论文｜看腾讯如何玩转实时推荐-TencentRec

原创 Thinkgamer 搜索与推荐Wiki 2020-03-11

收录于话题
#论文笔记

19个

今天要分享的一篇论文是有关腾讯如何利用协同过滤（Collaborative Filtering）、基于内容和图进行实时推荐的，我们都知道协同过滤是传统的推荐算法，但在实际应用中取得的效果却很好，因此在各大公司应用的也非常广泛。

协同过滤的改进经常出现了各种硕士研究生的毕业论文中，那都属于学术界的研究和实现，且是离线的，在工业界的CF算法实现和如何实时基于CF进行推荐的资料却是少之又少，接下来的内容将会带领你真正了解一下腾讯是如何在工业界进行实时的CF推荐。这里把论文中涉及算法和架构称之为 **TencentRec框架**

> 论文（TencentRec Real-time Stream Recommendation in Practice）获取，公众号内回复【TencentRec】获得。

背景

其实本篇文章的开篇也简单说明了TencentRec的背景，接下来再补充说明一下。

推荐系统在工业界应用的十分广泛，但在大数据背景下，如何将实时、准确度结合在一起却十分的难，因为这几者本身是互斥的。而腾讯技术团队则基于Storm，并设计研发了相关架构优化了ItemCF、ContentBased、Graph几种算法解决了推荐系统中的大数据、实时性、准确度的问题。

架构概述

平台选择

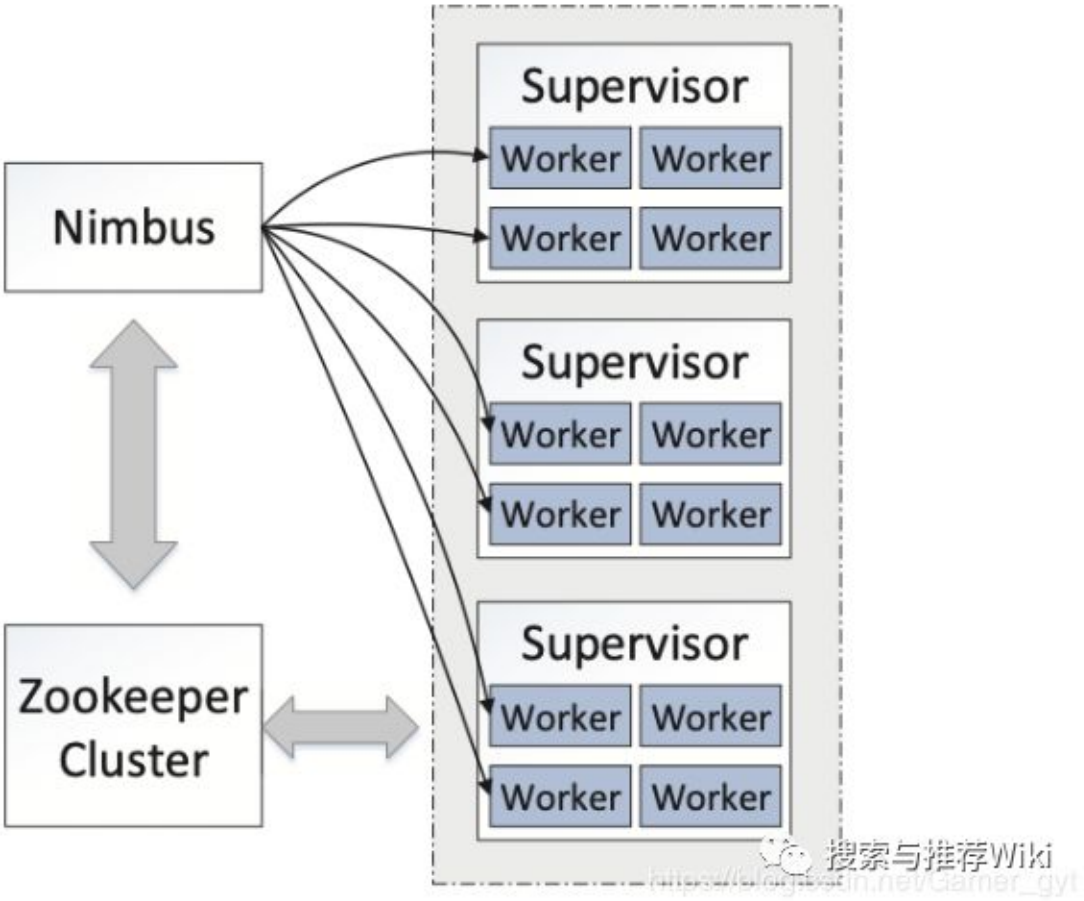
在腾讯内部，日活用户早已突破了1亿（2015年），每天产生的用户行为数据达到了PB级别，为了处理庞大的数据，一个分布式的数据计算集群是不可或缺的，对于计算平台的选型有三点要求：

