中山大学2018年本科生实验报告

课程名称:*云计算*

授课教师：王昌栋

院系与专业	数据科学与计算机学院（移动信息工程）
实验人	蔡政、曹广杰、崔博彦
学号	**15352014、15352015、15352068**
组长	蔡政（**15352014**）

实验题目

MapReduce实现Apriori算法

实现内容

1. 在`hadoop`平台上实现`Apriori`算法；
2. 使用`MapReduce`架构；
3. 参照基于Apriori算法的关联规则挖掘以及改进[1]进行实现

算法介绍：

Apriori算法用于识别几个项集中的最大频繁项集——所谓最大频繁项集表示在所有项集中出现的次数超过一个阈值（最小支持度）的项的集合，最大频繁项集可以用于实现斜决策树等有监督学习分类器。Apriori算法是一种最有影响的挖掘关联规则频繁项集的算法,它的主要特点是利用频繁项集的先验知识,使用一种称作逐层搜索的迭代方法,K一项集用于探索(K+1)一项集,找每个频繁K一项集都需要一次数据库扫描。[1]

定义2.5

若干项的集合,称为项集。项集中所包含的项的个数称为项集的长度,长度 为K的项集称为K一项集。包含项集的事务数,称为项集的频率或支持计数。 项集中各项按字典次序排列,每个项集有一个Count域,用于保存该项集的支持计数, 其初始值为零。

定义.26

当项集的支持计数大于或等于min-sup与D中事务总数|D|的乘积时,称为 频繁项集(frequent item set),当项集的支持计数可能大于等于min-sup x |D|时,称为 侯选项集。

Hadoop的MapReduce架构可以切分成一系列运行于分布式集群中的map和reduce任务，每个任务只运行全部数据的一个指定的子集，以此达到整个集群的负载平衡。Map通常为加载，解析，转换，过滤数据，每个reduce处理map输出的一个子集。Reduce任务会去map任务端copy中间数据来完成分组，聚合。

由于时间关系，笔者仅仅实现了该论文中绪论中的基础关联规则算法，并没有实现核心的算法。

实验的实现过程[2]

KNN模型在应对大量的数据集的时候，只选择与测试数据相近的信息，缩小计算范围由此减少计算量——这种方式减少了与测试样本无关的数据对样本可能产生的影响。

Map过程的实现

Map的主要功能通常为加载，解析，转换，过滤数据，为每个reduce输出一个子集以便Reduce处理。Map过程将会输出中间数据，发送给Reduce部分处理。在该算法中，Map的过程主要完成的任务是分析已有的文件内容并输出中间结果。

整体结构如下：

```
public static class Map extends MapReduceBase implements
            Mapper<LongWritable, Text, Text, IntWritable> {

  public void configure(JobConf job) {
    // collect directory from files;
  }

  @Override
  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter report)
    throws IOException {
    // collect vocabularies from text;
  }

}
```

在map过程中的具体逻辑如下：

```
@Override
public void map(LongWritable key, Text value,
                OutputCollector<Text, IntWritable> output, Reporter report)
  throws IOException {

  String[] dd = line.split(",");
  // data still exist in the file;
  if(!count.equals("false")){
    for(String sd : dd){
      List<String> dstr = new ArrayList<String>();
      dstr.add(sd);
      word = new Text(dstr.toString());
      output.collect(word, one);
    }
  }
```

```
  else{
    List<String> dstr = new ArrayList<String>();
    for(String ss: dd){
      dstr.add(ss);
    }
    // after reading done this record, read next
    for(int i = 0 ; i< nextrecords.size();i++){
      if(dstr.containsAll(nextrecords.get(i))){
        word = new Text(nextrecords.get(i).toString());
        output.collect(word, one);
      }
    }
  }
}
```

可以看到，map的主要过程是根据已有的文件内容，根据标识符实现文件中数据的收集，并将收集到的数据统一添加到output中。这里的output表示的是map过程的output，即中间结果。这些中间结果的输出将会用于此后的reduce过程的实现。

Reduce过程的实现

Reduce获取上一部分的输出文件，并根据key合并，综合成一个大的数据文件。排序的目的是让key相邻，方便在reduce阶段迭代处理。需要实现key的选择和可以自定义用于分组的比较器。该算法中，reduce会根据对于key值的统计，选择不小于最小支持度的项，作为该阶段的输出。

