关联规则推荐的高效分布式计算框架

李昌盛[1)] 伍之昂[1),2)] 张 璐[1),2)] 曹 杰[1),2)]

[1)](南京财经大学信息工程学院 南京 210003)

[2)](南京财经大学江苏省电子商务重点实验室 南京 210003)

摘 要 关联规则推荐模型是在电子商务网站应用最广泛的商用推荐引擎之一,目前已有的工作大多聚焦于如何挑选高质量规则,以提升推荐精度.然而,关联规则数量庞大,且用户并发访问量通常极大,如何快速匹配用户浏览记录和关联规则库,为海量在线用户产生近实时推荐,成为制约关联规则推荐能否胜任真实电子商务网站推荐的重要因素.为此,本文研究关联规则推荐的效率问题,提出服务于高效关联规则推荐的分布式计算框架,将规则挖掘与推荐计算无缝衔接.具体而言,本文首先设计有序模式森林,用于压缩存储频繁模式;然后将候选规则挖掘转化为森林上的路径搜索计算,并提出高效的单机路径搜索算法;最后提出负载均衡的数据分割策略,同时降低分布式规则挖掘与推荐计算中的任务最迟完成时间.在 3 个公开数据集的实验结果表明基于有序模式森林的推荐计算比传统穷举匹配策略降低 6 倍以上时间,同时所提出的分布式计算框架可随计算节点数量达到近线性扩展.

关键词 推荐系统;关联规则;频繁模式;FP-growth 算法;Spark;负载均衡

中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2019.01218

An Efficient Distributed-Computing Framework for Association-Rule-Based Recommendation

LI Chang-Sheng[1)] WU Zhi-Ang[1),2)] ZHANG Lu[1),2)] CAO Jie[1),2)]

[1)](*School of Information Engineering, Nanjing University of Finance and Economics, Nanjing* 210003)

[2)](*Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing* 210003)

Abstract The association rule based recommendation model is one of the most widely used commercial recommendation engines in e-commerce websites. A variety of techniques, mainly including eligible rule selection and multiple rules combination, have been developed to create effective recommendation. Unfortunately, the efficiency of the association rule based recommendation has been paid for little attention. In real-life online shopping sites, the concurrency traffic is usually very high, that is, there are a vast amount of users visiting sites simultaneously and persistently adding commodities into their baskets. In the meanwhile, the volume of frequent patterns are usually very large and thus the number of association rules derived from these patterns is much larger because a pattern is able to generate several rules. As the large amount of the association rules and the concurrent access of users, how to match the browsing histories of users with the large set of rules efficiently in order to offer nearly real-time recommendations for massive online users, has become a vital concern which restricts whether the association rule based recommendation model could be used on the real-life e-commerce websites. To address this problem, this paper focuses on the efficiency of the association rule based recommendation and develops a distributed computing framework for improving the computational performance of rule based recommendation,

收稿日期:2018-01-15;在线出版日期:2018-12-24.本课题得到国家自然科学基金项目(71571093,91646204,71801123)资助.**李昌盛**,硕士,主要研究方向为数据挖掘和推荐系统.E-mail:lichshe@hotmail.com.**伍之昂**(通信作者),博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为数据挖掘和推荐系统.E-mail:zawuster@gmail.com.**张 璐**,博士,讲师,主要研究方向为数据挖掘.**曹 杰**,博士,教授,主要研究领域为数据挖掘和商务智能.

which seamlessly fuses the association rule mining and the recommendation computing. Firstly, based on the summarization of existing rule-based approaches, a tree-type structure called Ordered Patterns Forest (OPF) is designed for the compact representation and storage of frequent patterns, without missing any basic information that will be useful for the subsequent recommendation such as support of a pattern and its nested patterns. Secondly, we transform the two-step rule based recommendation to a series of operations on the data structure, and then develop the corresponding efficient algorithms for these operations which are responsible for mining eligible patterns as well as computing recommendation scores. Specifically, we transform the candidate rules mining into a path searching problem on the OPF and thus present a path searching algorithm running on the single machine. Finally, the real-time recommendation is impossible to be completed in a single machine. Hence, in order to handle the ever-increasing of online customers and patterns, we devise a distributed computing framework in which a novel load balanced strategy for data partitioning is also proposed to reduce the running time of the task that finishes lastly and thus further improves the overall performance. At last, we implement the proposed framework and algorithms on Spark, a widely used memory-based distributed computing engine, and evaluate the framework and algorithms on three real-world datasets, i.e., Accidents, Webdocs and Amazon. The experimental results demonstrate that the efficiency improved by the proposed OPF with the path searching algorithm is more than six times that of the traditional brute force method. Moreover, the proposed load balanced strategy is effective to further improve the performance of the proposed distributed association rule based recommendation framework, which can achieve nearly linear scalability along with the increase of computational nodes.

Keywords　recommender systems; association rules; frequent patterns; FP-growth; Spark; load balancing

1　引　言

推荐系统在近十年来受到学术界的充分重视和深入研究，并在电子商务网站、社交平台及视频音乐网站上得到广泛应用[1-2]. 各种各样的推荐模型和方法相继被提出，主要包含基于内容的方法、协同过滤模型、混合式模型、基于关联规则的方法等[3-4]. 关联规则最早应用于购物篮分析，可揭示一组经常被一起购买的商品，因此自然而然地成为一种简明且可解释性极佳的推荐模型. 例如，由于集合{单反相机，单反镜头，三脚架}中的商品经常一起被购买，当用户购买或频繁浏览“单反相机”和“单反镜头”时，就将“三脚架”推荐给用户. 基于关联规则的推荐本质上利用了项目(item)之间的关联性，其机理类似于基于项目的协同过滤模型(Item-based Collaborative Filtering)[5]，因而很多文献如[3,6]将基于关联规则的推荐归类于协同过滤模型.

关联规则推荐是一类非常流行的推荐方法，大量电子商务网站将基于关联规则的方法作为其商用推荐引擎，比如：YouTube 使用关联规则推荐视频[7]、淘宝和亚马逊网站上的“购买此商品的顾客也同时购买”及“经常一起购买的商品”等推荐方式也依托于关联规则构建. 关联规则推荐能获得大量实际应用的原因是：(1) 推荐结果可由用户操作动态触发，即用户浏览或购买记录发生变化时，经过与关联规则库的快速匹配，推荐结果可以快速更新；(2) 关联规则能揭示推荐结果与浏览或已购买商品之间的共现关联，这使得推荐结果具有良好的可解释性.

由于频繁模式的反单调性，一个频繁模式包含很多频繁子模式，而一个频繁模式也能导出多个关联规则，因此关联规则数量巨大、且多个规则蕴含同个目标项目的情况广泛存在. 为了提升推荐准确率，大量研究工作围绕如何挑选高质量规则展开. 早期的研究大多挑选置信度(confidence)最高的规则来推荐[8-10]，文献[11]挑选最长规则，文献[12]则提出综合利用置信度、支持度(support)和长度对规则进行排序. 近年来，一些新指标能进一步提升关联规则

推荐的性能，比如校正置信度(adjusted confidence)[13]、分离置信度(disjunctive confidence)[14]、互信息[15]等.还有一些研究认为需融合与目标项目相同的规则集以此来得到综合推荐分值，文献[16-17]将多个规则的统计量相加获得推荐分值，文献[18]引入D-S证据理论合成多规则的分段支持度值(partitioned support)作为推荐分值.上述研究工作的本质区别在于推荐分值计算方式不同，而候选规则集的匹配过程是相同的，设有规则"{单反相机，单反镜头}→三脚架"，若其前项{单反相机，单反镜头}包含用户的浏览或购买记录，但结果项{三脚架}却不包含，则该规则就是此用户的一条候选规则，结果项为待推荐的目标项目.

在实际电子商务网站上，用户并发访问量极高，据统计，淘宝并发在线用户经常高达千万[19]，而关联规则数量又十分庞大.如何为如此大量在线用户搜索候选规则的计算效率，成为制约关联规则推荐实际应用的瓶颈问题，尤其是当用户浏览和购买记录动态变化，推荐结果需实时生成的情形.但是，大部分已有研究主要关注关联规则推荐的准确性及关联规则的挖掘效率，而面向大规模在线用户和规则的匹配计算效率却未曾得到关注.本文试图提出面向关联规则推荐的可扩展性计算框架，能无缝兼容已有研究所提出的推荐分值计算方法，缓解关联规则推荐面临的大数据挑战.

本文第2节总结相关工作；第3节正式定义问题并总体介绍面向关联规则推荐的可扩展分布式框架；第4节针对分布式框架中各个模块的展开详细介绍；第5节为实验结果和分析；第6节总结全文.

