻内容。

反观国内，推荐系统的相关研究工作稍有滞后，与国际研究水平存在一定的差距，但是推荐系统的浪潮仍然对国内的众多互联网公司产生了较大影响。同样作为广泛应用的搜索引擎，百度在它的多个业务中加入了个性化推荐服务[13]；国内最大电子商务公司阿里巴巴旗下的电商网站（淘宝网、天猫等）加入了数十项个性化推荐的应用场景，通过推荐交互和反馈优化等技术手段，为用户实时提供高质量的各类推荐信息[14]。个性化音乐网络电台——豆瓣电台[15]，通过分析用户一定时间的反馈，形成用户的兴趣模型，从而使电台的个性播放内容更加地契合用户的听歌爱好。

总的来说，国内现有的推荐系统在并发数据处理量以及推荐精准度等方面都有很大的发展空间，推荐系统所涉及的协同过滤算法优化、相关性计算问题、系统冷启动问题等多个方面，都是值得继续做深入的理论研究，并整合到实际应用中的。

1.3 论文主要研究内容

本课题研究的主要内容是，基于流计算技术设计实现一个分布式流计算环境下的推荐系统。系统主要选用基于用户的协同过滤推荐算法作为个性化推荐的计算方法，选择分布式流计算框架 Storm 作为系统实现的平台，利用 Trident 计算框架完成推荐算法的计算流程，从而达到推荐系统处理海量数据的性能以及实时响应的需求。本论文研究工作的主要流程如下：

（1）通过搜集阅读书籍、期刊论文、技术文档，了解流计算技术和推荐系统相关理论知识、研究成果和实际应用状况；

（2）通过分析互联网领域推荐系统应用案例，了解推荐系统整体需求以及发展趋势，确定本文的推荐系统应用场景；

（3）依照上一步需求分析结果，进行系统总体架构设计及相关模块的主要功能设计，保证各个模块工作的可行性，明确实现过程中的重点与难点；

（4）按照系统总体架构设计对功能模块进行实现，其中推荐算法在 Trident 计算框架上的实现是系统重点部分；

（5）通过对比实验，评测推荐系统的工作性能与推荐精准度，并根据实验结果总结系统的优势与可改进之处，明确可继续深入研究的方向。

1.4 论文组织结构

根据上述本项目的工作内容和实现目标，本文共分为六章进行阐述，每章的主要内容安排如下：

第一章首先讲述本课题的研究背景与选题意义，分别简述流计算技术和推荐系统在国内外研究领域的发展现状及其研究成果。之后介绍文章的主要研究内容以及论文的组织结构。

第二章对本文研究中主要涉及的流计算技术、Storm 平台和推荐系统分别进行引述，包括流计算的概念、处理流程以及使用场景；Storm 平台的基本原理、关键特性、以及与时下热门的 Hadoop 框架的区别对比；推荐系统的基本工作原理、现存的主流推荐算法和评价指标。

第三章首先描述协同过滤方法的基本原理，应用场景及优势比较。之后重点介绍基于用户的协同过滤推荐算法在 Trident 计算框架上的具体实现思路，包括用户相似度计算、相似邻居计算和物品推荐过程。

第四章主要为基于 Storm 的推荐系统设计与实现，首先分析推荐引擎的系统需求，明确本文中推荐系统的设计目标、系统架构和工作流程；之后分别对系统中的各个子模块工作流程进行详细介绍，其中重点展示了流计算模块的实现流程。

第五章为系统测试和结果分析，首先介绍实验环境和测试数据集，之后分别从系统启动性能、数据吞吐量和推荐算法准确性三个方面开展测试工作和实验结果比对。

第六章总结与展望，总结了系统研究与实现成果，分析了系统的不足之处以及可进行改进的地方，从而明确下一步继续研究的方向。

第二章 相关支撑技术

2.1 流计算技术

2.1.1 流计算概念

在过往的数据处理流程中，通常先对数据进行搜集，之后存放在数据库中，当需要使用的时候通过从数据库中对数据调用查询以选取想要的资料。这个过程实际包含了两个前提：当对数据库进行查询的时候，库中的数据实际上是过去某一时刻数据的一个映射，在某些需求下，数据已经过期了；这样的流程中，通常人们主动地进行数据查询，可以概括为一个人类主动而数据被动的过程。然而在某些环境下，这两个前提都不具备，比如股票交易中，数据是在持续产生的，用户需要通过当前的数据实时地做出判断，并且因为数据量的巨大，人们需要设定某种阈值，当数据达到阈值时系统可以自主通知并且自动的进行相应操作。在这种情形下，前提产生了变化：对数据流能够及时做出响应；过程转化为人类被动而数据主动。正因为有了这样的需求，逐步发展出了针对数据流的处理技术。

下图（图 2.1）是一个直观的静态数据计算与流数据计算对比图：

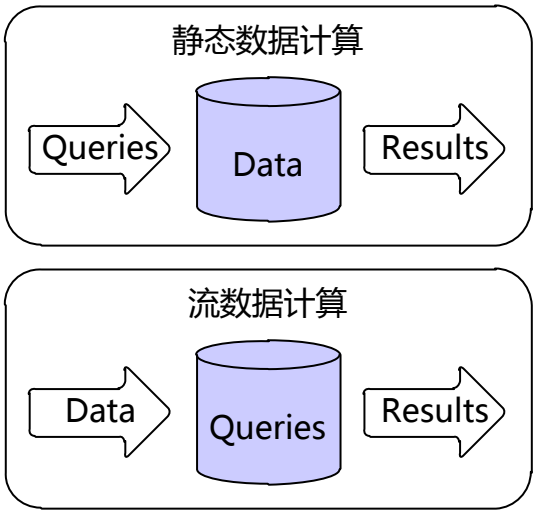

图 2.1 静态数据计算与流数据计算

顾名思义，流计算就是对于流式数据的实时性计算，其中的流数据意思是将数据以数据流的方式进行操作。数据流是在时间和数量上持续分布的连续动态数据集合，数据元是构成数据流的最小单元。流数据具备数据实时到达、过程连续持久、数据源丰富、格式繁杂并且数据规模大等特点，不关注存储、注重数据的整体价值而不关注个体数据也是其重要特性[16]。综上所述，进行流计算的关键在于：数据的价值随着时间的流逝而降低，因此消息出现后需

要尽早进行处理，而非存储以后再进行成批处理。

2.1.2 流计算处理流程

网络上大批量数据的实时计算流程通常分为如下几个阶段进行：数据的产生与收集、传递和计算分析、存储及供外界进行调用。以下将分别对这三个阶段进行引述：

（1）数据实时采集

数据实时采集阶段主要为流计算发送实时数据，在反馈时间上确保了及时有效，并且具备设置简单、方便部署的特点。目前，互联网行业应用的开源分布式日志收集系统主要有：Facebook 的 Scribe、LinkedIn 的 Kafka、Cloudera 的 Flume 以及 Hadoop 的 Chukwa 等[17]，它们都能够达到每秒几百 MB 的数据收集与传输需要。下图（图 2.2）为数据实时收集的基本架构：

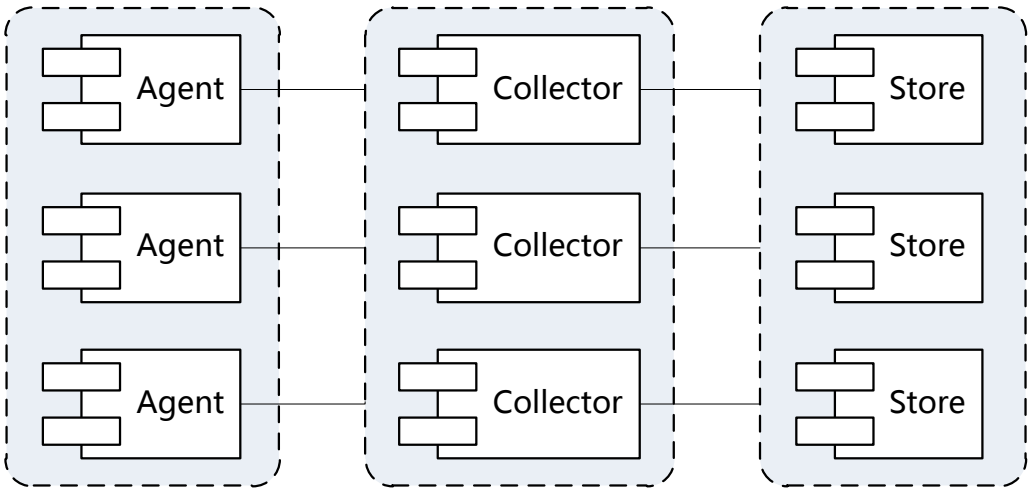

图 2.2 数据实时采集系统基本架构

数据收集的基本架构主要由如下几个模块构成：

（a）Agent：负责收集数据，之后将数据发送给接收器 collector；

（b）Collector：将接收到的数据进行顺序高效的转发；

（c）Store：用于存储 collector 转发的消息，并进行直接计算。

（2）数据实时计算

对于海量实时性数据，流计算的分析过程是在流数据持续变化的过程中实时运算的，当系统发现对用户有意义的信息时，就将计算结果发送给用户。在这一整套分析过程中，系统的计算是主动的，相反，用户一直处在被动接收的状态。下图（图 2.3）为数据流实时计算示意图：

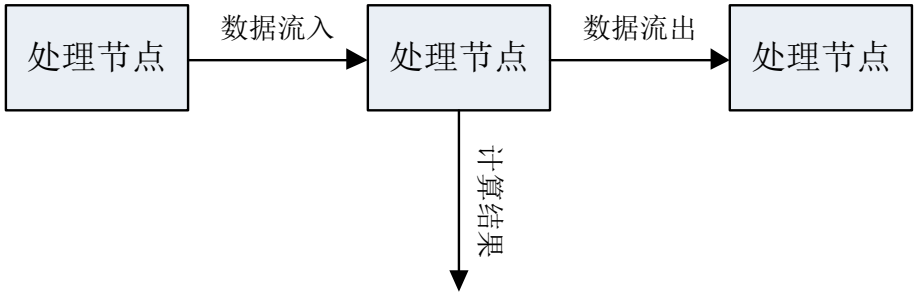

图 2.3 数据流实时计算示意图

（3）实时查询服务

通过流计算系统分析出的结果可进行实时查询、展示或存储，以下是不同数据存储环境下的数据处理方案：

（a）全内存：直接进行数据读取工作，周期性跳转到存储模块进行数据持久化操作；

（b）半内存：通过内存数据库开展即时查询业务，计算结果通过这些系统开展持久化处理。

（c）全磁盘：使用以分布式文件系统（HDFS）为基础的数据库。

2.1.3 流计算应用场景

对于经常产生流数据的场景，以及有实时性需求的领域，流计算适合进行广泛的应用。比如证券、银行等经常生成海量数据流的行业，或者各类实时网络应用。例如站点流量统计分析、系统日志文件处理、实时广告的精准投放以及资讯类应用的个性化推荐。对于一个流计算系统来说，应当包含以下特点：

（1）高性能：高性能是进行海量数据计算的基本前提，例如在秒级对大批量数据的处理；

（2）海量式：支持 TB 级甚至是 PB 级的数据规模；

（3）实时性：系统需要在低延迟状态下进行消息反馈，达到秒级，甚至是毫秒级别；

（4）分布式：支持大数据的基本架构，可扩展性强；

（5）易用性：能够快速进行开发和部署；

（6）可靠性：能可靠地处理流数据。

在不同的使用环境下，对于流处理的性能会有各自的需求。但对于海量数据的实时计算，在数据采集和数据处理阶段都需要达到秒级响应。

2.2 Storm 流计算框架

Storm 是一个开源的分布式实时流计算框架[18]，能够简单、可靠地处理海量的数据流。通过把任务划分后分配给各个组件，让他们各自处理一项。输入到 Storm 集群的数据流通过一个名为 Spout 的组件进行管理，而 Bolt 专门负责处理消息。从 Spout 接收到数据后，Bolt 将数据进行存储，或者继续传递给其它的 Bolt。总体来说，一个 Storm 集群的运作方式就是将 Spout 输入的数据在众多的 Bolt 之间进行处理。下图（图 2.4）是对 Storm 计算拓扑结构的简单展示：

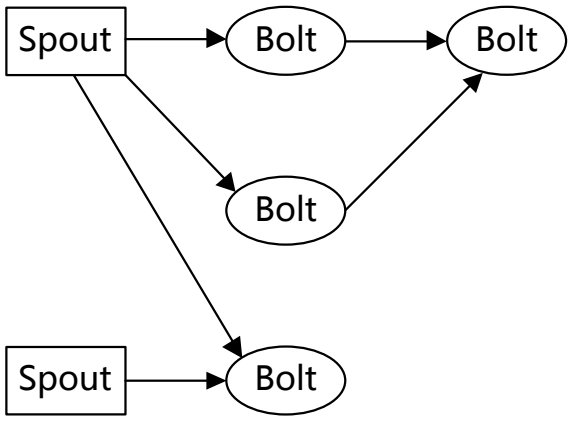

图 2.4 Storm 计算拓扑结构图

Storm 有着丰富的应用环境，而且 Storm 集群的部署和运营维护十分方便，并且更加关键的是可以通过各类语言进行应用研发。

2.2.1 基本原理

在 Storm 的集群里面有两种节点：控制节点（master node）和工作节点（worker node）。为了分配任务给工作节点，并监视运行情况，控制节点中运转着一种后台程序：Nimbus，Nimbus 的作用与 Hadoop 当中的 JobTracker 相似，而每个工作节点中都运转着一个监视器进行监听。监视器会监听分配给自己的任务，根据实际需求启动或关闭工作进程。每一个工作进程运行一个计算拓扑的子集，一个运行的计算拓扑通过运行在集群中的许多工作进程构成。下图（图 2.5）是一个 Storm 集群的工作流程图：

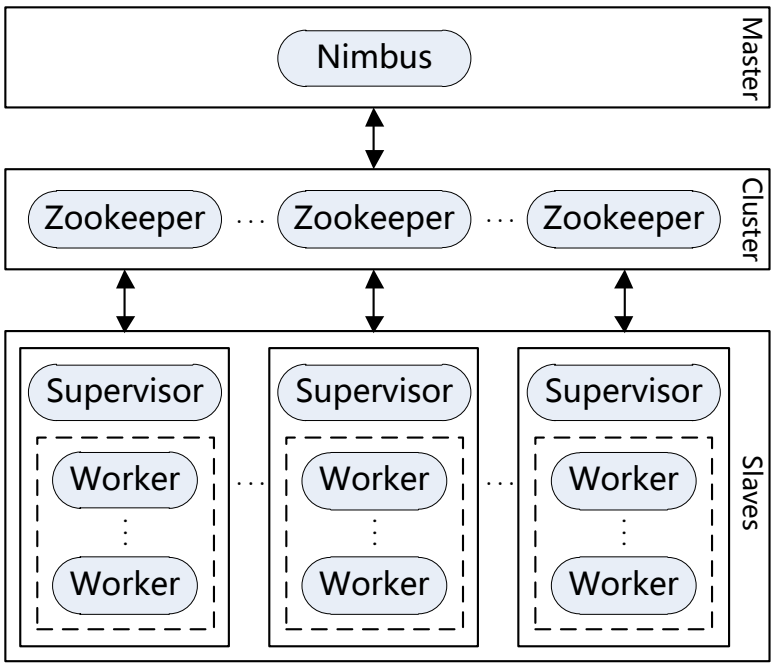

图 2.5 Storm 集群工作流程图

Storm 集群的稳定性很大程度上依靠 Zookeeper 集群来负责。Zookeeper 集群处在 Nimbus 和监视器中间，处理两者之间的协调工作，而 nimbus 和监视器都是迅速执行并且不保留状态的，这就需要 Zookeeper 进行 nimbus 和监视器状态的保存。

以下是对 Storm 中的关键基本原理的引述[19]，相关的基本概念是理解 Storm 的架构和工作原理的前提：

（1）计算拓补：Topologies

计算拓扑是由 Spouts 和 Bolts 组成的图状结构，计算拓扑里的每个处理节点都封装了计算程序的处理逻辑，而链接 Spouts 和 Bolts 的则是流分组策略（Stream groupings）。

