華南理工大學

SOUTH CHINA UNIVERSITY OF TECHNOLOGY

学生研究计划(SRP)项目验收
项目成员个人结题报告

参加项目名称：	基于分布式计算框架的 实时推荐算法研究
参加项目编号：	X201910561600
参加起止时间：	2019 年 4 月 至 2020 年 4 月
指导教师姓名：	吴广潮
指导教师所在学院：	数学学院
学生姓名：	周泽华
学生学号：	201730471263
学生手机号：	17329906489
学生所在学院：	数学学院
学生所学专业：	信息与计算科学
填表日期	2020 年 4 月

学生研究计划(SRP)项目验收项目成员个人结题报告

（基于分布式计算框架的实时推荐算法研究）

目录

一、引言

面对海量的商品信息，利用有效的推荐系统提高服务质量，吸引和留住优质用户已经成为各大电商平台关注的焦点。而商品数量的指数级别增长已经使得传统的单机算法模型到达性能瓶颈，因此如何提高推荐结果的质量与时效性是当前推荐系统面临的重大挑战。本项目在 Spark 分布式框架上，利用 spark streaming 部件对传统的单机推荐算法进行改写，以用户行为日志记录作为数据流，根据用户的兴趣偏好对推荐结果集进行更新，达到实时推荐的目的。

1.1 文献调研

书籍：

《推荐系统实践》

有关实时推荐算法的文献：

TencentRec: Real-time Stream Recommendation in Practice

Real-time Video Recommendation Exploration

Fast Matrix Factorization for Online Recommendation with Implicit Feedback

1.2 研究方案

1.从分布式计算框架入手，了解分布式计算的思想和使用方法。

2.学习推荐算法的基础思想，对推荐算法有基础的了解。

3.检索实时推荐算法和分布式推荐算法的相关文献，通过文献进一步深化了解分布式计算思想和实时推荐算法思想。

3.将推荐算法进行分布式实现。

4.将分布式推荐算法进行分布式实时化。

5.对分布式实时推荐算法进行试验及评估

1.3 项目组成员间分工

周泽华（本人）：主要负责实时推荐算法的研究和实现。

彭清桦：主要负责对小组成员算法思想改进并予以实现。

刁光明：主要负责对分布式框架使用场景和性能特点研究。

钟沅：主要负责文件检索和对文献中各种算法优缺点分析。

刘盛豪：主要负责推荐算法的分布式实现和优化。

苏沁涵：主要负责对小组成员收集的相关文献进行梳理和整合，将文献中算法进行数学推导。

1.4 本人主要工作和成效

本人的主要工作是负责实时推荐算法的研究。本人在推荐算法研究过程中提高了自身的文献检索能力，对每种实时推荐算法优点和使用条件有较为全面的理解，并对分布式推荐算法予以实现和对分布式实时推荐算法予以实现。

二、摘要

2.1 关键词

分布式，推荐算法，实时

2.2 研究目的

实现分布式的实时推荐算法，并评估算法的各项性能指标。

2.3 研究方法

通过分布式计算框架和框架可用的高级语言来实现算法，并用公开数据集对算法性能进行评测。

2.4 研究结论

分布式实时推荐算法较传统分布式推荐算法有着速度更快，推荐实时性更强的优点。

三、架构概述

在这一节里会概述我们的计算平台选择，流数据获取方法，以及数据存储办法。

3.1 计算平台选择

我们选择 Spark。

Apache Spark 是一个快速通用的集群计算系统，使用最先进的 DAG 调度器、查询优化器和物理执行引起，实现了批处理和流式数据的高性能。Spark 提供了 Java、Scala、Python、R 的高级 api，我们选择了其中的 Python api。Spark 集群可以 Standalone，也可以跑在 Hadoop、Apache Mesos、Kubernetes 等集群中，而我们选择将 Spark 跑在较传统的 Hadoop 集群中，使用 YARN 对 Spark job 进行调度。

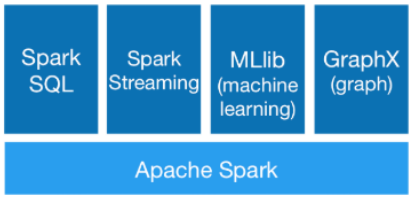


图 1 Spark 技术栈

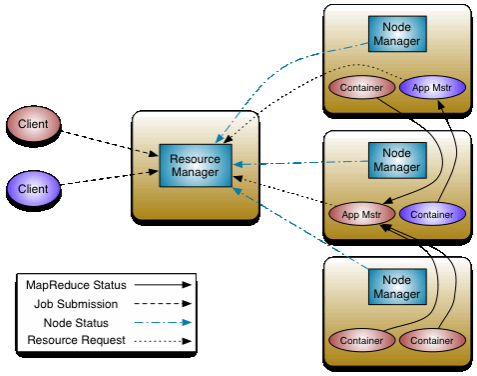


图 2 YARN 调度逻辑图

Apache Spark 不仅可以通过 SparkCore 进行十分快速的基于内存的 Map Reduce 模型计算与开发，同时也一栈式提供了为 SQL 计算和 DataFrame 计算的 SparkSQL 库、提供了为流处理使用的 SparkStreaming 库。

我们选用 SparkCore 与 SparkSQL 来对数据进行计算，选用 SparkStreaming 来对流数据进行处理。

3.2 流数据获取

我们选用 Flume 和 Kafka。

Flume 是一个分布式的、高可靠高可用的分布式海量日志采集、聚合和传输系统。Flume 支持在日志系统中定制各类数据发送方，用于收集数据，同时 Flume 提供对数据进行简单处理，并写到各种数据接收方的能力。

Kafka 是一个高吞吐量的分布式发布订阅消息系统。Kafka 可以发布和订阅记录流，类似于消息队列或企业消息传递系统。Kafka 可以用于简历实时流数据管道，在系统或应用程序之间可靠地获取数据。

