

《MapReduce海量数据处理》

实验5 频繁项集挖掘

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 计算机科学与技术系 | | | | | |
| **小组成员** | **：** | 林喜鹏 李朝阳 饶璐 唐玉婷 | | | | | |
| **学号** | **：** | MG1833046 MF1833035 MF1833059 MF1833070 | | | | | |
| **小组组长** | **：** | 林喜鹏 | | | | | |
| **时间** | **：** | 2018 | 年 | 12 | 月 | 15 | 日 |

1. 实验要求

**实验背景**

频繁项挖掘或者关联规则挖掘问题是数据挖掘中的基本问题，一个经典的实例就是购物篮问题。超时或者网店可以根据顾客的购买记录进行频繁项或关联规则挖掘，从而发现客户的购买习惯。比如说，购买产品X的同时购买Y，于是可以根据这种购买习惯进行货架调整，将关联度高的商品放在一起以提高销售量。

**实验任务**

Apriori算法是频繁项集挖掘中的经典算法。Apriori算法经过多伦迭代的方式来逐步挖掘频繁项集。在第一轮迭代中，计算事务数据中每一项的支持度并找出所有频繁项。在之后的每轮迭代中，将前一轮生成的频繁k-项集作为本轮迭代的种子项集，以此来生成候选k+1项集，这些候选项集在整个事务数据集中可能是频繁的，也可能不是，在本轮迭代中，需要计算每个候选k+1项集在事务数据中的实际支持度，来找出全部的k+1频繁项集将其作为下一轮的种子项集。这样子一直迭代下去，直到不再产生新的频繁项集为止。

**输入要求**

输入数据仅为一个文件。该文件由若干行组成，每一行代表一个事物，由用空格分割的整数组成：

A B C D…

最小支持度则是0-1以内的小数。

**输出要求**

请输出在最小支持度min\_supp下的所有极大频繁项集（包括给出极大频繁项集的项数及其支持度）。请在实验结果压缩包中包含所有结果。

**其他要求**

本次实验要求自己动手搭建Spark环境（可采用单机伪分布式模式）。语言可以使用python,scala,java等Spark支持的语言。

**提交格式**

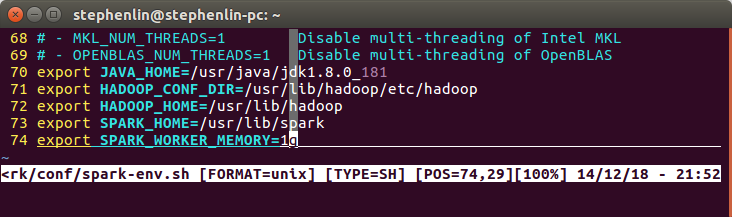
压缩包,包括:源代码、JAR 包、JAR 包执行说明及实验报告。

注意：在代码中，我是按照维基百科的定义把实验输入的最小**支持度**(0-1的小数)乘以总样本数(也即最小支持项数)称为**最小支持度**；实验输入的最小支持度在注释中称为**置信度。**

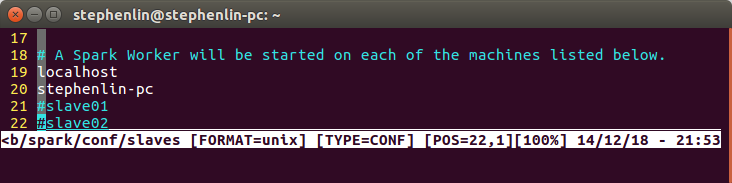
1. Spark环境搭建
2. 下载Spark：由于以后可能需要使用Spark,所以下载的是最新的2.4.0版本,具体链接为:

<https://spark.apache.org/downloads.html>

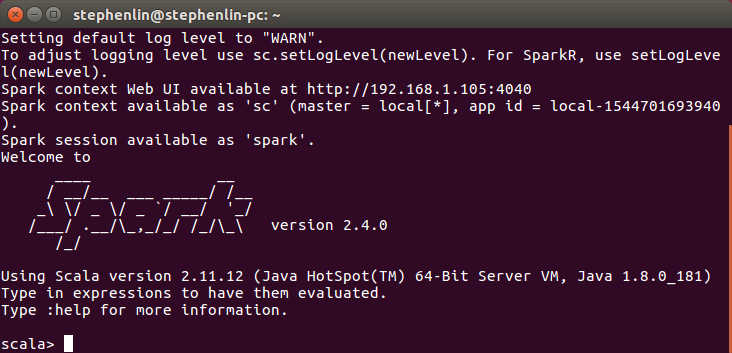
1. 解压到安装目录，并配置spark\_env.sh文件如下(主要配置环境变量):



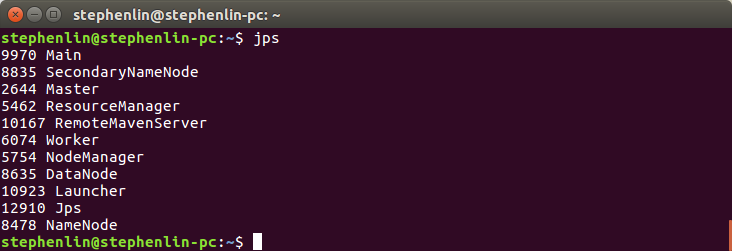
1. 配置slaves文件(stephenlin-pc是我主机名字):



1. 当然还要在.bashrc中配置Spark的环境变量,配置后运行spark-shell:



1. 启动spark以及yarn伪分布式模式,启动后使用jps查看:



可以看到都已经正确启动！

1. 算法设计思路及各个类设计说明

算法的设计思路其实和MapReduce基本差不多,只不过Spark提供了JavaRDD(Scala熟练度不够,本次实验我仍旧采用的是Java)可以很方便地进行各类“map”和“reduce”操作；

首先，对于输入文件，我们把每一行按空格分得每个项（数字）并拆分为多行，每个项单独一行，然后把每一项**map到[item,1]**

再按照键值item进行**reduce得到[item,n]**,也就得到了每个项的项数,接着把**小于最小支持度的项过滤(filter)掉**,然后对剩下的项**进行两两求并集并找出项数恰好增大1的项集作为候选集,**然后把候选集和原来的数据集进行比对即可得到新的项集,**并返回一个布尔值**foundItemSet**表示成功找到新的项集**,重复这个过程直到返回的布尔值为False。

具体的各个函数的设计说明如下：

1. Mining()：最外层循环,当返回的布尔值foundItemSet为False时停止挖掘；
2. run()：挖掘频繁1-项集,并得到后面计算频繁多项集的候选集；
3. run(set, k): 根据k-1项集挖掘k项集，跟run()基本相同，只不过需要先得到实际存在的候选k项集；
4. getKItemSet(set)：根据频繁k-1项集两两连接生成可能的频繁k项集；
5. getItems(set,line)：对照原来的数据集的每一行找到实际存在的候选频繁k项集；
6. 主要代码
7. 初始化和Mining函数:



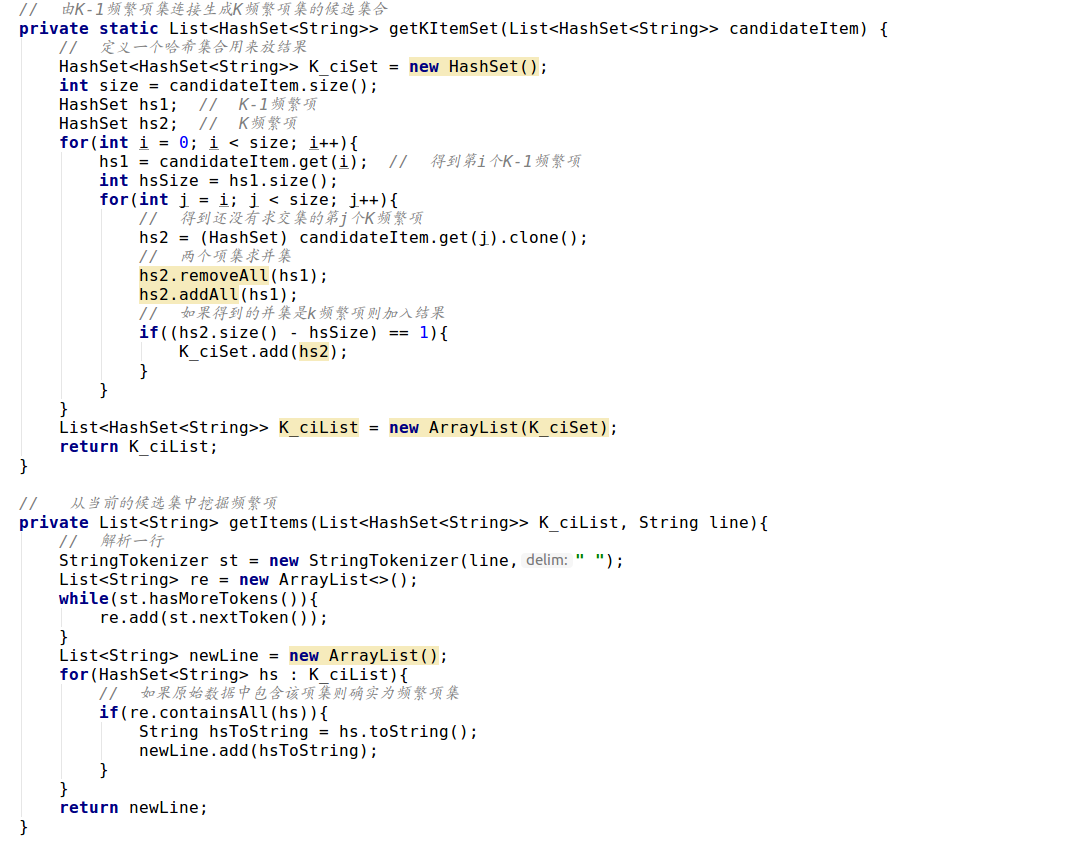
1. 挖掘频繁1项集run()：



1. 挖掘频繁k项集run(set, k):



1. 得到候选频繁k项集getKItemSet(set)和getItems(set,line):

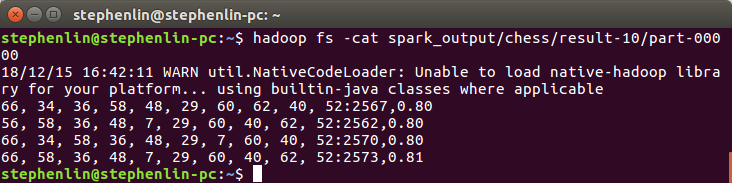


1. 实验结果(输出文件截图)

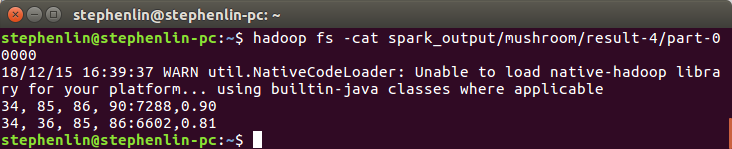
格式是：a b c…d:x,y

其中a b c…d是极大频繁项,x是项数,y是支持度;

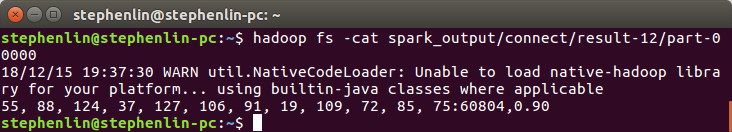
1. Chess数据集:



1. Mushroom数据集：



1. Connect数据集：



1. 程序运行性能的分析

程序在chess和mushroom数据集上的表现还是不错的，运行时间不到1分钟，但是在connect数据集上比较慢，具体运行时间如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | Chess | Mushroom | Connect |
| 运行时间 | 52s | **13s** | 1.6h |