（2）消息流：Streams

消息流是 Storm 框架中最为核心的抽象。消息流为无界限的元组队列，Storm 负责执行消息流的传递工作。Storm 提供的最基本的处理消息流方案是 Spout 和 Bolt。

（3）消息源：Spouts

消息源 Spout 是计算拓扑中的消息提供者。通过从外界接收数据，之后封装成元组形式，Spout 会向计算拓扑发送消息元组。Spout 分为可靠与不可靠两种类型，如果消息元组没有被 Storm 成功处理，可靠的 Spout 会再重新发送一个；但不可靠的 Spout 则不会重发。Spout 是自动执行类的组件，通过内置的函数，Storm 可以持续地调用。

（4）消息处理者：Bolts

Bolt 负责计算输入的消息流，并生成消息流传递给下一个 Bolt。Bolt 可以执行过滤、聚合、操作数据库等多种操作。Bolt 是被动执行的组件，通过其内部包含一个函数，Storm 在收

到消息之后可以进行调用。根据实际需求，可直接在此方法中操作处理逻辑，所有的消息处理逻辑被封装在 Bolt 里面。

（5）Data Model：数据模型

Storm 使用元组作为数据模型，通常会对元组进行数据格式的设置，而且不同元组的对应数据格式需要相同。在默认情况下，元组的字段类型为：integer，long，short，byte，string，double，float，boolean 和 bytearray。如果实现了对应的序列化器，还可以对类型自定义。

（6）Stream groupings：流分组策略

消息流分组方案解决了计算拓扑发送元组的问题，定义一个计算拓扑的关键是定义 Bolt 如何接收输入流。Stream grouping 可以设置一个 stream 怎样为 Bolts 上面的多个任务（Task）进行分配。从任务的角度观察一个运行的计算拓扑，其结构如图（图 2.6）所示：

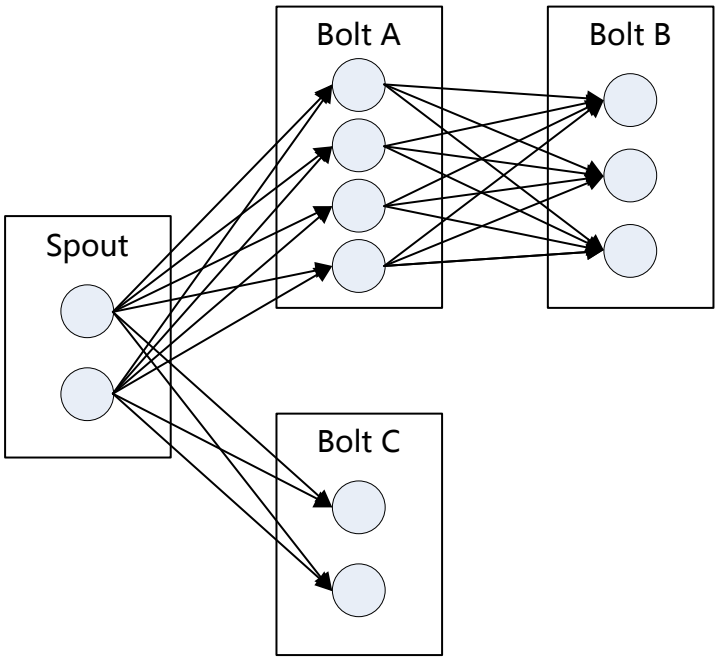

图 2.6 任务处理流程图

Storm 里面有 7 种类型的 Stream groupings：

（a）ShuffleGrouping：随机分配消息流中的元组，但消息处理者接收的元组个数须一致；

（b）FieldsGrouping：按格式分组，例如根据用户 ID 的不同，元组会被发送给不同的消息处理者；

（c）AllGrouping：广播分组，发出的每一个元组，会重复分配到所有的 Bolts；

（d）GlobalGrouping：全局分组，元组将发送到计算拓扑中消息处理者的其中一个任务，详细来说是分配给 ID 最低的任务；

（e）NonGrouping：不进行分组，这种策略的思想是消息流只发送元组，并不负责元组

被哪一个任务处理。目前 NonGrouping 和 Shufflegrouping 分发方式相同，区别是计算拓扑对消息处理者的执行方式；

（f）DirectGrouping：直接分组，直接将元组发送到指定的任务来处理，计算拓扑直接对消息处理者分配具体的任务但要进行事先声明。消息处理者根据计算拓扑具体环境能够获得处理其消息的任务 ID（OutputCollector.emit 方法也会返回任务 ID）；

（g）Localorshufflegrouping：若消息处理这有多项任务在共同执行，则会将所有元组随机分组给进程中的任务。否则，效果和之前介绍的 shufflegrouping 一样。

（7）任务：Tasks

在操作 Spout 和 Bolt 时，它们会以多个任务的形式在集群中执行。每一个任务对应到一个线程，而流分组策略则是定义如何从一部分任务发送元组到其他部分任务。若果要设置并发任务数，可通过调用 TopologyBuilder.setSpout( )和 TopBuilder.setBolt 进行设置。

（8）工作进程

一个计算拓扑的运行是分配在多个工作进程中同步进行的，为了应对负载均衡，计算拓扑会平均地分配工作量。

（9）配置

Storm 当中的配置分为系统级与 Topology 级，通过参数配置可以改变 Nimbus，Supervisor 以及 Topology 的运行方式，所有的默认配置存储在 default.xml 之中。通过设置 storm.xml 可以覆盖默认配置，也可以在程序中配置 Topology 相关的配置信息，这几种配置的优先级为：TOPOLOGY-SPECIFIC，之后是 storm.xml 和 default.xml。

2.2.2 与 Hadoop 区别对比

在数据挖掘领域，Hadoop[20]已经是使用最为广泛的分布式系统架构，Hadoop 具备高可靠性、高扩展性、高效性、高容错性和低成本等优势。但伴随数据量爆发增长，对数据的实时计算需求逐步扩大，并成为众多企业和研究机构面对的难题。Hadoop 专注于海量数据的批处理，受限于系统架构的特性，与实时处理系统在应用需求上有着一定的差别，对流数据的实时计算并不是它的强项。要达到对数据的实时操作，不仅需要将传递来的数据实时地发送处理，而且对数据也要进行流式的划分，而 HDFS[21]在这方面的性能已无法满足实时性的要求。

宏观上分析，Storm 和 Hadoop 十分相似，在 Storm 上运行的是计算拓扑。两者的关键区别是：一个 MapReduce 的任务执行完成后会结束，而计算拓扑会处在一直运行的状态，除非

主动去结束进程。虽然从概念上分析，Storm 继承了 Hadoop 的许多概念、术语以及操作方式，但同样面对大批量数据，两者的计算方法是主要的差异。与 Hadoop 框架不同，Storm 起初就是为了实现流计算与实时处理而进行的框架设计。在 Hadoop 的计算模式中，主要实现一种名为 MapReduce 的计算思想，通过把数据分片从而处理海量离线数据。文件系统将接收到的数据分配至每个工作节点进行运算，并将得到的处理结果发送回文件系统，这适用于大数据的高效批量处理模式；而 Storm 从构建一个计算拓扑开始，持续处理位于实时消息队列的数据流，系统的计算拓扑是持久运行的。

下表（表 2.1）从系统角色、应用名称、组件接口三个方面展示了 Hadoop 与 Storm 之间的对应关系[22]。

表 2.1 Hadoop 与 Storm 对应关系

	Hadoop	Storm	作用
系统角色	JobTracker	Nimbus	任务调度，资源管理
	TaskTracker	Supervisor	启动和停止执行进程，汇报节点状态
	Child	Worker	业务逻辑具体执行的进程
应用名称	Job	Topology	用户自定义任务
组件接口	Mapper/Reducer	Spout/Bolt	编程模型

2.2.3 关键特性

Storm 的一些关键特性如下：

（1）使用场景广泛：Storm 支持众多使用场景，例如实时处理、人工智能、持续流处理、分布式 RPC（远程过程调用协议）以及 ETL（Extraction Transformation Loading）等。

（2）可扩展性：Storm 支持灵活的水平扩展。对于扩展一个实时计算任务，主要方式在于扩大系统的集群规模以提高并发处理性能。Storm 集群上的每个节点能够包含多个工作进程，进程在工作中能够构建多个线程，各个线程能够运行多个任务，这是对系统扩展性的保证。

（3）高可靠性

在 Storm 中，Topology 会完整的处理元组。有些 Bolt 在处理元组后会继续发送其他的元组，这就构成了一个树状图，Storm 会跟踪各个由 Spout 的元组生成的树，直到其计算完成。通过 Topology 内部的消息超时设置，假如 Storm 在规定的时间阈值内没有接收到计算完成的消息通知，Topology 则会重新发送这个元组，并将之前的任务判定为执行失败。

如果需要有效运用 Storm 的可靠性特性，当发送新的元组或者你成功处理了一个元组的时候我们必须要通知 Storm。整个过程由 OutputCollector 进行。依靠其中的 emit 方法来通知一个新元组的产生，通过其中的 ack 方法通知一个元组已处理完成。

（4）高容错性：假如在数据处理的时候发现问题，系统会重新部署有问题的消息处理器。Storm 会让一个计算拓扑持续处于运行状态。另外，如果处理单元中保存了中间状态，那么在重新启动处理单元时，必须通过调用自身的处理中间状态进行恢复。

（5）编程模型简单：Storm 为实时流处理准备了简单高效的接口，有效减少了研发实时流计算应用的难度，有助于简单迅速地完成研发。

（6）运维和部署简单：实际部署时，只需根据实际使用场景配置节点的并发数，而不用考虑具体部署到集群中的某一个节点。通过命令操作，可实现全自动部署以及对 Topology 运行的结束。

2.3 推荐系统

网络作为一个充满海量数据信息的平台，让人们对有效资源的获取过程逐渐变得复杂。搜索引擎是应对信息爆炸时代有效的一种方法，使用搜索引擎的前提是用户对自己所要查找的内容十分明确，例如对关键字的检索以快速获取信息。如果一个用户对自身的需求较为模糊，或是无法用明确的文字进行表述，那么搜索引擎并不能对其进行有效的帮助。

推荐系统是解决上述难题的一个可靠方案，它是一个主动为用户挖掘有效信息的过程，在此场景下，用户对资源的获取从简单的搜索转化为更加契合个人爱好的信息发现，可谓是一种个性化的信息获取方式。

2.3.1 基本原理

推荐系统通过独特的信息过滤手段，将如新闻、图片、电影、书籍等不同的内容推荐给可能感兴趣的用户。总体上看，推荐系统的运作是利用将人的自身偏好与某一种设定的样本相互比对，从而预测用户对某些未接触过项目的偏好程度。参考特征的选取从项目本身的信息中选取，或者基于用户所在的环境。下图（图 2.7）展示了推荐系统的具体工作原理：

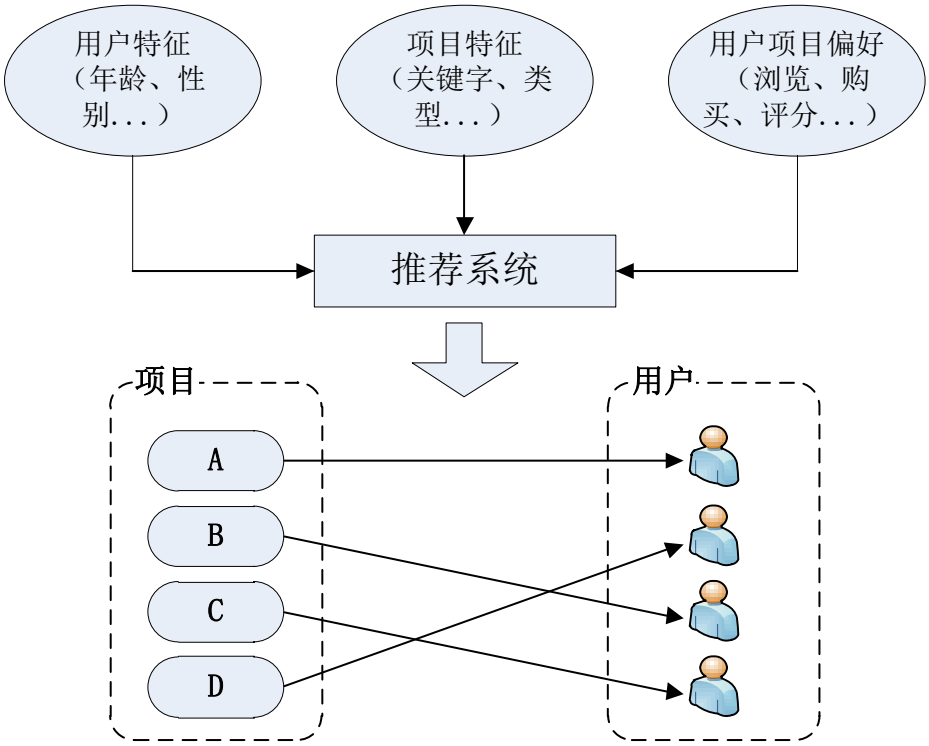

图 2.7 推荐系统工作原理图

2.3.2 主流推荐算法

（1）基于内容的推荐算法（Content-based Recommendation）

基于内容的推荐方法主要思想是依据所推荐项目的关键属性，找到项目之间的关联，之后通过对顾客之前对物品的兴趣情况进行分析，从而把有关联的发送给其他顾客。例如用户经常在 Amazon 购买编程方面的书籍，系统就会在“可能感兴趣的书籍”一栏对你陈列编程开发相关的图书。此种方法也是推荐系统概念提出的初期受到普遍运用的方法。

基于内容的推荐方法优点是它可以有效地构建用户的兴趣模型，以提供更加精准的推荐，但也同时存在以下几个难点：

（a）要事先对项目进行加以建模，算法的推荐性能完全由项目建模的精准度决定。在现今的使用场景中，关键字与标签（Tag）就是对项目关键属性的一种简单高效的描述；

（b）对物品之间的关联的分析只参考了项目本身的属性，并没有加入用户对物品的评价；

（c）由于需要以顾客的过往兴趣为基础进行推荐，因此算法初期的性能表现可能不佳。

（2）基于知识的推荐算法（Knowledge-based Recommendation）

基于知识的推荐方法也可看作是一向推理技术，与之前介绍的方法不同，基于知识的推荐方法通过利用对于某一特定领域构建的一个规则，从而进行基于规则和实例的推理。例如，

基于餐厅在菜品领域的专业性，从而推荐口味适合的餐厅[23]。专业知识是有关某一对象怎样达到另一个指定对象的知识，之后可以表明推荐的相互关系，从而使用到推荐系统。在推荐系统内，专业知识需要支持计算机的可识别。

（3）协同过滤推荐算法（Collaborative Filtering Recommendation）

协同过滤推荐方法[24]是推荐系统中非常经典的技术之一，自上世纪九十年代出现以来，协同过滤的研究成果为推荐系统的发展带来了关键性的突破，许多文章和研究都属于此范畴。随着网络技术的发展，集体智慧[25]的概念日渐趋于主流，人类的主动参与和贡献逐渐地被提倡。集体智慧指的是群体中会产生在单一个体上不会表现出的行为和智慧，集体智慧会逐渐使群体行动对个体产生影响。

基于协同过滤的推荐方法就是在上述环境下产生的，主要思想是，首先依据人对物品的兴趣情况，运算出物品之间的联系以及人与人之间的相似性，之后通过计算出的相似度求出用于推荐的结果。协同过滤方法通常划分成以下三种：基于用户（User-based CF）的协同过滤、基于项目（Item-based CF）的协同过滤、以及基于模型（Model-based CF）的协同过滤。

（4）组合推荐算法

由于推荐方法各有优劣，所以在大型企业的推荐应用中并不会单纯地选用独立的方案。而是将多种推荐方法组合使用，以适应自身的业务需求，从而达到更好的推荐效果。下面是较为常见的组合方法[26]：

（a）加权混合（Weighted Hybridization）：将不同推荐方法计算的结果，用公式通过一定的权值相互组合，再次排列，获取最终的推荐结果。详细的权值可依靠最终计算出的结果做出改变；

