计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191000110

Spark 平台中的并行化 FP_growth 关联规则挖掘方法

朱岸青[1]　李　帅[2]　唐晓东[3]
1 暨南大学管理学院　广州 510000
2 北京航空航天大学计算机学院　北京 100191
3 华南师范大学经济与管理学院　广州 510006

摘　要　为了提高关联规则挖掘效率,文中提出了一种适用于 Spark 平台的并行化 FP_growth 关联规则挖掘方法。首先,利用 Spark 平台在分布式系统中的所有节点的内存 RDD 中完成遍历扫描运算,得到频繁集,以便生成 FP_Table 并更新 FP_Tree。然后,引入时间序列来预测待挖掘的项目集,以便实现分布式系统中的所有节点能够均衡分担挖掘任务,从而充分利用各节点的 FP_Tree 遍历功能,获取 FP_growth 关联规则挖掘结果。实验结果显示,相比单机情况,并行化 FP_growth 关联规则挖掘在效率方面提高了约 60%。经过负载均衡处理后的 FP_growth 关联规则挖掘的效率更高,提高了约 14%,这说明各节点遍历任务的分配更均衡,并行化程度更高。

关键词:Spark 平台;FP_growth 算法;关联规则挖掘;频繁集;负载均衡
中图法分类号　TP311.13

Parallel FP_growth Association Rules Mining Method on Spark Platform

ZHU An-qing[1],LI Shuai[2] and TANG Xiao-dong[3]
1 School of Management,Jinan University,Guangzhou 510000,China
2 School of Computer Science and Engineering,Beihang University,Beijing 100191,China
3 School of Economics and Management,South China Normal University,Guangzhou 510006,China

Abstract　In order to improve the efficiency of association rule mining,a parallel FP_growth association rule mining method suitable for spark platform is proposed. First,the Spark platform is used to complete the traversal scan operation in the memory RDD of all nodes of the distributed system to obtain frequent sets in order to generate FP_Table and update FP_Tree. Then,the time series is introduced to predict the itemsets to be mined,so that all nodes in the distributed system can share the mining tasks in a balanced manner,so as to make full use of the traversal FP_Tree calculation function of each node to obtain the FP_growth association rule mining results. The experimental results show that compared to the single machine case,the parallelized FP_growth association rule mining improves the efficiency by about 60%. After the load balancing process,the mining efficiency of the FP_growth association rule is higher,increasing by about 14%,which indicates that the traversal task allocation of each node is more balanced and the degree of parallelism is higher.

Keywords　Spark platform,FP_growth algorithm,Association rules mining,Frequent sets,Load balancing

关联规则挖掘技术是数据挖掘的常用技术方法之一。关联规则挖掘的关键技术是通过各种算法得到频繁集[1-2],通过频繁集与支持度进行规则关联。FP_growth 算法就是频繁集生成技术的一种,在很多领域得到了广泛应用,基于 FP_growth 的关联规则挖掘的研究成果较多,主要原因是该算法通过扫描数据集与遍历 FP_Tree 能够获得满意的结果[3]。但是,FP_growth 算法在大数据关联规则应用方面存在不足,主要集中在挖掘效率方面,因此如何提高 FP_growth 关联规则挖掘的效率成为新的问题。

先进的分布式计算系统可以轻松完成各种结构类型数据的分布处理,适用于大规模数据的快速处理和计算任务[4],而现阶段主要有 Hadoop 和 Apache Spark。为了使 FP_growth

到稿日期:2019-10-16　返修日期:2020-03-10　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:广州市专利技术产业化项目(201601010207);国家自然科学基金面上项目(61672077);国家重点研发计划(2017YFF0106407);2017 国家自然科学基金青年基金项目(61702026)
This work was supported by the Guangzhou Patent Technology Industrialization Project(201601010207),General Project of National Natural Science Foundation of China(61672077),National Key R&D Program(2017YFF0106407) and 2017 National Natural Science Foundation Youth Fund Project(61702026).
通信作者:朱岸青(87651566@qq.com)

关联规则挖掘在分布式计算平台上有效运行，She 等[5]提出了基于 Hadoop 的 FP-growth 关联规则并行改进算法，得到了较好的效果。但是，相比 Hadoop 平台，Spark 平台不但继承了 Map-Reduce 计算模型，而且不再需要读写分布式文件系统(Hadoop Distributed File System，HDFS)，有效解决了大规模数据并行计算的时间问题，因此在数据挖掘方面更加具有优势。此外，由于大数据的上传下载较为频繁，特别适合在 Spark 平台上进行管理，而且考虑到大数据吞吐量的问题，在用户行为数据挖掘过程中，资源交互的流畅性尤为重要。