表1 程序在各个数据集上运行时间

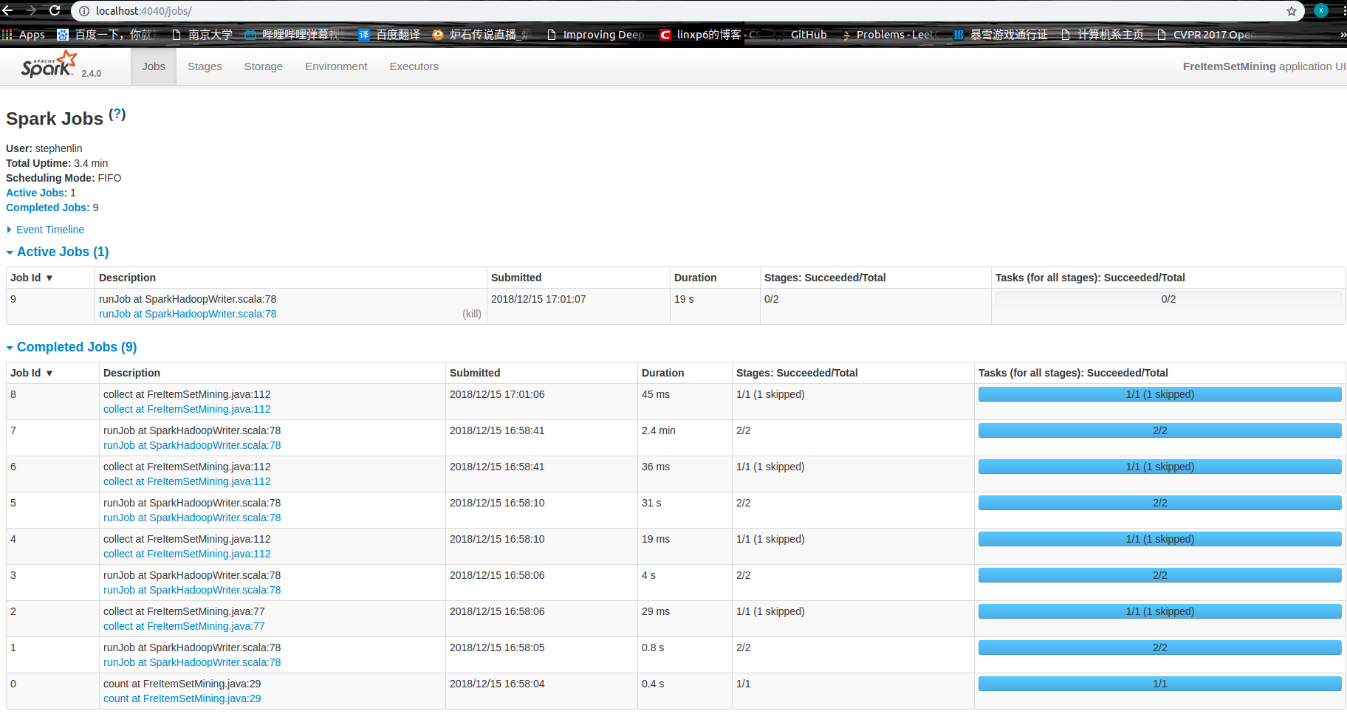
分析：

首先，程序性能的瓶颈**主要集中在得到候选频繁k项集,**因为需要和整个数据集进行比较，所以对于比较大的数据集connect，算法的运行时间就会大大增加，因为要比较的原始数据项太多了，导致算法的运行效率低下；

1. 性能不足与可能改进之处
2. 程序在大数据集connect上运行时间过长；

可能的改进方案：对于大数据集，可以采取**采样的方法**，先对大数据集做多次采样，然后每次不用与全部数据进行比对，而与所采样本进行比较即可，**只要得到的候选集存在并在所有样本的平均支持度大于所要求的最小支持度即认为是合法项集**。根据大数定理，这样改进算法只有极小的概率得不到最优解；

1. WebUI执行报告截图



1. 实验总结

本次实验主要是使用Spark实现Apriori算法并在三个数据集上进行实验。实验难点主要集中在学习Spark的RDD提供的各类高效的方法进行数据处理，算法的设计本身其实与MapReduce基本一致；

通过本次实验，我们进一步对Spark的编程有了更进一步的学习，同时也对如何利用Spark进行算法设计，从而处理更大数据集有了非常深刻的理解。