（b）切换混合（Switching Hybridization）：各种方案应用在适合的推荐场景，由物品和用户规模、算法执行状态等因素判断。切换混合，即是让算法通过不同的场景选用最合适的推荐方案；

（c）分区混合（Mixed Hybridization）：指通过应用各个类型的推荐方法[27]，然后在各个范围呈现不同计算方法得到的值。在分区混合算法下，能够计算出比较精确的推荐结果，之后依照自身的偏好选取结果；

（d）分层混合（Meta-Level Hybridization）：把各类方案进行分层关联，把上层推荐算法运算出的结果发送给下层推荐算法继续计算，结合多种推荐方法的利弊，算出更加精准的推荐，只是不同分层关联方案也会让最后的计算结果不相同。

2.3.3 评价指标

（1）用户满意度

作为推荐系统的一个主要对象，用户对推荐结果的满意度是衡量算法质量的关键因素。但该指标无法线下运算，仅可依靠对顾客进行调研和试验[28]。在线环境下，需要分析用户行为并通过计算得到结果。比如在购物网站，顾客若是选择了被推销的物品，则表明他接受了系统的推荐，对推荐结果持满意态度。所以可依靠用户对系统推荐商品的选择比例去衡量该项指标。通过在用户交互界面直接加入咨询用户对相关项目满意度的模块，也是对直接有效的收集方法。例如电影网站中对电影的评分，或者音乐推荐网站中的对歌曲的“喜欢”和“不喜欢”。

（2）预测准确度

该项指标是评价一个推荐算法预测用户偏好的成功率的重要指标[29]。预测准确度可以通过离线试验算出，这对于系统的初期运行调试以及科学研究带来了极大的便利。在推荐系统概念产生的同时，预测准确度即是所有企业与科研机构都会关注的指标。计算时要对包含过往数据的数据集进行测试，数据集通常应该含有用户过去的信息。

（3）覆盖率

覆盖率表示推荐系统对项目长尾[30] [31]的发现能力。该指标最简单的理解是算法可以计算出来的物品占物品总量的多少。假设用户集合为 U，算法为用户推荐数量为 N 的项目集合为 R(u)。则覆盖率用以下公式（公式 2.1）进行运算：

$$Coverage = \frac{|\mathrm{U}_{u\in \mathrm{U}}\mathrm{R(u)}|}{|I|} \tag{2.1}$$

不同于其他评测指标，覆盖率是项目供应者更加关注的指标，覆盖率达到 100%，说明推荐系统对每个项目都能够进行一次推荐。新闻、微博类网站的推荐系统更注重对热点项目的推荐，其推荐覆盖率相对较低，而电子商务网站的推荐系统需要有比较高的覆盖率。

（4）多样性

为达到多样性需求，推荐项目列表中应该包括用户的所有偏好方向[32]。用户的个人兴趣偏好通常是不变的，但也分各种方向，当用户访问某一网站时，他所呈现出来的偏好是单一的，如果推荐系统提供的偏好方向与用户当时的关注点有偏差，其推荐结果就难以让用户满意。如果推荐系统给出的结果较为多样化，涵盖了用户的大部分偏好，用户发现偏好项目的几率就会提高。

（5）新颖性

新颖性指的是向顾客推销他之前从未接触过的内容[33]。要衡量新颖性是否充足，最直接方法是参考去看计算结果是否热门，如果算法计算出的项目平均流行度不高，则推荐结果也许是新颖的。

用热门程度衡量新颖性依然有很多难点：例如在视频网站里，给用户推荐新鲜的视频就不应该选择用户浏览过、评价过的视频。但是，用户可能在其他站点接触过待推荐的视频，所以只在自身领域检查的用户接触过的项目仍无法实现可靠的新颖性。

（6）信任度

信任度也是衡量推荐算法的一项关键点，当用户相信推荐系统时，它与推荐系统的交互会变得频繁。在电子商务场景下，企业更加注重顾客对推荐结果的相信程度。在相同的推荐结果下，通过让顾客信任的手段对顾客进行推荐，则顾客接受的概率就越高。单纯通过广告或推销的方法进行推荐则难以在效率上有所保证。评测信任度是较为复杂的过程，只能依靠对用户的直接咨询进行调研。

透明公开和好友推荐[34]是目前提高推荐系统的信任度的常用方案。透明公开即是向用户描述推荐解释：通过让用户明确某一个项目被推荐给自己的原因，以及推荐流程的具体原理。好友推荐指的是依靠顾客的社交圈子，通过已经偏好了某一项目的朋友为基础对目标用户进行推荐，通常好友间的信任度相对较高，因而好友间进行的相互推荐成功率也会较高。

（7）实时性[35]

在许多业务场景中，对项目的推荐有着实时性的需求，例如新闻资讯，热门微博话题等，这就需要项目仍具备时效性的时候就对其进行推荐。

2.4 本章小结

本章分别对流计算技术，Storm 流计算框架以及推荐系统基本原理进行了引述，明确了本文主要研究工作中具体使用到的相关技术。

第三章 基于 Trident 的协同过滤算法

协同过滤推荐算法是推荐系统中最经典的算法之一，算法在科研及商用领域都得到了广泛的研究与应用。协同过滤推荐算法主要分为三类：基于用户的协同过滤算法（User-based Recommendation）、基于项目的协同过滤算法（Item-based Recommendation）以及基于模型的协同过滤算法（Model-based Recommendation）。本文通过对协同过滤算法的比较分析，选取了更加注重实时性的基于用户的协同过滤算法，并且在 Trident 计算框架上进行了具体的实现。

3.1 协同过滤算法主要思想

协同过滤推荐算法的主要思想是，通过对现有用户的历史行为以及对项目的评价情况进行分析，以预测目标用户最有可能偏好的项目，这种类型的推荐模式已经在研究及商务领域有了多年的使用。协同过滤是运用集体智慧的一个经典方案，相对于集体智慧，协同过滤从一定程度上保留了个体特性，即个人的兴趣偏好，因此它更加适应个性化推荐的算法思想。

3.1.1 基于用户的协同过滤推荐

基于用户的协同过滤推荐算法的基本思想十分简单，通过基于用户对项目的偏好发现与其相似的用户，之后将相似用户偏好的项目的推荐给目标用户。在计算流程中，就是将用户对所有项目的偏好作为向量算出用户间的相似度。通过计算固定数量的相似邻居后，根据邻居的相似度值以及他们对项目的偏好，预测目标用户可能偏好的项目，并过滤掉已经偏好的项目，从而列出一个根据推荐度倒序排列的项目列表进行推荐。下图（图 3.1）给出了基于用户的协同过滤工作原理图：

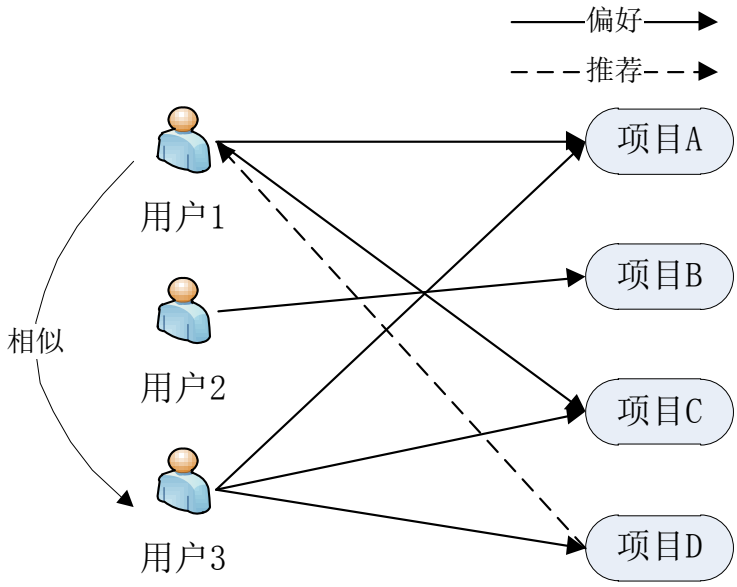

图 3.1 基于用户的协同过滤推荐原理图

如上图所示，用户 1 喜欢物品 A 和物品 C，用户 2 喜欢物品 A、物品 C 与物品 D。由于用户 1、用户 2 都同时喜欢物品 A 与 C，所以用户 1 和用户 2 有相似的喜欢，可以被理解为邻居，最后将用户 2 喜欢的物品 D 推荐给用户 1。

基于用户的协同过滤推荐方法可以很好的满足用户兴趣的多样性，它是依靠兴趣相似的用户来进行相互推荐的，过程中不考虑物品之间是否有关联，因而推荐的物品可能隶属完全不同的类别。缺点是在用户的主要兴趣领域进行推荐的效率不高，且挖掘长尾物品的能力较为欠缺，这种的推荐方法适合用户数量较少而物品数量较多的环境。

3.1.2 基于项目的协同过滤推荐

基于项目的协同过滤推荐方法与基于用户的协同过滤推荐方法大致相同，主要区别在于选取相似用户时站在项目的角度，而非用户本身，也就是通过用户对项目的偏好发现相似的项目，之后根据目标用户的偏好类型，把相似的项目推荐给他。在计算流程中，就是将所有用户对某一项目的偏好作为向量以算出项目间的相似度，得到相似项目之后，依据用户自身的偏好情况预测他还没有偏好的项目，同样列出一个根据推荐度倒序排列的项目列表进行推荐。下图（图 3.2）给出了基于项目的协同过滤工作原理图：

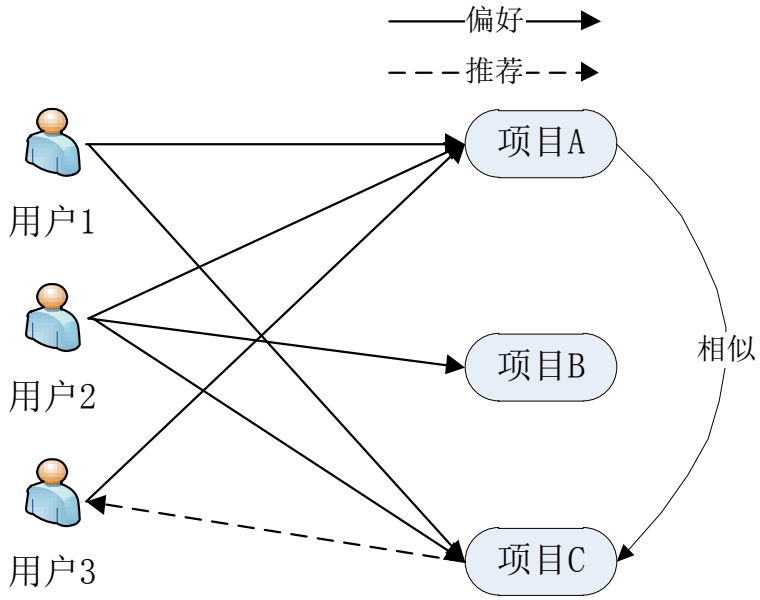

图 3.2 基于项目的协同过滤推荐原理图

如上图所示，用户 1 同时喜欢物品 A 和物品 C，用户 2 同时喜欢物品 A、物品 B 和物品 C，从上述情形可以判断物品 A 和 C 是类似的。由于用户 3 已经喜欢了物品 A，因此把与物品 A 类似的物品 C 推荐给用户 3。

基于用户的协同过滤推荐方法可以很好的满足用户兴趣的多样性，它是依靠兴趣相似的用户来进行相互推荐的，过程中不考虑物品之间是否有关联，因而推荐的物品可能隶属完全不同的类别。缺点是在用户的主要兴趣领域进行推荐的效率不高，且挖掘长尾物品的能力较为欠缺，这种的推荐方法适合用户数量较少而物品数量较多的环境。

基于项目的协同过滤推荐方法是主要偏重对用户的自身兴趣领域物品的推荐，并不能有效发现其他兴趣范围的内容，造成了推荐多样性的缺失，不太适合进行热门资讯的推荐，但挖掘长尾物品的能力较好，这种推荐方法适合用户数量较多而物品数量较少的环境。

3.1.3 基于模型的协同过滤推荐

基于用户的协同过滤推荐方法和基于项目的协同过滤推荐方法都属于以记忆为基础（Memory based）[36]的协同过滤技术范畴，它们都在不同程度上面临着数据稀疏性问题[37]。而以模型为基础的协同过滤(Model-based Collaborative Filtering)是先用历史资料先建立一个预测模型，再用此模型进行预测[38]，由于在推荐系统进行计算之前就已经建立好模型，系统运算性能也得到了进一步提升。虽然基于模型的推荐方法在理论上推荐精度更高，但当面临海量数据时会造成模型的扩展性问题。下图（图 3.3）给出了基于模型的协同过滤工作原理图：

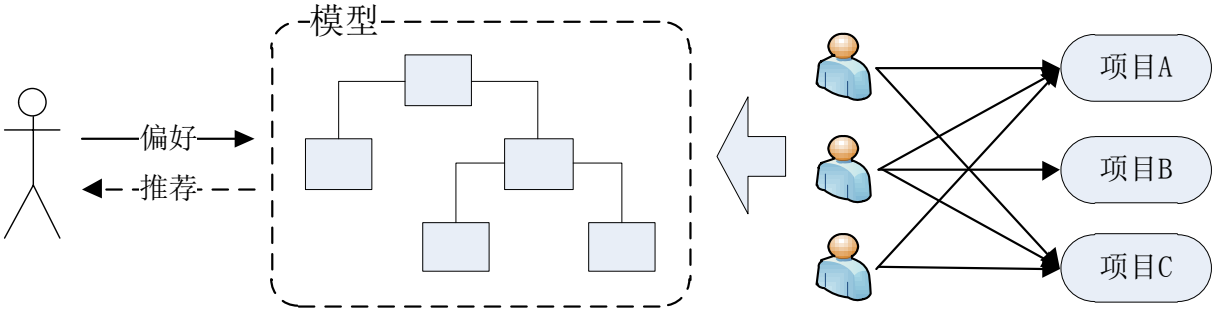

图 3.3 基于模型的协同过滤推荐原理图

3.2 相似度计算方法

确定目标用户的相似用户集合是协同过滤推荐算法中至关重要的环节，判断用户与用户是否相似，需要一套衡量的标准，即用户间的相似度计算。以下是几种常用的相似度计算方法：

（1）余弦相似度（CosineSimilarity）

CosineSimilarity[39]通过向量空间中两个向量夹角的余弦值（Cosine）进行区别对比。比较距离测量，CosineSimilarity 更加侧重向量在方位上的差别。CosineSimilarity 计算方法是把用户偏好设为坐标系中的一个点，并与原点连成一条直线，用户间的相似度的值即是两条直线夹角的余弦值，用户间的相似度与夹角的大小成反比。在三角函数中，余弦值的取值范围为-1 到 1，0° 的余弦值为 1，180° 的余弦值为-1。CosineSimilarity 计算公式如下（公式 3.1）：

$$sim(X,Y)=\cos(\theta)=\frac{X\cdot Y}{\|X\|\|Y\|}=\frac{\sum_{i=1}^{n}x_i y_i}{\sqrt{\sum_{i=1}^{n}x_i^2}\sqrt{\sum_{i=1}^{n}y_i^2}} \quad (3.1)$$

其中 X 与 Y 分别表示两个用户偏好的向量，当 Cosine 等于 1 时，两个用户的兴趣偏好完全一致；当 Cosine 接近 1 时，用户偏好相似；Cosine 越小，用户相似度则越来越低。

（2）皮尔森相关系数

皮尔森相关系数[40]反应了两个变量之间的线性相关程度，系数取值在-1 到 1 之间。当变量的线性关系增强时，相关系数更接近 1 或-1。当两个变量同时增大时，变量之间是正相关的，相关系数大于 0；如果一个变量增大，另一个变量变小，表示它们之间是负相关的，相关系数小于 0 当相关系数等于 0 时，它们之间不存在线性相关关系。皮尔森相关系数的计算公式如下（公式 3.2）：

$$pearson(u,v)=\frac{\sum_{i\in C}^{n}(r_{u,i}-\overline{R_u})(r_{v,i}-\overline{R_v})}{\sqrt{\sum_{i\in C}^{n}(r_{u,i}-\overline{R_u})^2}\sqrt{\sum_{i\in C}^{n}(r_{v,i}-\overline{R_v})^2}} \quad (3.2)$$