由上述分析可以看出，将分布式多节点运算方法与 Spark 平台相结合，可以有效提高 FP_growth 关联规则挖掘的效率。因此，为了进一步提高关联规则效率，本文结合分布式计算的特点，基于 Spark 平台引入负载均衡的多节点运算策略来实现 FP_growth 关联规则挖掘。

1 FP_growth 关联规则挖掘

1.1 FP_growth 频繁集的产生

FP_growth 频繁集的产生主要通过扫描数据库来完成，下文以图 1 为例，对频繁集产生过程进行说明。首先，设数据库中有 12 组数据集合，每组集合包含 26 个字母中的若干字母，将这 12 组原始集合称为原始集，统计原始集中每个字母的个数，得到字母候选频繁集，设最小支持度为 2，那么可以得到字母频繁集。

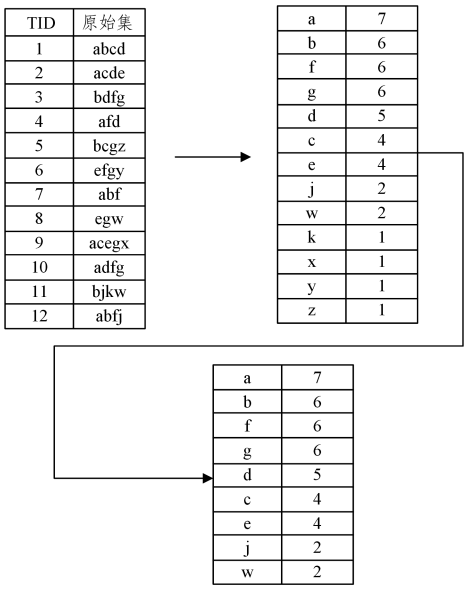

图 1 频繁集生成

Fig. 1 Frequent set generation

FP_growth 的构造流程为：首先对数据进行统计，获得每个字母在所有事务标签(Transaction ID，TID)中的统计个数；然后建立二维表[4]，从第一个 TID 开始，对每个字母进行计数，并将计数个数存至哈希表中；最后进行第二个 TID 统计，若第二个 TID 的字母在哈希表中已有记录，则直接将记录中该字母的计数个数更新，若第二个 TID 的字母在当前哈希表中无记录，则将该字母添加至哈希表的末尾，并将计数个数存至哈希表中。依据此操作，遍历所有 TID。

根据支持度要求，删除低于设定的支持度的字母及所对应的计数个数[6]，并更新哈希表，同时根据计数个数从多至少的顺序对哈希表的行重新排序，生成新的哈希表，即 FP_Table。这样可以通过序号及字母检索查询到各字母所对应的计数个数。

1.2 FP_growth 的关联规则挖掘

FP_growth 算法在对 FP_Tree 实现第一次遍历后，对原始集合的所有项进行计数，根据计数结果与设置的最小支持度，对原始集合进行整理，整理之后便可得到去掉了非频繁集合的数据集合。下文以 7 棵 FP_Tree 为例，描述 FP_growth 的关联挖掘过程，如表 1 所列。

表 1 示例表格

Table 1 Example table

树编号	原始集合	FP_growth 整理后的集合
T_1	I_1, I_5, I_6	I_1, I_6, I_5
T_2	I_2, I_1, I_6, I_8	I_1, I_2, I_6, I_8
T_3	I_2, I_1, I_5, I_6	I_1, I_2, I_6, I_5
T_4	I_2, I_1, I_4, I_9	I_1, I_2, I_4
T_5	I_1, I_3, I_4, I_{10}	I_1, I_4, I_3
T_6	I_2, I_3, I_7, I_4	I_2, I_4, I_3
T_7	I_2, I_5, I_6, I_8	I_2, I_6, I_5, I_8

表 1 中，对 7 棵 FP_Tree 进行第一次遍历时，对集合中的项进行计数统计，其中 I_7，I_9 和 I_{10} 的计数均为 1，小于设定的最小支持度 2，因此经过 FP_growth 整理后的集合删除了这 3 项，对 FP_Tree 进行了更新，并且根据其支持度大小进行了递减排序。因此，在第一次遍历后得到的项头表如表 2 所列。