- 能够进行实时计算
- 线性伸缩且可扩展
- 具有一定的容错能力

在调研了Yahoo S4，Spark Streaming和Storm之后，选择了Storm作为计算平台，主要是因为以下三点：

- Storm支持实时数据流计算
- Storm具有良好的可扩展性，可以动态增加或者删除集群中的节点
- Storm具有容错功能，可快速的进行故障恢复

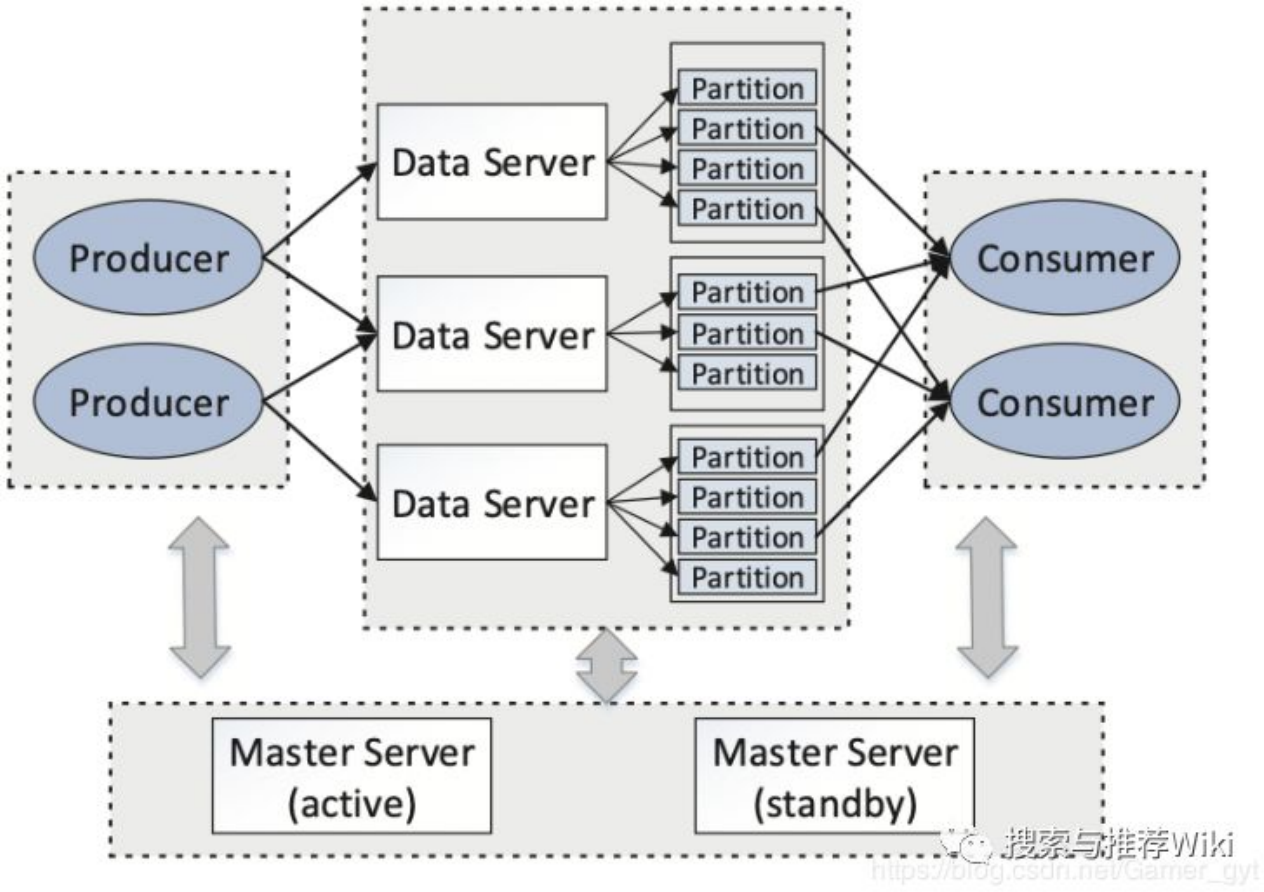
下图简单说明了Storm集群的架构



Structure of Storm Cluster

数据处理

TencentRec框架设计的目的是作为一个通用的推荐框架，并应用在各种应用中。因此一个关键的因素是如何处理不同应用的非格式化的庞大的数据，因此研发了一个Tencent Data Acess模块，提供数据收集和分发的接口，并把数据源和数据处理系统进行解耦。



