

# 金庸的江湖—大数据处理综合实验报告

2019 年 7 月 20 日

小组成员:

组长:邓开圣 161220031

组员:戚赞 161220097

组员:成威 161220024



## 摘要

我们小组选做了实验二”金庸的江湖“。通过金庸的江湖通过一个综合数据分析案例:”金庸的江湖——金庸武侠小说中的人物关系挖掘“,来学习和掌握MapReduce 程序设计。通过本课程设计的学习,可以体会如何使用 MapReduce 完成一个综合性的数据挖掘任务,包括全流程的数据预处理、数据分析、数据后处理等。

本次实验我们首先利用Ansj分词包和MapReduce的结合,将每本小说中的每段的人物提取出来进行输出,从而过滤掉了大部分的无用信息,仅仅保留了小说中的人物关系的信息。输出的每一行为同一段的人名。实验二我们统计了人物的同现关系,规定同一段落出现的两个人名即为同现一次。通过人物同现的输出,我们可以清楚的看到两个人物之间的关系紧密程度。实验三将数据进行邻接图进行处理并且进行了归一化,目的是找出一个人物和其他人物的同现频率,在样本大的时候近似同现概率。方便后面PageRank和标签传播算法的展开。实验一二三可以看作数据的预处理。

实验四实现了经典的PageRank算法,PageRank算法很经典。我们很完整进行了复现,同时尝试了不同的收敛条件对于PageRank值得影响,Hadoop和Spark不同实现来进一步确定了正确性。对于结果我们还进行了排序等数据后处理输出。从而获得了正确得排名属性。其中Hadoop实现的是贡献算法,而Spark版本实现的是矩阵算法。

实验五实现了半监督学习得标签传播算法,我们分别实现了标签传播得同步实现和异步实现,比较了同步实现和异步实现得结果,同时学习到了标签传播得同步得不足——二分图震荡现象。异步的实现没有出现这个现象。从标签传播算法的编写中,我们体会到机器学习算法处理的强力。

此外我们还利用Spark进行完整的实验流程的编写,从而比较了Spark和Hadoop两者的差异,体会到基于弹性的内存计算RDD的强大之处和Spark的使用的方便性。

**关键词:** 金庸 Ansj分词 Hadoop Mapreduce Spark PageRank 标签传播 数据处理

# 目录

<b>1</b>	<b>运行环境</b>	<b>5</b>
1.1	Hadoop	5
1.2	Spark	5
1.3	可视化软件	5
<b>2</b>	<b>实验任务完成情况</b>	<b>5</b>
<b>3</b>	<b>实验分工</b>	<b>5</b>
<b>4</b>	<b>实验内容</b>	<b>5</b>
4.1	Task 1 数据预处理	5
4.1.1	任务描述	5
4.1.2	算法原理	6
4.1.3	Hadoop	6
4.1.4	Spark	7
4.1.5	实验结果展示	8
4.1.6	问题及优化解决	8
4.2	Task 2 特征抽取：人物同现统计	9
4.2.1	任务描述	9
4.2.2	算法原理	9
4.2.3	Hadoop	10
4.2.4	Spark	10
4.2.5	实验结果展示	11
4.2.6	问题及优化解决	11
4.3	Task 3 特征处理：人物关系图构建与特征归一化	12
4.3.1	任务描述	12
4.3.2	算法原理	12
4.3.3	Hadoop	13
4.3.4	Spark	14
4.3.5	实验结果展示	15
4.3.6	问题及优化解决	15
4.4	Task 4 数据分析：基于人物关系图的 PageRank 计算	15
4.4.1	任务描述	15
4.4.2	算法原理	16
4.4.3	Hadoop	16
4.4.4	Spark	18
4.4.5	实验结果展示	20
4.4.6	问题及优化解决	21
4.5	Task 5 数据分析：在人物关系图上的标签传播（选做）	21
4.5.1	任务描述	21
4.5.2	算法原理	21
4.5.3	同步算法的Hadoop设计	22
4.5.4	异步算法的Hadoop设计	25
4.5.5	Spark	27
4.5.6	实验结果展示	28

4.5.7	问题及优化解决	29
4.6	Task 6 分析结果整理（选做）	31
4.6.1	任务描述	31
4.6.2	Hadoop	31
4.6.3	Spark	32
4.6.4	实验结果展示	32
4.6.5	可视化结果	33
4.6.6	可视化遇到的问题及解决	34
5	实验总结	34
6	参考文献	35
7	附录：实验截图——HadoopWebUI	36
7.1	Task1	36
7.2	Task2	36
7.3	Task3	37
7.4	Task4	37
7.5	Task5	38
7.6	Task6	39

# 1 运行环境

## 1.1 Hadoop

Java版本	Hadoop版本	Linux	Ide for Java	分词版本包
jdk1.7	hadoop2.7.1	Ubuntu	IDEA	ansj_seg-5.1.1.jar

## 1.2 Spark

Scala版本	Spark版本	安装模式
2.12.6	2.4.3	Standalone

## 1.3 可视化软件

Gephi 0.92

# 2 实验任务完成情况

我们完成了必做实验一到四和选做实验五、六，其中标签传播算法完成了同步传播和异步传播两种传播算法，同时我们对实验四五的结果进行了后处理，最后利用可视化软件Gephi进行了可视化图片的生成和输出，并对输出结果的正确性进行了检验，对于实验算法的效率和改进也进行了检验。

# 3 实验分工

邓开圣	实现了Hadoop部分的任务二：统计人名同现次数和任务五：标签传播算法的异步算法
	实现了Spark版本的任务一：分词和任务四：PageRank的矩阵算法，PageRank结果自动排序
	完成了任务六：完成了对于实验五结果的整理
	完成了对应的实验报告
成威	完成了Hadoop部分的任务一：分词和任务四：PageRank的标准算法
	完成了Spark的任务三：图的构建和任务六：实验五的结果输出整理
	完成Hadoop任务六：在PageRank输出过程中完成了对于输出的从大到小排序
	完成对应的实验报告，并将小组成员的报告进行整合输出
	Spark程序的预期展示
戚赞	实现了Hadoop部分的任务三：图的构建和任务五：标签传播算法的同步算法
	实现了Spark版本的任务二：同现次数输出和任务五：标签传播算法，按标签结果自动排序
	完成了对于可视化的任务，输出可视化结果
	完成对应的实验报告
同时每个人还负责自己jar包的maven管理、打包和集群的测试，保证实验结果的正确性。	

# 4 实验内容

## 4.1 Task 1 数据预处理

### 4.1.1 任务描述

本任务的主要工作是从原始的金庸小说文本中，抽取与人物互动相关的数据，而屏蔽掉不人物关系无关的文本内容，为后面的基于人物共现的分析做准备。

输入：

- (1) 全本的金庸武侠小说文集(未分词)；
- (2) 金庸武侠小说人名列表。

输出：分词后，仅保留人名的金庸武侠小说全集。

#### 示例

输入：狄云和戚芳一走到万家大宅之前，瞧见那高墙朱门、挂灯结彩的气派，心中都是暗自嘀咕。戚芳紧紧拉住了父亲的衣袖。戚长发正待向门公询问，忽见卜垣从门里出来，心中一喜，叫道：“卜贤侄，我来啦。”

输出：狄云戚芳戚芳戚长发卜垣

#### 4.1.2 算法原理

使用Ansj\_seg中文分词工具5.1.1，对金庸武侠小说文集的每一行进行分词、保留金庸小说人名列表文件中的人名。

因为每本小说的预处理结果存放在不同的文件中，所以使用**MultipleOutputs**根据处理行所在的文件名决定存储的文件路径。

##### Mapper

处理输入文件的每一行，将处理结果及当前文件名发射给MultipleOutputs。

	含义	类型	值
keyInput	行偏移	LongWritable	行索引
valueInput	行内容	Text	输入文件的行内容
keyOutput	空	NullWritable	NullWritable.get()
valueOutput	仅保留人名的非空行	Text	输入行仅保留人名的结果

##### MultipleOutputs

根据Mapper传入的文件名获取输出路径，将Mapper的输出键值对写入文件。

#### 4.1.3 Hadoop

伪代码中变量含义如下：

变量名	变量类型	含义
multipleOutputs	MultipleOutputs	实现多集合文件输出
fileName	String	输入行的文件名
ret	String	仅保留人名的非空行
result	Result	分词结果
term	Term	包含每个词及其词性

---

## 算法 1 数据预处理

---

输入: *key*: 行偏移量, *value*: 行内容

输出: *key*: NullWritable, *value*: 仅保留人名的非空行

```
1: function SETUP(context)
2:   DicLibrary  $\leftarrow$  insert all names in list
3:   multipleOutputs  $\leftarrow$  new MultipleOutputs < NullWritable, Text >
4: end function
5:
6: function MAP(key, value, context)
7:   fileName  $\leftarrow$  name of the file of context
8:   ret  $\leftarrow$  null
9:   result  $\leftarrow$  Participle of value
10:  for each term  $\in$  result do
11:    if term is the name in list then
12:      line  $\leftarrow$  line + term + " "
13:    end if
14:  end for
15:  multipleOutputs.write(< NullWritable.get(), result >, fileName)
16: end function
```

---

### 4.1.4 Spark

spark实现思想与hadoop类似, 区别在于scala语言的函数式编程思想, 解析每一行后将其传入getNamesInLine函数进行分词。

---

```
object spark_task1 {
  // get all input files
  def getFiles(path:String):Array[File] = {
    val subPath = new File(path).listFiles()
    val files = subPath.filter(!_.isDirectory)
    val dirs = subPath.filter(_.isDirectory)
    files ++ dirs.flatMap(dir => getFiles(dir.getPath))
  }
  // parse one line of text
  def getNamesInLine(line: String, dictKey: String, strNature: String): String = {
    val res = ArrayBuffer[String]()
    val terms = DicAnalysis.parse(line, DicLibrary.get(dictKey))

    terms.forEach(term =>
      if (term.getNatureStr.equals(strNature)){
        res += term.getName
      }
    )
    res.toArray.mkString(" ")
  }
}

def main(args: Array[String]): Unit = {
  val conf = new SparkConf().setAppName("FinalLab_task1").setMaster("local")
  val sparkContext = new SparkContext(conf)
  val nameListPath = "people_name_list.txt"
```

```

val inputPath = "novels"
val dataFiles = getFiles(inputPath)
// customize dictionary
val dictKey = "MyDict"
val strNature = "name"
DicLibrary.put(dictKey, dictKey, new Forest())
val nameList = sparkContext.textFile(nameListPath)
nameList.foreach(name => DicLibrary.insert(dictKey, name.toString.trim, strNature, 1000))
// parse all files
for(dataFile <- dataFiles){
    val data = sparkContext.textFile(dataFile.getPath)
    val filePath = "task1_output/" + dataFile.getName.split("\\.")(0)
    data.map(line => getNamesInLine(line, dictKey, strNature)).filter(line => line.length >
        0).saveAsTextFile(filePath)
}
}
}

```

---

#### 4.1.5 实验结果展示

##### jar包执行方法

**hadoop jar HadoopTask1.jar /data/task2/people\_name\_list.txt /data/task2/novels task1\_output**

部分输出数据如下：

```

戚长发 狄云
狄云 狄云 狄云
狄云 狄云 万震山
万震山 狄云 狄云 万震山
狄云 狄云
狄云 戚长发 戚芳 狄云
狄云
万震山
狄云
戚芳 狄云
万震山 戚长发 戚长发

```

#### 4.1.6 问题及优化解决

(1) 问题：将人名列表添加到字典中进行分词，原默认字典中可能会存在人名词性的词产生干扰。

解决方案：为了只获取人名列表中的人名，自定义DicLibrary的key，将人名列表中的所有人名以自定义词性”name”插入字典中，使用用户自定义词典优先策略的分词方式DicAnalysis进行分词，词性为”name”的词即为人名。

(2) 问题：不能将“沙广天”这一人名正确分词。

解决方案：经过测试，发现Ansj\_seg 5.1.6在Linux下分词会分不出“沙广天”，最后选择回退版本5.1.1，在Linux下运行Hadoop和在Windows下运行Spark都能得到正确的分词结果。