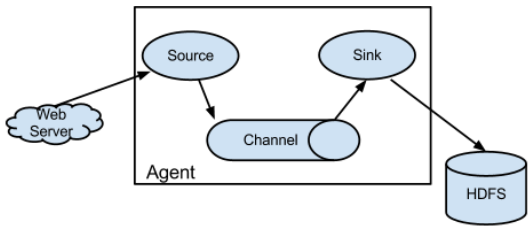


图 3 Flume 结构图

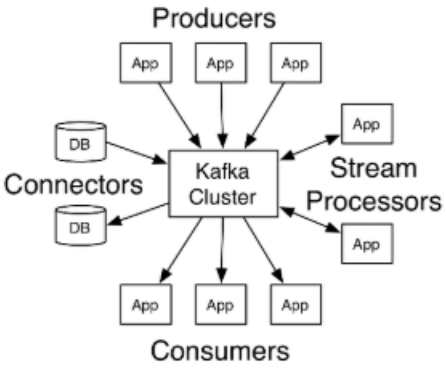


图 4 Kafka 结构图

我们使用 Flume 来对日志变化进行收集，将数据流写入到 Kafka 进行数据缓冲，最后由 Kafka 将数据流发布到 SparkStreaming 进行流处理

3.3 数据存储

我们选用 MySQL 和 Redis

MySQL 是一个关系型数据库管理系统，是最流行的关系型数据库管理系统之一。关系型数据港将数据保存在不同的表中，使用 SQL 语音。MySQL 具有体积小，速度快等优点。

Redis 是一个基于内存的亦可持久化的日志型、Key-Value 数据库，即 Redis 是一个 NoSQL 数据库。Redis 是一个高性能的 Key-value 数据库。Redis 的读取速度可以达到 110000 次/s，写的速度可以达到 81000 次/s。

我们使用 MySQL 来对计算结果进行存储，使用 Redis 来对需要存储的计算中间结果进行存储（内存，而不是硬盘），以提高计算速度。

图 5 Redis

图 6 MySQL

四、算法设计

这一节我们会深入阐述我们的算法公式和思想。

鉴于目前对推荐算法的熟悉程度和对分布式计算的熟悉程度，我们实现了分布式的实时基于物品的协同过滤算法（Distributed Real Time Item-based Collaborative Filtering Algorithm）。

由于对分布式机器学习及分布式机器学习框架仍不够熟悉，我们暂时还未实现采用基于隐语义（LFM）/矩阵分解（Matrix Factorization）的分布式实时推荐算法。

4.1 基于物品的协同过滤（Item-based Collaborative Filtering）算法

Item-based Collaborative Filtering，基于物品的协同过滤算法是目前业界应用最多的算法之一，亚马逊，Netflix，Hulu，YouTube 等网站的推荐算法的基础都是该算法。

基于物品的协同过滤算法（简称 ItemCF）的一个简单易懂的其作用解释是——给用户推荐那些和他们之前喜欢过的物品相似的物品。比如，该算法因为用户之前喜欢《射雕英雄传》，给用户推荐《天龙八部》。ItemCF 算法主要是通过分析用户行为记录计算物品之间的相似度。ItemCF 认为，物品 A 和物品 B 具有很大的相似度是因为喜欢物品 A 的用户大都也喜欢物品 B。

基于物品的协同过滤算法主要分为两步：

（1） 计算物品之间的相似度（称为 similarity）

（2） 根据物品的相似度和用户的历史行为给用户生成推荐列表

我们先假定，我们的推荐系统中一共有n个用户(users)，m个物品(items)。我们将用户u给物品p的评分记为$r_{u,p}$，那么每个元素由$r_{u,p}$组成的一个矩阵R便是一个$n * m$的矩阵。一个物品可以被一个由n个用户的评分所组成的向量表示，定义向量$i_p = [r_{1,p}, r_{2,p}, \dots, r_{n,p}]$，代表物品$p$被$n$个用户的评分，显然${i_p}^T$便是$R$的第$p$列的列向量。

我们可以用如下公式定义物品p和物品q的相似度：

$$sim(i_p, i_q) = \frac{i_p \cdot i_q}{\|i_p\| \|i_q\|} = \frac{\sum_{u \in U} r_{u,p} r_{u,q}}{\sqrt{\sum {r_{u,p}}^2} \sqrt{\sum {r_{u,q}}^2}} \tag{1}$$

其中U表示用户的集合，$\|\cdot\|$为二范数（L2-norm）。特别地，如何用户u没有给物品p评分，则这个$r_{u,p} = 0$。

得到物品的相似度后，我们需要预测用户u对一个物品p的评分来决定是否将这个物品p推荐给用户u。

我们用如下公式来预测用户u给物品p的分数：

$$\hat{r}_{u,p} = \frac{\sum_{i_q \in N^k(i_p)} sim(i_p, i_q) r_{u,q}}{\sum_{i_q \in N^k(i_p)} sim(i_p, i_q)} \tag{2}$$

其中$N^k(i_p)$代表与物品p的k邻近集(k neighbors)，换句话说，就是与物品p相似度($sim(i_p, i_q)$)最高的k个物品q

4.2 解决隐反馈问题（Implicit Feedback Problem Solution）

算法精确的关键地方在于能否准确地捕捉用户对物品的偏好，这些偏好通常只能由用户的评分来反映。然而在实际情况中，大量的数据是隐式反馈，并不总是能得到明确的反馈，即用户的行为是间接的。例如算法的数据如果只来自用户对物品的星级评分这一行为，而不考虑用户别的行为，并且由于我们只能猜测用户的偏好，如果处理不当，隐式反馈这个问题可能会降低推荐系统的性能。为了解决隐式反馈问题，我们建议为用户的多种行为设置不同的评分。用户的多种行为如：单击、浏览、购买、共享、评论等，我们可以给浏览行为对应为 1 星级评分，而购买为 3 星级评分。而对于用户表现出多重兴趣的物品，即多重行为，我们采用这些行为对应的最大评分作为用户对项目的评分，以减少各种隐式反馈带来的混乱影响（noise）。

我们定义 co-rating 为用户u给物品p和物品q的较小评分

$$co\_rating(i_p, i_q) = \min(r_{u,p}, r_{u,q}) \tag{3}$$

其中$r_{u,p}$是用户u给物品p的所有行为评分中的最高分。

由于$co_rating(i_p, i_q)$我们将$r_{u,p}r_{u,q}$这一项改为了$\min(r_{u,p}, r_{u,q})$，所有我们需要重新定义物品p和物品q的相似度公式来确保相似度落在范围$[0,1]$。重新定义的相似度公式如下：

$$sim(i_p, i_q) = \frac{\sum_{u \in U} \min(r_{u,p}, r_{u,q})}{\sqrt{\sum r_{u,p}}\sqrt{\sum r_{u,q}}} \tag{4}$$