表 2 项头表

Table 2 Header table

项	支持度
I_1	5
I_2	5
I_6	4
I_4	3
I_5	3
I_3	2
I_8	2

再次扫描数据库，对频繁集进行统计，如扫描到 T_2 时，$\langle I_2, I_1\rangle$，$\langle I_2, I_6\rangle$，$\langle I_2, I_8\rangle$，$\langle I_1, I_6\rangle$，$\langle I_1, I_8\rangle$，$\langle I_6, I_8\rangle$的各项计数加 1。根据遍历结果可以得到计数矩阵[7-8]。根据计数矩阵，可以得到每个项的频繁集及对应的支持度值，根据频繁集及支持度得到关联规则挖掘结果。

2 Spark 平台的关联规则挖掘

2.1 Spark 平台的关联规则挖掘流程

Spark 平台是继 Hadoop 平台之后应用于大数据计算的重要平台，在处理大规模数据时具有明显优势[9]。其与 Hadoop 平台一样，采用 Map-Reduce 计算模型来完成数据的高效处理。Spark 平台将迭代运算放在内存中完成，提高了计算过程中的数据存取效率[10]。

Spark 平台能够在内存中实现迭代，其最重要的原因是采用了弹性分布式数据块(RDD)，这也是 Spark 平台最大的优势。分布式系统的所有节点内存都承载着数据存储的功

能，不需要在外部存储器之间进行来回的数据存取，从而提高了数据的处理速度。在实际的频繁集扫描操作过程中，数据集的遍历及扫描过程均在 Spark 平台的所有节点内存的 RDD 中完成，而 FP_growth 关联规则挖掘过程的大部分运算也基本在这两个步骤中完成，那么挖掘过程中运算耗时最长的遍历及扫描都在内存中完成，提高了数据的输入输出效率。因此，基于 Spark 平台的 FP_growth 关联规则挖掘算法更有时间优势。

基于 Spark 平台的 FP_growth 关联规则挖掘的流程如图 2 所示。

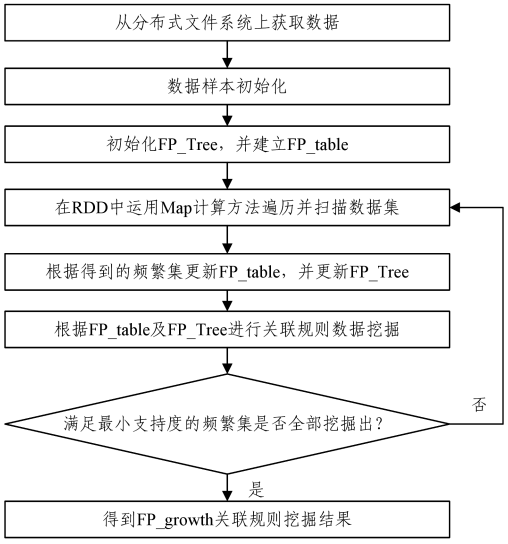

图 2 Spark 平台上 FP_growth 关联规则挖掘的流程

Fig. 2 FP_growth association rules mining process on Spark platform

在图 2 所示的流程中需要计算支持度并将其保存在 RDD 中。并行化执行 FP-Growth 的关键代码如下：

```
val g=f.flatMap(line=>{
      var l=List[(String,Int)]()
      for (i<-line._1) {
        l=(i,line._2)::l
      }
      l
    }
val d_result=f_list.flatMap(t=>{
      fp_growth(t._2,support_num,t._1 * g_size+1 to (((t._1+
      1) * g_size) min g_count))
})
```

通过上述功能代码可以有效利用 Spark 平台的内存计算优势，解决了原有 FP_growth 关联规则由于不断递归挖掘 K 频繁项集而造成的过量 IO 进程。

2.2 Spark 平台各节点的负载均衡策略

在 Spark 平台中，利用分布式系统中的多节点技术来并行化 FP_growth 关联规则挖掘，以提高挖掘效率。为了尽可能提高并行挖掘效率，本文采用负载均衡策略，保证各节点能够均衡分担 FP_growth 算法中的遍历任务，发挥每个节点参与运算的潜能。具体的实现方法如下[11]：设负载数量为 n，在 t 时间段内建立负载监测模型，负载的变化在时间轴上是无规律的，分析接入 Spark 平台的负载可以在时间序列上对遍历的负载数量进行预判，根据上一时间段待遍历的 FP_Tree 的实际数量和预测数量来建立当前时间段的预测模型，通过不断修正权重来逼近实际数量。