整体结构如下：

```
public static class Reduce extends MapReduceBase implements
            Reducer<Text, IntWritable, Text, IntWritable> {
  // overload reduce function;
  @Override
  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter report)
    throws IOException {}
}

```

Reduce的函数内部逻辑如下：

```
@Override
public void reduce(Text key, Iterator<IntWritable> values,
                   OutputCollector<Text, IntWritable> output,
                   Reporter report)
    throws IOException {
    int sum = 0;
    // 使用迭代器，对key值进行统计；
    while (values.hasNext()) {
      sum += values.next().get();
    }
    // 如若大于等于最小支持度，则记录；
    if (sum >= minnum) {
      output.collect(key, new IntWritable(sum));
```

```
    }
}
```

根据MapReduce架构，统计key值是非常简单的，而且分布式运算的运算效率很高，这里的reduce就根据已经获得的中间结果，计算频繁项集的信息——根据key值的统计，选择不小于最小支持度的项，作为该阶段的输出，该输出即为最大频繁项集。

Main的调用

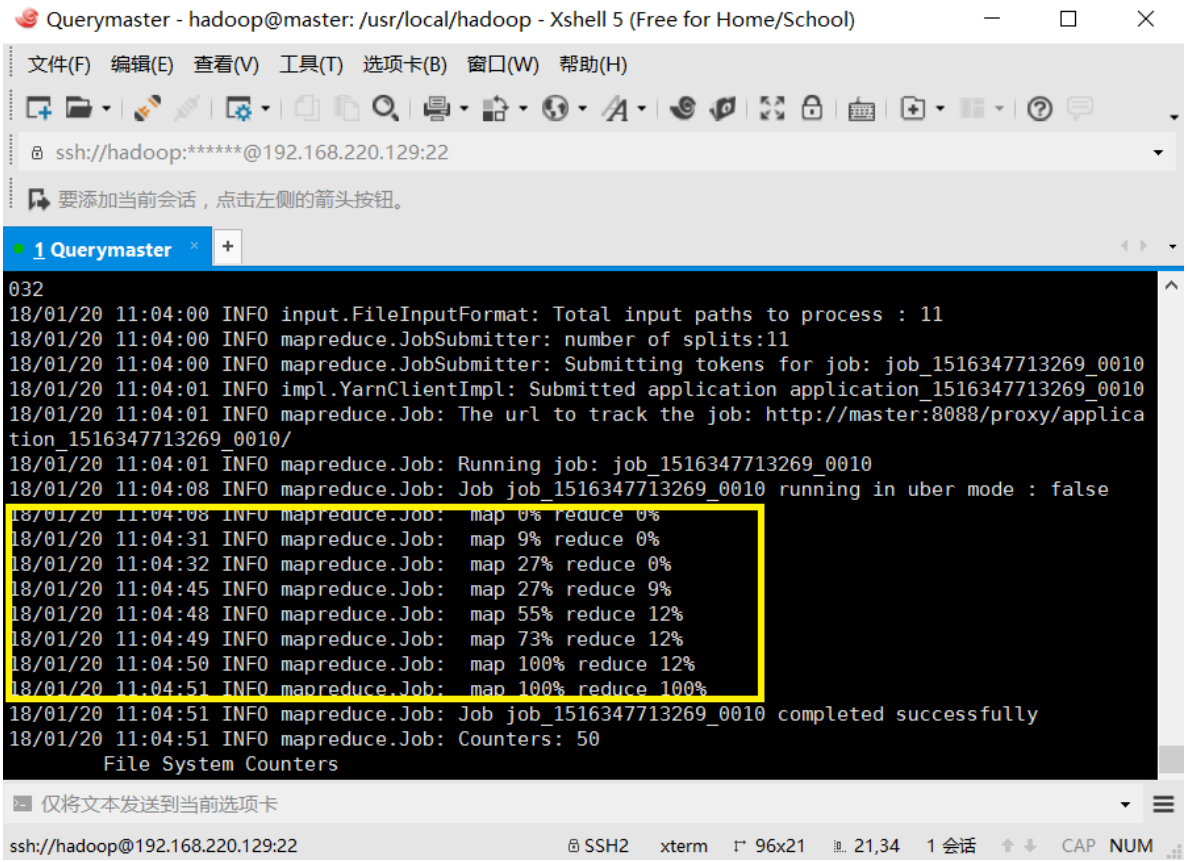
main函数对于以上参数的调用其实比较常规，因为MapReduce的架构已经确定，这里对于其二者的使用方式并不能自定义：

```
        new JobConf(getConf(), AprioriMapReduce.class);
        conf.setJobName("apriori");

        conf.setMapperClass(Map.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);

        conf.setReducerClass(Reduce.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
```