其中将公式(1)中为二范数的$\|i_p\|$改为$\sqrt{\sum r_{u,p}}$替代

此外，为了更加清晰地描述，我们这样认为：当用户对某个物品表现出隐含兴趣时，用户会对该物品进行评价或者说用户喜欢该物品。

4.3 可横向扩展的增量更新（Scalable Incremental Update）

除了隐式反馈的问题，由于用户的评分数据$r_{u,p}$不断变化，实时（real-time）的 ItemCF 有着需要频繁更新相似度矩阵（表）R的问题。在传统的推荐系统中，大多采用静态时间间隔更新的策略，也就是隔一段固定的时间，更新整个相似度矩阵。然而，周期性的更新不能满足实时推荐的要求，因为这是一个快速变化的场景。为了要捕获用户的实时兴趣，我们需要一个采用增量更新策略的，易横向扩展的 ItemCF 算法。

我们将实际的 ItemCF 算法中将物品对（item pair）的相似度分为三部分：

$$sim(i_p, i_q) = \frac{pairCount(i_p, i_q)}{\sqrt{\text{itemCount}(i_p)}\sqrt{\text{itemCount}(i_q)}} \tag{5}$$

其中

$$\text{itemCount}(i_p) = \sum r_{u,p} \tag{6}$$

$$pairCount(i_p, i_q) = \sum_{u \in U} co\_rating(i_p, i_q) = \sum_{u \in U} \min(r_{u,p}, r_{u,q}) \tag{7}$$

在这个方法中，物品对的相似度可以被分成$\text{itemCount}(i_p)$，$\text{itemCount}(i_q)$，和$pairCount(i_p,i_q)$，最后再组合，这样就可以被增量更新了。我们可以独立地计算$\text{itemCount}(i_p)$，$\text{itemCount}(i_q)$，和$pairCount(i_p,i_q)$这三个公式的新的值，再组合这三个公式新的值来计算得出新的相似度。所以，我们的相似度可以被增量更新用如下的公式：

$$sim(i_p,i_q)' = \frac{pairCount(i_p,i_q)'}{\sqrt{\text{itemCount}(i_p)'}\sqrt{\text{itemCount}(i_q)'}}$$

$$= \frac{pairCount(i_p,i_q)+\Delta co\_rating(i_p,i_q)}{\sqrt{\text{itemCount}(i_p)+\Delta r_{u,p}}\sqrt{\text{itemCount}(i_q)+\Delta r_{u,q}}} \quad (8)$$

其中$sim(i_p,i_q)'$表示在得到用户对物品的新评分<user,item,rating>元组（tuple）后计算得到的新的相似度。

我们使用三个并行计算层来实时增量更新物品对的相似度，如图所示。

第一步，将得到的包含了用户 ID，物品 ID，评分等的信息的元组<user,item,rating>按用户 ID 分组（group by userID），这样可以让我们并行地使用不同用户的行为历史（User History）。此外，我们需要在用户的行为历史中保存旧的评分记录和旧的co_rating，这样我们就可以通过新的评分记录对比旧的评分记录来确定哪些评分需要被更新和哪些co_rating需要被重新计算。在这之后，我们将新评分信息$\Delta r_{u,p}$和更新了的$\Delta co_rating(i_p,i_q)$按 itemID 分组（group by itemID）后传到下一层（第二层）。

第二步，即在第二层，我们存储有每个物品的$itemCount$的旧值和物品对的$pairCount$的旧值。通过得到上一层的新评分信息$\Delta r_{u,p}$与更新了的$\Delta co_rating(i_p,i_q)$，由公式（8），我们可以增量地并行地为每个物品更新$itemCount$和每个物品对更新$pairCount$。我们将更新了的$itemCounts$和$pairCounts$送到下一层（第三层）。

第三步，在第三层，在我们得到更新了的$itemCounts$和$pairCounts$，我们可以通过公式（5）计算新的相似度。注意到，只要每个物品对相似度在一个特定的节点上计算，这样计算就可以被安全的放缩。

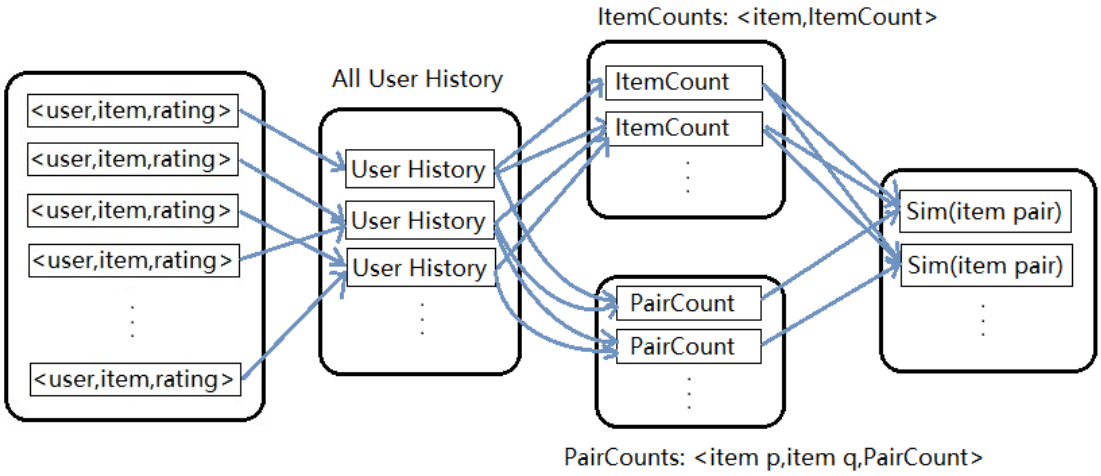


图 7 多个并行计算层的 ItemCF

4.4 实时剪枝（Real-time Pruning）

在 ItemCF 中，如果每个用户有过 100 个操作记录，则每一个新的用户操作都会导致数百次计算，其中大部分都是不必要的，因为很大一部分的物品之间是不太相似的——因为只有等式（2）中的$N^k(i_p)$的项对我们预测评分有用，从而导致浪费了很多的计算资源。

为了解决这个问题，我们利用霍夫丁界（Hoeffding Bound）理论，开发了一种实时的剪枝（real-time pruning）方法，即利用霍夫丁界来为数据流建立一个非常快速的决策树。

霍夫丁界理论可以表述为：一个实变量x，x的取值范围是R（对于相似度，则范围是$[0,1]$），如果我们对这个变量x做了n个相互独立的观测，并计算了这些观测值的均值$\bar{x}$。霍夫丁界指出：在给定概率$1-\delta$下，变量x的真实平均值至多为$\bar{x}+\varepsilon$，其中