设负载数量为 n，$M(t)$ 为 t 时间段内的负载数量预测值，$m(t)$ 为 t 时间段内的实际负载数量，λ 为权重因子。可以根据式(1)计算下一时间段的负载数量预测值[12]。

$$M_{t+1}=M_t+\lambda(m_t-M_t) \tag{1}$$

设 ε 为相邻两次负载数量预测的变化值，则有：

$$\varepsilon=M_k-M_{k-1} \tag{2}$$

为了解决单次计算带来的误差，本文采用多次预测值求平均的方法来增加预测精度，计算式如式(3)所示[13]：

$$\overline{M}=\frac{\sum_{k=1}^{n}M_k}{n},k=1,2,\cdots,n \tag{3}$$

根据 $|\varepsilon|$ 和 $\overline{M}$ 的差值来判断负载量的变化情况，当 $|\varepsilon|\geqslant\overline{M}$ 时，表示当前时间段内该节点负载变化明显，延续上一时间段的各节点负载时间片设置策略；当 $|\varepsilon|<\overline{M}$ 时，表示该节点负载变化小，需要采取措施及时调整该节点的负载时间片策略，适当增加其负载量。

3 实例仿真

为了验证 FP_growth 关联规则挖掘算法的性能，本节进行实例仿真。实例仿真分别在单机和 Spark 分布式节点上进行。

Spark 分布式节点共计 10 个，包含 1 个控制节点和 9 个普通节点。Spark 平台由 scala 编写，10 个节点可以选择性地参与遍历运算，且 10 个节点的硬件配置相同。

挖掘数据来源于某大型电商平台，选取的 5 个数据文件及大小分别为：Data1(637.2 KB)，Data2(241.75 MB)，Data3(536.25 MB)，Data4(3.79 GB)，Data5(10.21 GB)。

3.1 Spark 平台的加速比仿真

为了验证 Spark 平台的加速性能，对基于 Spark 平台的 FP_growth 关联规则挖掘与单机的 FP_growth 关联规则挖掘的速度进行比较，求解其加速比，计算式为[14-15]：

$$S=\frac{T_a}{T_s} \tag{4}$$

其中，T_a 及 T_s 分别表示基于单机和 Spark 分布式节点的 FP_growth 关联规则挖掘的时间。

采用基于 Spark 平台的均衡负载 FP_growth 算法和单机 FP_growth 算法分别对 5 个数据文件进行关联规则挖掘[16]，并对 Spark 平台参与运算的分布式节点数进行差异化设置，设置最小支持度为 0.2，不同大小文件及不同节点数的加速性能统计结果如表 3 所列。从表 3 可以看出，随着参与计算节点数的增加，加速效果逐渐增加，表示参与遍历计算的节点数越多，分布式计算的优势就越明显，相比单机，Spark 平台的加速优势更明显。当样本集文件不大时，加速比接近 1.000，表明此时采用 Spark 平台和单机的速度差异并不明显，只有当样本文件较大时，如 Data4 和 Data5，两种方法的速度差异较大，当参与节点数为 10、文件大小为 10.21 GB 时，加速比达到了 27.737，基于 Spark 平台的速度远快于单机的速度。

表 3 Spark 平台的加速性能

Table 3 Acceleration performance of Spark platform

样本集	文件大小	参与计算的节点数	加速比
Data1	637.2 KB	1	1.000
		2	1.001
		3	1.001
		5	1.001
		10	1.001
Data2	241.75 MB	1	1.000
		2	1.004
		3	1.008
		5	1.011
		10	1.011
Data3	536.25 MB	1	1.000
		2	1.312
		3	5.375
		5	9.031
		10	12.772
Data4	3.79 GB	1	1.000
		2	1.682
		3	7.923
		5	9.937
		10	13.147
Data5	10.21 GB	1	1.000
		2	3.152
		3	9.427
		5	14.362
		10	27.737

3.2 关联规则挖掘的时间性能仿真

为了更进一步地对 FP_growth 关联规则挖掘的时间性能进行全方面测试，对不同支持度和信任度下的时间性能进行仿真。

3.2.1 不同支持度下的时间性能

在不同支持度等级下，将最小信任度固定为 0.5[17]，分别对单机 FP_growth、Spark 平台 FP_growth 和 Spark 平台的负载均衡 FP_growth 这 3 种方法的运行时间进行仿真，结果如表 4 所列。