2 相关工作

本文将提出分布式计算框架旨在将规则挖掘与推荐计算两个阶段无缝衔接，并能支撑现有的推荐分值计算方法.因此，本节将从关联规则挖掘和基于关联规则推荐两个方面回顾相关工作.

2.1 关联规则挖掘

关联规则由频繁模式生成，频繁模式挖掘的核心是提升计算效率，所有挖掘方法均基于反单调性质对格空间进行剪枝，已有挖掘方法的区别仅在于格空间的遍历次序以及原始数据的组织方式.Agrawal等人[20]基于广度优先遍历提出第一个频繁模式挖掘算法Apriori，随后Han等人[21]利用树型结构组织原始数据并基于深度优先遍历提出FP-growth算法，Zaki等人[22]利用垂直方式组织原始数据、同样基于深度优先遍历提出Eclat算法.上述三种算法成为频繁模式挖掘领域公认的经典方法.

随着数据量的增大，频繁模式挖掘算法的可扩展性日益突出，大量研究围绕如何将Apriori、FP-growth和Eclat算法的分布式化展开.由于Apriori和Eclat算法均需要从k项集生成$k+1$项集，即本轮计算依赖于上一轮计算的结果，因此其分布式机制需要在共享内存架构中实现[23].而在Han等人[21]提出FP-growth时就指出可以按每个项目形成投影数据集，将原始数据集划分成独立的若干个投影数据集，从而可将FP-growth挖掘分解成若干独立的子任务.Grahne等人[24]将单项投影扩展至组投影，有效控制了独立子任务的数量，成为FP-growth分布式实现中数据逻辑分割的核心技术.尽管Spark内存计算模式的出现，能极大地提升分布式Apriori和Eclat算法的效率，但是FP-growth依然是最适合分布式化的算法，这也是本文在挖掘频繁模式时选择FP-growth作为基准方法的原因.

基于组投影技术，文献[25]在Hadoop平台上基于MapReduce提出了FP-growth分布式版本PFP，但是任务间的负载均衡问题未得到考虑，而是简单地将FList分割成均等长度的组.MLib库[26]沿用PFP的思路，提供了Spark环境下的开源FP-growth实现.Zhou等人[27]注意到负载均衡对分布式FP-growth算法性能的重要影响，提出用项目在FList中排序位置的对数值衡量挖掘项目的负载，缓解了均等分割法导致的负载极度不均衡问题.本文的FP-growth分布式方案依然采用经典的组投影思路[24]，但对FList的负载均衡分割方案做了进一步优化，提出投影数据集规模的估测指标，并以此作为FList的划分依据，既汲取了分布式FP-growth优势，又提升了现有分布式FP-growth的性能.在实验部分将给出本文分布式FP-growth算法与分布式Apriori和Eclat算法以及采取其他不同负载均衡策略的分布式FP-growth算法之间的性能比较结果.

值得一提地是，最近一些研究尝试从其他角度提升频繁模式的挖掘效率，比如Chon等人[28-29]的研究利用BitMap技术压缩表示数据，以提升Apriori候选项集生成和支持度计数的速度，并用GPU编程

进一步提升 BitMap 计算速度；Song 等人[30]则提出增量频繁模式挖掘方法，并在 Hadoop 上将增量挖掘方法并行化.

2.2 基于关联规则推荐

围绕高质量关联规则选择以及推荐分值计算方法，国内外学者展开了广泛研究，以期提升基于关联规则推荐的准确率. 除了以置信度作为挑选高质量关联规则的依据[8-10]，Rudin 等人[13]将校正置信度作为选取关联规则的依据，Ghoshal 等人[14-15]相继提出分离置信度和互信息两个指标. Li 等人[12]则提出多规则排序方法选取关联规则，依次优先考虑置信度、支持度和前项长度. 针对蕴含同个目标项目的多个关联规则，很多研究提出将这些关联规则的统计量融合以获得合理的推荐分值，Lin 等人[16]认为待推荐项目的推荐分值应该由多条以此项目为结果的关联规则的支持度和置信度乘积之和来确定. Wang 等人[17]在计算推荐分值时，将具有相关结果的关联规则的余弦值进行求和. Wickramaratna 等人[18]引入 D-S 证据理论合成多规则的分段支持度值作为推荐分值. 此外，文献[31]对如何选择关联规则推荐以提升冷门商品的覆盖率做了研究.

已有的研究均认为制约基于关联规则的推荐在于频繁模式的挖掘阶段，例如文献[25]利用关联规则进行查询推荐，但是仅考虑了关联规则的快速挖掘问题. 因此，关联规则挖掘出来之后的推荐计算（即匹配用户记录与关联规则集合以获得候选规则集合）效率尚未获得充分的重视，而这恰恰是在针对大规模在线用户时实现近实时推荐的关键. 本文试图将关联规则挖掘和推荐计算两阶段无缝衔接，提出一个面向关联规则推荐的可扩展分布式总体框架. 同时，本文提出的计算框架具有优良的通用性，能支撑已有的不同推荐分值计算方案，这将在 4.3 节中进行讨论.

3 问题描述和总体框架

3.1 关联规则推荐的可扩展性问题

假设 $I=\{i_1,i_2,\cdots,i_m\}$ 是 m 个不同项目的集合，事务数据集 D 表示所有用户的历史浏览或购买记录. 在关联分析中，包含多个项目的集合称为项集或模式，项集 X 的支持度为包含 X 的事务个数占 D 中事务的百分比，用 $supp(X)$ 表示. 给定支持度阈值 $minisupp$，在数据集 D 上挖掘获得的频繁模式集记为 $P=\{p_1,p_2,\cdots,p_s\}$，其中任意一条频繁模式 $p_j=\{i_{j1},i_{j2},\cdots,i_{j|p_j|}\}$，$|p_j|$ 表示频繁模式 p_j 中包含的项目的个数. 令 $R_{jk}:A_{jk}\rightarrow i_{jk}$ 为频繁模式 p_j 产生的一条关联规则，A_{jk} 是关联规则前项，i_{jk} 是关联规则后项，其中 $A_{jk}=p_j/\{i_{jk}\}$. 在推荐中，规则后项仅包含一个项目，因此频繁模式 p_j 可以产生至多 $|p_j|$ 个能用于推荐的关联规则. 令 T_u 为在线用户 u 的当前浏览或购买记录，T_u 的候选规则集合定义如下.

定义 1. 候选规则集合. 给定用户 u 当前浏览或购买的记录 T_u 和关联规则集合 R，用户 u 的候选规则集合定义为：$R_u=\{R_k:A_k\rightarrow i_k \mid A_k\subseteq T_u, i_k\notin T_u, R_k\in R\}$.

由定义 1，候选规则（eligible rule[12-13]）是前项包含于 T_u、且后项不被 T_u 包含的规则集合，这些规则是目标项目推荐分值的计算依据. 尽管选择高质量规则和计算推荐分值的方法千差万别（如 2.2 节所述），根据 T_u 搜索出候选规则集合是关联规则推荐的基础步骤，也是制约推荐效率的关键. 对于给定的 T_u 和任意一条关联规则 R_j，判定当前规则是否为 T_u 的候选规则最朴素的方法就是判断关联规则前项和后项的每一个项目是否包含于 T_u 中，显然这种匹配策略极为耗时. 在最坏的情况下，挖掘一个用户所有候选规则的时间复杂度是 $O(\sum_{R_j\in R}|R_j|+|T_u|)$，其中 $|T_u|$ 和 $|R_j|$ 分别表示 T_u 和 R_j 所包含项目的个数. 当用户数和关联规则数目都很庞大时，候选规则集合搜索代价将无法忍受，严重制约着各种基于关联规则推荐的真正应用.

