# 面向大数据的高效关联规则推荐算法技术报告

## 1. 并行化设计思路和方法

(1) 设计思路

本算法设计时的基本思路为，由于Aprori等算法复杂度无法满足要求，选用FP-Growth算法实现频繁项集挖掘，并在Spark平台上实现其分布式版本（PFP算法）。在得到频繁项集后，关联规则应当与用户进行匹配，为加速这一过程，设计了一棵由所有可能的关联规则构成的规则树，对于每个用户在规则树上并行查找。由于用户间天然并行，设计思路非常清晰。

(2) 具体算法

整个算法的大致流程如下：

1. 选用PFP-Growth算法生成频繁项集

2. 根据频繁项集产生规则树，并将其广播到各个分区

3. 对每个用户，在规则树上查找得到最大的置信度和推荐右项

(3) 优化技巧

首先，FP-Growth算法可以通过并行建树和并行挖掘的方式加速频繁项集的寻找过程。由于用户集和规则集都较大，一一匹配会导致时间复杂度过高的问题，我们用一棵预先生成的关联规则树进行匹配，并进行可行性剪枝和最优性剪枝。用户通过分区的方式，并行地在关联规则树上匹配，并行度很高。

## 2. 详细算法设计与实现

(1) 实现方案

1. 利用FP-Growth算法，利用合理的分区策略，并行地生成一棵FP-Tree，然后并行地进行频繁项集的挖掘，基本按照PFP-Growth算法所描述的过程，不再赘述。
2. 根据FP-Growth算法的两步，生成所有的频繁项和频繁项集，用频繁项集中的每个元素生成相应的关联规则，置信度计算过程用Map查找，以起到加速的效果。
3. 生成所有的规则后，根据规则生成一棵类似于FPTree的RulesTree，其中包含Node与Leaf两种节点。Node节点是RulesTree的骨架，是由规则前项按照FPTree的格式生成的，包含key信息、子树列表children以及以其为根的树所包含的最大置信度max\_conf。Leaf节点包含推荐项以及置信度，从根到Leaf的路径正好是这个Leaf节点所对应的规则前项。创建子树时候使用串行插入的方法。插入规则时，在RulesTree上沿着 一路向下找，如果没有对应节点就创建一个插入到子树里。在查找过程中，更新Node中max\_conf的值，并保证子树列表的Node的max\_conf降序排列（Leaf永远在最前面）。到达后，插入Leaf(b, conf)即可。

4.RulesTree构建完成后，可以使用DFS检索出对于某个用户置信度最大的商品。到达一个Node节点时，首先看它对应的key是否在用户购物篮中，如果不在，则这个节点所对应的子树都是无效规则，全部剪枝。其次，再看已经发现的最大置信度found\_conf是否大于子树的最大置信度max\_conf，如果小于，则不可能发现更高置信度的商品，也全部剪枝，否则开始递归DFS。检索到Leaf节点时，检查其是否已经在购物篮中，在则不能推荐, 否则就比较当前推荐商品和已经发现的最大置信度商品进行对比，如果当前商品更佳就进行更新。

(2) 细节优化

1. 程序中使用的树采用immutable的数据结构，大部分采用函数式的方式书写，增加了程序的可读性和便于调试性。
2. 程序采用了KryoSerializer，在FP-Tree并行构建的过程中起到了重要优化作用。
3. 对重要的RDD进行cache，避免无意义的重复计算。
4. 优化Shuffle过程。

## 实验结果与分析

A过程（频繁项集挖掘过程），所用时间在不出现节点崩溃问题的情况下为2.7h。

其余过程（关联规则推荐过程），所用时间为3min。

理论上该算法的运行速度远远大于串行算法。仅关联规则推荐一项，约200万的关联规则和30万的用户购物篮，3分钟即完成了全部的推荐过程。

频繁项集的挖掘过程没有明显的对比。

## 4. 程序代码说明

(1) 程序代码结构

src

└── main

└── scala

├── FPGrowth.scala FP-Growth算法的实现

├── FPTree.scala FP-Growth算法中FP-Tree的实现

├── Main.scala 控制程序主流程与输入输出

└── RulesTree.scala Rules-Tree的实现

(2) 程序代码的编译方法

本程序使用sbt进行编译。在根目录（即src文件夹所在的目录）下直接执行sbt package，可执行文件会产生在target/scala-2.11/。

(3) 运行使用方法

程序共有三个参数，第一个参数为输入文件路径，第二个参数为输出文件路径，第三个参数为临时工作路径。

## 5. 参考文献

1. FP-growth 的分布式实现：<http://spark.apache.org/mllib/>