表 4 不同支持度下各算法的运行时间

Table 4 Running time of each algorithm under different support levels

(单位：ms)

支持度/%	单机 FP_growth	Spark 平台 FP_growth	负载均衡的 Spark 平台 FP_growth
50	47	33	32
40	55	41	39
30	63	52	47
20	81	62	60
10	117	86	82
5	356	273	269
1	783	619	611
0.1	3347	2131	1792
0.01	4923	3017	2633

从表 4 可以看出，当支持度阈值逐渐降低时，3 种算法的关联规则挖掘时间逐渐增加，这主要是因为满足最小支持度的项目集增多，需要耗费更多的时间来遍历 FP_Tree。但通过比较发现，相比单机 FP_growth 关联规则挖掘，基于 Spark 平台的 FP_growth 关联规则挖掘的运行时间明显缩短，当支持度阈值为 0.01%时，运行时间缩短了 63.18%。而引入了负载均衡的 Spark 平台 FP_growth 关联规则挖掘的运行时间被进一步缩短，当支持度阈值为 0.01%时，相比未经过负载均衡的运行时间缩短了 14.58%。

3.2.2 不同信任度下的时间性能

在不同信任度等级下[18-19]，将最小支持度固定为 0.2，分别对单机 FP_growth、spark 平台 FP_growth 和 spark 平台的负载均衡 FP_growth 这 3 种方法的运行时间进行仿真，结果如表 5 所列。

表 5 不同信任度下各算法的运行时间

Table 5 Running time of each algorithm under different trust levels

(单位：ms)

支持度/%	单机 FP_growth	Spark 平台 FP_growth	负载均衡的 Spark 平台 FP_growth
70	36	28	25
60	67	53	51
50	81	62	60
40	126	99	92
30	347	265	249
20	723	604	589
10	1173	893	772
5	2921	2074	1982
1	4278	2761	2439

从表 5 可以看出，当信任度阈值逐渐降低时，3 种算法的关联规则挖掘时间逐渐增加，这主要是因为满足最小信任度的项目集增多。通过对比可知，相比单机 FP_growth 关联规则挖掘，基于 Spark 平台的 FP_growth 关联规则挖掘的运行时间明显缩短，当支持度阈值为 1%时，运行时间缩短了 54.94%。而引入了负载均衡的 Spark 平台 FP_growth 关联规则挖掘的运行时间被进一步缩短，当支持度阈值为 1%时，相比未经过负载均衡的运行时间缩短了 13.20%。

综上所述，在支持度和信任度相同的条件下，采用了负载均衡的 Sprak 平台 FP_growth 关联规则挖掘在运行时间方面具有明显优势，这是由于采用了负载均衡的 Sprak 平台中的 Job 内存机制，因此延迟较低。

3.3 均衡性能仿真

为了检验分布式系统中各节点参与遍历运算的有效性，对分布式系统中各节点的运行时间进行性能仿真。实验数据来自于某电商平台的用户购买记录，通过对用户历史购买数据进行分析，来实现用户购买习惯的关联规则挖掘。设定最小支持度为 0.2，对不同的用户消费数据集进行实例仿真。选取 5 个分布式节点，分别对 FP_growth 算法(FP)和负载均衡 FP_growth 算法(LBFP)进行仿真，结果如表 6 所列。

表 6 不同挖掘算法的各节点的运行时间

Table 6 Running time of nodes with different mining algorithms

(单位：ms)

数据集	算法	节点编号					频繁项数
		1	2	3	4	5	
500 MB	FP	22	31	41	24	37	5
	LBFP	23	24	23	22	24	5
1 GB	FP	77	88	109	94	112	6
	LBFP	85	83	85	89	86	6
2 GB	FP	167	182	202	191	177	8
	LBFP	164	170	168	171	166	8
5 GB	FP	477	484	463	492	505	8
	LBFP	422	420	424	425	422	8
10 GB	FP	1033	976	974	987	992	12
	LBFP	868	871	869	868	869	12

从表 6 可以看出，FP 算法和 LBFP 算法在最小支持度为 2 的情况下得到的频繁项相等，两种算法均能达到数据挖掘的要求。但相比 FP 算法，LBFP 算法在挖掘过程中各节点所承担的遍历运算任务更均衡，且这种均衡特性不随数据集的大小而发生变化，比较稳定。