其中，$r_{u,i}$ 表示用户 u 对项目 i 的评分，C 表示用户 u 与 v 的共同评分集，$\overline{R_u}$ 和 $\overline{R_v}$ 分别表示用户 u，v 的平均评分值。

（3）欧几里得距离（Euclidean Distance）

Euclidean Distance [41]是十分简单的一种相似度计算方法。假设 X，Y 是 N 维空间的两个点，那么 X 和 Y 的 Euclidean Distance 可以用公式（3.3）表示：

$$d(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \quad (3.3)$$

使用 Euclidean Distance 表示相似度，通过使用公式（3.4）进行转换：

$$sim(X,Y)=\frac{1}{1+d(X,Y)} \quad (3.4)$$

用户和项目的属性值可表示为 N 维欧几里得空间中，某个点在每个维度上的值。由公式（3.4）可知，X 和 Y 的 Euclidean Distance 越小，X 和 Y 所表示的用户或项目的相似度则越大。

（4）基于谷本系数的相似性度量

谷本系数[42]和之前介绍的相关度计算方法有着一定的区别，它不关心用户项目评分，只看两者之间是否存在关联。谷本系数主要用于计算 boolean 测得的相似度。谷本系数也称为杰拉德系数。谷本系数计算方法如下（公式 3.5）：

$$Tanimoto(u,v)=\frac{|I_u\cap I_v|}{|I_u\cup I_v|}=\frac{|I_u\cap I_v|}{|I_u|+|I_v|-|I_u\cap I_v|} \quad (3.5)$$

其中，I_u 和 I_v 表示顾客 u 和 v 的评价集合，$|I_u\cap I_v|$ 表示 u 和 v 共同评价的物品数，$|I_u\cup I_v|$ 表示 u 和 v 评价并集的物品数。计算结果处于[0，1]之间，如果两者偏好的物品完全一致，值为 1；如果没有任何关联，交集为空，值为 0。

（5）Log-likehood 系数

Log-likehood 系数[43]是对数似然函数值，也是一种忽略评分值，只关注二分喜好的相似性度量方法。Log-likehood 用来求一个样本集的相关概率密度函数的参数。其取值范围为[0，1]。越接近 1 表示相似度越高。

3.3 使用场景及优势对比

基于用户的协同过滤（User-based CF）和基于项目的协同过滤（Item-based CF），是协同过滤中两种最典型的算法，而针对不同的使用场景，它们都拥有各自的性能优势。

（1）使用场景

对于新闻资讯类、微博、社交网站等推荐系统当中，其数据资源的主要特点都表现为可推荐内容的数量远远超过网站的用户数量，并且这些内容的更新速度非常频繁，因此 User-based CF 算法是它们主要选择的推荐手段。User-based CF 算法更加关注用户的偏好属性，通过对用户的关注话题类型、用户好友类型、自身兴趣爱好等进行社交关系的建模，以帮助他们发现感兴趣的人物以及资讯，这种方法更容易增加用户对推荐解释的信任程度。

而在另一个领域，在非社交类型的网站如电子商务类网站、视频推荐类网站等，用户的数量则远远超过了网站可推荐内容的数量，并且可推荐内容的数据相对稳定，通常不需要频繁地更新，所以求解内容的相似度工作量相对较小。对于这些类型的场景，内容内在的关联是更为关键的推荐原则，所以 Item-based CF 算法能比 User-based CF 算法给出更加有效的推荐理由。

（2）优势对比

在推荐内容多样性方面，基于用户的推荐方法无论在其可推荐内容的数据量和更新效率上都更具有优势，而因为基于物品的推荐方法更偏向于给用户推荐与他以前看的内容最相似的物品。

而在推荐内容的覆盖率方面，基于物品更加能够给所有用户提供丰富的选择，因为基于用户的推荐方法更倾向于对热门内容的推荐，基于物品的推荐方法则擅长于对长尾物品的挖掘。

除了考虑算法的应用优势，用户对推荐算法的适应度[44]同样是一个关键点。当基于用户时，前提是假设顾客会喜欢和他有相同偏好的顾客所喜欢的物品，当某个顾客周围没有相同偏好的顾客时，该算法的推荐效果会受到很大影响，因此用户对 User-based CF 算法的适应度是与他有相同偏好的用户数量成正比的。而基于物品时，前提是假设用户会喜欢和他以前偏好的内容相似的物品，通过计算一个用户偏好内容的自相似度，如果自相似度较大，则表明他喜欢的物品都十分相似，也说明了用户对 Item-based CF 方法的适应度比较好；相反，如果自相似度偏小，就说明用户的偏好不满足 Item-based CF 方法的前提假设，这样推荐效果自然就不会很理想。

3.4 算法的具体实现

3.4.1 Trident 计算框架

Trident 是在 Storm 框架中进行实时流计算的高级抽象[45]，提供了对实时流的聚集、投影、过滤等操作，从而有效精简了开发 Storm 任务的工作量。其具体工作理念类似于 Pig 或者 Cascading[46]。除此之外，Trident 提供了原语处理针对数据库或其他持久化存储的有状态的、增量的更新操作。

“Stream”（也称 TridentTuple）是 Trident 中的核心数据模型，它被当作一系列的 batch 来处理。在 Storm 集群的节点之间，一个 stream 被划分成很多 partition（分区），而对流的操作是在每个 partition 上并行进行的。

Trident 有五类操作：

（1）Partition-local operations，对每个分区进行局部操作，在这个操作过程中并不会产生网络传输；

（2）Repartitioning operations：对数据流的重新分配操作此操作只是对流的划分，并不改变其内容，在这个操作过程中会产生网络传输；

（3）Aggregation operations：聚合操作；

（4）Operations on grouped streams：作用在分组流上的操作；

（5）Merge、Join 操作。

当我们对 Trident 计算拓扑进行执行时，它会被编译成一个高效的 Storm 计算拓扑。只有在数据需要被重新分配的时候，才会将元组（Tuple）通过网络进行发送，如图（图 3.4）所示，当你有一个如下的 Trident 计算拓扑时：

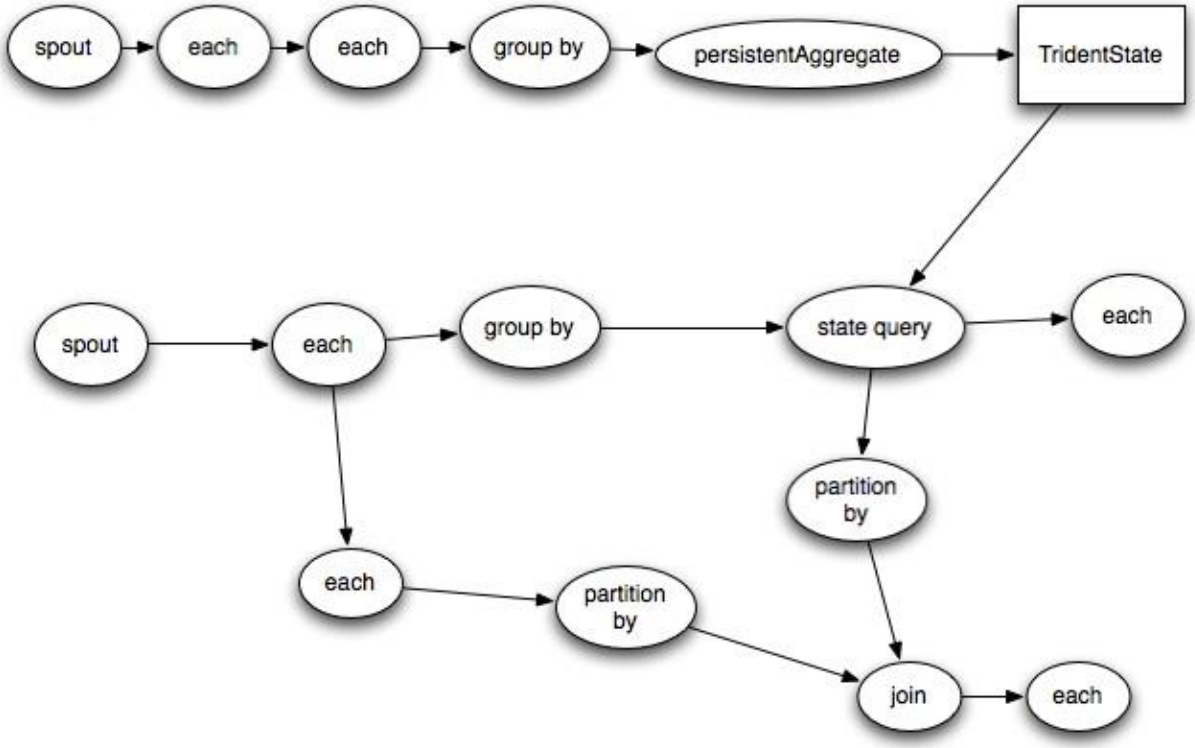

图 3.4 Trident 计算拓扑结构图

在执行时它会被编译成如下图（图 3.5）所示的 Storm 计算拓扑：

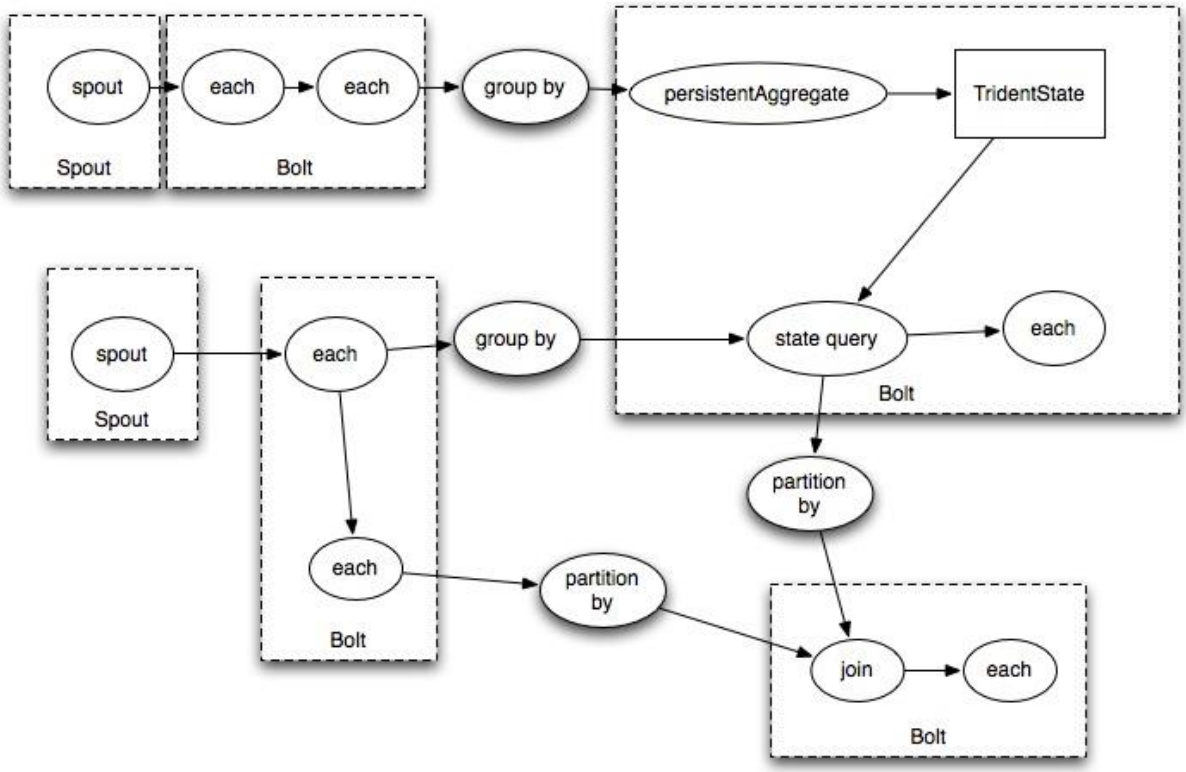

图 3.5 编译后的计算拓扑图

3.4.2 算法整体思路

如图（图 3.6）所示，是基于用户的协同过滤推荐算法的运行流程。下面详细叙述每一个步骤的功能以及实现。

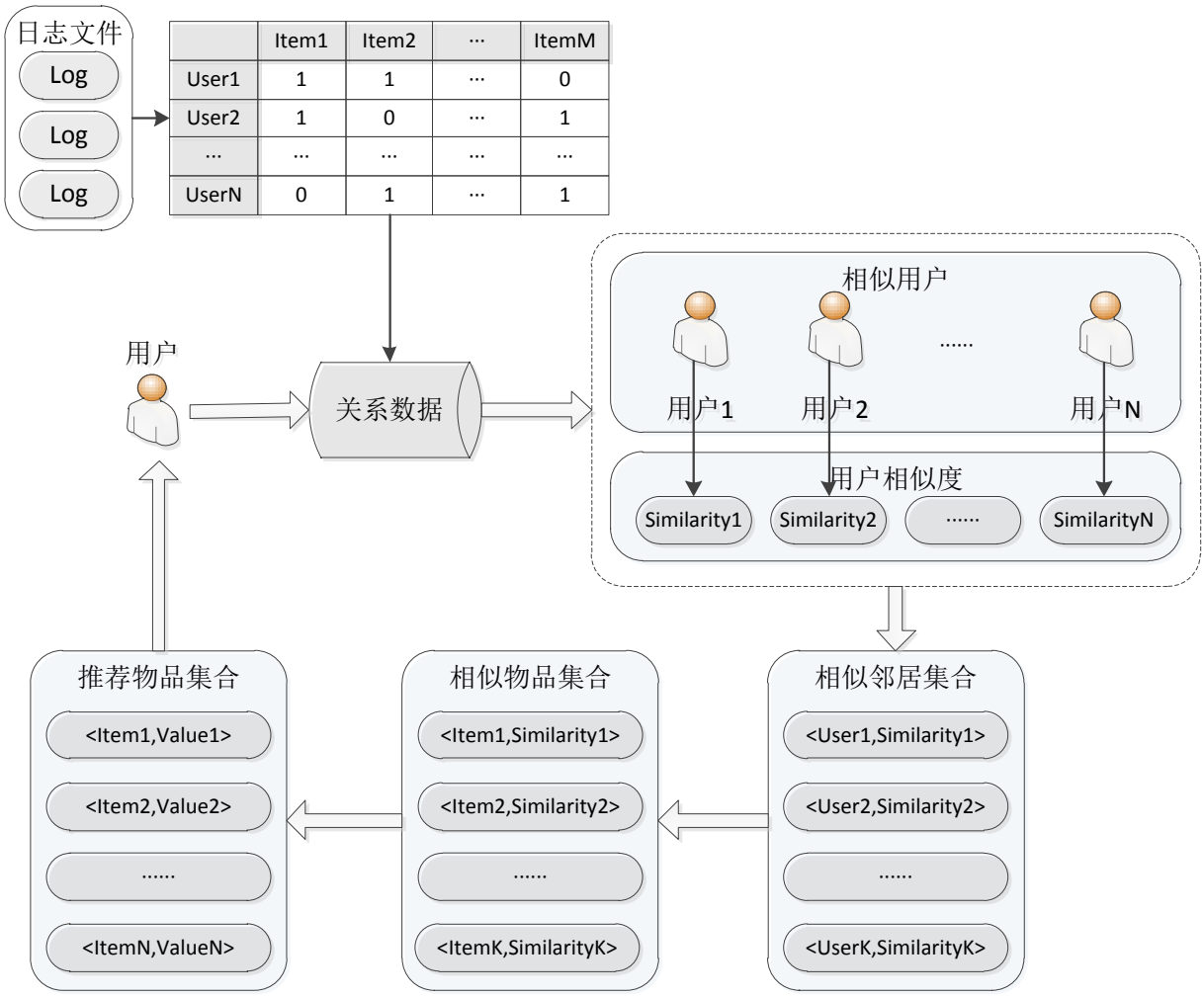

图 3.6 基于用户的协同过滤推荐算法的运行流程

（1）通过分析日志文件等，搜集用户行为信息，将用户与项目信息存储在二维表中；

（2）在关系数据中，根据目标用户 ID 计算出与目标用户相似的用户集合和每个用户与目标用户的相似度（Similarity）；

（3）优化用户的关系数据，通常一个实际的应用系统中用户数据量十分巨大，而且用户相似度的值在推荐计算过程中是十分重要的数据，在相似用户集合中筛选出适合推荐的相似邻居，对后续产生的推荐结果及计算复杂度等都有较为直接的影响；