## 4.2 Task 2 特征抽取：人物同现统计

### 4.2.1 任务描述

本任务的主要工作是完成基于单词同现算法的人物同现统计。在人物同现分析中，如果两个人在原文的同一段落中出现，则认为两个人发生了一次同现关系。我们需要对人物之间的同现关系次数进行统计，同现关系次数越多，则说明两人的关系越密切。

输入：task1的输出文件

输出：在金庸的所有武侠小说中，人物之间的同现次数。

### 4.2.2 算法原理

#### Mapper

解析输入文件的每一行，发射<人物同现对, 1>键值对。

	含义	类型	值
keyInput	行偏移量	LongWritable	每行索引
valueInput	行内容	Text	输入数据每行的文本内容
keyOutput	一个人物姓名的同现对	Text	由每一行中出现的人物姓名决定
valueOutput	输出同现对的出现次数	IntWritable	1

示例：

狄云 戚芳 戚芳 戚长发 卜垣  $\Rightarrow$  HashSet去重  $\Rightarrow$  狄云 戚芳 戚长发 卜垣  $\Rightarrow$  <狄云, 戚芳, 1> , <狄云, 戚长发, 1> , <狄云, 卜垣, 1> , <戚芳, 戚长发, 1> , <戚芳, 卜垣, 1> , <戚长发, 卜垣, 1>

#### Combiner

将mapper输出的相同key的value进行累加，减少网络数据传输负载。

#### Partitioner

将mapper输出的键按','拆开，根据第一个人物姓名对键值对进行分发。

#### Reducer

将收集到的相同key的键值对的value进行累加。

	含义	类型	值
keyInput	一个人物姓名的同现对	Text	Mapper的输出
valueInput	该人物姓名同现对出现次数	IntWritable	Mapper的输出
keyOutput	一个人物姓名的同现对	Text	
valueOutput	该同现对的出现次数	IntWritable	对该同现对出现次数的累加

### 4.2.3 Hadoop

---

**算法 2** 人物同现统计

---

输入 *key*: 行偏移量, *value*: 行内容

输出 *key*: 一个人名姓名的同现对, *value*: 该同现对的出现次数

```
1: function MAP(key, value, context)
2:   (LineOffset, Line)  $\leftarrow$  (key, value)
3:   Names  $\leftarrow$  Line.split(" ")
4:   NameSet  $\leftarrow$  Names.toSet
5:   NamesWithoutRepeat(NWR)  $\leftarrow$  NameSet.toArray
6:   for i = 0  $\rightarrow$  NWR.Size - 1 do
7:     for j = 0  $\rightarrow$  NWR.Size - 1 do
8:       if i  $\neq$  j then
9:         context.write(< NWR[i], NWR[j] >, 1)
10:      end if
11:    end for
12:  end for
13: end function
14: function COMBINE(key, [values], context)
15:   valueSum  $\leftarrow$  Sum([values])
16:   context.write(key, valueSum)
17: end function
18: function PARTITION(key, value, numReduceTasks)
19:   (hostName, neighborName)  $\leftarrow$  key
20:   super.Partition(hostName, value, numReduceTasks)
21: end function
22: function REDUCE(key, [values], context)
23:   valueSum  $\leftarrow$  Sum([values])
24:   context.write(key, valueSum)
25: end function
```

---

### 4.2.4 Spark

对于读取的每一行, 每个人名只需要一次, 而且只需要将这个人和其他所有人加起来输出1次, 最后合并值再输出, 这样的话就能够实现同现计算。那么因为每个人名都只需要一次, 我利用了set函数来进行存放和输出。

---

```
object spark_task2 {
  //string to Array of pairs<x,y>
  def getNamesInLine(line:String): Array[String] = {
    val strs = line.split(" ")
    val myset = mutable.Set[String]()
    strs.foreach(x => myset.add(x))
    val nameinline = myset.toArray

    val uniquePairs = for {
      (x, idxX) <- nameinline.zipWithIndex
      (y, idxY) <- nameinline.zipWithIndex
      if idxX != idxY
    }
```

```

    } yield (<" + x + ", " + y + ">")

    return uniquePairs
}

def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("GroupSort").setMaster("local")
    val sc = new SparkContext(conf)
    val lines = sc.textFile("task1_output/*")
    val resArray = lines.flatMap(x => getNamesInLine(x)).filter(x => x.length > 0).collect()
    //make <x,y> => (<x,y>,1)
    val res = sc.parallelize(resArray).map((_,1)).reduceByKey(_+_).sortByKey().map(line => {
        val word = line._1
        val cnt = line._2
        word + "\t" + cnt
    })
    res.saveAsTextFile("task2_output")
    sc.stop()
}
}

```

---

#### 4.2.5 实验结果展示

##### jar包执行方法

先将输入数据放入task1-ouput文件夹，上传到HDFS上，保证不存在task2\_output目录，然后运行命令：

```
hadoop jar HadoopTask2.jar task1_output task2_output
```

部分输出数据如下：

```

<张信,袁承志> 1
<张信,鸠摩智> 1
<张全祥,乔峰> 4
<张全祥,全冠清> 4
<张全祥,执法长老> 1
<张勇,上官> 1
<张勇,双儿> 3
<张勇,司徒伯雷> 2
<张勇,司徒鹤> 1
<张勇,吴三桂> 12

```

#### 4.2.6 问题及优化解决

1. 实验采用了map方法进行设计，我们知道RDD操作时并行操作，而RDD.colletc()之后就转换成非RDD，则此时操作不再时内存并行化操作，而是串行，所以后面操作要sc.parallelize()操作来转换为RDD，避免过多并行编程。

2. 设计了一个函数来解析String类型，最后返回是人物对的Array，然后将转化为(key,value)键值对，然后相加，最后在sort，输出。输出的结果和Hadoop的几乎一样。

### 4.3 Task 3 特征处理：人物关系图构建与特征归一化

#### 4.3.1 任务描述

当获取了人物之间的共现关系之后，我们就可以根据共现关系，生成人物之间的关系图了。人物关系图使用邻接表的形式表示，方便后面的 PageRank 计算。在人物关系图中，人物是顶点，人物之间的互动关系是边。人物之间的互动关系靠人物之间的共现关系确定。

如果两个人之间具有共现关系，则两个人之间就具有一条边。两人之间的共现次数体现出两人关系的密切程度，反映到共现关系图上就是边的权重。边的权重越高则体现了两个人的关系越密切。为了使后面的方便分析，还需要对共现次数进行归一化处理：将共现次数转换为共现概率，具体的过程见后面的示例。

输入：任务2的输出

输出：归一化权重后的人物关系图

示例

输入：

```
<狄云, 戚芳> 1    <戚长发, 狄云> 1
<狄云, 戚长发> 1    <戚长发, 戚芳> 1
<狄云, 卜垣> 1    <戚长发, 卜垣> 1
<戚芳, 狄云> 1    <卜垣, 狄云> 1
<戚芳, 戚长发> 1    <卜垣, 戚芳> 2
<戚芳, 卜垣> 2    <卜垣, 戚长发> 1
```

输出：

```
狄云 [戚芳,0.33333|戚长发, 0.33333|卜垣 0.33333]
戚芳 [狄云,0.25|戚长发, 0.25|卜垣 0.5]
戚长发 [狄云,0.33333|戚芳, 0.33333|卜垣 0.33333]
卜垣 [狄云 0.25|戚芳,0.5|戚长发, 0.25]
```

#### 4.3.2 算法原理

实验的思路就是将”<狄云, 戚芳>1 ”进行拆解，代表狄云后面链接了戚芳且同现的次数为1。同时我们要将所有狄云链接的输出到狄云的目标当中去，那么我们就在Map的过程输出中就必须将“狄云” 作为key的全部或者一部分，而将“count” 作为value的一部分进行输出。”戚芳”的设计需要一定的打量，首先如果作为value的一部分进行输出的话也就是输出”<name,neighbour+”, ”+count>”，首先对于Combiner的设计会比较繁琐进行拆解，同时也不会进行排序。

利用上课的高级MapReduce算法来进行处理的话，采用复合键值对的思想让利用MapReduce系统的排序功能完成排序从而能够实现结果是排完序的，所以其使用的键值对如下”<name + ”,”+ Neighbour , count>”，输出格式为key = Text , value = IntWritable。这样同时可以使得combiner的设计也比较简单。但此时又会带来一个新的问题就是要保证相同的name要被划分到同一个节点上去，那么我们就需要重新设计partitioner，将key = name + ”,” + neighbour进行拆开，按照name划分，从而使得划分到同一个reduce节点上去。

同时对于Reducer的设计也需要一点技巧。因为要除以所有人的值，那么我们需要一个Map<String,int>来统计一个人物的邻居节点的个数，从而能进行归一化操作。同时因为未完全输出，所以在cleanup()函数中进行最后结果的输出，来防止最后一个未输出。

### 4.3.3 Hadoop

#### Mapper

输入的格式:  $\text{InputFormat} = \text{TextInputFormat}$ , 因此输入为一行

输出的格式:  $\text{key} = \langle \text{Text}, \text{IntWritable} \rangle = \langle \text{name} + ", " + \text{neighbour}, \text{count} \rangle$

Map部分的伪代码如下:

---

#### 算法 3 Mapper

---

输入  $\text{key}$ : 行偏移量,  $\text{value}$ : 行内容

输出  $\text{key}$ : 人名+邻居,  $\text{value}$ : 同现次数

```
1: function MAP( $\text{key}, \text{value}, \text{context}$ )
2:    $\text{temp} \leftarrow \text{value.split}("\t")$ 
3:    $\text{count} \leftarrow (\text{Int})\text{temp}[1]$ 
4:    $\text{pair} \leftarrow \text{temp}[0].\text{replace}("< ", ",").\text{replace}("> ", ",").\text{split}(", ")$ 
5:    $\text{name} \leftarrow \text{pair}[0]$ 
6:    $\text{neighbour} \leftarrow \text{pair}[1]$ 
7:    $\text{Emit} \langle (\text{name}, \text{neighbour}), \text{count} \rangle$ 
8: end function
```

---

#### Combiner

输入的格式:  $\langle \text{Text}, \text{IntWritable} \rangle = \langle (\text{name}, \text{neighbour}), \text{count} \rangle$

输出的格式:  $\langle \text{Text}, \text{IntWritable} \rangle = \langle (\text{name}, \text{neighbour}), \text{count} \rangle$

伪代码如下:

---

#### 算法 4 Combiner

---

输入  $\text{key}$ : 人名+邻居,  $\text{values}$ : 键为key的所有同现次数

输出  $\text{key}$ : 输入key,  $\text{value}$ : 同现次数

```
1: function COMBINE( $\text{key}, \text{values}, \text{context}$ )
2:    $\text{sum} \leftarrow 0$ 
3:   for each  $\text{val} \in \text{values}$  do
4:      $\text{sum} \leftarrow \text{sum} + \text{val}$ 
5:   end for
6:    $\text{Emit} \langle \text{key}, \text{sum} \rangle$ 
7: end function
```

---

#### Partitioner

原理就是将 $\text{key} = \text{name} + ", " + \text{neighbour}$ 拆开, 利用name进行节点的划分

伪代码如下:

---

#### 算法 5 Partitioner

---

输入  $\text{key}$ : 人名+邻居,  $\text{values}$ : 同现次数

```
1: function GETPARTITION( $\text{key}, \text{value}, \text{numReduceTasks}$ )
2:    $\text{token} \leftarrow \text{key.toString().split(", ")[0]}$ 
3:   return  $\text{super.getPartition}(\text{newText}(\text{token}), \text{value}, \text{numReduceTasks})$ 
4: end function
```

---

#### Reducer

输入的格式:  $\langle \text{key}, \text{value} \rangle = \langle \text{Text}, \text{IntWritable} \rangle$  Map输出的key-value对