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{9}$$

在我们的推荐算法中，我们将两个物品p，q之间的相似度作为随机变量x，设t为两个物品相似度列表的最小阈值的实际计算结果，即$N^k(i_p)$，$N^k(i_q)$之间的最小相似度的值。给定δ，霍夫丁界保证：如果已经有n次相似度的更新，并有$\varepsilon < t - \bar{x}$，则物品之间的正确相似度不大于$t$的概率为$1-\delta$。换句话说，给定$\delta$下，已有$n$次相似度的更新并满足$\varepsilon < t - \bar{x}$，则这两个物品的相似度都不在对方 Topk 相似度列表（$N^k(i_p)$）中的概率为$1-\delta$。由于相似度计算涉及两个物品，所以剪枝是双向的，即对两个物品都要剪枝。

下面给出了具有实时剪枝的 ItemCF 算法，其中n_{ij}记录了$sim(i_i, i_j)$的更新次数，L_i记录了物品i需要修剪的物品集合。由于 SparkStreaming 是微批处理，并不是真正的实时处理，而实时剪枝算法适合真正的实时处理框架。我们只给出实时剪枝的伪代码，不在算法中应用。

Algorithm 1: Item-based CF Algorithm with Real-time Pruning

Input: user rating action recording user u and item i

Get L_i
for *each item* j *rated by user* u **do**
  if j *in* L_i **then**
    Continue
  end
  Update pairCount(i, j)
  Get itemCount(i) and itemCount(j)
  Compute $sim(i, j)$ using Equation 5
  Increment n_{ij}
  Get threshold t_1 of i's similar-items list
  Get threshold t_2 of j's similar-items list
  $t = min(t_1, t_2)$
  Compute ϵ using Equation 9
  if $\epsilon < t - sim(i, j)$ **then**
    Add j to L_i
    Add i to L_j
  end
end

图 8 实时剪枝算法伪代码

4.6 冷启动问题（Cold Start）

冷启动问题主要分三类：

1. 用户冷启动：主要解决如何给新用户做个性化推荐的问题
2. 物品冷启动：主要解决如何将新的物品推荐给可能对它感兴趣的用户这一问题

3. 系统冷启动：主要解决如何在一个新开发的系统上（还没有用户和用户行为）设计个性化推荐服务

解决方案：

用户冷启动：

我们给新用户推荐热门程度最大的物品。由于用户容易对热门程度大的物品产生行为，这样我们就可以根据新的用户行为给用户推荐更多物品，这是 ItemCF 算法的特性之一。

物品冷启动：

由于 ItemCF 的特性，需要有用户对这个物品产生过行为才能为这个物品计算相似度然后才能将这个物品推荐给其他用户，所以需要与其他算法结合才能更好的解决物品冷启动这个问题。这个问题不再叙述。

系统冷启动：

同物品冷启动问题，由于 ItemCF 是基于用户行为的算法，需要用户行为才能计算相似度才能进行推荐，而不能很好地解决在没有用户行为数据的系统冷启动问题。这问题不再叙述。我们只能利用已有数据来进行计算来启动推荐系统。

五、实施细节

在本节，我们将介绍推荐系统实施的细节。介绍了几个程序的概述，以及实施过程中如何解决问题的说明。（本文还揭示了实现中的一些优化。）

5.1 技术栈

在我们的实时推荐系统中，我们将整个推荐流程**分为三个阶段**，所以将技术栈也分为三个层次。第一个层是数据获取层，用于获取计算所需要的数据，如用户操作行为数据，用户评分数据等。第二层是算法层，用于实施我们的实时推荐算法。第三层是存储层，用于保存计算中间结果或者保存计算结果。技术栈如图所示。我们现在开始结合三个阶段和三层技术栈来说明我们的推荐系统运行流程。

第一阶段：

新的用户操作行为数据或者用户评分数据，首先会存储在日志服务器上的日志文件中。Flume 首先监听日志服务器上的日志文件的变化，将日志文件变化的内容收集，并传输到 Kafka 中进行缓冲。Kafka 得到了消息，转换成消息队列并将消息送到 SparkStreaming，由 SparkStreaming 接受，这样就进入第二阶段。

第二阶段：

SparkStreaming 得到了 Kafka 的消息，便开始使用这个消息并结合已有的数据：$UserHistory$、$ItemCount$和$PairCount$来增量更新计算出新的$UserHistory$、$ItemCount$和$PairCount$。在增量更新结束后，计算出新的相似度（$Similarity$）和将相似度的 TopK 进行存储（$TopKSimilarity$）。在所有增量更新后，我们可以为当前活跃的用户推荐或者为所有用户计算出推荐结果。

当然，我们也可以在 SparkStreaming 得到了消息后，不更新相似度直接利用已有的 TopK 相似度为当前活跃用户进行推荐计算。

第三阶段：

在$UserHistory$、$ItemCount$和$PairCount$增量更新计算后，将这三个结果均存储到 Redis 中。TopK 相似度也可以选择是否存入 Redis，存入的话易于直接利用相似度对用户进行推荐计算。