3.2 总体框架

本节给出面向关联规则推荐的可扩展分布式总体框架，如图 1 所示，框架主要包含两个模块：频繁模式挖掘及推荐计算. 第一个模块从全量用户历史事务数据集 D 中挖掘频繁模式，通常采用定期离线计算方式，如每 3 小时更新频繁模式库，由于事务数据集 D 规模通常极大，我们利用 Spark 封装的分布式 FP-growth 算法加速关联规则挖掘过程. 分布式 FP-growth 算法包含两次 MapReduce 过程[25-26]：首先，对水平分割存储在 Spark RDD（Resilient Distributed Datasets）中的各数据片进行局部计算，由 Reduce 函数汇总排序获得频繁 1-项集 FList；然后，设计 FList 负载均衡分割算法（见 4.4 节），获得一定数量的分组，将各分组结果广播到水平分割 RDD；最后，触发第 2 次 MapReduce 过程，以组投影

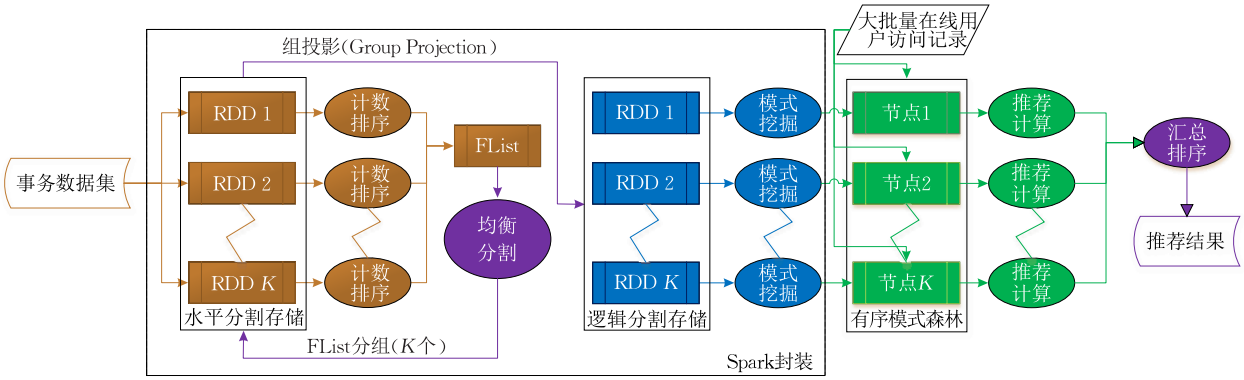


图 1 面向关联规则推荐的可扩展分布式总体框架

方法(见 4.1 节)形成新的逻辑分割 RDD,并调用 FP-growth 进行局部挖掘.在分布式关联规则挖掘中,提出 FList 负载均衡分割算法,提升传统分布式 FP-growth 挖掘框架的效率.图 1 给出的分布式框架也可以在传统的 Hadoop MapReduce 框架上实现,Spark 的优势在于以内存计算的方式大幅度降低了 I/O 代价,文献[26]验证了 Spark 在内存计算较传统 MapReduce 计算框架在支撑分布式数据挖掘算法上的性能优势.

推荐计算模块为实时在线计算任务,我们提出有序模式森林存储挖掘出的频繁模式(见 4.2 节).有序模式森林包含跨节点存储的一系列树型结构,在每个节点上存储由 FP-growth 局部挖掘获得的部分频繁模式.将大批量在线访问用户,即大量 T_u,广播到所有计算节点,为每个 T_u 挖掘候选规则集,将候选规则挖掘转化为有序模式森林中的路径搜索问题(见 4.3 节).注意,推荐分值计算往往可以融入到路径搜索过程中,以进一步提升效率.最终,将每个 T_u 在所有节点上的推荐分值汇总排序,生成最终的推荐列表.

4 技术细节

4.1 组投影与分布式 FP-growth 算法

FP-growth 算法分布式实现的核心是 Grahne 和 Zhu 提出的组投影方法[24],通过将 FList 划分成 K 组,根据每组包含的项集,在事务数据集 D 上进行投影,从而将 D 分割成互不相交的 K 个数据子集,然后分别对 K 个数据子集构建 FP 树[21]并利用 FP-growth 算法挖掘.我们在 Spark 上实现的 FP-growth 分布式算法也将利用组投影方法对原始数据进行逻辑分割.组投影的形式化定义如下.

定义 2. 组投影[24].设 FList 被划分成 K 组,即 $\mathrm{FList}=\beta_1 \cup \beta_2 \cup \cdots \cup \beta_K$,其中 $\beta_k=\{i_{k1}, i_{k2}, \cdots, i_{kr}\}, k \in \{1,2,\cdots,K\}$,是由 FList 当中 r 个连续的项目组成,于是 $D_k=\{T_p \cap (\cup_{j=1}^{k} \beta_j) \mid T_p \cap \beta_k \neq \varnothing, T_p \in D\}$ 表示事务数据集 D 在 β_k 上的组投影数据集.

由定义 2 可知,D_k 上的每条记录需满足两个条件:(1)至少包含 β_k 中的一个项;(2)不包含排在 β_k 中支持度最小项 i_{kr} 之后的所有项.

设有表 1 所示的事务数据集,按支持度降序的 FList 为 $\{G,F,E,D,C,B,A\}$.如果将 FList 分成连续的 3 组:$\beta_1=\{G,F\}$、$\beta_2=\{E,D\}$、$\beta_3=\{C,B,A\}$,根据组投影定义,将得到如表 2 所示的 3 个事务数据子集,其中小括号内数字表示该条记录重复出现的次数.以第 1 分组 $\beta_1=\{G,F\}$ 为例,$\{F,G\}(4)$ 表示 $\{F,G\}$ 重复出现 4 次,由表 1 事务 ID 1、5、6 和 8 导出,值得注意地是,表 1 事务 ID 为 10 的记录 $\{A,B,D,E\}$ 与 $\{G,F\}$ 的交集为空,因此事务 ID 10 在 $\{G,F\}$ 上不产生任何投影记录.

表 1 事务数据集示例

事务 ID	事务	事务 ID	事务
1	B,C,F,G	6	C,F,G
2	D,E,G	7	B,C,D,G
3	B,C,D,E,G	8	E,F,G
4	A,D,E,F	9	A,B,C,D,E,F
5	A,C,F,G	10	A,B,D,E

表 2 组投影分割数据集示例

组	组投影数据
$\{G,F\}$	$\{F,G\}(4),\{G\}(3),\{F\}(2)$
$\{E,D\}$	$\{D,E,G\}(2),\{D,E,F\}(2),\{D,G\},\{E,F,G\},\{D,E\}$
$\{C,B,A\}$	$\{B,C,F,G\},\{B,C,D,E,G\},\{A,D,E,F\},\{A,C,F,G\},$ $\{C,F,G\},\{B,C,D,G\},\{A,B,C,D,E,F\},\{A,B,D,E\}$

组投影过程结束后,数据事务集 D 被划分成了 K 个互不相交的数据子集,K 个计算节点在得到各自的组投影数据集 D_k 后,分别建立 FP 树,利用 FP-growth 算法,挖掘每个分组的频繁模式.需要注

意两点：(1) FP-growth 自上而下遍历 FList 递归构建 FP 树时仅需遍历 β_k 分组中的项，而非 FList 全部；(2) 在 D_k 上挖掘出包含 β_k 分组中项的所有频繁模式.

4.2　有序模式森林的定义

分布式 FP-growth 将在每个计算节点上挖掘出部分频繁模式，即包含 β_k 分组中至少一个项的频繁模式. 同时，FP-growth 自底向上的遍历方式使得挖掘出的每个频繁模式遵循 FList 偏序关系. 类似于 FP 树可以对每条记录进行压缩存储一样，本节提出一种树型结构压缩存储频繁模式. 由于频繁模式分布式存储于 K 个节点，这种树型结构本质上是一个森林，称为有序模式森林，定义如下.

定义 3.　有序模式森林(Ordered-Patterns Forest, OPF). 有序模式森林由多棵多叉树组成，每个多叉树的节点包含四个域：*item*、*child_list*、*parent* 和 *statinfo*，分别对应项目名称、孩子节点、父亲节点与用于推荐计算的统计量.

在有序模式森林中，节点的 *parent* 域保存指向父节点的指针，可以通过回溯到根节点的方式获取完整的频繁模式；*statinfo* 域保存根据关联规则推荐的不同机制灵活定义的统计量，参与推荐分值的计算，将在 4.3 节进一步阐明. 算法 1 给出了构建有序模式森林的伪代码，其中虚根节点 v_0 用来保存指向多叉树根节点的指针.

算法 1.　有序模式森林构建算法.

输入：在 D_k 上得到的局部频繁模式集 P_k

输出：虚根节点 v_0

```
1.  创建指针 f 和虚根节点 v0
2.  FOR 每一条频繁模式 pj∈Pk DO
3.      f←v0   /*f 指针指向虚根节点 v0 */
4.      FOR 每一个项目 ijk∈pj DO
5.          IF 存在 f 孩子节点 x 的名称等于 ijk THEN
6.              f←x   /*f 指针下移一层 */
7.          ELSE
8.              创建节点 y, y.item←ijk, y.parent←f
9.              将 y 添加到 f.child_list, f←y
10.         END IF
11.     END FOR
12.     将 pj 的统计量保存到 y.statinfo 中
13. END FOR
14. RETURN v0
```

以表 1 数据集为例，当 *minisupp* 为 20% 时，在 β_1、β_2、β_3 分组对应的 3 个计算节点上共挖掘出 41 条频繁模式，利用算法 1 可以构建出如图 2 所示的有序模式森林. 图中的节点展示了 *item* 和 *statinfo* 属性，*statinfo* 存储了频繁模式的支持数. 以 β_3 分组中 C 为根节点的多叉树为例，除根节点 C 外，其它每个节点都代表一条频繁模式. 例如，第 2 层节点 E 到根节点的路径为〈C,E〉，代表频繁模式{C,E}，该模式的统计量存放在尾项节点 E 上.