输出的格式:  $\langle \text{Text}, \text{Text} \rangle$  为<人名,邻接矩阵>

伪代码如下：

---

**算法 6 Reducer**

---

输入 *key*: 人名+邻居, *values*: 键为key的所有同现次数

输出 *key*: 人名, *value*: 邻居及其同现概率

```
1: Cur  $\leftarrow$  null
2: Pre  $\leftarrow$  null
3: function REDUCE(key, values, context)
4:   Cur = key.split("\t")[0]
5:   if Pre! = Cur&&Pre! = "" then
6:     res  $\leftarrow$  ""
7:     count  $\leftarrow$  sum of neighbour
8:     for each item  $\in$  neighbour.keys do
9:       res  $\leftarrow$  res + item + ", " + neighbour[item]/count
10:    end for
11:    Emit < Pre, res >
12:    Pre  $\leftarrow$  Cur
13:    neighbour  $\leftarrow$  null
14:  else
15:    neighbour[key.split("\t")[1]] += count
16:  end if
17: end function
```

---

#### 4.3.4 Spark

首先计算每个人物的总共现次数，存入Map中；再遍历每一个共现对< a, b>，则在a的共现概率出边中，b的共现概率即为 < a, b> 的共现次数/a的总共现次数，将所有与a共现的b的共现概率合在一起即可。

---

```
object spark_task3 {
def mapper(item:String)={
  // input: <name1, name2> n
  val itemInfo = item.trim().substring(1)
  val lineInfo = itemInfo.split("\t")
  val mapKeyInfo = lineInfo(0).split(",")(0)
  // output: name1,n
  (mapKeyInfo, lineInfo(1).toInt)
}

def reducer(item:String, keysIndex:collection.Map[String, Int]) ={
  // input: <name1, name2> n
  val itemInfo = item.trim().substring(1)
  val lineInfo = itemInfo.split("\t")
  val mapKeyInfo = lineInfo(0).split(",")(0)
  var mapValueInfo = lineInfo(0).split(",")(1)
  mapValueInfo = mapValueInfo.substring(0, mapValueInfo.length-1)
  mapValueInfo = mapValueInfo+","+((lineInfo(1).toFloat/keysIndex(mapKeyInfo)).formatted("%.4f"))
  // output: name1, name2,n/map[name1]
  (mapKeyInfo, mapValueInfo)
}
```

```
def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("task3").setMaster("local")
    val sc = new SparkContext(conf)
    val inputFile = "task2_output"
    val outputFile = "task3_output"
    val rdd = sc.textFile(inputFile)
    // calculate the count of co-occurrence, store it as map<name, count>
    val keysIndex = rdd.map(line => mapper(line))
        .reduceByKey(_ + _).collectAsMap()
    // calculate the ratio of its co-occurrence characters, name1 nr1,n1|nr2,n2|...
    val results = rdd.map(line => reducer(line, keysIndex))
        .reduceByKey(_ + "|" + _).map(line => line._1+"\t["+line._2+"]")
    // results.foreach(println)
    results.saveAsTextFile(outputFile)
}
}
```

---

#### 4.3.5 实验结果展示

##### jar包执行方法

**hadoop jar HadoopTask4.jar task3\_output task4\_output1 task4\_output2 task4\_output**

输出的部分截图如下所示:

```
一灯大师 [鲁有脚,0.0093|哑梢公,0.0023|杨康,0.0070|耶律燕,0.0070...
丁不三 [褚万春,0.0061|大悲老人,0.0061|封万里,0.0242|汉子,0.0303...
丁不四 [吕正平,0.0458|封万里,0.0163|范一飞,0.0425|汉子,0.0196...
丁勉 [方证,0.0115|教长,0.0115|英颢,0.0115|岳不群,0.0690...
丁同 [李三,0.2222|霍元龙,0.1111|老头子,0.1111|李文秀,0.5556]...
丁坚 [岳不群,0.0147|丹青生,0.1471|方生,0.0147|向问天,0.0882...
丁大全 [汉子,0.0588|郭襄,0.0588|王惟忠,0.1765|大汉,0.2353...
丁敏君 [殷梨亭,0.0306|宋青书,0.0175|杨逍,0.0087|汉子,0.0087...
丁春秋 [范百龄,0.0160|包不同,0.0256|康广陵,0.0192|叶二娘,0.0048...
丁游 [梅剑和,0.0526|刘培生,0.0526|袁承志,0.1579|焦公礼,0.0526—...
万圭 [鲁坤,0.0299|老王,0.0030|孙均,0.0150|...
王大平 [汉子,0.1667|刘菁,0.1667|史登达,0.3333|刘正风,0.3333]
```

#### 4.3.6 问题及优化解决

1. 优化: 采用了课上讲到过的符合键值对的思想进行利用系统自动对输出结果进行排序并且利用了combiner和partitioner来保持实验结果的正确性。这也对于实验结果的可理解性加深了。同时我们利用caseStudy获取了这个算法的本质就是倒排索引算法。

### 4.4 Task 4 数据分析: 基于人物关系图的 PageRank 计算

#### 4.4.1 任务描述

在给出人物关系图之后, 我们就可以对人物关系图进行一个数据分析。其中一个典型的分析任务是PageRank值计算。通过计算 PageRank, 我们就可以定量地分析金庸武侠江湖中的“主角”们是哪些。

输入：任务3的输出  
输出：所有人物的PageRank值

#### 4.4.2 算法原理

应用PageRank的随机浏览模型的简化：

$$PR(u) = 1 - d + d * \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

其中，d为阻尼系数，设置为0.85； $B_u$ 表示u的入边点的集合； $L(v)$ 表示点v的出边数。因为任务3已经计算了共现概率，所以上式可写为：

$$PR(u) = 1 - d + d * \sum_{v \in B_u} PR(v) * L_v(u)$$

其中， $L_v(u)$ 表示人物v到任务u的共现概率。

PageRank过程分为三个阶段：

1. GraphBuilder：建立人物之间共现的超链接图，设定初始的PageRank值
2. PageRankIter：迭代计算各人物的PageRank值
3. RankViewer：按PageRank值将人物及其PageRank值从大到小输出

#### 4.4.3 Hadoop

首先，d设置为**0.85**，PageRank的初始值设置为**1.0**，PageRank每次计算保留4位小数，**迭代终止条件为：PageRank值排名不变**。

##### 1. GraphBuilder

设定初始的PageRank值，保留任务3的共现概率图。

**Mapper**

---

##### 算法 7 GraphBuilder

---

**输入**    *key*: 人名, *value*: 与其共现的人名及共现概率

**输出**    *key*: 输入key, *value*: PageRank初始值及输入value

```

1: function MAP(key, value, context)
2:   Emit < key, (1.0#value) >
3: end function

```

---

##### 2. PageRankIter

迭代计算每个人物的PageRank值，重复这一任务直到排名不变为止。

**Mapper**

发射两种键值对：共现概率的图结构、人物对另一人物的PageRank贡献值



---

**算法 8 PageRankIter Mapper**

---

输入 *key*: 人名, *value*: PageRank值和与其共现的人名及共现概率

输出 *key*: 输入key/与其共现的人名, *value*: 与其共现的人名及共现概率/输入key对该人物的PageRank贡献值

```
1: function MAP(key, value, context)
2:   pr, graph  $\leftarrow$  value.split(" #")
3:   Emit < key, graph >
4:   persons  $\leftarrow$  graph.split(" |")
5:   for each person  $\in$  persons do
6:     name, ratio  $\leftarrow$  person.split(" ,")
7:     Emit < name, pr * ratio >
8:   end for
9: end function
```

---

**Reducer**

每个人物累加与其他人物共现得到的PageRank值, 并保留其共现概率的图结构。

---

**算法 9 PageRankIter Reducer**

---

输入 *key*: 人名, *values*: 键为key的所有值

输出 *key*: 人名, *value*: PageRank值和与其共现的人名及共现概率

```
1: function REDUCE(key, values, context)
2:   pr  $\leftarrow$  0.0
3:   d  $\leftarrow$  0.85
4:   graph  $\leftarrow$  null
5:   for each info  $\in$  values do
6:     if info.split(" ,").length > 1 then
7:       graph  $\leftarrow$  info
8:     else
9:       pr  $\leftarrow$  pr + info
10:    end if
11:  end for
12:  pr  $\leftarrow$  1 - d + d * pr
13:  Emit < key, (pr # graph) >
14: end function
```

---

**3. RankViewer**

按PageRank值将人物及其PageRank值从大到小输出。

继承FloatWritable.Comparator重写Sort的比较函数, 使shuffle排序按从大到小进行。

Partitioner设为TotalOrderPartitioner, 使得排序过程的划分尽量负载均衡。

**Mapper**

将PageRank值作为键, 其对应的人名作为值。

---

**算法 10** RankViewer Mapper

---

输入 *key*: 人名, *value*: PageRank值和与其共现的人名及共现概率

输出 *key*: PageRank值, *value*: 输入*key*

```
1: function MAP(key, value, context)  
2:   pr  $\leftarrow$  value.split("#")[0]  
3:   Emit < pr, key >  
4: end function
```

---

**Reducer**

Shuffle已经自动按照键——PageRank值从大到小排序, Reducer将键值对反过来输出即可。

---

**算法 11** RankViewer Reducer

---

输入 *key*: PageRank值, *values*: 键为*key*的所有人名

输出 *key*: 人名, *value*: PageRank值

```
1: function REDUCE(key, values, context)  
2:   for each info  $\in$  values do  
3:     Emit < info, key >  
4:   end for  
5: end function
```

---

控制上述三个阶段的主程序

GraphBuilder  $\rightarrow$  PageRankIter  $\rightarrow$  RankViewer

这三个阶段是顺序组合的, 但其中PageRankIter是迭代的MapReduce作业, 每次迭代结束后, 读取输出文件获取PageRank值的排名, 若该排名与上次迭代结束后的排名相同, 则结束迭代。

#### 4.4.4 Spark

实现思想

输入: 图结构文件, 即task3的输出

输出: 人物姓名及其最终的pr值, 按照pr值降序排序

首先定义一个人物类Character, 其包含以下字段:

- id: 人物ID, 即类别号, Int类型
- label: 人物标签, 即人物姓名, String类型
- rank: 人物pr值, Double类型, 用于pageRank计算和最终结果的排序

另外, 在Character类中覆盖**compare**函数, 用于计算完成后按照自定义的方法对Characters的数组进行排序, 排序方法为先按pr值进行降序排序, 若两个人物的pr值相同, 则根据人物姓名按字典序升序排序; 还覆盖了**toString**函数, 用于排序完成后按自定义的方法对Character类的对象进行输出。

算法的每一轮, 根据图结构文件的每一行将当前人物的pr值根据其与其邻居的连接权重传送给其邻居, 具体传输的值为:

$$prHost \times weight \times prRandomPara$$

其中prHost为当前人物的pr值, weight为该人物与某个邻居的连接权重, prRandomPara为随机游走模型的参数, 此处设为0.85。

此传递过程在parseLine函数中完成。对于图结构文件的每一行, 此过程均返回一个 $1 \times nCharacter$ 格

式的向量，其中nCharacter为人物总数量，若当前人物与某个人物没有连接，则向量中对应位置的元素值为0。

利用RDD的map操作将数据集并行地传递给parseLine函数进行解析，并利用reduce操作将解析的结果传递给addVector函数，将所有行解析得到的向量累加，即可得到该轮迭代计算的结果，即更新后的人物pr值。

