K-频繁项集挖掘并行化算法

1. 设计思路

本题目的要求是在规定的标准数据集上进行频繁项集的挖掘，因而自然而然地想到了两种著名的算法：Apriori算法和FP-Growth算法。由于测试数据集约含1800万条事务记录，预赛时考虑到Apriori算法在计算过程中需要重复扫描数据集（如题要求1-频繁项集到8-频繁项集的挖掘，则需要扫描8遍数据集）以及产生大量的候选集，而FP-Growth算法只需进行2遍数据集的扫描，以将事务记录压缩成树来推导频繁项集的方式来代替候选集的生成，效率更高。因而最终决定采用并行化的FP-Growth算法来解决这一问题。

然而在决赛中，作为对比，尝试实现了Spark上的Apriori算法，发现其运行速度要远远超过并行化的FP-Growth算法。经查找原因，发现是由于数据集的不稀疏性，经过FP-Growth的Map过程，原数据集并不能被划分为一个个大小均等的小数据集，往往被划分后的最后一个子数据集的大小与原数据集基本相同；同时在对生成的FP-Tree迭代挖掘的过程中很难做到挖掘频繁项集的并行化。而Apriori算法在可并行化的程度上要大大超过FP-Growth算法，而且在不同稀疏性的数据集下的运行效率基本一致。因而在决赛上提交的程序最终选定为并行化的Apriori算法实现。

2. 实现方案

本算法的实现主要分为以下两个步骤：

2.1 挖掘1-频繁项集

利用简单的类单词计数算法求出不同项的出现频数，并根据支持度进行过滤，将过滤后的1-频繁项集存储到哈希表oneItemHS中。根据oneItemHS哈希表对原始数据中的每一条事务记录进行过滤，仅保留包含在该哈希表中的项，最终形成filteredTransactions。

2.2 递归用K-频繁项集生成(K+1)-频繁项集

1. 将之前生成的K-频繁项集lastItemSet两两union，distinct之后选择长度为(K+1)的候选项并作剪枝生成候选项集（将候选项集中的每条候选项转换成由逗号分隔的字符串）。

注：剪枝的依据为若生成的(K+1)长度的候选项的某子集不曾出现在K-频繁项集lastItemSet中，则该候选项被剪枝。

2. 根据上一步生成的候选项集，利用类似2.1的单词计数算法求出每条候选项在filteredTransactions中的出现频数，并根据支持度进行过滤，将过滤后的(K+1)-频繁项集存储到lastItemSet中。

3. 递归执行上述两步。

3. 测试结果

3.1 jar包使用说明

spark-submit

—class edu.seu.cloud.jn3

—master <master-url>

jn3.jar

input-path

output-path

3.2 任务提交

```
[bookcold@namenode mathpanda]$ hadoop dfs -rmr /user/mathpanda/jn3
Picked up _JAVA_OPTIONS: -Xms30g -Xmx80g
Picked up _JAVA_OPTIONS: -Xms30g -Xmx80g
Deleted hdfs://namenode:9000/user/mathpanda/jn3
[bookcold@namenode mathpanda]$ nohup ../spark-1.0.1/bin/spark-submit --class edu
.seu.cloud.jn3.Main  --master spark://192.168.1.18:7077 ./jn3.jar /user/mathpand
a/apriori_data.bat /user/mathpanda/jn3 &
[1] 15603
[bookcold@namenode mathpanda]$ nohup: 忽略输入并把输出追加到"nohup.out"

[bookcold@namenode mathpanda]$
```

3.3 运行情况

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150324221812-0006	JN3	192	40.0 GB	2015/03/24 22:18:12	bookcold	FINISHED	15 min

minPartitions = 192时，Duration = 15min

3.4 执行结果