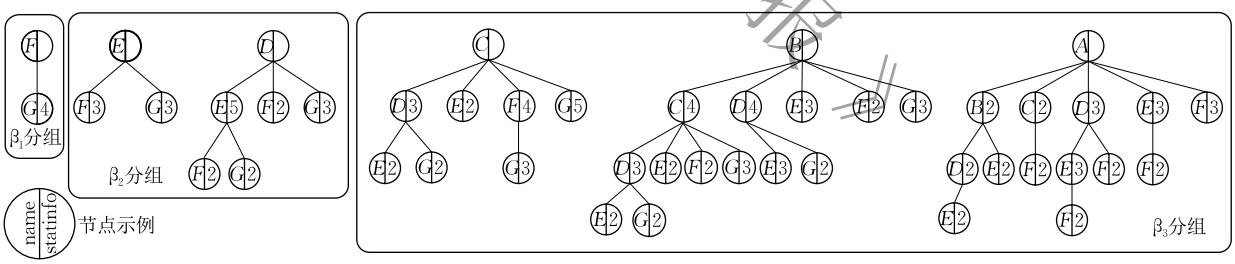


图 2　有序模式森林示例

OPF 中每条始于根节点止于任意节点的路径对应一条频繁模式，因此 OPF 的空间复杂度为 $\Theta(|P|)$，即等同于频繁模式集合的大小. Mobasher 等人[10]设计频繁项集图(Frequent Itemset Graph, FIG)用于存储频繁模式，FIG 中每个节点存储一条完整的频繁模式，包含模式所有项及其支持度. OPF 通过排序以频繁模式尾项代表一条模式，相比于 FIG 极大地降低了存储空间. 另外，Grahne 等人[32]提出 MFI 树型结构保存最大频繁模式，无法支撑所有频繁模式的存储，容易丢失关联规则推荐所需的信息.

4.3　基于路径搜索的推荐计算

有序模式森林完成了频繁模式在内存中的压缩存储，本节将讨论如何基于有序模式森林挖掘 T_u 的候选规则集合. 具体地，首先定义目标路径集合，并证明其与 T_u 的候选规则集合呈现出一一对应关系. 然后提出一种目标路径集合搜索算法，可在单机上完成对 T_u 候选规则集合的高效挖掘.

定义 4.　目标路径集合. 给定用户记录 T_u，令 $V_{ul}=\langle v_0, v_1, \cdots, v_l\rangle$ 为 OPF 上的一条路径，若 V_{ul} 为目标路径集合 V_u 中的一条目标路径，则 V_{ul} 满足如下条件：(1) v_0 是 OPF 中的一棵多叉树的根节点，对

于任意 $0\leq j<l$，$v_{j+1}\in v_j.child\_list$；(2) 存在 v_t，$v_t.item\notin T_u$，并且对于任意 $1\leq j\leq l, j\neq t, v_j.item\in T_u$. $v_t.item$ 被称为目标项目.

由定义 4 可知，如果某条路径是 T_u 的目标路径，则该条路径可以拆分成一条或多条目标路径，且这些目标路径具有相同的目标项目. 例如，令 $\langle B,C,D,E\rangle$ 是 T_u 的目标路径，目标项目为 C，则 $\langle B,C,D\rangle$ 和 $\langle B,C\rangle$ 是 T_u 的目标路径，目标项目均为 C.

定理 1. 给定用户记录 T_u 和有序模式森林，T_u 的目标路径集合和候选规则集合是一一对应的.

证明. 给定目标路径集合 V_u 和候选规则集合 R_u，首先证明目标路径集合中的每条路径与候选规则集合中一条候选规则相对应. 由 4.2 节可知，T_u 的任意目标路径 $V_{ul}\in V_u$ 都代表一条频繁模式 p_l. 由于目标项目不属于 T_u，因此 p_l 产生一条规则 $R_{ul}: A_l\rightarrow i_l$. 根据定义 1，$R_{ul}$ 是 T_u 的候选规则，即 $R_{ul}\in R_u$.

其次证明候选规则集合中的每条候选规则与目标路径集合中的一条目标路径相对应. 对于 R_u 中的任意一条候选规则 $R_{ul}: A_l\rightarrow i_l$，由于产生此条候选规则的频繁模式 p_l 被保存在有序模式森林的一条路径 V_{ul} 上，又因为有且仅有一个项目 $i_l\notin T_u$，因此根据定义 4，该条路径是 T_u 的一条目标路径，即 $V_{ul}\in V_u$.

综上，T_u 的目标路径集合和候选规则集合是一一对应的. 证毕.

由定理 1 可知，每条目标路径都对应一条候选规则，因此，搜索出 T_u 所有的目标路径，就可以获得 T_u 所有的候选规则.

算法 2 和 3 给出了搜索 T_u 目标路径集合的伪代码. 总体而言，目标路径搜索算法的骨架是多叉树深度优先遍历(算法 2 第 1～10 行，由堆栈 S 控制)，其中通过在每个节点引入 $color$ 变量巧妙地区分搜索状态. 具体地，$color$ 变量取值为：$white$、$gray$ 和 $black$. 初始化时所有节点着 $white$ 色，当路径上每次发现不包含于 T_u 中的项时(算法 3 第 1～2 行)，算法 3 第 2 行的 $DeepenColor$ 函数将 $color$ 变量加深一级. 因此，$gray$ 色表示第一次发现未在 T_u 中的项，即发现目标项目，$black$ 色表示路径上出现第二个未包含于 T_u 中的项，这意味着本次路径搜索可以停止(算法 2 第 5 行).

算法 2. 目标路径搜索算法.

输入：虚根节点 v_0，用户记录 T_u

输出：字典结构 V_u 用于保存 T_u 的目标路径集合

```
1.  创建堆栈 S, S.PUSH(v0.child_list, white)
2.  WHILE S≠∅ DO
3.    v←S.POP()
4.    Vu←PathOperator(v, Tu) /*见算法 3*/
5.    IF v.color≠black THEN
6.      FOR vv in v.child_list DO
7.        S.PUSH(vv, v.color)
8.      END FOR
9.    END IF
10. END WHILE
11. RETURN Vu
```

算法 3. *PathOperator* 函数.

输入：节点 vv 和用户记录 T_u

输出：字典结构 V_u 用于保存 T_u 的目标路径集合

```
1.  IF vv.item∉Tu THEN
2.    DeepenColor(vv.color)
3.    IF vv.color=gray THEN
4.      iq=vv.item, Update(Vu[iq]) /*iq是目标项目*/
5.    END IF
6.  END IF
7.  IF vv.item∈Tu AND vv.color=gray THEN
8.    Update(Vu[iq]) /*更新 iq的目标路径*/
9.  END IF
10. RETURN Vu
```

为更清晰地说明目标路径搜索算法过程，我们给出如图 3 的一个简单例子，其中 $T_u=\{A,C,D\}$. 首先查看路径 $\langle A,B,D,E\rangle$，A 先入栈并初始化 $color$ 为 $white$，A 出栈，由于 $A\in T_u$，不做任何操作. 此时 B 入栈，继承父节点的 $color$ 变量(算法 2 第 7 行)，在 B 出栈时，由于 $B\notin T_u$，B 节点的 $color$ 加深为 $gray$，表示目标项目为 B，获得第一条以 B 为目标项目的目标路径 $\langle A,B\rangle$，对应的频繁模式为 $\{A,B\}$. 接着 D 继承父节点的 $color$ 入栈，当 D 出栈时，由于 $D\in T_u$ 且 D 的 $color$ 值为 $gray$，挖掘出第二条以 B 为目标项目的目标路径 $\langle A,B,D\rangle$. E 继承父节点的 $color$ 入栈，在 E 出栈时，由于 $E\notin T_u$，E 的 $color$ 值被加深为 $black$，此时该条路径上遍历结束. 当遍历完整个多叉树，将得到如图 3 所示的每个节点的匹配状态，其中方框标注节点代表每条路径的目标项目.