---

```
object spark_task4 {
  class Character(var id: Int, var label: String, var rank: Double) extends Ordered[Character]{
    this.id = id
    this.label = label
    this.rank = rank

    override def compare(that: Character): Int = {
      val rankComparison = rank.compareTo(that.rank)
      if(rankComparison != 0)
        -rankComparison
      else
        label.compareTo(that.label)
    }
    override def toString: String = this.label + "\t" + this.rank.toString
  }

  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("spark_task4").setMaster("local")
    val sc = new SparkContext(conf)

    val prRandomPara = 0.85
    val iterTimes = 15
    val resultPath = "task4_output"

    val graphDataRDD = sc.textFile("task3_output")
    var labelCount = 0
    val nameList = graphDataRDD.map(line => {
      labelCount = labelCount + 1
      (labelCount, line.toString.split('\t')(0))
    })

    val nameListCollection = nameList.collect()

    val nCharacters = nameListCollection.length

    var characters: Array[Character] = new Array[Character](nCharacters)

    val nameLabels = collection.mutable.Map[String, Int]()

    for(i <- nameListCollection.indices){
      characters(i) = new Character(i + 1, nameListCollection(i)._2, 0)
      nameLabels.put(nameListCollection(i)._2, nameListCollection(i)._1 - 1)
    }

    var ranks: Array[Double] = new Array[Double](nCharacters)
```

```

for(i <- ranks.indices){
  ranks(i) = 1
}

def parseLine(host:String, suffix: String, currentRanks: Array[Double]): Array[Double] = {
  val res: Array[Double] = new Array[Double](nCharacters)
  val hostId = nameLabels.get(host)
  val pairs = suffix.replace("[", "").replace("]", "").split("\\|")
  for(pair <- pairs){
    val neighbor = pair.split(",")(0)
    val weight = pair.split(",")(1).toDouble
    res(nameLabels.getOrElse(neighbor, 0)) = currentRanks(nameLabels.getOrElse(host,0)) *
      weight * prRandomPara
  }
  res
}

def addVector(v1: Array[Double], v2:Array[Double]): Array[Double] = {
  for(i <- v1.indices){ v1(i) += v2(i) }
  v1
}

for(i <- 0 until iterTimes){
  var tmpRanks = graphDataRDD.map(line => parseLine(line.split('\t')(0), line.split('\t')(1),
    ranks))
  .reduce((a,b) => addVector(a,b))
  ranks = tmpRanks.map(ele => ele + (1 - prRandomPara))
}

for(i <- characters.indices){ characters(i).rank = ranks(i) }
characters = characters.sorted
val out = new FileWriter(resultPath, true)
for(character <- characters){ out.write(character.toString + '\n')}
out.close()
}
}

```

---

#### 4.4.5 实验结果展示

##### jar包执行方法

**hadoop jar HadoopTask4.jar task3\_output task4\_output1 task4\_output2 task4\_output**

最终的结果为task4\_out

部分输出数据如下：

韦小宝	33.905
令狐冲	20.1766
张无忌	19.7111
郭靖	15.8956
杨过	14.5438
袁承志	13.7979
段誉	13.7219
黄蓉	13.5701
胡斐	13.1232
汉子	12.9527

spark实现的输出结果与此类似。

#### 4.4.6 问题及优化解决

1. 优化：PageRankIter阶段的Mapper可能产生很多键相同的值，可增加Combiner将相同key的value累加。减少Map阶段输出中间结果的数据量，以便降低数据的网络传输开销，缩短程序计算时间。
2. PageRank阶段从共现概率图结构中获取信息时，使用split("|")运行报错。  
解决方案：java split()函数的参数是正则表达式，"|"恰是正则表达式中的特殊字符，要根据"|"划分字符串，需要将其进行转义。使用"\\|"即可。

### 4.5 Task 5 数据分析：在人物关系图上的标签传播（选做）

#### 4.5.1 任务描述

本任务的主要工作是利用标签传播（Label Propagation）算法进行图分析，并为图上的顶点打标签，进行图顶点的聚类分析，从而在一张类似社交网络中完成社区发现（Community Detection）。

输入：任务3的输出

输出：人物的标签信息

#### 4.5.2 算法原理

标签传播算法是基于图的半监督学习方法，基本思路是从已标记的节点的标签信息来预测未标记的节点的标签信息，利用样本间的关系，建立完全图模型。

每个节点标签按相似度传播给相邻节点，在节点传播的每一步，每个节点根据相邻节点的标签来更新自己的标签，与该节点相似度越大，其相邻节点对其标注的影响权值越大，相似节点的标签越趋于一致，其标签就越容易传播。在标签传播过程中，保持已标记的数据的标签不变，使其将标签传给未标注的数据。最终当迭代结束时，相似节点的概率分布趋于相似，可以划分到一类中。标签传播算法分为同步和异步标签传播算法。

**LPA的做法比较简单：**

第一步：为所有节点指定一个唯一的标签；

第二步：逐轮刷新所有节点的标签，直到达到收敛要求为止。对于每一轮刷新，节点标签刷新的规则如下：对于某一个节点，考察其所有邻居节点的标签，并进行统计，将出现个数最多的那个标签赋给当前节点。当个数最多的标签不唯一时，随机选一个。

标签传播算法的节点标签更新策略主要分成两种，一种是**同步更新**，另一种是**异步更新**：

1. 同步更新：在执行第t次迭代更新时，仅依赖第t-1次更新后的标签集。

2. 异步更新：在执行第t次迭代更新时，同时依赖t次迭代已经更新的标签集以及在t-1更新但t次迭代中未更新的标签集，异步更新策略更关心节点更新顺序，所以在异步更新过程中，节点的更新顺序采用随机选取的方式。

LPA算法适用于非重叠社区发现，针对重叠社区的发现问题，学者提出了COPRA（Community Overlapping Propagation Algorithm）算法。该算法提出所有节点可以同时属于V个社区，V是个人为设定的全局变量，很显然v的选择直接影响算法的效果，针对v的选择需要足够的先验知识，在真实的社区网络中，v的选择不能很好的被控制。

其中小组成员戚赞实现了标签传播的同步更新算法，小组成员邓开圣实现了标签传播的异步算法。

#### 4.5.3 同步算法的Hadoop设计

首先是数据结构初始化，为每个人物将自己作为标签，然后进行迭代。迭代设计是重点。

因为是同步算法，第X轮更新要用到 X-1轮的标签和已有的图结构。那么我们则需要保存，最先可以想到的是和PageRank进行相同的算法，就是输出人物加标签加图结构。想当于只读取一个文件，但是这样带来一个问题，那就是如果维护图结构，那么人物必定在Reduce的key当中作为一部分被包含，这样的话标签传播算法在Map里面是必须是更新完标签的，但是这是无法实现的。

那么一般来讲，我们可以设置一个全局变量来存放人物和标签的对应关系，但是全局变量的设计及不符合MapReduce的思想，于是我的实现是如下，每次迭代的输入文件为图结构，而利用Configuration传递上一次存放的标签节点的文件目录，在Map的setup()方法类的时候，进行读取并进行存取，虽然会重复降低效率，但避免了全局变量的使用，同时效率浪费也不是很多。

所以Map在setup会读取标签词典，在Map过程中，对于每一行的人物name，获取其邻居的名字，权重weight和标签label，并且进行输出，输出为<name+label, weight>。key, value类型都为Text()。这里没有相对于实验三，同样使用复合键值对，原因是方便combiner计算。

Partitioner也将相同的人物保证划分到同一个Reduce节点上。

最后Reduce节点接受到某个人物的所有邻居标签值和权重，取权重和最大的标签作为自己的标签进行输出，如果便签相等，取第一个最大值。之所以取第一个最大值的原因，是为了之后比较选取策略对于结果的影响。

设计了三个类来进行实验：

1. LpaIterDriver：调用LpaIterBuild和LpaIterIter来实现算法
2. LpaIterBuild：进行初始化的建立，每个元素的标签是自己
3. LpaIterIter：迭代计算

主要给出LpaIterIter的伪代码实现：

Configuration传递的全局变量为上一次标签传播的结果所在的文件夹

##### Mapper

全局变量: Map<String,String> keyLabel = new HashMap <tring,String> (); //read

---

```
Setup(){
    String temp = Conf.get("lastpath")
    BufferedReader in = null;
    FSDataInputStream fsi = null;
    String line = null;
    for(int i = 0 <- fileList.length){
        if(fileList[i].isDirectory() == False){
            fsi = fs.open(fileList[i].getPath());
            in = new BufferedReader(new InputStreamReader(fsi,"UTF-8"));
            while((line = in.readLine())!=null){
                String []temp = line.split("\t");
                keyLabel.put(temp[0],temp[1]); //end
            }
        }
    }
}
```

```

    }
  }
}

map(Object key, Text value, Context context){
    String t = value.split("\\ t")
    String name = t[0]
    String people = t[1].replace("[", "").replace("]", "").split("|")
    for(item in people){
        String label = keyLabel[item.name]
        context.write(name+label,weight)
    }
}

```

---

## Combiner

进行简单的聚合

```

reduce(Text key, Iterable<FloatWritable> value, Context context)
throws IOException,InterruptedException{
    float sum = 0;
    for(FloatWritable a : value){
        sum += a.get();
    }
    context.write(key,new FloatWritable(sum));
}

```

---

## Partitioner

拆分键，用新键将Map节点的输出分区到Reduce节点

```

getPartition(Text key, FloatWritable value, int numReduceTasks) {
    String token = key.toString().split(",")[0];
    return super.getPartition(new Text(token), value, numReduceTasks);
}

```

---

## Reducer

选取最大的标签输出

全局变量：

String Cur, Pre;

Map<String,Float> weight = null;

伪代码如下：

---

**算法 12 Reducer**

---

输出 *key*: 人名, *value*: 标签

```
1: function REDUCE(key, values, context)
2:   Cur = key.split("\\t")[0]
3:   if Pre! = Cur && Pre! = "" then
4:     res  $\leftarrow$  ""
5:     for each item  $\in$  weight.keys do
6:       res  $\leftarrow$  maxlabel(res, item)
7:     end for
8:     Emit < Pre, res >
9:     Pre  $\leftarrow$  Cur
10:    weight  $\leftarrow$  null
11:  else
12:    weight[key.split("\\t")[1]]+ = count
13:  end if
14: end function
```

---

同步标签算法的结束条件设置:

最后是迭代结束的条件设置, 我原本想用标签不变作为终止条件, 但是实验出现了一些问题。一开始怎么也不会收敛, 对比结果才发现了出现了同步标签算法的一个重要的bug就是同步标签算法存在二分图震荡的现象。需要注意的是, 这种同步更新的方法会存在一个问题, 当遇到二分图的时候, 会出现标签震荡, 如下图:

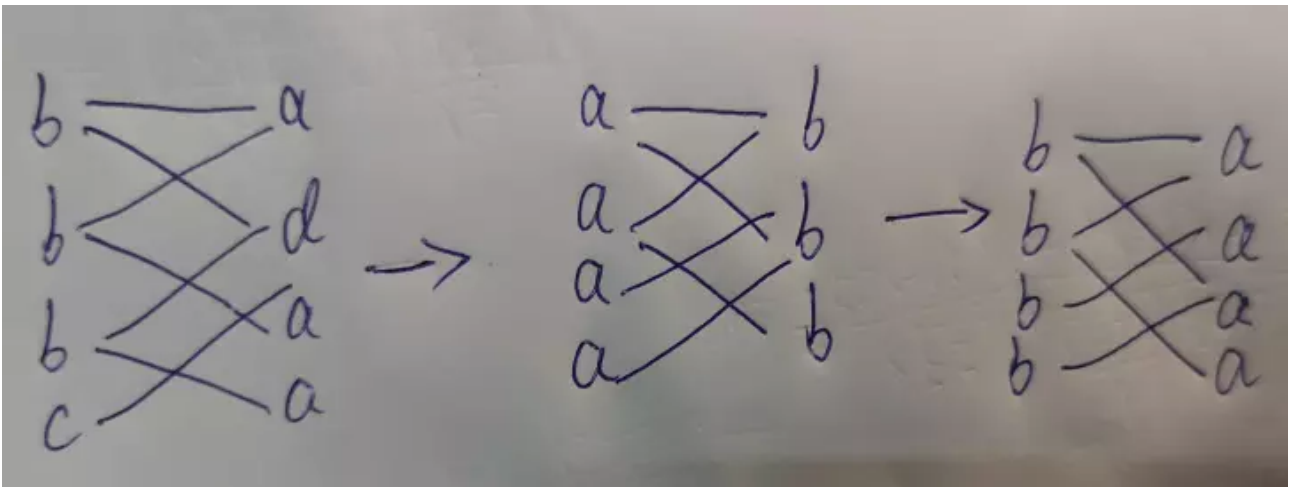


图 1: 二分图标签震荡示意图

那么迭代条件为标签不变无法生效, 那么根据这个实际情况, 我设置最大轮数为15轮, 因为15轮开始后就仅剩二分图震荡的更新。二分图震荡的人物如下:



殷吉，曹云奇，陶子安  
郑三娘，陶子安，曹云奇  
熊元献，陶子安，曹云奇  
周云阳，曹云奇，陶子安  
田青文，曹云奇，陶子安  
刘元鹤，陶子安，曹云奇  
静智大师，曹云奇，陶子安  
耶律重元，耶律涅鲁古，耶律重元  
曹云奇，陶子安，曹云奇  
陶子安，曹云奇，陶子安  
耶律涅鲁古，耶律重元，耶律涅鲁古  
陶百岁，曹云奇，陶子安

虽然标签算法是收敛的但是无法处理标签震荡的原则，队友**邓开圣**实现了标签的异步算法，异步算法解决了标签震荡的问题，下面会进行展示。

4.5.4 异步算法的Hadoop设计

异步标签传播算法的Hadoop实现也可分为三个部分：

- LabelBuilder：进行标签的初始化
- LabelIter：标签传播迭代计算
- LabelDriver：作为主函数调用LabelBuilder和LabelDriver，运行整个算法

以下依次进行介绍。

**LabelBuilder**

该部分的任务是收集task3的输出文件中出现过的所有人名，赋予初始标签。

输入：task3的输出结果

输出：所有人物及其对应的初始标签

思路：设置一个全局行号统计变量labelCount，初始化为0，每统计一行自增1

**Mapper**

	含义	类型	值
keyInput	行偏移	LongWritable	每行索引
valueInput	行内容	Text	输入数据每行的文本内容
keyOutput	该行人物姓名	Text	由输入数据决定
valueOutput	该人物对应的标签值	Text	# labelCount的值

**Reducer**

直接将Mapper的输出结果输出，不做任何处理。

## LabelIter

该部分的任务是进行标签传播算法的迭代计算，计算的每一轮完成后都更新人物及其对应的标签。

输入1: task3的输出，即图结构文件

输入2: 第一轮输入为LabelBuilder输出的人物初始标签，之后每一轮的输入为前一轮更新后的人物标签文件

输出: 每一轮迭代计算后人物标签的更新结果

思路: 在Mapper中解析图结构文件，逐个输出每个人物对应的邻居及其相应的权重；在Reducer的setup函数中读取当前的人物标签，在reduce函数中，对每一个人物，将其同一类的邻居的权重相加，取权重和最大的一类的标签，作为当前人物的标签，每获得一个人物的新标签，立即更新 [人物姓名→类别标签] 映射表，待所有人物的标签更新完毕后，将此结果写入指定路径，作为下一轮迭代计算的输入。利用这种异步更新的方法不仅可以加快算法收敛速度，还能有效解决同步LPA算法的二分图震荡问题。

## Mapper

	含义	类型	值
keyInput	行偏移	LongWritable	每行索引
valueInput	行内容	Text	输入数据每行的文本内容
keyOutput	该行人物姓名	Text	由输入数据决定
valueOutput	该人物的邻居姓名及其权重	Text	<邻居姓名, 权重>

## Reducer

	含义	类型	值
keyInput	当前行人物姓名	Text	由Mapper输出决定
valueInput	该人物的邻居姓名及其权重	Text	<邻居姓名, 权重>
keyOutput	当前行人物姓名	Text	
valueOutput	该人物对应的标签值	Text	# 标签传播算法计算结果

## LabelDriver

该部分的任务是标签传播算法运行的主类，首先调用一次LabelBuilder构建人物初始标签，然后根据设定的迭代次数调用LabelIter，每一轮迭代更新LabelIter的输入和输出路径，使当前轮的输入为上一轮的输出。

输入: task3的输出文件

输出: 每一轮标签传播后人物的分类结果以及最终分类结果

以下给出LabelIter部分的伪代码:

---

**算法 13** 异步标签传播 (LPA) 算法迭代器

---

**输入** *key*: 包含所有人物姓名的列向量, *value*: 每个人物对应的初始类别标签

**输出** *key*: 包含所有人物姓名的列向量, *value*: 每个人物在每轮LPA算法计算后更新的类别标签

```
1: function MAP(key, value, context)
2:   (LineOffset, Line)  $\leftarrow$  (key, value)
3:   (HostName, NeighborNamesAndWeights)  $\leftarrow$  Line.split("\\t")
4:   [(NeighborNames, Weights)]  $\leftarrow$  NeighborNamesAndWeights.split("|")
5:   for (NeighborName, Weight)  $\in$  [(NeighborNames, Weights)] do
6:     context.write(HostName, < NeighborName, Weight >)
7:   end for
8: end function
9: function SETUPOFREDUCER(context)
10:  nameLabelsHashMap < String, Integer >  $\leftarrow$  < name, label > data output of last iteration
11: end function
12: function REDUCE(key, [values], context)
13:  weightMap < Integer, Float >  $\leftarrow$   $\emptyset$ 
14:  HostName  $\leftarrow$  key
15:  for value  $\in$  values do
16:    (NeighborName, Weight)  $\leftarrow$  value
17:    NeighborType  $\leftarrow$  nameLabelHashMap.getValueByKey(NeighborName)
18:    if NeighborType  $\in$  weightMap.keys then
19:      weightMap.getValueByKey(NeighborType) + = Weight
20:    else
21:      weightMap.addNewKeyValuePair(NeighborType, Weight)
22:    end if
23:  end for
24:  Sort weightMap by values in descending order
25:  NewType  $\leftarrow$  weightMap[0].key
26:  Set the type of HostName to NewType
27:  Renew the nameLabelsHashMap by nameLabelsHashMap.put(HostName, NewType)
28:  context.write(HostName, NewType)
29: end function
```

---

#### 4.5.5 Spark

由于可以使用全局变量, 于是不同于hadoop可以使用一个全局变量来存放标签, 同时标签可以存放为自己的结果, 那么我们每次只要读取图结构并且进行输出即可。

定义了一个函数:

```
def getNewLabel(line:String, MapOfLabel:collection.mutable.Map[String,String]): String = {}
```

来确定对于每一个人物的新的标签, MapOfLabel为全局变量, 每轮会进行迭代更新值, 结束之后更新标签即可。

---

```
object spark_task5 {
def getNewLabel(line:String, MapOfLabel:collection.mutable.Map[String,String]): String = {
  //get max weight label
  var line_ = line.replace("[", "")
  var Neighbours = line_.replace("]", "").split("\\|")
  var FinalLabel = collection.mutable.Map[String, Float]()
}
```

```

    for (name <- Neighbours) {
    var temp = name.split(",")
    //get name and neighbour's name
    var label = MapOfLabel(temp(0))
    //ger label
    var value = temp(1).toFloat
    if (FinalLabel.contains(label)) {
        var newValue = value + FinalLabel(label)
        FinalLabel.put(label, newValue)
    }
    else {
        FinalLabel.put(label, value)
    }
    }
    //Get max
    FinalLabel.maxBy(_._2)._1
}

def main(args: Array[String]): Unit = {

    val outputDir = "task5_output"
    val conf = new SparkConf().setAppName("LPA").setMaster("local")
    val sc = new SparkContext(conf)
    val task3_out = sc.textFile("task3_output").filter(x => x.length > 0)

    var MapOfLabel = collection.mutable.Map[String,String]()
    //initial
    val init = task3_out.map(x => (x.split("\t")(0),x.split("\t")(0))).collect()
    init.foreach(a => MapOfLabel.put(a._1,a._2))
    //iteration
    for(x <- 0 until 15){
        println("Iterating round "+x+" ...")
        val Graph = sc.parallelize(sc.textFile("task3_output").map(x => x.split("\t")).collect())
        println("Begin label propagation...")
        var res = Graph.map(line => (line(0),getNewLabel(line(1),MapOfLabel))).collect()
        res.foreach(a => MapOfLabel.put(a._1,a._2))
    }

    //parallelize and sort and save
    sc.parallelize(MapOfLabel.toSeq).sortBy(_._2).map(line => {
    val word = line._1
    val label = line._2
    word + "\t" + label
    }).saveAsTextFile(outputDir)
    }
}

```

---

#### 4.5.6 实验结果展示

##### jar包执行方法

先将输入数据放入task3\_output文件夹，上传到HDFS上，保证不存在task5\_output目录，然后运行命令：

```
hadoop jar dks_Hadoop_Task5.jar task3_output task5_output
```

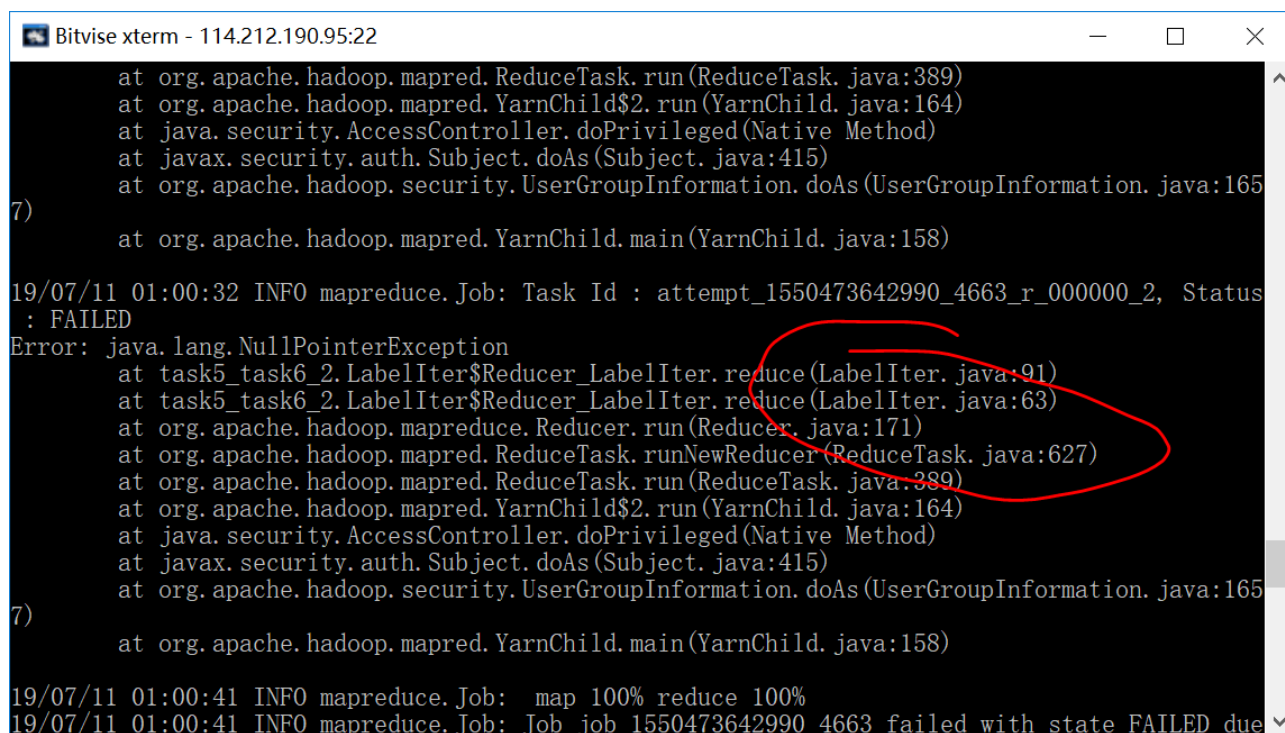
部分输出数据如下:

凌退思	狄云
梅念笙	狄云
宝象和尚	狄云
常长风	盖一鸣
花剑影	盖一鸣
盖一鸣	盖一鸣
逍遥子	盖一鸣
成自学	石破天
柯万钧	石破天
耿万钟	石破天

#### 4.5.7 问题及优化解决

1. 结果和hadoop的差别很小，只有10个人不同这是因为相等权重的人选择算法得不一致性，所以我们可以得知，即使相同得时候固定选择一个和随机选择也是相差得很少，正确率大约为 99.2%。所以标签传播算法得正确性很高。
2. 一开始写字典遇到一些问题那就是加入不进去，后来发现是RDD惰性加载的原因，添加到字典里面的值必须是经过collect的Array，所以一开始我跳出循环之后字典内始终没有变量，这就是问题所在。
3. 相比较Hadoop的程序编写，一是Spark的可以使用全局变量，所以加快了速度，二是Spark可以有方便的接口，于是我利用sortBy函数得出结果，比Hadoop只能按key排序来说操作方便很多。
4. 并行方面进行处理，很多用RDD进行操作，因为Spark的思想就是基于内存进行计算，所以我们也不需要向hadoop读写文件操作。更加方便且快捷，速度快的很多。