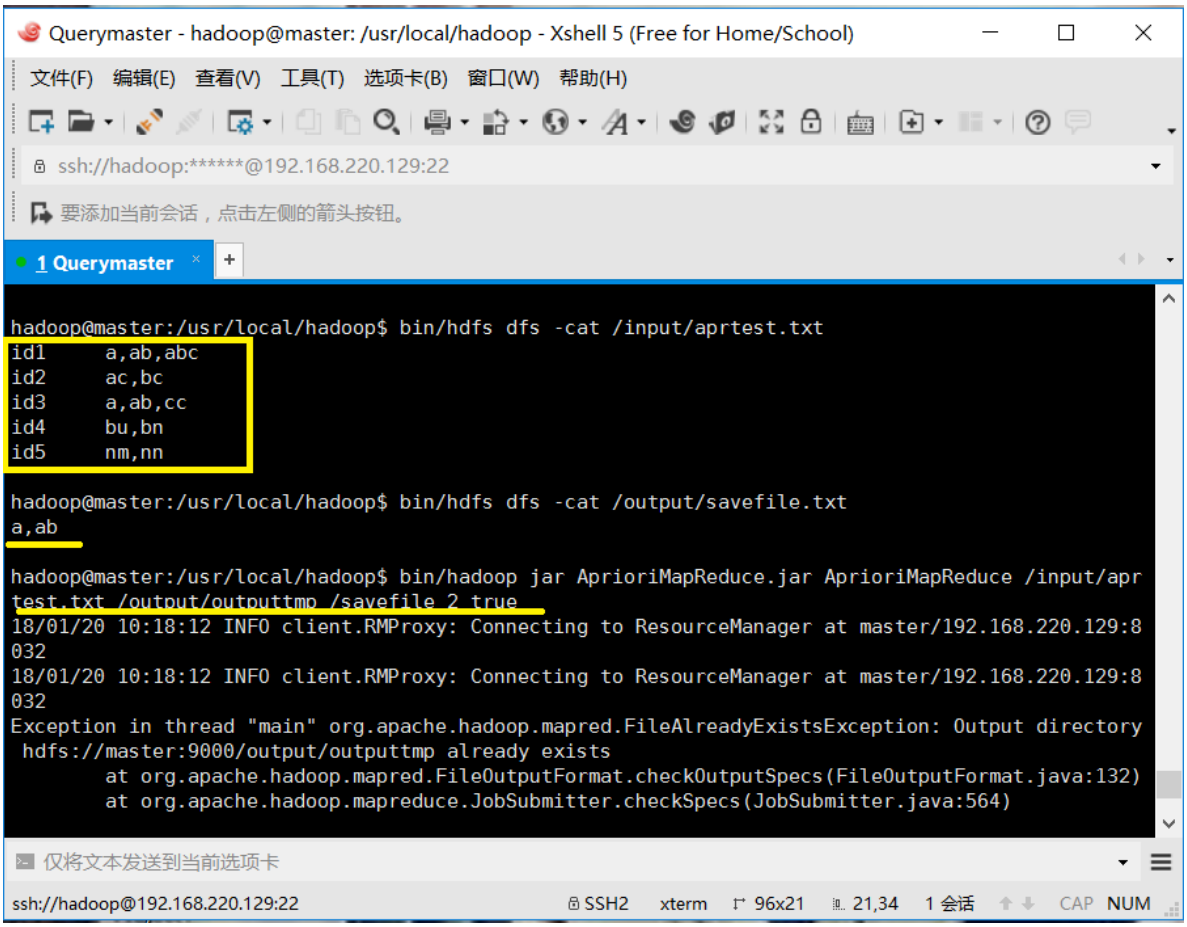
这里有一点需要注意的是，对于输入的参数，此算法中的最小支持度值需要作为一个新的参数使用，所以需要有新的参数添加进来，即需要`arg[2]`；