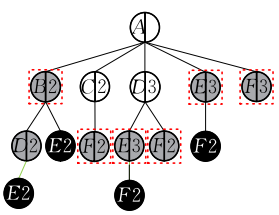


图 3 目标路径搜索示例

在搜索 T_u 的目标路径时，最坏的情况下，有序模式森林上所有的节点都被检查一遍，每个节点均需要与 T_u 比较，复杂度为 $\Theta(|T_u|)$，所以算法 2 的时间复杂度为 $O(|P||T_u|)$，因此搜索出所有待推荐用户的目标路径需要 $O(\sum_{T_u\in D}|P||T_u|)$. 如 3.1 节所述，若采用穷举法获取所有用户候选规则集合需要的时间复杂度为 $O(\sum_{T_u\in D}\sum_{R_j\in R}|R_j||T_u|)$. 一条频繁模式可以产生多条关联规则，即 $|P|\ll\sum_{R_j\in R}|R_j|$，所以有 $O(\sum_{T_u\in D}|P||T_u|)\ll O(\sum_{T_u\in D}\sum_{R_j\in R}|R_j||T_u|)$. 从以上时间复杂度分析可知，基于有序模式森林的目标路径搜索方法可大幅度提升候选规模挖掘效率.

Update 函数的灵活设计. 最朴素的 *Update* 函数是将候选规则保存到字典结构中(算法 3 第 4 和 8 行)，供后续推荐分值计算时使用. 事实上，如果推荐计算方法事先确定，我们可以重构 *Update* 函数将推荐分值计算融入到目标路径的搜索过程中，同时 OPF 中 *statinfo* 域存储相应的统计信息. 如 2.2 节所述，国内外学者提出了很多不同的高质量规则选择及其推荐分值计算方案，下面我们举两个例子来说明通过灵活设计 *statinfo* 域和 *Update* 函数，本文所提出的计算框架可以支撑已有的关联规则推荐方法.

例 1. 基于最大置信度规则的推荐[8-10]. OPF 中 *statinfo* 域保存频繁模式的支持度，每次发现新目标路径时调用 *Update* 函数，设该目标路径和目标项目得出的关联规则是 $R_l: A_l\rightarrow i_l$，计算 R_l 的置信度，需要获得 $supp(A_l,i_l)$ 和 $supp(A_l)$，该路径对应频繁模式 $\{A_l,i_l\}$，即当前节点 *statinfo* 域即为 $supp(A_l,i_l)$. 再搜索路径 A_l 即可获得 *statinfo* 域中的 $supp(A_l)$. 因此，每条新目标路径对应规则的置信度可以通过 *statinfo* 域计算，*Update* 函数针对每个目标项目保存最大置信度的规则，即比较 $V_u[i_q]$ 存储的规则与最新获得规则直接的置信度，将 $V_u[i_q]$ 更新最大置信度规则.

例 2. 基于多规则融合的推荐[16]. 推荐分值由所有相同目标项目的关联规则融合得出，即对任意目标项目 i_{jk}，推荐分值计算为

$$Score(i_{jk})=\sum_{R_{jk}}supp(R_{jk})conf(R_{jk}) \tag{1}$$

类似于例 1，*Update* 函数首先计算目标路径对应规则的支持度和置信度，$V_u[i_q]$ 存储当前的 i_q 推荐分值，并将最新获得规则分值由式(1)加到 $V_u[i_q]$ 当前值.

由上面两个例子，通过灵活设计 *Update* 函数可以将推荐分值计算融入到目标路径搜索过程中，兼容已有的关联规则推荐方法，体现出本文所提出计算框架的优良通用性.

4.4　负载均衡分割方法

分布式系统的运行时间取决于最迟完成任务节点的计算时间，因此，均衡各计算节点的负载是降低算法总执行时间的关键[33]. 图 1 所示的分布式总体框架包含频繁模式挖掘与推荐计算两个阶段，因此我们试图定义一个合理的负载指数(load indicator)，能同时刻画两阶段的负载，在此基础上提出多项式级的负载均衡分割算法.

在频繁模式挖掘过程中，负载指数应反映出 FList 中每个项目对应的条件 FP 树的规模. 设在组投影过程中，每个分组只包含一个项目，那么每个分组的投影数据的长度和数量，是衡量每个项目负载的重要指标. 假设有项目 i_k，定义 2 要求每条投影记录必须包含 $\{i_k\}$，因此事务数据集 D 在分组 $\{i_k\}$ 的投影数据集由 $|N_k|=|D|supp(i_k)$ 条数据组成. 定义 2 同时要求每条投影记录仅包含支持度大于 $\{i_k\}$ 的项，投影记录平均长度可以用式(2)估算：

$$L_k=\sum_{q=1}^{k}supp(i_q) \tag{2}$$

于是分组 $\{i_k\}$ 的投影数据集的规模可以估算为

$$|D_k|=|N_k|L_k|=|D|supp(i_k)\sum_{q=1}^{k}supp(i_q) \tag{3}$$

由式(3)可知，FList 中项目的支持度正比于其投影数据集规模，即条件 FP 树规模，因此本文提出利用项目支持度作为负载指数，可衡量频繁模式挖掘的计算负载. 在给出负载指数后，频繁模式挖掘的负载均衡分割问题就转化为：将 FList 中项目按照支持度划分成多组，使得每个分组具有相近的支持度和. 若将 FList 中所有项目在 K 个分组上的支持度和的平均值定义为

$$\overline{S}=\frac{\sum_{i_q\in \mathrm{FList}}supp(i_q)}{K} \tag{4}$$

由定义 2，组投影 FList 分成 K 组，即 $\mathrm{FList}=\beta_1\cup\beta_2\cup\cdots\cup\beta_K$，$\beta_k=\{i_{k1},i_{k2},\cdots,i_{kr}\}$，负载均衡分割试图使任意 β_k 满足：

$$\min_{\beta_k}|S_k-\overline{S}|,\ S_k=\sum_{q=1}^{r}supp(i_{kq}) \tag{5}$$

众所周知，将一个集合划分为多个和相等的子

集称为集合平分问题，为 NP 完全问题. 然而，由定义 2，组投影分割 FList 时要求每个分组内是由若干连续的项目组成，即按支持度排序将 FList 有序地分成 K 个分组. 在这种特殊情形下，可以设计出多项式时间度的算法获得 FList 的均衡分割，伪代码如算法 4 所示.

算法 4. 负载均衡分割方法.

输入：FList 和分组数 K

输出：K 个分组的集合

1. 计算 K 个分组支持度和的平均值 $\bar{S}$
2. $S'\leftarrow 0, S\leftarrow 0, k\leftarrow 1, s\leftarrow 1, q\leftarrow 1$
3. FOR 按照支持度降序，对于每个项目 $i_q\in$FList DO
4. $S'\leftarrow S, S\leftarrow S+supp(i_q)$
5. IF $S\geq\bar{S}$ THEN
6. IF $S'=0$ OR $|S-\bar{S}|<|S'-\bar{S}|$ THEN
7. $\beta_k\leftarrow\{i_s,\cdots,i_q\}$, $s\leftarrow q+1$, $k{+}{+}$, $S\leftarrow 0$
8. ELSE
9. $\beta_k\leftarrow\{i_s,\cdots,i_{q-1}\}, s\leftarrow q, k{+}{+}, S\leftarrow supp(i_q)$
10. END IF
11. END IF
12. END FOR
13. RETURN $\{\beta_1,\cdots,\beta_K\}$

第 5 行的判定条件意味着获得了第一次超过平均值 $\bar{S}$ 的分割，由于与 $\bar{S}$ 尽量接近的分割可能在 $S_k<\bar{S}$ 时就出现，同时，我们在 S' 中保存减少一个项时的支持度和. 因此，第 6～10 行通过判定 $|S-\bar{S}|$ 与 $|S'-\bar{S}|$ 的大小来决定最后一个项 i_q 是否被纳入到当前分组. 显然，算法 4 遍历一遍 FList 就可完成，其时间复杂度为 $\Theta(|\text{FList}|)$.

由于负载均衡按 FList 划分成支持度和接近相等的 K 组，在分布式 FP-growth 挖掘结束后，每个计算节点上的有序模式森林中树根节点的支持度之和也就接近相等. 对于任意一颗有序模式森林中树，由于算法 1 按支持度升序插入模式，其根节点支持度越小，树的规模越大. 因此，每个计算节点上树根节点支持度的均衡，使得分散存储的各个有序模式森林规模接近均衡. 对于同样的 T_u，算法 2 的搜索效率取决于森林的规模，因此将项目支持度作为负载指数既能刻画频繁模式挖掘，又可衡量推荐计算的负载.

5 实验结果和分析

本节在 3 个真实数据集上验证所提出的方法对提升关联规则推荐计算效率的有效性.