（4）获取用户偏好项目，通过查找第三步中相似邻居的偏好项目，及其对项目的评价；

（5）在相似邻居偏好的项目中，通过累计其所对应的用户相似度，得到一个项目推荐值（Value），同时过滤到目标用户已偏好的项目，

（6）根据第五步计算得出的所有待推荐项目的推荐值，按推荐值的大小降序排列，生成推荐列表，并根据用户初始化时所设定推荐项目数，生成最终的推荐列表，并将表中项目推荐给目标用户。

3.4.3 算法实现步骤

本节通过一个实例对基于用户的协同过滤算法在Trident计算框架上的实现流程进行具体的说明。

（1）通过搜集处理用户行为信息，可以得出用户的偏好信息，将其整理成用户—项目评分表（表 3.1）：

表 3.1 用户—项目评分表

	Item0	Item1	Item2	Item3	Item4	Item5
User0	1	1	1	1	0	0
User1	0	0	1	1	1	0
User2	0	0	0	0	1	1
User3	0	1	0	1	1	1

上表中，行表示用户，用户编号从 0 到 3；列表是物品，物品编号从 0 到 5；矩阵中的任意项表示用户对物品的评分，在本算法里以布尔值度量的个体间的相似度。

（2）根据用户—项目评分表列出用户偏好状态，将计算结果暂存在 State 中，提供给后续的计算任务进行调用。下表是用户偏好状态表（表 3.2）：

表 3.2 用户偏好状态表

User	Item
0	0,1,2,3
1	2,3,4
2	4,5
3	1,3,4,5

表中直观地列出了每个用户的偏好项目，

（3）与上一步类似，根据用户—项目评分表列出每个项目的被偏好状态，并将这些内容在 State 中进行保存。下表是项目被偏好状态表（表 3.3）：

表 3.3 项目被偏好状态表

User	Item
0	0
1	0,3
2	0,1
3	0,1,3
4	1,2,3
5	2,3

（4）通过谷本系数实现相似度的计算，谷本系数的计算原理已经在之前进行了简要的介绍，通过计算特征集之间的相交比率，进而得出用户或物品的相似度。通过矩阵乘法运算[47]，得出各个用户之间评分项的交集，也就是共同评分的项目数：将用户—项目评分表用矩阵 D 表示，则转置后的项目—用户表以矩阵 D^T 表示。如图（图 3.7）所示：

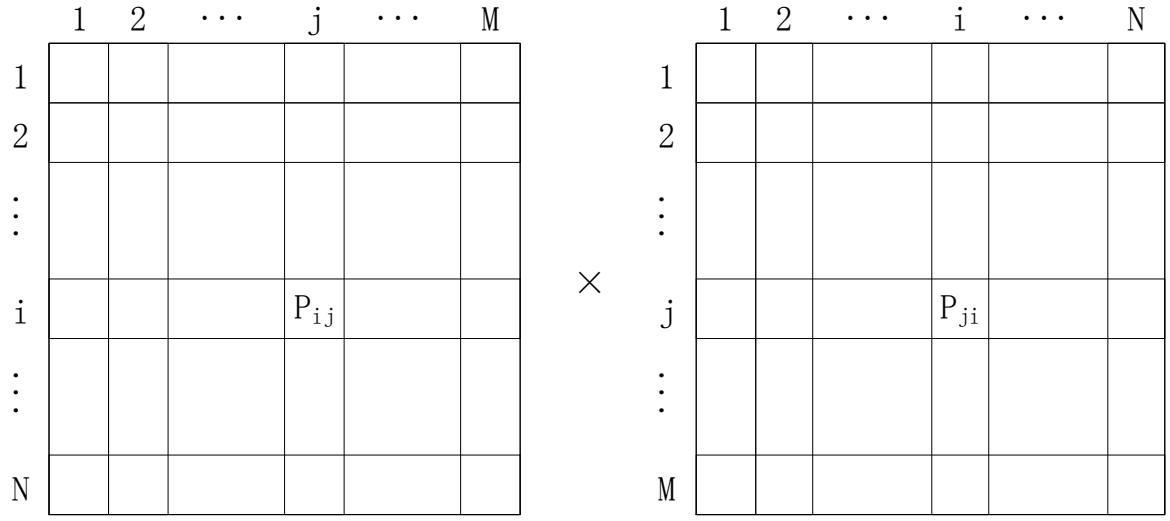

图 3.7　评分项交集计算图

上图两个矩阵相乘之后的结果将得到一个 N*N 的矩阵，矩阵中的各项代表各用户之间评分项的交集。

通过矩阵乘法运算，我们得到了用户—项目评分表中用户之间的评分项交集，具体计算过程如公式（公式 3.6）所示：

$$\begin{bmatrix}1&1&1&1&0&0\\0&0&1&1&1&0\\0&0&0&0&1&1\\0&1&0&1&1&1\end{bmatrix}\times\begin{bmatrix}1&0&0&0\\1&0&0&1\\1&1&0&0\\1&1&0&1\\0&1&1&1\\0&0&1&1\end{bmatrix}=\begin{bmatrix}4&2&0&2\\2&3&1&2\\0&1&2&2\\2&2&2&4\end{bmatrix}\qquad(3.6)$$

经过对不需要数据的过滤，得到了用户评分项交集表（表 3.4）：

表 3.4　用户评分项交集表

〈User1,User2〉	count(∩)
〈0,1〉	2
〈0,2〉	0
〈0,3〉	2
〈1,2〉	1
〈1,3〉	2
〈2,3〉	2

（5）从 State 中取出之前在第二步中被保存的用户偏好状态，根据用户 ID 统计各个用户进行的项目评分数，记为 PreferenceCount。下表是用户评分项统计表（表 3.5）：

表 3.5 用户评分项统计表

User	PreferenceCount
0	4
1	3
2	2
3	4

通过各个用户进行项目评分的集合以及用户共同评分的项目数，我们可以通过使用谷本系数计算出用户间的相似度（公式 3.7）：

$$Tanimoto(User1,User2)=\frac{\left|PreferenceCount1\cap PreferenceCount2\right|}{\left|PreferenceCount1\cup PreferenceCount2\right|} \quad (3.7)$$

下表（表 3.6）即为用户间相似度表：

表 3.6 用户间相似度表

<User1,User2>	PreferenceCount(User1)	PreferenceCount(User2)	count(∩)	Similarity
<0,1>	4	3	2	0.40
<0,2>	4	2	0	0
<0,3>	4	4	2	0.33
<1,2>	3	2	1	0.25
<1,3>	3	4	2	0.40
<2,3>	2	4	2	0.50

将相似度计算结果更新到 State 中，在下节的相似邻居计算过程中，会对相似度计算结果进行调用。

（6）查询之前保存的项目被偏好状态，根据 Item 查询 User2，意义是找出所有评分项所在列的评分用户。

3.4.4 相似邻居计算

得到用户间相似度结果后，可直接根据相似度寻找用户—物品的邻居，这一步称为相似邻居计算，常用的选择邻居的方法主要分为两种：“固定数量的邻居”（图 3.8）以及“基于相似度阈值的邻居”[48]（图 3.9）。

（1）固定数量的邻居（K-neighborhoods）：

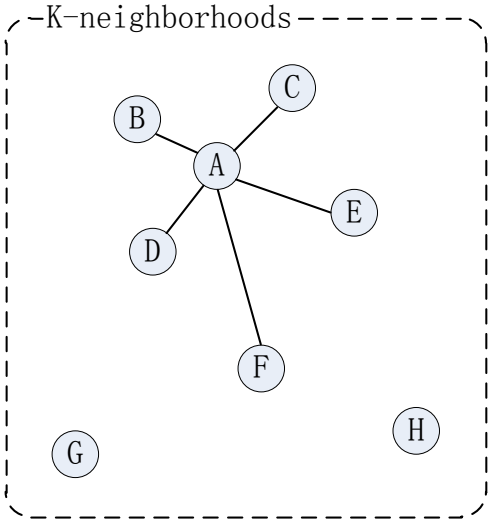

图 3.8 固定数量的邻居

如上图所示，固定个数邻居的思想是：无论邻居的相似度是多少，只选取最接近的 K 个，作为相似邻居。如图所示，比如要计算最接近用户 A 的 5 个邻居，通过点与点之间的距离进行测算，所选取到的最近的 5 个邻居分别是：用户 B、用户 C、用户 D、用户 E 和用户 G。但通过上图可以很直观地看出，由于要选取固定个数的邻居，当目标用户附近没有足够多的相似邻居时，算法会扩大搜索范围选取一些不太相似的点作为邻居，这样在一定程度上影响了邻居的相似度，但在应对推荐系统冷启动方面具有一定的优势，尤其在用户数量较少而物品数量丰富的情况下。

（2）基于相似度阈值的邻居（Threshold-based neighborhoods）：

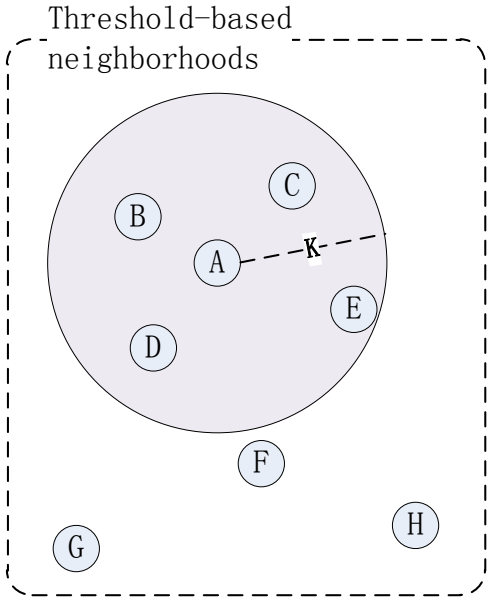

图 3.9 基于相似度阈值的邻居

如上图所示，与计算固定个数邻居的原则不同，基于相似度阈值的邻居的思想是：在选取邻居前事先设置一个最大距离限制，以目标用户为中心，把一定范围内的所有用户都作为相似邻居。如图所示，以目标用户 A 为圆心，K 为半径设定一个邻居选择范围，则相似邻居

分别是：用户 B、用户 C、用户 D 和用户 G。通过这种方法得到的邻居数量不受控制，但每个邻居的相似度不会出现较大的误差。

通过对两种邻居推荐方法的比较分析，本文选择了 K 邻居方法以良好地适应系统初期运行时存在的冷启动问题。

3.4.5 物品推荐过程

在本文实现的推荐系统中，我们选用固定数量的邻居（K-neighborhoods）作为相似邻居计算的方法。

（1）首先，从用户相似度状态中根据用户 ID 查询出指定数量的相似邻居，按相似度倒序排列。根据相似邻居 ID 从用户偏好状态表中获取评分项目。以上述示例中的用户“User0”为例，则查询出的相似邻居及其评分项如下表（表 3.7）所示：

表 3.7 邻居、评分项表

Neighbor	Similarity	Item
1	0.40	2,3,4
3	0.33	1,3,4,5

（2）通过用户 ID 从用户偏好状态表中获取用户的评分项目，之后从相似邻居的评分项目中移除当前用户已经偏好的项目，针对每一个备选评分项目累加对应的相似度，从而得到项目的推荐值，用“Value”表示，则得到以下项目—推荐值列表（表 3.8）：

表 3.8 项目—推荐值表

Item	Value
4	0.73
5	0.33

（3）最后，根据推荐任务的实际需求，将项目—推荐值列表按项目推荐值倒序排列，并选取固定数量的项目，并推荐给目标用户。

3.5 本章小结

本章首先对几种协同过滤推荐算法的基本思想进行了对比，选用基于用户的方案通过结合 Trident 计算框架进行了推荐算法设计。其中通过结合谷本系数进行用户间相似度计算，之后通过 K 邻居进行相似用户推荐，从而实现算法的整体思路。

第四章 基于 Storm 的推荐系统设计与实现

在充分了解流计算技术工作原理和协同过滤推荐算法的基础上，以 Storm 平台为基础，结合 Trident 计算框架，设计了一个基于 Storm 的协同过滤推荐系统，其中提供了基于用户的协同过滤推荐算法的分布式实现。

在本章接下来的内容里，首先分析了基于海量数据的个性化推荐的实际需求，并针对推荐场景的需求进行系统架构设计，然后对系统架构中的各个子系统和组件进行描述。其中重点介绍了基于用户的协同过滤推荐算法在分布式流计算平台 Storm 上的实现，通过算法计算流程图，从具体实现的角度介绍系统中一些关键功能的实现和所使用的技术，并给出了具体的 Trident 计算实现流程。

4.1 系统需求分析

推荐系统在如今的互联网等行业中被广泛的使用，包括大家经常使用的电子商务产品推荐、相关搜索、社交网络上的好友推荐、话题推荐等等。在目前许多互联网领域的主要公司中，推荐系统都是重要的产品与服务，例如亚马逊的个性化产品推荐、Google 的个性化广告、Facebook 的好友推荐以及 Netflix 的视频推荐，它们都拥有庞大的用户基数，通过持续地获取用户交互数据以及用户偏好来提供个性化服务。而在上述的使用场景中，推荐系统的实时交互需求以及海量数据处理能力的需求正在慢慢扩大。

因此本文提出的推荐系统在设计时主要考虑到以下方面：

（1）用户的访问具有随机性，在访问过程中会产生观看、购买、评分等各种行为，并且可以获取到其历史行为记录。

（2）在用户偏好发生变化的同时，系统需要有及时响应的能力，并且实时地向用户进行推荐。

从需求上看，系统不仅需要具备针对海量数据的并行计算能力，也要达到能够与用户实时交互的特性。用户交互数据通常以连续的流数据形式出现，对用户数据的获取是一个不断增量的过程，系统需要支持随机写入。总体来说，系统需要同时具备在线随机读写和数据并行处理的能力。

从算法性能上看，系统需要以流计算的方式完整地实现基于用户的协同过滤推荐算法，达到已有算法的预测准确度。在进行推荐的同时，能够收集到活跃用户的反馈数据，以持续优化系统的推荐质量。

4.2 总体架构设计

本文在对系统架构进行设计时，主要结合了电子商务网站的运行特点，通过结合 Storm 流计算框架的分布式特点，并依据基于 Trident 的协同过滤算法计算流程。如下图 4.1 所示，系统总体架构分为以下几个模块：数据收集模块、DRPC Server 模块、数据流计算模块以及 Redis 存储模块。各个模块之间伴随着数据的大量交互，图中的箭头表示了各模块间的数据交互关系。

数据收集模块主要负责对来自子商务网站的用户及商品数据、用户访问网站过程中的行为数据进行收集，并传递给推荐系统进行计算；DRPC Server 模块将发来的数据请求分配到 Storm 集群进行计算[49]，并且将接收到的计算结果返回给客户端；数据流计算模块主要通过 Trident 计算框架实现用户相似度计算以及相似邻居推荐这两个流程；Redis 存储模块负责计算过程中相关数据在内存中的持久化存储[50]。下文将分别对各个模块的功能及主要运行流程进行详细的介绍。