实验结果

运行的中间过程：


Querymaster - hadoop@master: /usr/local/hadoop - Xshell 5 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://hadoop:******@192.168.220.129:22
要添加当前会话，点击左侧的箭头按钮。
1 Querymaster
032
18/01/20 11:04:00 INFO input.FileInputFormat: Total input paths to process : 11
18/01/20 11:04:00 INFO mapreduce.JobSubmitter: number of splits:11
18/01/20 11:04:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1516347713269_0010
18/01/20 11:04:01 INFO impl.YarnClientImpl: Submitted application application_1516347713269_0010
18/01/20 11:04:01 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/applica
tion_1516347713269_0010/
18/01/20 11:04:01 INFO mapreduce.Job: Running job: job_1516347713269_0010
18/01/20 11:04:08 INFO mapreduce.Job: Job job_1516347713269_0010 running in uber mode : false
18/01/20 11:04:08 INFO mapreduce.Job: map 0% reduce 0%
18/01/20 11:04:31 INFO mapreduce.Job: map 9% reduce 0%
18/01/20 11:04:32 INFO mapreduce.Job: map 27% reduce 0%
18/01/20 11:04:45 INFO mapreduce.Job: map 27% reduce 9%
18/01/20 11:04:48 INFO mapreduce.Job: map 55% reduce 12%
18/01/20 11:04:49 INFO mapreduce.Job: map 73% reduce 12%
18/01/20 11:04:50 INFO mapreduce.Job: map 100% reduce 12%
18/01/20 11:04:51 INFO mapreduce.Job: map 100% reduce 100%
18/01/20 11:04:51 INFO mapreduce.Job: Job job_1516347713269_0010 completed successfully
18/01/20 11:04:51 INFO mapreduce.Job: Counters: 50
File System Counters
仅将文本发送到当前选项卡
ssh://hadoop@192.168.220.129:22
SSH2 xterm 96x21 21,34 1 会话
CAP NUM

运行的结果：


Querymaster - hadoop@master: /usr/local/hadoop - Xshell 5 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://hadoop:******@192.168.220.129:22
要添加当前会话，点击左侧的箭头按钮。
1 Querymaster
hadoop@master:/usr/local/hadoop$ bin/hdfs dfs -cat /input/aprtest.txt
id1 a,ab,abc
id2 ac,bc
id3 a,ab,cc
id4 bu,bn
id5 nm,nn
hadoop@master:/usr/local/hadoop$ bin/hdfs dfs -cat /output/savefile.txt
a,ab
hadoop@master:/usr/local/hadoop$ bin/hadoop jar AprioriMapReduce.jar AprioriMapReduce /input/apr
test.txt /output/outputtmp /savefile 2 true
18/01/20 10:18:12 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.220.129:8
032
18/01/20 10:18:12 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.220.129:8
032
Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory
hdfs://master:9000/output/outputtmp already exists
at org.apache.hadoop.mapred.FileOutputFormat.checkOutputSpecs(FileOutputFormat.java:132)
at org.apache.hadoop.mapreduce.JobSubmitter.checkSpecs(JobSubmitter.java:564)
仅将文本发送到当前选项卡
ssh://hadoop@192.168.220.129:22
SSH2 xterm 96x21 21,34 1 会话
CAP NUM

在每一行中，都会统计每一项出现的次数，并选择出满足最小支持度的项（这里是a和ab），在最后将其合并为一个集合，称为频繁项集。通常认为在该项集中的项都有较强的关联性，该算法就已经挖掘出了各项之间的关联规则。

总结

本次实现算法使用了MapReduce的结构，对输入的数据进行了分布式运算，主要使用了hadoop开发环境。针对关联规则挖掘算法Apriori进行了实现，为该算法设计了小数据集，并将输出结果以截图的形式展现在运行结果的部分。实验过程中，熟悉了Linux的使用命令以及MapReduce的运作结构体系。

小组分工

曹广杰：运行环境的搭建与代码的实现与修改，实验报告的撰写；

蔡政：部分代码的修改、实验报告的撰写；

崔博彦：实验报告的撰写；

---

1. [1]王培吉. 基于Apriori算法的关联规则挖掘及改进[D].内蒙古大学,2003.↩↩

2. 选自杯子K的博客：http://blog.csdn.net/sinat_33982461/article/details/52453284 ↩