5.1 实验设置

数据集. 本文选用 3 个来自不同领域的数据集：Accidents①、Webdocs② 以及 Amazon③，Accidents 和 Webdocs 数据集为关联分析领域广泛使用的标准数据集；Amazon 数据集是亚马逊购物网上书籍评分记录，其中每条记录表示用户评过分的书籍集合. 表 3 列出了实验数据集的基本统计信息.

表 3 实验数据集统计信息

数据集	事务数	项目数	事务最短长度	事务最长长度	事务平均长度	稀疏度/%
Accidents	340183	468	18	51	34	7.22391
Webdocs	1692082	5267656	1	71472	177	0.00336
Amazon	8026324	2330066	1	43201	3	0.00012

评价指标. 本文利用执行时间和加速比（*Speedup*）两项指标作为衡量算法效率的指标，执行时间直观地反应计算效率，而加速比用以刻画执行时间的提升倍数，由式(6)计算：

$$Speedup=T_1/T_2 \tag{6}$$

其中，T_1 和 T_2 分别表示两种不同方法完成同一任务的计算时间. 此外，我们引入负载不均衡指数[34]（记为 I）刻画负载均衡性能，计算公式为

$$I=T_{\max}/T_{\text{avg}}-1 \tag{7}$$

其中，$T_{\max}$ 表示最后完成任务所花费的时间，T_{avg} 表示所有节点完成各自任务的平均时间. 负载不均衡指数数值越大，意味着负载越不均衡，即负载均衡策略越差.

实验环境. 本文在 Spark 集群上实现关联规则推荐的分布式计算框架. 集群共包含 16 个节点，节点间使用千兆网络互联. 每台节点的配置为：Intel 4 核 E5-2650v2 型 CPU（主频 2.6 GHz），128 GB 内存，240 GB SSD 硬盘以及 600 GB SAS 硬盘. 实验代码采用 Java 语言编写，Spark 版本为 1.6.2，默认将 16 个计算节点全部用于分布式计算.

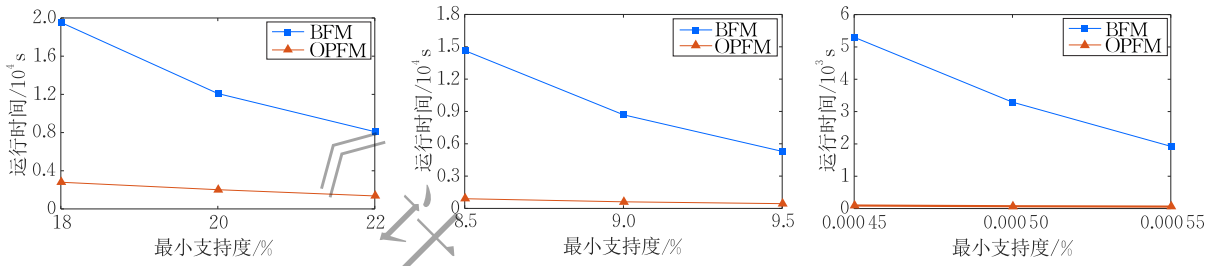
5.2 推荐计算模块的效率分析

本节验证当关联规则被提取之后，面对大规模待推荐用户时的推荐匹配计算效率. 首先验证单机环境下有序模式森林对推荐计算效率的提升，然后

① http://fimi.ua.ac.be/data/accidents.dat
② http://fimi.ua.ac.be/data/webdocs.dat.gz
③ http://snap.stanford.edu/data/web-Amazon.html

验证分布式有序模式森林相较于单机环境对推荐计算效率的提升.

由于已有研究聚焦于关联规则挖掘效率及推荐计算准确率的提升，尚未有研究考虑推荐计算阶段的效率问题，因此我们只能选用朴素的穷举法（下文简称 BFM）作为单机环境下的比较对象，即逐一比对 T_u 和每条频繁模式. 本文从 3 个数据集中分别抽取 30 000 条事务作为待推荐用户 T_u，图 4 给出了 BFM 和 OPFM 在 3 个实验数据集设置不同支持度阈值上的推荐计算时间对比结果. 可以发现，对于所有数据集，有序模式森林都极大地缩短了用户交易记录与频繁模式之间的匹配时间. 以 Accidents 数据集为例，当支持度设置为 18% 时，一共可挖掘出 1 347 146 条频繁模式，使用 OPFM 将待推荐用户交易记录与挖掘出的频繁模式进行匹配时需耗时 2796 s，其中 OPF 构建时间仅需 621 ms，为每个用户产生推荐仅耗 10 ms，达到了实时推荐，而 BFM 推荐计算则需要耗时 19 542 s. 其原因在于 BFM 在匹配具有相同前缀的频繁模式时，进行了大量的重复比较，而在 OPFM 中，具有相同前缀的模式共享路径前缀，每个节点都代表一条频繁模式，因此，相同的比较只需要执行一次，从而极大地缩短了匹配时间.



(a) Accidents　(b) Webdocs　(c) Amazon

图 4　单机环境下 BFM 与 OPFM 的推荐计算时间对比

此外，3 组实验都具有同一趋势，随着支持度减小，BFM 耗费的时间急剧上升，而 OPFM 耗费的时间增长缓慢. 产生这种现象的主要原因是，随着支持度的减小，频繁模式的数量急剧增加，BFM 对于每一条频繁模式都需要和用户交易记录进行匹配，这就导致耗费的时间增长过快，而 OPFM 能够充分利用有序模式森林的优势，显著降低具有相同前缀的频繁模式与用户交易记录的匹配时间. 因此，对于具有多用户、大规模频繁模式的推荐计算，OPFM 更具有优势.

表 4　OPFM 相对于 BFM 的加速比

数据集	加速比（支持度阈值%）		
Accidents	5.92(22)	5.99(20)	6.99(18)
Webdocs	11.94(9.5)	14.08(9)	16.26(8.5)
Amazon	31.66(0.00055)	45.74(0.0005)	55.99(0.00045)

除了对运行时间进行比较，本文还比较了不同数据集和支持度下，OPFM 相对于 BFM 的加速比，这里加速比定义为 $T_{\mathrm{BFM}}/T_{\mathrm{OPFM}}$，其中 T_{BFM} 和 T_{OPFM} 分别代表 BFM 和 OPFM 所需的时间. 从表 4 所列出的结果可看出，在较小规模数据集 Accidents 中，OPFM 获得加速比约为 6，随着数据集规模的扩大，OPFM 所获加速比急速上升，在稀疏数据集 Amazon、支持度阈值取最低时，加速比达到惊人的 56. 由于稀疏数据集能较早满足 OPFM 的搜索中止条件（即两次在 T_u 上匹配不成功），从而使得 OPFM 在稀疏数据集上表现出更好的效果.

上述实验结果说明在单机环境下，OPFM 较传统的 BFM 极大地提升了推荐算法的效率，而分布式框架的引入将进一步提升有序模式森林方法的性能. 本文选用 Webdocs 和 Amazon 两个大规模数据集对单机与分布式环境下的 OPFM 进行比较. 对于 Webdocs 数据集，*minisupp* 设置为 8% 和 8.5%，分别挖掘出 1 743 639 和 1 090 406 条频繁模式；对于 Amazon 数据集，*minisupp* 设置为 0.0004% 和 0.00045%，分别挖掘出 1 484 121 和 864 992 条频繁模式. 此外分别从两个数据集中抽取 10、30 以及 50 万条事务作为用户的交易记录，每条事务对应一个用户. 由于单机和分布式环境下构建 OPF 的耗时均在 1 s 左右，远小于在 OPF 中进行搜索的时间，因此，本文关注在 OPF 上的目标路径搜索时间，即计算推荐结果所需的时间.

表 5 给出在单机和分布式环境下 OPFM 在两种数据集、不同规模用户数上的运行时间及加速比. 这里加速比采用 $T_{\text{stand-alone}}/T_{\text{parallel}}$，其中 $T_{\text{stand-alone}}$ 表示单机环境下算法的运行时间，T_{parallel} 表示分布式环境下算法的运行时间. 可以发现，基于有序模式森林的分布式算法能够显著缩短推荐计算所需要的时间，在绝大部分情况下，较单机版算法能达到超过 7 倍

的加速比.此外,在相同支持度下,随着用户规模增大,分布式算法消耗的时间近似线性增长,因此,基于有序模式森林的分布式推荐算法具有较好的可扩展性,能够胜任大规模用户的推荐任务.