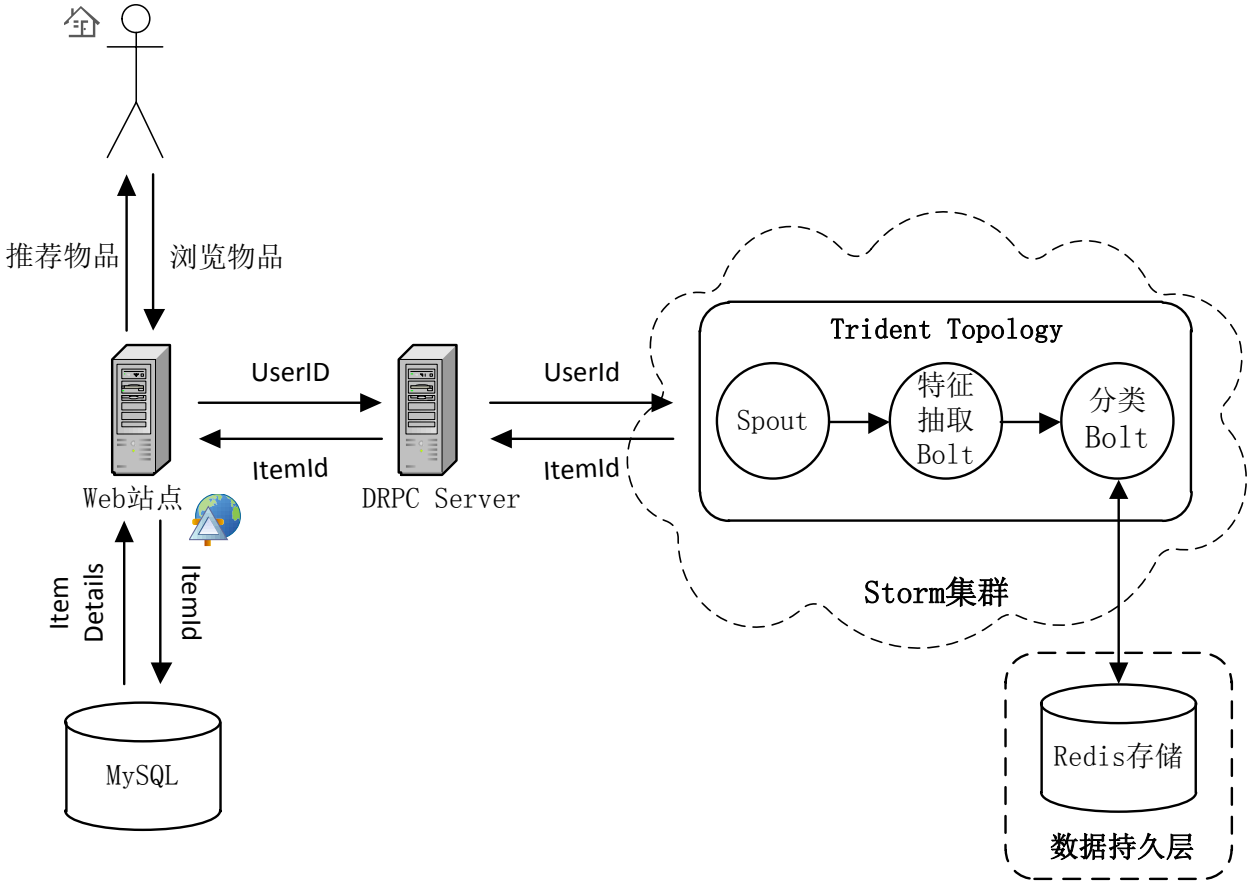

图 4.1　系统总体架构设计图

4.3 数据收集模块

本文设计的推荐系统数据来源假设从以下几个环境获取：

（1）来自电子商务网站的各类用户信息和商品信息；

（2）存储在系统日志中的用户行为数据，例如对某物品的浏览记录；

（3）用户对物品的直接操作，例如购买记录、评价记录等。

对于用户信息和物品信息可以共享的网站数据，在系统中会持续地迭代更新。从系统提取的日志文件中主要是解析出用户对物品的评价，依据模块需要转化为评分或其他指标。数据收集模块设计成多类收集和数据转化方案，可以根据推荐系统对数据的具体要求进行收集。不同的推荐系统也需要不同的数据，所以数据收集的方式可以适用于不同的推荐系统，通过设计一些公共接口，以根据不同的推荐系统选择不同的数据收集方案。

4.4 DRPC Server 模块

Storm 是一个分布式实时处理框架，它支持以 DRPC（distributed RPC，DRPC）方式调用。在 Storm 集群中，DRPC 提供了处理功能的访问接口。DRPC 用于对 Storm 上大量的函数调用进行并行计算过程。对于每一次函数调用，Storm 集群上运行的拓扑接收调用函数的参数信息作为输入流，并将计算结果作为输出流发射出去。DRPC 并非 Storm 的特性，是通过 Storm 的基本组件：Streams，Spouts，Bolts，Topologies 而衍生的一个模式。DRPC 可独立作为一项服务进行发布，但在分布式消息处理方面十分重要，因此在 Storm 集群中得到广泛的应用。下图（图 4.2）是 DRPC Server 在本系统中的具体工作流程：

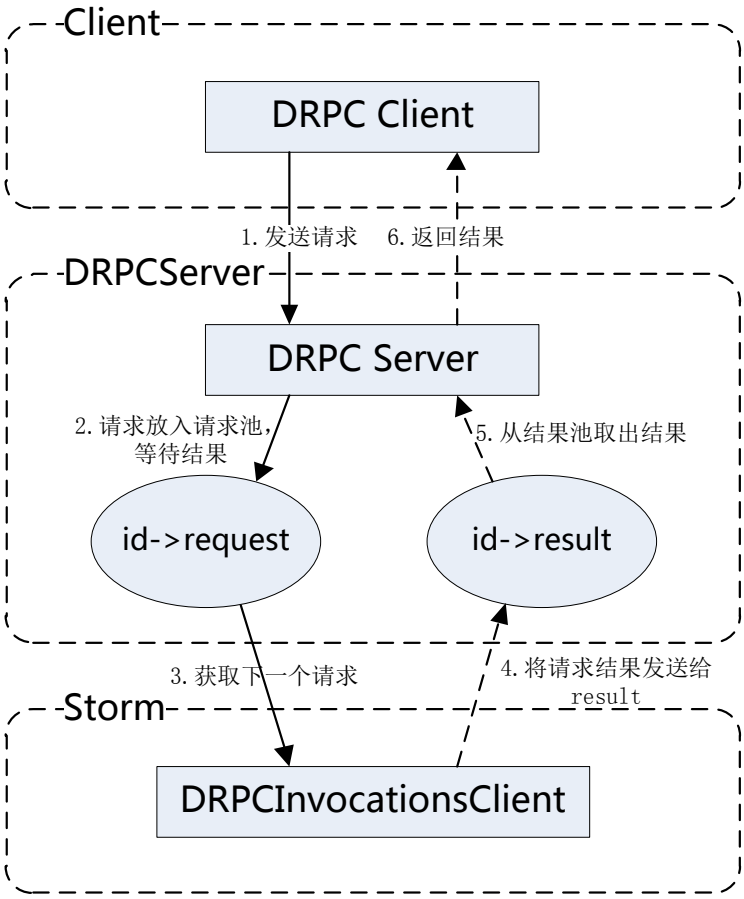

图 4.2 DRPC Server 工作流程图

在本文设计的推荐系统中，DRPC Server 的整体工作流程如下：

（1）首先负责接收客户端发来的 RPC 调用请求，一般包含被调用执行的 DRPC 函数名称及参数；

（2）DRPC Server 发送请求到 Storm 集群上的拓扑，即 Trident Topology；

（3）Trident Topology 通过 DRPC Spout 实现这一函数，从 DPRC Server 接收到函数调用流；

（4）每当调用一次函数 DRPC Server 会产生一个独立的 id；

（5）Trident Topology 开始进行计算，最后通过一个 Return Results 的 Bolt 连接到 DRPC Server，发送指定 id 的计算结果；

（6）DRPC Server 从 Storm 集群上接收计算结果，通过使用之前为每个函数调用生成的 id，将结果关联到对应的发起调用的客户端，将计算结果返回给客户端。

4.5 数据流计算模块具体实现

基于 Storm 推荐系统在算法实现上分为了用户相似度计算以及相似邻居推荐这两个流程，这也是本项目的核心模块，本节将重点介绍用户相似度计算流程和相似邻居推荐流程在 Trident 计算框架下的具体实现。

4.5.1 用户相似度计算流程

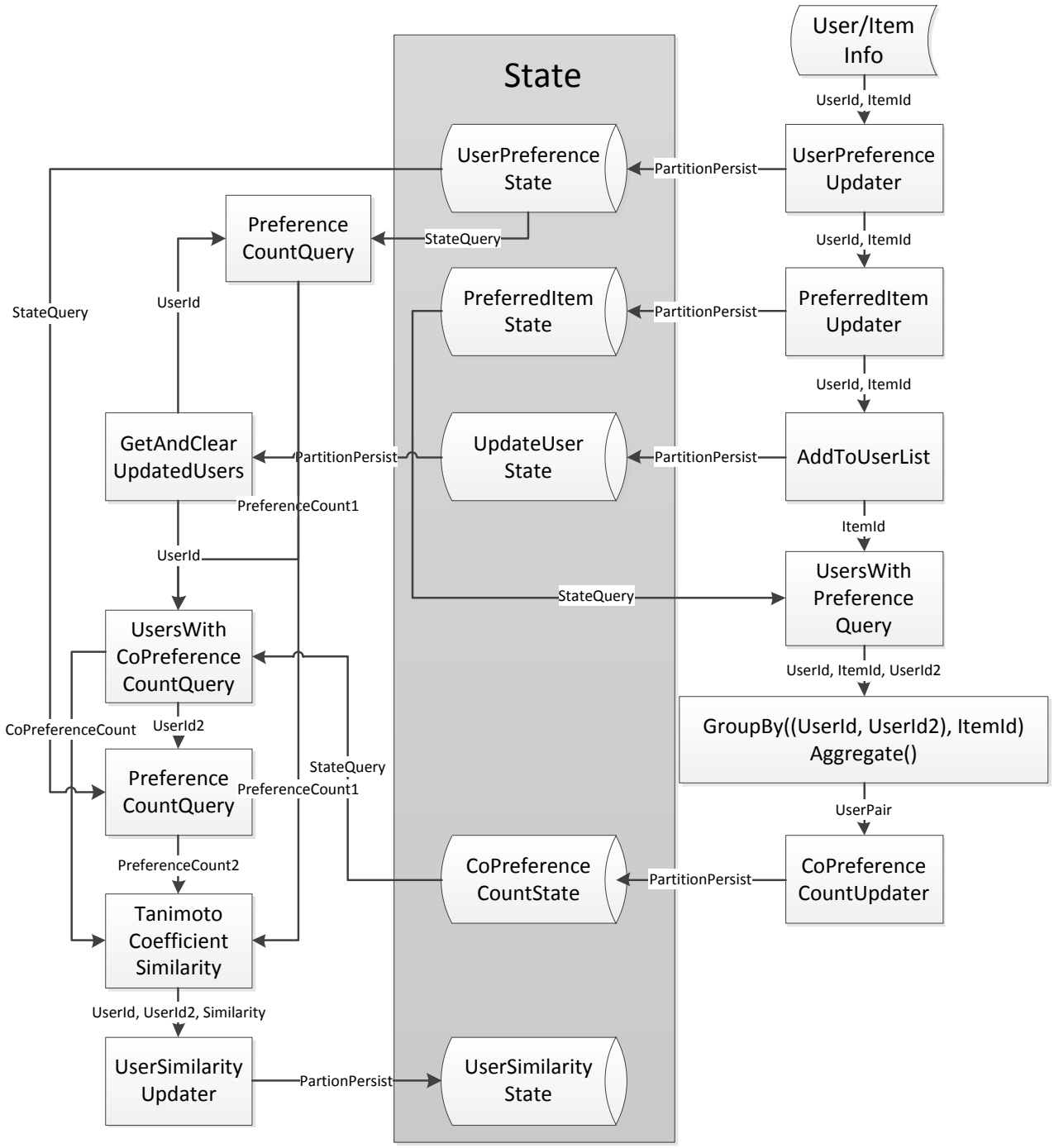

图 4.3 用户相似度计算流程图

上图（图 4.3）为用户相似度计算的具体实现流程，接下来将分为 UpdateUserPreferenceTopology 以及 UpdateSimilarityStream 两个阶段介绍图中计算流程的具体实现：

（1）UpdateUserPreferenceTopology 阶段

在第一阶段，首先构建一个 Trident 计算拓扑，它会提供了相应的接口去构建 Trident 计

算方法。通过从输入源（User/Item Info）中读取数据，创建新的数据流。Trident 在处理输入 stream 的时候会把输入转换成 batch（包含若干个 tuple）来处理，而处理这些 batch 的操作主要有 groupby、join、aggregation、执行 function 和 filter 等，并提供了功能来实现 batch 之间的聚合，并将计算结果存储到 Redis 内存数据库中。

第一阶段主要通过 PartitionPersist 操作将数据更新到 State 中暂存，以便后续的流计算使用。partitionPersist 操作会更新一个 State，其内部是将 State 和一批更新的 Tuple 交给 StateUpdater，由 StateUpdater 完成相应的更新操作。partitionPersist 会返回一个 TridentState 对象来表示被这个 Trident Topology 更新过的 location db。然后可以使用这个 state 在 topology 的任何地方进行查询操作。

（a）通过 UserPreferenceUpdater，将用户 ID 及其对应的具体偏好项目更新到 UserPreferenceState 中；

（b）通过 PreferredItemUpdater，将项目 ID 及其对应的用户 ID 更新到 PreferredItemState 中；

（c）通过 AddToUserList，将用户 ID 更新到 UpdateUserState 中；

（d）查询 PreferredItemState，根据项目 ID 查询对应的用户 ID（UserID2），将 UserID2 加入到流中；

（e）将 UserID，UserID2 封装成 UserPair 对象，根据 UserPair，Item 进行 GroupBy 操作，groupBy 操作先对流中的指定字段做 partitionBy 操作，让指定字段相同的 tuple 能被发送到同一个 partition 里，然后在每个 partition 里根据指定字段值对该分区里的 tuple 进行分组。在这一步，同时通过 Aggregate()方法对流进行聚合操作，以去除重复的 UserPair。Trident 中有 aggregate()和 persistentAggregate()方法对流进行聚合操作。aggregate()在每个 batch 上独立的执行，persistemAggregate() 对所有 batch 中的所有 tuple 进行聚合，并将结果存入 state 源中。aggregate()对流做全局聚合，当使用 ReduceAggregator 或者 Aggregator 聚合器时，流先被重新划分成一个大分区(仅有一个 partition)，然后对这个 partition 做聚合操作；另外，当使用 CombinerAggregator 时，Trident 首先对每个 partition 局部聚合，然后将所有这些 partition 重新划分到一个 partition 中，完成全局聚合。

（f）将聚合后的 UserPair 更新到 CoPreferenceCountState 中，并统计用户间偏好项目交集的个数。

（2）UpdateSimilarityStream 阶段

第二阶段主要通过状态查询（StateQuery）从 State 中获取需要的数据，Trident 提供了一个 QueryFunction 接口用来实现 Trident 中在一个 sourcestate 上查询的功能。同时还提供了一

个 StateUpdater 来实现 Trident 中更新 sourcestate 的功能。QueryFunction 的执行分为两部分。首先 Trident 收集了一个 batch 的 read 操作并把他们统一交给 batchRetrieve，batchRetrieve 会接受到多个用户 id。batchRetrieve 返还一个和输入 tuple 数量相同的 result 序列。result 序列中的第一个元素对应着第一个输入 tuple 的结果，result 序列中的第二个元素对应着第二个输入 tuple 的结果，以此类推。

StateQuery 会自动地批量处理。如果当前处理的 batch 中有 20 次更新需要被同步到存储中，Trident 会自动的把这些操作汇总到一起，只做一次读一次写，而不是进行 20 次读 20 次写的操作，在便捷执行计算的同时，保证了优异的性能。

（a）GetAndClearUpdatedUsers 从 UpdateUserState 中获取用户 ID，并清空，将获取到的用户 ID 发送 Tuple；

（b）PreferenceCountQuery 从 UserPreferenceState 中根据用户 ID 查询偏好项目的总数；

（c）UsersWithCoPreferenceCountQuery 从 CoPreferenceCountState 中根据 UserID 获取 UserID2 以及对应的偏好项目交集个数；