结束语 本文采用 Spark 平台实现并行化 FP_growth 关联规则挖掘，可以有效地提高大规模数据挖掘的效率。而采用负载均衡策略可以进一步优化各节点的效率，提高并行化程度。但是，现阶段的研究仅在运行时间方面进行了测试，而数据挖掘的准确率无明显的提升。后续将针对如何在提高运行效率的同时进一步增强准确率展开讨论。

参 考 文 献

[1] BELALEM G, ABBACHE A, BELKREDIM F Z, et al. Arabic Query Expansion Using WordNet and Association Rules[J]. International Journal of Intelligent Information Technologies, 2016, 12(3): 51-64.

[2] MAI T, VO B, NGUYEN L T T. A lattice-based approach for mining high utility association rules[J]. Information Sciences, 2017, 399: 81-97.

[3] YU B, LIU S Q. An improved association rule mining algorithm based on FP-growth algorithm [J]. Computer and Network, 2017, 43(14): 68-71.

[4] LIN W T, CHU C P. Determining the appropriate number of nodes for fast mining of frequent patterns in distributed computing environments [J]. Parallel Algorithms & Applications, 2014, 30(5): 1-13.

[5] SHAO X Y, ZHANG L. An improved parallel algorithm for FP-Growth association rules based on Hadoop [J]. Computer Applied Research, 2018, 35 (1).

[6] DIVYAVARMA K, REMYA M, DEEPA G. An Enhanced Bug Mining for Identifying Frequent Bug Pattern Using Word Tokenizer and FP-Growth[M] // Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications. 2017: 525-532.

[7] MOHAMED Y S, NAJIB M, ABDELAZIZ E, et al. APRICOIN: An adaptive approach for prioritizing high-risk containers inspections[J]. IEEE Access, 2017, 5(99): 18238-18249.

[8] XU F, LU H. The Application of FP-Growth Algorithm Based on Distributed Intelligence in Wisdom Medical Treatment[J]. International Journal of Pattern Recognition & Artificial Intelligence, 2017, 31(4): 232-237.

[9] CHEN J G, LI K L, MEMBER S, et al. A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment[J]. IEEE Transactions on Parallel & Distributed Systems, 2017, 28(4): 919-933.

[10] SHI W, ZHU Y, YU P S, et al. Effective Prediction of Missing Data on Apache Spark over Multivariable Time Series[J]. IEEE Transactions on Big Data, 2018, 4(4): 473-486.

[11] ZHANG L Z, CUI Y, LUO G C, et al. Dynamic Load Balancing Algorithms for Large Data Distributed Storage [J]. Computer Science, 2017, 44(5): 178-183.

[12] LI Z Y, YU J, BIAN C, et al. Data flow dynamic load balancing strategy based on load perception [J]. Computer Applications, 2017, 37(10): 2760-2766.

[13] LUO J, LEI R, XUE L. Spatio-Temporal Load Balancing for Energy Cost Optimization in Distributed Internet Data Centers[J]. IEEE Transactions on Cloud Computing, 2017, 3(3): 387-397.

[14] LI X, LI T. Design and implementation of recommendation system based on Spark [J]. Computer technology and development, 2018, 28(10): 201-205.

[15] MESTRE D G, PIRES C E S, NASCIMENTOD C, et al. An efficient spark-based adaptive windowing for entity matching[J]. Journal of Systems & Software, 2017, 128: 1-10.

[16] CHEN J G, LI K L, MEMBER S, et al. A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment[J]. IEEE Transactions on Parallel & Distributed Systems, 2017, 28(4): 919-933.

[17] YAN Y L, CHEN M, SADIQ S, et al. Efficient Imbalanced Multimedia Concept Retrieval by Deep Learning on Spark Clusters [J]. International Journal of Multimedia Data Engineering & Management, 2017, 8(1): 1-20.

[18] CAO N, WANG C, LI M, et al. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data [J]. IEEE Transactions on Parallel & Distributed Systems, 2014, 25(1): 222-233.

[19] HU J, PAN H A. Improved incremental updating algorithm of association rules[J]. Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition), 2018, 30(4): 558-563.

ZHU An-qing, born in 1976, Ph.D, associate professor. Her main research interests include internet of things and enterprise management.