表 5 分布式环境与单机环境下 OPFM 效率的对比

数据集	支持度/%	测试用户数	单机时间/s	分布式时间/s	加速比
Webdocs	8	100000	5562	804	6.9
		300000	16269	2507	6.5
		500000	28096	4140	6.8
	8.5	100000	3250	333	9.8
		300000	10176	947	10.8
		500000	17565	1579	11.1
Amazon	0.0004	100000	544	33	16.5
		300000	1178	96	12.3
		500000	1830	159	11.5
	0.00045	100000	307	25	12.3
		300000	854	78	10.9
		500000	1403	128	11.0

5.3 分布式总体框架负载均衡性分析

5.3.1 频繁模式挖掘模块的效率及负载均衡性分析

为验证频繁模式挖掘阶段的效率,引入两种著名的方法 Apriori 和 Eclat 作为非 FP-growth 方法的代表,公平起见,将上述两种方法在 Spark 环境下的分布式版本作为比较对象(代码来源于 github①),分别称为 Apriori+和 Eclat+.另一方面,为验证在分布式FP-growth方法中所提出的负载均衡分割的有效性,称为 LBP_O,引入两种负载均衡分割策略作为比较对象:(1)均等分割(Equal Partitioning,EP),即 FList 按项数量等分,这是 Mahout、Spark MLlib 等机器学习库普遍采用的分组机制;(2)对数负载均衡分割(LBP_L)[27],以项在 FList 中排序位置的对数作为负载指标.

图 5 中的双 y 轴图展示了 5 种分布式频繁模式挖掘方法的执行时间(见与左 y 轴对应的条柱)以及 3 种 FP-growth 负载均衡策略的负载不均衡指数对比(见与右 y 轴对应的折线).首先,从不同类型的分布式频繁模式挖掘方法上看,Apriori+效率最低,Eclat+比分布式 FP-growth 略差,这说明 FP-growth 确实是最适合扩展至分布式针对大数据频繁模式挖掘任务的方法.其次,从 3 种分布式 FP-growth 下负载均衡分割策略上看,在所有实验场景下,EP 策略表现最差,即 EP 的执行时间和负载不均衡指数均最高;总体上,我们的 LBP_O 比 Zhou 等人[27]提出的 LBP_L 更加优越,在最小规模数据集 Accidents 上,LBP_O 和 LBP_L 性能接近,而在 Webdocs 和 Amazon 两个大数据集上,LBP_O 在所有支持度阈值情形下均优于 LBP_L.这得益于 LBP_O 综合估算了投影数据集的规模,而不像 LBP_L 单纯以 FLsit 长度作为 FP-growth 挖掘时间的衡量标准.

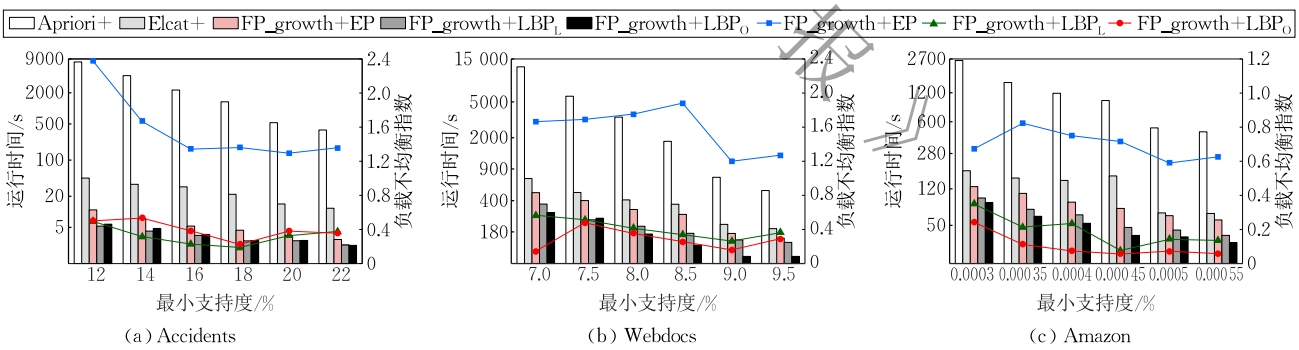


图 5 分布式频繁模式挖掘算法的效率对比

接下来,考察分组数量 K 对上述 3 种负载均衡分割策略的影响,分组数量 K 对应于参与分布式计算的节点数量,K 越大、计算任务就越多,负载不均衡问题也更易凸显.这组实验选取 Amazon 数据集,*minsupp* 设为 0.0003%,改变分组数量 K,对比算法在运行时间和加速比上的差异,这里加速比定义为 $T_{\text{stand-alone}}/T_{\text{parallel}}$,其中 $T_{\text{stand-alone}}$ 和 T_{parallel} 分别表示 FP-growth 算法在单机环境和分布式环境中的执行时间.图 6 给出了实验对比结果,从图 6(a)可以看出,EP 策略依然最弱,而 LBP_O 优于 LBP_L. 图 6(b)的加速比指标则表达出另一个有趣现象,当分组数

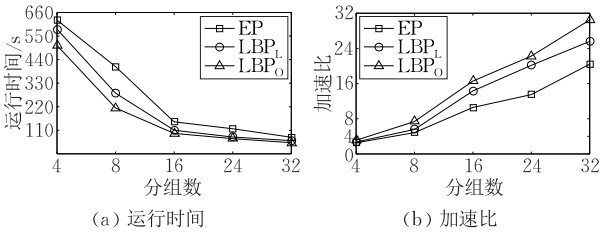


图 6 分组数对负载均衡分割策略的影响(Amazon)

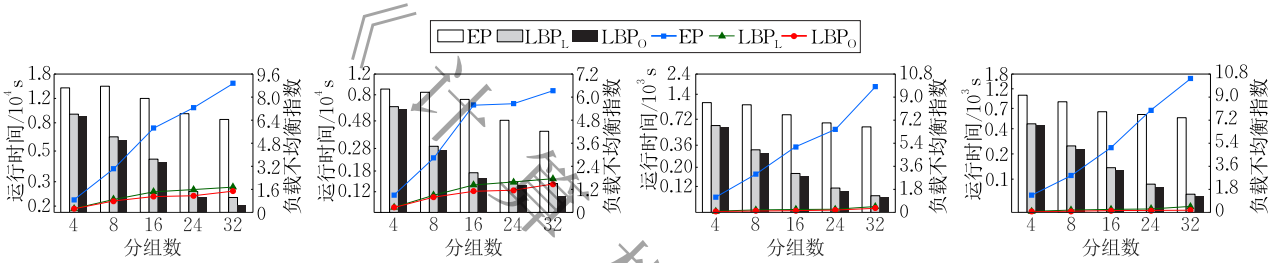
① https://github.com/YP-Liu/Apriori

量增大，计算任务变多时，LBP_O之间 LBP_L的差距在逐渐增大，这说明当投影数据集切分越细，简单由 FList 估算 FP-growth 运行时间的方法变得越不精准. 这也反过来说明本文所提出的 LBP_O策略具有更好的可扩展性，即随着计算节点的增加，加速比的增加越来越明显.

5.3.2　推荐计算模块的负载均衡性分析

本节验证基于关联规则推荐的分布式总体框架在推荐计算部分的负载均衡性. 实验选取 Amazon 数据集和 Webdocs 数据集，分别验证 3 种负载均衡分割策略在不同计算节点数量条件下对第 2 阶段推荐计算的影响. 此时，待推荐用户 T_u数量取 50 万条，同时对 2 个数据集各取 2 组不同的支持度阈值. 推荐计算方法使用本文提出的方法，在挖掘过程结束后，直接在各个节点上，利用当前节点在挖掘阶段产生的频繁模式构建有序模式森林，将待推荐用户广播到各个节点上，进行下一步的匹配工作.

实验结果如图 7 所示，总体而言，LBP_O策略较之于其他 2 种策略在推荐计算模块依然表现出最佳性能. 然而，LBP_O与 LBP_L的执行时间和不均衡指数均非常接近，究其原因，本文提出的分布式 OPFM 匹配计算速度极快，如 5.2 节所述，OPFM 在百万级频繁模式上，完成一条 T_u的推荐计算仅需 10 ms. 尽管 LBP_L策略在挖掘阶段表现欠佳，即每个计算节点上频繁模式数量差异比较大，但是，分布式 OPFM 高计算速度削弱了由模式数量不均导致的差异，这使得 LBP_O与 LBP_L的执行时间差距变小.