我们将推荐的结果存入 MySQL，利于前段工程师进行推荐结果的使用。我们也可以定期将 Redis 中的数据存入 MySQL，以进行备份。

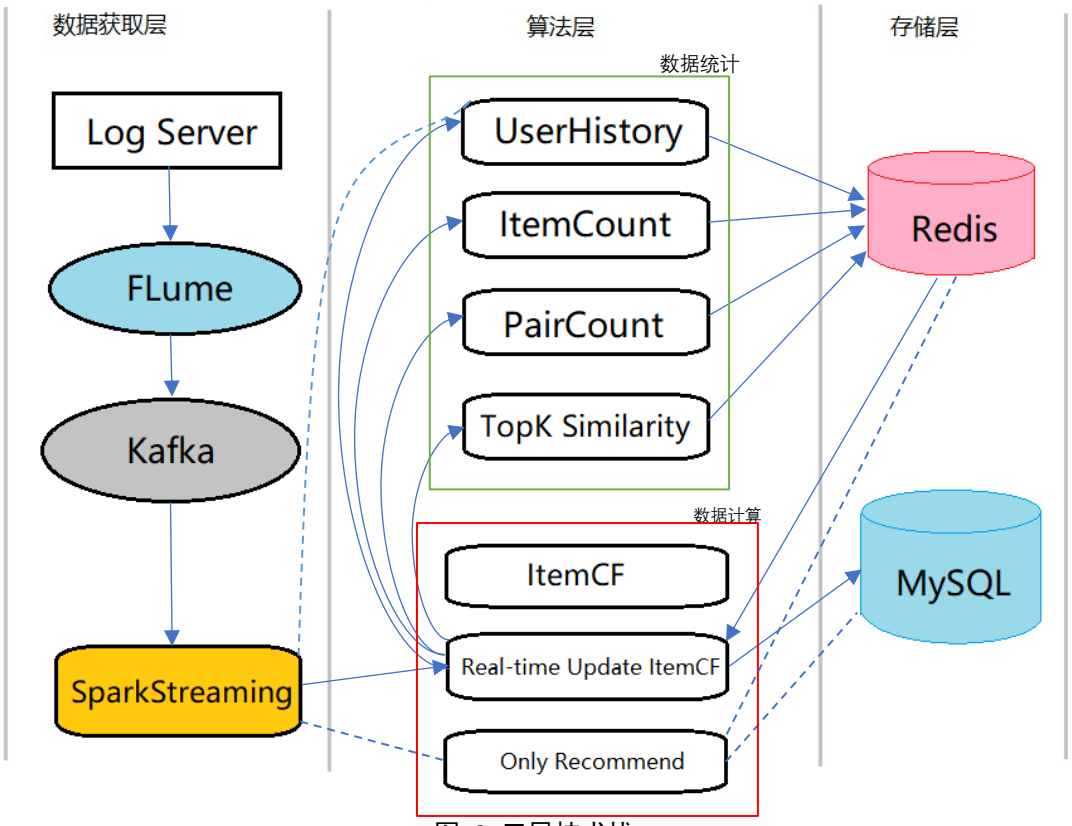


图 9 三层技术栈

5.2 数据获取层细节

Flume 细节

Flume 选用两层 Agent 设计。第一层使用 exec 命令检测日志文件变化，从 exec 收集数据(exec source)，将结果发送到 avro(avro sink)。第二层使用从 avro 收集数据(avro source)，将结果发送到 Kafka（kafka sink)。

选用两层的优点在于：如果 Hadoop\Spark 集群、Kafka 需要停机维护或升级，对外部第一 1 层 Flume Agent 没有影响，只需要在第二层做好数据的接收与缓冲即可，待维护或升级结束，继续将第二层缓存的数据导入到数据存储系统（Kafka)。

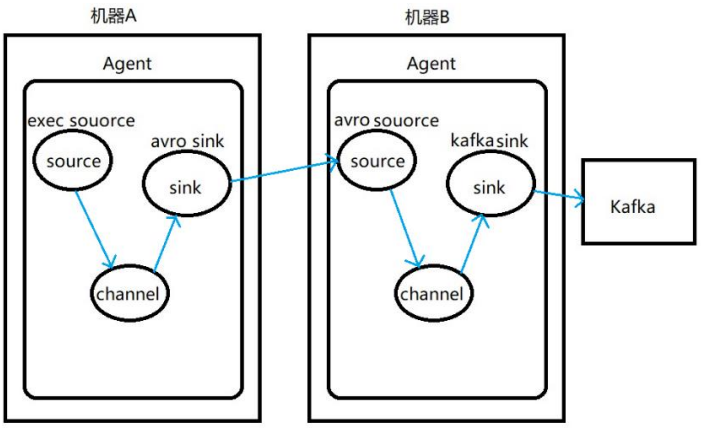


图 10 Flume 细节

Kafka 细节

Kafka 使用的生产者（producer）为 Flume，消费者（consumer）为 SparkStreaming，主题（topic）为 TencentRec。

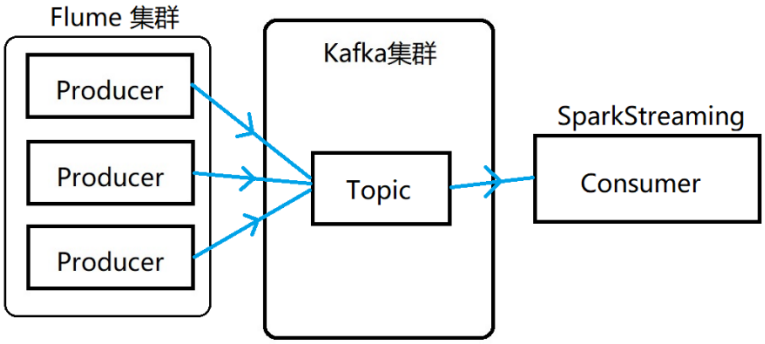


图 11 Kafka 细节

5.3 算法层细节

每一段时间 SparkStreaming 从 Kafka 消费了新的用户行数据形成一个 batch 后，我们首先需要从 Redis 读取用户历史和我们的曾经的计算结果：$UserHistory$、$ItemCount$、$PairCount$。读取数据后，我们用新的数据更新$ItemCount$、更新$PairCount$、更新$UserHistory$。接着我们计算新的相似度并计算 TopK 相似度并用这 TopK 相似度结合新的$UserHistory$为刚刚产生了行为的用户计算新的推荐列表。

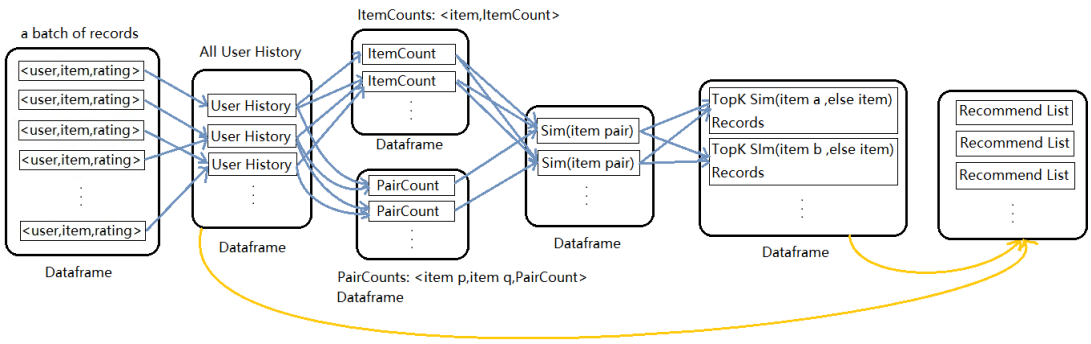


图 12 算法层细节

5.4 存储层细节

每一段时间 SparkStreaming 从 Kafka 消费了新的用户行数据形成一个 batch 后，我们首先需要从 Redis 读取上次计算的结果，也就是$UserHistory$、$ItemCount$、$PairCount$的旧数据。通过新到来的记录计算出新的$UserHistory$、$ItemCount$、$PairCount$的数据，并写入 Redis 用于下次计算使用。推荐结果写入 MySQL。