The Architecture of TDAccess

上图展示了TDA（Tencent Data Acess）的架构，将发布/订阅模型作为通信模型。生产者是各个应用，消费者是数据处理系统。数据服务器（Data Server）负责数据缓存、数据的发布和订阅、生产者和消费者之间的通信。

上图展示了两个Master服务器（Active、Standby），Standy是备用服务器，当Active的服务器出现宕机或者故障的时候能够代替，从而保证系统的稳定性和可用性，这一点的设计和Yarn集群是类似的。

设计和实现TDAccess的关键因素如下：

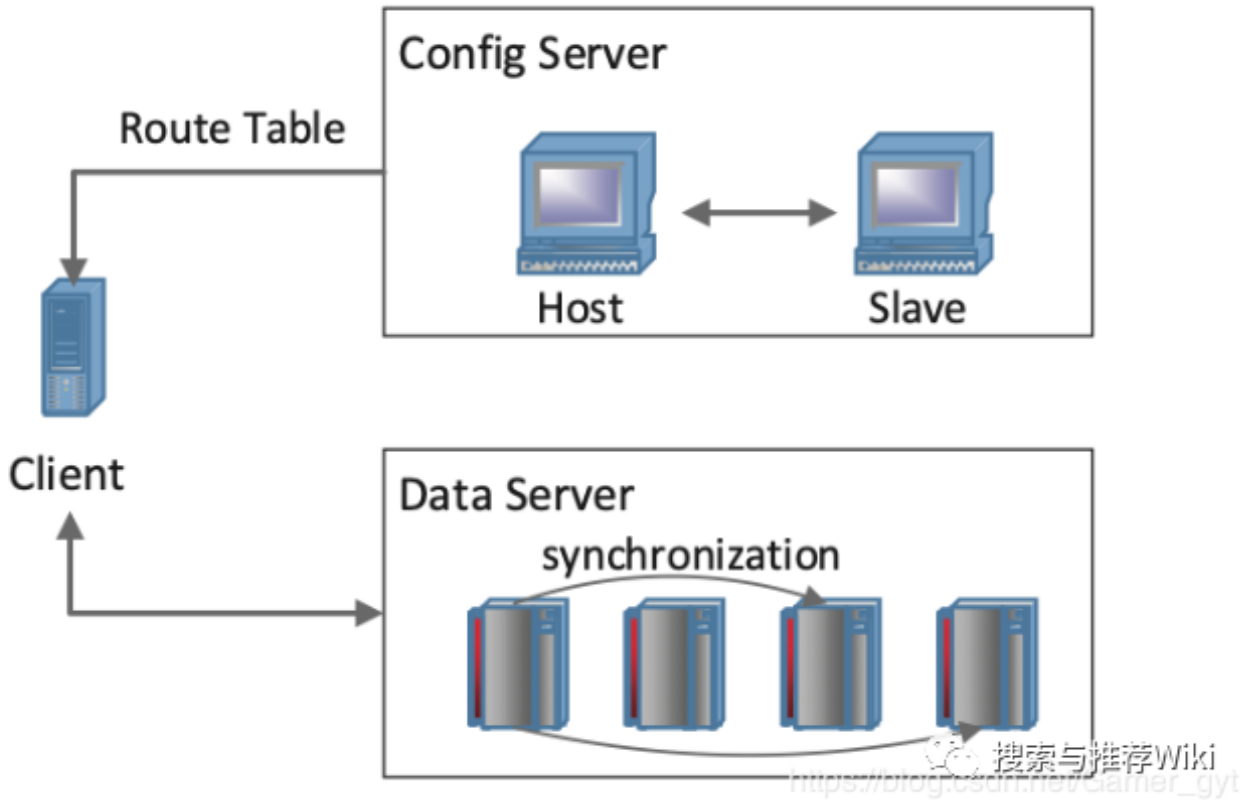
- 数据服务器不需要共享由主服务器管理的数据状态
- 在缺少计算资源或者需要历史数据进行离线计算的情况下会把数据缓存在硬盘上
- 为了实现更好的并行，将数据缓存在机器上的许多分区上

状态数据存储

Storm由于是一个无状态的系统，因此具有强大的缓存能力，但是推荐系统需要保持记录的历史数据的状态，最直接的解决办法是将状态数据和程序一起存储在程序中，但是这将导致以下问题：

- 限制了程序的设计。不同的工作人员需要不同的状态数据，因此需要在程序中管理状态数据的分配和转换，同时需要保证数据的一致性。
- 降低系统的健壮性。如果机器无法保证状态数据将会导致错误的计算结果。

为了解决这些问题，设计了基于内存的分布式键值数据存储模块（Tencent Data Store， TDS）用来保存推荐计算中使用的状态数据。通过这种方式，保证了TencentRec的鲁棒性，Storm保证程序的运行，TDS保证状态数据的恢复。