(a) Webdocs (8.5%)　(b) Webdocs (8%)　(c) Amazon (4.5%)　(d) Amazon (4%)

图 7　负载均衡策略对分布式推荐计算的影响

6　结　论

本文提出面向关联规则推荐的可扩展分布式总体框架，框架主要包含频繁模式挖掘及推荐计算两个模块，并在 Spark 平台上实现两个模块的无缝衔接. 具体地，本文首先提出有序模式森林的数据结构用来压缩存储所有的频繁模式. 其次将候选规则的挖掘过程转变为有序模式森林中的路径搜索问题，并提出一个高效的目标路径搜索算法，同时将规则选择和推荐分值的计算融入到目标路径的搜索过程中，以降低空间和时间开销. 最后提出负载均衡的数据分割策略，以提高分布式频繁模式挖掘与分布式推荐计算任务整体效率. 实验采用 3 个公开数据集验证分布式计算框架的效率和负载均衡性的问题，实验结果表明基于有序模式森林的推荐计算比传统穷举匹配策略降低 6 倍以上时间，同时提出的分布式计算框架可随计算节点数量的变化达到近线性扩展.

参　考　文　献

[1] Zhang Yu-Jie, Du Yu-Lu, Meng Xiang-Wu. Research on group recommender systems and their applications. Chinese Journal of Computers, 2016, 39(4): 745-764(in Chinese)
(张玉洁，杜雨露，孟祥武. 组推荐系统及其应用研究. 计算机学报，2016，39(4)：745-764)

[2] Gan M, Jiang R. FLOWER: Fusing global and local associations towards personalized social recommendation. Future Generation Computer Systems, 2018, 78: 462-473

[3] Adomavicius G, Tuzhilin A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(6): 734-749

[4] Liu Shu-Dong, Meng Xiang-Wu. Recommender system in location-based social networks. Chinese Journal of Computers, 2015, 38(2): 322-336(in Chinese)
(刘树栋，孟祥武. 基于位置的社会化网络推荐系统. 计算机学报，2015，38(2)：322-336)

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

[6] Nakagawa M, Mobasher B. A hybrid web personalization model based on site connectivity//Proceedings of the 5th International WebKDD Workshop Web Mining as a Premise to Effective and Intelligent Web Applications. Washington, USA, 2003: 59-70

[7] Davidson J, Liebald B, Liu J, et al. The YouTube video recommendation system//Proceedings of the 4th ACM Conference on Recommender Systems. Barcelona, Spain, 2010: 293-296

[8] Sandvig J J, Mobasher B, Burke R. Robustness of collaborative recommendation based on association rule mining//Proceedings of the 1st ACM Conference on Recommender Systems. Minneapolis, USA, 2007: 105-112

[9] Zaïane O R. Building a recommender agent for e-learning systems//Proceedings of the International Conference on Computers in Education. Auckland, New Zealand, 2002: 55-59

[10] Mobasher B, Dai H, Luo T, et al. Effective personalization based on association rule discovery from web usage data//Proceedings of the 3rd International Workshop on Web Information and Data Management. Atlanta, USA, 2001: 9-15

[11] Wang F H, Shao H M. Effective personalized recommendation based on time-framed navigation clustering and association mining. Expert Systems with Applications, 2004, 27(3): 365-377

[12] Li W, Han J, Pei J. CMAR: Accurate and efficient classification based on multiple class-association rules//Proceedings of the International Conference on Data Mining. San Jose, USA, 2001: 369-376

[13] Rudin C, Letham B, Salleb-Aouissi A, et al. Sequential event prediction with association rules//Proceedings of the 24th Annual Conference on Learning Theory. Budapest, Hungary, 2011: 615-634

[14] Ghoshal A, Sarkar S. Association rules for recommendations with multiple items. INFORMS Journal on Computing, 2014, 26(3): 433-448

[15] Ghoshal A, Menon S, Sarkar S. Recommendations using information from multiple association rules: A probabilistic approach. Information Systems Research, 2015, 26(3): 532-551

[16] Lin W, Alvarez S A, Ruiz C. Efficient adaptive-support association rule mining for recommender systems. Data Mining and Knowledge Discovery, 2002, 6(1): 83-105

[17] Wang Y, Wu J, Wu Z, et al. Popular items or niche items: Flexible recommendation using cosine patterns//Proceedings of the International Conference on Data Mining Workshops. Shenzhen, China, 2014: 205-212

[18] Wickramaratna K, Kubat M, Premaratne K. Predicting missing items in shopping carts. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(7): 985-998

[19] Ren Z, Wan J, Shi W, et al. Workload analysis, implications, and optimization on a production Hadoop cluster: A case study on Taobao. IEEE Transactions on Services Computing, 2014, 7(2): 307-321

[20] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases//Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. Washington, USA, 1993, 22(2): 207-216

[21] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation//Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, USA, 2000: 1-12

[22] Zaki M J. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(3): 372-390

[23] Parthasarathy S, Zaki M J, Ogihara M, et al. Parallel data mining for association rules on shared-memory systems. Knowledge and Information Systems, 2001, 3(1): 1-29

[24] Grahne G, Zhu J. Mining frequent itemsets from secondary memory//Proceedings of the 4th International Conference on Data Mining. Brighton, UK, 2004: 91-98

[25] Li H, Wang Y, Zhang D, et al. PFP: Parallel FP-growth for query recommendation//Proceedings of the 2nd ACM Conference on Recommender Systems. Lausanne, Switzerland, 2008: 107-114

[26] Sparks E R, Talwalkar A, Smith V, et al. MLI: An API for distributed machine learning//Proceedings of the 13th International Conference on Data Mining. Dallas, USA, 2013: 1187-1192

[27] Zhou L, Zhong Z, Chang J, et al. Balanced parallel FP-Growth with MapReduce//Proceedings of the 2011 IEEE Youth Conference on Information Computing and Telecommunications. Beijing, China, 2011: 243-248

[28] Chon K W, Hwang S H, Kim M S. GMiner: A fast GPU-based frequent itemset mining method for large-scale data. Information Sciences, 2018, 439: 19-38

[29] Chon K W, Kim M S. BIGMiner: A fast and scalable distributed frequent pattern miner for big data. Cluster Computing, 2018, 18: 1-14

[30] Song Y G, Cui H M, Feng X B. Parallel incremental frequent itemset mining for large data. Journal of Computer Science and Technology, 2017, 32(2): 368-385

[31] He Ming, Liu Wei-Shi, Zhang Jiang. Association rules recommendation algorithm supporting recommendation nonempty. Journal on Communications, 2017, 10: 18-25(in Chinese)
(何明, 刘伟世, 张江. 支持推荐非空率的关联规则推荐算法. 通信学报, 2017, 10: 18-25)

[32] Grahne G, Zhu J. Fast algorithms for frequent Itemset mining using FP-Trees. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(10): 1347-1362

[33] Zhang Peng, Duan Lei, Qin Pan, et al. Mining Top-*k* distinguishing sequential patterns using spark. Journal of Computer Research and Development, 2017, 54(7): 1452-1464(in Chinese)
(张鹏, 段磊, 秦攀等. 基于 Spark 的 Top-*k* 对比序列模式挖掘. 计算机研究与发展, 2017, 54(7): 1452-1464)

[34] Menon H, Kalé L. A distributed dynamic load balancer for iterative applications//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC). Denver, USA, 2013: 1-11

LI Chang-Sheng, M. S. His research interests include data mining and recommender systems.

WU Zhi-Ang, Ph. D., professor. His research interests include data mining and recommender systems.

ZHANG Lu, Ph. D., lecturer. His research interests include data mining.

CAO Jie, Ph. D., professor. His research interests include data mining and business intelligence.

Background

The association-rule-based recommendation model is one of the most popular recommendation engines in e-commerce websites. Much efforts in this area mainly focus on the selection of eligible rules or the combination of multiple rules for effective recommendations. However, little attention has been paid to the efficiency of the rule-based recommendation. With the rapid growth of both online customers and rules, the computational complexity of rule-based methods has become a vital concern of the real e-commerce websites. To cope with this issue, we propose a distributed-computing framework for improving the computational efficiency of rule-based recommendation. In detail, a tree-typed structure called Ordered-Patterns Forest (OPF) is designed for the compact representation of frequent patterns. Then, we transform candidate rules mining to a path-searching problem on the OPF, and present a path-searching algorithm running on single machine. Finally, a load-balanced strategy for data partitioning is proposed to decrease the running time of the task that finishes lastly. Experimental results demonstrate that the effectiveness of the proposed framework.

This work is mainly supported by the Key Project of the National Science Foundation of China under Grant 91646204. This project delivers the fundamental researches on the topics of data mining and e-commerce intelligence with applications, where the personalized recommendation is an important topic. Our project team has worked on these research areas for more than 10 years and published more than 50 refereed journal and conference papers, including KDD, TKDE and so on. The proposed framework is able to help make up the inefficiency of traditional association-rule-based recommendation methods and possess practical values to many real-life e-commerce platforms.