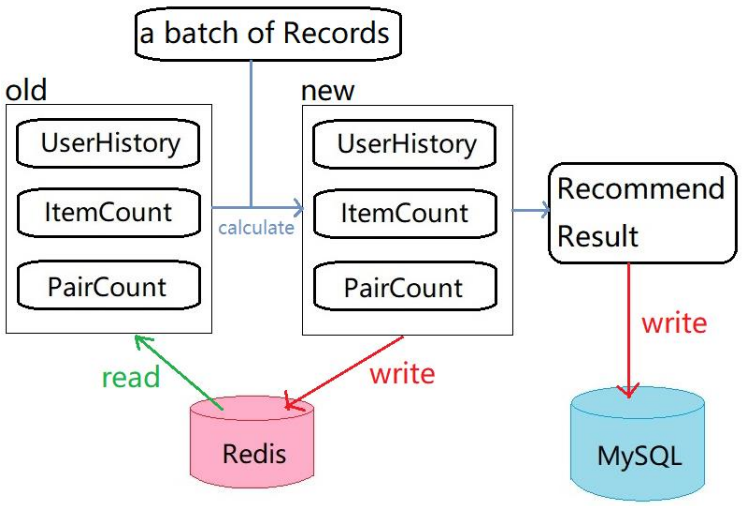

a batch of Records
old
UserHistory
ItemCount
PairCount
calculate
new
UserHistory
ItemCount
PairCount
Recommend
Result
write
read
write
Redis
MySQL

图 13 存储层细节

六、实验设计

本节将描述推荐算法的实验目的，实验设计方法及推荐算法性能的评测指标设计。

6.1 实验目的

我们需要评测 ItemCf 算法的推荐结果性能以及实时更新的 ItemCF 的计算速度与于普通 ItemCF 的计算速度的对比，并与利用已有的 TopK 相似度直接实时推荐的计算速度对比。

6.2 实验设计方法

首先将用户行为数据集按照均匀分布随机分成M份（本文章取$M = 5$），我们挑选其中一份作为测试集，将剩下$M - 1$份作为训练集。为了评测指标不是过拟合（over fitting）的结果，需要进行M次试验，并且每次都是使用不同的集。然后将M次实验测出的评测指标的平均值作为最终的评测指标。

我们选择将数据集进行M次随机划分，就可以得到M个不同的训练集和测试集，然后分别进行实验，用M次实验的评测指标的平均值作为最后的评测指标。

推荐结果性能实验设计

在训练集上建立 ItemCF 模型，然后计算出推荐结果——每个用户的N个物品的推荐列表（本文取$N = 10$），并在测试集上对用户行为进行预测，统计出相应的评测指标。

实时更新 ItemCF 计算性能实验设计

在训练集上建立 ItemCF 模型。在测试集上使用 Real Time Update 对训练集上的 ItemCF 进行更新，然后计算出推荐结果，记录评测指标。在训练集用 ItemCF 进行计算，得出 TopK 相似度的结果，直接利用结果对测试集用户进行推荐，记录评测指标。在整个数据集使用 ItemCF 普通计算，然后计算出推荐结果，记录评测指标。对比三个方法的评测指标。

6.3 评测指标

准确率(Precision)/召回率(Recall)

对用户u推荐N个物品（记为$R(u)$），令用户u在测试集上喜欢的物品集合为$T(u)$，然后可以通过准确率/召回率评测推荐**算法的精度**

准确率：

$$Recall = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |T(u)|} \tag{10}$$

召回率：

$$Precision = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |R(u)|} \tag{11}$$

准确率描述最终的推荐列表中有多少比例是发生过用户-物品评分记录。召回率描述有多少比例的用户-物品评分记录包含在最终的推荐列表中。

覆盖率(Coverage)

除了计算推荐算法的精度，我们还计算算法的覆盖率。覆盖率反映了推荐算法发掘长尾的能力，覆盖率越高，说明推荐算法越能够将长尾的物品推荐给用户。这里我们采用最简单的覆盖率定义：

覆盖率：

$$Coverage = \frac{|\cup_{u \epsilon U} R(u)|}{|I|} \tag{12}$$

覆盖率表示最终的推荐列表中包含多大比例的物品。如果所有的物品都被推荐给至少一个用户，那么覆盖率就是 100%。

新颖度(Novelty)

我们还要评测推荐的新颖度，这里用推荐列表中物品的平均评分次数的倒数？的来度量推荐结果的新颖度。如果物品的平均评分次数很高，说明推荐的物品较热门，推荐结果新颖度较低，否则说明推荐结果比较新颖

新颖度：

$$Novelty = \left(\frac{\sum_u \sum_{i \epsilon R(u)} Popularity(i)}{\sum_u |R(u)|}\right)^{-1} \tag{13}$$

其中$Popularity(i)$表示物品i历史上总共被评分的次数。

计算速度(Calculate Speed)

我们要对比普通 ItemCF 的计算速度与实时更新的 ItemcF 的计算速度。我们用他们的计算时间的来展现他们的计算速度，如果时间很长，说明计算速度很慢，反之说明计算速度很快

七、实验结果

本节我们将描述实验如何实施以及实验结果展示

7.1 实验实施

数据集选择

我们采用 GroupLens 提供的 MovieLens 数据集。MovieLens 数据集有三个不同的版本，本文实验选择的版本的数据集包含 6000 多用户对 4000 多部电影的 100 万条评分。该数据将是一个评分数据集，，用户可以给电影评 5 个不同等级的分数（1～5 分）。