The Framework of TDStore

上图展示了TDS的架构，图中分为两部分：

- 配置服务器：由主机配置服务器和备份配置服务器组成，它们管理路由表并跟踪数据服务器。
- 数据服务器：负责数据存储，支持内存数据库（MDB），级别数据库（LDB），Redis数据库（RDB）和文件数据库（FDB）存储引擎。每个实例数据都有多个备份。

算法设计

为了实现可扩展的推荐，实时推荐系统中常用的方式是进行数据采样，以此满足在有限的计算资源内产生推荐，但会降低结果的准确性。因此将所有数据都应用在计算中则是必须的。出于这样的考虑，将传统的推荐算法进行了分布式实现。

ItemCF实现

由于篇幅的限制，论文中只介绍了ItemCF的详细实现。首先回顾一下CF的思想，CF算法包括：

- UserCF：首先计算出用户的相似用户，再把相似用户有行为的物品推荐给用户
- ItemCF：首先计算出物品的相似物品，再把用户有行为物品的相似推荐推荐给用户

为了在实践中使用ItemCF算法，同时满足“大数据”、“准确度”、“实时性”三方面的要求，主要考虑以下三方面的优化：

- 隐式反馈优化问题
- 可扩展的增量更新机制
- 实时修剪技术

传统的ItemCF

传统的ItemCF算法分为：物品的相似度计算和产生推荐两步。具体的算法逻辑这里就不解释了，感兴趣的可以阅读：https://blog.csdn.net/Gamer_gyt/article/details/51346159

物品的相似度计算公式如下（公式1.1）：

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

预测用户对物品的评分如下（公式1.2）：

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

隐式反馈优化

精准的推荐算法通常需要根据用户对物品的评分来捕获用户对物品的偏好，但是在实际情况下，并非总是可以获取明显的反馈。大部分都是隐式反馈，即间接反应用户对物品的喜好程度。

在腾讯内部的场景中，用户对物品的行为包括点击，浏览，购买，共享，评论等。不同的用户行为代表着不同程度的用户兴趣，并且应该对推荐算法产生不同的影响。因此，我们为不同的动作类型设置了不同的权重（例如，点击行为计作一分，购买行为计作三分）。**如果一个用户对一个物品表现出了多种行为，我们将最大权重的行为权重值作为用户对物品的评分，这样可以减小各种凌乱的隐式反馈带来的噪音。**

用户对物品i_p, i_q的协同评分被定义为用户对物品评分中的最小权重值，如下公式所示（公式1.3）

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

其中$r_{u,p}$表示用户u对物品i_p的行为中的最大权重值。

则物品i_p, i_q的相似度被重新定义为如下公式（公式1.4）：

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

可扩展的增量更新机制

在传统的推荐系统中，协同过滤算法会采用定期更新的机制，但这种策略并不难满足实时推荐系统的需求，因为在TencentRec中设计了可扩展的增量更新机制。

在论文【Scalable collaborative filtering using incremental update and local link prediction】中，作者提出了一种用于连续显式评级的基于增量项的CF方法，受这种增量更新机制的启发，TencentRec在实践中将相似度的计算分为三个部分。

首先物品i_p, i_q的相似度定义为（公式1.5）：

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

其中：

$$itemCount(i_p) = \sum_{u \in U} r_{u,p} pairCount(i_p, i_q) = \sum_{u \in U} co\_rating(i_p, i_q)$$

> 公式1.5 和 公式1.4的定义是一样的。

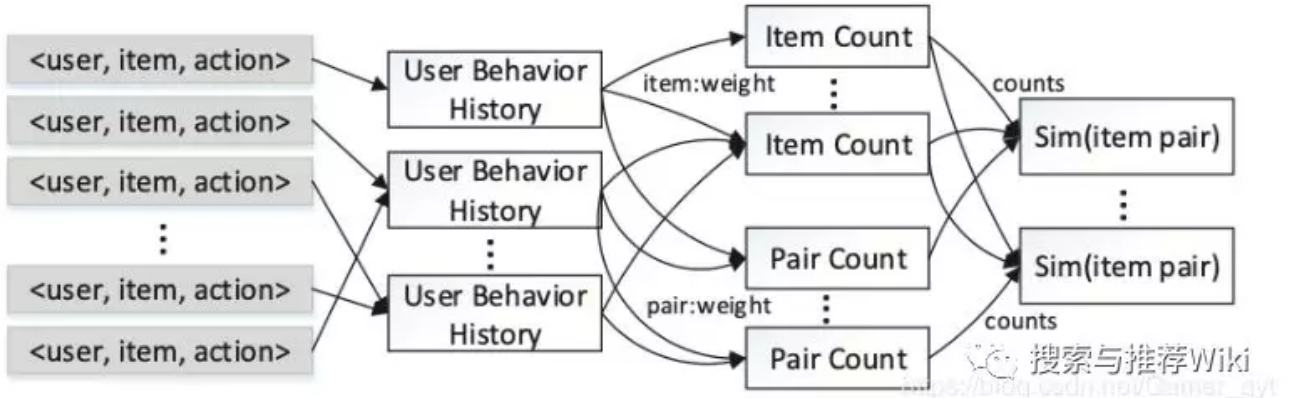
在公式1.5中，物品的相似度可以通过 $pairount(i_p, i_q)$ 、 $itemCount(i_p)$ 、 ite进行计算，而这三项也可以很自然的进行更新。我们可以独立的计算这三项的新增值，并将这些值融合到公式1.5中重新计算相似度。