#### 5. 集群运行时出现NullPointerException报错



```
Bitvise xterm - 114.212.190.95:22
at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:389)
at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:415)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1657)
at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
19/07/11 01:00:32 INFO mapreduce.Job: Task Id : attempt_1550473642990_4663_r_000000_2, Status : FAILED
Error: java.lang.NullPointerException
at task5_task6_2.LabelIter$Reducer_LabelIter.reduce(LabelIter.java:91)
at task5_task6_2.LabelIter$Reducer_LabelIter.reduce(LabelIter.java:63)
at org.apache.hadoop.mapreduce.Reducer.run(Reducer.java:171)
at org.apache.hadoop.mapred.ReduceTask.runNewReducer(ReduceTask.java:627)
at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:389)
at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:415)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1657)
at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
19/07/11 01:00:41 INFO mapreduce.Job: map 100% reduce 100%
19/07/11 01:00:41 INFO mapreduce.Job: Job job_1550473642990_4663 failed with state FAILED due to
```

图 2: 集群运行NullPointerException报错

经排查，发现是由于LabelIter中使用了全局变量，而程序在分布式环境下运行，此时全局变量无法被所有分块访问到。

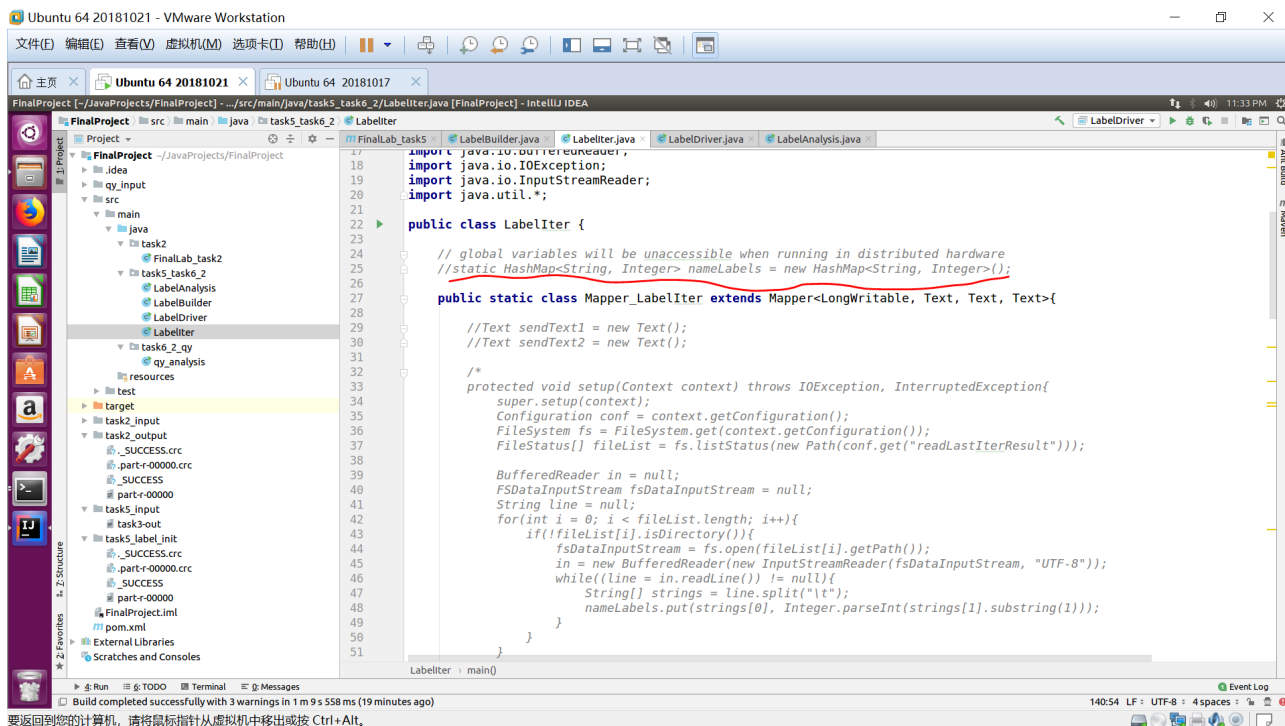


图 3: 错误原因为LabelIter中使用了全局变量

解决方案是不使用全局变量保存人物姓名与标签的映射，改为Reducer的局部变量并在setup函数中进行读取。

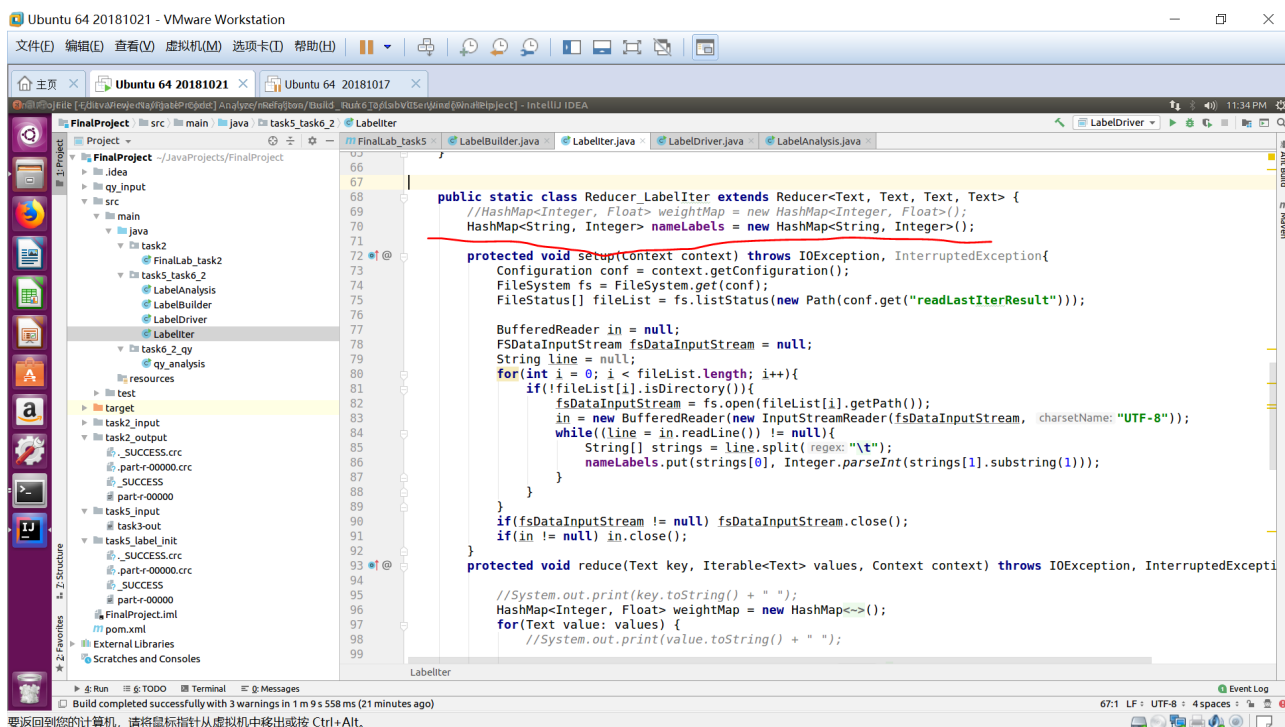


图 4: NPE解决方案

## 4.6 Task 6 分析结果整理（选做）

### 4.6.1 任务描述

任务3和任务4默认的输出内容是杂乱的，从中无法直接地得到分析结论，因此我们需要对上述分析的结果进行整理，从而很容易地从分析结果中发现一些有趣的结论。对于任务 4(PageRank)的输出，可以通过写一个全局排序的程序对人物的 PageRank 值进行排序，从而很容易地发现 PageRank 值最高的几个人物。排序工作可以通过一个排序 MapReduce 程序完成，也可以将 PageRank 值导入 Hive 中，然后利用 Hive 完成排序。我们在task4中使用RankViewer任务完成了对PageRank值的排序，即task4的输出是有序的。对于任务6(标签传播)的输出，我们针对同步算法和异步算法分别编写了一个MapReduce程序，将属于同一个类别的人物输出到了一起，从而便于查看标签传播的结果。

### 4.6.2 Hadoop

#### 同步算法输出结果的整理

同步算法的输出结果是<人物姓名， 人物类别标签>的形式，其中人物类别标签用另外一个人物的姓名表示。整理的思路是在Mapper中将人物类别标签作为Key，人物姓名作为Value输出，通过这种方式可以方便地在Reducer中获取属于同一个类别的所有人物姓名。以下给出核心部分代码实现：

---

```
public static class Mapper_qyAnalysis extends Mapper<LongWritable, Text, Text, Text>{

    protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
        String[] strings = value.toString().split("\\t");
        // reverse the location of key and value
        context.write(new Text(strings[1]), new Text(strings[0]));
    }
}

public static class Reducer_qyAnalysis extends Reducer<Text, Text, Text, Text>{

    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
        InterruptedException{
        StringBuilder stringBuilder = new StringBuilder();
        // gather the names of the same class
        for(Text value: values){
            stringBuilder.append(value.toString() + "|");
        }
        String result = stringBuilder.substring(0,stringBuilder.length()-1);
        context.write(key, new Text(result));
    }
}
```

---

#### 异步算法输出结果的整理

异步算法的输出结果已经是<人物姓名， 人物类别标签>的形式。整理的思路是在Mapper中将人物类别标签解析为IntWritable类型并作为Key输出，人物姓名作为Value输出，这样可以很方便地在Reducer中获取属于同一个类别的所有人物姓名，并且还可以利用MapReduce框架的Shuffle过程将代表人物类别的标签进行从小到大排序。 以下给出核心部分代码实现：

---

```
public static class Mapper_LabelAnalysis extends Mapper<LongWritable, Text, IntWritable, Text>{

    protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
```

---

```

String[] strings = value.toString().split("\t");
// parse the character's class label as integer
int typeIndex = Integer.parseInt(strings[1].substring(1));
String characterName = strings[0];
// set the integer class label as key
context.write(new IntWritable(typeIndex), new Text(characterName));
}
}

public static class Reducer_LabelAnalysis extends Reducer<IntWritable, Text, IntWritable, Text>{

protected void reduce(IntWritable key, Iterable<Text> values, Context context) throws IOException,
    InterruptedException{
    StringBuilder stringBuilder = new StringBuilder();
    // gather the names of the same class
    for(Text value: values){
        stringBuilder.append(value.toString() + "|");
    }
    String result = stringBuilder.substring(0,stringBuilder.length()-1);
    context.write(key, new Text(result));
}
}

```

---

### 4.6.3 Spark

将相同标签的人名用“—”连接，写入输出文件即可。

```

object spark_task6 {
def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("task6").setMaster("local")
    val sc = new SparkContext(conf)

    val inputDir = "task5_output"
    val outDir = "task6_output"

    val rdd = sc.textFile(inputDir)

    // combine the names which have a same label with "|"
    val results = rdd.map(line => (line.split("\t")(1), line.split("\t")(0)))
    .reduceByKey(_ + "|" + _).map(line => line._1+"\t"+line._2)

    // results.foreach(println)
    results.saveAsTextFile(outDir)
}
}

```

---

### 4.6.4 实验结果展示

部分输出数据如下：



耶律涅鲁古 耶律重元

杨过 点苍渔隐|李莫愁|公孙止|耶律燕|金轮法王|吊死鬼|陆二娘|无相禅师|...

苏普 李文秀|车尔库|苏普|丁同|马家骏|史仲俊|陈达海|上官虹|桑斯|...

曹云奇 静智大师|田青文|殷吉|陶子安|陶百岁|郑三娘|周云阳

令狐冲 方生|狄镖头|宁中则|曲洋|何三七|玉音子|仇松年|侯人英|...