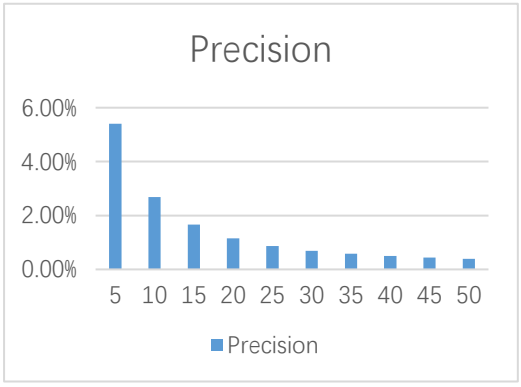
7.2 实验结果展示

一下结果均为在 MovieLens 数据集上 ItemCF 算法的离线实验的各项性能指标的评测结果。结果均为该算法在不同K值下的性能。

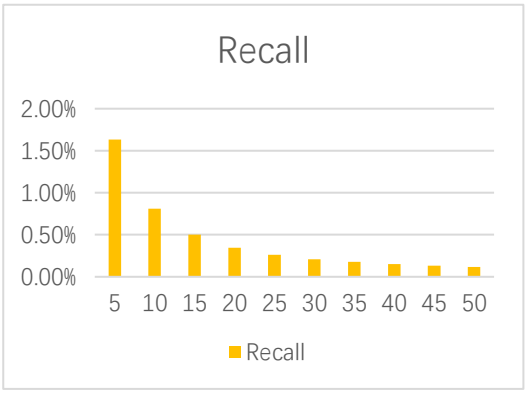
准确率(Precision)/召回率(Recall)

图表 1 为准确率随 k 值变化的实验结果，图表 2 为召回率随 k 值变化的实验结果。

准确度/召回率：可以看到 ItemCF 的推荐结果的准确率与召回率均与K大致成负相关



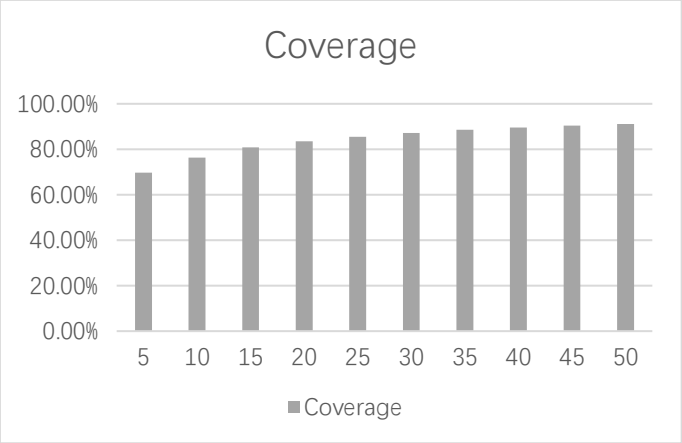
图表 1 准确率



图表 2 召回率

覆盖率(Coverage)

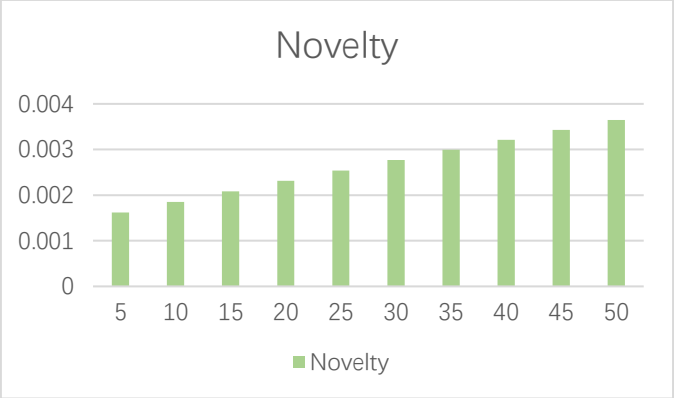
图表 3 为覆盖率随 k 值变化的实验结果，参数K对 ItemCF 的推荐覆盖率影响大致呈正相关，K值越大，推荐结果的覆盖率越高。



图表 3 覆盖率

新颖度(Novelty)

图表 4 是新颖度随K值的变化实验结果图，参数K对 ItemCF 的新颖度影响大致呈正相关，K值越大，推荐结果的新颖度越高。



图表 4 新颖度

推荐结果性能

表 1 推荐结果性能总表列出了在 MovieLens 数据集上 ItemCF 算法离线实验的各项性能指标的评测结果。该表包含了算法在不同K值下的性能指标的值。

在K值增加后，然覆盖率和新颖度有所上升，可是换来较低的准确率和召回率，高准确率召回率与覆盖率新颖率不可兼得。同时分析，在 MovieLens 数据集上，都是电影的评分，当新颖度底时，推荐的大多都是口碑好的电影，自然观看概率高，而当新颖度高时，推荐的电影可能口碑较差，所以准确度和召回率低。所以K值的选择基于我们推荐算法的结果需要。如果想给用户推荐新颖的结果，就使用较大的K值；如果想要用户产生更多行为，就使用较小的K值。

k	Recall	Precision	Coverage	Novelty
5	1.63%	5.41%	69.65%	0.0016205
10	0.81%	2.68%	76.34%	0.0018527
15	0.50%	1.66%	80.84%	0.0020829
20	0.35%	1.15%	83.56%	0.0023173

25	0.26%	0.87%	85.47%	0.0025413
30	0.21%	0.69%	87.19%	0.0027722
35	0.18%	0.59%	88.56%	0.0029928
40	0.15%	0.50%	89.59%	0.0032149
45	0.13%	0.44%	90.45%	0.003431
50	0.12%	0.39%	91.17%	0.0036462

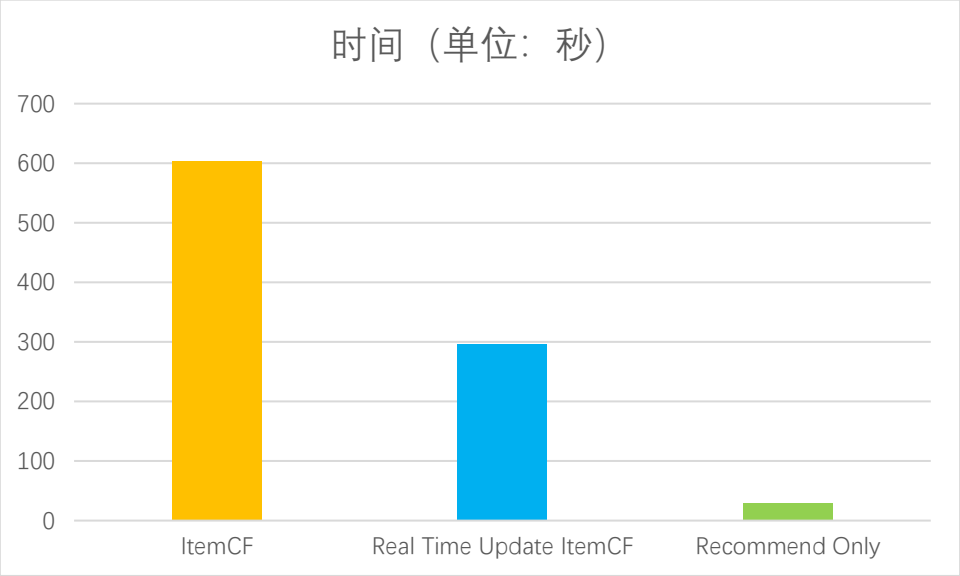
表 1 推荐结果性能总表

注：离线实验指标和在线实验指标有差距，离线实验指标仅供参考

计算速度(Calculate Speed)