计算的表达式如下（公式1.6）：

$$sim(i_p, i_q)' = \frac{pairCount(i_p, i_q)'}{\sqrt{itemCount(i_p)'}\sqrt{itemCount(i_q)'}} = \frac{pairCount(i_p, i_q) + \Delta co\_rating(i_p, i_q)}{\sqrt{itemCount(i_p) + \Delta r_{u,p}}\sqrt{itemCount(i_q) +}$$

其中$sim(i_p, i_q)'$表示新的物品间相似度。

整个实时更新的计算流程图如下所示：



The Multi-layer Item-based CF

实时修剪技术

在ItemCF算法中，如果用户倾向于对两个物品进行评分，则将其视为相关。这里设置了链接时间，即在链接时间内有行为的两个物品才视为相关。在腾讯内部，每天有很多用户物品的行为记录，以新闻为例，平均每个用户每天会对十个以上的新闻产生行为，如果把新闻之间的链接时间设置为6个小时，则对出现在6个小时内的新闻构建物品对。

对于大多数情况，比如电商网站，链接时间通常设置为3天或者7天，每个用户操作会生成近一百个物品对，此时，每个用户的操作都会导致数百次计算，由于大部分物品对并不相似或者没有那么相似，因此大多数操作是没有必要的，会浪费大量的计算资源。

为了解决这个问题，TencentRec基于霍夫边界（Hoeffding Bound）理论开发一种实时修剪技术。其表达为：x表示 实数R范围内的一个实数值（在相似度中范围是0-1），假设对该变量进行了n次独立观察，并计算出了他们的均值$\hat{x}$，霍夫边界以$1-\delta$的概率认为该变量的真实均值最大为：$\hat{x}+\epsilon$，其中ϵ为（公式1.7）：

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

其中δ需要自己进行设定，在实践中$\hat{x}$用t代替。

ItemCF的实时修剪算法如下：

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

数据稀疏性问题

在真实的实践中，数据往往是非常稀疏的，在进行计算时，会消耗大量的计算资源，也会影响推荐系统的效果。TencentRec采用了两种机制来解决数据的稀疏性问题：

- 人口统计聚类：根据用户的属性（例如年龄、性别、学历）将其分为不同的组。通常认为不同的组内具有相同的兴趣偏好。在不同的组内运行推荐算法，往往能获得更加准确的结果。