（d）PreferenceCountQuery 从 UserPreferenceState 中根据 UserID2 查询偏好项目的总数；

（e）TanimotoCoefficientSimilarity 通过从 PreferenceCountQuery 获取 PreferenceCount1，PreferenceCount2；从 UsersWithCoPreferenceCountQuery 获取 CoPreferenceCount，得到用以进行相似度计算的三种参数；再通过基于谷本系数的相似性计算方法得到用户间的相似度；

（f）TanimotoCoefficientSimilarity 将用户间相似度传递给 UserSimilarityUpdater，后者将数据更新至 UserSimilarityState。

至此，系统实现了对用户相似度的计算工作，在下一节，通过对 UserPreferenceState 以及 UserSimilarityState 的查询，实现对指定数量相似邻居的选取并向目标用户推荐物品。

4.5.2 相似邻居推荐流程

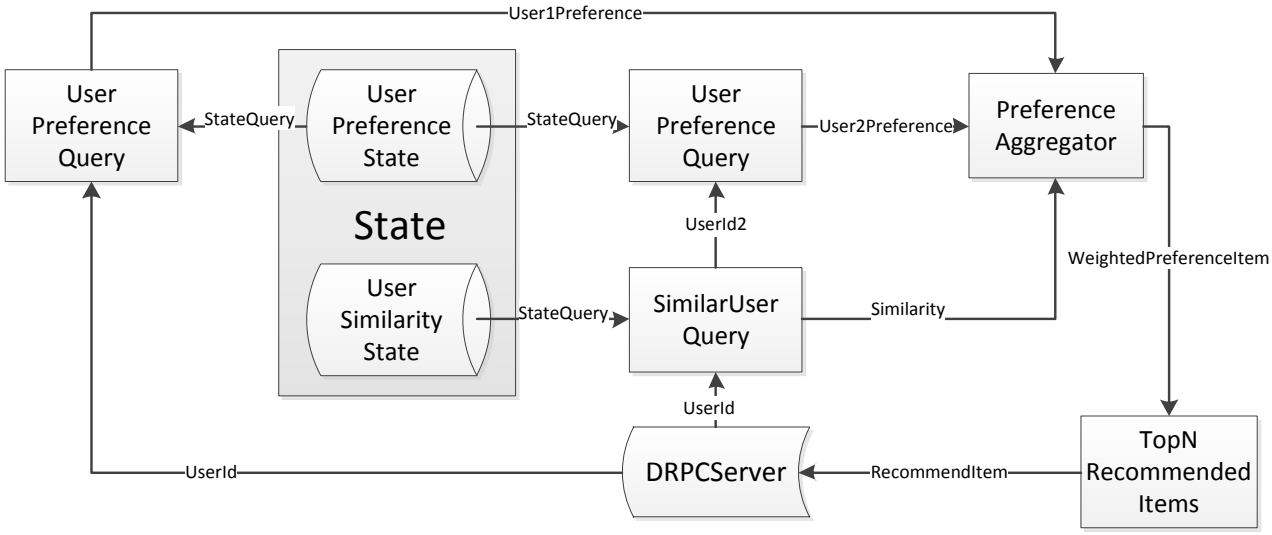

图 4.4 相似邻居推荐流程图

上图（图 4.4）为相似邻居选取并进行物品推荐的具体实现流程，接下来将详细介绍图中计算流程的具体实现：

（1）UserPreferenceQuery 对 UserPreferenceState 进行状态查询，根据用户 ID 从 UserPreferenceState 中获取用户偏好项目；

（2）SimilarUserQuery 根据用户 ID 从 UserSimilarityState 中查询指定数量个相似邻居（相似邻居数可自行设定参数）；

（3）UserPreferenceQuery 根据 SimilarUserQuery 传递的相似邻居 ID，从 UserPreferenceState 中查询其偏好的项目；

（4）PreferenceAggregator，从 UserPreferenceQuery 获取 User1Preference 和 User2Preference，从 SimilarUserQuery 获取邻居的相似度，通过 aggregate()方法对流进行聚合操作，从相似邻居偏好的项目中移除目标用户已经偏好的项目。将可推荐项目及相似度封装成 WeightedPreference 对象，并累加可推荐物品所对应的邻居相似度。

（5）在 TopNRecommendedItems 中，对 WeightedPreference 根据推荐度逆序排序，选择前 N 个项目（N 可自行设定）；

（6）最后，通过投影操作反保留 TopNRecommendedItems 中的 RecommendItem 字段。

至此，系统完成了对目标用户的物品推荐。

4.6 Redis 存储模块

在本系统的用户相似度计算流程中，Redis 负责将 UserPreferenceState、PreferredItemState、UpdateUserState、CoPreferenceCountState 以及 UserSimilarityState 持久化存储在内存中，提供给后续的流计算工作进行状态查询。

在读写有状态的数据源方面，Trident 具备优秀的抽象封装。数据状态既可以保留在 Storm 计算拓扑的内存，也可以存放至外部存储，在本系统中，我们选择 Redis 存储系统完成数据持久化的工作。

作为一款开源的键—值（key-value）存储工具，Redis 具备良好的存储性能。相比 Memcached 等类似的工具，Redis 能够支持存储的 value 类型更加丰富，包括字符串（strings）、哈希（hashes）、列表（lists）、集合（sets）和有序集合（sorted sets）等数据类型。这些数据类型都支持 push/pop、add/remove 以及取交集、并集和差集等更多类型的操作，而且这些操作都是原子性的。

Redis 通过使用内存中数据集（dataset）的方式来获取高效的存储性能。同时，Redis 通过周期性地将数据集进行存储，或者通过在日志的末端添加操作命令，以支持数据的持久化操作。

Redis 同样支持主从复制（master-slave replication），并且具有非常快速的非阻塞首次同步（non-blocking first synchronization）、网络断开自动重连等功能。同时 Redis 还具有其它一些特性，其中包括简单的事物支持、发布订阅（pub/sub）、管道（pipeline）和虚拟内存（vm）等。

4.7 本章小结

本章首先进行系统需求分析，之后对系统的总体架构设计进行展示，并分别对数据收集模块、DRPC Server 模块、数据流计算模块以及 Redis 存储模块的具体功能进行详细地介绍。

第五章 系统测试和结果分析

5.1 实验环境与测试数据集

5.1.1 实验运行环境

本系统的实验环境为通过流计算框架 Storm 搭建的分布式集群，Storm 版本为 0.9.2，Redis 版本为 3.0.0 RC4，一共使用四台虚拟机进行组网，模拟 Storm 集群环境，每台虚拟机分配单核 CPU 进行实验，详细配置情况如下表（表 5.1）所示：

表 5.1 集群环境配置表

主机名	配置参数	系统
Master	CPU：i7 2.3GHz 内存：2G	Ubuntu 14.04
Slave1	CPU：i7 2.3GHz 内存：2G	Ubuntu 14.04
Slave2	CPU：i7 2.3GHz 内存：2G	Ubuntu 14.04
Slave3	CPU：i7 2.3GHz 内存：2G	Ubuntu 14.04

5.1.2 实验数据集

由于对实际网站的系统日志文件、用户信息以及商品数据等进行采集具有较大的难度与阻碍，因此在对本文实现的推荐系统进行实验时，我们选用 MovieLens 数据集进行相关测试。

GroupLens 研究小组通过对从 20 世纪 90 年末到 21 世纪初由 MovieLens 用户提供的电影评分数据进行长时间的采集，形成了如今在推荐系统研究领域使用范围最广的 MovieLens 数据集[51]。MovieLens 数据集主要包括用户 ID 信息、产品 ID 信息以及用户对产品的评分信息（Rating），并提供了从 100KB、1MB 到 10MB 的多种规模数据集。

（1）MovieLens 100KB：由 1000 名用户对 1700 部电影进行的 100000 项评分信息组成；

（2）MovieLens 1MB：由 6000 名用户对 4000 部电影进行的 1000000 项评分信息组成；

（3）MovieLens 10MB：由 72000 名用户对 10000 部电影进行的 10000000 项评分信息以及 100000 条标签组成。

5.2 实验方案设计

本系统的测试主要针对两个方面开展，首先是针对基于 Storm 的协同过滤推荐系统的工作性能测试，通过对不同 batch 大小的设定，以及中间状态存储方法的选择分别进行试验，调试出推荐算法执行时最优化的参数，从而确定最适合本系统实际运行时的设定方案；之后是对本系统中基于用户的协同过滤推荐算法运行准确度的测试，该部分测试主要通过准确率（precision）/召回率（recall）进行度量，以验证算法计算准确度是否达到传统方案中算法运行的精确度。

5.2.1 性能测试

（1）冷启动性能测试

系统冷启动性能测试，在不同集群规模的情况下，展示系统首次启动时的数据加载以及相似度计算完成的时间。系统完成启动之后，推荐系统在 Storm 集群中处于持续运转的状态，新加入的数据所带来的计算工作一般可在秒级进行响应，因此本文主要针对系统的首次计算任务进行性能测试。

（2）数据吞吐量测试

在 Storm 中，数据中间状态的保存方法默认为 Memory 模式，也就是直接内存存储。而上一章所介绍的 Redis 存储同样是中间状态存储的一种解决方案，并且更适用于分布式环境，本文通过在不同集群规模的环境下，通过设定不同大小的 batch，分别对两种存储方案的数据吞吐量进行测试，选择出更适用于本系统环境的方案。

5.2.2 准确率测试

在本系统的推荐准确率测试中，我们主要选取准确率/召回率度量的方法[52]，准确率和召回率是进行推荐系统评测时广泛应用的两个指标，用以评价推荐结果的质量。准确度是指计算出的相关项目数量与计算出的项目总数量的比率，衡量的是推荐算法的精准率；召回率是指计算出的相关项目数量与整体项目总数量的比率，衡量的是推荐算法的查全率。一般来说，准确度就是用于推荐的项目有多少是准确的，召回率就是所有可推荐的项目有多少被计算出来了。

令 R(u)是根据用户在训练集上的行为给用户做出的推荐列表，而 T(u)是用户在测试集上的行为列表。那么，计算推荐结果准确率的公式如下（公式 5.1）：

$$Precision = \frac{R(u) \cap T(u)}{R(u)} \quad (5.1)$$

同样，计算推荐结果召回率的公式如下（公式 5.2）：

$$Recall = \frac{R(u) \cap T(u)}{T(u)} \quad (5.2)$$

准确率是评估计算出的项目中推荐成功的项目所占的比例；召回率是从关注范围中，召回目标类别的比例。取值范围都在 0 到 1 之间，结果越趋向 1，准确率和召回率就越高。

本文通过选取不同的推荐列表长度 K，计算出不同推荐列表长度下的每组准确率/召回率，并从中选择最优方案。

5.3 实验结果分析

（1）冷启动性能测试

在 Batch 大小设定为 200，数据持久层通过 Redis 进行中间状态存储的情况下，对 10000 组（即 1MB 大小）的 MovieLens 数据集进行测试，得到的系统启动性能测试结果为：

表 5.2 系统启动性能测试

集群规模（台）	数据集加载时间（s）	相似度计算时间（s）
1	1229	462
2	743	247
3	521	179
4	397	126

通过表 5.1 中的实验结果可以看出，随着集群规模的扩大，系统性能会有较为明显的提升，但首次冷启动无论数据集加载时间还是相似度计算时间都维持在分钟级以上，因为在首次加载数据时，系统对内存运算压力较大。在之后有数据进入运算后，系统都可以在秒级进行响应。

（2）数据吞吐量测试

在数据吞吐量的测试中，测试数据集固定选取 10000 组（即 1MB 大小）的 MovieLens 数据集。分别在数据持久层通过自带的 Memory 存储方式以及通过 Redis 存储方案两种情况下进行性能对比，运行时间都维持在 10 秒，测量三次取平均值进行对比。

在集群数量为 1 台情况下，数据吞吐量测试结果如下表（表 5.3）所示：

表 5.3 单台机器数据吞吐量测试

BatchSize（p/s）	Memory	Redis
10	50/49/53	55/51/49
50	264/223/257	223/236/241
200	917/1062/1021	583/529/457

在集群数量为 2 台情况下，数据吞吐量测试结果如下表（表 5.4）所示：

表 5.4 两台机器数据吞吐量测试

BatchSize（p/s）	Memory	Redis
10	51/47/53	99/104/101
50	261/254/249	396/405/412
200	1025/965/997	991/972/1003

在集群数量为 3 台情况下，数据吞吐量测试结果如下表（表 5.5）所示：

表 5.5 三台机器数据吞吐量测试

BatchSize（p/s）	Memory	Redis
10	48/55/54	156/139/154
50	252/239/257	594/607/599
200	979/1037/1004	1521/1458/1483

在集群数量为 4 台情况下，数据吞吐量测试结果如下表（表 5.6）所示：

表 5.6 四台机器数据吞吐量测试

BatchSize（p/s）	Memory	Redis
10	53/50/49	207/185/193
50	228/241/257	823/810/796
200	1058/977/968	1893/1944/1972

将四张表中的数据取平均值并整理成折线图之后，我们可以直观的看出集群数量对系统吞吐量性能的影响。具体对比如下图（图 5.1）所示：

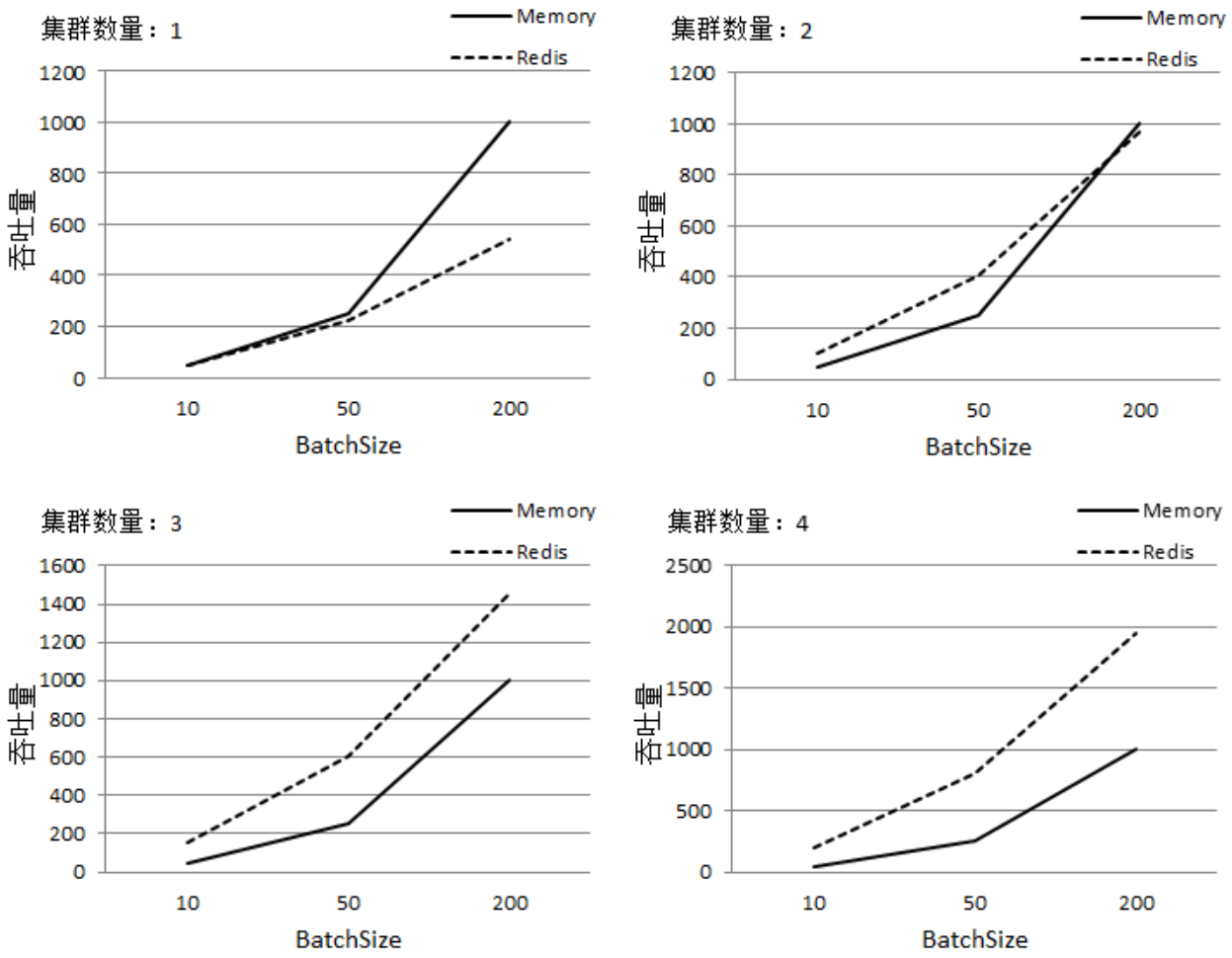

图 5.1 数据吞吐量测试对比图

当集群数量为一台时，随着 BatchSize 的变大，Memory 的数据吞吐量要明显优于 Redis，主要原因在于 Redis 是针对集群环境设计的存储系统，由于执行计算时集群需要做节点同步，因此在性能上会有一定的损失；而 Memory 模式只能在单台机器上进行运算，无法通过集群的方式进行内存共享，因此也无法达到性能水平扩展的目的。从上述的测试数据中也能看出，随着集群数量的增加，Memory 模式的数据吞吐量并没有得到提升，Memory 模式只能通过提升单台机器性能的方式提高运算效率。