通过图标 5 看出，在用户操作数据到达时，如果直接使用 ItemCF 算法从头计算，将会花费大量时间，导致得到结果的时间滞后，不够迅速。实时更新的 ItemCF 算法速度较从头计算的 ItemCF 算法有不少提升，可以显著提升得到结果的速度。利用已有的 TopK 相似度直接计算推荐结果速度十分快，比起实时更新还要快得多。

注：图标中只是计算时间，并未记录数据从硬盘的读写时间



图表 5 三种方法计算时间

八、结论及未来展望

8.1 结论

分布式实时推荐算法较传统分布式推荐算法有着速度更快，推荐实时性更强的优点。然而实时更新相似度仍需要庞大计算量，仍然不够迅速，如果对推荐结果产生速度要求极高，应该利用已有的 TopK 相似度直接产生推荐结果。并且，我们实现的分布式的实时基于物品的协同过滤算法是从普通的基于物品的协同过滤算法的推广，所以也同样继承了原本 ItemCF 的优缺点。

优点：

1. 长尾物品丰富，可以将冷门物品推荐出来
2. 用户有新行为，一定会导致推荐结果变化
3. 新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
4. 对于不同需求可以选择不同的K值，来决定推荐结果的目的——为使用户有更多的行为来推荐较热门物品或给用户推荐冷门物品

缺点：

1. 无法在用户没物品产生行为就推荐物品给他
2. 一个物品如果没有用户行为，则无法将这个物品推荐
3. 如果物品很多，计算物品相似度代价很大
4. 推荐结果性能依赖K值，给选择合适的K值提出了挑战

8.2 未来展望

我们的实时推荐算法仍然有改进空间，比如未来可以通过优化代码逻辑来提升速度，也可以通过加入实时剪枝算法来提升性能。

我们仅仅对一个分布式实时推荐算法进行了实现和部署，而真正的生产上面需要的是多个推荐算法协同工作的推荐系统。我们仍然需要深入学习更多推荐算法，未来希望能实现多种分布式实时推荐算法，如 LMF/矩阵分解，并将多种算法整合，形成一个完整的推荐系统。

九、心得体会

在我的 SRP 项目刚开始的时候，学长向我们说明要做这个项目需要如何去学习如何去做的时候，一直用手机便签记录的我那是一头雾水。那一页的便签几乎全是当时的我听不懂的名词，什么“itemcf，slope one，卡夫卡，redis，spark streaming ”之类的词。我不仅不知道是什么，甚至连便签中记录的关键词有相当一部分是错误的文字。当时的我很迷茫，连怎么入手都不知道，只知道学长发来的几个文件夹的视频可以看。

什么都不知道的我一开始只能一步步地看视频跟着视频做。我跟着视频，开始学习 Spark 框架，入门居然要我用 Scala，Maven，并且在 Linux 上部署环境。而我不会 Maven，不会 Scala，我甚至连 Linux 的基本使用都不会。原本我以为只要照着视频中的做就可以成功，结果不同的电脑 Linux 系统配置也大相庭径，我踩了一个又一个的坑，我的 Linux 虚拟机还有一次差点被一个我甚至不知道功能的 rm -r 命令摧毁，幸好有备份。一塌糊涂的我知道了，要从基础学起。于是我开始自己学 Linux。从买了《鸟哥的 Linux 私房菜》开始，我慢慢地看，逐渐地可以部署 Linux 开发环境。我开始学 Python，我从一个 Python 小白，到熟悉使用数据分析类的工具包。我开始学 Scala，没有 Java 基础的我，入门 Scala 又是一塌糊涂，而我一步步地克服困难。Maven，Hadoop 集群，Linux 上的 MySQL，Git，我都一步步的地克服。虽然是从头学起，虽然基础并不扎实，但是学会的知识在后面发挥了无法替代的作用，以后的学习相比之前也越来越有方法。我学会查阅官方文档，我学会了查看源码来理解各种方法的用法，学会了很多以往都不会的学习方法。

我开始阅读有关推荐算法的书籍，查阅实时推荐算法的文献，开始对推荐算法有了逐步了解，而分布式计算思想仍不熟悉的我，第一次将算法分布式化就遇到了困难。在阅读了更多的文献后，我才逐渐对分布式计算有了进一步了解，我也能逐渐写出不同的分布式算法，并将推荐算法分布式化。

我开始学习流数据处理的框架和工具，flume，Kafka 我都手到擒来，慢慢地熟悉了流处理，并实现了实时的分布式推荐算法。

接着这个学期，我开始学习机器学习的知识，对先前阅读的有关机器学习的推荐算法开始有了清晰的了解，我的下一个目标就是认真学习机器学习知识，并将机器学习有关的推荐算法予以实现，并通过多个不同的分布式实时推荐算法整合成分布式实时推荐系统。

万事开头难，千里之行始于足下，就算你什么都不会，现在开始都不晚，只有不开始才是真怠惰。

十、参考文献

[1] 项亮：推荐系统实践

[2] Yanxiang Huang♯ Bin Cui♯ Wenyu Zhang§ Jie Jiang§ Ying Xu♯：TencentRec: Real-time Stream Recommendation in Practice

[3] Yanxiang Huang♯,§ Bin Cui♯ Jie Jiang§ Kunqiang Hong§ Wenyu Zhang§ Yiran Xie♯: Real-time Video Recommendation Exploration

[4] Xiangnan He Hanwang Zhang Min-Yen Kan Tat-Seng Chua ： Fast Matrix Factorization for Online Recommendation with Implicit Feedback

[5] http://spark.apache.org/docs

[6] https://www.cnblogs.com/langren1992/p/10142348.html

[7] https://blog.csdn.net/qq_40999403/article/details/101759558

[8] https://blog.csdn.net/suzyu12345/article/details/79673606

[9] https://blog.csdn.net/qq_40999403/article/details/97788531

[10] https://blog.csdn.net/woniu201411/article/details/81020932

[11] https://www.cnblogs.com/cssdongl/p/6203726.html

[12] https://blog.csdn.net/u010022051/article/details/50954533