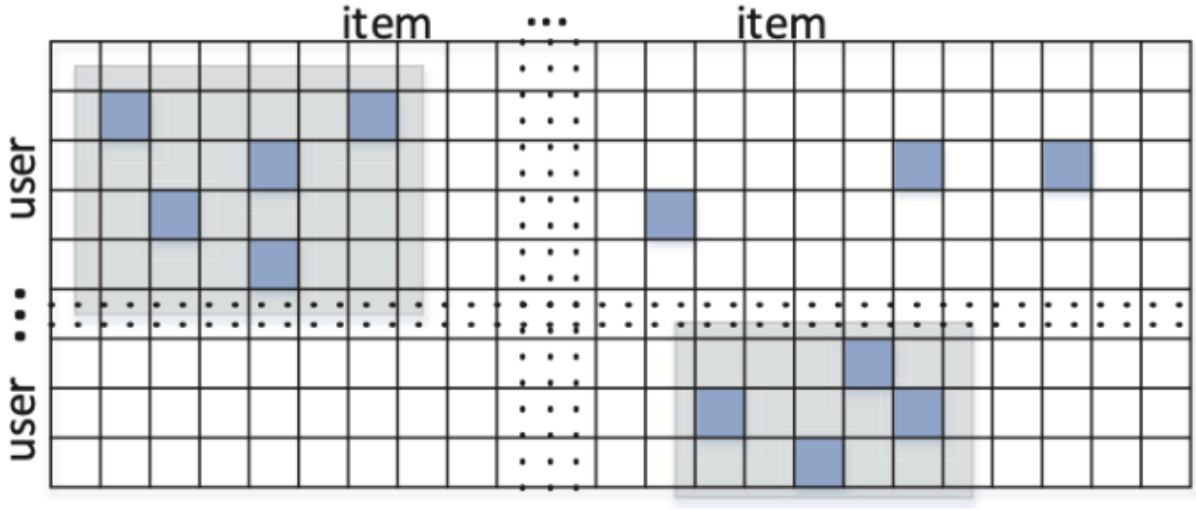


Figure 5: User-item Matrix in Big Data Era

User-item Matrix in Big Data Era

- 基于人口统计学的补充：对于历史行为数据很少的用户，TencentRec提出了基于人口统计学的算法以进一步完善推荐结果，旨在根据用户群向用户进行推荐。具体来说，我们会计算每个用户所属组的热门物品，即该组中最受欢迎的物品推荐给用户。

实时过滤机制

在类似TencentRec的实时推荐系统中，推荐结果会不断的发生变化，我们利用两种技术来捕获用户的实时兴趣，包括：滑动窗口和实时个性化过滤。使用滑动窗口技术捕获全局实时趋势，并使用实时个性化过滤来满足个人用户的实时需求。

滑动窗口技术是实时数据计算流中用于丢弃旧数据的一种常用技术，滑动窗口处理当前时间窗口中的数据，然后丢弃一小部分，继续处理接下来的一部分数据。在TencentRec中将时间窗口划分为几个会话，并且在相似度计算中只考虑了最近的W个会话，例如在滑动窗口内计算实时ItemCF算法中的$pairCount$ 、 $itemCount$，计算公式如下（公式1.8）：

$$sim(i_p, i_q) = \frac{\sum_{w \in W} pairCount_w(i_p, i_q)}{\sqrt{\sum_{w \in W} itemCount_w(i_p)} \sqrt{\sum_{w \in W} itemCount_w(i_q)}}$$

通过这种方式，用户可以自定义滑动窗口的大小和会话个数。

除了滑动窗口技术之外，还提出了一种实时的个性化过滤技术来满足不同用户的个性化需求。对于每个用户，记录他最感兴趣的k个物品。因为随着时间的流逝，我们认为用户只对最近的k个物品感兴趣。因此在考虑用户对未知物品的评分时，只考虑有行为的最近k个项目即可。

在ItemCF算法中，如果计算的物品对之间相似度过低时，可以采用基于组内热门数据补充的方法进行补充。

实现细节

实现细节这里就不着重说了，但需要注意的有两点：

- 行为爆发：在某些特殊的时间节点，用户对物品的行为数据要比以往更高。具体的解决办法是，采用流分组技术将相同的key映射到一个机器上，确保缓存的有效性，如果有其他程序需要读取数据，首先从缓存中读取，然后再进行缓存和TDS中的数据更新。
- 热门物品：采用组合器技术来解决热门物品的问题。即对相同健值对的记录进行部分合并，合并的操作包括：增量、加法、最大化。

当然这里还提到了一些其他的优化技巧，比如多重哈希技术，即保证同一用户的操作分配给一个机器，继而保证数据更新的准确性。

实践效果

TencentRec在自己内部的很多产品上进行了上线测试，涉及的产品有：

Figure 8: Some Applications Applying TencentRec

为了评估TencentRec的效果，采用了ABTest，其base组是：离线计算或者近实时计算，没有实时过滤机制，而实验组的话就是论文中主要描述的TencentRec，经过一个月的实验，对表效果如下：

Table 1: Overall Performance Improvement

	Algorithms	Performance Improvement (%)		
Applications		avg	min	max
News	CB	6.62	3.22	14.5
Videos	CF	18.17	7.27	30.52
YiXun	CF	9.23	2.53	16.21
QQ	CTR	10.01	1.75	25.1