当集群数量为两台时，Redis 存储方案的吞吐量已经基本和 Memory 模式一致。通过对集群数量的不断增加，我们可以看到，当集群数量增加为 3 至 4 台时，性能上的水平扩展使得 Redis 存储方案的吞吐量有了明显的提升。

由此可以看出，在单台机器上进行系统相关功能测试，或者验证单节点性能提升对系统运算效率的影响时，适合通过 Memory 模式进行操作。而 Redis 存储方案更适合系统在集群中的实际应用。

（3）准确率测试

同样，通过对 MovieLens 数据集进行准确率和召回率的测试，在相似邻居数 K 值的变化

下，本系统的算法性能指标如下表（表 5.7）所示：

表 5.7 准确率测试结果

相似邻居数（K）	准确率（precision）	召回率（recall）
5	17.03%	8.35%
10	20.46%	10.13%
20	23.17%	11.27%
40	24.73%	12.09%
80	24.94%	11.96%
160	24.82%	12.14%

从表中数据的变化趋势可以看出，推荐系统的准确率和召回率在相似邻居数超过 20 个之后趋于稳定，准确率保持在 25%左右，而召回率维持在 12%。算法的精度指标并不随着 K 值的增加而一直提高，因此找到合适的相似邻居数对于提高系统的预测精度更为关键。

5.4 本章小结

本章主要进行系统测试和结果分析，首先介绍实验环境和测试数据集，之后分别从系统启动性能、数据吞吐量和推荐算法准确性这三个方向开展测试工作和实验结果比对，并对实验结果进行相关分析。

第六章 总结与展望

随着信息技术和互联网的发展，人类逐渐从信息匮乏的时代走入了信息过载的时代，信息资源的展现需要更精准地符合每一位用户的个性化需求，推荐系统的关注度日益提升，在互联网众多领域的应用也逐渐的普及与深入。本人针对国内相关应用研究领域的空白，重点研究了分布式流计算框架 Storm，并深入学习了推荐系统相关理论知识，全面了解推荐系统相关技术原理，通过尝试将推荐算法与流计算技术结合使用，设计了一个基于流计算技术的推荐系统。

本论文基于开源的分布式流处理系统 Storm，利用其分布式 RPC 和 Trident 高层抽象等特性，结合 Redis 存储方案等相关技术，设计实现了一个基于 Storm 的协同过滤推荐系统，使得推荐系统中的数据能够在分布式环境下实时进行处理，适用于处理计算量大、并对执行效率需求较高的周期性数据更新任务，满足了推荐系统在数据实时性方面的需求。根据实验和实际应用效果，基于 Storm 的协同过滤推荐系统，相对于传统方案实现的协同过滤推荐系统无论在并行计算性能、数据实时反馈以及系统可扩展性方面都更具优势。通过 Storm 提供的框架实现的实时数据处理应用，有效降低了系统开发部署的复杂性。

由于时间、科研环境和个人水平等因素的制约，本系统仍有很大的改进空间。论文可以从以下几个方面继续进行探索研究：

（1）系统未能在实际的商业环境下进行工作，针对系统的测试，目前只使用了 MovieLens 的数据集，但这个数据集并不能完整反应系统对实际商品的推荐情况，因此需要针对更加真实的网站数据进行测试并对系统进行优化。

（2）在系统应用的推荐算法方面，本文只选用了经典的基于用户的协同过滤算法，主要因为考虑到在 Storm 框架下设计实现较复杂的算法在工作量和技术实现上还存在一定的难度。未来可以尝试研究将改进的协同过滤推荐算法或混合推荐方法在 Storm 框架上以流计算的方式实现。

（3）针对目前系统中的资源配置以及推荐算法的参数调整，采用的依然是比较传统的手动设置方法，如果通过运用 Web 管理推荐系统，并添加 Storm 框架相应的资源管理功能模块，可以实现更为便捷的系统操作并提升工作效率。

参考文献

[1] 冷亚军，陆青，梁昌勇．协同过滤推荐技术综述[J]．模式识别与人工智能，2014，(8). DOI:10.3969/j.issn.1003-6059.2014.08.007.

[2] Harman M, Afshin Mansouri S, Zhang Y. Search-based software engineering: Trends, techniques and applications[J]. ACM COMPUTING SURVEYS, 2012, 45(1):7-7.

[3] Ishii A, Suzumura T. Elastic Stream Computing with Clouds[J]. Cloud Computing (CLOUD), 2011 IEEE International Conference on, 2011:195 - 202.

[4] Neumeyer L, Robbins B, Kesari A. S4: Distributed Stream Computing Platform.[J]. Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, 2010:170 - 177.

[5] 尹绪森. Spark与MLlib：当机器学习遇见分布式系统[J]．程序员, 2014, (7):112-115.

[6] 王峰. Hadoop生态技术在阿里全网商品搜索实战[J]. 2014中国数据库技术大会, 2014

[7] 余伟．基于云计算的分布式搜索技术研究[D]．武汉科技大学, 2011. DOI:10.7666/d.y1943867.

[8] Goldberg D, Nichols D, Oki B M, et al. Using collaborative filtering to weave an information tapestry[J]. Communications of the ACM, 1992.

[9] Resnick P, Iacovou N, Suchak M, et al. GroupLens: An Open Architecture for Collaborative Filtering of Netnews[J]. In Proceedings of the 1994 ACM conference on Computer supported cooperative work, 1994.

[10] Linden G, Smith B, York J. Amazon.com recommendations: item-to-item collaborative filtering[J]. Internet Computing, IEEE, 2003, 7(1):76 - 80.

[11] Bell R M, Koren Y, Volinsky C. The BellKor solution to the Netflix Prize[J]. KorBell Team's Report to Netflix, 2007.

[12] Qin S, Menezes R, Silaghi M. A Recommender System for Youtube Based on its Network of Reviewers.[J]. Social Computing, IEEE Second International Conference on, 2010:323 - 328.

[13] 黄原原，中共天津市委党校，天津300191．百度搜索技术及其个性化信息搜索探析[J]．农业图书情报学刊, 2010, 22(2):84-87. DOI:10.3969/j.issn.1002-1248.2010.02.023.

[14] 惠佩瑶，王鹏．我国电子商务发展问题研究--以阿里巴巴为例[J]．商场现代化，2014，(9):39-39. DOI:10.3969/j.issn.1006-3102.2014.09.027.

[15] 马晓迪，宣琦，张哲等．协同过滤推荐算法在豆瓣网络数据上的研究[J]．计算机系统应用，2014，(8):18-24. DOI:10.3969/j.issn.1003-3254.2014.08.003.

[16] 李一辰，李绪志，阎镇．实时流计算在航天地面数据处理系统中的应用[J]．微电子学与计算机，2014，(9).

[17] Liu Y, Pan W, Cao N, et al. System anomaly detection in distributed systems through MapReduce-Based log analysis[J]. Advanced Computer Theory and Engineering, International Conference on, 2010:V6-410 - V6-413.

[18] 朱炳宇．基于Storm云平台的地图道路匹配算法研究[D]．云南大学, 2013.

[19] 黄馥浩．基于Storm的微博互动平台的设计与实现[D]．中山大学, 2013.

[20] Han J, Ishii M, Makino H. A Hadoop performance model for multi-rack clusters[C]. //Computer Science and Information Technology (CSIT), 2013 5th International Conference on. IEEE, 2013:265 - 274.

[21] Liu X, Han J, Zhong Y, et al. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS[C]. //Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on. IEEE, 2009:1 - 8.

[22] 黄承真，王雷，刘小龙等．Hadoop任务分配策略的改进[J]．计算机应用，2013，33(8):2158-2162. DOI:10.11772/j.issn.1001-9081.2013.08.2158.

[23] Hu J, Wang W, Gu C, et al. Retrieving Functional Knowledge from Existing Products for Reuse[J]. Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on, 2012, 01(11):3 - 6.

[24] Bobadilla J, Hernando A, Ortega F, et al. A framework for collaborative filtering recommender systems[J]. Expert Systems with Applications, 2011, 38(12):14609–14623.

[25] 周伟, 黄穗, 朱蔚恒等. 一种基于集体智慧的图书评价推荐方法[J]. 计算机工程与科学, 2011, 33(9):141-144. DOI:10.3969/j.issn.1007-130X.2011.09.025.

[26] 钟克吟. 基于混合推荐的学术资源推荐系统的服务模式与数据挖掘[J]. 图书馆学研究, 2013, (11).

[27] 周萍, 张子柯, 章恬等. 一种基于社会化媒体和社会网络结构的混合推荐模型[J]. 上海理工大学学报, 2014, (3):267-276. DOI:10.13255/j.cnki.jusst.2014.03.012.

[28] 陈志祥, 马士华, 王一凡. 用户满意度评价模型及实证分析[J]. 系统工程, 1999, (3):43-49.

[29] 王伟杰. 基于评分预测的协同推荐若干问题研究[D]. 华东师范大学, 2014.

[30] Park Y. The Adaptive Clustering Method for the Long Tail Problem of Recommender Systems[J]. Knowledge and Data Engineering, IEEE Transactions on, 2013, 25(8):1904 - 1915.

[31] Xin L, Song C. Cloud-based innovation of Internet long tail[C]. //Product Innovation Management (ICPIM), 2011 6th International Conference on. IEEE, 2011:603 - 607.

[32] 安维, 刘启华, 张李义. 个性化推荐系统的多样性研究进展[J]. 图书情报工作, 2013, 57(20):127-135. DOI:10.7536/j.issn.0252-3116.2013.20.022.

[33] 艾聪聪. 推荐系统中多样性和新颖性算法研究[D]. 湖南大学, 2014.

[34] 马建威, 徐浩, 陈洪辉. 信息推荐系统中的朋友关系预测算法设计[J]. 国防科技大学学报, 2013, 35(1):163-168. DOI:10.3969/j.issn.1001-2486.2013.01.029.

[35] Yuan-hong W, Xiao-qiu T. A real-time recommender system based on hybrid collaborative filtering[C]. //Computer Science and Education (ICCSE), 2010 5th International Conference on. IEEE, 2010:1909 - 1912.

[36] 杨福萍, 王洪国, 董树霞等. 基于记忆效应的协同过滤推荐算法[J]. 计算机工程, 2012, 38:63-66. DOI:10.3969/j.issn.1000-3428.2012.23.015.

[37] 刘庆鹏, 陈明锐. 优化稀疏数据集提高协同过滤推荐系统质量的方法[J]. 计算机应用, 2012, 32(4):1082-1085. DOI:10.3724/SP.J.1087.2012.01082.

[38] 李忠俊, 周启海, 帅青红. 一种基于内容和协同过滤同构化整合的推荐系统模型[J]. 计算机科学, 2009, 36(12):142-145. DOI:10.3969/j.issn.1002-137X.2009.12.034.

[39] 张振亚, 王进, 程红梅等. 基于余弦相似度的文本空间索引方法研究[J]. 计算机科学, 2005, 32:160-163. DOI:10.3969/j.issn.1002-137X.2005.09.041.

[40] Adler J, Parmryd I. Quantifying colocalization by correlation: The Pearson correlation coefficient is superior to the Mander's overlap coefficient[J]. Cytometry Part A, 2010, 77A(8):733–742.

[41] 宋宇辰, 张玉英, 孟海东等. 一种基于加权欧氏距离聚类方法的研究[J]. 计算机工程与应用, 2006, 26:179-180.

[42] Fligner M, A F M, Fligner M A, et al. A Modification of the Jaccard–Tanimoto Similarity Index for Diverse Selection of Chemical Compounds Using Binary Strings[J]. Technometrics, 2002, 44(2):110-119.

[43] Aiyer A, Pyun K (, Huang Y, et al. Lloyd clustering of Gauss mixture models for image compression and classification[J]. Signal Processing: Image Communication, 2005, 20:459–485.

[44] 马宏伟, 张光卫, 李鹏. 协同过滤推荐算法综述[J]. 小型微型计算机系统, 2009, 30(7):1282-1288.

[45] 金晓军. Trident Storm与流计算经验[J]. 程序员, 2012, (10).

[46] Christian Bosshard D. Cascading of Second-Order Nonlinearities in Polar Materials[J]. Advanced Materials, 1996, 8(5):385–397.

[47] 吴建平, 迟学斌. 分布式系统上并行矩阵乘法[J]. 计算数学, 1999, 21(1):99-108. DOI:10.3321/j.issn:0254-7791.1999.01.012.

[48] Sanchez J L, Serradilla F, Martinez E, et al. Choice of metrics used in collaborative filtering and their impact on recommender systems[C]. //Digital Ecosystems and Technologies, 2008. DEST 2008. 2nd IEEE International Conference on. IEEE, 2008:432 - 436.

[49] X Xia, L Tian. A Web Server Cluster Solution Based on Twitter Storm[J]. Journal of Data Analysis and

Information Processing. Vol.2 No.1(2014).

[50] 廖钢, 刘旭宁, 徐波. 基于Redis+Greasemonkey+J2EE技术的促销信息展示系统的设计与实现[J]. 软件导刊, 2012, 11(1):95-97.

[51] Chen Y, Harper F M, Konstan J, et al. Social Comparisons and Contributions to Online Communities: A Field Experiment on MovieLens[J]. The American Economic Review, 2010, (4):1358-1398.

[52] 刘建国, 周涛, 郭强等. 个性化推荐系统评价方法综述[J]. 复杂系统与复杂性科学, 2009, 6(3):1-10. DOI:10.3969/j.issn.1672-3813.2009.03.001.

附录 1 图表目录

附录 2 攻读硕士学位期间撰写的论文

（1）陈俊安、薛浩然、吉琨，A daily load forecasting model based on data mining，Environmental Science and Biological Engineering(ESBE2014)，已录用。

致谢

随着论文工作的完成，近三年的研究生学习生活即将画上句号。十分感谢在此期间给予我帮助与支持的老师、同学和朋友。

能在郑彦老师的指导下进行学习研究，我感到非常的荣幸。郑老师为我论文的完成提供了很多不可或缺的指导与帮助，无论在论文开题阶段研究方向的选择，还是研究工作中遇到的各种疑难，郑老师总会耐心地引导我找到解决问题的方法与思路。在此，我向郑老师表示由衷地感激。

感谢与我在同一师门的李雪迪、陈静、纪琳琳和李月同学。在教研室的学习生活中，我们都相互激励、同舟共济；在论文研究遇到困难时，我们齐心协力，彼此出谋划策。感谢在硕士生涯期间，有他们的相伴。

同样要感谢在系统实现过程中给予我帮助的好友陈帅，虽然在公司研发任务繁重，但他总能不厌其烦的为我解答技术上的细节问题，为我系统的实现带来了至关重要的协助。

最后，感谢评审老师在百忙之中对我的论文莅临指导。