#### 4.6.5 可视化结果

利用版本: Gephi-0.9.2

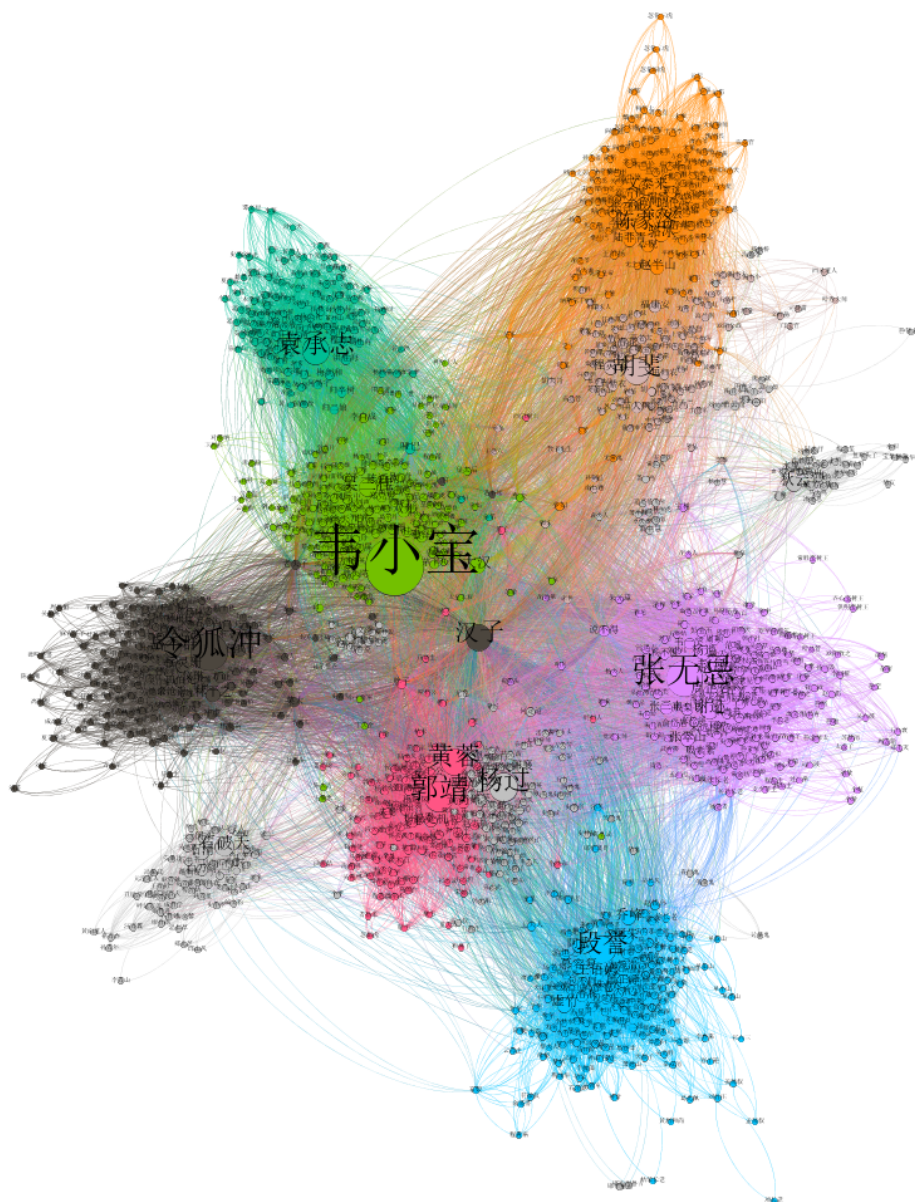
操作流程: 首先建立好输入输出csv, 节点id为数字, label为姓名, 同时带有两个属性, 一个是实验四的pageRank值, 一个是class为分的类别, 示例如下:

```
id, label, class, pagerank
0, 韦小宝, 韦小宝, 33.905
1, 令狐冲, 令狐冲, 20.1766
2, 张无忌, 张无忌, 19.7111
3, 郭靖, 郭靖, 15.8956
4, 杨过, 杨过, 14.5438
5, 袁承志, 袁承志, 13.7979
6, 段誉, 段誉, 13.7219
7, 黄蓉, 郭靖, 13.5701
8, 胡斐, 胡斐, 13.1232
9, 汉子, 令狐冲, 12.9527
10, 陈家洛, 陈家洛, 9.3365
11, 石破天, 石破天, 8.105
12, 吴三桂, 韦小宝, 7.7415
```

同时边为 source - target - count 类型, 代表同现次数, 示例如下:

```
source, target, weight
228, 294, 1
228, 38, 5
228, 1225, 1
228, 377, 17
228, 259, 1
228, 459, 2
228, 928, 1
228, 54, 28
228, 80, 1
228, 1049, 1
228, 16, 1
228, 777, 1
228, 1106, 3
228, 426, 2
228, 839, 3
228, 18, 8
```

新建工程，将节点和边导入，并且进行点大小按pageRank值排序，颜色设置按照class设置，布局为ForceAtlas布局，同时添加上标签即可。运行的效果图如下：



对于图我们可以发现基本上每个社区都是一本小说中的人物，而且核心的成员都是主角：比如韦小宝等，基本展示出我们实验结果的正确性。

#### 4.6.6 可视化遇到的问题及解决

1. 可视化遇到的问题是一开始csv作为节点输入的时候，将标签属性写成"label"，然而这个是和原来的定义冲突的，所以不能够使用，换成class即可。
2. 每次加边加点都要加入到当前的表中而不是新建一个区。

## 5 实验总结

本次实验我们使用了Hadoop的MapReduce框架和Spark分别完整地实现了流程的展示，对于其中遇到的问题我们都进行了寻找并解决，实现了综合性的数据挖掘任务，包括数据的预处理和数据后处理等。从Gephi的展示的实验结果来看，我们的实验结果是较为准确的，比较好的挖掘出小说的主要人物，对于

人物的分类也比较准确。我们对于问题尝试从多个角度来进行解决，包括pageRank的实验的迭代结束条件，还有标签传播算法的同步算法出现的震荡问题从而进一步实现异步算法的解决等。通过解决一个个出现的奇怪的问题，我们对于大数据处理的工具运用的熟练程度增加了，而且对于大数据的功能和巨大作用有了全新的认识。同时锻炼了我们的数据挖掘的能力。

最后感谢老师和助教本学期的辛苦的劳动，感谢在实验过程中给予我们帮助的同学，感谢大数据暑期实验课程给我们如此多的收获！

## 6 参考文献

- [1]标签传播算法 <https://blog.csdn.net/qjc937044867/article/details/50830399>
- [2]二分图的震荡 <https://www.cnblogs.com/LittleHann/p/10699988.html>
- [3]Spark的windows安装使用 <https://blog.csdn.net/hongxingabc/article/details/81565174>
- [4]Ansj\_seg 5.x自定义词典DicLibrary Demo [https://github.com/NLPchina/ansj\\_seg/blob/master/src/test/java/org/ansj/library/DicLibraryTest.java](https://github.com/NLPchina/ansj_seg/blob/master/src/test/java/org/ansj/library/DicLibraryTest.java)
- [5]ubuntu下安装scala的步骤 <https://blog.csdn.net/u014273195/article/details/70991010>
- [6]子雨大数据之Spark入门教程（Scala版） [http://dblab.xmu.edu.cn/blog/spark/?tdsourcetag=s\\_pcqq\\_aiomsg](http://dblab.xmu.edu.cn/blog/spark/?tdsourcetag=s_pcqq_aiomsg)

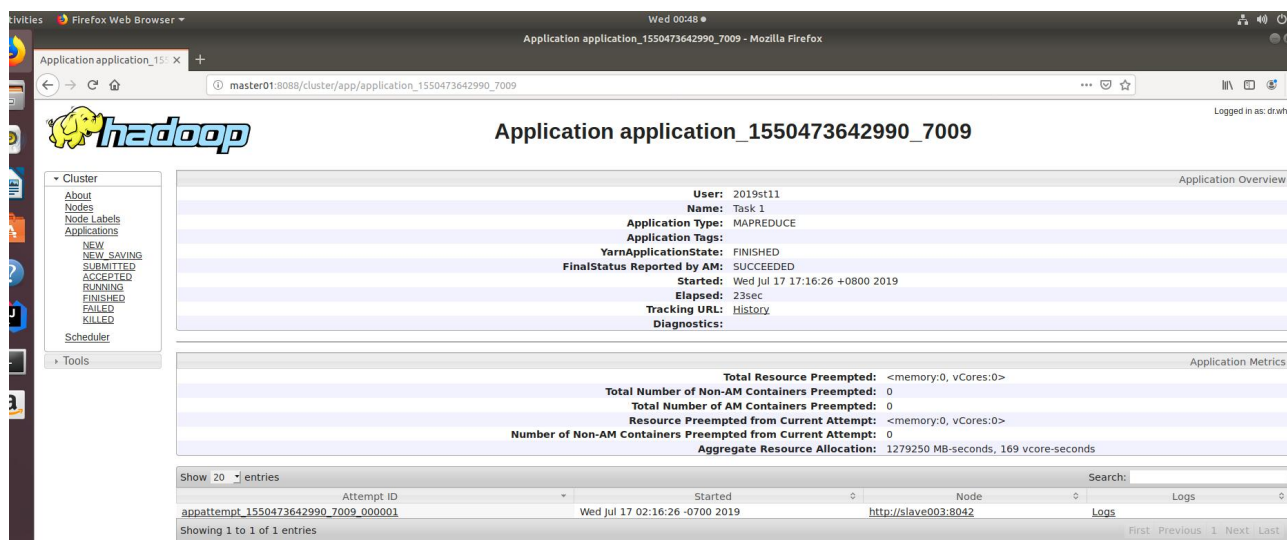
## 7 附录：实验截图——HadoopWebUI

### 7.1 Task1

任务执行：

application_1550473642990_7009	2019st11	Task 1	MAPREDUCE	root.2019st11	Wed Jul 17 02:16:26 -0700	Wed Jul 17 02:16:50 -0700	FINISHED	SUCCEEDED	<a href="#">History</a>
--------------------------------	----------	--------	-----------	---------------	---------------------------	---------------------------	----------	-----------	-------------------------

具体WebUI截图：



The screenshot shows the Hadoop WebUI interface for application\_1550473642990\_7009. The application overview section displays the following details:

- User: 2019st11
- Name: Task 1
- Application Type: MAPREDUCE
- Application Tags:
- YarnApplicationState: FINISHED
- FinalStatus Reported by AM: SUCCEEDED
- Started: Wed Jul 17 17:16:26 +0800 2019
- Elapsed: 23sec
- Tracking URL: [History](#)
- Diagnostics:

The application metrics section shows:

- Total Resource Preempted: <memory:0, vCores:0>
- Total Number of Non-AM Containers Preempted: 0
- Total Number of AM Containers Preempted: 0
- Resource Preempted from Current Attempt: <memory:0, vCores:0>
- Number of Non-AM Containers Preempted from Current Attempt: 0
- Aggregate Resource Allocation: 1279250 MB-seconds, 169 vcore-seconds

The attempt list shows one entry:

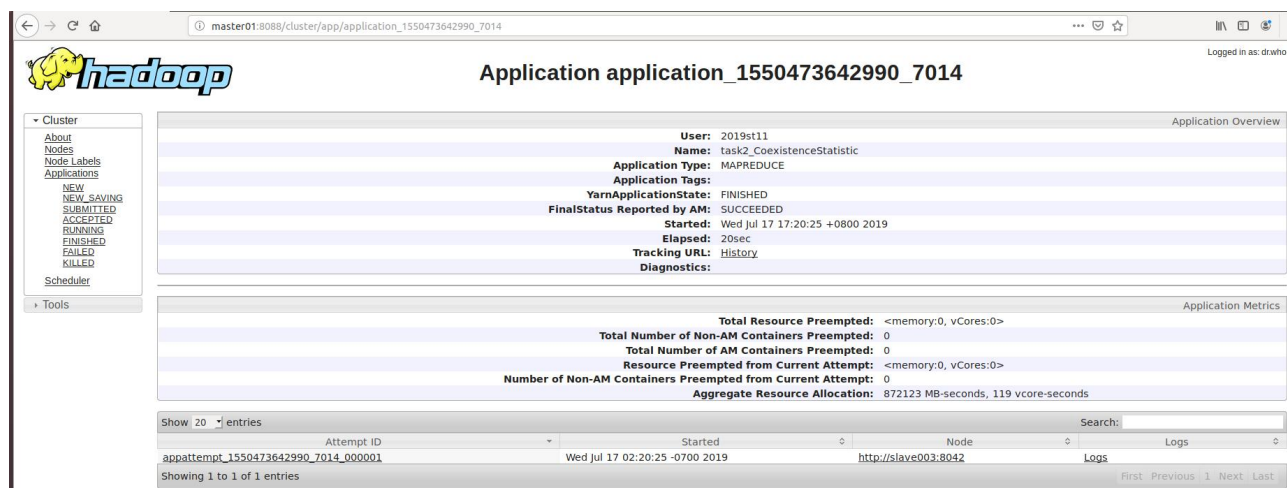
Attempt ID	Started	Node	Logs
appattemp1_1550473642990_7009_000001	Wed Jul 17 02:16:26 -0700 2019	<a href="http://slave003:8042">http://slave003:8042</a>	<a href="#">Logs</a>

### 7.2 Task2

任务执行：

application_1550473642990_7014	2019st11	task2_CoexistenceStatistic	MAPREDUCE	root.2019st11	Wed Jul 17 02:20:25 -0700 2019	Wed Jul 17 02:20:45 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
--------------------------------	----------	----------------------------	-----------	---------------	--------------------------------	--------------------------------	----------	-----------	-------------------------

具体WebUI截图：



The screenshot shows the Hadoop WebUI interface for application\_1550473642990\_7014. The application overview section displays the following details:

- User: 2019st11
- Name: task2\_CoexistenceStatistic
- Application Type: MAPREDUCE
- Application Tags:
- YarnApplicationState: FINISHED
- FinalStatus Reported by AM: SUCCEEDED
- Started: Wed Jul 17 17:20:25 +0800 2019
- Elapsed: 20sec
- Tracking URL: [History](#)
- Diagnostics:

The application metrics section shows:

- Total Resource Preempted: <memory:0, vCores:0>
- Total Number of Non-AM Containers Preempted: 0
- Total Number of AM Containers Preempted: 0
- Resource Preempted from Current Attempt: <memory:0, vCores:0>
- Number of Non-AM Containers Preempted from Current Attempt: 0
- Aggregate Resource Allocation: 872123 MB-seconds, 119 vcore-seconds

The attempt list shows one entry:


Attempt ID	Started	Node	Logs
appattemp1_1550473642990_7014_000001	Wed Jul 17 02:20:25 -0700 2019	<a href="http://slave003:8042">http://slave003:8042</a>	<a href="#">Logs</a>

## 7.3 Task3

任务执行:

application_1550473642990_7022	2019st11	buildGraph	MAPREDUCE	root.2019st11	2019 Wed Jul 17 02:22:39 -0700 2019	2019 Wed Jul 17 02:23:01 -0700 2019	FINISHED	SUCCEEDED		<a href="#">History</a>
--------------------------------	----------	------------	-----------	---------------	--	--	----------	-----------	--	-------------------------

具体WebUI截图:

 Application application\_1550473642990\_7022 Logged in as: drwho

Application Overview

User: 2019st11

Name: buildGraph

Application Type: MAPREDUCE

Application Tags:

YarnApplicationState: FINISHED

FinalStatus Reported by AM: SUCCEEDED

Started: Wed Jul 17 17:22:39 +0800 2019

Elapsed: 21sec

Tracking URL: [History](#)

Diagnostics:

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

Total Number of AM Containers Preempted: 0

Resource Preempted from Current Attempt: <memory:0, vCores:0>

Number of Non-AM Containers Preempted from Current Attempt: 0

Aggregate Resource Allocation: 172563 MB-seconds, 41 vcore-seconds

Show: 20 entries

Attempt ID	Started	Node	Logs
appattempt_1550473642990_7022_000001	Wed Jul 17 02:22:39 -0700 2019	<a href="http://slave003:8042">http://slave003:8042</a>	<a href="#">Logs</a>

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

## 7.4 Task4

具体WebUI和迭代次数截图:

All Applications - Mozilla Firefox										
master01:8088/cluster										
Show: 20 entries	ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress
	application_1563430216452_0185	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:21:31 -0700 2019	N/A	ACCEPTED	UNDEFINED	
	application_1563430216452_0184	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:21:06 -0700 2019	Thu Jul 18 02:21:29 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0183	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:20:37 -0700 2019	Thu Jul 18 02:21:01 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0182	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:20:15 -0700 2019	Thu Jul 18 02:20:35 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0181	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:19:44 -0700 2019	Thu Jul 18 02:20:09 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0180	2019st33	PreAna-1	MAPREDUCE	root.2019st33	Thu Jul 18 02:19:42 -0700 2019	Thu Jul 18 02:20:07 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0179	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:19:17 -0700 2019	Thu Jul 18 02:19:41 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0178	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:18:56 -0700 2019	Thu Jul 18 02:19:15 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0177	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:18:27 -0700 2019	Thu Jul 18 02:18:50 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0176	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:17:56 -0700 2019	Thu Jul 18 02:18:21 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0175	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:17:29 -0700 2019	Thu Jul 18 02:17:50 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>
	application_1563430216452_0174	2019st11	Graph Builder	MAPREDUCE	root.2019st11	Thu Jul 18 02:17:07 -0700 2019	Thu Jul 18 02:17:26 -0700 2019	FINISHED	SUCCEEDED	<a href="#">History</a>



ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking U
application_1563430216452_0212	2019st11	PageRank Viewer	MAPREDUCE	root.2019st11	Thu Jul 18 02:32:58 -0700 2019	Thu Jul 18 02:33:17 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0211	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:32:32 -0700 2019	Thu Jul 18 02:32:51 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0210	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:32:06 -0700 2019	Thu Jul 18 02:32:26 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0209	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:31:41 -0700 2019	Thu Jul 18 02:32:00 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0208	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:31:12 -0700 2019	Thu Jul 18 02:31:35 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0207	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:30:45 -0700 2019	Thu Jul 18 02:31:06 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0206	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:30:20 -0700 2019	Thu Jul 18 02:30:39 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0205	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:29:54 -0700 2019	Thu Jul 18 02:30:14 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0204	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:29:28 -0700 2019	Thu Jul 18 02:29:51 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0203	2019st11	PageRank Iter	MAPREDUCE	root.2019st11	Thu Jul 18 02:29:06 -0700 2019	Thu Jul 18 02:29:25 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>

```

2019st11@master01:~
File Edit View Search Terminal Help
Total time spent by all map tasks (ms)=5025
Total time spent by all reduce tasks (ms)=3875
Total vcore-seconds taken by all map tasks=5025
Total vcore-seconds taken by all reduce tasks=3875
Total megabyte-seconds taken by all map tasks=41164800
Total megabyte-seconds taken by all reduce tasks=31744000
Map-Reduce Framework
  Map input records=1283
  Map output records=37763
  Map output bytes=1194720
  Map output materialized bytes=205864
  Input split bytes=133
  Combine input records=37763
  Combine output records=2566
  Reduce input groups=1283
  Reduce shuffle bytes=205864
  Reduce input records=2566
  Reduce output records=1283
  Spilled Records=5132
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=59
  CPU time spent (ms)=4930
  Physical memory (bytes) snapshot=697057280
  Virtual memory (bytes) snapshot=12886597632
  Total committed heap usage (bytes)=2026897408
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=611549
File Output Format Counters
  Bytes Written=611549
Hadoop Map Task:
19/07/17 18:34:12 INFO client.RMProxy: Connecting to ResourceManager at master01/192.168.1.1:8032
19/07/17 18:34:12 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
19/07/17 18:34:13 INFO input.FileInputFormat: Total input paths to process : 1
19/07/17 18:34:13 INFO mapreduce.JobSubmitter: number of splits:1

```

## 7.5 Task5

### 任务执行:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking U
application_1563430216452_0230	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:53:41 -0700 2019	Thu Jul 18 03:54:01 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0229	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:53:08 -0700 2019	Thu Jul 18 03:53:34 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0228	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:52:45 -0700 2019	Thu Jul 18 03:53:05 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0227	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:52:16 -0700 2019	Thu Jul 18 03:52:42 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0226	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:51:50 -0700 2019	Thu Jul 18 03:52:14 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0225	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:51:19 -0700 2019	Thu Jul 18 03:51:44 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0224	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:50:57 -0700 2019	Thu Jul 18 03:51:16 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0223	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:50:34 -0700 2019	Thu Jul 18 03:50:55 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0222	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:49:59 -0700 2019	Thu Jul 18 03:50:27 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0221	2019st11	task5_LabelIter	MAPREDUCE	root.2019st11	Thu Jul 18 03:49:33 -0700 2019	Thu Jul 18 03:49:56 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0220	2019st11	task5_LabelBuilder	MAPREDUCE	root.2019st11	Thu Jul 18 03:49:05 -0700 2019	Thu Jul 18 03:49:25 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>
application_1563430216452_0212	2019st11	PageRank Viewer	MAPREDUCE	root.2019st11	Thu Jul 18 02:32:58 -0700 2019	Thu Jul 18 02:33:17 -0700 2019	FINISHED	SUCCEEDED	<div></div>	<a href="#">History</a>

具体WebUI截图:

**Application application\_1563430216452\_0230**

User: 2019st11  
 Name: task5\_Labelter  
 Application Type: MAPREDUCE  
 Application Tags:  
 YarnApplicationState: FINISHED  
 FinalStatus Reported by AM: SUCCEEDED  
 Started: Thu Jul 18 18:53:41 +0800 2019  
 Elapsed: 20sec  
 Tracking URL: [History](#)  
 Diagnostics:

Application Overview

---

Total Resource Preempted: <memory:0, vCores:0>  
 Total Number of Non-AM Containers Preempted: 0  
 Total Number of AM Containers Preempted: 0  
 Resource Preempted from Current Attempt: <memory:0, vCores:0>  
 Number of Non-AM Containers Preempted from Current Attempt: 0  
 Aggregate Resource Allocation: 168801 MB-seconds, 40 vcore-seconds

Application Metrics

Attempt ID	Started	Node	Logs
<a href="#">appattempt_1563430216452_0230_000001</a>	Thu Jul 18 03:53:41 -0700 2019	<a href="#">http://slave003:8042</a>	<a href="#">Logs</a>

Showing 1 to 1 of 1 entries


First Previous 1 Next Last

## 7.6 Task6

任务执行:

application_1563364017381_0348	2019st11	task6_LabelAnalysis	MAPREDUCE	root.2019st11	Wed Jul 17 19:34:57 -0700	Wed Jul 17 19:35:20 -0700	FINISHED	SUCCEEDED		<a href="#">History</a>
--------------------------------	----------	---------------------	-----------	---------------	------------------------------	------------------------------	----------	-----------	--	-------------------------

具体WebUI截图:



# Application application\_1563364017381\_0348

Cluster

- About Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler
- Tools

Application Overview

User:

2019st11

Name:

task6\_LabelAnalysis

Application Type:

MAPREDUCE

Application Tags:

YarnApplicationState:

FINISHED

FinalStatus Reported by AM:

SUCCEEDED

Started:

Thu Jul 18 10:34:57 +0800 2019

Elapsed:

22sec

Tracking URL:

History

Diagnostics:

Application Metrics

Total Resource Preempted:

<memory:0, vCores:0>

Total Number of Non-AM Containers Preempted:

0

Total Number of AM Containers Preempted:

0

Resource Preempted from Current Attempt:

<memory:0, vCores:0>

Number of Non-AM Containers Preempted from Current Attempt:

0

Aggregate Resource Allocation:

156226 MB-seconds, 39 vcore-seconds

Show 20 entries

Attempt ID	Started	Node	Logs
appattempt_1563364017381_0348_000001	Wed Jul 17 19:34:57 -0700 2019	http://slave003-8042	Logs

Showing 1 to 1 of 1 entries