Overall Performance Improvement

在新闻网站上实验一周后的数据效果如下：

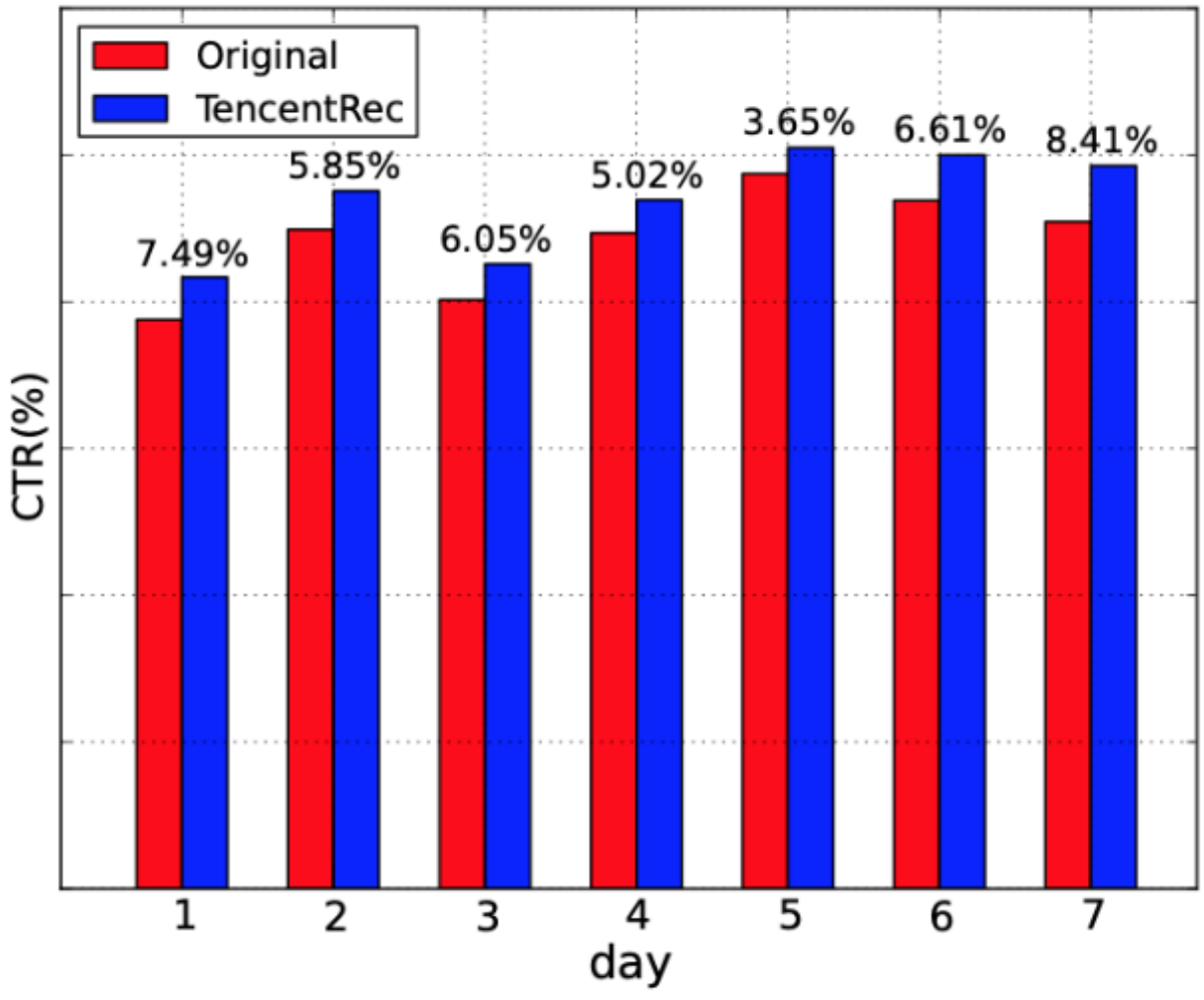


Figure 10: CTR of Tencent News in One Week

在易迅网上的两个推荐位进行对比的结果如下：

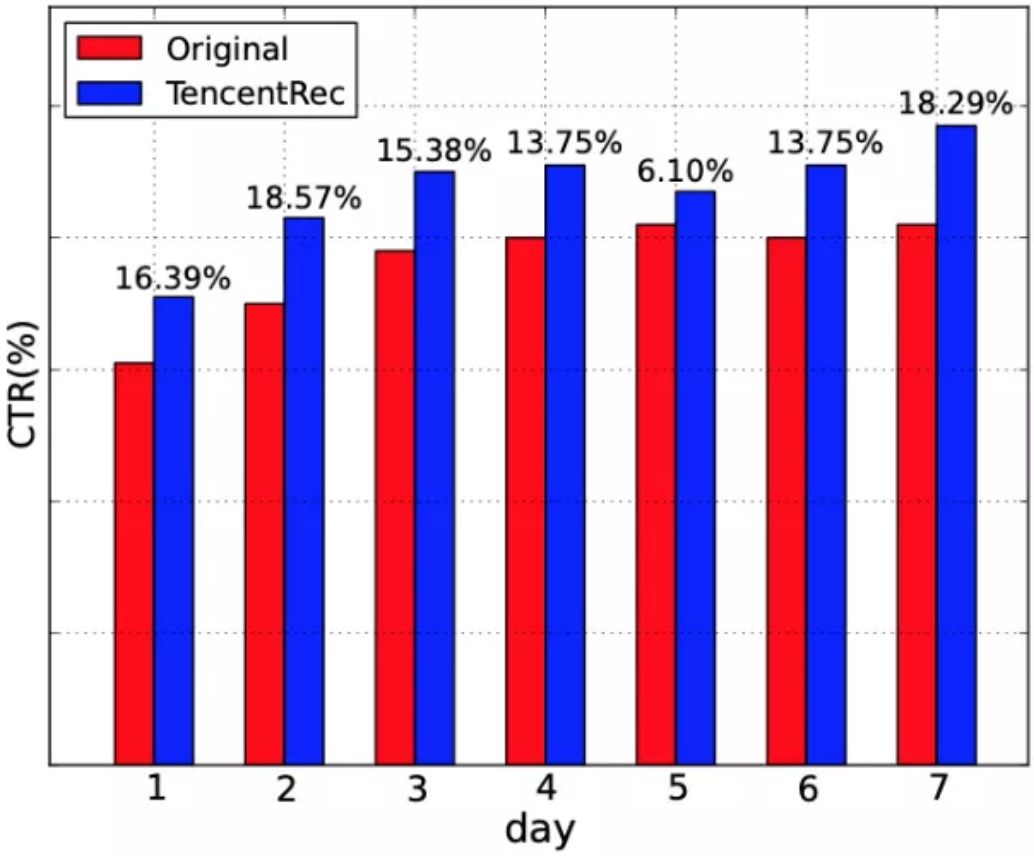


Figure 13: CTR of Similar Price Recommendation in YiXun

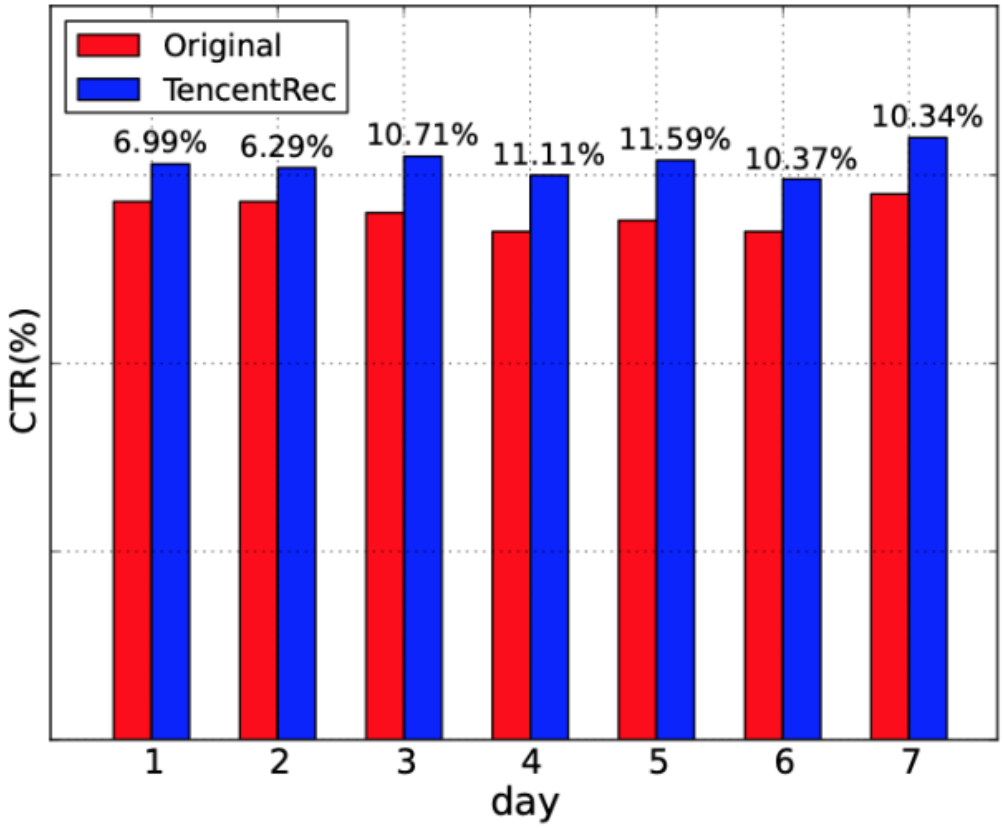


Figure 14: CTR of Similar Purchase Recommendation in YiXun

CTR of Similar Purchase Recommendation in YiXun

好了，到这里这篇论文的分享已经完了，通过这篇论文我们可以学到很多思路，可以应用到我们实际的生产环境中，当然对于离线的ItemCF算法Spark中也有实现-ALS算法，感兴趣的可以阅读这两篇文章：

- 终于搞明白了什么是交替最小二乘法(ALS算法)
- Spark排序算法系列之ALS算法实现分析

如果你觉得这篇文章不错，请分享给更多人！

原创不易，点个“在看”鼓励鼓励吧

收录于话题 #论文笔记·19个

上一篇
论文｜LinUCB论文的思想解读、场景应用与痛点说明

下一篇
论文｜组推荐系统及其应用研究

阅读原文